27th International Conference on
Automated Planning and Scheduling
June 19-23, 2017, Pittsburgh, USA

PlanRob 2017

Proceedings of the 5th Workshop on **Planning and Robotics**

Edited by:

Alberto Finzi, Erez Karpas, and Goldie Nejat

Organization

Alberto Finzi
University of Naples Federico II, Italy

Erez Karpas
Technion — Israel Institute of Technology, Israel

Goldie Nejat
University of Toronto, Canada

Program Committee

Sara Bernardini, Royal Holloway University of London
Marcello Cirillo, Scania
Robert Fitch, Australian Centre for Field Robotics
Malik Ghallab, LAAS-CNRS
Felix Ingrand, LAAS/CNRS
Luca Iocchi, Sapienza University of Rome
Sven Koenig, University of Southern California
Matteo Leonetti, University of Leeds
Daniele Magazzeni, King's College London
Karen Myers, SRI International
Daniele Nardi, University of Rome "La Sapienza"
Andrea Orlandini, National Research Council of Italy (ISTC-CNR)
Enrico Scala, ANU Research School in Computer Science
Tiago Stegun Vaquero, MIT

Additional Reviewers

Wolfgang Hoenig, University of Southern California
T. K. Satish Kumar, University of Southern California

Foreword

Robotics is one of the most appealing and natural applicative area for the Planning and Scheduling (P&S) research activity, however such a natural interest seems not reflected in an equally important research production for the Robotics community. In this perspective, the aim of the PlanRob workshop is twofold. On the one hand, this workshop would constitute a fresh impulse for the ICAPS community to develop its interests and efforts towards this challenging research area. On the other hand, it aims at attracting representatives from the Robotics community to discuss their challenges related to planning for autonomous robots (deliberative, reactive, continuous planning and execution etc.) as well as their expectations from the P&S community.

The PlanRob workshop aims at constituting a stable, long-term forum on relevant topics concerned with the interactions between the Robotics and P&S communities where researchers can discuss the opportunities and challenges of P&S when applied to Robotics. Started during ICAPS 2013 in Rome (Italy) and followed by a second edition at ICAPS 2014 in Portsmouth (NH, USA), a third one at ICAPS 2015 in Jerusalem (Israel), and a fourth one at ICAPS 2016 in the UK (London), the PlanRob WS series (http://pst.istc.cnr.it/planrob/) has gathered excellent feedback from the P&S community which is also confirmed by the organisation of a specific Robotics Track from ICAPS 2014.

This fifth edition of the PlanRob workshop has been proposed in synergy with the Robotics Track to further enforce its original goals and to maintain an informal forum where more preliminary/visionary works can be discussed. PlanRob 17 succeeded in achieving these objectives providing a rich and articulated program. Indeed, 13 papers have been accepted for oral presentation covering many relevant topics in Planning and Robotics such as high-level task planning, task and motion planning, planning and execution for robots, planning and learning, human-robot interaction, real applications and case studies. The workshop program is completed by an invited talk and a discussion panel.

The varieties of research topics and results collected in these proceedings reflect a stimulating and intense research activity along with a growing interest for a forum where the Planning and Robotics communities can find a common ground.

Among the numerous people that contribute to the success of PlanRob 2017, we would first of all thank the ones that submitted their research papers to the workshop and attended the event. Moreover, we sincerely thank the program committee for the important work on the reviewing process.

Alberto Finzi, Erez Karpas, and Goldie Nejat
June 2017

Contents

Intra-Robot Replanning to Enable Team Plan Conditions
Philip Cooksey and Manuela Veloso **1**

Expressing Campaign Intent to Increase Productivity of Planetary Exploration Rovers
Daniel Gaines, Gregg Rabideau, Gary Doran, Steve Schaffer, Vincent Wong, Ashwin Vasavada and Robert Anderson **8**

STRIPS Planning in Infinite Domains
Caelan Garrett, Tomas Lozano-Perez and Leslie Kaelbling **19**

Joint Perception And Planning For Efficient Obstacle Avoidance Using Stereo Vision
Sourish Ghosh and Joydeep Biswas **30**

On the Application of Classical Planning to Real Social Robotic Tasks
José Carlos González Dorado, Fernando Fernández Rebollo, Ángel García Olaya and Raquel Fuentetaja Pizán **38**

Augmenting Planning Graphs in 2-Dimensional Dynamic Environments With Obstacle Scaffolds
Spencer Lane, Kyle Vedder and Joydeep Biswas **48**

Human-in-the-Loop SLAM
Samer Nashed and Joydeep Biswas **53**

Towards CLIPS-based Task Execution and Monitoring with SMT-based Decision Optimization
Tim Niemueller, Gerhard Lakemeyer, Francesco Leofante and Erika Abraham **60**

Deep Spatial Affordance Hierarchy: Spatial Knowledge Representation for Planning in Large-scale Environments
Andrzej Pronobis, Francesco Riccio and Rajesh Rao **68**

Dynamics-Aware Reactive Planning for Unmanned Ground Vehicles to Avoid Collisions with Dynamic Obstacles on Uneven Terrains
Pradeep Rajendran, Brual C. Shah and S.K. Gupta **80**

Towards an Architecture for Discovering Domain Dynamics: Affordances, Causal Laws, and Executability Conditions
Mohan Sridharan and Ben Meadows **92**

Approximating Reachable Belief Points in POMDPs with Applications to Robotic Navigation and Localization
Kyle Wray and Shlomo Zilberstein **104**

Integrated Commonsense Reasoning and Probabilistic Planning
Shiqi Zhang and Peter Stone **111**

Intra-Robot Replanning to Enable Team Plan Conditions

Philip Cooksey[1] **and Manuela Veloso**[1]
[1]Carnegie Mellon University

Abstract

Individual team members are the building blocks of successful multi-robot teams in dynamic competitive domains. The current approach to designing a team is to divide the planning into a hierarchy by separating team coordination and task assignment – global planning – from task planning and execution – local planning. The global planner must make assumptions based on simplified models of dynamics and/or opponents, and as such certain conditions are assumed true when globally planning but are not always true at local execution time. In this paper, we describe several algorithms for intra-robot replanning that allow the individual robots to enable the conditions of their tasks. We then demonstrate improvements in task completion when the robots are capable of replanning their task(s) and their teammates' task(s) in a simplified robot soccer domain. We further show preliminary results on learning when to replan.

Introduction and Related Work

Cooperative multi-robot team planners in competitive robot domains often have limited computational time, have poorly modeled dynamic objects, and have incomplete knowledge of their environment. These issues remain with the individual robots, however they can gather state information and learn to improve their execution by choosing the best fit replanning algorithms. Involving the team planner to incorporate more information on every robot is not effective or practical for highly dynamic domains with potentially many robots. Likewise, team planners often reduce information and complexity in order to make team planning feasible within dynamic domains, so additional information can hinder performance.

The standard approach to team planning in the literature is to use a hierarchy, thereby dividing up the planning problem involved in controlling a team of robots (Yan, Jouandeau, and Cherif 2013), (Simmons et al. 2002), (Pecora and Cesta 2002). We follow the Skills, Tactics, and Plays (STP) hierarchy as described by (Browning et al. 2005). This division of computation is used to simplify the problem of controlling a team in dynamic, competitive environments and allows planning to happen at different abstraction layers. Skills are low level repeatable algorithms that are specific to the domain. Tactics combine skills towards accomplishing a more complex task using finite-state machines, consider

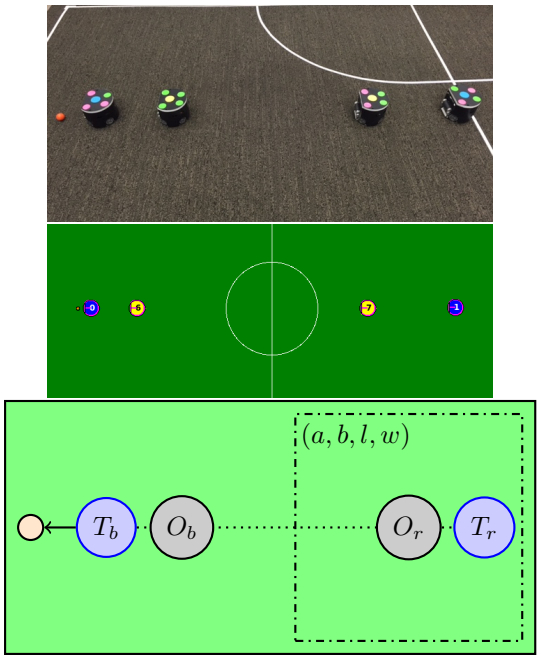


Figure 1: Passing with marking: T_b needs to pass the ball to T_r within the zone (a, b, l, w). T_b is facing the ball with the direction arrow. Opponents O_b and O_r try to remain on the line between the other robots to block or intercept the pass. The top image is the physical robots, middle is the simulated robots, and bottom is a simplified representation that we will use in this paper.

passing the ball which includes Skills like driving into the ball, positioning the ball, and kicking the ball. Plays guide the team towards their goal(s) and they define robot roles, positions, and a series of Tactics. Plays are changed based on defined state variables, i.e., number of robots and ball on defensive side. STP essentially separates planning into two levels, global (Plays) and local (Tactics).

In competitive dynamic domains, the global planner will assign a Play using incomplete information and simplified models of the environment. This missing information is often due to opponents' adversarial behavior but can also be attributed to the simplified models of the team members'

abilities. A Play will only change when new information is presented at the global level that triggers a failure in its defined state variables, so failures at the local level might not be considered. This is due to the different requirements of the global planner and the local planning robots with respect to their abstraction in planning, i.e., to local information gains and/or the global planner not fully modeling the domain. We can especially see this in competitive domains like the Small-Size Robot Soccer League (SSL).

The SSL league matches two teams of six omnidirectional robots and each team receives the same information (positions and headings) through overhead cameras (Zickler et al. 2010). Any further information must be generated by each team including velocities, predicting opponents, and the physics of ball movement at the expensive of each team's computation. In STP, a Play may assign a pass between two robots for reasons unknown to the local robot. Still, the computation for completing that pass is left to the Tactic, and opponents may make that exact pass impossible to the assigned robot. Therefore, knowing that the global planner has made assumptions, we argue that robots can collect local information to learn when to use *intra-robot replanning* algorithms to enable conditions during execution to make a more successful team.

We note that Plays provide a fully instantiated *team plan* for each team member robot T_i. The robots execute their Tactics according to their assigned variables. The global planner may at any time change the Play due to new information, but there is no guarantee for changes in Plays due to faults observed only at the local level. The Play was created with the Tactics' conditions considered true and that the robot could complete its role using those Tactics. An implicit assumption was that a failure to complete a Tactic would lead to a global failure, which the global planner would then solve. If however a local failure does not cause an immediate global failure, then the robot will continue failing until complete global failure. In robot soccer, a local failure is a robot maintaining control of the ball but never finding an open pass, which results in it driving around in circles looking for an open pass. A global failure would be a missed pass or stolen ball.

In this paper, we investigate *intra-robot replanning* algorithms to enable conditions that are preventing locally succeeding at the assigned task. The topic of *intra-robot replanning* has recently been investigated and formalized in (Talamadupula et al. 2013), which defines a more general model for replanning using different constraints. Replanning for individual robots has mostly been seen as a task of minimizing the changes to the current plan, and this same idea has been applied to multi-robot systems. However, in competitive domains, the robots have to compensate for the dynamic nature of the environment, so replanning must be quick and should be focused on enabling failed conditions with the highest chance of success. Optimality is often ill-defined in such complex domains so we focus on the robot accomplishing the assigned Tactic rather than failing, subsequently failing the team's objective. We contribute *intra-robot replanning* algorithms to enable the soccer robot to replan for failed conditions. We further demonstrate that the robot can learn which algorithm has the highest chance of success.

We highlight a few competitive domains that require *intra-robot replanning* but acknowledge that it can be useful in other domains like fleets of autonomous cars or drones coordinating towards a common goal while compensating for dynamic changes. In capture the flag (Atkin, Westbrook, and Cohen 1999), the team can have limited information about the opponent's team and their flag. The team plan that is created is limited to their available information. Assuming they can only globally plan on their side, they would need to locally replan for local changes in their information as well as their teammates'. Similarly, in robot soccer, the global planner has partial information about the opponent's behavior and it makes assumptions when creating Plays. The robot tasked with passing may need to replan its own position or the other robot's passing location to help it accomplish the pass. We will focus our efforts on a derivative of robot soccer to demonstrate the need for *intra-robot replanning* to enable conditions.

Problem Domain

Passing with marking, shown in Figure 1, is a sub-domain of robot soccer where the task is for the robot with the ball, T_b, to pass the ball to the receiving robot, T_r, while opponents try to steal and/or intercept the ball. One opponent is always placed near T_b to block the initial pass. In our example, the opponents, O_i, are placed on the line between the two T_i to block and intercept the straight pass.

An example Play generated by the global planner is given in Figure 2. The Play is made of Tactics: *Pass*, *Goto*, and *Receive*. Each Tactic has instantiated variables defining the assigned robot and the variable(s) for that Tactic. For *Pass*, T_b is assigned to kick the ball to $(5, 5)$. For *Goto*, T_r is going to location $(5, 5)$ to *Receive* the ball and must stay within the $Zone(a, b, l, w)$ defining a box with the left top corner at location (a, b) with length l and width w.

The Play assigned the Role kicker to T_b and receiver to T_r. In this case, T_r is assigned the best passing location within its zone (the dashed-dotted square in Figure 1), but clearly O_r will attempt to intercept the ball. The assignment for the best passing location and the generation of the probability of success for that pass follows the approach in (Biswas et al. 2014) and is outside the scope of this paper. The Play will often fail quickly from the opponents' interference. It can also fail due to randomness in the robot's performance and kicking ability in the physics simulator. There have been attempts to solve this issue in robot soccer as passing is a major requirement and machine learning has been a major focus of this research (Stone et al. 2006). However, our approach is to design replanning algorithms that enable failed conditions and learn when to use each algorithm.

Individuals Enabling Conditions

In (Mendoza et al. 2016), they describe an approach called pass-ahead that tries to sync the arrival of the receiver with the ball so that they arrive at the passing location at the same time. This approach alone does poorly in the *passing with marking* domain because of the opponent O_b that blocks or

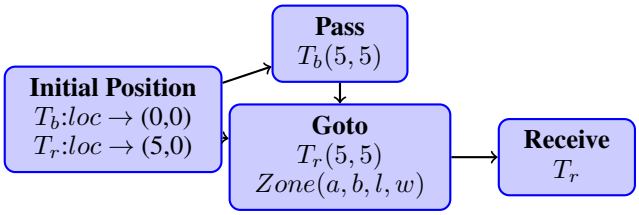


Figure 2: A team plan for T_b to pass the ball to T_r.

steals the ball. In (Cooksey, Mendoza, and Veloso 2016), they demonstrated the requirement of opponent aware algorithms for T_b to maintain possession of the ball, which improved pass-ahead's performance in the *passing with marking* domain (**Dribbling-Move** in Results Section). However, that algorithm does poorly in our example domain where an opponent was added to mark the receiving robot. The issue is that Dribbling-Move only focuses on replanning to enable one condition of the Tactic *Pass*, which is ball possession ($HasBall$). $HasBall$ is defined as true if the ball is closer to it than any other robot. And, the condition being failed is $Open$, which is defined true if the pass can be successful. $Open$ is ill-defined without knowledge of the opponents' intercept abilities and therefore is estimated by the global planner. Another issue is in T_b's ability to enable the $Open$ condition.

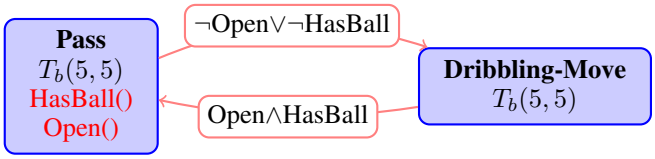


Figure 3: *Intra-robot replanning* to enable Pass's condition $HasBall$ through dribbling. HasBall() and Open() are conditional functions (T/F) and switch what Tactic is used through the links.

Dribbling-Move enables the condition $HasBall$ by dribbling – the robot has a rotating bar in its forward direction, the arrow in Figure 1, that applies a back spin on the ball for maintaining possession – and can implicitly enable the condition $Open$ by moving the ball towards the passing location while circumventing nearby opponents. This does not solve the issue of opponents marking the receiver or the ill-defined $Open$ condition. For now, $Open$ can be defined as true if the probability of a successful pass, given by the team planner, is above the defined threshold γ. The partial Play in Figure 3 illustrates the *intra-robot replanning* for T_b using Dribbling-Move. The robot changes its local position variable towards the passing location. As previously stated, the condition $Open$ is only being enabled implicitly as T_b is not attempting to find a new or better passing location. The condition will be enabled when the global planner declares the current pass probability above γ given some assumptions and simplification based on the current state. Dribbling-Move gives a relatively simple solution to enabling conditions $HasBall$ and $Open$, but it performs poorly in our example domain.

$HasBall$ can be defined as an **independent condition**. The ability to enable it is on a single robot's own ability to keep the ball or get the ball. Independent conditions can be enabled by changes in the robot's assigned variables to compensate for changes in the environment. $Open$ can be defined as a **dependent condition**. Its ability to be enabled is highly dependent on the other teammate(s) and/or opponent(s). Dependent conditions can be enabled by changes in the variables of the robots that are involved with enabling the condition. Obviously, the opponents' variables cannot be manipulated directly so it can be difficult to enable a dependent condition.

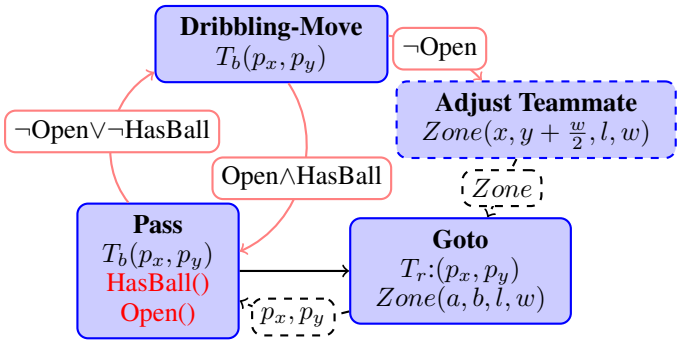


Figure 4: *Intra-robot replanning* to more explicitly enable $Open$ by using the action Adjust Teammate to change the zone variable.

To enable $Open$, T_b would need to change its variables – this occurs through dribbling – and the variable(s) of the teammate's Tactic involved in the pass. Here we add the action **Adjust Teammate** to Dribbling-Move which changes the $Zone$ variable of the receiver's Goto Tactic, shown in Figure 4. Adjust Teammate is used if $Open$ is not enabled when dribbling. It places the zone in front of T_b and towards T_r as shown in Figure 5. A new location is found within the new zone for the pass and given to T_b and T_r. T_b can now explicitly attempt to enable the $Open$ condition by moving the teammate to a zone better situated for its own dribbling abilities.

The $Zone$ becomes a sliding window towards the right based on T_b's location, shown in Figure 5. This ensures that T_b is always close to the passing $Zone$ and the global planner finds T_r a new pass location.

Experimental Results

Experimental Setup

We use the *passing with marking* domain as we described in our example with a field size of 9m long by 6m wide. T_b starts at (2m, 3m) with the ball directly in front of it at (1.86m, 3m) and opponent O_b behind it at (2.4m, 3m). Teammate T_r starts at (7m, 3m) with the opponent O_r in front of it at (6m, 3m). The Play is always the same with T_b passing to T_r, however the passing location changes quickly as the global planner attempts to find the best position within T_r's defined zone. The zone is defined as $(x = 4.5m, y =$

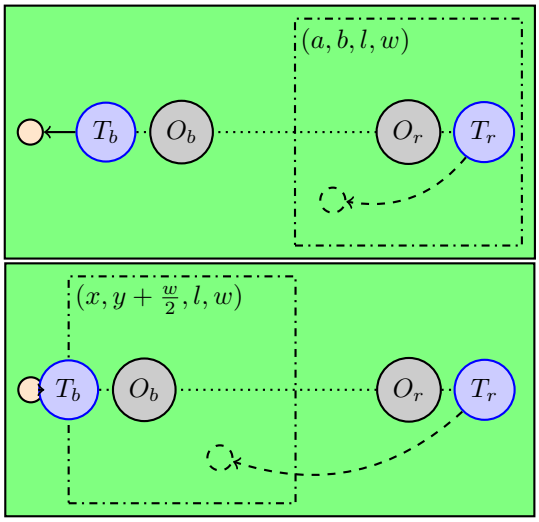


Figure 5: Demonstrating the $Zone$ adjustment, resulting in a change in pass location, from the plan in Figure 4.

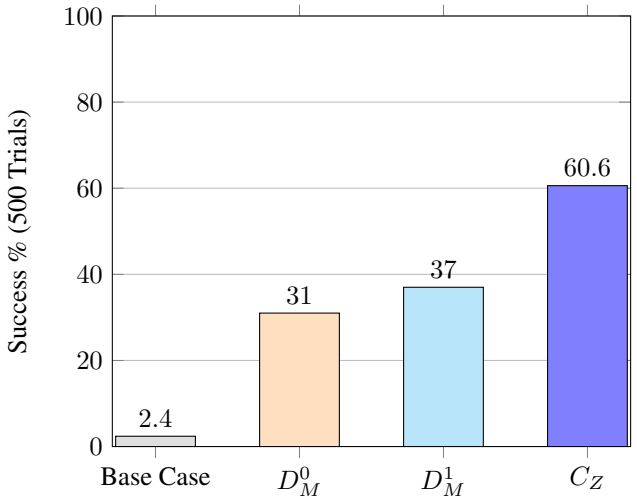


Figure 6: Results for four different algorithms used by T_b in the *passing with marking* domain.

$0m, l = 4.5m, w = -6m$), so the zone covers half of the field.

The pass must happen within one minute or it is considered a failed pass. If the ball goes outside field boundaries it is also considered a failed pass. Touching is defined as being within a robot radius plus a ball radius to the center of a robot. If the ball is touching an opponent for five video frames then it has been intercepted and the pass has failed. A pass succeeds if T_r is touching the ball for five video frames.

The opponents place themselves in the direct line of sight between the two T_i. O_b maintains a close distance of three robot radii (3*0.09m) from T_b. O_r maintains a distance of 0.5m from T_r. When the ball is kicked, i.e., its velocity goes above 0.9m/s and it is two robot diameters away from T_b, O_r attempts to intercept the ball.

T_r uses the pass ahead algorithm to receive the ball as previously referenced. We are introducing **Change-Zone** which follows Figure 4. The zone parameters were determined by trial and error for improving the performance of passing. We compare **Change-Zone** to three algorithms for T_b with varying degrees of replanning to enable conditions:

- **Base Case**: Positions to face the ball and pass location, then immediately kicks as planned (no replanning).
- **Dribbling-Move,** D_M^0**,** $\gamma = 0$: Previously referenced algorithm with γ set for any pass.
- **Dribbling-Move,** D_M^1**,** $\gamma = 0.15$: Only passes if the pass probability is above γ.
- **Change-Zone,** C_Z: Uses D_M^1 with the Adjust Teammate action to enable the condition $Open$ by altering the Teammate's Zone using: $(l = 2m, w = 6m)$.

Results

Each different T_b algorithm was run five hundred times in a physics-based simulator starting with the same formation of teammates, opponents, and the ball. The results can be found in Figure 6. Base Case fails almost every time and its few successes can be attributed to random chance as the ball is often stolen or blocked. Both Dribbling-Move algorithms improved the chance of a successful pass by enabling the condition $HasBall$, but passes were often intercepted by O_r. With the higher γ, Dribbling-Move further increased the success rate by waiting for γ to improve (even if just slightly as 0.15 is still very low). Change-Zone further increased the success rate by more explicitly attempting to enable $Open$ by moving T_r closer to find an open pass.

We used the binomial proportion test to verify that the different success rates of the algorithms were not due to randomness or limited sampling. The binomial proportion test is, as defined in (Ryan 2008): given a set of N_1 observations in a variable X_1 and a set of N_2 observations in a variable X_2, we can compute a normal approximation test that the two proportions are equal, or alternatively, that the difference of the two proportions is equal to 0. A standard approach is that a p value of less than 0.05 can reject the null hypothesis. In our case, the null hypothesis defines that there is not enough evidence to show a statistically significant difference between the results of the algorithms.

As shown in Figure 7 each improvement on the Base algorithm is statistically significant when compared to previous algorithms. The initial change, D_M^0, occurred when replanning enabled the condition $HasBall$ by dribbling. Our contribution, C_Z, shows further significant improvement by replanning to enable the condition $Open$ by moving the teammate into a more beneficial zone. Our data demonstrate the large improvement when individual robots within a team can replan to enable all the conditions needed by their Tactics.

The authors of the Dribbling-Move algorithm showed the improved success rate of passing when marked by an opponent similar to O_b. With the inclusion of the second opponent, O_r, the success rate dropped dramatically. We can see that the improvement of Dribbling-Move mainly contributed to the enabling of the condition $HasBall$ and its ability to

Binomial Proportion Test (Two-Tailed)		
Algorithm	**Algorithm**	**P Value**
Base Case	D_M^0	< 0.0001
D_M^0	D_M^1	0.0452
D_M^1	C_Z	< 0.0001

Figure 7: Testing for the null hypothesis that the two proportions are equal and the difference is due to randomness.

handle the close opponent, O_b. Changing-Zone's improvement was in changing the teammate's zone variable to benefit the dribbling algorithm's abilities. It brought the passing location closer for tighter, shorter passes and this helped improve the probability of enabling $Open$.

Choosing a Replanning Algorithm

In this section, we describe how to choose which algorithms to use during the execution of the robot's Tactic. First, we describe an approach using the *worst case* analysis for choosing the appropriate algorithm. Second, we describe a *state based* approach that learns which algorithm to use given a world state. This approach allows the robot to use simpler algorithms when possible, like the base case, while learning when to use the more complicated ones like C_Z.

Worst Case

The task of our robot is to pass the ball to another teammate. This involves releasing the ball and losing all further control over it. It is an irreversible action, which cannot be replanned or halted. With irreversible actions, we would like to choose an algorithm that would minimize the worst case failure rate of the robot, because most of the failures cause the other team to obtain the ball.

Given the individual robot must pick to execute one of the algorithms in order to accomplish the task of passing, we define a method of choosing an algorithm based on the worst case. A standard approach for choosing one algorithm over another is if the algorithm is better in the worst case independently of the worst case actually occurring. Let $Z = \{0, 1\}$, where 0 is a failure and 1 is a success, and p be the function that gives the probability $p(z)$, where $\sum_{z \in Z} p(z) = 1$, for obtaining the failure or success in Z. We then assign a value $v(z)$ to the values of Z. We set $v(z)$ to be the total number of success for the algorithm, in Figure 6. We use the definition in (Rubinstein 2006),

$$p \succsim q \text{ if } min\{v(z)|p(z) > 0\} \geq min\{v(z)|q(z) > 0\} \quad (1)$$

Therefore, the algorithms would be ordered $C_Z \succsim D_M^1 \succsim D_M^0 \succsim BaseCase$. So, the **Change-Zone** algorithm would have the largest minimum value and be chosen as the default algorithm.

State Based

The worst case method will choose the algorithm that minimizes the worst case scenario, but by definition it does not consider the actual probability of the worst case happening. This can be seen as an issue if the worst case rarely, if ever, happens. Another possible issue is that each of the replanning algorithms alters the team plan causing changes that may have been unnecessary. In a normal game, situations like our example domain may not occur with high probability. In other words, if we have an open pass to our teammate with no opponents nearby there would be no point in dribbling the ball and/or moving the teammate closer.

A better method for picking the replanning method would then be based on the state of the world. Using the state information, it is possible to learn the likelihood of that algorithm being successful. Then we can choose the algorithm given its likelihood of succeeding. We must reiterate that, because the domain is dynamic and adversarial, success is very probabilistic even in the exact same state. Our robots move with some randomness which means that predicting the likelihood of an algorithm being successful at any given state will be probabilistic. The robots will likely diverge quickly into very different states in the future as subtle differences and randomness influence their trajectories.

We use neural networks to learn a function from world state to the probability of success. The state is described in the reference frame of the robot using the algorithm in order to help generalize different situations across the field.

Input State

- T_b: (X, Y) position and velocity of T_b.
- **Ball**: (X, Y) position and velocity of ball in robot's frame of reference.
- **Teammates**: (X, Y) position and velocity of each teammate in robot's frame of reference.
- **Opponents**: (X, Y) position and velocity of each opponent in robot's frame of reference.

Output State

- **Probability Value**: Likelihood of input state leading to a successful pass using an algorithm.

The data needed to train the neural network can be collected over many runs of each algorithm in various scenarios. We can simulate similar episodes in our example domain with variations on ball location, opponent's locations, and teammate's locations. As an episode starts we begin saving state information for each video frame. Then once the pass has either been received, intercepted, or gone out of bounds (the episode ends), we label all the saved states with a success value of 1 or a failure value of 0. As previously mentioned, even starting in the same state does not mean the robots will execute the same path in the future even when using the same algorithm. There is therefore a probabilistic nature to succeeding from a given state.

The neural network can then be trained using a supervised learning method. Given that similar, even the exact same, states will most likely be labeled success and failure, the network will actually learn the probability of succeeding within the training data. Therefore, the output is the likelihood of an algorithm succeeding from a given state.

Given that we train multiple neural networks, each trained on one algorithm, we will have the likelihood of success

Predicting Success or Failure of D^1_M			
Data Sets	**Accuracy**	**Recall**	**Precision**
Training Data	0.70	0.31	0.74
Next 200,000	0.65	0.26	0.65
Next 200,000	0.71	0.24	0.50

Figure 8: Accuracy of predicting successes or failures for our example domain using the algorithm D^1_M.

for each algorithm given a state. We then must decide on a method of choosing which algorithm to run based on the probability value. We use the expected utility of the algorithm as described in (Rubinstein 2006):

$$p \succsim q \text{ if } \sum_{z \in Z} v(z)p(z) \geq \sum_{z \in Z} v(z)q(z) \quad (2)$$

where $v(z) = \frac{z}{C(a_i)}$ and $C(a_i)$ is an ordering cost for each algorithm, a_i.

Preliminary Experimental Evidence To test the state based method, we used the open source library OpenANN (Fabisch 2017). The neural network has 20 inputs, see *Input State*, and 1 output as the probability of success, see *Output State*. We used a fully-connected network with three hidden layers with 100 neurons in each layer. We used LOGISTIC activators for the neurons and Mini-batch Stochastic Gradient Descent (MBSGD) to solve the supervised learning problem.

Our preliminary work has focused on determining the accuracy of predicting the success or failure of an algorithm using our example domain. We used the D^1_M algorithm and repeatedly ran our example domain roughly five thousand times, gathering over 4 million states. We used two hundred thousand states to train the network over ten thousand iterations of MBSGD.

In Figure 8, we see that the accuracy of prediction is between 0.65-0.71. The low accuracy can be attributed to the randomness of a state leading to a success or failure because of the inherit randomness in the robot's performance. For example, the starting position of our *passing with marking* domain should be labeled as failure because on average it fails more often than it succeeds. This is confirmed as the start position produces the value 0.412. The neural network will always consider that position as a failure, which brings down the accuracy when the robot happens to succeed. In comparison to D^1_M's success rate in Figure 6, 0.37, the neural network has learned a similar approximation.

The low recall performance can be attributed to the large bias for failure in the dataset. The individual robot is started in an disadvantaged position and the likelihood of succeeding is already low. Similarly, when the robot actually succeeds in a given state it has probably failed multiple times from that exact same state in the dataset. The negative examples then highly outnumber the positive examples leading the neural network to perform poorly on recalling positive examples. A possible correction for this would be to balance the samples of positive and negative, however, this new bias would likely decrease the precision and accuracy of predicting if the algorithm will fail from a given state.

This preliminary work demonstrates the ability to learn the probability of success for a given state. Future work includes training neural networks for every algorithm and choosing which algorithm to execute based on their predictions and expected utilities.

Conclusion and Future Work

Competitive dynamic multi-robot domains are challenging tasks to solve, and the use of hierarchical planning, like STP, to divide the planning is a useful and successful method for tackling this enormous problem. In STP, research has focused on improving the creation of Plays and designing Tactics that react to the environment in real-time. However, there still lacks thorough understanding of the requirements of the individual teammate and its need for replanning within teams and hierarchical planners. Plays are often determined by global variables that change slowly or at a global view point. Replanning at the Play's level is focused on the task of fixing global problems for the team. Therefore, replanning at the Tactic's level has been left struggling to accomplish the predefined task as assigned by the global planner.

Intra-robot replanning is then a way for Tactics to enable the conditions of their tasks by changing and adjusting the variables of the team's plan. We demonstrated an increase in team performance by providing the Tactic level with the ability to replan its robot's location, the ball's location, and a team member's variable in order to enable the Tactic's conditions. In our example, this was through dribbling and adjusting the preplanned zone of the team member. As the types of conditions were different, independent and dependent, enabling them required a difference in what variables needed to be changed. Replanning to enable both types of conditions clearly provides a better alternative to failing and waiting for the global planner to change its task.

We showed preliminary results on predicting the probability of succeeding for an algorithm. Using state information gathered by the tactic during its execution, we could train a neural network afterwards to output the probability of success. Future work has been left to picking the algorithm with the highest expected utility such that the robot succeeds often without defaulting to an unnecessarily complicated algorithm. Research is also needed to learn how to change variables in different situations. Specifically, it would be beneficial to learn the actual zone size and zone location that improves the pass success beyond the hard coded zone used in our experiments.

References

Atkin, M. S.; Westbrook, D. L.; and Cohen, P. R. 1999. Capture the flag: Military simulation meets computer games. In *Proceedings of AAAI Spring Symposium Series on AI and Computer Games*, 1–5.

Biswas, J.; Mendoza, J. P.; Zhu, D.; Choi, B.; Klee, S.; and Veloso, M. 2014. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team.

In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 493–500. International Foundation for Autonomous Agents and Multiagent Systems.

Browning, B.; Bruce, J.; Bowling, M.; and Veloso, M. 2005. Stp: Skills, tactics, and plays for multi-robot control in adversarial environments. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 219(1):33–52.

Cooksey, P.; Mendoza, J. P.; and Veloso, M. 2016. Opponent-aware ball-manipulation skills for an autonomous soccer robot. In *Proceedings of the RoboCup Symposium*. Leipzig, Germany: Springer. Nominated for Best Paper Award.

Fabisch, A. 2017. Openann. `https://github.com/OpenANN/OpenANN`.

Mendoza, J. P.; Biswas, J.; Zhu, D.; Cooksey, P.; Wang, R.; Klee, S.; and Veloso, M. 2016. Selectively Reactive Coordination for a Team of Robot Soccer Champions. In *Proceedings of AAAI'16, the Thirtieth AAAI Conference on Artificial Intelligence*.

Pecora, F., and Cesta, A. 2002. Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*.

Rubinstein, A. 2006. *Lecture Notes in Microeconomic Theory*. Dordrecht: SUNY-Oswego, Department of Economics. 87–96.

Ryan, T. P. 2008. *Modern Engineering Statistics*. Number 124-126. Wiley.

Simmons, R.; Smith, T.; Dias, M. B.; Goldberg, D.; Hershberger, D.; Stentz, A.; and Zlot, R. 2002. *A Layered Architecture for Coordination of Mobile Robots*. Dordrecht: Springer Netherlands. 103–112.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, 93–105. Springer.

Talamadupula, K.; Smith, D.; Cushing, W.; and Kambhampati, S. 2013. A theory of intra-agent replanning. *ICAPS 2013 Workshop on Distributed and Multi-Agent Planning (DMAP)*.

Yan, Z.; Jouandeau, N.; and Cherif, A. A. 2013. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems* 10.

Zickler, S.; Laue, T.; Birbach, O.; Wongphati, M.; and Veloso, M. 2010. Ssl-vision: The shared vision system for the robocup small size league. In *RoboCup 2009: Robot Soccer World Cup XIII*. Springer. 425–436.

Expressing Campaign Intent to Increase Productivity of Planetary Exploration Rovers

Daniel Gaines, Gregg Rabideau, Gary Doran, Steve Schaffer, Vincent Wong, Ashwin Vasavada, Robert Anderson

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

Achieving consistently high levels of productivity has been a challenge for Mars surface missions. While the rovers have made major discoveries and dramatically increased our understanding of Mars, they often require a great deal of effort from the operations teams and achieving mission objectives can take longer than anticipated. Missions have begun investigating ways to enhance productivity by increasing the amount of decision making performed onboard the rovers. Our work focuses on the use of goal-based commanding as a means of more productively operating rovers. In particular, we are working on ways to convey the intent that operations team use to conduct science campaigns to the rover so that it can guide the rover in creating high quality plans and in identifying its own goals based on operator guidance. In addition to informing future surface exploration missions, this work is relevant for a wide range of applications in which operators must interact with a robotic system with limited communication opportunities.

Introduction

Maintaining high levels of productivity for the Mars exploration rover missions is highly challenging. While the Curiosity operations team has made significant accomplishments with the rover, doing so often requires a large amount of human effort in planning, coordinating, sequencing and validating the development of command products for the rover. Further, limitations in communication opportunities and anomalies on the vehicle can cause delays in accomplishing the team's objectives. These productivity challenges can result in the under-utilization of the vehicle's resources. These productivity challenges are anticipated to increase as our aging fleet of sun-synchronous orbiters are replaced by non-sun-synchronous orbiters, which do not provide a consistent pattern of "end-of-day" downlink relays.

The Jet Propulsion Laboratory has been exploring options for addressing these productivity challenges including conducting an extensive study of productivity factors in the Mars Science Laboratory mission (Gaines et al. 2016) and investigation into incorporating onboard activity scheduling for the Mars 2020 mission (Benowitz 2016).

In this paper we discuss work that is continuing the investigation into increasing the amount of decision making performed onboard rovers for future planetary exploration missions. In particular, we are leveraging existing technology and developing new technology to enable rovers to be more goal-directed, including following goals provided by ground operations as well as identify their own goals under the guidance of operators. We believe that this approach to goal-directed behavior will enhance surface mission productivity in the following ways:

Reducing operator effort: A goal-based interface presents a higher-level, more intuitive interface to rovers compared to the current highly-detailed command sequence interface. We believe that developing and validating command products with a goal-based interface will require less time and effort for the operations team.

Increasing rover resource utilization: The limited decision making of current rovers results in the operations team making highly conservative predictions of the amount of resources, e.g. time and energy, required to perform activity and results in significant amount of unused vehicle resources. By increasing onboard decision making, the rover can use knowledge of current vehicle resources to make more informed decisions about the goals that can be accomplished.

Reducing reliance on ground-in-the-loop cycles: Current operations relies on frequent interactions with the rover to maintain high levels of productivity in which the team assesses the rover's latest state and provides the detailed command products that direct the rover in accomplishing mission objectives. In contrast, a goal-based interface allows the team to provide objectives to the rover with reduced knowledge of the rover's state. In addition, by providing appropriate guidance, the rover is able to identify its own goals to accomplish mission objectives, further reducing the reliance on ground-in-the-loop contacts.

Although goal-based commanding has not been used on planetary rovers, it is of coarse a well-established form of interaction with robots and has also been used in other forms of space missions (Muscettola et al. 1998; Chien et al. 2005). In addition, the Opportunity and Curiosity rovers have a restricted form of goal selection in which they are able to

select targets for follow-up observations based on scientist guidance (Francis1 et al. 2016).

Our focus in this paper is providing rovers guidance on what goals to work on when:

1. the set of proposed goals over-subscribe vehicle resources and the planner must select a subset of goals to accomplish, and
2. the rover has a surplus of resource or has entered a new area that ground operators have not yet seen and should identify its own set of goals to pursue.

Our approach is to derive this guidance from the intent the science team uses when developing science campaigns. We go into more detail on campaign intent in later sections, but in general, we view intent as specifying relationships among goals. These relationships are used to determine the value of including a set of goals in a plan and, in some cases, how, or more specifically, when goals are accomplished. For example, the science team may be interested in collecting samples of a certain type of rock formation or performing a certain type of observation every X meters the rover drives. In the former case, the intent would specify the type of goal of interest (sampling a formation) and the value of accomplishing goals of this type (e.g. 2 or 3 samples is very important, additional samples are nice but less important). The latter case indicates a preference to periodically collect an observation and would indicate how important it is to accomplish the goals within a given tolerance of the indicated periodicity.

While our initial motivation for expressing campaign intent was for science objectives, this type of guidance is also relevant for many types of engineering maintenance activities performed by the rover. For example the team performs periodic activities to monitor various rover subsystems and dump system information. There is a cadence that must be followed in collecting this information, but there is flexibility in the exact timing of the activity. We have found that guidance for these engineering activities is similar to guidance for science campaigns.

In the next section we provide background on rover operations to help establish context. We then describe examples of rover science campaigns and how we derived campaign intent from these examples. Next we describe the specific semantics we developed to represent campaign intent and discuss how we are using this intent as guidance to enable the rover to generate high quality plans and identify its own goals.

Background on MSL Mission Operations

We begin with a brief overview of some important facets of MSL operations to provide context for the case study. This is not a comprehensive description of MSL operations, rather a description of some important aspects to help frame the case study.

One of the challenges of surface missions is the degree to which operations are impacted by a priori unknown and changing environmental conditions. While orbital imagery provides valuable information to guide activity, it does not capture all the conditions that affect the rover. For example, while orbital imagery may indicate that exploring a particular region is promising to achieve a science objective, the specific science targets are not known until additional data is collected from the rover itself.

As such, surface operations must be reactive and respond to the results of activity carried out during the previous sol (Martian day). This daily planning activity is referred to as "tactical" operations and is patterned after the tactical operations developed for the Mars Exploration Rovers (Mishkin et al. 2006).

MSL operations augments this tactical process with "strategic" and "supratactical" phases (Chattopadhyay et al. 2014). Strategic planning focuses on developing long-term plans, typically spanning weeks or months, to achieve high-level objectives. Examples of strategic planning include the development of strategies for exploring a large geographical area or a high-level traverse route for reaching a distant objective. The supratactical stage provides a bridge between the long-term strategic plan and the day-to-day, highly reactive tactical process. The process is designed to coordinate the complex science instruments and manage the constraints and resources required to conduct campaigns.

An Example Sol in the Life of the Rover

To provide an idea of how the team operates the rover, Figure 1 illustrates an example sol of rover activity. This is an example of a typical drive sol, derived from an actual Sol 780 command products. Following are some key aspects of the sol.

The plan for each sol begins with an "Uplink" window in which new commands products may be sent to the vehicle from Earth. There are various downlink windows throughout the sol in which the rover uses orbiter relays to send collected data back to Earth. While there are multiple downlink windows, certain downlinks have increased importance based on the time that data in the relay will reach operators. If data from a relay will reach operators by the start of the next tactical planning shift, then they relay is termed "decisional" because data from the relay can be used to make decisions in for the rover's next plan. Which relays are considered decisional depends on the relative timing between Earth and Mars along with latencies in the orbiter relay process. In Figure 1 the starred "MRO Relay" represents the decisional relay for this sol. It is important to realize that for this plan, only the data collected prior to this pass could be used to inform the next plan. While the remaining data will eventually be sent to Earth and may be used to inform future plans, it will arrive too late to inform the next plan.

Another important aspect of Figure 1 is how the team structures the sol into "blocks" of activity. For example, the main portion of the rover's day consists of a Pre-Drive Science block, a Drive with Mid-Drive Imaging bock and a Post-Drive Imaging block. The block structure organizes activity into related groups and allows a "Master" sequence to enforce timing between these major types of activity. The latter has to do with uncertainties in predicting the duration of activity in the plan. Due to environmental conditions such as lighting, scene content and terrain, the time to perform imaging and drive activities varies. The team uses the

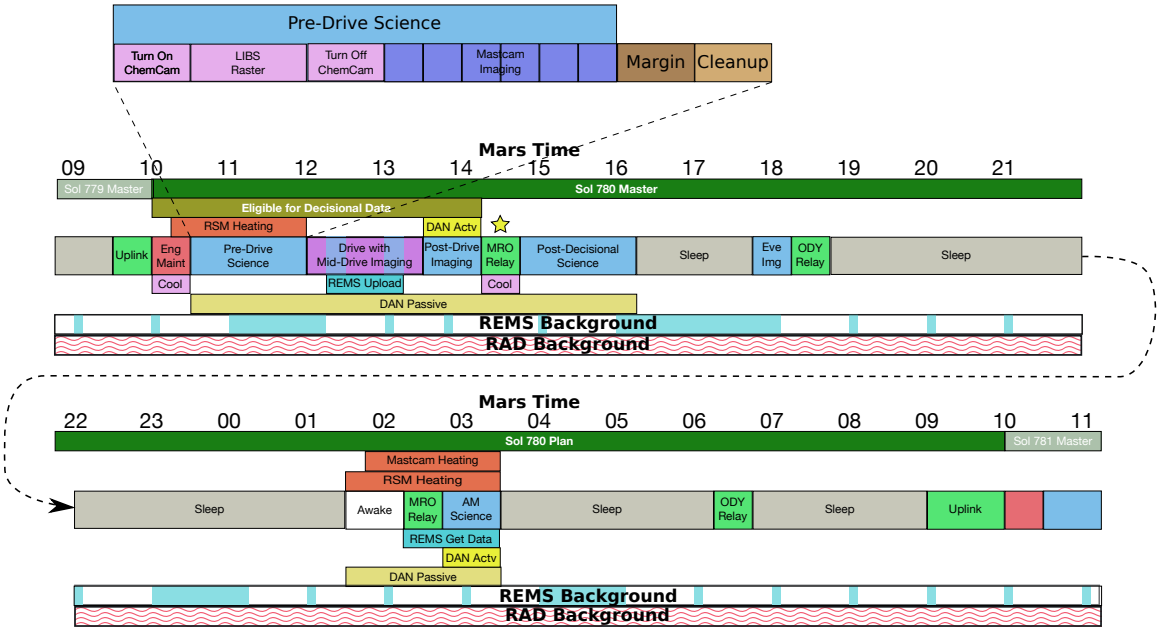


Figure 1: Example sol in the life of the rover.

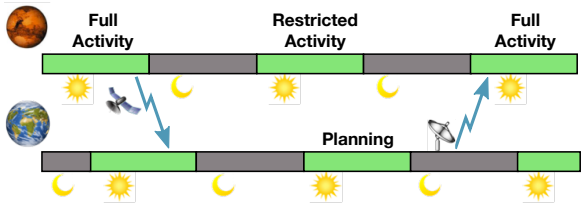


Figure 2: Mars activity vs. Earth planning.

block structure to ensure that if activity in one block runs longer than expected, it can be cut off to avoid interfering with subsequent activity. To protect against loss of data, the team builds "Margin" into each block, to allow activities to run longer than predicted. To deal with cases where durations exceed allocated margin, the team also sequences "Cleanups" after each block, to ensure that any activity is finished before the start of the next block.

Restricted Sols

The vast majority of the surface mission is conducted with the team restricting operations to daytime hours on Earth. The consequence is that the operations team is often out of sync with the activity of the rover on Mars. Figure 2 illustrates the impact this can have on the data available to the team during planning. In the diagram, the end-of-day relay from the rover arrives on the ground late in the Earth day. The team waits until the next Earth day to begin planning. Meanwhile the rover is waking up for its next Mars day without a new set of command products from Earth. By the time the team has completed the tactical process, they must wait for the subsequent Mars morning to uplink the products to the vehicle.

This often limits what the team can command the vehicle to do during the middle sol of Figure 2. If the vehicle were allowed to make significant changes to its state, in particular driving to a new location, this would significantly limit the types of activities the team could command on the subsequent sol. These limited activity sols are referred to as "restricted sols" because the latency of data often restricts the type of activity the team can perform.

A similar situation arises when the team takes days off for weekends and holidays. In these cases, the team will create plans that span multiple sols (aka multi-sol plans). Again, activities that result in significant changes to vehicle state are limited since they will impact the activity that can be done in later sols of the plan.

Given the current way in which we design and operate rovers, restricted sols are a major detractor from mission productivity. For example, with current surface operations, when the rover drives to a new location it must wait for imagery collected at this location to be sent to Earth and for the science and engineering teams to analyze the data and identify the specific set of activities to perform at the location to meet their current mission objectives. If the mission is in a restricted time period, this results in an entire sol in which the rover waits for these new activities.

Overall, 41% of sols on the MSL mission are restricted sols. This percentage is expected to be much higher if the mission were to rely on a highly eccentric relay orbiter such as the MAVEN orbiter.

Resource Prediction

An additional challenge to surface operations is that it is difficult for ground operators to predict the time and energy that will be required to perform these activities and the consequence of over-subscribing resources is severe (e.g. safing

the vehicle if energy is over-subscribed). As such, the team makes conservative estimates which almost always results in significantly under-utilizing available vehicle resources.

In the campaigns we analyzed in our MSL case study, we estimated that the rover could have conducted an additional 3 to 4 hours of activity each sol of the campaigns with the energy that went unused (Gaines et al. 2016). This would have resulted in a dramatic increase in productivity.

Identifying Campaign Intent

The previous section described the significant loss of productivity experienced by surface missions due to restricted sols and the challenge predicting resource usage. Our objective in this work is to increase the autonomy of rovers so that they can remain productive even in situations of reduced contact with human operators. Our approach is to convey mission intent to the rover so that mission operators can provide guidance to the rover even if they do not know the specific state the rover will be in when it receives the guidance. Further, operators will be able to provide a collection of goals that have the potential for over-subscribing available resources and let the rover select a high value subset based on actual available resources.

To enable more autonomous operation of a Mars rover, we aim to capture the intent behind activities planned for the rover during its service of a scientific campaign. Ultimately, the intent of all Mars rover activities is to advance our scientific understanding of the planet. However, by breaking up this larger intent into smaller, well-defined components, we can make some of this knowledge accessible to planning software for decision making. With the intent knowledge carried onboard, along with software that can use it, the rover can be more productive during times when ground interaction is limited.

The first step in this process is to understand the types of science and engineering intent that drives surface missions. We looked at several MSL campaigns in an effort to identify common relationships between planned activities and the objectives they are meant to achieve. Specifically, we investigated relationships that influence the inclusion or exclusion of an activity, as well as relationships that influence the timing of the included activities. From our initial investigation, we found three general types of relationships: samples of a class, temporally periodic observations, and sampling based on changes in rover or environmental state. In this section, we describe each of these types of relationships and the MSL examples that motivate them.

Sampling from a Class

The first relation we discuss is "samples of a class". In this relationship the operations team has some class in mind that they wish study and they want the rover to collect observations of examples of this class. This type of relation may be used to identify general areas to study or it may result is the selection of specific targets.

For example, during the Pahrump Hills Walkabout campaign the team performed a reconnaissance loop with the high level objective of studying a rock formation named

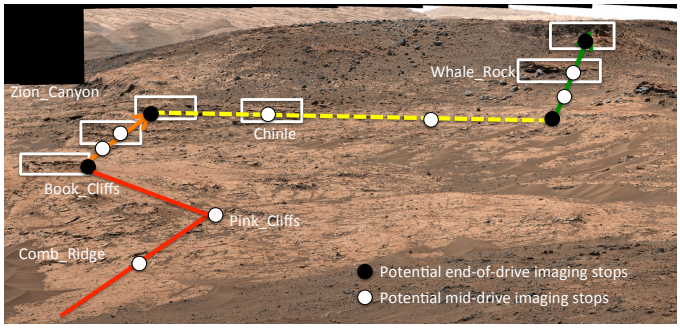


Figure 3: Curiosity's planned route for Pahrump Hills Walkabout Pass 1 at start of campaign.

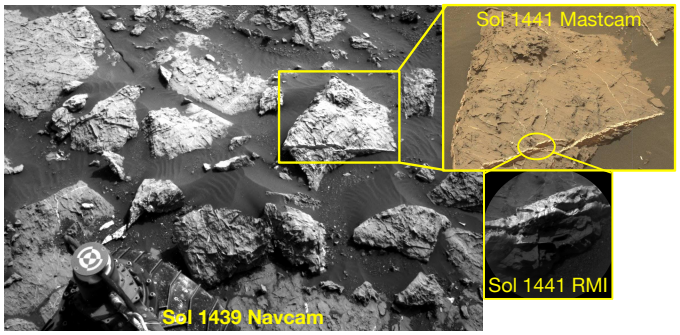


Figure 4: Catabola, an example vein target (NASA/JPL-Caltech/MSSS/LANL/CNES/IRAP/LPGNANTES/CNRS/IAS).

Murray formation, the strata recognized as lower Mount Sharp (Stack et al. 2015). During the strategic planning phase of this campaign, the team used imagery from the rover to identify resistant beds and other examples of unique rock textures that they wished to explore to develop a better understanding of the area. Figure 3 shows these identified areas, white boxes, as well as the initial route that was planned to visit them. These white boxes are an example of general areas that represent samples of class (e.g. resistant beds) the team is interested in studying.

This type of sampling from a class frequently occurs at a more local scale. For example, the team is often interested in studying veins that run through rocks (Nachon et al. 2015). Figure 4 shows the Catabola target, a vein identified in imagery acquired following a drive on Sol 1439. On Sol 1441, the operations commanded the rover to acquire the corresponding Mastcam and ChemCam data which resulted in the detection of high levels of boron (Jet Propulsion Laboratory Press Release 2016a). Notice that although the imagery containing Catabola was first collected on Sol 1439, it was not until Sol 1441 that the follow-up observations were collected. This is because the team was in restricted sols during this time and the rover spent Sol 1440 collecting untargeted observations without ground interaction.

Figure 5 shows another example of sampling from a class. In this case, the team identified two targets that were good examples of light-toned rocks, Elk and Lamoose. These

Figure 5: Navcam image taken after the Sol 991 drive, showing the Elk and Lamoose targets.

turned out to be particularly interesting targets as follow-up observations indicated they contained high levels of silica. The study of the high-silica targets lead to the conclusion that the introduction of silica represented one of the most recent water-rock interactions observed in Gale crater (Frydenvang et al. 2016). The data would also contribute to the study of the subsequently visited Bridger Basin area (Gasda et al. 2016).

In each of these examples, there are typically several instances that would serve as examples of a class. Given the limited time and resources available as well as other objectives the team has, the operations team must select a subset of candidate targets to perform follow-up observations. We would like to explicitly capture this notion of sampling from a class and allow operators to convey the value of collecting multiple samples from the class to enable the rover to make similar trade-offs. For example, it may be important to collect at least 3 samples of a class. Additional samples are nice to have, if additional resources are available, but with reduced additional value.

Temporally-Periodic Sampling

Other relationships between activities result from the desire to sample the Martian terrain and atmosphere at regular time intervals. These sampling intervals can determine how many observations are needed, as well as preferences on the temporal separations between observations. For example, throughout the mission, a variety of periodic observations are collected to monitor the Martian environmental conditions. Example monitoring includes periodically acquiring images of the sun at different times of day to measure atmospheric opacity (Mason et al. 2017). The SAM (Sample Analysis at Mars) is used to perform period sampling of the atmosphere with the intent to monitor the seasonal evolution of methane in the atmosphere (Webster et al. 2015).

While our primary focus on capture intent is for increasing science productivity, there are also many type of engineering activities that have similar intent relationships. For example, the team performs maintenance of different rover subsystems and collects detailed dumps of various vehicle telemetry on periodic cycles. The team also performs a variety of instrument calibration activities including periodic viewing of calibration targets for ChemCam (Chemical Camera), APXS (Alpha Particle X-Ray Spectrometer) and MAHLI (Mars Hand Lens Imager) calibration targets, to name just a few. While these maintenance activities may consume time and resources that would otherwise be used for immediate science, they must be considered to keep the rover safe and healthy for future science activities.

For both science and engineering periodic activities there is a preference on scheduling observations at a particular cadence, but there is flexibility in deviating from a precise interval to allow these periodic activities to be inter-mixed with other objectives vying for the same pool of rover resources. Thus, when capturing periodic intent, we want to express the preferred period as well as how value decreases as particular observations deviate from the preferred timing.

State-Based Sampling

The final type of intent relation we consider are requests that are based on changes in state that occur as the rover operates on the Martian surface. For example, as the rover traverses across the landscape there is interest in collecting certain types of observations including using navigation cameras to perform clast surveys (Yingst et al. 2013) and using the DAN (Dynamic Albedo of Neutrons) instrument to search for signs of subsurface water (Litvak et al. 2013). Due to the complexities of current operations practices the team is limited to acquiring these observations at the end of drives, which have high variance in their lengths, rather than at optimal distances to support systematic sampling. Part of our objective in increasing rover autonomy is to make it easier for the rover to perform these types of surveys closer to their ideal locations.

In recent operations, Curiosity has been climbing Mount Sharp to reach a layer of hematite, as seen in Figure 6. The team is interested in performing systematic surveys along the route in order to study variations that occur up the slope. In this case, the sampling strategy is based on change in elevation rather than strictly distance.

There are also engineering maintenance activities that correspond to changes in rover state. It is well known that the rover's wheels have suffered damage during its explorations. The engineering team conducts a periodic wheel wear monitoring activity based on distance traveled (Jet Propulsion Laboratory Press Release 2016b). The team also has the rover perform an attitude update activity in which the relative location of the sun is combined with accelerometer data to update the rover's attitude knowledge. This attitude

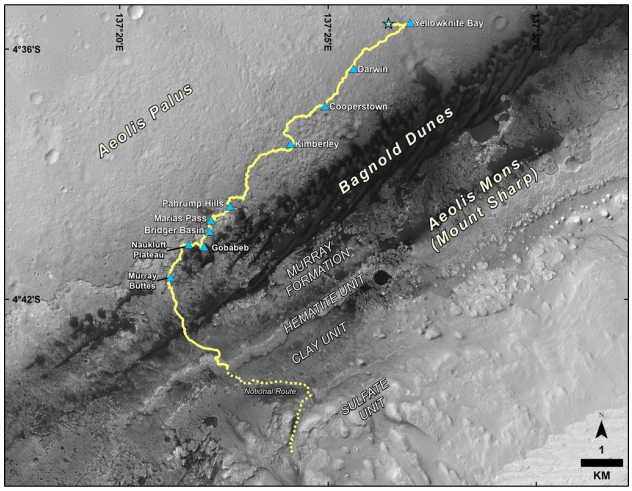


Figure 6: Curiosity's planned route to reach hematite layer (NASA/JPL-Caltech/Univ. of Arizona).

update activity is also scheduled as a function of distance traveled.

As with temporally periodic activities, this state-based periodic activities also come with a preferred cadence with flexibility about the actual timing.

In summary, we have reviewed several MSL science and engineering campaigns to identify patterns of relationships between activities and the objectives they serve. We identified three common types of relationships:

Sampling from a class: goals are selected based on how well they exemplify a class, there is typically an increase in value as more examples are collected, but with diminishing returns after a certain number of samples is reached.

Temporally-periodic sampling: goals are selected and scheduled based on a periodic temporal relationship, there is typically a preference on the cadence but with some amount of allowed flexibility in specific timing.

State-based sampling: goals are selected and scheduled based on changes in state of the rover and/or terrain, there is typically a preference on the cadence but with some amount of allowed flexibility in specific sampling.

In the next section, we discuss these types in more detail, including how they are implemented to capture intent and enable autonomous plan generation and repair.

Expressing Campaign Intent

We define a new type of planning construct, called a Plan Campaign, in order to guide automated planning algorithms towards solutions that better satisfy high-level science campaigns. These Plan Campaigns impose relationships between activities in the plan as a way of capturing the intent of performing those activities. In addition, a Plan Campaign provides an assessment of the current plan, indicating how well it is satisfying the constraints and relationships of the

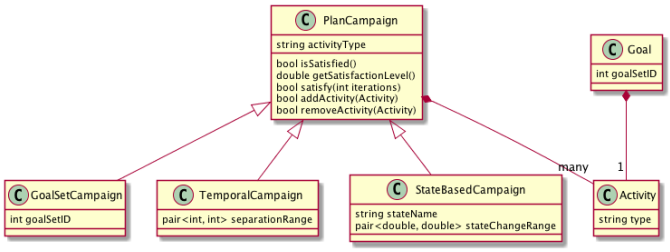


Figure 7: ASPEN classes for the three campaign types.

campaign. This satisfaction level, for example, may be defined in terms of the number of activities in the plan compared to the maximum requested by the Plan Campaign. A Plan Campaign also provides a set of satisfaction methods, which define ways that the plan might be changed to better satisfy the campaign.

We implement three types of Plan Campaigns in the ASPEN planning system (Fukunaga et al. 1997), corresponding to the most common relationships identified in the previous section. They are:

Goal Set Campaign: activities are scheduled based on a specific group of goals that request them. This type can be used to implement the "Sampling from a class" relationships.

Temporal Campaign: activities are scheduled based on temporal separation constraints. This type can be used to implement the "Temporally periodic sampling" relationships.

State-based Campaign: activities are scheduled based on state-change separation constraints. This type can be used to implement the "State-based sampling" relationships.

Each plan campaign type has its own definition for satisfaction level, and its own satisfaction method. A partial class diagram can be seen in Figure 7.

A Goal Set Campaign is a request to include in the plan some of the goals from a predefined set, such as those to sample a rock classification. In ASPEN, a goal is a request for an activity with specific parameter settings. For example, one rover goal may be to observe a particular rock at a close distance using stereo cameras. A goal might also generate new goals, such as when the intent is for the rover to collect measurements of some rock outcrop, and the particular location of the measurement can be generated onboard after approaching the outcrop and acquiring more detailed images not yet available on the ground for targeting. Each goal can be assigned an ID to specify the goal set to which it belongs. The Goal Set Campaign then references the same ID to request the set of goals. It also has parameters to specify the minimum and maximum number of goals to select from the set, which become constraints in ASPEN. The level of satisfaction increases as more goals are added to the plan. We use this type of Plan Campaign, for example, to capture a set of targeted observations for the rover that may all investigate the same type of rock formation. Multiple Goal Set Campaigns could be used for different types of formations, with

each competing for time and resources in the plan. For example, driving from one formation to another will take time and energy, but may provide more value than additional observations at the current location.

A Temporal Campaign is a request to include activities at regular time intervals. Again, a minimum and maximum number of activities can be provided. This type of campaign is very similar to the "Repeat" campaign type used to schedule Rosetta science observations (Chien et al. 2015). In addition, a minimum and maximum temporal separation is specified and captured as an ASPEN constraint to ensure an even sampling. Environmental monitoring, such as measuring dust levels in the atmosphere, is often specified in this way. In this case, the satisfaction level will be a function of not only the number of activities, but how well they are spaced.

The last, called a State-based Campaign, is the type of campaign implemented to support sampling based on rover state. In planning systems, state predictions are often plotted on a timeline. This prediction can then be used to determine where to schedule activities that were requested by the State-based Campaign. For example, as we include drives in the plan, we can predict the total distance traveled along a timeline. If a request was made to sample every 100 meters, these sampling activities can be placed on the timeline where the drive distance changes by 100m. A range of acceptable spacings can be provided using minimum and maximum parameters in the State-based Campaign definition. This, for example, would allow a campaign to request samples between 90m and 110m apart. Note that the drive, which may have been added to support a different campaign, may need to be interrupted to perform the sampling. Again, campaigns will compete for time and resources, and the satisfaction level of each campaign can be used to make planning decisions for requested activities. As with the Temporal Campaign, the State-based Campaign satisfaction level will partly depend on activity spacing. In this case, however, the spacing is driven by the change in a particular attribute of the rover state. Requested activities can be moved along planned state changes, or other activities can be used to explicitly change the planned state. For example, a drive could be added solely to increase spatial sampling, if no specific target location is provided.

All three types of Plan Campaigns have been implemented as extended features in the ASPEN planning and scheduling system. Problem-specific campaigns definitions are specified along with activity definitions in the ASPEN Modeling Language (AML). Once a set of campaigns have been provided, ASPEN scheduling functions can be used to generate campaign activities, repair campaign constraint violations, or optimize campaign satisfaction.

Using Campaign Intent

In this section, we discuss how captured campaign intent can be used for activity planning, as well as autonomous goal generation.

Activity Planning

One of the primary reasons for capturing and expressing campaign intent is to provide guidance for on-board plan generation and repair. During plan generation, the objective is to create and schedule activities that best satisfy all campaigns according to their preferences and relative priorities. During plan repair, the objective is to change the plan to better serve the campaigns in light of new state information. While plan generation may be done at regular times of relatively low activity (e.g. during the night), plan repair will most likely be triggered during execution when a new plan is needed quickly to keep the rover busy. We describe the algorithms implemented for generating and repairing plans based on a set of input campaigns that have been expressed in the manner discussed in the previous section.

Our approach to plan generation is based on branch-and-bound search. Here, various options (i.e. branches) are created from a partially generated plan, starting with the empty plan. As they are generated, the various options are evaluated and pruned based on a threshold (i.e. bound) on plan quality. Specifically, we compute the best possible quality of any plan that could be built from a new partial plan, and compare this to the worst possible quality of partial plans that have already been considered. If the new option can do no better, then that option, and all possibilities that extend from it, are pruned. The quality metric used for pruning is also used to periodically sort the options under consideration. After sorting, the partial plans that have the highest potential quality are expanded first. This is often referred to as "best-first" search. Quality of a partial plan is based on the level of satisfaction for the prioritized set of campaigns. Finally, after being selected, a partial plan is expanded forward in time. For periodic campaigns (Temporal and State-based), this means that the earliest activity is added first, while the next expansion will schedule the next activity based on the campaign separation constraint. For non-periodic campaigns (Goal Set), the partial plan is expanded multiple times, one for each goal scheduled to occur after the periodic activities scheduled so far. In the end, the search will evaluate all possible combinations of including or excluding campaign activities, as well as all possible orderings of those included.

While this can be time-intensive, it is guaranteed to find the optimal plan as defined by the campaign preferences and priorities. Further, a time limit can be placed on the search, to ensure that a plan is returned in a timely manner. Although this may not be a globally optimal plan, it will enable the rover to continue to be productive, and can be adjusted by more time-efficient repair strategies.

For plan repair, we use a greedy, local search algorithm to make fast improvements to the plan (Rabideau, Engelhardt, and Chien 2000). This algorithm can be run iteratively as the plan executes and new state information is received. On each iteration, the campaigns and their preferences are evaluated. From this, one preference is greedily selected based on its prioritized contribution to plan quality. An example might be a pair of activities occurring hours apart that are contributing to a high-priority campaign that is requesting periodic activities with a one hour separation.

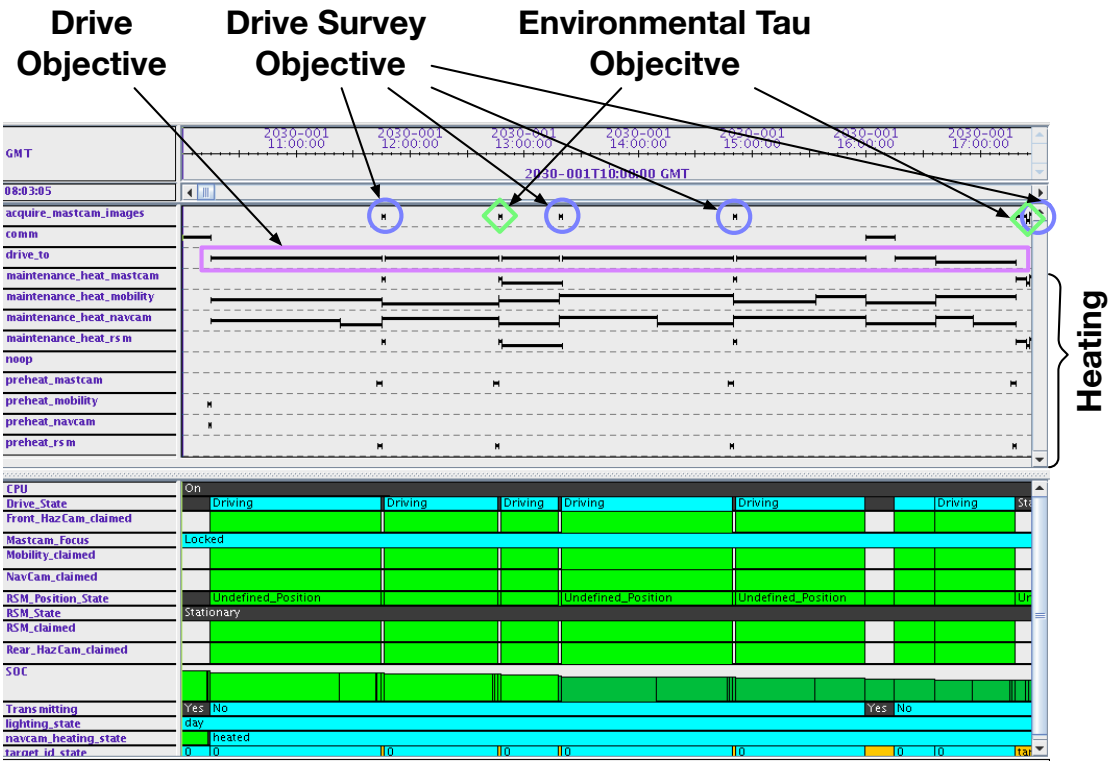


Figure 8: Example generated plan illustrating a long-range drive objective that was split up to support two different types of campaign objectives.

Once a poor-performing preferences is selected, an attempt is made to modify the plan to better satisfy the preference. For the periodic example, this would simply mean moving one of the activities closer to the other. As another example, consider a state-based campaign that is requesting observations every 100 meters as the rover drives. If a drive takes longer than expected, future observations can be postponed to better match the 100m separation preference. While local improvements to quality issues may be sub-optimal, the response time can be much shorter, making this method more suitable during execution.

Figure 8 shows an example plan generated by our system. The planning model is derived from the activity model used for MSL rover operations and includes important aspects of the mission such as science activities, communication windows and device heating. The example illustrates how the plan generator uses provided campaign relationships to coordinate rover activity.

For this example, the rover was given a long range drive objective along with two different campaign relationships: acquire environmental tau (atmospheric opacity measurement) observations every 3 hours (Temporal Campaign) and perform a mid-drive survey activity every 75 meters (State-Based Campaign). The resulting plan shows the drive objective being paused at different points to support interleaving these campaign activities.

Autonomous Goal Generation

In addition to plan generation and repair, the other use of campaign objectives in our system is to identify new goals for the system based on scientist guidance. This is applicable in cases where the operations team does not have up to date information of the area around the rover but want the rover to continue performing productivity activities. As discussed in the Background section, this situation can arise during restricted sol phases of the mission.

A high-level campaign goal can be used to generate more specific goals using onboard software. For example, suppose scientists are interested in remote-sensing, compositional measurements of a rock formation seen previously and known to exist in a region the rover is approaching. Using the TextureCam software (Thompson et al. 2012), scientists can train a model to detect the rock formation using labeled examples of the formation in previous navigation camera images (Figure 9, left). Then, upon driving into the new region, the rover can run TextureCam onboard to compute a probability map of the regions most likely to contain that rock formation (Figure 9, center). The probability map can be used to select the best locations for measurement, as well as the likelihood that each measurement satisfies the scientific intent of measuring the rock formation (Figure 9, right). Each measurement becomes a new goal, and the planner can use the probability information to reason about the tradeoffs between acting upon the various generated goals.

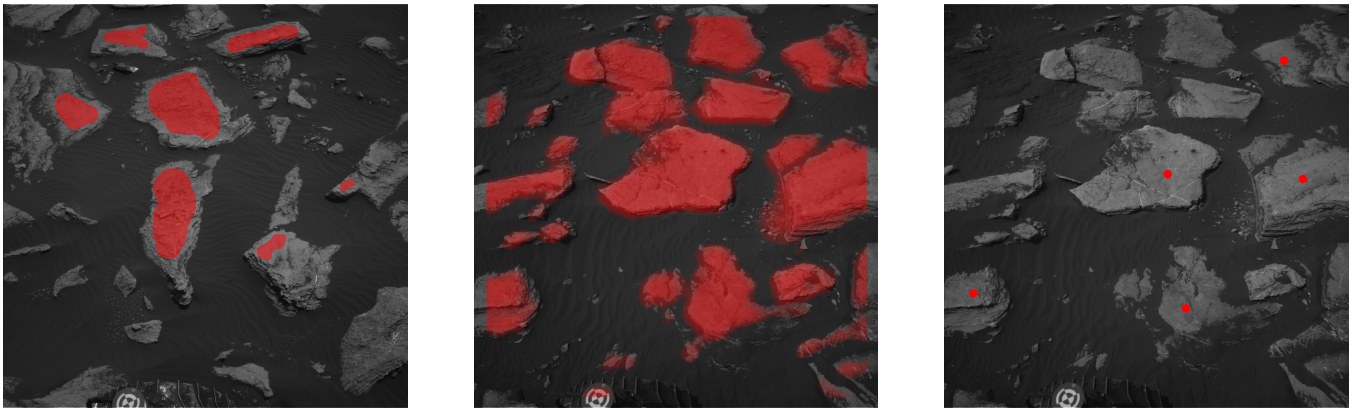


Figure 9: An example showing how scientists can use TextureCam to express intent to autonomously generate new goals on board. The left image shows hand-labeled regions of a geological formation of interest. The center image shows the estimated probabilities that regions in a new image are of the same formation, given a model trained from labels. The right image shows the top five software-selected locations for diverse observations of the rock formation, each corresponding to a new goal for the planning system.

Related Work

Shalin, Wales, & Bass, (2005) conducted a study of Mars Exploration Rovers operations to design a framework for expressing the intent for observations requested by the science teams. Their focus was the use of intent to coordinate planning among human operators and the resulting intent was not captured in a manner that would be conducive for machine interpretation. Our approach codifies some of the fields in their framework in a way suitable for the rover. In particular, the authors defined a "Related Observations" field as a way for scientists to identify relationships among different observations, which need not be in the same plan. Our work on campaign intent can be seen as a way of defining a specific semantics to these types of relationships to facilitate reasoning about these relationships by the rover.

Their framework also includes information that we agree is essential for effective communication among operators but that we do not currently express to the rover. For example, the "Scientific Hypotheses" field is used to indicate what high-level campaign objective is being accomplished by the requested observation. We are not yet providing these higher-level campaign objectives to the rover, though it is an interesting area of future research.

Mali (2016) views intent as a means for a user to place constraints on the types of plans a planner is allowed to produce such as only generating plans that have at most one instance of a class of actions or that plans must limit the use of a particular action. The primary role of our use of intent is to allow the planner to assess the value of achieving a given set of goals. However, some of our campaign intent does imply constraints and preferences on how, or more specifically, when goals are accomplished. For example, the periodic campaign intent specifies a timing relationship among goals and a preference on how close to comply with the desired timing.

There are some similarities between our campaign definitions and those used for Rosetta science planning (Chien et al. 2015). Both use campaigns to express requests for variable-sized groups of observations with relationships and priorities. Rosetta plans covered much longer time periods (e.g. weeks) and required more complex temporal patterns, such as repeating groups of observations. But observation patterns were primarily driven by the predictable trajectory of the spacecraft, allowing relationships to be expressed as temporal constraints. This is not sufficient for rovers, where many observations are dictated by the rover location and surrounding terrain, and the duration of many activities cannot be accurately predicted. State-based and goal set relationships more accurately represent some of the science intent found on surface missions.

There have been a variety of autonomous science systems deployed or proposed for rovers include AEGIS system running on the Opportunity and Curiosity rovers (Francis1 et al. 2016), and the SARA component proposed for an ExoMars rover (Woods et al. 2009). These systems allow the rover to identify targets in its surroundings that match scientist-provided criteria. The introduction of campaign relationships broadens the scope of the type of guidance that scientists can provide these systems, allowing scientists to express the amount of observations they would like for their different objectives along with the relative priorities of the high-level objectives.

The ProViScout project has similar objectives to our work (Paar et al. 2012). ProViScout is an integrated system to conduct planetary scouting and exploration. It includes autonomous science capabilities to enable onboard identification of science targets. Similar to our approach, the system selects follow-up observations for identified targets and submits these requests to an onboard planner to determine if there are sufficient resources to accomplish these new objectives. The campaign intent concepts we have developed would also be applicable to ProViScout as a way to increase the expressivity for providing scientist intent to the rover.

There is an active area of research in intent recogni-

tion (Sukthankar et al. 2014). The general goal of this area is to identify the objectives of other agents (human or otherwise) from observations of the agents' actions. In contrast, in our work, it is acceptable for users to explicitly identify their intent, rather than require the system to attempt to infer intent. Indeed, there is interest in the operations team to clearly document their intent for the purpose of communication among teams and as a record of what activity was planned for the rover and why. As such, rather than try to infer user intent, our objective is to increase the expressivity of the rover's interface in order to more closely reflect mission intent.

Conclusions

We have discussed a formalism for encoding aspects of science campaign intent in a manner suitable for reasoning by a rover. Campaign intent specifies relations among goals. These relations provide guidance to the rover to enable it to select a high-value subset of goals to accomplish among a set of goals that oversubscribe available resources. Further, they provide guidance to the rover when it identifies its own goals to work on.

We have begun implementing this campaign intent framework within the ASPEN system and integrating it with a research rover at JPL. Over the next years we will be conducting mission-relevant, multi-sol scenarios with the rover at the JPL Mars Yard to evaluate its ability to support productive operations with limited ground-in-the-loop interactions.

Acknowledgments

This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was funded by the Jet Propulsion Laboratory Research and Technology Development program.

References

[Benowitz 2016] Benowitz, E. 2016. Concepts for the mars 2020 rover onboard scheduler. In *Proceedings of the 2016 Workshop on Spacecraft Flight Software (FSW-16)*.

[Chattopadhyay et al. 2014] Chattopadhyay, D.; Mishkin, A.; Allbaugh, A.; Cox, Z. N.; Lee, S. W.; Tan-Wang, G.; and Pyrzak, G. 2014. The Mars Science Laboratory supratactical process. In *Proceedings of the SpaceOps 2014 Conference*.

[Chien et al. 2005] Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; Shulman, S.; and Boyer, D. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication* 2:196–216.

[Chien et al. 2015] Chien, S.; Rabideau, G.; Tran, D.; Doubleday, J.; Nespoli, F.; Ayucar, M.; Sitje, M.; Vallat, C.; Geiger, B.; Altobelli, N.; Fernandez, M.; Vallejo, F.; Andres, R.; and Kueppers, M. 2015. Activity-based scheduling of science campaigns for the rosetta orbiter. In *Proceedings of IJCAI 2015*.

[Francis1 et al. 2016] Francis1, R.; Estlin, T.; Gaines, D.; Doran, G.; Gasnault, O.; Johnstone, S.; Montano, S.; Mousset, V.; Verma, V.; Bornstein, B.; Burl, M.; Schaffer, S.; and Wiens, R. C. 2016. AEGIS intelligent targeting deployed for the Curiosity rover's ChemCam instrument. In *Proceedings of the 47th Lunar and Planetary Science Conference*.

[Frydenvang et al. 2016] Frydenvang, J.; Gasda, P.; Hurowitz, J.; Grotzinger, J.; Wiens, R.; Newsom, H.; Bridges, J.; Gasnault, O.; Maurice, S.; Fisk, M.; Ehlmann, B.; Watkins, J.; Stein, N.; Forni, O.; Mangold, N.; Cousin, A.; Clegg, S.; Anderson, R.; Payr, V.; Rapin, W.; Vaniman, D.; Morris, R.; Blake, D.; Gupta, S.; Sautter, V.; Meslin, P.-Y.; Edwards6, P.; M.Rice; Kinch, K.; Milliken, R.; Gellert, R.; Thompson, L.; Clark, B.; Edgett, K.; Sumner, D.; Fraeman, A.; Madsen, M.; Mitrofanov, I.; Jun, I.; Calef, F.; and Vasavada, A. 2016. Discovery of silica-rich lacustrine and eolian sedimentary rocks in Gale crater, Mars. In *Proceedings of the 47th Lunar and Planetary Science Conference*.

[Fukunaga et al. 1997] Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Towards an application framework for automated planning and scheduling. In *International Symposium on Artificial Intelligence, Robotics and Automation for Space*.

[Gaines et al. 2016] Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; Mackey, R.; and Estlin, T. 2016. Productivity challenges for Mars rover operations: A case study of Mars Science Laboratory operations. Technical Report D-97908, Jet Propulsion Laboratory.

[Gasda et al. 2016] Gasda, P. J.; Frydenvang, J.; Wiens, R. C.; Grotzinger, J. P.; Watkins, J. A.; Stein, N.; Edgett, K. S.; Newsom, H.; Clark, B.; Anderson, R.; Bridges, N.; Clegg, S.; and Maurice, S. 2016. Potential link between high-silica diagenetic features in both eolian and lacustrine rock units measured in Gale crater with MSL. In *Proceedings of the 47th Lunar and Planetary Science Conference*.

[Jet Propulsion Laboratory Press Release 2016a] Jet Propulsion Laboratory Press Release. 2016a. Boron in calcium sulfate vein at 'Catabola,' Mars. `http://mars.nasa.gov/multimedia/images/2016/boron-in-calcium-sulfate-vein-at-catabola-mars&s=2`.

[Jet Propulsion Laboratory Press Release 2016b] Jet Propulsion Laboratory Press Release. 2016b. Routine inspection of rover wheel wear and tear. `http://www.jpl.nasa.gov/spaceimages/details.php?id=PIA20334`.

[Litvak et al. 2013] Litvak, M.; Mitrofanov, I.; Behar, A.; Boynton, W.; Deflores, L.; Fedosov, F.; Golovin, D.; Hardgrove, C.; Harshman, K.; Jun, I.; Kozyrev, A.; Kuzmin, R.; Lisov, D.; Malakhov, A.; Milliken, R.; Mischna, M.; Moersch, J.; Mokrousov, M.; Nikiforov, S.; Sanin, A.; Shvetsov, V.; Starr, R.; Tate, C.; VİṪret'yakov; Varenikov, A.; and Vostrukhin, A. 2013. The water bulk distribution along MSL Curiosity traverse measured by dan instrument. In *Proceedings of the European Planetary Science Congress*, volume 8.

[Mali 2016] Mali, A. D. 2016. Expressing user intent in planning by instance rewriting. In *Proceedings of the 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI 2016)*.

[Mason et al. 2017] Mason, E.; Lemmon, M.; de la Torre, M.; and Smith, M. 2017. A quick look estimation of optical depth measurements from the rover environmental monitoring station ultraviolet sensors. In *Proceedings of the Sixth International Workshop on the Mars Atmosphere: Modelling and Observations*.

[Mishkin et al. 2006] Mishkin, A. H.; Limonadi, D.; Laubach, S. L.; and Bass, D. S. 2006. Working the Martian night shift: The MER surface operations process. *IEEE Robotics and Automation Magazine* 13(2):46–53.

[Muscettola et al. 1998] Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103(1-2):5–47.

[Nachon et al. 2015] Nachon, M.; Mangold, N.; Cousin, A.; Forni, O.; Anderson, R. B.; Blank, J.; Calef, F.; Clegg, S.; Fabre, C.; Fisk, M.; Gasnault, O.; Kah, L.; Kronyak, R.; Lasue, J.; Meslin, P.-Y.; Moulic, S. L.; Maurice, S.; Oehler, D.; Payre, V.; Rapin, W.; Sumner, D.; Stack, K.; Schrder, S.; and Wiens, R. 2015. Diagenetic features analysed by ChemCam/Curiosity at Pahrump Hills, Gale crater, Mars. In *Proceedings of the European Planetary Science Congress*, volume 10.

[Paar et al. 2012] Paar, G.; Woods, M.; Gimkiewicz, C.; Labrosse, F.; Medina, A.; Tyler, L.; Barnes, D. P.; Fritz, G.; and Kapellos, K. 2012. PRoViScout: a planetary scouting rover demonstrator. In *Proceedings of SPIE Vol. 8301 ntelligent Robots and Computer Vision XXIX: Algorithms and Techniques*.

[Rabideau, Engelhardt, and Chien 2000] Rabideau, G.; Engelhardt, B.; and Chien, S. 2000. Using generic preferences to incrementally improve plan quality. In *International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*.

[Shalin, Wales, and Bass 2005] Shalin, V. L.; Wales, R. C.; and Bass, D. S. 2005. Communicating intent for planning and scheduling tasks. In *Proceedings of HCI International*.

[Stack et al. 2015] Stack, K. M.; Grotzinger, J. P.; Gupta, S.; Kah, L. C.; Lewis, K. W.; McBride, M. J.; Minitti, M. E.; Rubin, D. M.; Schieber, J.; Sumner, D. Y.; Beek, L. M. T. J. V.; Vasavada, A. R.; and Yingst, R. A. 2015. Sedimentology and stratigraphy of the Pahrump Hills outcrop, lower Mount Sharp, Gale Crater, Mars. In *Proceedings of the 46th Lunar and Planetary Science Conference*.

[Sukthankar et al. 2014] Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; and Bui, H. H. 2014. An introduction to plan, activity, and intent recognition. In Sukthankar, G.; Goldman, R.; Geib, C.; Pynadath, D.; and Bui, H. H., eds., *Plan, Activity, and Intent Recognition,*. Elsevier.

[Thompson et al. 2012] Thompson, D. R.; Abbey, W.; Allwood, A.; Bekker, D.; Bornstein, B.; Cabrol, N. A.; Castano, R.; Estlin, T.; Fuchs, T.; and Wagstaff, K. L. 2012. Smart cameras for remote science survey. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

[Webster et al. 2015] Webster, C. R.; Mahaffy, P. R.; Atreya, S. K.; Flesch, G. J.; Mischna, M. A.; Meslin, P.-Y.; Farley, K. A.; Conrad, P. G.; Christensen, L. E.; Pavlov, A. A.; Martn-Torres, J.; Zorzano, M.-P.; McConnochie, T. H.; Owen, T.; Eigenbrode, J. L.; Glavin, D. P.; Steele, A.; Malespin, C. A.; Jr., P. D. A.; Sutter, B.; Coll1, P.; Freissinet, C.; McKay, C. P.; Moores, J. E.; Schwenzer, S. P.; Bridges, J. C.; Navarro-Gonzalez, R.; Gellert, R.; and Lemmon, M. T. 2015. Mars methane detection and variability at Gale crater. *Science* 347(6220):415–417.

[Woods et al. 2009] Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an ExoMars roverlike mission. *Journal of Field Robotics* 26(4):358–390.

[Yingst et al. 2013] Yingst, R. A.; Goetz, W.; Hamilton, V.; Hipkin, V.; Kah, L.; Madsen, M.; Newsom, H.; Williams, R.; Bridges, J.; Frias, J. M.; and King, P. 2013. Characteristics of pebble and cobble-sized clasts along the Curiosity rover traverse from Sol 0 to 90. In *Proceedings of the 44th Lunar and Planetary Science Conference*.

STRIPS Planning in Infinite Domains

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139, USA
{caelan,tlp,lpk}@csail.mit.edu

Abstract

Many robotic planning applications involve continuous actions with highly non-linear constraints, which cannot be modeled using modern planners that construct a propositional representation. We introduce STRIPStream: an extension of the STRIPS language which can model these domains by supporting the specification of blackbox generators to handle complex constraints. The outputs of these generators interact with actions through possibly infinite streams of objects and static predicates. We provide two algorithms which both reduce STRIPStream problems to a sequence of finite-domain planning problems. The representation and algorithms are entirely domain independent. We demonstrate our framework on simple illustrative domains, and then on a high-dimensional, continuous robotic task and motion planning domain.

Introduction

Many important planning domains naturally occur in continuous spaces involving complex constraints among variables. Consider planning for a robot tasked with organizing several blocks in a room. The robot must find a sequence of *pick*, *place*, and *move* actions involving continuous robot configurations, robot trajectories, block poses, and block grasps. These variables must satisfy highly non-linear kinematic, collision, and motion constraints which affect the feasibility of the actions. Each constraint typically requires a special purpose procedure to efficiently evaluate it or produce satisfying values for it such as an inverse kinematic solver, collision checker, or motion planner.

We propose an approach, called STRIPStream, which can model such a domain by providing a generic interface for blackbox procedures to be incorporated in an action language. The implementation of the procedures is abstracted away using *streams*: finite or infinite sequences of objects such as poses, configurations, and trajectories. We introduce the following two additional stream capabilities to effectively model domains with complex predicates that are only true for small sets of their argument values:

- **conditional streams**: a stream of objects may be defined as a function of other objects; for example, a stream of possible positions of one object given the position of another object that it must be on top of or a stream of possible settings of parameters of a factory machine given desired properties of its output.
- **certified streams**: streams of objects may be declared not only to be of a specific type, but also to satisfy an arbitrary conjunction of predicates; for example, one might define a certified conditional stream that generates positions for an object that satisfy requirements that the object be on a surface, that a robot be able to reach the object at that position, and that the robot be able to see the object while reaching.

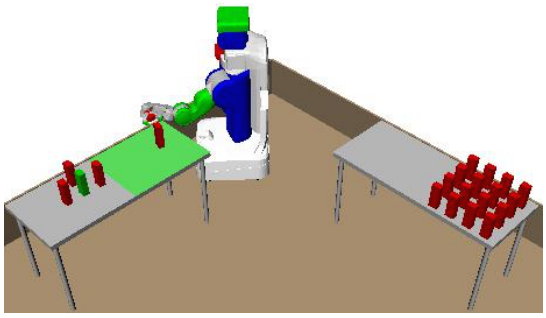


Figure 1: Problem 2-16.

Through streams, STRIPStream can compactly model a large class of continuous, countably infinite, and large finite domains. By conditioning on partial argument values and using sampling, it can even effectively model domains where the set of valid action argument values is lower dimensional than the possible argument space. For example, in our robotics domain, the set of inverse kinematics solutions for a particular pose and grasp is much lower dimensional than the full set of robot configurations. However, using a conditional stream, we can specify an inverse kinematics solver which directly samples from this set given a pose and grasp.

The approach is entirely domain-independent, and reduces to STRIPS in the case of finite domains. The only additional requirement is the specification of a set of streams that can generate objects satisfying the static predicates in the domain. It is accompanied by two algorithms, a simple and a focused version, which operate by constructing and solving a sequence of STRIPS planning problems. This strat-

egy takes advantage of the highly optimized search strategies and heuristics that exist for STRIPS planning, while expanding the domain of applicability of those techniques. Additionally, the focused version can efficiently solve problems where using streams is computationally expensive by carefully choosing to only call potentially useful streams.

Related work

There are a number of existing general-purpose approaches to solving planning problems in infinite domains, each of which has some significant limitation when modeling our robot domain.

Temporal planning, such as defined in PDDL2.1 (Fox and Long 2003), is often formulated in terms of linear constraints on plan variables and is typically solved using techniques based on linear programming (Hoffmann and others 2003; Coles *et al.* 2013). PDDL+ (Fox and Long 2006) extends PDDL2.1 by introducing exogenous events and continuous processes. Although PDDL+ supports continuous variables, the values of continuous variables are functions of the sequence of discrete actions performed at particular times. Thus, time is the only truly non-dependent continuous variable. In contrast, our motivating robot domain has no notion of time but instead a continuously infinite branching factor. Many planners solve PDDL+ problems with nonlinear process models by discretizing time (Della Penna *et al.* 2009; Piotrowski *et al.* 2016). Some recent planners can solve PDDL+ problems with polynomial process models exactly without time discretization (Bryce *et al.* 2015; Cashmore *et al.* 2016). However, even in simplified robotics domains that PDDL+ can model, modern PDDL+ planners are ineffective at planning with collision and kinematic constraints (both highly non-polynomial constraints), particularly in high-dimensional systems.

General-purpose lifted and first-order approaches, such as those based on first-order situation calculus or Prolog, provide semi-decision procedures for a large class of lifted planning problems. However, the generality tends to come at a huge price in efficiency and these planning strategies are rarely practical.

The Answer Set Programming (ASP) literature contains analysis on reasoning in infinite domains through finitary, ω-restricted, finitely ground, and finite domain ASPs (Bonatti *et al.* 2010). The DLV-Complex system (Calimeri *et al.* 2009) is able to solve feasible finitely ground programs by extending the DataLog with Disjunction (DLV) system to support functions, lists, and set terms. We believe that the language of ASP allows specification of conditional and certified streams. However, the ground ASP solver still has to address a much more general and difficult problem and will not have the appropriate heuristic strategies that make current domain-independent STRIPS planners so effective.

Semantic attachments (Dornhege *et al.* 2009), predicates computed by an external program, also provide a way of integrating blackbox procedures and PDDL planners. Because semantic attachments take a state as input, they can only be used in forward state-space search. Furthermore, semantic attachments are ignored in heuristics. This results in poor planner performance, particularly when the attachments are expensive to evaluate such as in robotics domains. Finally, because semantic attachments are restricted to be functions, they are unable to model domains with infinitely many possible successor states.

Many approaches to robotics planning problems, including motion planning and task-and-motion planning, have developed strategies for handling continuous spaces that go beyond *a priori* discretization. Several approaches, for example (Kaelbling and Lozano-Pérez 2011; Erdem *et al.* 2011; Srivastava *et al.* 2014; Garrett *et al.* 2015; Dantam *et al.* 2016; Garrett *et al.* 2016), have been suggested for integrating these sampling-based robot motion planning methods with symbolic planning methods. Of these approaches, those able to plan in realistic robot domains have typically been quite special purpose; the more general purpose approaches have typically been less capable.

Representation

In this section we describe the representational components of a planning domain and problem, which include static and fluent predicates, operators, and streams. *Objects* serve as arguments to predicates and as parameters to operators; they are generated by streams.

A *static predicate* is a predicate which, for any tuple of objects, has a constant truth value throughout a problem instance. Static predicates generally serve to represent constraints on the parameters of an operator. We restrict static predicates to only ever be mentioned positively because, in the general infinite case, it is not possible to verify that a predicate does not hold.

An *operator* schema is specified by a tuple of formal parameters $(X_1, \ldots, X_n)$ and conjunctions of static positive preconditions **stat**, fluent literal preconditions **pre**, and fluent literal effects **eff** and has the same semantics as in STRIPS. An operator instance is a ground instantiation of an operator schema with objects substituted in for the formal parameters. When necessary, we augment the set of operator schemas with a set of axioms that naively use the same schema form as operators. We assume the set of axioms can be compiled into a set of derived predicates as used in PDDL.

A *generator* $g = \langle \bar{o}^1, \bar{o}^2, ... \rangle$ is a finite or infinite sequence of object tuples $\bar{o} = (o_1, ..., o_n)$. The procedure $\text{NEXT}(g)$ returns the next element in generator g and returns the special object **None** to indicate that the stream has been exhausted and contains no more objects. A *conditional generator* $f(\bar{x})$ is a function from $\bar{x} = x_1, ..., x_n$ to a generator $g_{\bar{x}}$ which generates tuples from a domain not necessarily the same as the domain of $\bar{x}$.

An *stream* schema, $\sigma(\bar{Y} \mid \bar{X})$, is specified by a tuple of input parameters $\bar{X} = (X_1, ..., X_m)$, a tuple of output parameters $\bar{Y} = (Y_1, ..., Y_n)$, a conditional generator **gen** $= f(\bar{X})$ defined on $\bar{X}$, a conjunction of input static atoms **inp** defined on $\bar{X}$, and a conjunction of output static atoms **out** defined on $\bar{X}$ and $\bar{Y}$. The conditional generator f is a function, implemented in the host programming language, that returns a generator such that, for all $\bar{x}$ satisfying the conditions **inp**, $\forall \bar{y} \in f(\bar{x}), (\bar{x}, \bar{y})$ satisfy the conditions **out**. A stream instance is a ground instantiation of a stream schema with objects substituted in for input parameters $(X_1, \ldots, X_n)$; it is

conditioned on those object values and, if the **inp** conditions are satisfied, then it will generate a stream of tuples of objects each of which satisfies the certification conditions **out**.

The notion of a conditional stream is quite general; there are two specific cases that are worth understanding in detail. An *unconditional stream* $\sigma(\bar{Y} \mid ())$ is a stream with no inputs whose associated function f returns a single generator, which might be used to generate objects of a given type, for example, independent of whatever other objects are specified in a domain. A *test stream* $\sigma(() \mid \bar{X})$ is a degenerate, but still useful, type of stream with no outputs. In this case, $f(X_1, ..., X_m)$ contains either the single element $()$, indicating that the **inp** conditions hold of $\bar{X}$, or contains no elements at all, indicating that the **inp** conditions do not hold of $\bar{X}$. It can be interpreted as an implicit Boolean test.

A *planning domain* $\mathcal{D} = (\mathcal{P}_s, \mathcal{P}_f, \mathcal{C}_0, \mathcal{A}, \mathcal{X}, \Sigma)$ is specified by finite sets of static predicates $\mathcal{P}_s$, fluent predicates $\mathcal{P}_f$, initial constant objects $\mathcal{C}_0$, operator schemas $\mathcal{A}$, axiom schemas $\mathcal{X}$, and stream schemas Σ. Note that the initial objects (as well as objects generated by the streams) may in general not be simple symbols, but can be numeric values or even structures such as matrices or objects in an underlying programming language. They must provide a unique ID, such as a hash value, for use in the STRIPS planning phase.

A STRIPS*tream problem* $\Pi = (\mathcal{D}, O_0, s_0, s_*)$ is specified by a planning domain $\mathcal{D}$, a finite set of initial objects O_0, an initial state composed of a finite set of static or fluent atoms s_0, and a goal set defined to be the set of states satisfying fluent literals s_*. We make a version of the closed world assumption on the initial state s_0, assuming that all true fluents are contained in it. This initial state will not be complete: in general, it will be impossible to assert all true static atoms when the universe is infinite.

Let $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ be the universe of all objects and the set of true initial atoms that can be generated from a finite set Σ of stream schemas, a finite set $C_0 \cup O_0$ of initial objects, and initial state s_0. We give all proofs in the the appendix.

Theorem 1. *$\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ are recursively enumerable (RE).*

A *solution* to a STRIPStream problem Π is a finite sequence of operator instances π_* with object parameters contained within $\mathcal{O}_\Pi$ that is applicable from $\mathcal{S}_\Pi$ and results in a state that satisfies s_*. STRIPStream is undecidable but semi-decidable, so we restrict our attention to feasible instances.

Theorem 2. *The existence of a solution for a* STRIPS*tream problem* Π *is undecidable.*

Theorem 3. *The existence of a solution for a* STRIPS*tream problem* Π *is semi-decidable.*

Planning algorithms

We present two algorithms for solving STRIPStream problems: the *incremental* planner takes advantage of certified conditional streams in the problem specification to generate the necessary objects for solving the problem; the *focused* planner adds the ability to focus the object-generation process based on the requirements of the plan being constructed. Both algorithms are sound and complete: if a solution exists they will find it in finite time.

Both planners operate iteratively, alternating between adding elements and atoms to a current set of objects and initial atoms and constructing and solving STRIPS planning problem instances. A STRIPS problem $(\mathcal{P}, \mathcal{A}, O, s_{init}, s_*)$ is specified by a set of predicates, a set of operator schemas, a set of constant symbols, an initial set of atoms, and a set of goal literals. Let S-PLAN$(\mathcal{P}, \mathcal{A}, O, s_{init}, s_*)$ be any sound and complete planner for the STRIPS subset of PDDL. We implement S-PLAN using FastDownward (Helmert 2006).

Incremental planner

The incremental planner maintains a queue of stream instances Q and incrementally constructs set $\mathcal{O}$ of objects and set $\mathcal{S}$ of fluents and static atoms that are true in the initial state. The *done* set D contains all streams that have been constructed and exhausted. In each iteration of the main loop, a STRIPS planning instance is constructed from the current sets $\mathcal{O}$ and $\mathcal{S}$, with the same predicates, operator and axiom schemas, and goal. If a plan is obtained, it is returned. If not, then $K \geq 1$ attempts to add new objects are made where K is a meta-parameter. In each one, a stream $\sigma(\bar{Y} \mid \bar{x})$ is popped from Q and a new tuple of objects $\bar{y}$ is extracted from it. If the stream is exhausted, it is stored in D. Otherwise, the objects in $\bar{y}$ are added to $\mathcal{O}$, the output fluents from σ applied to $(\bar{x}, \bar{y})$ are added to $\mathcal{S}$, and a new set of streams Σ_n is constructed. For all stream schemas σ and possible tuples of the appropriate size $\bar{x}'$, if the input conditions $\sigma'.\textbf{inp}(\bar{x}')$ are in $\mathcal{S}$, then the instantiated stream $\sigma'(\bar{Y}' \mid \bar{x}')$ is added to Q if it has not been added previously. We also return the stream $\sigma(\bar{Y} \mid \bar{x})$ to Q so we may revisit it in the future. The pseudo-code is shown below.

```
INCREMENTAL(((P_s, P_f, C_0, A, X, Σ), O_0, s_0, s_*), S-PLAN, K):
    O = C_0 ∪ O_0; S = s_0; D = ∅
    Q = QUEUE({σ(Ȳ | x̄) | σ(Ȳ | X̄) ∈ Σ,
               x̄ ∈ O^|X̄|, σ.inp(x̄) ⊆ S})
    while True:
        π_* = S-PLAN(P_s ∪ P_f, A ∪ X, O, S, s_*)
        if π_* ≠ None:
            return π_*
        if EMPTY(Q):
            return None
        for k ∈ {1, ..., MIN(K, LEN(Q))}:
            σ(Ȳ | x̄) = POP(Q)
            ȳ = NEXT(σ.f(x̄))
            if ȳ = None:
                D = D ∪ {σ(Ȳ | x̄)}
                continue
            O = O ∪ ȳ; S = S ∪ σ.out((x̄, ȳ))
            for σ'(Ȳ' | X̄') ∈ Σ:
                for x̄' ∈ O^|X̄'|:
                    if σ'.inp(x̄') ⊆ S, σ'(Ȳ' | x̄') ∉ (Q ∪ D):
                        PUSH(Q, σ'(Ȳ' | x̄'))
            PUSH(Q, σ(Ȳ | x̄))
```

In practice, many S-PLAN calls report infeasibility immediately because they have infinite admissible heuristic values. We prove the incremental algorithm is complete in the appendix.

Theorem 4. *The incremental algorithm is complete.*

Focused planner

The focused planner is particularly aimed at domains for which it is expensive to draw an object from a stream; this occurs when the stream elements are certified to satisfy geometric properties such as being collision-free or having appropriate inverse kinematics relationships, for example. To focus the generation of objects on the most relevant parts of the space, we allow the planner to use "dummy" abstract objects as long as it plans to generate concrete values for them. These concrete values will be generated in the next iteration and will, hopefully, contribute to finding a solution with all ground objects.

As before, we transform the STRIPStream problem into a sequence of PDDL problems, but this time we augment the planning domain with abstract objects, two new fluents, and a new set of operator schemas. Let $\{\gamma_1, ..., \gamma_\theta\}$ be a set of *abstract objects* which are not assumed to satisfy any static predicates in the initial state. We introduce the fluent predicate $Concrete$, which is initially false for any object γ_i but true for all actual ground objects; so for all $o \in \mathcal{O}$, we add $Concrete(o)$ to s_{init}. The planner can "cause" an abstract object γ_i to satisfy $Concrete(\gamma_i)$ by generating it using a special *stream operator*, as described below. We define procedure TFORM-OPS that transforms each operator scheme $a(x_1, ..., x_n) \in \mathcal{A}$ by adding preconditions $Concrete(x_i)$ for $i = 1, ..., n$ to ensure that the parameters for a are grounded before its application during the search.

To manage the balance in which streams are called, for each stream schema σ, we introduce a new predicate $Blocked_\sigma$; when applied to arguments $(X_1, \ldots, X_n)$, it will temporarily prevent the use of stream $\sigma(Y_1, ...Y_m \mid X_1, ..., X_n)$. Additionally, we add any new objects and static atoms first to sets $\mathcal{O}_t$ and $\mathcal{S}_t$ temporarily before adding them to $\mathcal{O}$ and $\mathcal{S}$ to ensure any necessary existing streams are called. Alternatively, we can immediately add directly to $\mathcal{O}$ and $\mathcal{S}$ a finite number of times before first adding to $\mathcal{O}_t$ and $\mathcal{S}_t$ and still preserve completeness. Let the procedure TFORM-STREAMS convert each stream schema into an operator schema σ of the following form.

STREAMOPERATOR$_\sigma(X_1, ..., X_m, Y_1, ..., Y_n)$:
pre = σ.**inp** $\cup$ $\{Concrete(X_i) \mid i = 1, ..., m\} \cup \{\neg Blocked_\sigma(X_1, ..., X_m)\}$
eff = σ.**out** $\cup \{Concrete(Y_i) \mid i = 1, ..., n\}$

It allows S-PLAN to explicitly plan to generate a tuple of concrete objects from stream $\sigma(Y_1, ...Y_m \mid x_1, ..., x_n)$ as long as the x_i have been made concrete and the stream instance is not blocked.

The procedure FOCUSED, shown below, implements the focused approach to planning. It takes the same inputs as the incremental version, but with the maximum number of abstract objects $\theta \geq 1$ specified as a meta-parameter, rather than K. It also maintains a set $\mathcal{O}$ of concrete objects and a set $\mathcal{S}$ of fluent and static atoms true in the initial state. In each iteration of the main loop, a STRIPS planning instance is constructed: the initial state is augmented with the set of static atoms indicating which streams are blocked and fluents asserting that the objects in $\mathcal{O}$ are concrete; the set of operator schemas is transformed as described above and augmented with the stream operator schemas, and the set of objects is augmented with the abstract objects. If a plan is obtained and it contains only operator instances, then it will have only concrete objects, and it can be returned directly. If the plan contains abstract objects, it also contains stream operators, and ADD-OBJECTS is called to generate an appropriate set of new objects. If no plan is obtained, and if no streams are currently blocked as well as no new objects or initial atoms have been produced since the last reset, then the problem is proved to be infeasible. Otherwise, the problem is reset by unblocking all streams and adding $\mathcal{O}_t$ and $\mathcal{S}_t$ to $\mathcal{O}$ and $\mathcal{S}$, in order to allow a new plan with abstract objects to be generated.

FOCUSED$(((\mathcal{P}_s, \mathcal{P}_f, \mathcal{C}_0, \mathcal{A}, \mathcal{X}, \Sigma), O_0, s_0, s_*), \text{S-PLAN}, \theta)$:
  $\mathcal{O} = \mathcal{C}_0 \cup O_0$; $\mathcal{S} = s_0$; $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \beta_p = \emptyset$
  $\bar{\mathcal{A}} =$ TFORM-OPS$(\mathcal{A})$; $\bar{\Sigma} =$ TFORM-STREAMS(Σ)
  while True:
    $\pi =$ S-PLAN$(\mathcal{P}_s \cup \mathcal{P}_f, \bar{\mathcal{A}} \cup \mathcal{X} \cup \bar{\Sigma}, \mathcal{O} \cup \{\gamma_1, ..., \gamma_\theta\}, \mathcal{S} \cup \beta_t \cup \beta_p \cup \{Concrete(o \in \mathcal{O})\}, s_*)$
    if $\pi \neq$ **None**:
      if $\forall a \in \pi$, SCHEMA$(a) \in \bar{\mathcal{A}}$:
        return π
      ADD-OBJECTS$(\pi, \mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p, \bar{\Sigma})$
    else
      if $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \emptyset$:
        return None // Infeasible
      $\mathcal{O} = \mathcal{O} \cup \mathcal{O}_t; \mathcal{S} = \mathcal{S} \cup \mathcal{S}_t$
      $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \emptyset$ // Enable all objects & streams

Given a plan π that contains abstract objects, we process it from beginning to end, to generate a collection of new objects with appropriate conditional relationships. Procedure ADD-OBJECTS initializes an empty binding environment and then loops through the instances a of stream operators in π. For each stream operator instance, we substitute concrete objects in for abstract objects, in the input parameters, dictated by the bindings bd, and then draw a new tuple of objects from that conditional stream. If there is no such tuple of objects, the stream is exhausted and it is permanently removed from future consideration by adding the fluent $Blocked_\sigma(\bar{o}_x)$ to the set β_p. Otherwise, the new objects are added to $\mathcal{O}_t$ and appropriate new static atoms to $\mathcal{S}_t$. This stream is temporarily blocked by adding fluent $Blocked_\sigma(\bar{o}_x)$ to the set β_t, and the bindings for abstract objects are recorded.

ADD-OBJECTS$(\pi, \mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p, \bar{\Sigma})$:
  $bd = \{\ \}$ // Empty dictionary
  for $\sigma(\bar{y} \mid \bar{x}) \in \{a \mid a \in \pi$ **and** SCHEMA$(a) \in \bar{\Sigma}\}$:
    $\bar{o}_x =$ APPLY-BINDINGS$(bd, \bar{x})$
    if $\bar{o}_x \neq$ **None**:
      $\bar{o}_y =$ NEXT$(\sigma.f(\bar{o}_x))$
      if $\bar{o}_y \neq$ **None**:
        $\mathcal{O}_t = \mathcal{O}_t \cup \bar{o}_y; \mathcal{S}_t = \mathcal{S}_t \cup \sigma$.**out**$((\bar{o}_x, \bar{o}_y))$
        $\beta_t = \beta_t \cup \{Blocked_\sigma(\bar{o}_x)\}$ // Temporary
        for $i \in \{1, ..., |\bar{y}|\}$:
          $bd[y_i] = o_{y,i}$
      else
        $\beta_p = \beta_p \cup \{Blocked_\sigma(\bar{o}_x)\}$ // Permanent
  return $\mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p$

The focused algorithm is similar to the lazy shortest path

algorithm for motion planning in that it determines which streams to call, or analogously which edges to evaluate, by repeatedly solving optimistic problems (Bohlin and Kavraki 2000; Dellin and Srinivasa 2016). Stream operators can be given meta-costs that reflect the time overhead to draw elements from the stream and the likelihood the stream produces the desired values. For example, stream operators that use already concrete outputs can be given large meta-costs because they will only certify a desired predicate in the typically unlikely event that their generator returns objects matching the desired outputs. A cost-sensitive planner will avoid returning plans that require drawing elements from expensive or unnecessary streams. We can combine the behaviors of incremental and focused algorithms to eagerly call inexpensive streams and lazily call expensive streams. This can be seen as automatically applying some stream operators before calling S-PLAN.

Theorem 5. *The focused algorithm is complete.*

Example discrete domain

Although the specification language is domain independent, our primary motivating examples for the application of STRIPStream are pick-and-place problems in infinite domains. We start by specifying an infinite discrete pick-and-place domain as shown in figure 2. We purposefully describe the domain in a way that will generalize well to continuous and high-dimensional versions of fundamentally the same problem. The objects in this domain include a finite set of blocks (that can be picked up and placed), an infinite set of poses (locations in the world) indexed by the positive integers, and an infinite set of robot configurations (settings of the robot's physical degrees of freedom) also indexed by the positive integers. In this appendix, we give a complete Python implementation of this domain in STRIPStream The static predicates in this domain include simple static types ($IsConf$, $IsPose$, $IsBlock$) and typical fluents ($HandEmpty$, $Holding$, $AtPose$, $AtConfig$). In addition, atoms of the form $IsKin(P, Q)$ describe a static relationship between an object pose P and a robot configuration Q: in this simple domain, the atom is true if and only if $P = Q$. Finally, fluents of the form $Safe(b', B, P)$ are true in the circumstance that: if object B were placed at pose P, it would not collide with object b' at its current pose. Because the set of blocks B is known statically in advance, we explicitly include all the $Safe$ conditions. These predicate definitions enable the following operator schemas definitions:

MOVE(Q_1, Q_2):
stat = $\{IsConf(Q_1), IsConf(Q_2)\}$
pre = $\{AtConf(Q_1)\}$
eff = $\{AtConf(Q_2), \neg AtConf(Q_1)\}$

PICK(B, P, Q):
stat = $\{IsBlock(B), IsPose(P), IsConf(Q), IsKin(P, Q)\}$
pre = $\{AtPose(B, P), HandEmpty(), AtConfig(Q)\}$
eff = $\{Holding(B), \neg AtPose(B, P), \neg HandEmpty()\}$

PLACE(B, P, Q):
stat = $\{IsBlock(B), IsPose(P), IsConf(Q), IsKin(P, Q)\}$
pre = $\{Holding(B), AtConfig(Q)\} \cup \{Safe(b' \in \mathcal{B}, B, P)\}$
eff = $\{AtPose(B, P), HandEmpty(), \neg Holding(B)\}$

We use the following axioms to evaluate the $Safe$ predicate. We need two slightly different definitions to handle the cases where the block B_1 is placed at a pose, and where it is in the robot's hand. The $Safe$ axioms mention each block independently which allows us to compactly perform collision checking. Without using axioms, PLACE would require a parameter for the pose of each block in $\mathcal{B}$, resulting in an prohibitively large grounded problem.

SAFEAXIOM(B_1, P_1, B_2, P_2):
stat = $\{IsBlock(B_1), IsPose(P_1), IsBlock(B_2),$
$IsPose(P_2), IsCollisionFree(B_1, P_1, B_2, P_2)\}$
pre = $\{AtPose(B_1, P_1)\}$
eff =$\{Safe(B_1, B_2, P_2)\}$

SAFEAXIOMH(B_1, B_2, P_2):
stat = $\{IsBlock(B_1), IsBlock(B_2), IsPose(P_2)\}$
pre = $\{Holding(B_1)\}$
eff =$\{Safe(B_1, B_2, P_2)\}$

Discrete stream specification Next, we provide stream definitions. The simplest stream is an unconditional generator of poses, which are represented as objects POSE(i) and satisfy the static predicate $IsPose$.

POSE-U$(P \mid ())$:
gen = **lambda**() : $\langle$(POSE(i)) **for** $i = 0, 1, 2...\rangle$
inp = $\emptyset$
out = $\{IsPose(P)\}$

The conditional stream CFREE-T is a test, calling the underlying function COLLIDE(B_1, P_1, B_2, P_2); the stream is empty if block B_1 at pose P_1 collides with block B_2 at pose P_2, and contains the single element ( ) if it does not collide. It is used to certify that the tuple (B_1, P_1, B_2, P_2) statically satisfies the $IsCollisionFree$ predicate.

CFREE-T$(() \mid B_1, P_1, B_2, P_2)$:
gen = **lambda**(B_1, P_1, B_2, P_2) :
$\langle()$ **if not** COLLIDE$(B_1, P_1, B_2, P_2)\rangle$
inp = $\{IsBlock(B_1), IsPose(P_1), IsBlock(B_2), IsPose(P_2)\}$
out = $\{IsCollisionFree(B_1, P_1, B_2, P_2)\}$

When we have a static relation on more than one variable, such as $IsKin$, we have to make modeling choices when defining streams that certify it.

We will consider three formulations of streams that certify $IsKin$ and compare them in terms of their effectiveness in a simple countable pick-and-place problem requiring the robot gripper to pick block A at a distant initial pose $p_0 >> 1$, shown in figure 2.

KIN-U specifies an unconditional stream on block poses and robot configurations; it has no difficulty certifying the $IsKin$ relation between the two output variables, but it has no good way of producing configurations that are appropriate for poses that are mentioned in the initial state or goal.

KIN-U$(P, Q \mid ())$:

gen = lambda$() : \langle(\text{POSE}(i), \text{CONF}(i))$ **for** i $= 1, 2, ...\rangle$
inp $= \emptyset$
out $= \{IsPose(P), IsConf(Q), IsKin(P, Q)\}$

KIN-T specifies a test stream that can be used, together with the POSE-U stream and an analogous stream for generic configurations to produce certified kinematic pairs P, Q. This is an encoding of a "generate-and-test" strategy, which may be highly inefficient, relying on luck that the pose generator and the configuration generator will independently produce values that have the appropriate relationship.

KIN-T$((\,) \mid P, Q)$:

gen = lambda$(P, Q) : \langle()$ **if** $Q = \text{INVERSE-KIN}(P)\rangle$
inp $= \{IsPose(P), IsConf(Q)\}$
out $= \{IsKin(P, Q)\}$

Finally, KIN-C specifies a conditional stream, which takes a pose P as input and generates a stream of configurations (in this very simple case, containing a single element) certified to satisfy the $IsKin$ relation. It relies on an underlying function INVERSE-KIN(p) to produce an appropriate robot configuration given a block pose.

KIN-C$(Q \mid P)$:

gen = lambda$(P) : \langle(\text{INVERSE-KIN}(P))\rangle$
inp $= \{IsPose(P)\}$
out $= \{IsConf(Q), IsIK(P, Q)\}$

In our example domain, both KIN-U and KIN-T require the enumeration of poses and configurations (p_i, q_i) from $i = 0, 1, ..., p_0$ before certifying $IsIK(p_*, q_*)$, allowing STRIPS to make a plan include the operator PICK(A, p_*, q_*). Moreover, KIN-T will test all pairs of configurations and poses. In contrast, KIN-C can produce q directly from p_0 without enumerating any other poses or configurations. The conditional formulation is advantageous because it produces a paired inverse kinematics configuration quickly and without substantially expanding the size of the problem.

Table 1 validates this intuition though an experiment comparing these stream specifications. The initial pose of the object p_0 is chosen from $1, 100, 1000$. All trials have a timeout of 120 seconds and use the incremental algorithm with $K = 1$ implemented in Python. As predicted, the KIN-U and KIN-T streams require many more calls than KIN-C as p_0 increases and lead to substantially longer runtimes for a very simple problem.

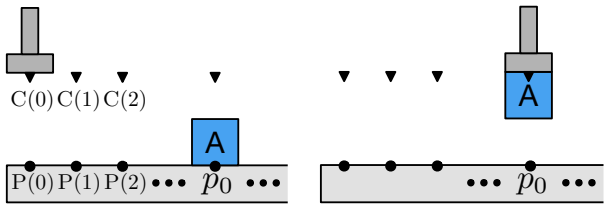


Figure 2: The initial state and goal state in an infinite, discrete pick-and-place problem requiring picking block A.

p_0	KIN-U			KIN-T			KIN-C		
	t	i	c	t	i	c	t	i	c
1	.1	3	2	.2	6	9	.1	3	2
100	29	102	101	71	303	10360	.1	3	2
1000	-	180	179	-	381	16383	.1	3	2

Table 1: The runtime (t), number of search iterations (i), and number of generator calls (c) for the countable pick-and-place KIN stream representation experiment.

Continuous domains

The STRIPStream approach can be applied directly in continuous domains such as the problem in figure 3. In this case, the streams will have to generate samples from sets of continuous dimensions, and the way that samples are generated may have a significant impact on the efficiency and completeness of the approach with respect to the domain problem. (Note that the STRIPStream planing algorithms are complete with respect to the streams of enumerated values they are given, but if these value streams are not, in some sense, complete with respect to the underlying problem domain, then the resulting combined system may not be complete with respect to the original problem.) Samplers that produce a *dense* sequence (LaValle 2006) are good candidates for stream generation.

Continuous stream specification

With some minor modifications, we can extend our discrete pick-and-place domain to a bounded interval $[0, L]$ of the real line. Poses and configurations are now continuous objects $p, q \in [0, L]$ from an uncountably infinite domain. The stream POSE-U now has a generator that samples $[0, L]$ uniformly at random.

While in the discrete case the choice of streams just affected the size of the problem, in the continuous case, the choice of streams can affect the feasibility of the problem. In the continuous simple pick-and-place domain, suppose that the blocks have width 1 and the gripper has width $\delta \geq 1$. A kinematics pair (p, q) is valid if and only if the gripper is entirely over the block, i.e., $p + 1/2 \leq q + \delta/2$ and $p - 1/2 \geq q - \delta/2$. Consider the case where KIN-U and KIN-C are implemented using random samplers. KIN-U will almost certainly generate a sequence of infeasible STRIPS problems, because the probability that the point p_0 is produced from its generator is zero. For $\delta > 1$, the configuration stream has nonzero probability of generating a q that would constitute a valid kinematics pair with p as certified by KIN-T. But this probability can be made arbitrary small as $\delta \to 1$. Only the KIN-C strategy is robust to the choice of δ. Table 2 shows the results of an experiment analogous to the one in table 1, but which varies $\delta \in \{1.5, 1.01\}$ instead of varying p_0. KIN-U was unable to solve either problem and KIN-T could not find a solution in under two minutes for $\delta = 1.01$. But once again, the conditional formulation using KIN-C performs equivalently for different values of δ.

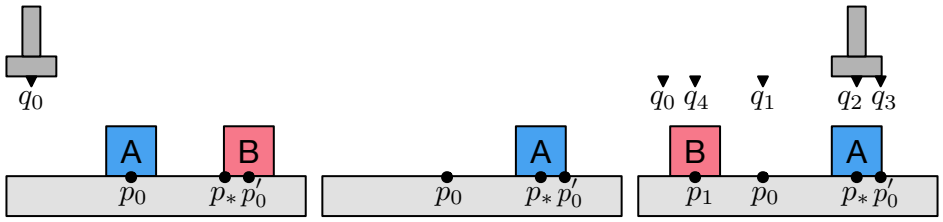


Figure 3: Initial state for countable pick-and-place problem requiring picking and placing block A, with a single obstacle.

δ	KIN-U			KIN-T			KIN-C		
	t	i	c	t	i	c	t	i	c
1.5	-	191	190	3.1	75	745	.1	2	1
1.01	-	181	180	-	297	18768	.1	2	1

Table 2: The runtime (t), number of search iterations (i), and number of generator calls (c) for the continuous pick-and-place KIN stream representation experiment.

Focused algorithm example

The previous examples investigated the effect of different representational choices on the tractability and even feasibility of the resulting STRIPStream problem.

The example in figure 3 illustrates the behavior of the focused algorithm on continuous a pick-and-place problem with the goal condition that block A is at pose p_*. Because block A, when at p_*, collides with block B at its initial pose p'_0, solving this problem requires moving block B out of the way to place block A. Suppose we use KIN-C to model the problem. We will omit MOVE operators for the sake of clarity, and use capital letters to denote abstract objects. On the first iteration, the focused algorithm will produce the following plan (possibly ordered slightly differently):

$$\pi_1 = \big(\text{KIN-C}(Q_1 \mid p_0), \text{PICK}(A, p_0, Q_1), \text{KIN-C}(Q_2 \mid p_*), \text{CFREE-T}(() \mid B, p'_0, A, p_*), \text{PLACE}(A, p_*, Q_2)\big)$$

The generation of values proceeds as follows. KIN-C$(Q_1 \mid p_0)$ will produce $Q_1 \leftarrow q_1$. KIN-C$(Q_2 \mid p_*)$ will produce $Q_2 \leftarrow q_2$. However, CFREE-T$(() \mid B, p'_0, A, p_0)$ will produce the empty stream because p'_0 collides with p_*. Thus, the plan π_1 definitively cannot be completed. The algorithm adds q_1 and q_2 to the current PDDL problem and records the failure of CFREE-T$(() \mid B, p'_0, A, p_0)$. On the next iteration, the focused algorithm will produce the following plan.

$$\pi_2 = \big(\text{KIN-C}(Q_1 \mid p'_0), \text{PICK}(B, p'_0, Q_1), \text{POSE-U}(P_1 \mid ()), \text{KIN-C}(Q_2 \mid P_1), \text{CFREE-T}(() \mid A, p_0, B, P_1), \text{PLACE}(B, P_1, Q_2), \text{PICK}(A, p_0, q_1), \text{CFREE-T}(() \mid B, P_1, A, p_*), \text{PLACE}(A, p_*, q_2)\big)$$

The generation of values proceeds as follows. KIN-C$(Q_1 \mid p'_0)$ will produce $Q_1 \leftarrow q_3$. POSE-U$(P_1 \mid ())$ will produce $P_1 \leftarrow p_1$. KIN-C$(Q_2 \mid p_1)$ will produce $Q_2 \leftarrow q_4$. Let's assume that $P_1 \leftarrow p_1$ is randomly sampled and turns out to not be in collision with p_*. If p_1 turned out to be in collision with p_*, the next iteration would first fail once, then repeat this process on the next iteration to generate a new P_1. So, CFREE-T$(() \mid A, p_1, B, p_0)$ will produce the stream $\langle() \rangle$ indicating that p_1 and p_0 are not in collision. Thus, all of the properties have been successfully satisfied, so the following plan is a solution. It is critical to note that, for example, had there been several other pose constants appearing in the initial state, focused would never have found inverse kinematic solutions for them: because the planner guides the sampling, only stream elements that play a direct role in a plausible plan are generated.

$$\pi_* = \big(\text{PICK}(B, p'_0, q_3), \text{PLACE}(B, p_1, q_4), \text{PICK}(A, p_0, q_1), \text{PLACE}(A, p_*, q_2)\big)$$

Realistic robot domain

Finally, we extend our continuous pick-and-place to the high-dimensional setting of a robot operating in household-like environments. Poses of physical blocks are 6-dimensional and robot configurations are 11-dimensional. We introduce two new object types: grasps and trajectories. Each block has a set of 6D relative grasp transforms at which it can be grasped by the robot. Trajectories are finite sequences of configuration waypoints which must be included in collision checking. The extended PICK operator, CFREE-T test and KIN-C stream templates are:

PICK(B, P, G, Q, T):
- **stat** = $\{IsBlock(B), ..., IsTraj(T), IsKin(P, G, Q, T)\}$
- **pre** = $\{AtPose(B, P), HandEmpty(), AtConfig(Q)\} \cup \{Safe(b', B, G, T) \mid b' \in \mathcal{B}\}$
- **eff** = $\{Holding(B, G), \neg AtPose(B, P), \neg HandEmpty()\}$

CFREE-T$(() \mid B_1, P_1, B_2, G, T)$:
- **gen** = **lambda**(B_1, P_1, B_2, G, T) : $\langle ()$ **if not** COLLIDE$(B_1, P_1, B_2, G, T)\rangle$
- **inp** = $\{IsBlock(B_1), ..., IsTraj(T)\}$
- **out** = $\{IsCollisionFree(B_1, P_1, B_2, G, T)\}$

KIN-C$(Q, T \mid P, G)$:
- **gen** = **lambda**(P) : $\langle (Q, T) \mid Q \sim$ INVERSE-KIN(PG^{-1}), $T \sim$ MOTIONS$(q_{rest}, Q)\rangle$
- **inp** = $\{IsPose(P), IsGrasp(G)\}$
- **out** = $\{IsKin(P, G, Q, T), IsConf(Q), IsTraj(T)\}$

Π	incremental, $K = 1$				incremental, $K = 100$				focused			
	%	t	i	c	%	t	i	c	%	t	i	c
1	88	2	23	268	68	5	2	751	84	11	6	129
2-0	100	23	85	1757	100	9	3	2270	100	2	3	180
2-8	0	-	-	-	100	55	5	17217	100	7	3	352
2-16	0	-	-	-	100	112	6	36580	100	19	3	506

Table 3: The success percentage (%), runtime (t), search iterations (i), and number of stream calls (c) for the high-dimensional task and motion planning experiments.

PICK adds grasp G and trajectory T as parameters and includes $Safe(b', B, G, T)$ preconditions to verify that T while holding B at grasp G is safe with respect to each other block b'. $Safe(b', B, G, T)$ is updated using SAFEAXIOM which has a $IsCollisionFree(B_1, P_1, B_2, G, T)$ static precondition. Here, a collision check for block B_1 at pose P_1 is performed for each configuration in T. Instead of simple blocks, physical objects in this domain are general unions of convex polygons. Although checking collisions here is more complication than in 1D, it can be treated in the same way, as an external function.

The KIN streams must first produce a grasp configuration Q that reaches manipulator transform PQ^{-1} using INVERSE-KIN. Additionally, they include a motion planner MOTIONS to generate legal trajectory values T from a constant rest configuration q_{rest} to the grasping configuration Q that do not collide with the fixed environment. In this domain, the procedures for collision checking and finding kinematic solutions are significantly more involved and computationally expensive than in the previous domains, but their underlying function is the same.

Experiments

We applied the incremental and focused algorithms on four challenging pick-and-place problems to demonstrate that a general-purpose representation and algorithms can be used to achieve good performance in difficult problems. For both algorithms, test streams are always evaluated as soon as they are instantiated. We experimented on two domains shown in figure 4, which are similar to problems introduced by (Garrett *et al.* 2015). The first domain, in which problem 1 is defined, has goal conditions that the green object be in the right bin and the blue object remain at its initial pose. This requires the robot to not only move and replace the blue block but also to place the green object in order to find a new grasp to insert it into the bin. The second domain, in which problems 2-0, 2-8, and 2-16 are defined, requires moving an object out of the way and placing the green object in the green region. For problem 2-n where $n \in \{0, 8, 16\}$ there are n other blocks on a separate table that serve as distractors. The streams were implemented using the OpenRAVE robotics framework (Diankov and Kuffner 2008). A Python implementation of STRIPStream can be found here: `https://github.com/caelan/stripstream`.

The results compare the incremental algorithm where $K = 1$ and $K = 100$ with the focused algorithm. Table 3 shows the results of 25 trials, each with a timeout of 120 seconds. The incremental algorithms result in significantly more stream calls than the focused algorithm. These calls can significantly increase the total runtime because each inverse kinematic and collision primitive itself is expensive. Additionally, the incremental algorithms are significantly affected by the increased number of distractors, making them unsuitable for complex real-world environments. The focused algorithm, however, is able to selectively choose which streams to call resulting in significantly better performance in these environments.

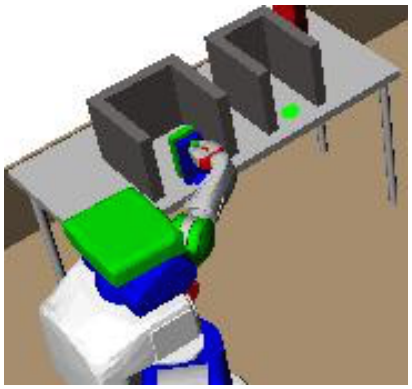


Figure 4: Problem 1.

Conclusion

The STRIPStream problem specification formalism can be used to describe a large class of planning problems in infinite domains and provides a clear and clean interface to problem-specific sampling methods in continuous domains. The incremental and, in particular, focused planning algorithms take advantage of the specification to provide efficient solutions to difficult problems.

References

Robert Bohlin and Lydia E Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 521–528. IEEE, 2000.

Piero Bonatti, Francesco Calimeri, Nicola Leone, and Francesco Ricca. Answer set programming. In *A 25-year perspective on logic programming*, pages 159–182. Springer-Verlag, 2010.

Daniel Bryce, Sicun Gao, David J Musliner, and Robert P Goldman. SMT-based nonlinear PDDL+ planning. In *AAAI*, 2015.

Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. An asp system with functions, lists, and sets. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 483–489. Springer, 2009.

Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full pddl+ language into smt. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 79–87. AAAI Press, 2016.

Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research (JAIR)*, 46(1):343–412, 2013.

Neil T. Dantam, Z. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems (RSS)*, 2016.

Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. Upmurphi: A tool for universal planning on pddl+ problems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, 2008.

Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 114–121. AAAI Press, 2009.

Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

Maria Fox and Derek Long. Pddl2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:2003, 2003.

Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.(JAIR)*, 27:235–297, 2006.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Backward-forward search for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. FFRob: Leveraging symbolic planning for efficient task and motion planning. *arXiv preprint arXiv:1608.01335*, 2016.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246, 2006.

Jörg Hoffmann et al. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res.(JAIR)*, 20:291–341, 2003.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl: The planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

Wiktor Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for pddl+ domains. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

Appendix

Theorem 6. *$\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ are recursively enumerable (RE).*

Proof. Consider an enumeration procedure for $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$:

- The first sequences of elements in $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ are $\mathcal{C}_0 \cup O_0$ and s_0 respectively.
- Initialize a set of stream instances $\Sigma_\Pi = \emptyset$.
- Repeat:
  - For each stream schema $\sigma \in \Sigma$, add all instantiations $\sigma(\bar{Y} \mid \bar{x})$ where $x \subseteq \mathcal{O}_\Pi$ such that $\sigma.\textbf{inp}(\bar{x})$ is contained within $\mathcal{S}_\Pi$, to Σ_Π. There are finitely many new elements of Σ_Π.
  - For each stream instance $\sigma(\bar{Y} \mid \bar{x}) \in \Sigma_\Pi$, add $\bar{y} = \text{NEXT}(\sigma.f(\bar{x}))$ to $\mathcal{O}_\Pi$ and add $\sigma^*.\textbf{out}((\bar{x}, \bar{y}))$ to $\mathcal{S}_\Pi$. There are finitely many new elements of $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$.

This procedure will enumerate all possible objects and all possible initial atoms generated within the problem Π. □

Theorem 7. *The existence of a solution for a* STRIPS*tream problem Π is undecidable.*

Proof. We use a reduction from the halting problem. Given a Turning machine TM, we construct a STRIPStream problem Π_{TM} with a single operator HALT(X) with **stat** $= \{IsReachable(X)\}$, **pre** $= \emptyset$, and **eff** $= \{Reached(X)\}$ where $IsReachable$ and $Reached$ are a static and fluent predicate defined on TM's states. There is a single unconditional stream REACHABLE-U$(X \mid ())$ which enumerates the states of TM by simulating one step of TM upon each call. Let $s_0 = \emptyset$ and $s_* = \{Reached(a)\}$ where a is the accept state for TM. Π_{TM} has a solution if and only if TM halts. Thus, STRIPStream is undecidable. □

Theorem 8. *The existence of a solution for a* STRIPS*tream problem Π is semi-decidable.*

Proof. From the recursive enumeration of $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ we produce a recursive enumeration of finite planning problems. Planning problem i is grounded using all objects and static atoms enumerated up through element i. Plan existence in a finite universe is decidable. Thus, for feasible problems, applying a finite decision procedure to the sequence of finite planning problems will eventually reach a planning problem for which a plan exists and produce it. □

Theorem 9. *The incremental algorithm is complete.*

Proof. The incremental algorithm constructs $\mathcal{O}$ and $\mathcal{S}$ in the same way as theorem 6 for $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ except that it calls NEXT in batches of K. Thus, any finite subsets of $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ will be included in $\mathcal{O}$ and $\mathcal{S}$ after a finite number of iterations. Let π_* be a solution to a feasible STRIPStream problem Π. Consider the first iteration where $\mathcal{O}$ and $\mathcal{S}$ contain the set of objects used along π_* and static atoms supporting π_*. On that iteration, S-PLAN will return some solution (if not π_*) in finite time because it is sound and complete. □

Theorem 10. *The focused algorithm is complete.*

Proof. Define an episode as the focused algorithm iterations between the last reset ($\mathcal{O}_t = \mathcal{S}_t = \beta_t = \emptyset$) and the next reset. Consider a minimum length solution π_* to a feasible STRIPStream problem Π. Let $\mathcal{O}_* \subseteq \mathcal{O}_\Pi$ be the set of objects used along π_* and $\mathcal{S}_* \subseteq \mathcal{S}_\Pi$ be the set of static atoms supporting π_*.

For each episode, consider the following argument. By theorem 6, there exists a sequence of stream instance calls which produces $\mathcal{O}_*$ and $\mathcal{S}_*$ from the current $\mathcal{O}$ and $\mathcal{S}$. Let Σ_* be the minimum length sequence that satisfies this property. Σ_* may include the same stream instance several times if multiple calls are needed to produce the necessary values. On each iteration, FOCUSED creates a finite STRIPS problem $\tilde{\Pi}$ by augmenting Π with the abstract objects $\{\gamma_1, ..., \gamma_\theta\}$ and the stream operators $\tilde{\Sigma}$. Because $\mathcal{O}_t$ and $\mathcal{S}_t$ are withheld, $\mathcal{O}$ and $\mathcal{S}$ are fixed for all iterations within the episode. Thus, a finite number of simple plans are solutions for $\tilde{\Pi}$. One of these plans, $\tilde{\pi}_*$, is Σ_* concatenated with π_* where additionally any object $o \in (\mathcal{O}_* \setminus \mathcal{O})$ is replaced with some abstract object γ. Assume all redundant stream operators are removed from $\tilde{\pi}_*$. The same γ can stand in for several o on $\tilde{\pi}_*$ at once. Thus, FOCUSED will be complete for any $\theta \geq 1$.

We will show that at least one stream instance in Σ_* will called performed during each episode. On each iteration, S-PLAN will identify a plan π. If π does not involve any abstract objects and is fully supported, it is a solution. Otherwise, ADD-OBJECTS will call each stream $\sigma(\bar{Y} \mid \bar{o}_x)$ associated with π. It adds $Blocked_\sigma(\bar{o}_x)$ to β_t, preventing π and all other plans using $\sigma(\bar{Y} \mid \bar{o}_x)$ from being re-identified within this episode. If π overlaps with $\tilde{\pi}_*$ and $\sigma(\bar{Y} \mid \bar{o}_x) \in \Sigma_*$, then the episode has succeeded. Otherwise, this process repeats on the next iteration. Eventually a stream instance in Σ_* will be called, or $\tilde{\pi}_*$ itself will be the only remaining unblocked plan for $\tilde{\Pi}$. In which case, S-PLAN will return $\tilde{\pi}_*$, and ADD-OBJECTS will call a stream instance in Σ_*.

$|\Sigma_*|$ strictly decreases after each episode. Inductively applying this, after a finite number of episodes, $\mathcal{O}_* \subseteq \mathcal{O}$ and $\mathcal{S}_* \subseteq \mathcal{S}$. During the next episode, S-PLAN will be guaranteed to return some solution (if not π_*).

□

Python example discrete domain

Figure 5 gives a complete encoding of the example discrete domain and a problem instance within it using our Python implementation of STRIPStream. In the specified problem instance, the initial state consists of three blocks placed in a row. The goal is to shift each of the blocks over one pose. The Python syntax of STRIPStream intentionally resembles the Planning Domain Definition Language (PDDL) (McDermott *et al.* 1998). We use several common features of PDDL that extend STRIPS. The resulting encoding is equivalent to previously described STRIPS formulation but is more compact. We use object types BLOCK, POSE, CONF instead of static predicates $IsBlock$, $IsPose$, and $IsConf$. Additionally, we use several Action Description Language (ADL) logical operations including OR, EQUAL, FORALL, and EXISTS. The universal quantifier (FORALL) is over BLOCK, a finite type, and thus is a finite conjunction.

```
from stripstream import Type, Param, Pred, Not, Or, And, Equal, Exists, \
  ForAll, Action, Axiom, GeneratorStream, TestStream, STRIPStreamProblem

blocks = ['block%i'%i for i in range(3)]
num_poses = pow(10, 10) # a very large number of poses
initial_config = 0 # initial robot configuration is 0
initial_poses = {block: i for i, block in enumerate(blocks)} # initial pose for block i is i
goal_poses = {block: i+1 for i, block in enumerate(blocks)} # goal pose for block i is i+1

BLOCK, POSE, CONF = Type(), Type(), Type() # Object types
B1, B2 = Param(BLOCK), Param(BLOCK) # Free parameters
P1, P2 = Param(POSE), Param(POSE)
Q1, Q2 = Param(CONF), Param(CONF)

AtConf = Pred(CONF) # Fluent predicates
AtPose = Pred(BLOCK, POSE)
HandEmpty = Pred()
Holding = Pred(BLOCK)
Safe = Pred(BLOCK, BLOCK, POSE) # Derived predicates
IsKin = Pred(POSE, CONF) # Static predicates
IsCollisionFree = Pred(BLOCK, POSE, BLOCK, POSE)

actions = [
  Action(name='pick', parameters=[B1, P1, Q1],
    condition=And(AtPose(B1, P1), HandEmpty(), AtConf(Q1), IsKin(P1, Q1)),
    effect=And(Holding(B1), Not(AtPose(B1, P1)), Not(HandEmpty()))),
  Action(name='place', parameters=[B1, P1, Q1],
    condition=And(Holding(B1), AtConf(Q1), IsKin(P1, Q1),
      ForAll([B2], Or(Equal(B1, B2), Safe(B2, B1, P1)))),
    effect=And(AtPose(B1, P1), HandEmpty(), Not(Holding(B1)))),
  Action(name='move', parameters=[Q1, Q2],
    condition=AtConf(Q1),
    effect=And(AtConf(Q2), Not(AtConf(Q1))))]
axioms = [
  Axiom(effect=Safe(B2, B1, P1), # Infers B2 is at a safe pose wrt B1 at P1
        condition=Exists([P2], And(AtPose(B2, P2), IsCollisionFree(B1, P1, B2, P2))))]

cond_streams = [
  GeneratorStream(inputs=[], outputs=[P1], conditions=[], effects=[],
                  generator=lambda: xrange(num_poses)), # Enumerates all the poses
  GeneratorStream(inputs=[P1], outputs=[Q1], conditions=[], effects=[IsKin(P1, Q1)],
                  generator=lambda p: [p]), # Inverse kinematics
  TestStream(inputs=[B1, P1, B2, P2], conditions=[], effects=[IsCollisionFree(B1, P1, B2, P2)],
             test=lambda b1, p1, b2, p2: p1 != p2)] # Collision checking

constants = []
initial_atoms = [AtConf(initial_config), HandEmpty()] + \
                [AtPose(block, pose) for block, pose in initial_poses.iteritems()]
goal_formula = And(AtPose(block, pose) for block, pose in goal_poses.iteritems())
return STRIPStreamProblem(initial_atoms, goal_formula, actions+axioms, cond_streams, constants)
```

Figure 5: STRIPStream Python code for the example discrete domain

Joint Perception And Planning For Efficient Obstacle Avoidance Using Stereo Vision

Sourish Ghosh* **and Joydeep Biswas†**

Abstract

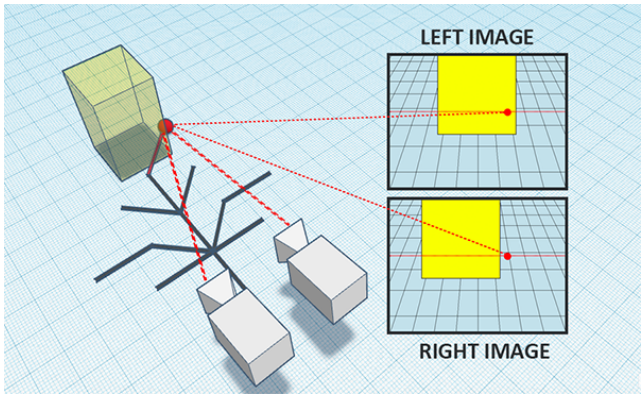
Stereo vision is commonly used for local obstacle avoidance of autonomous mobile robots: stereo images are first processed to yield a dense 3D reconstruction of the observed scene, which is then used for navigation planning. Such an approach, which we term Sequential Perception and Planning (SPP), results in significant unnecessary computations as the navigation planner only needs to explore a small part of the scene to compute the shortest obstacle-free path. In this paper, we introduce an approach to Joint Perception and Planning (JPP) using stereo vision, which performs disparity checks on demand, only as necessary while searching on a planning graph. Furthermore, obstacle checks for navigation planning do not require full 3D reconstruction: we present in this paper how obstacle queries can be decomposed into a sequence of *confident positive* stereo matches and *confident negative* stereo matches, which are significantly faster to compute than the *exact depth* of points. The resulting complete JPP formulation is significantly faster than SPP, while still maintaining correctness of planning. We also show how the JPP works with different planners, including search-based and sampling-based planners. We present extensive experimental results from real robot data and simulation experiments, demonstrating that the JPP requires less than 10% of the disparity computations required by SPP. The significant computational cost savings further allow JPP to run on higher resolution input images, thus further improving accuracy and robustness.

Introduction

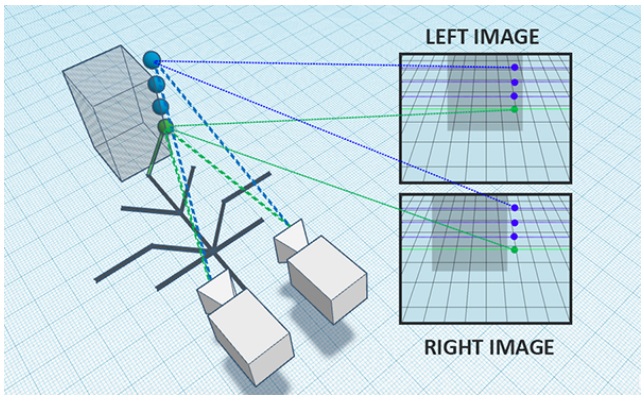
Existing approaches to local obstacle avoidance using stereo vision (Murray and Little 2000; Sabe *et al.* 2004) perform sequential perception and planning (SPP), where input stereo images are first processed to compute the disparity at each pixel. Such dense disparity is then used to infer depth for all image points, and hence obstacles in the world. However, the input images often include visual information that is irrelevant to the planning task at hand, thus wasting computational time at the perception step in SPP.

*Sourish Ghosh is with the Department of Mathematics, Indian Institute of Technology, Kharagpur, West Bengal 721302, India. Email: sourishg@iitkgp.ac.in

†Joydeep Biswas is with the College of Information and Computer Sciences, University of Massachusetts, Amherst, MA 01003, USA. Email: joydeepb@cs.umass.edu



(a) The red edge is not traversable because of an invalid confident positive match on the projections of the red point.



(b) The yellow obstacle is absent in this scenario. The green edge is hence traversable since the projection of the green point yields a confident positive match and the subsequent blue points above it yield confident negative matches indicating empty space points.

Figure 1: Given a graph (grey lines) for planning, JPP checks exploration edges for reachability by disparity match confidence between projected points on the left and right image. In (a) further verification of empty space points above the ground plane is not required since the ground plane is not reachable. Confidence matching is defined in Section *On-demand Stereo*.

In this paper, we introduce a novel approach to joint perception and planning (JPP) for obstacle avoidance using stereo vision that eliminates unnecessary disparity computations. JPP treats traversability queries by the

obstacle avoidance planner as on-demand disparity checks for perception. Thus, the only disparity computations performed by perception are those necessary for planning for the obstacle avoidance task. We further simplify the problem of identifying reachable configurations of the robot by verifying confident positive disparity matches for the ground plane around the configuration pose, and by verifying confident negative disparity matches for all points within the robot's safety radius and height of that pose. Verifying confident positive and negative checks are significantly computationally faster than evaluating the exact depth: while a confidence check requires only a single disparity comparison, evaluating the exact depth requires multiple disparity comparisons along the epipolar line. The confident checks are still exact: reachable and unreachable configurations of the robot are still correctly identified, albeit at a significantly lower computational cost. Figure 1 illustrates the decomposition of JPP into on-demand confidence disparity checks.

The JPP formulation is applicable to a wide range of planners for navigation, as long as the planner performs obstacle checks in the configuration space of the robot. We demonstrate JPP with an A* planner, and an RRT planner for local obstacle avoidance. Over extensive experimental evaluations, we show that JPP requires less than 10% of the disparity comparisons required by SPP. The contributions of this paper are thus three-fold: 1) We contribute a JPP formulation to integrate obstacle avoidance planning with on-demand stereo perception; 2) We present an exact simplification of the configuration-space obstacle check into a sequence of confidence checks over disparity; and 3) We empirically show that the JPP formulation results in substantial computational cost savings on a real robot, as well as an extensive set of simulated environments.

Related Work

Obstacle avoidance, as an essential ability of autonomous mobile robots, has been researched in great detail, and there exist a number of approaches, with various planning algorithms (*e.g.*, (Borenstein and Koren 1990; Simmons 1996; Tang *et al.* 2010)), sensor modalities (*e.g.*, (Arras *et al.* 2002; Biswas and Veloso 2012)) and efficient collision detection based on adaptive cell decomposition of the robot configuration space (Zhang *et al.* 2008). We thus focus on the related work most relevant to our own, covering existing approaches to obstacle avoidance using vision.

Vision based obstacle avoidance approaches can be broadly classified into monocular and stereo approaches. The two main approaches for obstacle detection using monocular vision are learning based methods (Michels *et al.* 2005) and using optical flow (Souhila and Karim 2007). Appearance-based obstacle detection (Ulrich and Nourbakhsh 2000) and visual sonar (Lenser and Veloso 2003) have also been shown to be effective for indoor ground robots.

Most stereo vision based approaches (Sabe *et al.* 2004; Simmons *et al.* 1996) perform either local epipolar matching or full scene reconstruction. Recognizing the significant computational cost of dense stereo reconstruction, a few recent methods (Kumano *et al.* 2000; Barry and Tedrake 2015) have tried to reduce computation time by doing sparse disparity checks. In Pushbroom Stereo (Barry and Tedrake 2015), obstacles are detected only at a constant disparity level, and by integrating this information with an onboard IMU and state estimator, position of obstacles at all other depths are recovered.

For a different problem of affordance based planning and a different sensor modality of LIDAR, joint perception and planning (Pryor *et al.* 2016) has been shown to successfully reduce combined planning and perception time.

While there have been partial informative approaches (Barry and Tedrake 2015) to obstacle avoidance using stereo vision, and joint perception and planning in other domains and with sensor modalities (Pryor *et al.* 2016), in this paper we contribute a joint perception and planning approach for the task of obstacle avoidance using stereo vision, where we avoid unnecessary full 3D reconstruction, and further relax the problem of checking reachable configurations into confidence disparity matching.

Epipolar Geometry and Stereo Vision

We use the left camera of the stereo pair as the sensor reference frame. Any visible 3D point X in space in the left camera reference frame has corresponding image points x and x' in the left and right camera images. The space point X, the image points x and x', and the camera centres are coplanar. The triangulation of these points is known as the epipolar constraint. For an image point x in the left image, there is a corresponding epipolar line l' in the right image and x' is constrained on l'. Similarly l is the epipolar line in the left image corresponding to the right image point x'. The epipolar geometry is algebraically encapsulated in a 3×3 matrix $\mathbf{F}$ known as the fundamental matrix. In homogeneous coordinates, $l' = \mathbf{F}x$ and $l = \mathbf{F}^\top x'$. The image points x and x' are constrained as $x'^\top \mathbf{F} x = 0$. Hence, when only one of the two image points is known, the corresponding point in the other image can be found by scanning along its epipolar line, resulting in a 1-D search.

Stereo camera calibration yields the left and right camera matrices, $\mathbf{K}$ and $\mathbf{K'}$ respectively. They constitute the intrinsic camera parameters. Let $\mathbf{R}$ and $\mathbf{t}$ denote the rotation and translation matrix from the left camera frame to the right camera frame, also known as the extrinsic parameters. The 3×4 projection matrices $\mathbf{P}$ and $\mathbf{P'}$ are defined as

$$\mathbf{P} = \mathbf{K}\,[\ \mathbf{I}\ |\ \mathbf{0}\ ] \qquad \mathbf{P'} = \mathbf{K'}\,[\ \mathbf{R}\ |\ \mathbf{t}\ ] \tag{1}$$

where $\mathbf{I}$ denotes the 3×3 identity matrix. Then $x = \mathbf{P}X$ and $x' = \mathbf{P'}X$ (in homogeneous coordinates).

The obstacle avoidance path planning is performed in the reference frame of the robot, where the origin coincides with the center of rotation of the robot projected on to the ground plane. Hence we find a transformation from the sensor reference frame to the robot reference frame. Let $\mathbf{R}_w$ and $\mathbf{t}_w$ denote the rotation and translation matrices of that transformation respectively. Therefore X in the robot reference frame is expressed as $X_w = \mathbf{R}_w X + \mathbf{t}_w$. We find $\mathbf{R}_w$ and $\mathbf{t}_w$ experimentally.

In sequential perception and planning, the depth of an image point is estimated by finding its correspondence along an epipolar line in the other image. In a rectified image coordinate system, the epipolar lines become horizontal scan lines. The horizontal shift or the difference in x-coordinate of two corresponding points in rectified coordinates is termed as disparity. Depth and disparity are related as

$$d = \frac{fB}{z} \tag{2}$$

where d denotes disparity, z denotes depth, f denotes the focal length of the camera, and B denotes the stereo baseline. Methods for generating disparity maps has been studied extensively (Scharstein and Szeliski 2002). The methods can be broadly classified into local and global methods. In local (window-based) methods the disparity computation only depends on the information of a finite window around the pixel under consideration. Global methods on the other hand make explicit smoothness assumptions and solve an optimization problem that minimizes a global cost function combining data and smoothness terms. Global methods are ideal for generating dense disparity maps whereas local methods are useful for generating sparse disparity maps.

Joint Perception and Planning

Unlike Sequential Perception and Planning (SPP), Joint Perception and Planning (JPP) performs a series of sparse stereo corresponding checks based on queries from the local path planner. The queries are to check if a robot pose is reachable. We explore two types of graph-based path planning algorithms: (1) search-based planning algorithms (Stentz 1994; Koenig and Likhachev 2002) (2) sampling-based algorithms (Karaman and Frazzoli 2011).

Let $\mathcal{X} \subset \mathbb{R}^2$ denote the robot configuration space, which is a set of robot poses $(x, y) \in \mathbb{R}^2$. We partition $\mathcal{X}$ into two sets $\mathcal{X}_{\text{free}}$ and $\mathcal{X}_{\text{obs}}$, where $\mathcal{X}_{\text{free}}$ denotes the set of poses reachable by the robot and $\mathcal{X}_{\text{obs}}$ denotes the set of poses not reachable by the robot. We define points belonging to the ground plane as those points $(x, y, z) \in \mathbb{R}^3$ such that $z = 0$. All other points are classified as obstacles. Let l and w denote the robot length and width respectively. Then we define the robot safety radius as $r = \max(\frac{l}{2}, \frac{w}{2})$. Any pose $(x, y) \in \mathcal{X}_{\text{free}}$ if all the points in the set $\mathbf{P}_r = \{(x', y', 0) : (x - x')^2 + (y - y')^2 < r^2\}$ are classified as ground plane and additionally the set $\mathbf{P}_h = \{(x', y', z) : 0 \leq z \leq h, (x', y', 0) \in \mathbf{P}_r\}$ contains *only* empty space points, where h is the robot height. Otherwise $(x, y) \in \mathcal{X}_{\text{obs}}$. Algorithm 1 outlines the procedure to check if a pose is reachable or not. The verification of space points belonging to the ground plane is done using confident positive matching while verification of points which are empty in space is done using confident negative matching.

Let x_{init} denote the initial pose of the robot and $\mathcal{X}_{\text{goal}} \subset \mathcal{X}_{\text{free}}$ be the set of final goal positions. Then the planning problem can be formally defined as the triplet $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. The local path planner explores a directed graph $G = (V, E)$ on the configuration space, where $V \subset \mathcal{X}_{\text{free}}$ denotes the set of vertices and E denotes the set of edges.

Search-based Planning. The configuration space is discretized into a square grid of size s. The neighbours of any node $v = (x, y)$ are $\mathbf{N}_v = \{(x + s, y + s), (x + s, y), (x + s, y - s), (x, y - s), (x, y + s)\}$. Any pose $p(x, y)$ is added to V if $p(x, y) \in \mathbf{N}_v$ for some $v \in V$ and REACHABLEPOSE(p) outlined in Algorithm 1 is *true*. We have presented results for JPP with A* search in this paper.

Sampling-based Planning. We have implemented JPP with RRT to represent sampling-based planning. A uniform sampler with a goal bias b samples a new pose $x \in \mathcal{X}$ in the configuration space. From V we find a pose y such that $\|x - y\|$ is minimum ($\|\cdot\|$ denotes Euclidean distance). Then a steering function STEER : $(x, y) \mapsto z$ returns a pose $z \in \mathcal{X}$ in the direction of x from y at distance of step size s. Let $L(y, z)$ denote the set of all poses lying on the edge joining y and z. z is added to the set of vertices V if $L(y, z) \subset \mathcal{X}_{\text{free}}$. Formally, (y, z) is a valid edge if for all $k \in L(y, z)$, REACHABLEPOSE(k) is *true*.

We check for valid edges in both the types of planning algorithms by performing on-demand stereo correspondence checks using a sum of absolute differences (SAD) of DAISY descriptors (Tola *et al.* 2010) of stereo image points.

On-demand Stereo

Since the stereo cameras are well calibrated we know the projection matrices $\mathbf{P}$ and $\mathbf{P}'$ (Equation 1) in the rectified coordinate system of the left and right images respectively. Therefore any space point X (represented as a 3×1 matrix) in the robot reference frame is first transformed into the left camera reference frame as

$$X_l = \mathbf{R}_w^{-1}(X - \mathbf{t}_w) \tag{3}$$

and then X_l is projected as image points p and p' in the rectified stereo images I and I', where $p = \mathbf{P}X_l$ and $p' = \mathbf{P}'X_l$ (in homogeneous coordinates). Given two image points $p(u, v)$ and $p'(u', v')$ in the rectified stereo images I and I', we define the SAD cost function as

$$\mathcal{C}(p, p', w) = \sum_{\substack{q \in \tau(p,w) \\ q' \in \tau(p',w)}} |\mathbf{D}(q) - \mathbf{D}(q')| \tag{4}$$

where $\tau(p, w)$ is a window of pixels of size $w \times w$ centered around the point p and similarly $\tau(p', w)$ is defined around p'. $\mathbf{D}$ denotes the DAISY descriptor of any image point. The descriptor parameters are characterised by 4 variables: R (radius), Q (radius quantization number), T (angular quantization number), H (histogram quantization number). The descriptor size is calculated as $(QT+1)H$. For an image resolution of 320 x 200, the values of these parameters are given in Table 1.

Given any point p in the the left image I, its disparity level d is determined by finding the best corresponding match in the right image I' by scanning along the epipolar line and finding a point q such that $\mathcal{C}(p, q, w)$ is minimum over the epipolar line. Formally

$$d = \arg\min_{d \in \mathcal{D}} \mathcal{C}(p(u, v), q(u - d, v), w) \tag{5}$$

where $\mathcal{D} = [0, d_{\max} - 1]$ is the set of possible disparity values of p. Disparity refinement is done using left-right consistency checks, and low confidence matches are neglected using a threshold on the ratio of cost of the top two disparity candidates. This local disparity matching method is implemented to generate dense disparity maps on high resolution images and is used to compare paths generated by this method and JPP.

Confident Positive Matching

A visible space point $X \in \mathbb{R}^3$ under consideration of the obstacle avoidance planner is first projected on to its corresponding rectified image coordinates p and p'. The confident positive match verifies that X belongs to the ground plane *i.e.*, it verifies that the z-coordinate of X is zero. To do this we need to verify that p and p' are valid stereo correspondences. A function $\mathcal{L}_+(X)$ used to label point X is defined as

$$\mathcal{L}_+(X) = \begin{cases} 1, & \text{if } \mathcal{C}(p, p', w) \leq \epsilon_g \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where ϵ_g is a constant associated with space points classified as ground plane. $\epsilon_g = 1.1$ is found experimentally, and it depends on the length of the DAISY descriptors, and the SAD window size. X is classified as ground plane if $\mathcal{L}_+(X) = 1$ and as an obstacle if $\mathcal{L}_+(X) = 0$. Sometimes X is wrongly classified as an obstacle when it is actually ground plane. Hence we apply a spatial filter around X to remove noisy estimates. We select a discretized window W with grid size w_x around X and count the number of points $X' \in W$ such that $\mathcal{L}_+(X') = 1$. Formally let $n_x = \sum_{X' \in W} \mathcal{L}_+(X')$. If $n_x > c_x n(W)$, where c_x denotes the spatial filter threshold and $n(W)$ denotes the cardinality of the set W, then we let $\mathcal{L}_+(X) = 1$ otherwise $\mathcal{L}_+(X) = 0$. Experimentally, for best results we set $W = 5$ cm $\times$ 5 cm, $w_x = 10$ mm, $c_x = 0.75$.

Confident Negative Matching

Confident negative matching is used to verify that a space point $X \in \mathbb{R}^3$ is empty. The procedure is almost similar to that of confident positive matching. First X is projected into corresponding rectified image point coordinates p and p'. Then a function $\mathcal{L}_-(X)$ used to label point X is defined as

$$\mathcal{L}_-(X) = \begin{cases} 1, & \text{if } \mathcal{C}(p, p', w) \geq \epsilon_o \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where ϵ_o is a constant associated with empty space points. $\epsilon_o = 0.5$ is found experimentally, and it also depends on the length of the DAISY descriptors, and the SAD window size. X is classified as empty if $\mathcal{L}_-(X) = 1$ and as non-empty if $\mathcal{L}_-(X) = 0$. We use the confident negative checks to verify that a column of points above a space point (classified as ground plane) is empty *i.e.*, it does not contain any obstacle. The column of points will be empty if for all points X belonging to that column, $\mathcal{L}_-(X) = 1$. If for some X, $\mathcal{L}_-(X) = 0$ then it is not an empty column. It is important to note here that $\epsilon_g \neq \epsilon_o$.

Reachable Poses

Using the definitions of confident positive and negative matching we can classify any pose $(x, y) \in \mathcal{X}$ as reachable or not reachable by the robot. Algorithm 1 outlines the procedure for the classification. The general idea of this algorithm is that for any pose (x, y) to be reachable by the robot, the set of space points points $\{(x', y', 0) : (x - x')^2 + (y - y')^2 < r^2\}$ (in the robot reference frame) need to be verified as points belonging to the ground plane by confident positive checks. Additionally, the column of points starting from $(x', y', 0)$ up to the robot height h *i.e.*, up to (x', y', h) should be empty by using the confident negative matching. r denotes the safety radius of the robot.

We consider two scenarios of the world: (1) *convex world* (2) *non-convex world*. In a convex world if any point $P(x, y, z)$ is not empty then the set of points $\{P'(x, y, z') : 0 \leq z' \leq z\}$ are also not empty. So in that case lines 11-16 in Algorithm 1 are not required *i.e.*, the confident negative matching is omitted. Omitting those extra confident negative checks would speed up our algorithm but accuracy may be lost. Convex world assumptions can be made for indoor environments where obstacles are mostly convex shaped, whereas it is reasonable to assume a non-convex world for outdoor environments.

Table 1 lists some important parameters crucial for joint perception and planning.

Algorithm 1 Check if pose p is reachable by the robot

1: **procedure** REACHABLEPOSE($p(x, y)$)
2: $w \leftarrow$ robot width
3: $l \leftarrow$ robot length
4: $h \leftarrow$ robot height
5: $r \leftarrow \max(\frac{w}{2}, \frac{l}{2})$ ▷ robot safety radius
6: $\mathbf{P}_r \leftarrow \{(x', y', 0) : (x - x')^2 + (y - y')^2 < r^2\}$
7: **for each** $P \in \mathbf{P}_r$ **do**
8: **if** $\mathcal{L}_+(P) == 0$ **then**
9: $\mathcal{X}_{\text{obs}} \leftarrow \mathcal{X}_{\text{obs}} \cup \{p\}$
10: **return false**
11: **else**
12: $\mathbf{P}_h \leftarrow \{(P_x, P_y, z) : 0 \leq z \leq h\}$
13: **for each** $P' \in \mathbf{P}_h$ **do**
14: **if** $\mathcal{L}_-(P') == 0$ **then**
15: $\mathcal{X}_{\text{obs}} \leftarrow \mathcal{X}_{\text{obs}} \cup \{p\}$
16: **return false**
17: $\mathcal{X}_{\text{free}} \leftarrow \mathcal{X}_{\text{free}} \cup \{p\}$
18: **return true**

Experimental Results

We performed two sets of experiments to 1) evaluate the computational cost of the JPP compared to SPP, and 2) to compare the path length of obstacle avoidacne as evaluated by online JPP compared to an offline, high-resolution dense SPP as a reference baseline. In both experiments we compared results using obstacle avoidance planning using both A* as well as an RRT planner. The first set of experiments were perfomed in simulation as well as on a real

Table 1: Thresholds and physical constants

Name	Symbol	Domain	Value
State space grid size	s	> 0	5 cm
RRT goal bias	b	$(0, 1)$	0.6
Conf. positive threshold	ϵ_g	> 0	1.1
Conf. negative threshold	ϵ_o	> 0	0.5
Spatial filter window size	w_x	> 0	10 mm
Spatial filter threshold	c_x	$(0, 1)$	0.75
DAISY radius	R	> 0	7
DAISY radius quantization	Q	> 0	3
DAISY angular quantization	T	> 0	3
DAISY histogram quantization	H	> 0	1

robot, a Clearpath Jackal UGV (Figure 2), equipped with two PointGrey Blackfly IMX 249 cameras, Kowa LM6HC lenses, and an Intel NUC for onboard processing. Over all experiments, the rectified stereo image resolution was scaled to 320 $\times$ 200 pixels. The grid/step size was set to 5cm.

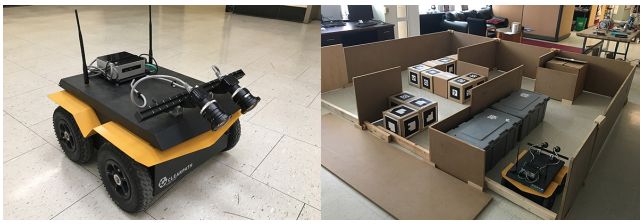


Figure 2: Clearpath Jackal UGV robot used for real-world experiments.

Computational Efficiency

Simulation Tests. We created a simulator that spawns random obstacles in front of a robot equipped with stereo cameras. The obstacles were in the shape of cylinders with a fixed radius and height. We assumed a world size of 6m x 6m. 100 obstacles with base radius of 8cm and height of 40cm were spawned randomly in each simulation. In each simulation we set an end waypoint of 2m ahead of the robot center. We ran around 46,000 simulations each for RRT and A*, and logged the total number of disparity cost computations (SAD checks) for both convex and non-convex world scenarios. The goal of this experiment was to compare JPP with SPP and evaluate the number of disparity cost computations. This experiment was also designed to find experimental bounds on the number of computations required for different planners. The number of computations required by SPP is a constant for every simulation since it reconstructs a dense 3D scene by a local cost aggregation method (2560000 for an image resolution of 320 x 200 with a maximum allowable disparity of 40 pixels). The computations required by the planner is negligible compared to computations for full reconstruction. We plotted the number of computations taken by JPP as a fraction of computations by SPP with the x-axis as the path length. Figure 3 shows that for A* in simulation, the fraction of computations is less than 0.9% for the non-convex scenario, and less than 0.2% for the convex scenario. For RRT the numbers are 10% and 2% respectively. Figure 4 shows a cumulative histogram of the fraction of the computations. We thus verify our hypothesis from these figures and also conclude that the complexity of JPP is a function of the complexity of the path planner.

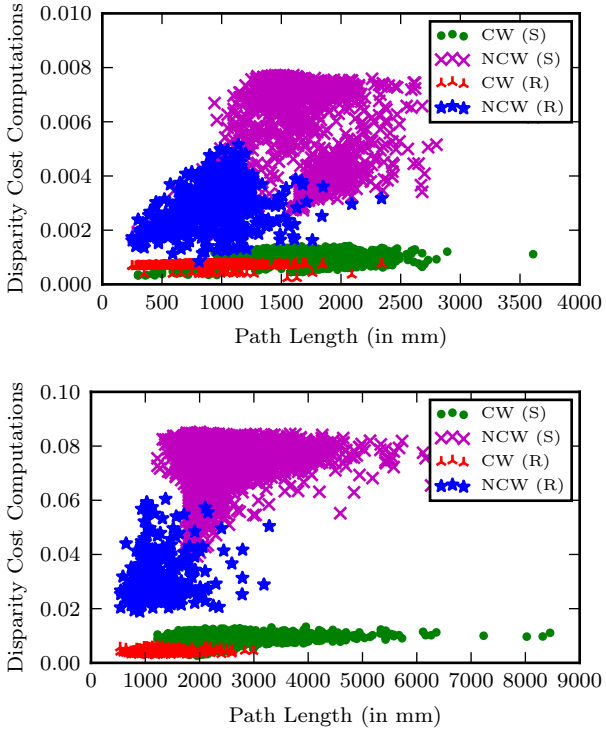


Figure 3: Fraction of total SAD computations of JPP compared to SPP, vs. Path Length. **Top**: A* planner. **Bottom**: RRT planner. Red and blue points are from real world data while purple and green are from simulation. The number of computations is expressed as a fraction of that required by SPP. **CW**: convex world. **NCW**: non-convex world. **S**: simulation. **R**: real world.

Real World Tests. We used the Jackal to verify the bounds found in simulation, and test robustness on detecting and avoiding obstacles in real world. The tests were done both for non-convex and convex world assumptions. Figures 3 and 4 clearly show that the number of computations required in real world is within the simulation bounds. For A*, the numbers are 0.7% (non-convex world) and 0.2% (convex world). For RRT the numbers are 7% and 2% respectively. Figure 6 shows that our method is robust and efficient in detecting obstacles and planning safe paths around them. Figure 6 also verifies that only sparse disparity checks ($\mathcal{L}_+$ and $\mathcal{L}_-$) are required for obstacle avoidance. We also deduce visually that the number of computations vary based on the type of planner we use. From the plots we can deduce that RRT does 10% more computations than A*.

Path Quality

To evaluate the quality of paths generated by JPP, we compared them to reference paths generated offline by SPP from dense 3D reconstructions on high resolution images of resolution 1920 $\times$ 1200 pixels. Note that the reference paths are indicative of the highest possible quality that can be generated from stereo vision, and cannot actually be run

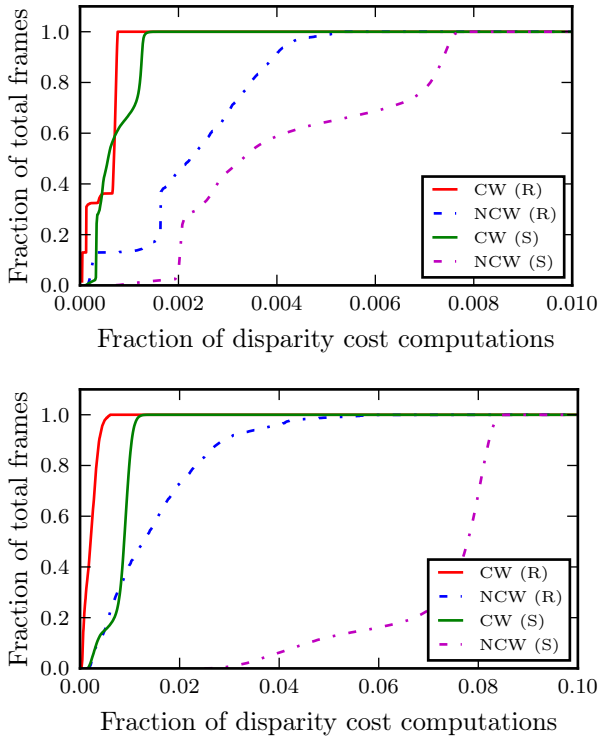


Figure 4: Cumulative histogram of the fraction of disparity cost computations compared to SPP. **Top**: A* planner. **Bottom**: RRT planner. **CW**: convex world. **NCW**: non-convex world. **S**: simulation. **R**: real world.

in real-time due to their significant computational cost: they require more than 2 minutes to generate per frame. We use the Hausdorff distance $\mathcal{H}$ to compare two paths $\mathbf{P}_1$ and $\mathbf{P}_2$. $\mathcal{H}$ is defined as

$$\mathcal{H}(\mathbf{P}_1, \mathbf{P}_2) = \max_{p \in \mathbf{P}_1} \left(\min_{q \in \mathbf{P}_2} (\|p - q\|) \right) \tag{8}$$

where p and q are points that make up the paths $\mathbf{P}_1$ and $\mathbf{P}_2$. Figure 5 shows a cumulative histogram of the Hausdorff distances for both RRT and A*. From the histogram we see that the paths generated by JPP and from high resolution 3D reconstruction are comparable as more than 80% of the cases have a Hausdorff distance of less than 0.6m.

The computational complexity of JPP is invariant of image resolution as the number of disparity checks is guided by the path planning algorithm. Recall that we project a space point in the robot reference frame into two image points and perform confidence matching only. In SPP we are forced to work with low resolution images, since 3D reconstruction is expensive for high resolution images. We take advantage of this fact, to perform more confident and robust disparity checks using SAD with a bigger window size on higher resolution images, and utilize the saved computation time.

Conclusion and Future Work

In this paper we introduced a novel joint perception and planning (JPP) algorithm for obstacle avoidance using stereo vision. We showed experimentally that the JPP requires significantly fewer computational resources, while still maintaining high path quality. Since the total number of SAD disparity cost checks in JPP is invariant of the image resolution, analysis of the number of disparity computations, as a function of the planning problem, is a promising direction for future work.

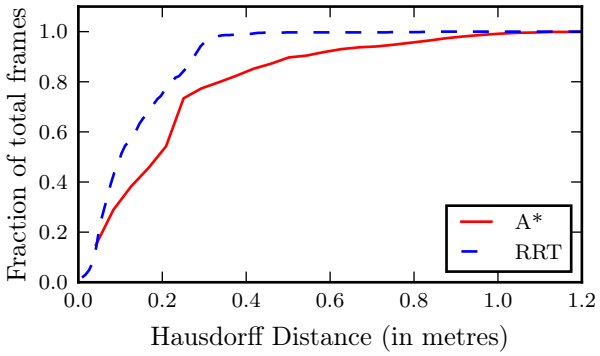


Figure 5: Hausdorff distance of paths generated by online JPP, compared to offline high-resolution dense SPP reconstruction.

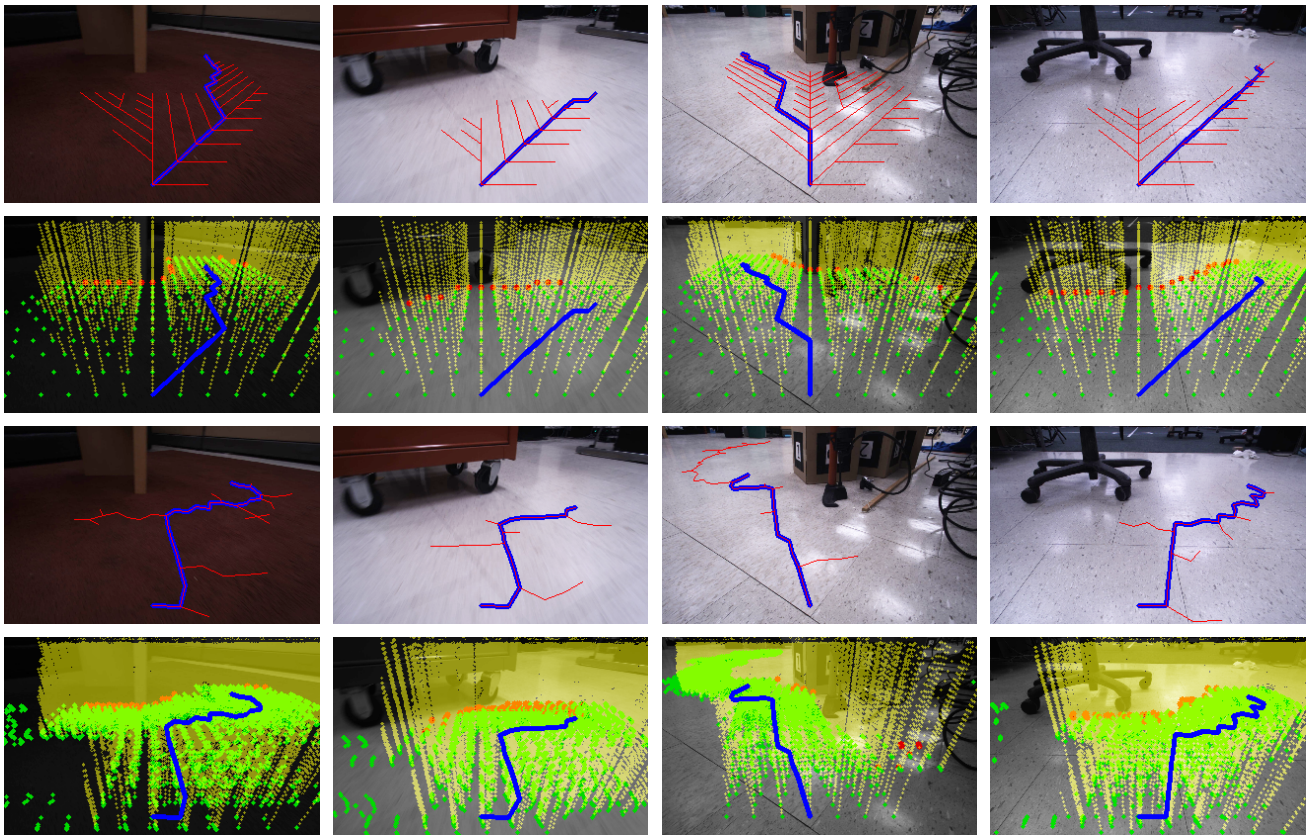


Figure 6: Joint perception and planning visualizations on the real-world dataset collected at AMRL, University of Massachusetts Amherst. **Row 1**: A* planner. Red curves indicate the explored planning graph, blue curve indicates the path planned. **Row 2**: Confident positive/negative matching visualizations. Green points belong to the ground plane which are found by confident positive checks, red points indicate configurations not reachable by the robot. Yellow points represent empty space points found by confident negative matching. **Row 3 and 4**: RRT planner. The colour coding is same as that of A*. It is evident from these visualizations that JPP performs sparse disparity checks as compared to dense reconstruction. Note that RRT performs more dense checks than A*.

References

Kai O Arras, Jan Persson, Nicola Tomatis, and Roland Siegwart. Real-time obstacle avoidance for polygonal robots with a reduced dynamic window. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3, pages 3050–3055. IEEE, 2002.

Andrew J Barry and Russ Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3046–3052. IEEE, 2015.

Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702. IEEE, 2012.

Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 572–577. IEEE, 1990.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

Sven Koenig and Maxim Likhachev. Dˆ* lite. In *AAAI/IAAI*, pages 476–483, 2002.

Masako Kumano, Akihisa Ohya, and Shinichi Yuta. Obstacle avoidance of autonomous mobile robot using stereo vision sensor. In *Intl. Symp. Robot. Automat*, pages 497–502, 2000.

Scott Lenser and Manuela Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 886–891. IEEE, 2003.

Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600. ACM, 2005.

Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8(2):161–171, 2000.

Will Pryor, Yu-Chi Lin, and Dmitry Berenson. Integrated affordance detection and humanoid locomotion planning. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 125–132. IEEE, 2016.

Kohtaro Sabe, Masaki Fukuchi, J-S Gutmann, Takeshi Ohashi, Kenta Kawamoto, and Takayuki Yoshigahara. Obstacle avoidance and path planning for humanoid robots using stereo vision. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 592–597. IEEE, 2004.

Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

Reid Simmons, Lars Henriksen, Lonnie Chrisman, and Greg Whelan. Obstacle avoidance and safeguarding for a lunar rover. In *AIAA Forum on Advanced Developments in Space Robotics*, 1996.

Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 4, pages 3375–3382. IEEE, 1996.

Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):2, 2007.

Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

Min Tang, Young J Kim, and Dinesh Manocha. Ccq: Efficient local planning using connection collision query. In *Algorithmic Foundations of Robotics IX*, pages 229–247. Springer, 2010.

Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):815–830, 2010.

Iwan Ulrich and Illah Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *AAAI/IAAI*, pages 866–871, 2000.

Liangjun Zhang, Young J Kim, and Dinesh Manocha. A simple path non-existence algorithm using c-obstacle query. In *Algorithmic Foundation of Robotics VII*, pages 269–284. Springer, 2008.

On the Application of Classical Planning to Real Social Robotic Tasks

José Carlos González, Fernando Fernández, Ángel García-Olaya and Raquel Fuentetaja
Planning and Learning Group, Department of Computer Science
Universidad Carlos III de Madrid, Leganés 28911 - Madrid, Spain
{josgonza, ffernand, agolaya, rfuentet}@inf.uc3m.es

Abstract

Automated Planning is now a mature area offering several techniques and search heuristics extremely useful to solve problems in realistic domains. However, its application to real and dynamic environments as Social Robotics requires much work focused, not only in the efficiency of the planners, but also in tractable task modeling and efficient execution and monitoring of the plan into the robotic control architecture. This paper identifies the main issues that must be taken into account while using classical Automated Planning for the control of a social robot and contributes some practical solutions to overcome such inherent difficulties. Some of them are the discrimination between predicates for internal control and external sensing, the concept of predicted nominal behavior with corrective actions or plans, the continuous monitoring of the plan execution and the handling of action interruptions. This manuscript highlights the dependencies between all the design and deployment activities involved: task modeling, plan generation, and action execution and monitoring. A task of Comprehensive Geriatric Assessment (CGA) is used as an illustrative example that can be easily generalized to any other interactive task.

Introduction

Simple or very specialized robots can work just with reactive behaviors, but autonomous robots that must take decisions and change their behavior according to the context, as social robots do, must have a deliberation process at some point. Automated Planning (Ghallab, Nau, and Traverso 2004) is very useful to manage this deliberation and, in particular, classical planning is able to exploit many of the latest contributions and advancements of the field. Classical planning already has impressive results finding plans very fast in multiple domains, but it still has a low deployment in real life applications. This manuscript tries to bring classical Automated Planning techniques closer to real social robotic scenarios.

Social robots (Leite, Martinho, and Paiva 2013) has to deal autonomously with people, requiring the monitoring of highly dynamic environments with a lot of uncertainty. Therefore, executing a plan of actions with real social robots is not straightforward because an action can fail or maybe the plan can become unfeasible due to some change in the environment. Joining planning and execution is not deeply studied in the literature although the need has already been pointed out (Ghallab, Nau, and Traverso 2014).

The execution must be controlled to change the plan depending on the evolution of the environment. Interruptions in the execution of the plan can interfere with a coherent social interaction, so the modeling of the domain has to take this matter into account. This environment can be composed of low-level information from the sensors that needs to be abstracted to high-level data.

In this work we apply classical planning interleaved with execution. Classical planning has several advantages over other automated planning models as hierarchical or probabilistic. It allows much more compatibility with existing planners which can be used as black boxes. Also, the newest heuristics are made for classical planning, so we can take advantage of them. Uncertainty is not represented explicitly, but it can be controlled with existing planning architectures based on replanning techniques. Regarding the representation language we use PDDL (Edelkamp and Hoffmann 2004; Fox and Long 2003), the standard language developed by the academic community. Specifically, our model requires types, negative preconditions and numeric fluents. PDDL facilitates the modeling of the knowledge in our domain.

The main objective of this manuscript is to highlight the issues that arise when joining classical planning and execution into a social robot.

Challenges of Social Robotics

Social Robotics is focused on all those robots that must interact socially with humans. Achieving a natural and fluent interaction is currently a huge challenge and an active research topic (Tapus, Matarić, and Scasselati 2007). Human communication includes many verbal and non-verbal elements and it is in the act of talking when this interaction becomes more sophisticated. A social robot will be useful only if it can interact socially and efficiently, so a social robot should have many of the characteristics of conversational agents.

In general, every social interaction involves some sort of transmission of information that can be driven by the structure of a conversation. If this structure is altered, then the communicative act could be incoherent or difficult to follow. The characteristics that a natural communication should have are also studied by fields as Pragmatics (Warren 2006)

and, depending on the objective, it has different phases that must be performed in order.

Apart from microphones and speakers to allow verbal communication, social robots can have non-verbal communication mechanisms as faces shown in a screen, robotic faces, lights, etc. Elements as touchscreens can also be used to circumvent current limitations of audio and image recognition, for instance. A mechanism to coordinate all these elements to achieve the most coherent and natural interaction as possible is essential in Social Robotics.

Additionally, these robots must function in very dynamic environments. In other words, they must respond coherently to a great amount of unexpected situations which can cause interruptions in the standard behavior. If after an interruption the robot resumes the conversation abruptly, if the response time is too high or if the robot does not respond to evident stimuli, the interaction will be negatively affected and the user will not consider it as something natural.

Apparently subtle problems can undermine this interaction and make it annoying or boring. To add more difficulty, details of all these elements as silences, interpersonal distances, the gaze direction or the forms of address can have very different meanings depending on the language, the culture, the sex or the specific person (Stivers et al. 2009).

Among all challenges with social robotics, this manuscript focuses specifically in planning the behavior of the robot at each moment. Since it is a social robot, the interaction through conversation is of capital importance. The needed deliberative process to make the robot behave coherently in one way or another depending on the situation can be solved with techniques as classical Automated Planning (Petrick and Foster 2013), avoiding to create and maintain huge finite-state machines or scripts. This manuscript also describes a real example of a social robot which uses Automated Planning and the complexity that these considerations can reach.

Related Work

Deciding the appropriate behavior of a social robot in stochastic, highly dynamic environments is a task that can be accomplished in several ways, some more convenient than others depending on the final objective. The two main aspects to consider include the way knowledge about the environment and capabilities of the robot is represented and how the reasoning using this knowledge is performed.

Knowledge Representation

Some aspects of the environment have to be represented somehow to reason about it. Humans gather information from the senses and store it in the brain in a subsymbolic way, within a neuronal network. There are robots that use this kind of knowledge representation (Baxter, de Greeff, and Belpaeme 2013; Prenzel, Feuser, and Gräser 2005). However, subsymbolic models are difficult to be reused for other solutions because they cannot be directly understandable by humans or machines, and usually they require a previous training process.

More commonly, the information to reason about the environment has been represented in a symbolic way. A direct way to work with it is through finite-state machines (Suárez-Mejías et al. 2013). In them, each state corresponds to a certain situation in which the robot can be during its execution, depending on the previous actions and the information perceived by its sensors. Each state is a combination of the modeled parameters of the environment and has a set of applicable actions. Depending on which one is executed, the robot will transit to some states or others. For simple robots this is a very fast mechanism to implement, but in more sophisticated robots it can be very hard to identify and specify correctly all possible states that could appear. Moreover, adding or modifying functionality afterwards can be very hard given that all behavior is heavily hardcoded.

When there are too many states to be maintained, there are techniques to delegate the selection of actions to an algorithm, which can be used as a black box. Automated Planning techniques are useful here. They use the description of the possible actions and the description of the initial environment to generate a plan of actions that makes the robot to accomplish some goals.

Classical Automated Planning

Automated Planning (Ghallab, Nau, and Traverso 2004), in particular action-based planning, uses two different concepts: actions and states. The execution of an action allows the transition from a certain state to another state. The objective of this technique is to find a sequence of actions to transit from an initial state to a final state in which a certain set of goals is fulfilled. A convenient way to represent this knowledge is through the domain and the problem. On one hand, the domain includes the catalog of possible action schemes, each one with preconditions that must be fulfilled in the state of the world (the modeled environment) to allow its execution and the effects in that state after its execution. On the other hand, the problem includes the description of the initial state of the world and a set of goals that must be accomplished in the final state to consider that the task is done. The domain and the problem are introduced into an automated planner that will try to find a plan of valid actions to transit from the initial state to the final state. There are many domain-independent automated planners available, most of them relying on advanced heuristic search techniques, that can be used as a black box to find a suitable plan as fast as possible. This allows the developers to focus on a higher level of abstraction; just the possible actions of the robot and the facts characterizing a state must be modeled using a symbolic language as the Planning Domain Definition Language (PDDL) (Fox and Long 2003). As a drawback, this technique is slower than domain specific techniques, like finite-state machines, so the design of the domain must be performed with care to be suitable for the fast response time required for social robots.

There are recent examples of Automated Planning for language generation and dialogue control (Steedman and Petrick 2007; Brenner and Kruijff-Korbayová 2008). Classical planning is deterministic and when interleaved with execution it requires mechanisms to recover from unexpected situations. Therefore, this work uses the planning and monitoring architecture PELEA (Alcázar et al. 2010) to con-

trol the deliberation according to the results of the execution. PELEA could be versatile enough to allow the use of learning techniques for Social Robotics (Arora et al. 2016) and plan repairing strategies (Fox et al. 2006). This kind of architecture also allows other improvements to minimize the response time of the robot by planning the first actions with precision and continue refining the final parts of the plan while the previous actions are being executed (Martínez 2016).

Much of the current research about Social Robotics relies on classical planning and replanning approaches (González, Pulido, and Fernández 2017; Chen, Yang, and Chen 2016; Vaquero et al. 2015; Rosenthal, Biswas, and Veloso 2010), as is discussed in this manuscript. There are also works about modeling social aspects of human-robot interaction for Automated Planning (Carlucci et al. 2015). However, to the best of our knowledge, none of them describes systematically the specific considerations that they had to follow while joining planning and execution to develop a competent social robot.

Every social robotic domain is hierarchical because it can be decomposed in subtasks, probabilistic because predictions of the future are needed in order to generate a plan and temporal because each action has a duration along it could be interrupted. The following subsections discuss hierarchical, probabilistic and temporal planning models and motivates the use of classical planning for Social Robotics.

Hierarchical Planning

In general terms, any complex enough activity has a hierarchic structure that is composed of a set of tasks that can be subdivided into more specific ones (Ghallab, Nau, and Traverso 2014). For instance, a social robot could have to do a questionnaire to a user. It will have to ask him several questions, each question will need to be read (through a text-to-speech mechanism) and then the answer will be heard. To read it is needed to find what to say and then play it trough speakers, etc. There are phases in the questionnaire and in the whole conversation that must be accomplished in order to finish the task correctly.

If the granularity of the deliberation is high enough then a hierarchical planner can be used directly (Nau et al. 2003). These planners use more domain knowledge to plan, sometimes improving planning times, but at the expenses of a less generic solution. Hierarchical planning also does not consider the probabilities nor the temporal aspects of Social Robotics domains.

Often, it is not needed to take deliberation to such fine-grain level and it is enough to plan at a higher level. The low level actions can be performed reactively. Depending on the robotic architecture, a component, or a reactor, could receive an action and execute it without more deliberation, like moving a robot from one point to another (Bandera et al. 2016). This has the advantage of distributing the robotic architecture into several specialized components (or reactors) for some complex reactive tasks, without having to deliberate with too much specific knowledge. This manuscript also describes a real example which uses reactors to delegate low-level actions.

Probabilistic Planning

Other planning techniques take directly into account the stochastic nature of the dynamic environments (Little and Thiébaux 2007) present in Social Robotics. A probabilistic contingent planner can plan a set of different contingent plans to be used in case that the standard plan cannot be executed due to more or less probable unforeseen events. The first drawback of these techniques in comparison to classical planning is that they can be much slower. Since the reaction time of these robots must be very fast, classical planning reaches faster planning times at the expense of a higher number of replannings. Moreover, the biggest advances in heuristics for planning are within the classical planning area, so probabilistic techniques cannot take full advantage of them.

In Social Robotics its impossible to know the exact probabilities of each effect of the action. Learning them could help to generate better plans, but at the end each person reacts in a particular way, and unpredictable interruptions can appear in any moment (as the the user leaving the room) that must be considered.

Thanks to the use of planning architectures as PELEA, it is possible to use deterministic techniques as classical planning into stochastic environments.

Temporal Planning

Temporal planning (Fox and Long 2003) uses durative actions to generate plans with concurrent actions which apply effects at the beginning or at the end of the action. Although in a social domain, as in a conversation, it is impossible to determine the duration of an action such as speak and the interaction is made in sequential steps which normally do not overlap among them, there are interesting features that can be taken from temporal planning. In particular, durative actions consider three types of conditions: “at start”, “over all” and “at end”. These conditions must be held along different moments of the execution of the action. These reasoning is needed to interrupt an action in the middle of its execution or at the end. It can be taken into account in PELEA by using specific labels directly in the PDDL code, avoiding incompatibilities between durative actions and many automated planners.

In essence, classical planning can be used along with a planning architecture as PELEA to control the hierarchical, probabilistic and temporal nature of these domains and also taking advantage of the ease of modeling of PDDL, advanced heuristics and high planner compatibility.

The Clarc Use Case

For the rest of this manuscript, the Clarc social robot (Bandera et al. 2016) is used as an example to illustrate the contributions. This section explains one of the use cases of this robot. It uses classical planning and the monitoring architecture PELEA to deliberate about its behavior.

The Clarc robot is part of an European ECHORD++ research project[1]. Its main goal is to save clinicians’ time

[1] http://www.clarc-echord.eu

by assisting elder people while performing Comprehensive Geriatric Assessment (CGA) tests to measure their general health, habits of their daily life and the ability to perform some activities without help. Many of these are questionnaire-based tests which usually are held in hospitals with the assistance of a clinician. There are many different tests to evaluate different aspects of the patients, but to better explain the social task, this section is focused on only two CGA tests.

Clarc 1 has a touchscreen, speakers, a microphone and a 3D sensor. CGA tests are very long and heavily based on speech, so the conversational requirements of this robotic platform are very demanding. The text-to-speech and speech recognition mechanisms are implemented in internal components of the robotic architecture, so they are out of the scope of this manuscript. Their capabilities are enough to reproduce the text of each question and to recognize the needed answers. The physical embodiment of the robot can also be used to improve the interaction, but the current prototype is specially focused on the development of the conversation for the tests. All the robot behavior has to be modeled within a PDDL domain.

Figure 1: The current Clarc robot prototype with a patient.

When the patient is sat in front of the robot, the clinician selects a test and the robot starts working autonomously. An easy test to introduce the basics of the use case is the Barthel (Mahoney and Barthel 1965) one. It measures the patient's autonomy in his daily life. The flow is very straightforward, every question has fixed answers that must be answered to finish correctly and return a final score. Figure 2 shows an extract of a possible initial plan of a Barthel test. As it can be seen, after configuring some elements of the test that is going to be executed, the robot introduces itself. The first parameter is a label which identifies the speech to be played through the speakers. All parameters starting with "p_" indicate the amount of time to wait until starting the next action. After introducing the test with some basic instructions, it starts reading the statement of each question slowly. The parameter "first" indicates that it is the first time that this question is read. Then it waits 10 seconds at most (as indicated by the last parameter "dur_10s") to receive the patient's answer. The executed flow of actions become more complex when the patient does not answer some questions or the robot cannot understand the answers. After a number of failed trials, the robot has to repeat the question in an alternative way to check if the patient can understand it. When all questions are made, the test is finished.

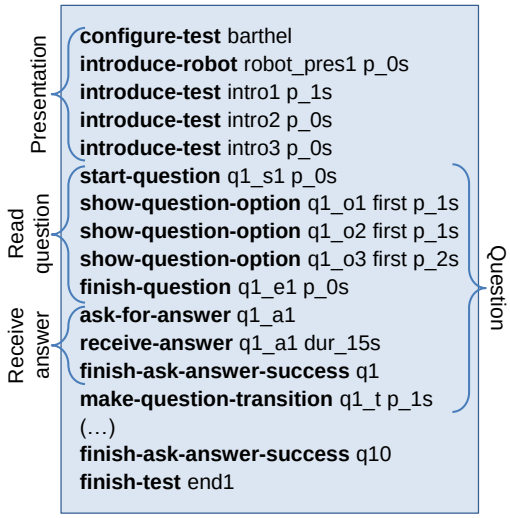


Figure 2: Example plan of a Barthel test.

All this process is executed assuming that the environment is suitable for the test. For instance, that the patient is in front of the robot, the test is not paused, there is enough battery, etc. If any of these variables change, then the execution may be interrupted and the robot must perform some corrective behavior accordingly to the detected situation. The structure of the conversation is controlled with the preconditions of the actions. For instance: after resuming the test, the robot must continue from a point which guarantees a coherent interaction with the patient, repeating some previously executed actions if needed. The timing of the conversation is controlled with the pauses and the maximum duration indicated in the parameters.

The Mini-mental test (Folstein, Folstein, and McHugh 1975) is much more complex than the Barthel one because all questions have open answers that the robot has to detect. The description provided here is to illustrate the extent of the complexity of the required behavior. The most basic questions are about the current day, month and season or the current building, city and country. The robot can detect a large set of possible answers that can be correct or incorrect. Sometimes an incorrect answer can be refined with another question for clarification. The test continues with questions about numeric operations (taking into account possible errors carried from earlier operations), repeating a list of words (altering the order is valid although not perfect), writing a syntactically correct sentence, following written commands as touching the nose and the right ear or closing the eyes, saying tongue twisters and even drawing two intersecting pentagons in the touchscreen. All this is in one long test, considering also all the corrective behaviors to manage

unexpected events.

As can be seen, each question has different interactive procedures that must be modeled in the domain that can be ruined by an unexpected interruption if it is not designed carefully. Apart from the challenges in speech recognition, computer vision and hardware coordination, the required behavior is too complex to model using finite-sate machines, so Automated Planning is useful here. However, joining the planning and the execution of the obtained plan together can be challenging too. In Clarc, the approach was to create a generic PDDL domain for questionnaire-based tests, generalizing each question as much as possible, and adapting the existing monitoring architecture PELEA to fit all communication requirements with the rest of the robotic architecture.

To design this kind of tasks properly, it is necessary to know the different issues that every developer of social robots with classical planning will need to face sooner or later. This manuscript focuses on describing these issues and their solutions.

Modeling

There are several challenges identified while modeling classical planning domains for Social Robotics. This section explains these points highlighting their impact on the interaction.

Island Domains

The social behavior has several rules that must be followed to achieve a coherent interaction. This usually implies plans that contains several ordered phases. In Clarc, for instance, these actions are the salutation, then the introductions, then the questions of the selected test and finally the farewell. We refer to these kind of domains as "island domains" where each phase constitutes a different isle.

From the planning perspective, every one of these phases can be modeled by introducing intermediate subgoals or landmarks and a mechanism to impose a total order among them. Previous work on landmarks (Hoffmann, Porteous, and Sebastia 2004) assumes that neither the landmarks nor their order are known a priori. They should be determined while planning. In "island domains" the challenge is how to build good and efficient models given that both the landmarks (phases) and their order are known.

Island domains would be very fast to plan because they are very sequential. This is important when planning for social robots, in which high planning time could undermine the interaction. However, depending on the size of problems and on how they have been modeled, planners can spend too much time on instantiation and preprocessing.

Nominal Behavior Prediction

While modeling a PDDL domain (Ghallab, Nau, and Traverso 2014) it is necessary to make some assumptions or predictions about the effects of the actions. In a real environment, these effects are stochastic because the actions can fail or an external event can change the environment. To deal with this uncertainty when planning, a "nominal behavior" can be assumed in the effects of each action, as shown in Figure 3. In the case of Clarc, this is the behavior which produces shorter plans because it is assumed that the patient will answer correctly to every question at the first try.

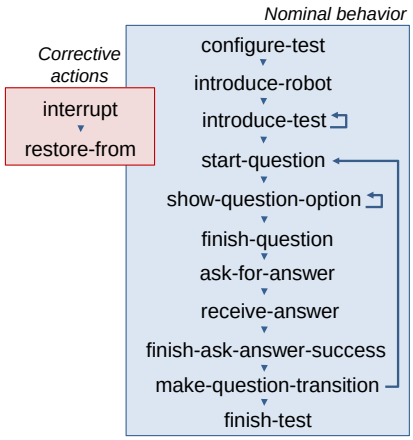


Figure 3: Possible plan of actions with the nominal behavior to perform a Barthel test in Clarc.

While executing this plan, it is very probable that the patient will fail at some point to answer a question. However, it is impossible to decide the amount of fails nor the precise questions in which they will occur. The design of this nominal behavior is at the developers' criteria. They could, for instance, to model a nominal behavior in which the patient fails each and every question, but this is less probable than answering correctly to all of them. As this is a design decision, developers could consider statistics, user preferences, etc. to increase the quality of the prediction.

Some kind of predicted behavior is needed in order to generate a valid plan. The nominal behavior can be seen as a guideline of the needed actions that are left to be executed to reach the goals.

Obviously, if the previous plan is no longer valid, another one must be generated, so replanning is considered part of the whole deliberative process even while modeling the domain.

State Decomposition

The execution of a plan requires a certain control of the external environment to check if the actions are correctly executed or not. When modeling the interaction it is important to represent both the robot internal state and the state of the environment. In Clarc, this state includes also the results of the interaction with the patient. Changes in the current state can be due to actions of the robot or to what is usually known as external or exogenous events. These external predicates can sometimes be predicted, but not always. We distinguish three different types of predicates in the state:

Internal predicates: These are used to control the domain and to organize the plan of actions. They can appear in the effects or preconditions of any action and will always have the expected value because they do not depend on anything external depending on the course of the execution. Internal predicates can represent a piece of information

as the predicate `(testIntroduction ?speech)` that indicates one of the speeches that has to be played during the introduction of a test, or they can be control predicates that denote the points of the interaction already planned (flags). Internal predicates are never changed externally during the execution.

Predicted predicates: Their value must be predicted in the nominal behavior in order to generate a complete plan, but the actual values depend only on external elements of the environment. For instance, a predicate as `validAnswer`, that represents the fact that the patient provided a valid answer to a question, needs to be predicted in the effects of an action like `CheckAnswer` to continue with the plan, following the nominal behavior. These predicates can be part of the effects and preconditions of any action, although the predicted values in the effects may differ from the actual ones obtained through the sensors.

Unpredictable predicates: These will never be part of the effects of the actions involved on the nominal behavior, although they will in the preconditions. Predicates as `pauseActivated`, representing the pause button was activated, are completely unpredictable because there is not any explicit action in the nominal behavior to change their value. A change in the value of these predicates is only triggered externally and in any moment of the execution.

This differentiation creates two types of world states, one in which the values of its predicates will be always as expected and another in which their predicates can change in any moment, invalidating the current plan in the middle of its execution.

Corrective Actions

When the expected state of the world differs from the actual state, a replanning may be needed. The new plan must contain some actions to correct the unexpected issue and to return to the normal flow of the nominal behavior. These are corrective actions which are never included in the initial plan for the nominal behavior.

For instance, Clarc must interrupt the execution of the plan if the patient leaves the test area. The new replanned plan should start calling the patient and searching for him before continuing with the rest of the test. After the execution of these corrective subplans, the nominal behavior can continue.

Modeling these corrective actions is important because it endows the system with much more responsiveness to the environment and a more coherent interaction. There is no need to increase the number of preconditions in the PDDL actions of the nominal behavior to check every considered issue. Only one unpredictable predicate in the preconditions indicating if the situation requires a corrective subplan is enough for most interactive applications. Then, the specific corrective subplan will be planned depending on the preconditions of the corrective actions.

Replanning in Interactive Steps

In any fluid social interaction there are several steps that must be followed in order. This is ensured by the nominal behavior, but when there are interruptions, the interaction can be compromised. For instance, in the Barthel test, Clarc must enumerate the options of a question to the patient immediately after reading the statement.

Suppose that the robot finishes reading the second option of a question and then, suddenly, there is an interruption that requires a replanning. The nominal behavior flow is deviated to fix the situation with a corrective subplan. After that, the execution returns to the nominal behavior, but in which point? Should the robot repeat the second option, none at all, start again from the first option or directly from the statement? Depending on the granularity of the actions, these decisions must be made thinking only in terms of interaction. Modeling them in the domain is somewhat special because the corrective plan must reset the values of internal predicates to repeat one or more previously executed actions to achieve a coherent interaction.

The number of actions to execute again after a replanning depends only in the moment in which the interruption occurs, so the execution is divided in interactive steps of different number of actions. An interactive step must be completely finished or then it has to be repeated again from its beginning. An example of these interactive steps can be seen in the sets of actions of Figure 2 like presentation, read question and receive answer.

Numerical Information

Socially interactive applications must be rich enough to be believable by their users. For instance, repeating too many times the same sentence can expose design problems in the social robot. Randomizing sentences can be easily done in a low level, but when the repetition involves a more complex behavior it should be taken into account in the planning domain.

Repetitions are only an example to illustrate the need of counting in domains for social interaction and in many other real world applications. In Clarc, especially for the Minimental test, the domain contains many numeric fluents to represent the order of the current question, the number of attempts for a question, the consecutive and total number of failed questions and so on. The use of numeric preconditions also saves much preprocessing time and simplifies the code of the domain. Modeling numbers or order relations with predicates would increase preprocessing time in Clarc to unbearable values, given that its current planning and replanning time is just below the limit for a fluid reaction in a social robot.

Using numeric fluents in big, real world domains, is easier to model, with less parameters in the actions (which implies less preprocessing time and less memory used) and there is no need to know the range of levels. Their downside is that many planners are not yet compatible with them nor have too many heuristics to ease the planning task.

For the modeling part, it is also important to note that the value of a numeric fluent could not appear in the standard output of the automated planner because it is not part of the parameters of the actions. This can be relevant while executing the plan and it is discussed later in this manuscript.

Execution

There are several specific aspects that must be taken into account to execute the generated plans into a real and dynamic environment. Some of these aspects have been already pointed out in the literature (Ghallab, Nau, and Traverso 2014), but others are more specific to Social Robotics. In fact, the application of Automated Planning in dynamic environments has not been studied that much, so the mechanism to join planning and execution is up to each developer. This section describes the main points that must be taken into account while executing the generated plans of actions in a social robot.

Continuous Monitoring

The first need that arises when executing plans in dynamic environments like a social robot is the ability to replan when something in the actual state of the world invalidates the current plan. This can happen, for instance, when the actual state differs from the expected state during or after the execution of each action. In fact, the model of the nominal behavior and corrective actions must take replannings into account, so monitoring is of capital importance.

Monitoring the execution continuously ensures that the robot can react when something unexpected happens. For this purpose, a mature planning framework as PELEA (Planning, Execution and Learning Architecture) (Alcázar et al. 2010) can be used. Figure 4 shows the main modules of this architecture. In essence, the working flow of PELEA is as follows. The Executive module has the domain and the problem with the initial internal state of the robot in PDDL. Then, it completes the predicted and unpredictable predicates of the state with the actual ones provided by the low-level sensors of the robot (steps 1, 2 and 3). This complete high-level state is sent to Monitoring (4) to check if it is compatible with the expected state of the world. If it is the first plan or if the previous plan is not valid anymore, Monitoring retrieves a plan from Decision Support which runs a certain automated planner (5, 6). This plan is stored in Monitoring, which returns the next action to Executive (7). If, in contrast, the actual state of the world is compatible with the expected one, then Monitoring just returns the next action of the previously planned plan (skips steps 5 and 6). Finally, Executive transform this high-level action into a set of low-level actions (8, 9) and sends it to the robot (10). Then it waits until the execution is finished. After that, it completes the expected state with the information of the sensors and the cycle starts again. In this scheme, the Executive has full control of the execution, timing the maximum duration of an action, pauses, etc. to control the pace of the interaction.

The compatibility criteria of the actual and expected states of the world depend on the developer or the application. Always replanning after a single change of a predicate could lead to unnecessary replannings due to this change could not affect to any precondition of the following actions. A more relaxed comparison could be much more interesting in cases when there is much information in the state and only a part of it is relevant to interrupt the nominal behavior. For instance, the criterion in Clarc is to check if the actual state of the world fulfills the preconditions of the next action. This criterion could not be suitable for other applications.

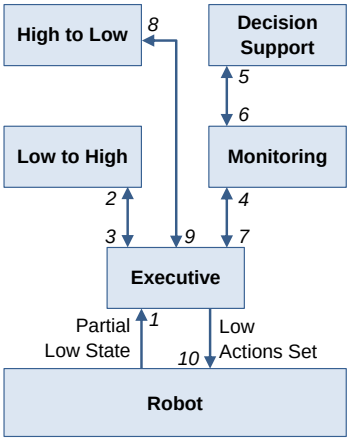


Figure 4: Basic planning and monitoring scheme of the Clarc robot with the PELEA architecture. Numbers indicate the communication flow between each module.

Also, as a side note, in these domains the goals should not be unexpectedly accomplished before finishing the execution of the nominal behavior.

Interrupting Actions

A true interactive system must be responsive in any moment of its execution. This is especially important while it is executing an action but it has not finished yet. A system that simply waits for the action to finish and then checks the state of the world is not realistic. For instance, if Clarc is executing a `Say` action, which involves a 20 seconds speech and the patient leaves the room in the 5th second, the robot must be able to interrupt the speech in the middle of its execution. If not, Clarc would continue speaking to nobody during the next 15 seconds before realizing that it has to call the patient to return to the test area.

To do this, the robot sends asynchronously the actual state of the world to Executive and then asks to Monitoring if the actual state is compatible with the action that is currently being executed. It is important to remark that the compatibility criteria after or during the execution of an action can be different. Clarc interrupts the execution if the new state violates any precondition of the action that is currently being executed. Tagging the parameters of each action in the PDDL code could easily indicate to PELEA if the value of a parameter must be the expected one only at the start of an action or during its whole execution. This is similar as temporal planning with its durative actions, but in Clarc these conditions are directly managed by PELEA because it is not needed to plan with overlapping actions.

All this indicates that it is important to pay attention to the granularity of the actions. Plans have a hierarchical nature, in which an action is later executed and refined by a component or "reactor" of the robotic architecture. More granularity of actions implies shorter ones and more autonomy for these reactors, and vice versa. Moreover, with shorter ac-

tions the domain will be more complicated in order to maintain the coherence of the interactive steps between replannings.

After the execution of an action, the robot must communicate to Executive that it has finished and that it is idle now. Then, Executive retrieves the last received actual state of the world and gives it to Monitoring to obtain the next previously planned or newly replanned action.

Planning Time Restrictions

Stochastic environments cause unpredictable situations that must be managed in these interactive systems. This means that replannings will appear, so it is very important to keep replanning times as low as possible to achieve a fluid interaction. After an interruption, the robot is going to stop in the middle of the interaction while it is replanning because it does not know the next action until it has the new plan. This means that in Social Robotics, planning must work in a way that there are not any detectable delay in the interaction.

For Clarc, a replanning time of 3 seconds is the maximum acceptable amount to achieve a fluid social interaction. Interestingly, almost all of its planning time is consumed on instantiation and preprocessing due to the large amount of different objects in the PDDL states of the world. Numeric fluents are useful to reduce the total planning time by decreasing the branching factor because they do not appear in the parameters of the actions.

Clarc, instead of stopping immediately after the triggering of an interruption, waits until the replanning is done. The previous action is interrupted only when Monitoring already has the new plan with the next action. This method avoids idle moments of the robot, allowing longer replanning times, but at the cost of slower reactions. The idea behind this is to keep the user engaged while planning proceeds.

Abstraction Levels in States and Actions

As it can be seen in Figure 4, actions and states are the main elements of information in a monitoring system as PELEA. But there are important differences in the abstraction level of these elements because the robot only works at low level and the planning system at high level. A conversion is needed to translate these actions to the robot and states from the robot. Figure 4 also shows the HighToLow and LowToHigh modules connected to Execution to manage this. This subsection expands the previous explanations detailing these abstractions:

- **High-level states**: PELEA uses them to plan. The actual state of the world is maintained by Execution and the expected state is maintained by Monitoring.
- **Low-level state**: The robot sends only a part of its low-level state to the Executive. This state always contains the latest information retrieved by the sensors and the robot. Executive sends this partial low-level state to the LowToHigh module, which abstracts this information into high-level predicates to complete the actual high-level state.
- **High-level actions**: These are the actions returned directly by the automated planner of Decision Support. Monitoring sends these high-level actions, one by one, to Executive.
- **Low-level actions**: When Executive receives a high-level action, it sends it to HighToLow to obtain a decomposition into a set of low-level actions. Then, Executive sends this set to the robot to execute these low-level actions. When all these actions are finished, the robot will communicate to Executive that it has finished the execution of the last low-level action set. Clarc considers that every low-level action in the set has to be executed in parallel, instead of sequentially. Each high-level action is modeled taking this into account because all of them have to be decomposed into a parallel set of low-level actions.

The HighToLow and the LowToHigh modules are ad hoc programmed for each domain because PDDL code cannot handle these decompositions and transformations between abstraction levels.

Retrieving the Value of Numeric Fluents

There are also some issues with the standard output of the automated planners. Numeric fluents are useful to model order relations. They do not appear in the parameters of the PDDL actions, reducing the branching factor, but this causes that their value is never shown through the output of the planner.

In Clarc, numeric values like the question number or the duration are needed when executing the actions and they are only in the expected state of the world, as the internal predicates. It uses the automated planners as black boxes, as PELEA does, so its solution to retrieve these values is generic, but inefficient. Its PDDL domain uses functions to relate the discrete values of the needed numeric fluents to predicates, inserting them into parameters in the actions to make them appear in the output of the planner. This cancels the benefits of using numeric fluents in such particular cases.

There are better solutions than increasing the number of parameters in the actions. Tagging the effects of actions in the PDDL code could be useful in this case too, and somewhat generic. It could be used to indicate that the value of certain numeric fluents must be printed after the action directly in the output plan.

Another solution could be, given the expected state, the domain and the plan, to recalculate the value of the needed numeric fluents in polynomial time. This could be done easily by parsing the output of the VAL application (Howey and Long 2003), which can be used to check the actual validity of the generated plans.

Main Performance Results with Clarc

The first prototype of Clarc included all previously discussed design considerations. The first evaluations were carried out with 24 senior end users. The main result is that the Clarc prototype finished 3 standard tests correctly, including the Barthel and Mini-mental, and reacted very fast to several unexpected events like calling the patient when he leaves the test area, offering clarifications to the patient when he suddenly asks for help or asking the patient to wait when the battery is discharged to a critical level.

Planning times increase as the length of the plan becomes longer. However, they are much lower than preprocessing times, which always remain the same for any length of the resulting plan. Figure 5 shows an example of the average times that Pelea needs to send a new planned or replanned action to Clarc. The average planning time is never higher than 3 seconds, which makes it suitable for the fast deliberations needed in this domain of Social Robotics. In the evaluation, all senior users interacted smoothly with Clarc and declared that they did not felt apprehension or discomfort with the robot.

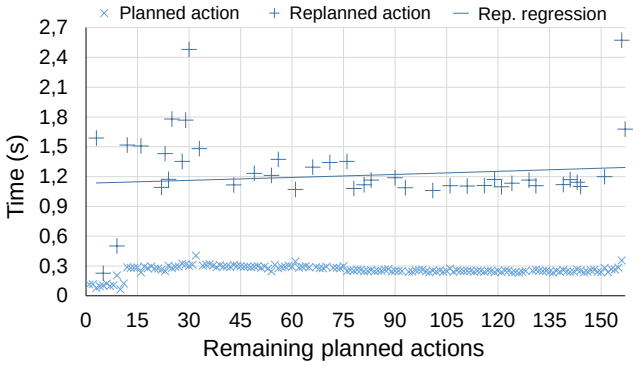


Figure 5: Average time to send a new planned or replanned action to the Clarc robot.

The next steps in the Clarc project include adding more than 10 standard geriatric tests and also customized ones by the clinician. This will require to modify and expand the Clarc domain taking the considerations described in this manuscript and check how generalizable is this solution.

Conclusions

One of the reasons behind the low impact of classical planning in real applications is the need of more research joining planning and execution. This manuscript contributes with the list of important aspects that must be taken into account while modeling a classical and deterministic PDDL domain for Social Robotics and executing the resulting plan in their highly dynamic and stochastic environments.

Constant monitoring of the execution and action interruptions are needed to be able to react to unexpected events, but fast replannings times are needed in order to achieve a fluid interaction. Numeric fluents are very important in many real problems with many objects to represent order relations without unneeded increments of the branching factor and preprocessing times.

As we have discussed here, planning, sensing and execution are interleaved tasks very dependent among them. All of them must be considered in each task when developing a complete social robotic platform.

References

[Alcázar et al. 2010] Alcázar, V.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindia, E. 2010. PELEA: Planning, Learning and Execution Architecture. In *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*.

[Arora et al. 2016] Arora, A.; Fiorino, H.; Pellier, D.; and Pesty, S. 2016. A Review on Learning Planning Action Models for Socio-Communicative HRI. In *Proceedings of the 7th Workshop on Artificial Companion, Affect and Interaction (WACAI)*.

[Bandera et al. 2016] Bandera, A.; Bandera, J.; Bustos, P.; Calderita, L.; Dueñas, A.; Fernández, F.; Fuentetaja, R.; García-Olaya, A.; García-Polo, F.; González, J.; Iglesias, A.; Manso, L.; Marfil, R.; Pulido, J.; Reuther, C.; Romero-Garcés, A.; and Suárez, C. 2016. CLARC: a Robotic Architecture for Comprehensive Geriatric Assessment. In *Proceedings of the 17th Workshop of Physical Agents (WAF)*, 1–8.

[Baxter, de Greeff, and Belpaeme 2013] Baxter, P. E.; de Greeff, J.; and Belpaeme, T. 2013. Cognitive architecture for humanrobot interaction: Towards behavioural alignment. *Biologically Inspired Cognitive Architectures* 6:30 – 39.

[Brenner and Kruijff-Korbayová 2008] Brenner, M., and Kruijff-Korbayová, I. 2008. A continual multiagent planning approach to situated dialogue. In *In Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue (LONDIAL)*, 61–68.

[Carlucci et al. 2015] Carlucci, F. M.; Nardi, L.; Iocchi, L.; and Nardi, D. 2015. Explicit Representation of Social Norms for Social Robots. In *Proceedings of the 28th International Conference on Intelligent Robots and Systems (IROS)*, 4191–4196.

[Chen, Yang, and Chen 2016] Chen, K.; Yang, F.; and Chen, X. 2016. Planning with Task-Oriented Knowledge Acquisition for a Service Robot. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 812–818.

[Edelkamp and Hoffmann 2004] Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classic part of the 4th International Planning Competition. Technical report, Institut für Informatik, Freiburg, Germany.

[Folstein, Folstein, and McHugh 1975] Folstein, M.; Folstein, S.; and McHugh, P. 1975. "Mini-mental state". A practical method for grading the cognitive state of patients for the clinician. *Journal of Psychiatric Research* 12(3):189–198.

[Fox and Long 2003] Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)* 20(1):61–124.

[Fox et al. 2006] Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Ghallab, Nau, and Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

[Ghallab, Nau, and Traverso 2014] Ghallab, M.; Nau, D.; and Traverso, P. 2014. The Actor's View of Automated Planning and Acting: A Position Paper. *Artificial Intelligence* 208:1–17.

[González, Pulido, and Fernández 2017] González, J. C.; Pulido, J. C.; and Fernández, F. 2017. A Three-layer Planning Architecture for the Autonomous Control of Rehabilitation Therapies Based on Social Robots. *Cognitive Systems Research*. Advance online publication: http://dx.doi.org/10.1016/j.cogsys.2016.09.003.

[Hoffmann, Porteous, and Sebastia 2004] Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

[Howey and Long 2003] Howey, R., and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in The International Planning Competition. In *Proceedings of the Workshop "The Competition: Impact, Organization, Evaluation, Benchmarks", ICAPS*, 28–37.

[Leite, Martinho, and Paiva 2013] Leite, I.; Martinho, C.; and Paiva, A. 2013. Social Robots for Long-Term Interaction: A Survey. *International Journal of Social Robotics* 5(2):291–308.

[Little and Thiébaux 2007] Little, I., and Thiébaux, S. 2007. Probabilistic Planning vs Replanning. In *Proceedings of the Workshop "International Planning Competition: Past, Present and Future", ICAPS*.

[Mahoney and Barthel 1965] Mahoney, F. I., and Barthel, D. W. 1965. Functional evaluation: The Barthel index. *Maryland State Medical Journal* 14:61–5.

[Martínez 2016] Martínez, M. 2016. *Automated Planning Through Abstractions in Dynamic and Stochastic Environments*. Ph.D. Dissertation, Universidad Carlos III de Madrid, Leganés, Spain.

[Nau et al. 2003] Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

[Petrick and Foster 2013] Petrick, R. P. A., and Foster, M. E. 2013. Planning for Social Interaction in a Robot Bartender Domain. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, 389–397.

[Prenzel, Feuser, and Gräser 2005] Prenzel, O.; Feuser, J.; and Gräser, A. 2005. Rehabilitation Robot in Intelligent Home Environment Software Architecture and Implementation of a Distributed System. In *Proceedings of the 9th International Conference on Rehabilitation Robotics (ICORR)*.

[Rosenthal, Biswas, and Veloso 2010] Rosenthal, S.; Biswas, J.; and Veloso, M. M. 2010. An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 915–922.

[Steedman and Petrick 2007] Steedman, M., and Petrick, R. P. A. 2007. Planning dialog actions. In *In Proceedings of the 8th Workshop on Discourse and Dialogue (SIGDIAL)*, 265–272.

[Stivers et al. 2009] Stivers, T.; Enfield, N.; Brown, P.; Englert, C.; Hayashi, M.; Heinemann, T.; Hoymann, G.; Rossano, F.; de Ruiter, J.; Yoon, K.; and Levinson, S. 2009. Universals and cultural variation in turn-taking in conversation. *Proceedings of the National Academy of Sciences of the United States of America* 106(26):1058710592.

[Suárez-Mejías et al. 2013] Suárez-Mejías, C.; Echevarría, C.; Núñez, P.; Manso, L.; Bustos, P.; Leal, S.; and Parra, C. 2013. Ursus: A Robotic Assistant for Training of Children with Motor Impairments. In *Converging Clinical and Engineering Research on Neurorehabilitation*, volume 1 of *Biosystems & Biorobotics*. Springer Berlin Heidelberg. 249–253.

[Tapus, Matarić, and Scasselati 2007] Tapus, A.; Matarić, M. J.; and Scasselati, B. 2007. Socially assistive robotics [Grand Challenges of Robotics]. *IEEE Robotics & Automation Magazine* 14(1):35–42.

[Vaquero et al. 2015] Vaquero, T. S.; Mohamed, S. C.; Nejat, G.; and Beck, J. C. 2015. The Implementation of a Planning and Scheduling Architecture for Multiple Robots Assisting Multiple Users in a Retirement Home Setting. In *Proceedings of the Workshop on "Artificial Intelligence Applied to Assistive Technologies and Smart Environments", AAAI*.

[Warren 2006] Warren, M. 2006. *Features of Naturalness in Conversation*, volume 152 of *Pragmatics & Beyond New Series*. John Benjamins.

Augmenting Planning Graphs in 2-Dimensional Dynamic Environments With Obstacle Scaffolds

Spencer Lane, Kyle Vedder, Joydeep Biswas
University of Massachusetts Amherst
140 Governors Dr.
Amherst, MA 01003
slane@cs.umass.edu, kvedder@umass.edu, joydeepb@cs.umass.edu

Abstract

In this paper, we present an extension to roadmap based path planners that allows for finer control over motions near dynamic obstacles. We utilize the fact that we know the shape of the dynamic obstacles offline and the location of the obstacles online. We supplement the roadmap graph by adding predefined graphs around obstacles, known as scaffold graphs. These graphs are inserted at query time and updated as the obstacles move throughout the environment. We performed a preliminary evaluation of our approach in the RoboCup SSL domain and show similar average case performance and improved performance in the case where there are obstacles between the start and the goal.

Introduction

The ability to find a collision free path between two points is required for the vast majority of tasks in robotics, regardless of the hardware those tasks are run on. A variety of solutions to this problem have been developed over the years, but we have chosen to focus on domains that have strict timing constraints, require high quality paths, where the shape of the dynamic obstacles is known offline and the locations of the dynamic obstacles are known online. The RoboCup Small Size League (SSL) is a good example of a domain with these properties.

RoboCup SSL is a RoboCup domain that focuses on multi-agent planning and coordination. Teams consist of 6 robots and a golf ball is used as a soccer ball. The game state evolves rapidly from one moment to the next, so a plan generated in the previous time step might not be valid. It is also desirable to update the robot control commands at the same rate new data is received; thus all processing and planning should be performed within 16 milliseconds. Finally, because RoboCup is a competitive domain, it is important to produce high quality paths within the time limit.

In order to improve path planning in domains with these properties, we propose a method of supplementing roadmap based path planners by constructing roadmaps, called scaffolds, around our obstacles offline and inserting them into the graph when updating the obstacle positions. We build on the framework introduced by Leven and Hutchinson to apply a roadmap path planner in a dynamic environment (Leven and Hutchinson 2002). Path planning is split into two steps: offline and online. In the offline step, a roadmap of the static environment is constructed. During the online step, the scaffolds around the moving obstacles are inserted and the edges blocked by obstacles are invalidated. In order to generate the initial roadmap, we utilize an existing sampling based path planner such as Probabilistic Road Map (PRM) (Kavraki et al. 1996), sPRM (Kavraki, Kolountzakis, and Latombe 1998), or PRM* (Karaman and Frazzoli 2011).

We begin with a brief review of the related literature. We then discuss the specific modifications to PRM introduced in this work. We discuss the formalisms for PRM graphs and scaffold graphs, review the primitive functions used in the construction of PRM graphs, and finally introduce a set of algorithms used for integrating scaffold graphs with existing roadmaps. Finally, we evaluate the performance of our scaffold approach in a variety of situations.

Related Work

PRM was introduced in 1996 as a method of path planning for robots working in static workspaces (Kavraki et al. 1996). In 1998, Kavraki, Kolountzakis, and Latombe introduced sPRM, which was shown to be asymptotically optimal (Kavraki, Kolountzakis, and Latombe 1998). In 2011, PRM* was introduced, which created an asymptotically optimal version of PRM with a reduced query time (Karaman and Frazzoli 2011).

Most PRM variant algorithms assume the workspace is static; however, a number of approaches exist for applying PRM in a dynamic environment. An early framework for doing so was proposed by Leven and Hutchinson (Leven and Hutchinson 2002). The key insight is to partition the workspace into the static and dynamic environments. Another approach to using PRM in dynamic environments is to use an RRT-like local planner to reconnect edges that have become invalid due to obstacles (Jaillet and Siméon 2004). van den Berg et al. place assumptions about the motion of obstacles in the environment, restricting them to a fixed set of configurations known a priori (van den Berg et al. 2005).

Within the RoboCup domain, it is common to use the execution extended RRT (ERRT) algorithm (Bruce and Veloso 2002). This extends RRTs to allow for re-planning in many situations. ERRT is used due to its fast planning, replanning and query times. Other approaches are typically not used as they often take too long to converge to a solution. There are a few exceptions however. For example, one team, SSH,

places four points around each of the obstacles and connects them to the start and goal points. They then use Dijkstra's Algorithm to find a path between the start and the goal. This is similar to a scaffold graph with four points and one layer; however, they do not add any additional points and it is unclear from the paper how well their algorithm would allow for routing around multiple robots (Emmerink et al. 2015).

PRM and Scaffold Definitions

In this section, we define the formalisms used in our approach. First, we review the definitions of graphs used in PRM and introduce the definition of a scaffold graph. Then, we discuss the primitive functions used in the construction of PRM graphs. Finally, we discuss the scaffold structure for circular obstacles and n-sided polygons.

Planning Graph

We start by defining a PRM Graph G as a set of edges E and vertices V. Each edge consists of a tuple $e = \langle v_1, v_2, a \rangle$ where $v_1, v_2 \in V$ and a is a boolean that indicates if the edge is active. For brevity of notation, a_e references that boolean for edge e. The default value of a is `True`, and a needs to be updated whenever the graph is queried as it depends on the positions of the moving obstacles.

A scaffold graph S_i is defined in a manner similar to a PRM graph with a set of edges E_i and vertices V_i. A particular scaffold S_i is associated with a particular obstacle $o_i \in O$ where O denotes the set of all obstacles in the environment. The positions of the vertices are defined relative to the obstacle. A transform T_i is defined that maps points from the obstacle reference frame to the planning frame.

This transform must consist solely of rotation and translation and is defined relative to the center of the obstacle. We define c_i to be the center of o_i

We also define a radius r_i which defines the bounding circle around the obstacle. For the sake of convenience, we refer to the kth point of the jth layer of scaffold i as $v_{i,j,k}$.

Each scaffold consists of a number of layers with the layer closest to the obstacle being defined as layer 1 and the outermost layer being defined as layer N, where N is the number of scaffold layers and and $N \geq 1$. We define a subset of the full scaffold graph $S_{i,j}$ to be the the set of edges and vertices on the jth level of scaffold i. We also define the number of points per scaffold layer to be m.

A scaffold layer $S_{i,j}$ defines a convex polygon enclosing the associated obstacle at a fixed distance $d_{i,j}$. Each point within that layer must be $d_{i,j}$ away from the closest point on obstacle.

Scaffolds From Geometric Primitives

Circle In order to define a scaffold around a circle, we begin by defining the distance from the obstacle to the lowest scaffold layer. As mentioned above, we define the distance between the obstacle and the vertices on the jth layer of the scaffold as $d_{i,j}$. For a circular obstacle, the closest point of a scaffold edge to the obstacle is the scaffold edge's midpoint. Thus, if we set this midpoint to be on the obstacle edge, we can derive an equation for the $d_{i,1}$:

$$d_{i,1} = \frac{2r_i}{(2cos\left(\frac{2\pi}{m}\right) + 2)^{\frac{1}{2}}} - r_i \tag{1}$$

Subsequent layers must have distances larger than $d_{i,1}$. We can then define the kth point of the jth layer as:

$$\begin{aligned} v_{i,j,k} &= \begin{bmatrix} (d_{i,j} + r_i) * cos(\theta_{i,j,k}) \\ (d_{i,j} + r_i) * sin(\theta_{i,j,k}) \end{bmatrix} \\ \theta_{i,j,k} &= (k-1) * \frac{2\pi}{m} + ((j-1) \bmod 2) * \frac{2\pi}{2m} \end{aligned} \tag{2}$$

The second term of the angle computation is used to give each point a slight offset from those on the previous layer. We can then define the set of edges to and from a given point. We use the $\rightleftharpoons$ operator to show that an edge exists. We connect the points as follows:

When j is even:

$$\begin{aligned} v_{i,j,k} \rightleftharpoons \forall v \in \{&v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j-1,k+1}, \\ &v_{i,j+1,k+1}, v_{i,j,k+1}, v_{i,j,k-1}\} \end{aligned} \tag{3}$$

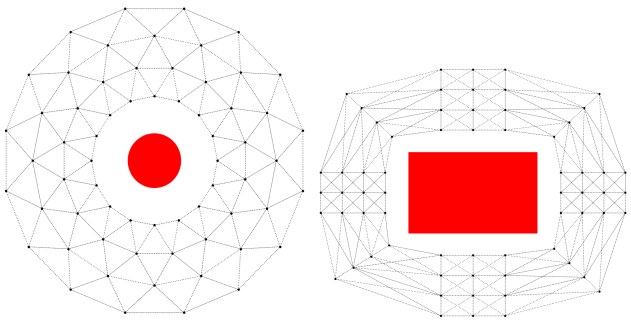
When j is odd:

$$\begin{aligned} v_{i,j,k} \rightleftharpoons \forall v \in \{&v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j,k+1}, \\ &v_{i,j+1,k-1}, v_{i,j,k-1}, v_{i,j-1,k-1}\} \end{aligned} \tag{4}$$

In all cases:

$$v_{i,j,0} = v_{i,j,m}, v_{i,j,m+1} = v_{i,j,1} \tag{5}$$

Edges to vertices where $j = 0$ and $j = N + 1$ are obviously ignored as those vertices do not exist. A completed circle scaffold is shown in Figure 1a. Note the circular obstacle is inflated such that a robot can sit on any of the scaffold points without colliding with the obstacle.



(a) Circular Scaffold (b) Rectangular Scaffold

Figure 1: Example scaffolds for geometric shapes. The number of layers is fixed to be 4 and the number of points per layer is fixed to be 16.

Convex Polygon In order to define a scaffold around a convex polygon, we first define the number of sides to be s. We impose the constraint that the number of points per scaffold layer, m, must be a multiple of s. We define the corner points of the polygon to be $p_1, p_2, \ldots p_s$. The corners of the ith edge are defined to be p_i and p_{i+1} with the corners of edge s being p_s and p_1. We also define the unit normals of the sides and corners with $\hat{n}_i$ representing the normal of side i and $\hat{n}_{p_i}$ representing the normal vector at corner p_i.

In order to space the points evenly throughout the scaffold, we divide the points evenly between the sides of the polygon. There are $m/s - s$ scaffold vertices associated with each side and s vertices associated with the corners. For example, a rectangle with 16 points per layer would have one point on each layer associated with each corner and 3 points per layer associated with each side.

If we associate $v_{i,j,1}$ with p_1, vertices are associated with corner points when $(k-1) \mod (m/s) = 0$ and with sides otherwise where $\mod$ is the modulus operator. In order to define the position of the vertices associated with the corners or side, we first need to derive an expression for determining which side a vertex is associated with based on its index k. We will define the index of the side as κ. The corner points associated with side κ are then p_κ and $p_{\kappa+1}$. The value of κ is defined as follows:

$$\kappa = \lfloor (k-1) * \frac{s}{m} \rfloor \mod s + 1 \tag{6}$$

Where $\lfloor\rfloor$ is the floor operator. The vertices associated with a corner points have their positions defined as follows:

$$v_{i,j,k} = p_\kappa + d_{i,j}\hat{n}_{p_\kappa} \tag{7}$$

Where $d_{i,j}$ is the distance of that layer from the obstacles and the $d_{i,1} = 0$. We also define the position of the points associated with each side as follows:

$$\begin{aligned} \lambda_{i,j,k} &= \frac{k - 1 - m(j-1) - \frac{m}{s}(\kappa - 1)}{m/s} \\ v_{i,j,k} &= \lambda_{i,j,k}||p_{\kappa+1} - p_\kappa|| + p_\kappa + d_{i,j}\hat{n}_j \end{aligned} \tag{8}$$

Where $\lambda_{i,j,k}$ represents the fraction along the line that the vertex covers. Using this notation, we can define set set of edges to and from a given scaffold vertex. We connect each vertex to the adjacent vertices.

$$\begin{aligned} v_{i,j,k} \rightleftharpoons \forall v \in \{&v_{i,j-1,k}, v_{i,j+1,k}, v_{i,j+1,k-1}, \\ &v_{i,j+1,k+1}, v_{i,j,k-1}, v_{i,j,k+1}, v_{i,j-1,k-1}, v_{i,j-1,k+1}\} \end{aligned} \tag{9}$$

As before, in all cases:

$$v_{i,j,0} = v_{i,j,m}, v_{i,j,m+1} = v_{i,j,1} \tag{10}$$

Again edges to vertices where $j = 0$ and $j = N + 1$ are ignored. A completed scaffold for a rectangle is shown in Figure 1b. Note the obstacle is inflated such that a robot can sit on any of the scaffold points without colliding with the obstacle.

Navigation and Planning with Scaffolds

In this section, we examine how to incorporate scaffold graphs into path planning algorithms. As with other PRM approaches designed to be used in dynamic environments, we partition the obstacles into two sets, static obstacles and dynamic obstacles. The static obstacles, those with a fixed location, are the only ones taken into account when generating the initial roadmap. Up until this point, we have only discussed using this approach with PRM. In practice, the scaffolds can be added to any roadmap based graph. This initial graph could be generated using a PRM variant, or it could be generated using a different roadmap approach such as a fixed grid or a Voronoi decomposition. Because the method used to generate the roadmap does not affect the application of the roadmap, we instead focus on defining an algorithm for updating the graph with the scaffold points. In order to define an initial approach to combining the graphs, we first review the primitive functions used in RRT and PRM and we then define a naïve graph update algorithm.

Primitive Functions

The existing PRM and RRT approaches rely on several primitive functions. These are sampling, nearest neighbor, near vertices, steering, and a collision test. These are well described in the literature but we will review the two used in the scaffolding approach.

Near Vertices: The `Near`(G, p, r) method returns the set of vertices that are within a given distance of a particular point. It takes as input a graph G, a point in space p, and a positive real number r. It returns the set of all vertices $v \in V$ that are within r of the specified point p.

Collision Test: The `CollisionFree`(p_1, p_2) method returns a boolean that indicates if the line segment between p_1 and p_2 does not collide with any obstacles.

Graph Update

We separate the graph update into two steps. First, the scaffold points and edges are added to the static graph, then any edges that are blocked are invalidated.

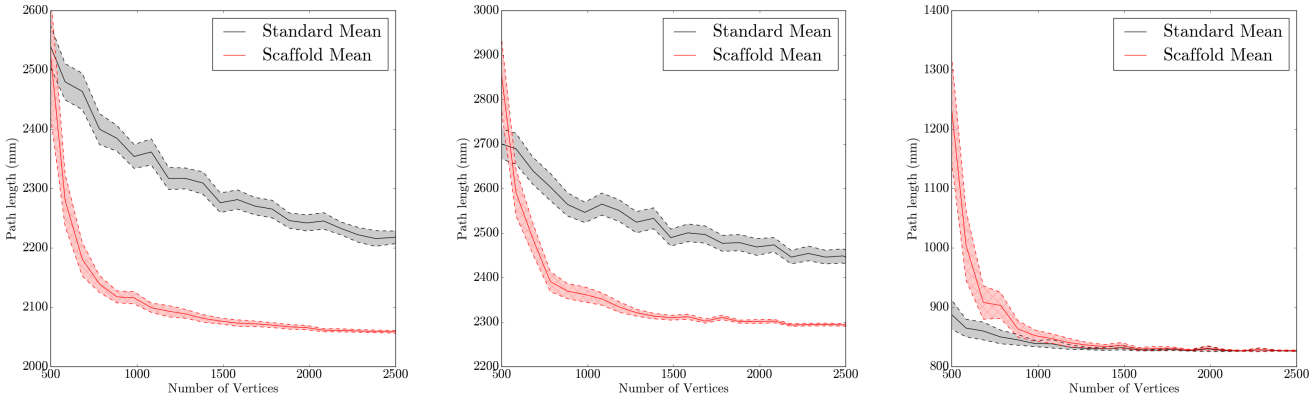
The scaffold points are inserted into the graph in the same way new points are added when generating a roadmap using sPRM. Each point is connected to all of the other points within a certain range of it as long as the line between the two points does not collide with an obstacle. Algorithm 1 presents the procedure for inserting a scaffold into the static graph.

Algorithm 1 SCAFFOLD INSERTION

```
1:  procedure INSERTSCAFFOLD(G,S_i)
2:      Input: Roadmap Graph G and scaffold graphs S_i
3:      Output: Updated graph G that contains the scaffold
        points and edges
4:      for all v ∈ V_i do
5:          neighbors ← Near(G, v, Max Edge Length)
6:          for all v_near ∈ neighbors do
7:              if CollisionFree(v, v_near) then
8:                  Add edge from v to v_near
9:      Append V_i to V
10:     for all e ∈ E_i do
11:         Add e to E
```

Finally, we define the method that invalidates the blocked edges. We perform this invalidation by setting the a indicator for edges that are blocked by dynamic obstacles. As mentioned above, an edge is defined by a tuple $e = \langle v_1, v_2, a \rangle$. We use $v_{1,e}$, $v_{2,e}$ and a_e to refer to the tuple values for a particular edge e. In order to update the graph, we check each edge to see if it collides with an obstacle. This is done by calling the `CollisionFree` primitive for each edge.

Note this approach can be further optimized and is part of our ongoing research.

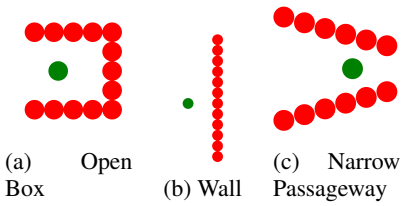


(a) Path Quality, Open Box (b) Path Quality, Wall (c) Path Quality, Narrow Passage

Figure 2: Path quality over the number of vertices in the base graph for the fixed configurations.

Experimental Methodology

In order to evaluate our approach, we compare it against an existing PRM approach, specifically sPRM. We evaluate performance in a number of specific obstacle configurations as well as random field configurations. These configurations consist of a partial box, a narrow opening and a wall. While these configurations are not going to occur during the course of a RoboCup match, they are used to showcase the behavior of the scaffolds in traditionally challenging configurations. In each case, we evaluated each algorithm with randomly generated goal locations. We used a standard size RoboCup SSL field (6m by 9m) and standard sized robots (15cm in diameter) for all of our experiments. Note, robots are considered dynamic obstacles. The static obstacles considered in these experiments are the edges of the field. The fixed configurations are shown in Figure 3.



(a) Open Box (b) Wall (c) Narrow Passageway

Figure 3: Obstacle layouts. The start point is shown in green. The obstacles are shown in red.

In order to directly compare scaffolding and sPRM, we create a base random graph of varying granularity ranging from 100 vertices to 2500 vertices. This graph is used for both approaches. We then add a number of vertices to the sPRM graph such that it has the same number of vertices as the base graph with the scaffolds added. We then compare the mean path lengths for both cases. A particular trial is only included in the mean calculation if solutions are found for both the scaffold PRM and sPRM graphs. For the fixed configurations we ran 100 trials at each level of granularity.

Preliminary Results and Conclusions

Figure 2 shows the path quality over the number of vertices for each configurations. The dotted lines and shaded areas represent 95% confidence intervals. In all three cases, the variance and mean starts significantly higher for the scaffolding approach than for sPRM but decreases as the number of vertices increases. For the open box and wall configurations, the found solutions for graphs with more than 700 vertices are significantly better for the scaffold approach. For the narrow pathway configuration, the scaffolding approach performs essentially the same as the sPRM approach. We believe these results can be attributed to the fact that the optimal path for the open box and wall configurations involves traversing obstacles whereas the optimal path for the narrow passageway that we constructed does not.

In addition to examining the path quality, we tabulated the amount of time it took to update the roadmap and find a path from the start to the goal. Over all of the cases, the update time with the scaffold took an average of 1.72 times as long as without. We are actively working on a more computationally efficient insertion algorithm that we believe will reduce the graph update time. In addition. there are a number of inefficiencies in our implementation and optimizations that we can make to further reduce the difference between update times.

We believe these preliminary results demonstrate that the scaffold graph approach provides an advantage over simply using a PRM graph in dynamic environments with known dynamic obstacle shapes. If the update time can be reduced, the scaffold approach presented here should provide benefits in RoboCup SSL and other similar domains.

In the future, we would like to extend the scaffold definition to additional obstacle shapes. In particular, we want to use the distance transform to define scaffolds around arbitrary shapes. We would also like to extend the notion of scaffolds to higher dimensional spaces as we believe this approach could assist with manipulation in cluttered environments and with multi-agent path planning in two dimen-

sional spaces.

References

Bruce, J., and Veloso, M. 2002. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, 2383–2388. IEEE.

Emmerink, T.; van den Berg, R.; Overeem, J.; Hakkers, T.; de Jong, J.; van Ommeren, J.; and Meulenkamp, R. 2015. Ssh team description paper for robocup 2015.

Jaillet, L., and Siméon, T. 2004. A prm-based motion planner for dynamically changing environments. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, 1606–1611. IEEE.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7):846–894.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4):566–580.

Kavraki, L. E.; Kolountzakis, M. N.; and Latombe, J.-C. 1998. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14(1):166–171.

Leven, P., and Hutchinson, S. 2002. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research* 21(12):999–1030.

van den Berg, J. P.; Nieuwenhuisen, D.; Jaillet, L.; and Overmars, M. H. 2005. Creating robust roadmaps for motion planning in changing environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, 1053–1059. IEEE.

Human-in-the-Loop SLAM

Samer Nashed and Joydeep Biswas
University of Massachusetts, Amherst
140 Governors Drive
Amherst, Massachusetts 01003

Abstract

Building large-scale, globally consistent maps is a challenging problem, made more difficult in environments with limited access, sparse features, or when using data collected by novice users. For such scenarios, where state-of-the-art mapping algorithms produce globally inconsistent maps and require additional data collection, we introduce a systematic approach to incorporating sparse human corrections, which we term Human-in-the-Loop Simultaneous Localization and Mapping (HitL-SLAM). Given an initial factor graph for pose graph SLAM, HitL-SLAM accepts approximate, potentially erroneous, and rank-deficient human input; infers the intended correction via expectation maximization (EM); back-propagates the extracted corrections over the pose graph; and finally jointly optimizes the factor graph including the human inputs as human factor terms, to yield globally consistent large-scale maps. We thus contribute an EM formulation for inferring potentially rank-deficient human corrections to mapping, and human factor extensions to the factor graphs for pose graph SLAM that result in a principled approach to joint optimization of the pose graph while simultaneously accounting for multiple forms of human correction. We present empirical results showing the effectiveness of HitL-SLAM at generating globally accurate and consistent maps even when given poor initial estimates of the map.

1 Introduction

Accurate metric mapping of environments is essential for autonomous mobile robot function. For a variety of reasons, including operator inexperience, sensor limitations, large or time-sensitive environments, and algorithmic limitations, generating accurate metric maps from data collected over a single deployment may require more resources than are available. Furthermore, it may be impossible or prohibitively expensive to improve the map with data collected by redeploying the robot. In these instances, we propose a general human-in-the-loop framework, along with a specific algorithm, which dramatically improves map accuracy while requiring limited additional compute resources and imposing minimal constraints on the human.

The goal of Human-in-the-Loop SLAM (HitL-SLAM) is to leverage a human's ability to outperform state-of-the-art algorithms in the data association problem, in order to construct more accurate maps. Moreover, humans often have meta-knowledge about an environment that presently far exceeds most robot learning capabilities. Allowing humans to endow robots with this knowledge can greatly increase robot capabilities and is necessary in certain cases where resources are constrained. Additionally, the information sharing between robot and human necessitates a method for gathering and evaluting human input which is robust to human error and requires minimal user effort. Fig. 1 presents an example of HitL-SLAM in practice.

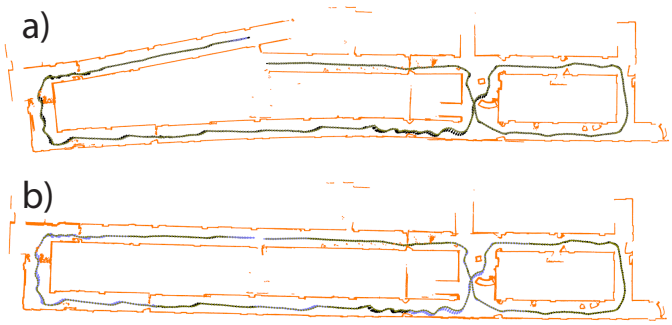


Figure 1: Before and after maps of a floor in our department. Observations are shown in orange, and poses are shown as black arrows. a) initial map with no loop closure and no human constraints. b) the final map produced by HitL-SLAM. Poses which are involved in one or more human constraints are shown in blue.

In this paper we present three contributions. First, a principled framework for dealing with human input motivated by both functional and theoretical considerations, which allows the human to efficiently convey their knowledge. Second, an algorithm for constructing human factors and performing joint factor graph optimization, which we use to solve HitL-SLAM. And third, a 2-stage SLAM backend, which uses a combination of analytical and numerical techniques to produce globally consistent maps with minimal distortion in the presence of rank deficient contraints.

HitL-SLAM is an iterative process, wherein the human and robot alternate proposing the most likely map. The robot operates on poses in a factor graph and displays the resulting map, while the human operates on the observations and those operations are translated to factors in the factor graph. The key idea is that humans impose constraints more natu-

rally by relating observations instead of the underlying poses since they are more likely to be knowledgeable about the structure of the observations rather than the poses. Our proposed approach thus consists of the following steps:

1. *Extract Likely Input:* Use EM (Dempster, Laird, and Rubin 1977) to decide the most likely parameters of human input (Section 4).
2. *Establish Initial Estimate:* Use COP-SLAM (Dubbelman and Browning 2015) to find a piece-wise optimal initial estimate. (Section 5a)
3. *Solve Optimization Problem:* Construct human factors and jointly solve a non-linear least-squares optimization problem. (Section 5b)

To the best of our knowledge, we present the first algorithm and framework for incorporating human input into the optimization of a pose-graph. Our approach is evaluated using a series of tests designed to measure the accuracy and scalability of HitL-SLAM. We find that HitL-SLAM is able to produce highly accurate maps in a variety of scenarios, and requires less time than it would take to re-deploy the robot over the same map.

2 Related Work

Solutions to robotic mapping and SLAM have improved dramatically in recent years, but a significant gap between state-of-the-art fully autonomous systems and the requirements for robust, large-scale deployment still exists, due in part to the difficulty of the data association problem (Dissanayake et al. 2011; Bailey and Durrant-Whyte 2006; Aulinas et al. 2008; Thrun and others 2002). The notion that humans and robots can or should collaborate in map building is not new, and has given birth to a field known as Human-Augmented Mapping (HAM).

Most work within HAM can be thought of as belonging to one of two groups, depending on whether the human and robot are conjunctive in time and space during exploration (C-HAM), or whether the human enters the loop remotely or after the fact (R-HAM). Many C-HAM techniques exist to address semantic (Nieto-Granda et al. 2010; Christensen and Topp 2010) and topological (Topp and Christensen 2006) mapping. A number of approaches have also been proposed for integrating semantic and topological information, along with human trackers (Milella et al. 2007) and interaction models (Topp et al. 2006), into *conceptual spatial maps* (Zender et al. 2007), which are typically organized in a hierarchical manner.

There are two major weaknesses implicit in these C-HAM approaches. First, a human must be present with the robot during exploration. This places physical constraints on the type of environments which can be mapped, as they must be accessible and traversable by a human. Second, these methods are inefficient with respect to the human's attention, since most of the time the human's presence is not critical to the robot's function, for instance during navigation between waypoints. These approaches, which focus mostly on semantic and topological mapping, also typically assume that the robot is able to construct a nearly perfect metric map entirely autonomously. While this is reasonable for small environments, metric mapping of large, dynamic spaces is still an open research question.

In contrast, most of the effort in R-HAM has been concentrated on either incorporating humans into the loop during deployment in order to teleoperate and gain situational awareness such as in the Urban Search and Rescue (USAR) problem (Murphy 2004; Nourbakhsh et al. 2005), or to do high level decision making such as goal assignment or coordination of multiple agents (Olson et al. 2013; Parasuraman et al. 2007; Doroodgar et al. 2010). Some R-HAM techniques for metric mapping and pose estimation have also been explored, but these involve either having the robot retrace its steps to fill in parts missed by the human (Kim et al. 2009) or by having additional agents and sensors in the environment (Kleiner, Dornhege, and Dali 2007), neither of which is efficient.

Ideally, a robot could explore an area only once with no need for human guidance or input during deployment, and later with minimal effort, a human could make any corrections necessary to achieve a near-perfect metric map. This is precisely what HitL-SLAM does, and it is worth noting that the HitL-SLAM framework and algorithm place no restrictions on the temporal or spatial conjunctivity of the human and robot.

3 Human-in-the-Loop SLAM

HitL-SLAM operates on a factor graph G defined as a set of poses X representing estimated poses along the robot's trajectory, and a set of factors $F = \{R, H\}$, which encode information about both relative pose constraints arising from odometry and observations, R, and constraints supplied by the human, denoted H. In this report, the initial factor graph G_0 provided by our SLAM front-end, Episodic non-Markov Localization (EnML) (Biswas and Veloso 2014) , does not contain any loop closure information beyond the length of the episode specified by EnML. However, HitL-SLAM is capable of handling constraints in G_0 with or without loop closure.

HitL-SLAM runs iteratively, with the human specifying constraints on observations in the map, and the robot then enforcing those constraints along with all previous constraints to produce a revised estimate of the map. Since humans often enter only approximately correct input given a desired effect, some interpretation of the human input is necessary before human constraints can be computed and added to the factor graph. Formally, we say the robot first proposes an initial graph $G_i = \{X_i, F_i\}$, the human then supplies a set of correction factors H_i, and finally the robot re-optimizes the poses in the factor graph, producing $G'_i = \{X'_i, F_i \cup H_i\}$.

Each iteration of HitL-SLAM proceeds in two steps, shown in Fig. 2. First, the human input is gathered, interpreted, and a set of human correction factors are instantiated. Second, a combination of analytical and numerical techniques is used to jointly optimize the factor graph using the human correction factors and the relative pose factors, producing the final map.

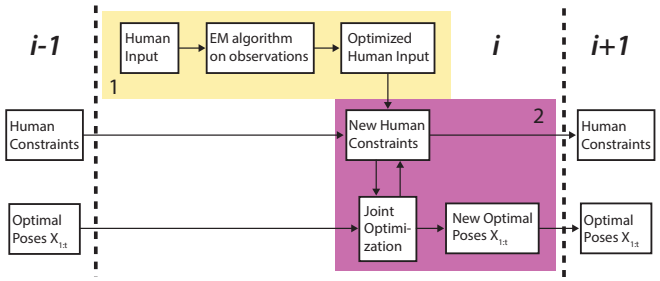


Figure 2: Flow of information during processing of the i^{th} human input. Block 1 (yellow) outlines the evaluation of human input, and block 2 (purple) outlines the factor graph construction and optimization processes. Note that the joint optimization process optimizes *both* pose parameters *and* human constraint parameters.

When interpreting human input, we define human correction factors $h \in H$ as tuples $h = \{P_a, P_b, S_a, S_b, X_a, X_b, m\}$, where P_a and P_b are sets of points in $\mathbb{R}^N$ (for N-dimensional HitL-SLAM) specified by the human. Sets S_a and S_b are subsets of all observations S and contain observations which are near the features defined by P_a and P_b, respectively. X_a and X_b are subsets of the poses $X_{1:t}$, where a given pose $x \in X_a$ if there are observations in S_a which originate from x. X_b is defined analogously. $m \in M$ defines the geometry of the constraints the human is enforcing over the observations, such as colocation, colinearity, perpendicularity, *etc*.

We frame the problem of interpreting human input as finding the observation subsets S_a, S_b and human input sets P_a, P_b which maximize the joint correction input likelihood, $p(S_a, S_b, P_a, P_b | P_a^0, P_b^0, m)$, which is the likelihood of selecting observation sets S_a, S_b and point sets P_a, P_b, given initial human input P_a^0, P_b^0 and correction mode m. To find S_a, S_b and P_a, P_b we use the sets P_a^0, P_b^0 and observations in a neighborhood around P_a^0, P_b^0 as initial estimates in an Expectation Maximization approach, detailed in Section 4. As the pose parameters are adjusted during joint optimization in later iterations of HitL-SLAM, the locations of points in P_a, P_b may change, but once an observation is established as a member of S_a or S_b its status is not changed.

Once P_a, P_b and S_a, S_b are determined for a new constraint, then given m we can find the set of poses $X_{1:t}^*$ which best satisfy all given constraints. We first compute an initial estimate $X_{1:t}^0$ by analytic back-propagation of the most recent human correction factor, considering sequential constraints in the pose-graph. Next, we construct and solve a joint optimization problem over the relative pose factors f and the human correction factors h. This amounts to finding the set of poses $X_{1:t}^*$ which minimize the sum of the cost of all factors,

$$X_{1:t}^* = \text{argmin}_{X_{1:t}} \left[\sum_{i=1}^{|R|} c_r(r_i) + \sum_{j=1}^{|H|} c_m(h_j) \right], \qquad (1)$$

where $c_r : R \to \mathbb{R}$ computes the cost from relative pose-graph factor r_i, and $c_m : H_m \to \mathbb{R}$ computes the cost from human correction factor h_j with correction mode m. Section 5 covers the construction of the human correction factors and the formulation of the optimization problem.

4 Evaluating Human Input

Human Input Interface

Although the specifics of how P_a, P_b, and m are input are not relevant mathematically, we outline the input interface here for concreteness and clarity going forward. HitL-SLAM users provide input by first entering the 'provide correction' state by holding down a set of keys (SHIFT + CTRL) and left clicking. Once in the 'provide correction' state, they enter points for P_a, P_b by clicking and dragging along the feature (line segment) they wish to specify. The mode m is determined by which key is held down during the click and drag. For instance, CTRL alone maps to colocation, while SHIFT alone maps to colinear. To finalize their entry, the user exits the 'provide correction' state with the same command they used to enter it. Exit from the 'provide correction' state triggers the algorithm in full, and the user must wait until a revised version of the map is presented in order to specify additional corrections.

Human Input Evaluation

Because humans do not have perfect control of a mouse or touch screen, what the human actually enters and what they intend to enter may differ slightly. We formulate the problem of evaluating human input as finding the sets S_a, S_b and P_a, P_b which the human most likely meant to identify (S_a, S_b) and provide (P_a, P_b), given the observations S and the initial human input P_a^0 and P_b^0. To do this we use the EM algorithm, which maximizes the log likelihood

$$\sum_i \sum_{z_i} p(z_i | s_i, \theta^{\text{old}}) \log(p(z_i, s_i | \theta)), \qquad (2)$$

where the parameters $\theta = \{P_a, P_b\}$ are the human input, the $s_i \in S$ are the observations, and the latent variables z_i are indicator variables denoting the inclusion or exclusion of s_i from S_a or S_b. The expressions for $p(z_i | s_i, \theta^{\text{old}})$ and $p(z_i, s_i | \theta)$ come from a generative model of human error based on the normal distribution,

$$p(z_i | s_i, \theta) = \frac{1}{\sqrt{2\sigma^2 \pi}} \exp\Big(- \frac{(s_i - f(s_i, \theta))^2}{2\sigma^2} \Big). \qquad (3)$$

Here, σ is the standard deviation of the human's accuracy when specifying points with the mouse, and $f(s_i, \theta)$ is a distance function which evaluates the distance between a given observation s_i and the feature parameterized by θ. Since all modes m in this study can be specified by one or more line segments and $p(z_i | s_i, \theta)$ is convex, then if we define $f(\theta)$ as the distance between the specified line segment and the given observation s_i, the EM problem reduces to iterative least-squares fitting over a changing subset of S which occur near the latest estimates for P_a, P_b.

Once the initial P_a, P_b have been determined, along with observations S_a, S_b, we can find the poses responsible for

those observations X_a, X_b, thus fully defining the human correction factor h. To make this process more robust, a given pose is only allowed in X_a or X_b if there exist a minimum of T_p elements in S_a or S_b corresponding to that pose, where T_p is a threshold related to observation density.

5 Solving HitL-SLAM

Initializing Human Corrections

HitL-SLAM allows three types of constraints. More complicated data associations may be constructed from a combination of the following:

1. *Colocation:* A full rank constraint specifying that two sets of observations are colocated.
2. *Colinearity:* A rank deficient constraint specifying alignment of two sets of observations along a single translation direction, as well as orientation.
3. *Co-orientation:* A rank deficient constraint specifying only orientation relationships.

Although the user may select sets of observations in any order, we define P_a to be the input which selects observations S_a arising from poses X_a such that $\forall x_i \in X_a$ and $x_j \in X_b$, $i < j$, where X_b is defined analogously by observations S_b specified by input P_b. That is, all poses $x_i \in X_a$ occur before all poses $x_j \in X_b$.

Given P_a and P_b, we find the affine transformation A which transforms the constellation P_b to the correct location relative to constellation P_a, as specified by mode m. If the correction mode is rank deficient, we force the motion of the observations as a whole to be zero along the null space dimensions. For co-orientation, this means that the translation components of A are zero, and for colinearity the translation along the axis of colinearity is zero. To be clear, this does not mean that any given point $p_i \in P_b$ will not experience motion along one or more null dimensions. Only the motion of the center of mass of the constellation is constrained in this way. Fig. 3 shows the effect of applying different types of constraints to a set of point clouds. Note that the rank deficient constraints do not produce motion of the point cloud as a whole along the null space dimensions.

After finding A we then consider the poses in X_b to constitute points on a rigid body, and transform that body by A. The poses x_k such that $\forall x_j \in X_b$, $k > j$, are treated similarly, such that the relative transformations between all poses occurring during or after X_b remain unchanged.

If $X_a \cup X_b$ does not form a contiguous sequence of poses, then this explicit change creates at least one discontinuity between the earliest pose in X_b, x_0^b and its predecessor, x_c. We define affine transformation C such that $x_0^b = A_{cb} C x_c$, where A_{cb} was the original relative transformation between x_c and x_0^b. Given C, and the pose and covariance estimates for poses between X_a and X_b, we use COP-SLAM over these intermediate poses to transform x_c without inducing further discontinuities.

The idea behind COP-SLAM is essentially a covariance-aware distribution of translation and rotation across many poses, such that the final pose in the pose-chain ends up at the correct location and orientation. The goal is to find a set of updates U to the relative transformations between poses in the pose-chain such that $C = \prod_{i=1}^{n} U_i$. An approximation could of course be to take the n^{th} root of C, but COP-SLAM achieves piece-wise optimality with respect to transformation magnitude normalized by covariance, by weighting the magnitude of the update rotations and translations according to the inverse covariance.

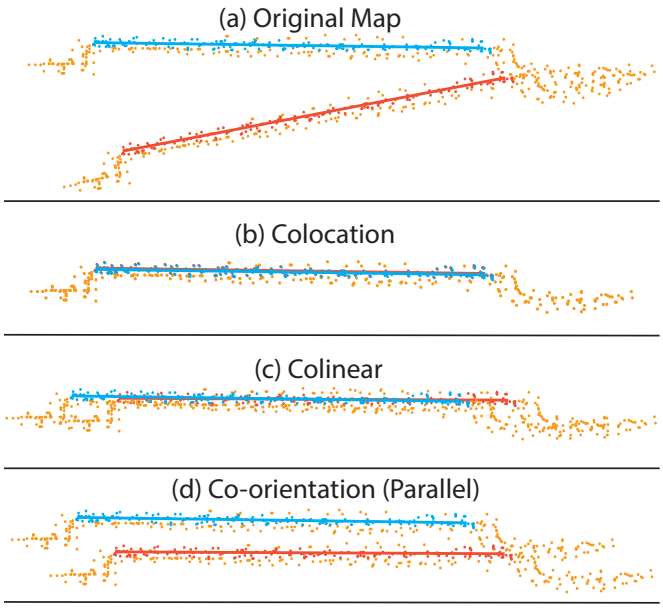


Figure 3: Result of transforming observation point clouds based on different human constraints, showing (a) Original map, (b) Colocation constraint, (c) Colinear constraint, (d) Co-orientation constraint. In all subfigures P_a and P_b are given by the red and blue lines, respecively. S_a and S_b are shown as the red and blue point clouds. $S \setminus (S_a \cup S_b)$ appear in orange.

COP-SLAM has two primary weaknesses. First, it requires uncertainty estimates to be isotropic, which is not true in general. Second, COP-SLAM deals poorly with nested loops, where it initially produces good pose estimates but during later adjustments may produce inconsistencies between observations, since COP-SLAM is not able to simultaneously satisfy both current and previous constraints. Because of these issues, we use COP-SLAM as an initial estimate to a non-linear optimization problem, which produces a more robust, globally consistent map.

HitL-SLAM Optimization

Without loop closure, a pose-chain of N poses has $O(N)$ factors. With most loop closure schemes, each loop can be closed by adding one additional factor per loop. In HitL-SLAM, the data provided by the human is richer than most front-end systems, and reflecting this in the factor graph could potentially lead to a prohibitively large number of factors. If $|X_a| = n$ and $|X_b| = m$, then a naive algorithm which adds a factor between every pair (x_i, x_j), where $x_i \in X_a$ and $x_j \in X_b$, would add mn factors *every loop*.

This is a poor approach for two reasons. One, the large number of factors can slow down the optimizer and potentially prevent it from reaching the global optimum. And two, this formulation implies that every factor is independent of every other factor, which is incorrect.

Thus, we propose a method for reasoning about human correction factors jointly, in a manner which creates a constant number of factors per loop while also preserving the structure and information of the input. Given a human correction factor $h = \{P_a, P_b, S_a, S_b, X_a, X_b, m\}$, we define c_m as the sum of three residuals, R_a, R_b, and R_p. The definitions of R_a and R_b are the same regardless of m:

$$R_a = \left(\frac{\sum_{i=1}^{|S_a|} \delta(s_i^a, P_a)}{|S_a|}\right)^{\frac{1}{2}}, \quad R_b = \left(\frac{\sum_{i=1}^{|S_b|} \delta(s_i^b, P_b)}{|S_b|}\right)^{\frac{1}{2}}. \tag{4}$$

Here, $\delta(s, P)$ denotes the squared Euclidean distance from observation s to the closest point on the feature defined by the set of points P. All features used in this study are line segments, but depending on m, more complicated features with different definitions for $\delta(s, P)$ may be used. R_a implicitly enforces the interdependence of different $x_a \in X_a$, since moving a pose away from its desired relative location to other poses in X_a will incur cost due to misaligned observations. The effect on X_b by R_b is analogous.

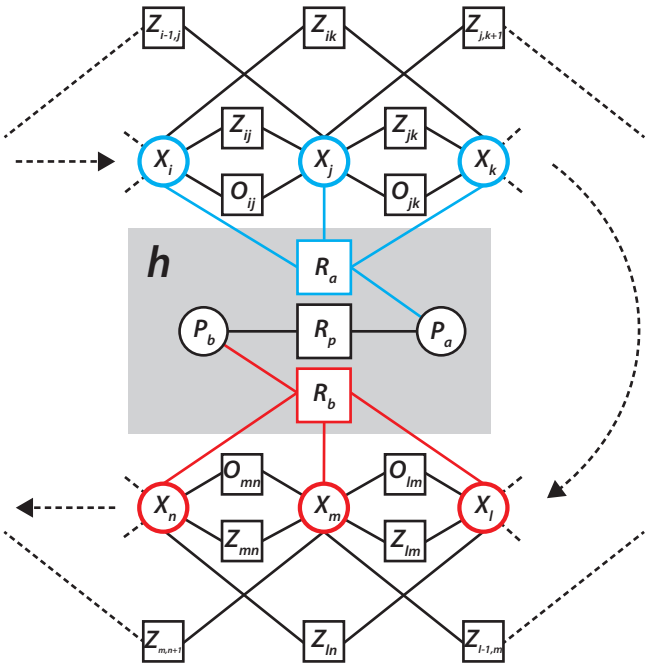


Figure 4: Subset of a factor graph containing a human factor h. Factors R_a and R_b drive observations in S_a and S_b toward features P_a and P_b, respectively. Factor R_p enforces the geometric relationship between P_a and P_b. Note that parameters in X_a (blue poses) and X_b (red poses) as well as P_a and P_b are jointly optimized.

The relative constraints between poses in X_a and poses in X_b are enforced indirectly by the third residual, R_p. Depending on the mode, colocation ($+$), colinearity ($-$), co-orientation parallel ($\|$), co-orientation perpendicular ($\perp$), the definition changes:

$$\begin{aligned} R_3^+ &= K_1||cm_b - cm_a|| + K_2(1 - (\hat{n}_a \cdot \hat{n}_b)) \\ R_3^- &= K_1||(cm_b - cm_a) \cdot \hat{n}_a|| + K_2(1 - (\hat{n}_a \cdot \hat{n}_b)) \\ R_3^{\|} &= K_2(1 - (\hat{n}_a \cdot \hat{n}_b)) \\ R_3^{\perp} &= K_2(\hat{n}_a \cdot \hat{n}_b) \end{aligned} \tag{5}$$

Here, cm_a and cm_b are the centers of mass of P_a and P_b, respectively, and $\hat{n}_a$ and $\hat{n}_b$ are the unit normal vectors for the feature (line) defined by P_a and P_b, respectively. K_1 and K_2 are constants that determine the relative costs of translational error (K_1) and rotational error (K_2). The various forms of R_p all drive the constellation P_b to the correct location and orientation relative to P_a. During optimization the solver is allowed to vary pose locations and orientations, and by doing so the associated observation locations, as well as points in P_a and P_b. Fig. 4 illustrates the topology of the human correction factors in our factor graph.

6 Results

Evaluation of our method is carried out through two experiments. First, we construct a data set by limiting the range of our robot's laser so that it sees a wall only when very close. We then drive it around a room for which we have ground truth. This creates sequences of "lost" poses throughout the pose-chain which have to rely purely on odometry to localize, thus accruing error over time. We then impose human constraints on the resultant map and compare to ground truth, shown in Fig. 5. HitL-SLAM finds a room width of 6.31m, while our hand measurement gives a width of 6.33m. Additionally, HitL-SLAM produces opposite walls which are within 1° of parallel. Note that due to the limited sensor range, at no point are both walls simultaneously visible to the robot. Thus, correctness must come from proper application of human constraints to the "lost" poses between wall observations.

For larger maps where exact ground truth was not available, we introduce a different metric for evaluation. We define the inconsistency $I_{i,j}$ to be the area (or volume in 3D) which observations from pose x_i show as free space and observations from pose x_j show as occupied space. Although there are cases where inconsistency is not a useful metric, for instance a map of a long hallway without loop closure may be consistent even if it is severely bent and does not correspond well to ground truth, it is often what humans look at when evaluating a map and allows us to monitor the progress of HitL-SLAM over multiple iterations. On average, the inconsistency of the final map was reduced to 9% of its initial value. This makes a compelling argument for the use of inconsistency as a metric for evaluating maps, as well as a tool to guide robots and humans when deciding what constraints to impose on a given map to increase accuracy. Fig. 6 offers some qualitative examples of our algorithm's performance.

All maps shown in Fig. 6 were completed by the human in under 15 mins, well within the time it would take to redeploy the robot and then run our localization algorithm on

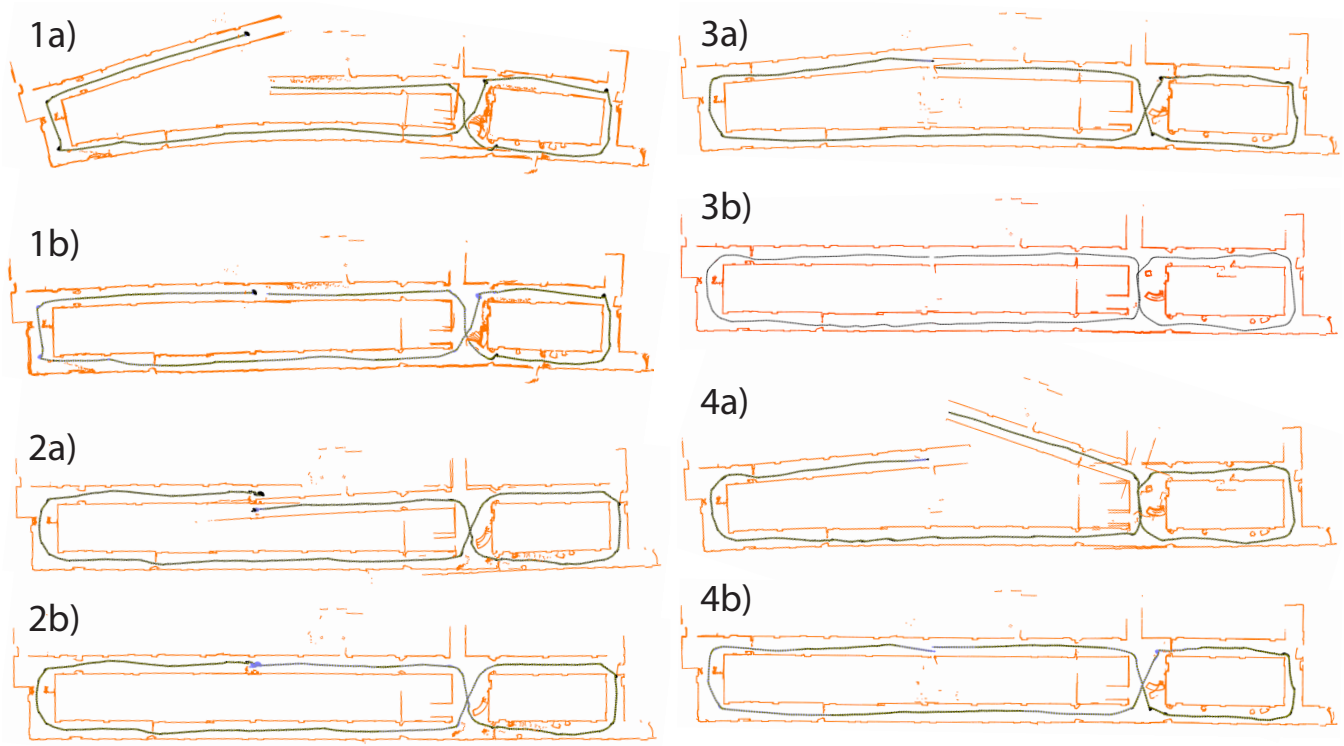


Figure 6: Initial and final maps from HitL-SLAM. Each map is of the same floor, and consist of between 600 and 700 poses. All maps labeled a) are the initial maps, and all those labeled b) are the final maps. Observations are shown in orange and poses are shown as arrows. Poses which are part of a human constraint are blue, while those which are not are in black. Note the varying degree of degradation in each map; some took only a couple human inputs, while others required closer to 10. Our factor graph formulation allows the optimizer to reach minima in roughly 30-40 seconds, even when there are hundreds of poses involved in human constraints.

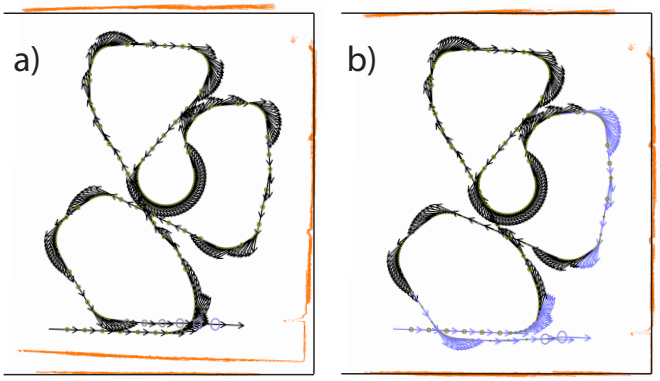


Figure 5: Lost poses experiment. Observations are shown in orange, poses are black arrows, and ground truth (walls) is represented by the black lines. Poses involved in human constraints are colored blue. The initial map is shown in a), and the final map is shown in b).

both datasets to generate a new map from both deployments. Furthermore, this data set contains 2 features which may not be solved by re-deployment. One is a severely bent hallway, in subfigure 1a), and the other is a sensor failure, in subfigure 4a) which caused the robot to incorrectly estimate its heading by roughly 30 degrees between two poses about a fifth of the way into its deployment. Combined, these results show that incorporating human input into metric mapping can be done in a principled, computationally tractable manner, which allows us to solve some types of metric mapping problems in less time and with higher accuracy than previously possible, at the expense of a small amount of human input.

7 Conclusion

We present Human-in-the-Loop SLAM (HitL-SLAM), an algorithm designed to leverage human ability and meta-knowledge as they relate to the data association problem for robotic mapping. HitL-SLAM contributes a generalized framework for interpreting human input using the EM algorithm, as well as a factor graph based algorithm for incorporating human input into pose-graph SLAM. Future work in this area should proceed towards further minimizing the human requirements, and extending this method for higher dimensional SLAM and for different sensor types.

References

Aulinas, J.; Petillot, Y. R.; Salvi, J.; and Lladó, X. 2008. The slam problem: a survey. In *CCIA*, 363–371. Citeseer.

Bailey, T., and Durrant-Whyte, H. 2006. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine* 13(3):108–117.

Biswas, J., and Veloso, M. 2014. Episodic non-markov localization: Reasoning about short-term and long-term features. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 3969–3974. IEEE.

Christensen, H. I., and Topp, E. A. 2010. Detecting region transitions for human-augmented mapping.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39(1):1–38.

Dissanayake, G.; Huang, S.; Wang, Z.; and Ranasinghe, R. 2011. A review of recent developments in simultaneous localization and mapping. In *2011 6th International Conference on Industrial and Information Systems*, 477–482. IEEE.

Doroodgar, B.; Ficocelli, M.; Mobedi, B.; and Nejat, G. 2010. The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2858–2863. IEEE.

Dubbelman, G., and Browning, B. 2015. Cop-slam: Closed-form online pose-chain optimization for visual slam. *IEEE Transactions on Robotics* 31(5):1194–1213.

Kim, S.; Cheong, H.; Park, J.-H.; and Park, S.-K. 2009. Human augmented mapping for indoor environments using a stereo camera. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5609–5614. IEEE.

Kleiner, A.; Dornhege, C.; and Dali, S. 2007. Mapping disaster areas jointly: Rfid-coordinated slam by hurnans and robots. In *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, 1–6. IEEE.

Milella, A.; Dimiccoli, C.; Cicirelli, G.; and Distante, A. 2007. Laser-based people-following for human-augmented mapping of indoor environments. In *Artificial Intelligence and Applications*, 169–175.

Murphy, R. R. 2004. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 34(2):138–153.

Nieto-Granda, C.; Rogers, J. G.; Trevor, A. J.; and Christensen, H. I. 2010. Semantic map partitioning in indoor environments using regional analysis. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1451–1456. IEEE.

Nourbakhsh, I. R.; Sycara, K.; Koes, M.; Yong, M.; Lewis, M.; and Burion, S. 2005. Human-robot teaming for search and rescue. *IEEE Pervasive Computing* 4(1):72–79.

Olson, E.; Strom, J.; Goeddel, R.; Morton, R.; Ranganathan, P.; and Richardson, A. 2013. Exploration and mapping with autonomous robot teams. *Communications of the ACM* 56(3):62–70.

Parasuraman, R.; Barnes, M.; Cosenzo, K.; and Mulgund, S. 2007. Adaptive automation for human-robot teaming in future command and control systems. Technical report, DTIC Document.

Thrun, S., et al. 2002. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium* 1:1–35.

Topp, E. A., and Christensen, H. I. 2006. Topological modelling for human augmented mapping. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2257–2263. IEEE.

Topp, E. A.; Huettenrauch, H.; Christensen, H. I.; and Eklundh, K. S. 2006. Bringing together human and robotic environment representations-a pilot study. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4946–4952. IEEE.

Zender, H.; Jensfelt, P.; Mozos, O. M.; Kruijff, G.-J. M.; and Burgard, W. 2007. An integrated robotic system for spatial understanding and situated interaction in indoor environments. In *AAAI*, volume 7, 1584–1589.

Towards CLIPS-based Task Execution and Monitoring with SMT-based Planning and Optimization

Tim Niemueller
Gerhard Lakemeyer
Knowledge-Based Systems Group
RWTH Aachen University

Francesco Leofante
Erika Ábrahám
Theory of Hybrid Systems
RWTH Aachen University

Abstract

In robotics, automated task planning is still the exception rather than the norm. While generating a plan certainly represents a crucial aspect, another often neglected one, in particular relevant to integration in robotics, is the task-level executive. It forms the coherent behavior out of the determined plan to achieve the mission goals. This work presents first results towards the integration of a knowledge-based plan executive. This work is embedded in a project to generate guaranteed-quality plans based on Satisfiability Modulo Theory (SMT) in industrial scenarios. The initial prototype is integrated based on the scenario of logistics robots in simulation.

1 Introduction

The goal of automated task planning is often considered completed as soon as a plan has been generated. That notion is carried, for example, by the International Planning Competition. While this is certainly a crucial aspect, acting according to the plan is another. Certainly so when it comes to robotics scenarios where the goal is to facilitate a certain function in a real (or simulated) environment, achieving goals according to a plan and monitoring that execution are equally important. Furthermore, the integration of planning systems with the many other components that compose a robotic system often proves to be challenging.

In this paper, we present two early developments in our current research: first, we describe how to use a Satisfiability Modulo Theory (SMT) solver with optimization as a plan generator. Doing so, we intend to explore the possibilities and limits leveraging the more expressive power of SMT to solve a real-world scenario. State-of-the-art solvers offer more features than just solving constraints expressed in expressive theories. For instance, some solvers allow incremental solving where constraints about task feasibility can be updated; others instead offer optimization features allowing to state and solve (multiple) optimization problems (it should be noted that SMT solvers can handle problems which cannot be handled by traditional linear optimization tools, as Boolean combinations of constraints can be present). Therefore, a crucial point is to *make use of these optimization capabilities* to not only synthesize feasible plans, but guaranteed-quality solutions (i.e., optimal) for plans involving arbitrary combinations of theories. Second, we build on our existing incremental task-level reasoning system (Niemueller, Lakemeyer, and Ferrein 2013) based on the CLIPS rule-based production system and extend it into a knowledge-based plan executive that supports concurrency, execution monitoring and (limited domain-specific) contingency mitigation. It has been used before to model the full scenario, from world modeling, decision making, and execution. However, the previous system required an extensive modeling covering many situations and it always made the decision for just the next action. No look-ahead strategy was employed to plan for cooperation and anticipate the optimal use of multiple robots on a single task.

The general evaluation scenario is the Planning Competition for Logistics Robots in Simulation (Niemueller et al. 2016a) which is based on the RoboCup Logistics League (RCLL). For this first prototype, we focus on the Exploration Phase, where robots need to identify the positions of machines in the environment in a limited time. The task is then to efficiently distribute the overall task to the group of three robots. While this phase is not part of the simulation competition, it provides a sufficient testbed to demonstrate the overall prototype integration.

This work is embedded into a joint project[1] that strives to explore possibilities to employ SMT for optimizing the logistics of autonomous robots in a smart factory. We build on previous work of both involved research groups to create an integrated system. We model the RCLL natively in SMT in order to explore different encoding strategies in terms of solving efficiency and to leverage its optimization capabilities. Building on and extending our proven CLIPS-based executive allows to off-load some crucial run-time aspects like monitoring and failure recovery, and to reason about subproblems and goals to achieve to reduce the computation requirements of the SMT solving process. The executive is also used in other projects with different planners.

In Section 2 we describe some general aspects concerning plan execution and present the evaluation scenario in Section 3. In Section 4 we describe the SMT-based planning approach before explaining our CLIPS-based executive prototype in Section 5. After describing some initial results in Section 6 we conclude in Section 7.

[1] Optimizing the Performance of Robot Fleets in Production Logistics Scenarios Using SMT Solving: `https://ths.rwth-aachen.de/research/projects/smt4robots/`

2 Plan Execution and Related Work

Following Verma et al. (2005b), a *plan* is specified as a series of actions designed to accomplish a set of goals but not violate any resource limitations, temporal or state constraints, or other operation rules. Desirable characteristics of a plan are that it be valid, complete and optimal (or of high quality). Algorithms that can reason about achieving goals over a future time period and in the face of various constraints are called *planners*. As the authors pointed out, a plan usually needs further specification and a system to execute the plan. An *executive* is then a software component that realizes such plans.

While planning certainly is combinatorially and computationally the more complex problem, execution of plans is a very intricate one. It tends to be more platform- and environment-specific than the planning part because the models are often more detailed. For the planning model, the tendency is to strip details to increase planning efficiency or because it is irrelevant for generating the task pattern achieving the goal. Particular challenges may also be slack during execution, or uncertainty, e.g., in travel times due to other agents in the environment. In the multi-robot case, issues of synchronization and mutual exclusion may be relevant.

While there is the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) as a (somewhat) unified planning language, there is no such generally accepted language for execution. Planners do not even output in a common output format. Even if using PDDL as a common input language, and yielding PDDL actions to execute, there is most often additional non-uniform information along with the plan that requires special processing, such as planning times or metrics output.

Several execution systems have been proposed in the past. Most executives mentioned in (Verma et al. 2005b) are associated with a specific modeling language. For example, the Universal Executive (Verma et al. 2006) is a general processor for the PLEXIL (Verma et al. 2005a) language. It allows to describe the execution flow as a number of hierarchically structured nodes consisting of a set of conditions when to execute and a body that describes what to execute. The Universal Executive then ties these descriptions with interfaces to actual actors. While PLEXIL is more of a control language, Procedural Reasoning Systems (Ingrand et al. 1996) lean more towards a knowledge-based representation with an explicit fact base, a notion of goals to achieve or maintain, and activation conditions for procedures. An advantage here is less constrained execution flow, however, this gain in expressiveness may easily come with unintended execution orders without the required caution. A more recent system integrating planning and execution is ROSPlan (Cashmore et al. 2015). It provides a general framework where the individual components can be exchanged (with a varying degree of effort). One of the available dispatchers uses a representation of the plan as an Esterel (Berry and Gonthier 1992) program. There, a plan is described as a set of modules interconnected with signals and receiver slots. However, at this point the translation and execution is opaque and no influence can be exerted on the formulation of the program. There is currently only a limited form of concurrency available. A slightly different approach that has been compared to Esterel is RMPL (Ingham, Ragno, and Williams 2001). Instead of a signal flow, it models the flow more as an evolution of states. Both provide primitives for sequential or parallel execution of code blocks, and conditionals. An earlier system to provide an extensible PDDL-based planning system was TFD/M with semantic attachments (Dornhege et al. 2009). However, the executive was a C++ program which had to be augmented each time for the respective available actions and did not provide a flexible specification language. A more unified approach was recently taken through integrating continual planning in Golog (Hofmann et al. 2016). The overall domain model and execution specifics are encoded in Golog. For planning (sub-)problems the model is translated into PDDL and a planner is called. The specification contains assertions to deal with incomplete knowledge and improve planning efficiency. However, the modeling in Golog can be somewhat tedious and it is often deeply intertwined with its Prolog implementation.

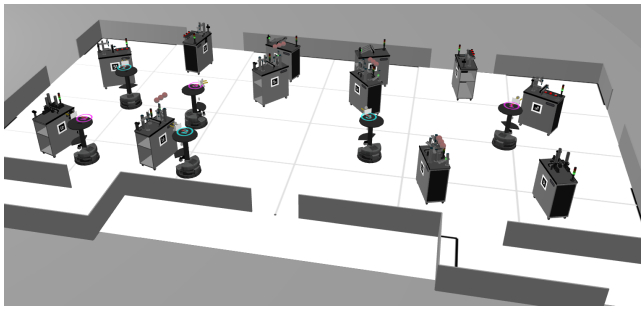


Figure 1: The simulation environment used for our experiments.

In this work, we propose a new formulation of the execution as a rule-based system. With the experience of modeling the decision making, multi-robot coordination, and task execution for the RoboCup Logistics League (Niemueller et al. 2016b), we intend to generalize the framework to be applicable with various planning, reasoning, and decision making components. By this we decouple planning from execution. This comes at the cost of linking two separate models in a consistent way. However, it provides a great flexibility for the executive to choose the appropriate planning system and to add domain-specific interpretations of the plan easily. The planning system we are going to study in more detail in this paper is based on SMT and described in the following.

3 Logistics Robots in Simulation

The example domain chosen for our first integration is based on the Planning Competition for Logistics Robots in Simulation (Niemueller et al. 2016a) and the RoboCup Logistics League (Niemueller, Lakemeyer, and Ferrein 2015). There, the task is to control a group of three robots to transport workpieces among a number of machines. These machines offer processing steps such as mounting a colored ring or a cap. Orders that denote the products which must be constructed with these operations are only posted at run-time and therefore require quick planning and scheduling. Orders come with a delivery time window introducing a temporal

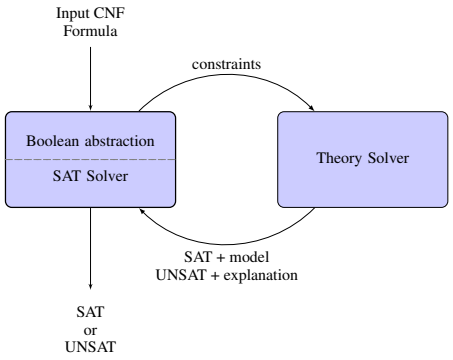


Figure 2: The SMT solving framework.

component into the problem. Furthermore, some machines may undergo maintenance at unknown times at which they may not be used. These aspects make this an integrated planning and execution challenge.

The original competition's game is structured in two phases, the exploration and the production phase. While the latter constitutes the main part of the game, in this work we focus on the former as a simpler scenario for this early proof-of-concept work.

In the exploration phase, the robots must roam the environment and determine where the team's own machines are positioned. For this, the playing field is divided into 24 virtual zones, of which 12 belong to each of the two teams (operating at the same time in the environment increasing execution duration uncertainty considerably). In 6 of these zones will be actual machines. Therefore, the task is to efficiently assign the three robots to the 12 zones, identify the zones which contain a machine, and the precisely determine the position of the machine and recognize a industrial light signal for verification.

4 SMT-based Planning and Optimization

Satisfiability Modulo Theory (SMT) is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the Boolean satisfiability problem (SAT) – see (Franco and Martin 2009) for an overview – by adding expressive background theories such as the theory of real numbers, the theory of integers, and the theories of data structures (e.g., lists, arrays and bit vectors). The idea behind SMT is that the satisfiability of a constraint network can be decided in two steps, as shown in Figure 2. Given an input formula φ in Conjunctive Normal Form (CNF), a Boolean abstraction is first built, e.g., the formula

$$x \geq y \wedge (y > 0 \vee x > 0) \wedge y \leq 0$$

is converted as follows

$$A \wedge (B \vee C) \wedge \neg B$$

where $A, B, C \in \{0, 1\}$ and "$\neg$" is the symbol for Boolean negation. A specialized Boolean satisfiability (SAT) solver can now be invoked to compute an assignment that satisfies all the constraints. In the example above, $A = 1, B = 0, C = 1$ satisfies all the constraints. Notice that if no such assignment exists, then the set of arithmetic constraints is trivially unsatisfiable. The second step is to check the consistency of the assignment in the corresponding background theory. In our example, we therefore need to check whether the system of linear inequalities

$$\begin{cases} x \geq y & (A) \\ y \leq 0 & (\neg B) \\ x > 0 & (C) \end{cases}$$

is feasible, which is clearly the case here. Checking that the Boolean assignment is feasible in the underlying mathematical theory can be performed by a specialized reasoning procedure, i.e., SMT solvers like `Z3` (de Moura and Bjørner 2008), `SMT-Rat` (Corzilius et al. 2015), `MathSAT5` (Cimatti et al. 2013) and `Yices2` (Dutertre 2014). If the constraints are consistent in the theory and the SAT solver's assignment is already complete then, a satisfying solution (also called *model*) is found for the input formula. If the consistency check fails the theory solver returns an *explanation* for the conflict (i.e., the infeasible subset of input constraints), then a new Boolean assignment is requested and the SMT solver goes on until either an theory-consistent Boolean assignment is found, or no more Boolean assignments can be found.

Yet, determining whether a set of constraints is satisfiable or not might not be sufficient, especially when planning is involved. This problem being well recognized in the SMT community, state-of-the-art solvers have introduced the possibility to state and solve optimization objective. Back to our working example, we may want to find solutions that, e.g., maximize y. Given our quantifier free formula φ and the objective function f, we first compute and assignment that satisfies φ (say, e.g., $x = 3$ and $y = -2$). Then, use this assignment to compute a local optimum opt for f, which is trivial in our case as our only objective is to maximize y. Now, we can search for the next assignment that improves the current solutions by feeding the solver with φ updated as follows

$$\varphi \leftarrow \varphi \wedge f > opt \wedge \neg(x = 3 \wedge y = -2)$$

Repeating this procedure discarding all previously computed assignments will eventually lead to the assignment $y = 0$.

Planning with SMT Solving

Over the last decade, SMT solvers have emerged as a core technology in many areas that require analysis of large state spaces – see (Ábrahám and Kremer 2016) for an overview. This is because large, even infinite, sets of system states can be compactly represented as formulas in first-order logic, which may lead to improvements in the scalability of state space analysis. Moreover, compared to traditional SAT solvers, SMT solvers provide a more expressive interface with useful features for expressing constraints in robotics domains. For instance, Nedunuri et al. (2014) and Wang et al. (2016) use an SMT solver to generate task and motion plans from a static roadmap, employing plan outlines to

guide the planning process. Dantam et al. (2016) propose an algorithm to perform task and motion planning which leverages incremental solving in `Z3` to update constraints about motion feasibility. (Saha et al. 2014) present a motion planning framework where SMT solving is used to combine motion primitives so that they satisfy safety requirements specified in linear temporal logic (LTL). SMTPlan (Cashmore et al. 2016) takes a PDDL-based model for a hybrid task and translates it into a SMT problem through a generic transformation. In contrast, we aim to directly model problems using SMT. Especially when using optimization, more direct control and possibly other ways of formulation that are not action-based might allow for more tuning and better results. SMTPlan could be used as a pre-processor to generate a SMT formulation which can then be modified and improved , e.g., enriching it with more (potentially domain-specific) constraints to increase efficiency.

The key differences between the works listed above and ours are that *i.* we do not use additional knowledge (i.e., motion primitive, plan outlines) to seed the search performed by the SMT solver *ii.* we exploit the optimization features offered by solver to synthesize plans that are not only *feasible*, but also *optimal* with respect to some criteria.

Encoding

Formal Description The planning task introduced in Section 3 can be formalized as follows. For each robot $i \in \{1, 2, 3\}$ and each step $j \in \{-3, M\}$ belonging to the path of robot i, $pos_{i,j} = k$ states that robot i visits a potential machine k in the $j-$th step of his plan. Note that locations $-3, -2, -1$ are used to represent the initial location of each robot and 0 the start position. Furthermore, we introduce a stop location -4 to represent the moment when a robot does not move anymore, and $d_{i,j} \in \mathbb{R}$ to store the distance traveled by each robot until step j. Finally, let m_i be a Boolean variable used to represent the maximum over distances $d_{i,M} \quad \forall i$. As our task is to identify all machines in the environment within a time limit, we start by specifying the optimization objectives as

$$\text{minimize} \quad \sum_{i=1}^{3} m_i * d_{i,M}$$

$$\text{minimize} \quad \sum_{i=1}^{3} d_{i,M}$$

so as to minimize the total traveled distance, while minimizing the maximum distance traveled by each robot at the same time. Our problem is then initialized according to the following constraints:

$$pos_{i,0} = 0 \quad \forall i = 1, 2, 3$$
$$pos_{i,-1} = -1 \quad \forall i = 1, 2, 3$$
$$pos_{1,-3} = pos_{1,-2} = pos_{1,-1}$$
$$pos_{2,-2} = pos_{2,-3} = pos_{3,-2} = -2$$
$$pos_{3,-3} = -3$$
$$d_{i,0} = 0 \quad \forall i = 1, 2, 3$$

The above constraints simply fix an order on the way robots can start moving considering physical constraints coming from their initial locations. Also, the cost of reaching the start location is assumed to be 0. Once reached the start location, each robot will start the exploration phase. The choice of the first machined to be visited is modeled according to

$$\forall i = 1, 2, 3 \quad \bigvee_{k=1}^{M} \Big[pos_{i,1} = k \wedge d_{i,1} = dist(0, k) \Big]$$

meaning that for all robots i, the destination of the first visit can be chosen among all k machines and the individual cumulative distance needs to be updates accordingly. Successive steps are modeled as follows

$$\forall i = 1, 2, 3 \quad \forall j = 2, ..., M$$
$$\Big[\bigvee_{k=1}^{M} \bigvee_{\substack{l=1 \\ l \neq k}}^{M} pos_{i,j-1} = k \wedge pos_{i,j} = l \wedge d_{i,j} = d_{i,j-1} + dist(k, l) \Big]$$
$$\vee \Big[pos_{i,j} = -4 \wedge d_{i,M} = d_{i,j-1} \Big]$$

where for all robots i and planning steps j (starting from the second) a robot can either decide to move from machine k to machine l and update the cumulative distance accordingly, or to stop moving at all. In the latter case, the robot is set to stop and the cumulative is not increased anymore.

In order to speed up the search for an optimal solution, we enforce that each machine can be visited only once using

$$\forall k = 1, ..., M$$
$$\bigvee_{i=1}^{3} \bigvee_{j=1}^{M} \Big[pos_{i,j} = k \wedge \bigwedge_{u=1}^{3} \bigwedge_{v=1}^{M} ((i = u \wedge j = v) \vee pos_{u,v} \neq k) \Big]$$

In short, this constraint encodes that if a robot i is visiting a machine k in its $j-$th step, then for all other robots $u \neq i$ and respective planning steps $v \neq j$, machine k cannot be visited. In order to encode $\max_i d_{i,M}$ we specify the following constraints

$$\forall i = 1, 2, 3 \quad \Big[m_i = 0 \vee (m_i = 1 \wedge \bigwedge_{\substack{l=1 \\ l \neq i}}^{3} d_{l,M} < d_{i,M}) \Big]$$

which enforces that variable m_i be set to `true` only if the corresponding distance $d_{i,M}$ is greater than other robots'.

Practical notes on the encoding The problem as described above can be written in `SMT-LIB` format (Barrett, Stump, and Tinelli 2010) and later fed to a solver. Variables can be declared as follows

```
(declare-fun d_1_1 () Real)
(declare-fun pos_1_1 () Int)
```

while constraints can be asserted using prefix notation as

```
(assert (= (* pos_1_1 d_1_1) 3))
```

The optimization objectives introduced before can be specified as

```
(minimize (+ (* m_1 d_1_12) ...))
(minimize (+ d_1_12 d_2_12 d_3_12))
```

Notice that multiple optimization objectives are handled differently by different solvers. `Z3`, the solver we used, solves objectives in lexicographical order by default (other strategies are allowed). This means that objectives are solved in the order they are specified in the `SMT-LIB` file. We can finally ask the solver to check whether the constraints are satisfiable by stating `(check-sat)`.

If a model can be found, we can traverse it so as to collect all variable assignments relevant to our plan. In our case, the solution obtained will have elements of the form `(pos_1_1 5)`. Converting such solution into a plan is now an easy task as a result of how the problem was originally formulated. Recall that the assignment $pos_{i,j} = k$ states that robot i visits machine k at the $j-$th step of its plan. Given that the only action considered in this problem is `move`, the sample solution above can be read as "the $j-$th action executed by robot i is `move` to machine k".

5 CLIPS-based Execution and Monitoring

The goal of our executive is to allow for a flexible formulation of the plan execution that can work completely automatic given a plan, but can be extended for monitoring the execution and listening for events. The overall architecture is depicted in Figure 3. The CLIPS executive controls the overall execution. At a suitable time (when the game enters the appropriate phase indicated by the referee box), it triggers the SMT solving process to come up with a plan to explore the machines and encodes the available knowledge in a message. The solver queries travel times of the position for exploration from a navigation graph and uses a domain model phrased suitably for SMT. The result is then represented as a plan and sent to the CLIPS executive, which must translate it into a native representation for execution. This is then synchronized with all robots as part of the world model. The robots then execute their respective partial plans by invoking the appropriate basic behaviors through the behavioral and functional components of the Fawkes software framework (BE represents the Lua-based Behavior Engine that provides the basic skills to execute the plans, for details see below). Only through this framework do the reasoning systems interact with the simulation (which would make it easy to replace this with actual robots later).

In the following, we are going to describe the CLIPS rules engine which is used to implement the executive before describing the plan translation and execution in more detail.

CLIPS Rules Engine

CLIPS (Wygant 1989) is a rule-based production system using forward chaining inference based on the Rete algorithm (Forgy 1982). It has been used before for autonomous robot reasoning (Niemueller, Lakemeyer, and Ferrein 2013), knowledge-based instrumentation and control (Niemueller et al. 2016c), and inspired other systems such as Jess (Friedman-Hill 1999). CLIPS consists of three building blocks (Giarratano 2007): a fact base or working memory, the knowledge base, and an inference engine. *Facts* are basic forms representing pieces of information in the fact base. They usually adhere to structured types. The

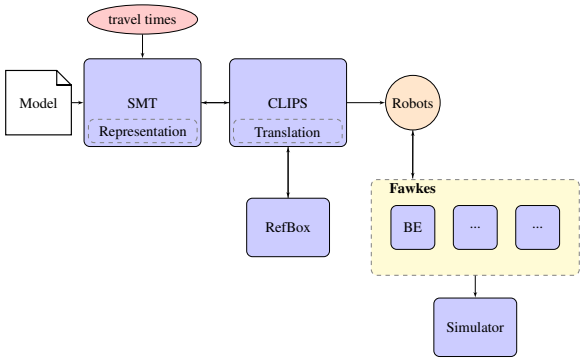


Figure 3: The overall architecture.

knowledge base comprises heuristic knowledge in the form of rules, and procedural knowledge in the form of functions. *Rules* are a core part of the production system. They are composed of an antecedent and consequent. The antecedent is a set of conditions, typically patterns which are a set of restrictions that determine which facts satisfy the condition. If all conditions are satisfied based on the existence, nonexistence, or content of facts in the fact base the rule is activated and added to the agenda. The consequent is a series of actions which are executed for the currently selected rule on the agenda, for example to modify the fact base. *Functions* carry procedural knowledge and may have side effects. They can also be implemented in C++. In our framework, we use them to utilize the underlying robot software, for instance to communicate with the reactive behavior layer described below. CLIPS' *inference engine* combines working memory and knowledge base performing fact updates, rule activation, and agenda execution until stability is reached and no more rules are activated. Modifications of the fact base are evaluated if they activate (or deactivate) rules from the knowledge base. Activated rules are put onto the agenda. As there might be multiple active rules at a time, a conflict resolution strategy is required to decide which rule's actions to execute first. In our case, we order rules by their salience, a numeric value where higher value means higher priority. If rules with the same salience are active at a time, they are executed in the order of their activation.

Planning and Plan Translation

To start planning, the information relevant to the planning procedure must be encoded in a way accessible to the planning system. In this work, we have used Google Protocol Buffers[2] (protobuf) to represent both, the initial planning situation as well as the resulting plan. Protobuf defines a schema for messages (for an example cf. Listing 2, explained in more detail below) for which code can be generated for a variety of languages. It provides a convenient transport, exchange, and storage representation that is easy to create and read. It also has powerful introspection capabilities which are particularly useful for generic access from typical reasoning systems, for example, the CLIPS-based access requires only the definition files and not any

[2] `https://developers.google.com/protocol-buffers/`

```
1  (defrule production-call-clips-smt
2    (phase PRODUCTION)
3    (team-color ?team-color&CYAN|MAGENTA)
4    (state IDLE)
5    (not (plan-requested))
6    (test (eq ?*ROBOT-NAME* "R-1"))
7  =>
8    (bind ?p
9      (smt-create-data
10       (smt-create-robots ?team-color)
11       (smt-create-machines ?team-color)
12       (smt-create-orders ?team-color)
13     )
14   )
15   (smt-request "explore-zones" ?p)
16   (assert (plan-requested))
17 )
```

Listing 1: CLIPS rule to trigger planning.

```
1  message ActorGroupPlan {
2    repeated ActorSpecificPlan plans = 1;
3  }
4  message ActorSpecificPlan {
5    required string  actor_name = 1;
6
7    oneof plan {
8      SequentialPlan sequential_plan = 2;
9      TemporalPlan   temporal_plan = 3;
10   }
11 }
12 message SequentialPlan {
13   repeated PlanAction actions = 1;
14 }
15 message PlanAction {
16   required string name = 1;
17   repeated PlanActionParameter params = 2;
18 }
19 message PlanActionParameter {
20   required string key = 1;
21   required string value = 2;
22 }
```

Listing 2: Plan representation in protobuf.

pre-generated code. The rule to initiate the planning process in shown in Listing 1. Here, once the game is started (ll. 2–4), the first robot (l. 6) will create the data structure with the relevant information (ll. 8–14, the `smt-create-*` calls are functions) and request a plan from the SMT solver (l. 15).

The SMT side notifies the executive once the plan is ready for retrieval. An excerpt of the message specifications for plan representation is shown in Listing 2. First, a list of plans that name a specific actor (robot) is defined in ll. 1–3. Lines 5–12 define such a plan. It names the actor for the plan and either a sequential or a temporal plan (`oneof` clause). For reasons of brevity, we focus on the sequential plan consisting of a series of actions (ll. 14–16). An action is defined by an operator name and a number of key-value pairs for the arguments (ll. 18–26). An example plan for two robots with two movements commands in shown in Listing 3.

Once the plan has been retrieved, it must be translated into the native CLIPS representation, that is as facts in the working memory. This translation in general is a straight-forward one. But it may require a domain-specific mapping from action names and parameters in the planner's representation to actions understood by the underlying base system. For example, in our system the "move" skill is called "drive_to". This conversion must be performed during translation of the plan. Likewise, differences in action parameters might have to be translated. Example facts for the translation of the plan is shown in Listing 4.

```
1  plans[0]:ActorSpecificPlan {
2    actor_name: "R-1"
3    sequential_plan:SequentialPlan {
4      actions[0]:PlanAction {
5        name: "move"
6        params[0]:PlanActionParameter {
7          key: "to"
8          value: "C-BS-I"
9        }
10     }
11     actions[1]:PlanAction {
12       name: "move"
13       params[0]:PlanActionParameter {
14         key: "to"
15         value: "C-DS-I"
16       }
17     }
18   }
19 }
20 plans[1]:ActorSpecificPlan {
21   actor_name: "R-2"
22   sequential_plan:SequentialPlan {
23     actions[0]:PlanAction {
24       name: "move"
25       params[0]:PlanActionParameter {
26         key: "to"
27         value: "C-CS1-I"
28       }
29     }
30     actions[1]:PlanAction {
31       name: "move"
32       params[0]:PlanActionParameter {
33         key: "to"
34         value: "C-RS2-I"
35       }
36     }
37   }
38 }
```

Listing 3: Plan represented through the messages from Listing 2 (shown in augmented JavaScript Object Notation).

The CLIPS representation denotes tasks as a number of consecutive steps that must be executed sequentially. There may be only a single task active per robot at a time. A step then triggers the execution of a basic behavior. If it completes successfully, the next step is executed, until all have been processed and the task is complete. If any of the steps fails the task is considered to have failed.

Plan Distribution and Execution

Once the plan has been added to the working memory, it has to be distributed to the robots for execution. In the current

```
(task (task-id 1910) (robot "R-1") (name explore)
      (state proposed) (steps 1911 1912))
(step (id 1911) (name drive-to) (state inactive)
      (machine C-BS) (side INPUT)
      (sync-id (next-sync-id)))
(step (id 1912) (name drive-to) (state inactive)
      (machine C-DS) (side INPUT)
      (sync-id (next-sync-id)))
```

Listing 4: Task representation in CLIPS.

version, we rely on the communication infrastructure otherwise used to share world model updates among the robots. It encapsulates fact base updates in protobuf messages and broadcasts them to the other robots. A (dynamically elected) master generates a consistent view and distributes it to the robots. We deem the plans as being part of the world model.

On each robot, the CLIPS executive has rules that automatically start new tasks. This happens once the plans have been received as world model updates. Since the actor is named in these plans, a robot will only ever execute its own plan. Updates to the tasks (e.g., that a certain task is currently in progress) is again distributed in the world model and hence the planning and execution instance remains informed of the state of the plan execution.

Basic behaviors in the current framework are provided by the Lua-based Behavior Engine (Niemueller, Ferrein, and Lakemeyer 2009), but could principally be provided by other sources, for example through ROS actions or the ROSPlan action dispatching mechanism. A step is executed by triggering the start of the execution of a behavior, i.e., executing a skill is an instant non-blocking operation. Then, information about the execution of the state is read and asserted in the working memory. Once it completes, a rule fires that denotes a step to be finished.

Execution Monitoring

Especially in robotics, execution often does not go as planned. Some actions may (and eventually some will) fail or not entirely give the expected result, e.g., by misplacing an object, or slack during execution could render a plan invalid, for example if a specified deadline cannot be made.

As described earlier, plans are translated into tasks and actions into steps of the task. Steps are invoked non-blocking, i.e., rule evaluation continues normally. This can be used to implement execution monitoring. Rule can be phrased identifying specific situations of interest where a step should be skipped or a task being aborted.

6 Preliminary Results

Even though this work is in an early stage of development, we have an initial fully integrated prototype. It is based on the planning competition reference architecture and extends it by a plugin to integrate the SMT-based planning functionality. Then, the existing code has been modified to call an SMT component with the relevant information. This component will generate a formula (according to Section 4), call the Z3 solver, and translate and execute the resulting plan with multiple robots in simulation. So far, we have used only the generic capabilities of the executive and do not have, for example, specific execution monitoring rules to react to unexpected (yet foreseen) situations.

We have also integrated the executive with other reasoning system based on Answer Set Programming and PDDL-based planning systems. However, work there is still in an earlier stage preventing a comparison between the approaches at this time.

7 Conclusion

We have shown two bodies of work and their integration: first, the CLIPS rules engine has been used to create a flexible and expressive executive for plans. Second, an SMT-based approach has been used to model and execute the task planning and optimization in an explorative setting. These two elements have been combined by exchanging information through well-structured protobuf messages. The work is in an early stage, but a fully working prototype has been developed and the full system has been run and tested. The executive is in control of the overall system and triggers the planning process as necessary. Then, the resulting plan is translated into a native representation, distributed among the robots, and executed concurrently by the individual robots.

Future work will deal with extending the executive model to be more flexible, complete and thoroughly test the integration with other reasoning approaches, and optimize and compare the SMT-based planning approach with these.

Acknowledgments. This work is supported by the ICT project house Foundations of a Digitized Industry, Economy, and Society of RWTH Aachen University.

T. Niemueller is supported by the German National Science Foundation (DFG) research unit *FOR 1513* on Hybrid Reasoning for Intelligent Systems.

References

[Ábrahám and Kremer 2016] Ábrahám, E., and Kremer, G. 2016. Satisfiability checking: Theory and applications. In *Int. Conf. on Software Engineering and Formal Methods*.

[Barrett, Stump, and Tinelli 2010] Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB Standard: Version 2.0. In *8th Int. Workshop on Satisfiability Modulo Theories*.

[Berry and Gonthier 1992] Berry, G., and Gonthier, G. 1992. The Esterel synchronous programming language: design, semantics, implementation. *Science of computer programming* 19(2).

[Cashmore et al. 2015] Cashmore, M.; Fox, M.; Magazzeni, D. L. D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *25th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Cashmore et al. 2016] Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *26th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*.

[Cimatti et al. 2013] Cimatti, A.; Griggio, A.; Schaafsma, B. J.; and Sebastiani, R. 2013. The MathSAT5 SMT Solver. In *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*.

[Corzilius et al. 2015] Corzilius, F.; Kremer, G.; Junges, S.; Schupp, S.; and Ábrahám, E. 2015. SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In *18th International Conference on Theory and Applications of Satisfiability Testing SAT*.

[Dantam et al. 2016] Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems XII*.

[de Moura and Bjørner 2008] de Moura, L. M., and Bjørner, N. 2008. Z3: an efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[Dornhege et al. 2009] Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic Attachments for Domain-Independent Planning Systems. In *19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Dutertre 2014] Dutertre, B. 2014. Yices 2.2. In *26th International Conference on Computer Aided Verification (CAV)*.

[Forgy 1982] Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19(1).

[Franco and Martin 2009] Franco, J., and Martin, J. 2009. *Handbook of Satisfiability*. IOS Press. chapter A History of Satisfiability.

[Friedman-Hill 1999] Friedman-Hill, E. J. 1999. Jess, the java expert system shell. Technical report, Sandia National Laboratories.

[Giarratano 2007] Giarratano, J. C. 2007. *CLIPS Reference Manuals*. `http://clipsrules.sf.net/OnlineDocs.html`.

[Hofmann et al. 2016] Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *30th Conference on Artificial Intelligence (AAAI)*.

[Ingham, Ragno, and Williams 2001] Ingham, M.; Ragno, R.; and Williams, B. 2001. A Reactive Model-based Programming Language for Robotic Space Explorers. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*.

[Ingrand et al. 1996] Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, volume 1.

[McDermott et al. 1998] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical report, AIPS-98 Planning Competition Committee.

[Nedunuri et al. 2014] Nedunuri, S.; Prabhu, S.; Moll, M.; Chaudhuri, S.; and Kavraki, L. E. 2014. Smt-based synthesis of integrated task and motion plans from plan outlines. In *IEEE Int. Conference on Robotics and Automation (ICRA)*.

[Niemueller et al. 2016a] Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016a. Planning Competition for Logistics Robots in Simulation. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Niemueller et al. 2016b] Niemueller, T.; Neumann, T.; Henke, C.; Schönitz, S.; Reuter, S.; Ferrein, A.; Jeschke, S.; and Lakemeyer, G. 2016b. Improvements for a Robust Production in the RoboCup Logistics League 2016. In *RoboCup Symposium – Champion Teams Track*.

[Niemueller et al. 2016c] Niemueller, T.; Zug, S.; Schneider, S.; and Karras, U. 2016c. Knowledge-Based Instrumentation and Control for Competitive Industry-Inspired Robotic Domains. *KI - Künstliche Intelligenz* 30(3).

[Niemueller, Ferrein, and Lakemeyer 2009] Niemueller, T.; Ferrein, A.; and Lakemeyer, G. 2009. A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In *RoboCup Symposium 2009*.

[Niemueller, Lakemeyer, and Ferrein 2013] Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2013. Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In *AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI*.

[Niemueller, Lakemeyer, and Ferrein 2015] Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *25th International Conference on Automated Planning and Scheduling (ICAPS) – WS on Planning in Robotics*.

[Saha et al. 2014] Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Verma et al. 2005a] Verma, V.; Estlin, T.; Jónsson, A.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005a. Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences. In *9th Int. Symposium on Artificial Intelligence, Robotics and Automation in Space*.

[Verma et al. 2005b] Verma, V.; Jónsson, A.; Simmons, R.; Estlin, T.; and Levinson, R. 2005b. Survey of Command Execution Systems for NASA Spacecraft and Robots. In *Int. Conference on Automated Planning and Scheduling (ICAPS) – Workshop "Plan Execution: A Reality Check"*.

[Verma et al. 2006] Verma, V.; Jónsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations. In *American Institute of Aeronautics and Astronautics – Space 2006*.

[Wang et al. 2016] Wang, Y.; Dantam, N. T.; Chaudhuri, S.; and Kavraki, L. E. 2016. Task and motion policy synthesis as liveness games. In *26th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Wygant 1989] Wygant, R. M. 1989. CLIPS: A powerful development and delivery expert system tool. *Computers & Industrial Engineering* 17(1–4).

Deep Spatial Affordance Hierarchy: Spatial Knowledge Representation for Planning in Large-scale Environments

Andrzej Pronobis, Francesco Riccio, Rajesh P. N. Rao*

Abstract

Domain-specific state representations are a fundamental component that enables planning of robot actions in unstructured human environments. In case of mobile robots, it is the spatial knowledge that constitutes the core of the state, and directly affects the performance of the planning algorithm. Here, we propose Deep Spatial Affordance Hierarchy (DASH), a probabilistic representation of spatial knowledge, spanning multiple levels of abstraction from geometry and appearance to semantics, and leveraging a deep model of generic spatial concepts. DASH is designed to represent space from the perspective of a mobile robot executing complex behaviors in the environment, and directly encodes gaps in knowledge and spatial affordances. In this paper, we explain the principles behind DASH, and present its initial realization for a robot equipped with laser-range sensor. We demonstrate the ability of our implementation to successfully build representations of large-scale environments, and leverage the deep model of generic spatial concepts to infer latent and missing information at all abstraction levels.

1 Introduction

Many recent advancements in the fields of robotics and artificial intelligence have been driven by the ultimate goal of creating artificial agents able to perform service tasks in real environments in collaboration with humans (Aydemir et al. 2013; Hanheide et al. 2016). While significant progress have been made in the area of robot control, largely thanks to the success of deep learning (Levine et al. 2016), we are still far from solving more complex scenarios that require forming plans spanning large spatio-temporal horizons.

In such scenarios, domain-specific state representations play a crucial role in determining the capabilities of the agent and the tractability of the solution. In case of mobile robots operating in large-scale environments, it is the spatial knowledge that constitutes the core of the state. As a result, the way in which it is represented directly affects the actions the robot can plan for, the performance of the planning algorithm, and ultimately, the ability of the robot to successfully reach the goal. For complex tasks involving interaction with humans, the relevant spatial knowledge spans multiple levels of abstraction and spatial resolutions, including detailed geometry and appearance, global environment structure, and high-level semantic concepts. Representing such knowledge is a difficult task given uncertainty and partial observability governing real applications in human environments.

In this work, we propose Deep Spatial Affordance Hierarchy (DASH, ref. Fig. 1), a probabilistic representation of spatial knowledge designed to support and facilitate planning and execution of complex behaviors by a mobile robot. The representation encodes the belief about the state of the world. However, more importantly, it also provides information about spatial affordances, i.e. the possibilities of actions on objects or locations in the environment. It does so by leveraging a hierarchy of sub-representations (layers), which directly correspond to a hierarchical decomposition of the planning problem. The layers represent multiple spatial knowledge abstractions (from geometry and appearance to semantic concepts), using different spatial resolutions (from voxels to places), frames of reference (allo- or ego-centric), and spatial scopes (from local to global). The goal is to represent spatial knowledge in a way that directly corresponds to how it will be utilized by the robot and its planning algorithm.

DASH includes both instance knowledge about the specific robot environment as well as default knowledge about generic human environments. The latter is modeled using a recently proposed Deep Generative Spatial Model (DGSM) (Pronobis and Rao 2017). Specifically, DGSM leverages recent developments in deep learning, providing fully probabilistic, generative model of spatial concepts learned directly from raw sensory data. DGSM unifies the layers of our representation, enabling upwards and downwards inferences about spatial concepts defined at different levels of abstraction. Finally, DASH is designed to explicitly represent and fill gaps in spatial knowledge due to uncertainty, unknown concepts, missing observations or unexplored space. This brings the possibility of using the representation in open-world scenarios, involving active exploration and learning.

*A.Pronobis and R. Rao are with Computer Science & Engineering, University of Washington, Seattle, WA, USA. A. Pronobis is also with Robotics, Perception and Learning Lab, KTH Royal Institute of Technology, Stockholm, Sweden. F. Riccio is with Dept. of Computer, Control, and Management Engineering, Sapienza University of Rome, Rome, Italy. {pronobis,rao}@cs.washington.edu, riccio@diag.uniroma1.it. This work was supported by the Swedish Research Council (VR) project SKAEENet.

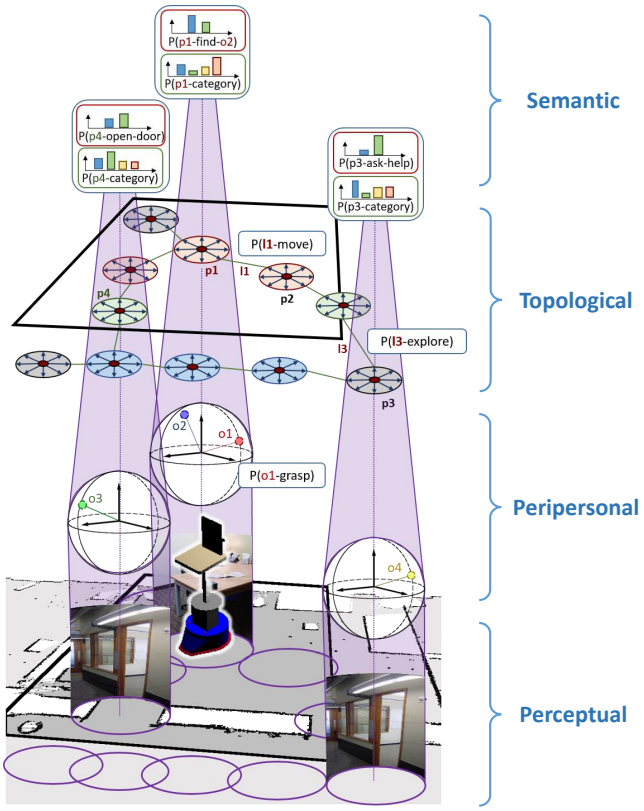


Fig. 1: The multi-layered architecture of Deep Spatial Affordance Hierarchy. The *perceptual layer* integrates perceptual information from the robot sensors. The *peripersonal layer* represents object and landmark information and affordances in the space immediately surrounding the robot. The *topological layer* encodes global topology and coarse geometry and navigation action affordances. Finally, the *semantic layer* relates the internal instance knowledge to human semantic concepts. The four layers are connected by the probabilistic *deep default knowledge model* (shaded purple columns), which provides definitions of generic spatial concepts and their relations across all levels of abstraction.

In this paper, we describe the general architecture of DASH and present an initial realization of the representation for a mobile robot equipped with a laser range sensor. We perform a series of experiments demonstrating the ability of the representation to perform different types of inferences, including bottom-up inferences about semantic spatial concepts and top-down inferences about geometry of the environment. We then showcase its ability to build semantic representations of large-scale environments (e.g. floors of an office building).

We begin the presentation of DASH with a description of the scenario, an analysis of roles and desired properties of a spatial knowledge representation (Sec. 2). Then, we describe the architecture of DASH (Sec. 3), present its initial realization (Sec. 4) and describe the details of the deep generative model of default spatial knowledge (Sec. 5). We follow with the experimental evaluation in Sec. 6.

2 Analysis of the Problem

We recognize that the ultimate purpose of a spatial knowledge representation for an autonomous mobile robot is to enable and facilitate successful planning and execution of actions in the robot environment. Here, we focus specifically on scenarios involving large-scale, dynamic, human environments, such as office buildings, homes, and hospitals. We assume that a mobile robot is physically capable of sensing the environment using on-board sensors. The sensors are likely to have limited field of view, and might be attached to actuators, such as pan-tilt units. Furthermore, the robot is capable of moving around the environment and performing basic manipulation tasks (e.g. grasping objects or pushing buttons). Finally, we assume that the robot can interact and collaborate with humans in order to accomplish its tasks (e.g. by asking for additional information or requesting help when a task cannot be accomplished by the robot itself). We follow with an analysis of roles of a spatial knowledge representation in the context of the considered scenarios as well as a discussion of its desired properties.

Role of a Spatial Knowledge Representation

Referring to the discussion of roles of a knowledge representation in (Davis, Shrobe, and Szolovits 1993), and a more specific analysis for spatial knowledge in (Pronobis et al. 2010b), we formulate a set of roles of a domain-specific spatial knowledge representation for a mobile robot. Such a representation can be seen as:

a) A substitution (surrogate) for the world that allows the robot to reason about actions involving parts of the environment beyond its sensory horizon. The surrogate can either represent the belief about the state of the world (what the world looks like), or more directly, the belief about affordances (what the robot can do at a specific place or involving a specific spatial entity). It is important to note that it is inherently imperfect, i.e. it is incomplete (some aspects of the world are not represented), inaccurate (captured with uncertainty), and likely to become invalid (e.g. due to dynamics of the world).

b) A set of commitments that determine the terms in which the robot thinks about space. The representation defines which aspects of the world are relevant, and specifies the formalism used to represent and relate them. To this end, it defines the levels of abstraction at which spatial entities exist, spatial frames of reference used to relate them (absolute or relative, allo- or ego-centric) as well as their persistence. It is worth noting that these commitments significantly affect the ability of the robot to plan and execute specific actions. Furthermore, the representation does not have to be more expressive than required to successfully act. Therefore, we can think of the commitments in the representation as defining part of the action space of the robot.

c) A set of definitions that determine the reasoning that can be (and that should be) performed within the framework. This includes reasoning about the location of the robot with respect to the internal frames of reference (whether metric, topological or semantic), inferring more abstract concepts from observations (e.g. affordances, semantic descriptions),

or generating missing lower-level information from high-level descriptions (e.g. expected position of occluded objects in rooms of known functional category).

d) A medium of communication between the robot and humans. In scenarios involving human-robot collaboration, spatial knowledge provides a common ground for communication and knowledge transfer. The representation must therefore be capable of relating human spatial concepts to those internal to the robot.

e) A way of structuring the spatial information so that it is computationally feasible to perform inferences and action planning in a specified time (e.g. in real time) despite limited resources.

Desired Properties of the Representation

Having in mind the specifics of the scenario, the roles of a representation, practical limitations, and experience resulting from existing approaches and robotic systems (Thrun et al. 1998; Kuipers 2000; Marder-Eppstein et al. 2010; Hanheide et al. 2016), we identify several desired properties of a spatial knowledge representation for mobile robots.

Spatial knowledge in realistic environments is inherently uncertain and dynamic. Given the local nature of the robot's sensing, it is futile to represent the environment as accurately as possible. A very accurate representation is likely to be intractable and will require a substantial effort to be kept up-to-date. Moreover, its usability will remain constrained by robot capabilities. Hence, our primary assumption is that the representation should instead be minimal and the spatial knowledge should be represented only as accurately as it is required to support the functionality of the robot.

Planning is a computationally demanding process and its complexity increases exponentially with the size of the environment and number of considered spatial entities. However, due to the way real-world environments are structured and limitations of robot sensors and actuators, decomposing the planning problem hierarchically can greatly reduce its complexity while maintaining highly optimal results. This naturally leads to a hierarchy of higher-level, long-term, global plans involving lower-level short-term, local behaviors. In fact, hierarchical planners are used in the majority of existing robotic systems (Marder-Eppstein et al. 2010; Aydemir et al. 2013; Hanheide et al. 2016) due to their tractability. Moreover, behavioral analyses found hierarchical spatial planning in humans (Balaguer et al. 2016). In order to support such strategies, a spatial representation should perform knowledge abstraction, providing symbols corresponding to spatial phenomena of gradually increasing complexity, anchored to reference frames of increasing spatial scope and decreasing resolution. This leads to discretization of continuous space, which significantly reduces the number of states for planning (Hawes et al. 2009) and provides a basis for higher-level conceptualization (Zender et al. 2008).

Due to the dynamic properties of the real world, abstracted knowledge is more likely to remain valid over time. At the same time, high-resolution up-to-date spatial information is required for executing actions in the robot peripersonal space. Yet, it can also be re-acquired through perception. Therefore, the representation should correlate the levels of abstraction with the persistence of information, employing local working-memory representations for integrating high-resolution spatial information (visual servoing being the extreme example). In other words, the robot should use the world as an accurate representation whenever possible.

Representing uncertainty in the belief state is crucial for the robot to make informed decisions in the real-world, including planning for epistemic actions and anticipating future uncertainty. In this context, decision-theoretic planning algorithms rely on probabilistic representations of uncertainty, therefore, it is desirable for a knowledge representation to also be probabilistic in nature.

Furthermore, a representation should not only represent what is known about the world, but also what is unknown. This includes explicit representation of missing evidence (e.g. due to occlusions), unexplored space (e.g. exploration frontiers) or unknown concepts (e.g. unknown object categories). Representing knowledge gaps can be exploited to address the open-world problem (in the continual planning paradigm (Hanheide et al. 2016)), trade exploration vs exploitation, or drive learning.

3 Deep Spatial Affordance Hierarchy (DASH)

As a result of the problem analysis, we propose Deep Spatial Affordance Hierarchy (DASH). A general overview of the architecture of the representation is shown in Fig. 1. DASH represents the robot environment using four sub-representations (layers) focusing on different aspects of the world, encoding knowledge at different levels of abstraction and spatial resolutions as well as in different frames of reference of different spatial scope. The characteristics of the layers were chosen to simultaneously support both action planning and spatial understanding for the purpose of localization and human-robot interaction. In particular, the former objective is realized by directly representing spatial affordances, which we define as the possibilities of actions on objects or locations in the environment relative to the capabilities and state of the robot. The characteristics of the layers are summarized in Table 1.

DASH is organized as a hierarchy of spatial concepts, with higher-level layers providing a coarse, global representation comprised of more abstract symbols, and lower-level layers providing a more fine-grained representation of parts of the environment anchored to the higher-level entities. The layers are connected by a crucial component of the representation, the probabilistic *deep default knowledge model*, which provides definitions of generic spatial concepts and their relations across all levels of abstraction.

The hierarchy directly relates to a similar, hierarchical decomposition of the planning problem. A global planner can derive a navigation plan relying only on the top layers for representing its beliefs, a local planner can be used to plan specific manipulation actions using intermediate layers, with a controller realizing them base on knowledge in the lowest-level representation. Below, we provide details about each component of the representation.

	Perceptual	Peripersonal	Topological	Semantic
World Aspects Captured	Detailed geometry and appearance	Object/landmark info, coarse local geometry	Large-scale topology, coarse global geometry	Human semantic descriptions
Reference Frame	Metric (allo-centric, sliding window)	Collection of: Metric (epi-centric)	Topological (allo-centric) Metric (allo-centric)	Relational
Spatial Scope	Sensory horizon	Local	Global	Global
Spatial Entities	Voxels	Objects/landmarks	Places, paths, views	Relations to human concepts
Affordances	—	Manipulation and epistempic actions	Navigation and epistemic actions	Human interaction actions Tasks involving human concepts
Robot Pose	Center of the window	Relative to objects/landmarks	Place/view ID	Described semantically
Knowledge Gaps	Missing observations	Missing evidence Unknown objects	Unexplored space Unknown places	Novel semantic concepts

Table 1: Characteristics of the four layers of DASH.

Perceptual Layer

At the bottom level of the representation is the *perceptual layer*. The layer maintains an accurate representation of the geometry and appearance of the local environment obtained by short-term spatio-temporal integration of perceptual information from (possibly multiple and directional) sensors with finite horizon. Spatial information in perceptual layer is represented in an allo-centric metric reference frame, which facilitates integration of perception from multiple viewpoints and sensors. However, the representation is always centered at the current location of the robot, and spans a radius roughly corresponding to the maximum range of the robot sensors (essentially a sliding window). Information outside the spatial scope is forgotten, which makes the layer akin to a working memory, and enables consistent large-scale higher-level representations without the need to maintain low-level global consistency. The layer provides a more complete input for further abstractions with reduced occlusions and noise. It enables tracking of the relative movements of the robot, and forms a basis for deriving low-level control laws for manipulation and obstacle avoidance. Missing observations (e.g. due to unresolved occlusions) are explicitly represented.

Peripersonal Layer

Above the perceptual layer is the *peripersonal layer*, which captures spatial information related to object and landmark instances from the perspective of an agent performing actions at different locations in the environment. To support planning, the layer represents object affordances related to actions that can be performed directly by the robot. This includes manipulation (e.g. possibility of reaching/grasping an object or pressing a button), interaction in relation to objects (e.g. possibility of pointing at an object), and epistemic affordances (e.g. possibility of observing an object). Furthermore, the layer captures object and landmark descriptors that are internal to the robot as well as spatial relations between objects and landmarks in relation to the robot (and therefore coarse local geometry). Finally, it serves as an intermediate layer of the deep default knowledge model, used to generate descriptions of locations in terms of higher-level concepts (e.g. room categories or place affordances).

To reflect the local and robo-centric nature of the captured information, the peripersonal layer relies on a collection of ego-centric, metric reference frames, each focusing on the space immediately surrounding the robot at a different location in the environment (see Fig. 1). The spatial scope of each of the reference frames is defined primarily by the peripersonal space of the robot, within which objects can be grasped and manipulated. However, to support epistemic affordances, interaction about objects, and higher-level conceptualization, the scope can be extended to include context in the form of knowledge about objects that directly relates to the functionality of the location. For instance, a reference frame centered in front of a desk might include information about shelves and books in the room, even beyond the reach of the robot. While recent results from neuropsychology suggest existence of local, body-centered representations in animals and humans (Holmes and Spence 2004), our motivation for such decomposition is primarily the efficiency of the planning problem.

The peripersonal layer explicitly represents gaps in knowledge about the local space due to missing evidence (e.g. resulting from occlusions) and unknown objects. The latter occurs when the default knowledge model is not familiar with an object, and cannot produce a certain object descriptor or affordance information.

Topological Layer

The topological layer provides an efficient representation of large-scale space, including coarse geometry and topology, and serves several key roles in DASH. First, it provides a way to express the global pose of the robot. Second, it captures navigation and exploration action affordances associated with locations in the environment. Third, it is a global counterpart to the local peripersonal representations and anchors them in the large-scale space. Finally, it captures internal descriptors of places and serves as an intermediate layer

of the deep default knowledge model used to derive semantic place descriptions.

To this end, the layer performs a bottom-up discretization of continuous space into a set of locations called *places*. Places correspond to locations in the environment previously visited by the robot, and are meant to represent space at a resolution sufficient for action execution, while maintaining efficiency and robustness to dynamic changes. In other words, the resolution is selected to ensure that high-level navigation can be planned using the topological layer only, with local behaviors planned using the knowledge in the peripersonal layer at the destination. Places are spatially related to other, neighboring places, which encodes coarse global geometry of the environment and allows for path integration.

For each place, the topological layer maintains a set of discrete headings, called *views*. Together with places, views can be used to efficiently represent the complete global pose of the robot. Moreover, views and places are used to anchor knowledge in the representation. First, the topological layer captures robot-internal descriptors of each view and place. The descriptors are derived from lower-level representations using the deep default knowledge model and serve as an intermediate layer of the model. Second, each visited place anchors a peripersonal representation describing the place in more detail.

Besides places and views, the layer also defines *paths* connecting neighboring places into a topological graph. The semantics of a path between two places is the possibility of navigating directly from one place to the other. Thus, essentially, paths represent navigation place affordances, which can be associated with probability indicating uncertainty estimated based on the current, detailed information in the peripersonal layer (e.g. based on visible obstacles). Furthermore, the topological nature of the graph of places and paths, enables planning of complex navigational tasks, such as involving elevators. The place in the elevator might afford navigating to places on different floors, depending on the information captured in the peripersonal layer (e.g. displayed floor number) or additional state information.

Existence of a path in the graph does not necessarily imply that it has previously been traveled by the robot. In fact, a path can indicate the possibility of navigating towards unexplored space. To this end, the topological layer utilizes the concept of *placeholders* (Pronobis et al. 2010b), which can be seen as candidate places, and are used to explicitly represent unexplored space. As a result, paths that lead to placeholders express the possibility of epistemic exploration actions. This can be used to address the open world problem, for instance, in the continual planning paradigm (Hanheide et al. 2016).

Semantic Layer

On top of DASH is the semantic layer, a probabilistic relational representation relating the spatial entities in the other layers to human semantic spatial concepts defined in the deep default knowledge model. This includes such concepts as object categories and attributes, place attributes, room categories, or the concept of a room itself. It is the semantic layer that captures the knowledge that an object is likely to be a cup, or that certain places are likely to be located in a kitchen. Furthermore, the layer plays an important role in planning complex tasks, by representing place affordances related to human interaction as well as actions characterized in terms of human concepts. For instance, it is the sensory layer that defines the affordance expressing the possibility of asking a person for help with making coffee or the possibility of finding a cup at a certain place. Finally, the layer enables transfer of knowledge from humans to the robot (e.g. capturing object category information provided by the user). Such knowledge can be utilized by the default knowledge model to generate lower-level information stored in other layers.

Deep Default Knowledge

The four layers representing knowledge about the specific robot environment are linked by the *deep default knowledge model*. The model provides definitions of generic spatial concepts, valid for typical human environments, and their relations across all levels of abstraction (from sensory input to high-level concepts). This includes robot-internal models of objects in terms of low-level perception, places in terms of objects, place and object affordances, or models of semantic categories and attributes of objects and places. In other words, the four layers can be seen as defining the traditional ABox of our spatial knowledge base, while the deep default knowledge model represents its TBox.

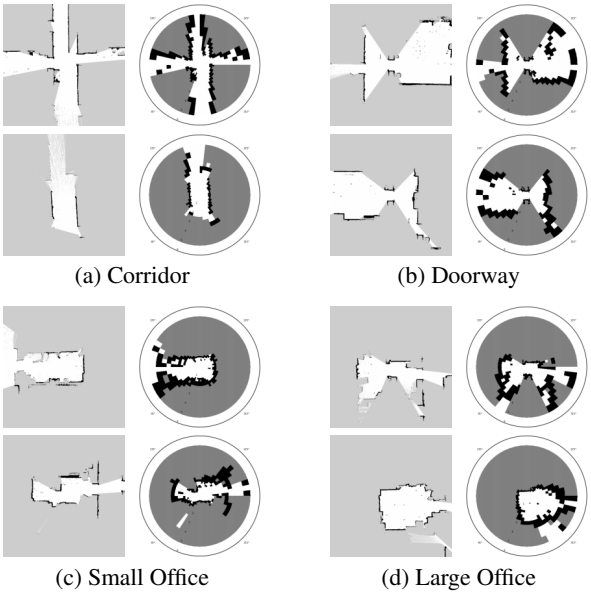
The role of the default knowledge model is to permit inferences about missing or latent aspects of the environment in each layer, based on the knowledge available in other layers. This includes bottom-up inferences (e.g. about semantic descriptions based on perception) and top-down inferences (e.g. about object presence or place affordances based on semantic descriptions). The resulting knowledge base constitutes a more complete (albeit uncertain) belief state for the planner. In this work, we implement this component using a deep generative probabilistic model based on Sum-Product Networks (see Sec. 5).

4 Realization of DASH for Laser-Range Data

In order to evaluate the architecture of DASH in practice, we provide its initial realization for a mobile robot equipped with a laser-range sensor. We utilize laser-range data to simplify the initial implementation, however the proposed algorithms can be easily extended to include 3D and visual information.

Perceptual Layer

To integrate local laser-range observations in the perceptual layer, we use a common occupancy grid representation. Specifically, we utilized a grid mapping approach based on Rao-Blackwellized particle filters (Grisetti, Stachniss, and Burgard 2007). We crop the resulting grid map to only retain a rectangular fragment of size 10x10m, centered at the current position of the robot. Consequently, we do not require global consistency of the grid map, as long as the local



(a) Corridor (b) Doorway

(c) Small Office (d) Large Office

Fig. 2: Visualization of spatial knowledge represented in the peripersonal layer for sample places of different semantic categories, expressed as both Cartesian and polar occupancy grids.

environment is mapped correctly. This will still result in partial maps (especially when the robot enters a new room), but it will help to accumulate observations over time. During our experiments, the robot was exploring the environment driving with a constant speed, while continuously gathering data and performing inferences based on the current state of the perceptual layer.

Peripersonal Layer

The peripersonal representation for each place is constructed from the current local occupancy grid in the perceptual layer. However, since the scope of the peripersonal representation is limited to the space immediately surrounding the robot and relevant context, we only retain information about the parts of the environment visible from the robot (grid cells that can be raytraced from the robot location). As a result, walls occlude the view and the resulting grid will mostly contain objects present in a single room. In order to include a more complete appearance of the objects, we additionally include observations behind small obstacles, and a small vicinity around every occupied cell visible from the robot (e.g. corners of furniture). Examples of such local occupancy grids can be seen in Fig. 2.

Next, every local grid map is transformed into an egocentric polar representation (compare polar and Cartesian grids in Fig. 2). This encodes high-resolution information about the geometry and objects nearby, and complements it with less-detailed context further away from the robot. Encoding spatial knowledge closer to the robot in more detail is important for understanding the semantics of the exact robot location (for instance when the robot is in a doorway). However, it also relates to how spatial information is used by a robot when planning and executing actions. It is in the vicinity of the robot that higher accuracy of spatial information is required. The polar grids in our implementation assumed radius of $5m$, with angle step of 6.4 degrees and resolution decreasing with the distance from the robot. It is worth noting that lack of evidence resulting from occlusions is explicitly represented in the cells of the polar representation. Such representation of peripersonal layer is clearly a simplification, however one that results from the nature of the laser-range data.

Topological Layer

The topological layer is maintained by a mapping algorithm discretizing continuous space into sets of *places*, *placeholders*, *views*, and *paths*. The goal is to generate an efficient discretization, which supports all the roles of the topological layer, including expression of the global robot pose, representation of affordances related to navigation and exploration, and anchoring of local spatial knowledge to the global space.

The mapping algorithm expands the topological layer incrementally, adding placeholders at neighboring unexplored locations, and connecting them with paths to existing places. Then, once the robot performs an exploration action associated with a specific path, a new place is generated to which a peripersonal representation, as well as place and view descriptors are anchored. At this point, the path between the two places signifies navigation affordance, and is associated with probability based on current, up-to-date information. In order to choose the location for a new placeholder, the algorithm relies upon information contained in the perceptual layer, including detailed local geometry and obstacles.

Similarly to (Chung et al. 2016), we formulate the problem of finding placeholder locations using a probability distribution that models their relevance and suitability. However, instead of sampling locations of all places in the environment at once, we incrementally add placeholders as the robot explores the environment, within the scope of the perceptual layer. Specifically, the probability distribution is modeled as a combination of two components:

$$P(E \mid G) = \frac{1}{Z}\prod_i \phi_I(E_i)\phi_N(\mathscr{E}), \tag{1}$$

where $E_i \in \{0,1\}$ determines the existence of a place at a location i in the perceptual layer, G is the perceptual occupancy grid, and $\mathscr{E}$ is a set of locations of all existing places within the scope of the perceptual representation.

The potential function ϕ_I models suitability of a specific location, and is defined in terms of three potentials calculated from G:

$$\phi_I(E_i) = \phi_O(E_i)(\phi_V(E_i) + \phi_P(E_i) - \phi_V(E_i)\phi_P(E_i)), \tag{2}$$

where:

- ϕ_O ensures that placeholders are created in areas that are safe from collisions with obstacles. It depends on the distance d_o to the nearest obstacle and is calculated similarly to the cost map used on our robot for obstacle avoidance (Marder-Eppstein et al. 2010). ϕ_o equals 0 for distance smaller than the radius r of the robot base and $1 - exp(-\alpha(d_o - r))$ otherwise.

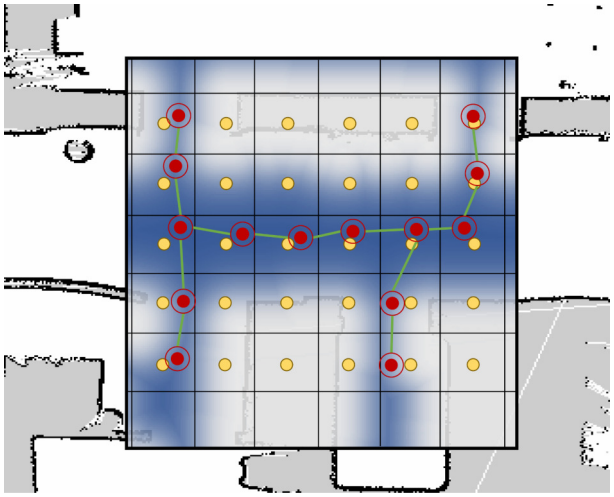


Fig. 3: Visualization of generated places and paths on top of the knowledge in the perceptual layer. The highlighted region corresponds to the spatial scope of the perceptual representation and displays the value of the potential ϕ_I. The low-resolution lattice is illustrated using yellow points, and red points indicate the final, optimized locations of places. Paths highlighted in green afford navigability throughout the environment.

- $\phi_V = exp(-\gamma d_c)$ depends on the distance d_c to the nearest node of a Voronoi graph of the 2D map. This promotes centrally located places that are often preferred for navigation.
- ϕ_P promotes places inside narrow passages (e.g. doors). The potential is generated by convolving the local map with a circular 2D filter of a radius corresponding to an average width of a door.

Overall, ϕ_I ensures that placeholders are located only in areas that are safe and preferred for navigation, and constitute useful anchors for information stored in other layers of the representation. The potential ϕ_N, models the neighborhood of a place and guarantees that places are evenly spread throughout the environment. To this end, the potential function promotes positions at a certain distance d_n from existing places:

$$\phi_N(E_i) = \sum_{p \in \mathscr{E}} e^{-\frac{(d(i,p)-d_n)^2}{2\sigma^2}},$$

where $d(i,p)$ is a Euclidean distance between the potential new place and an existing place.

Final location of new placeholders is chosen through MPE inference in $P(E \mid G)$. However, before adding a new placeholder to the map it is important to verify whether the robot will be able to navigate to it. To this end, we perform an A* search directly over the potential function, and quantify the navigability based on the accumulated potential. Only then, a *path* is created between an existing place and a placeholder. Similarly, the accumulated potential is used to quantify navigability of paths between existing places.

In order to incorporate knowledge about coarse global geometry into the topological representation, we further relate placeholders and places to a global low-resolution lattice (0.8m distance between points in our experiments), as illustrated in Fig. 3. As the robot moves through the environment, the lattice is extended, while preserving consistency with existing points. We assume that a place must be associated with a point of the lattice, and each lattice point can be associated with only one place. As a result, when performing MPE inference using $P(E \mid G)$, we assume that only one place might exist in a cell of a Voronoi tessellation established by the points of the lattice. The resulting set of placeholders (and eventually places) will uniquely correspond to lattice points, yet be created only in locations which are suitable, and can serve as navigation goals for the lower-level controller.

For each place that is created from a placeholder, we generate a set of eight *views*. The views are a discrete representation of the heading of the robot when located at a place, and are assumed to be vectors pointing from a point of the lattice to the eight immediately neighboring points. Since, places are associated uniquely with lattice points, each view will naturally point in the direction of only one neighboring place. As a result, each *path* connecting a place to another place or placeholder will be associated with a specific view.

Semantic Layer

In our initial implementation, the semantic layer captures the information about semantic categories of places in the topological map. This includes categories of rooms in which places are located, such as an office or a corridor, but also a functional place category corresponding to places located in a doorway. The layer is implemented as a simple relational data structure assigning place instances to semantic categories in the ontology of the deep default knowledge model. Each such relation is associated with probability value. Additionally, for each place, the layer captures the likelihood of the peripersonal representation of the place being observed for any of the semantic categories. That likelihood is used to detect and explicitly represent that a place belongs to a novel category, i.e. one that is not recognized by the default knowledge model.

5 Representing Default Knowledge

In our implementation, default knowledge is modeled using a recently proposed Deep Generative Spatial Model (DGSM) (Pronobis and Rao 2017), a probabilistic deep model which learns a joint distribution over spatial knowledge represented at multiple levels of abstraction. We apply the deep model to capture generic spatial concepts and relations between knowledge represented in peripersonal, topological, and semantic layers. Once learned, it enables a wide range of probabilistic inferences. First, based on the knowledge in the peripersonal layer, it can infer descriptors of views and places, as well as semantic categories of places. Moreover, it can detect that a place belongs to a novel category, not known during training. Inference can also be performed over the contents of the peripersonal representation. The model can infer missing geometry information resulting

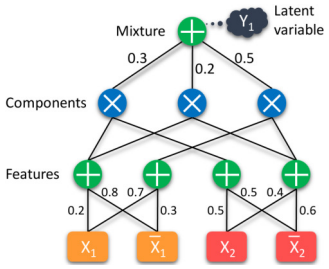


Fig. 4: An SPN for a naive Bayes mixture model $P(X_1, X_2)$, with three components over two binary variables. The bottom layer consists of indicators for each of the two variables. Weights are attached to inputs of sums. Y_1 represents a latent variable marginalized out by the top sum node.

from partial observations and generate prototypical peripersonal representations based on semantic information.

To this end, DGSM leverages Sum-Product Networks (SPNs), a novel probabilistic deep architecture (Poon and Domingos 2011; Peharz et al. 2015), and a unique structure matching the hierarchy of representations in DASH. Below, we give a primer on Sum-Product Networks and describe the details of the architecture of the DGSM model.

Sum-Product Networks

Sum-product networks are a recently proposed probabilistic deep architecture with several appealing properties and solid theoretical foundations (Peharz et al. 2015; Poon and Domingos 2011; Gens and Domingos 2012). One of the primary limitations of probabilistic graphical models is the complexity of their partition function, often requiring complex approximate inference in the presence of non-convex likelihood functions. In contrast, SPNs represent probability distributions with partition functions that are guaranteed to be tractable, involve a polynomial number of sums and product operations, permitting exact inference. While not all probability distributions can be encoded by polynomial-sized SPNs, recent experiments in several domains show that the class of distributions modeled by SPNs is sufficient for many real-world problems, offering real-time efficiency.

SPNs model a joint or conditional probability distribution and can be learned both generatively (Poon and Domingos 2011) and discriminatively (Gens and Domingos 2012) using Expectation Maximization (EM) or gradient descent. They are a deep, hierarchical representation, capable of representing context-specific independence. As shown in Fig. 4 on a simple example of a naive Bayes mixture model, the network is a generalized directed acyclic graph of alternating layers of weighted sum and product nodes. The sum nodes can be seen as mixture models, over components defined using product nodes, with weights of each sum representing mixture priors. The latent variables of such mixtures can be made explicit and their values inferred. This technique is often used for classification models where the root sum is a mixture of sub-SPNs representing multiple classes. The bottom layers effectively define features reacting to certain values of indicators for the input variables. Not all possible architectures consisting of sums and products will result in a valid probability distribution. However, following simple constraints on the structure of an SPN will guarantee validity (see (Poon and Domingos 2011; Peharz et al. 2015) for details).

Inference in SPNs is accomplished by an upward pass through the network. Once the indicators are set to represent the evidence, the upward pass will yield the probability of the evidence as the value of the root node. Partial evidence (or missing data) can easily be expressed by setting all indicators for a variable to 1. Moreover, it can be shown (Poon and Domingos 2011) that MPE inference can be performed by replacing all sum nodes with max nodes, while retaining the weights. Then, the indicators of the variables for which the MPE state is inferred are all set to 1 and a standard upward pass is performed. A downward pass then follows which recursively selects the highest valued child of each sum (max) node, and all children of a product node. The indicators selected by this process indicate the MPE state of the variables.

In this work, we learn the SPN using hard EM, which was shown to work well for generative learning (Poon and Domingos 2011) and overcomes the diminishing gradient problem. The reader is referred to (Pronobis and Rao 2017) for details about the learning procedure.

Architecture of DGSM

The architecture of DGSM is based on a generative SPN illustrated in Fig. 5. The model learns a probability distribution $P(C, D^P_1, \ldots, D^P_{N_p}, D^{V_1}_1, \ldots, D^{V_8}_{N_v}, X_1, \ldots, X_{N_x})$, where C represents the semantic category of a place, $D^P_1, \ldots, D^P_{N_p}$ constitute an internal descriptor of the place, $D^{V_1}_1, \ldots, D^{V_8}_{N_v}$ are descriptors of eight views, and $X_1, \ldots, X_C$ are input variables representing the occupancy in each cell of the polar grid of the peripersonal layer. Each occupancy cell is represented by three indicators in the SPN (for empty, occupied and unknown space). These indicators constitute the bottom of the network (orange nodes).

The structure of the model is partially static and partially generated randomly according to the algorithm described in (Pronobis and Rao 2017). The resulting model is a single SPN, which is assembled from three levels of sub-SPNs. First, we begin by splitting the polar grid of the peripersonal layer equally into eight 45 degree parts, corresponding to the *views* defined in the topological layer. For each view, we randomly generate a sub-SPN over the subset of X_i representing the occupancy within the view, as well as latent variables $D^{V_i}_1, \ldots, D^{V_i}_{N_v}$ serving as an internal view descriptor. The sub-SPN can be seen as a mixture model consisting of 14 components in our implementation. In the second level, we use the distributions defining the components from each view ($8 * 14$ in total) as inputs, and generate random SPNs representing each of the semantic place classes in the ontology. Each of such SPNs is itself a mixture model with the latent variable D^P_i being part of the place descriptor. Finally, in the third level, the sub-SPNs for place classes are combined by a sum node (mixture) forming the root of the whole network. The latent variable associated with the root node is

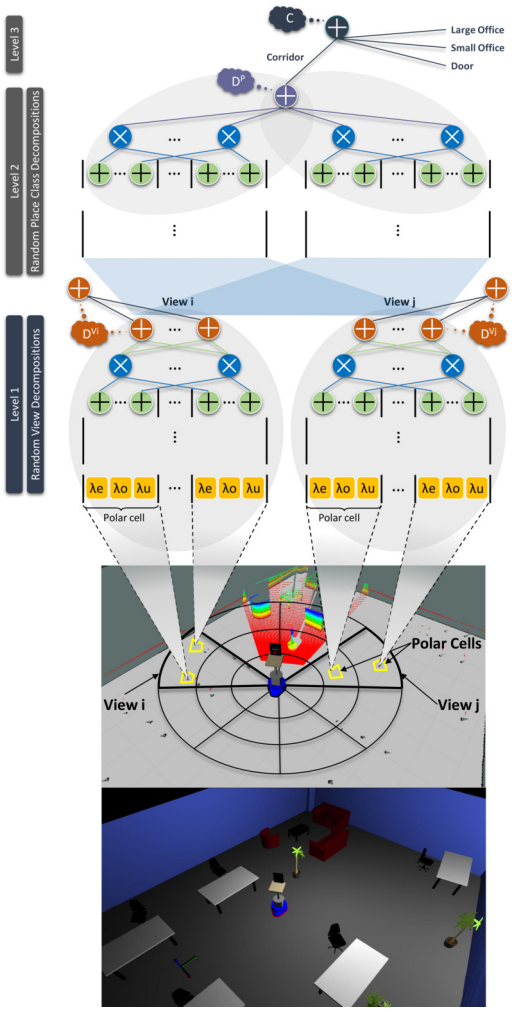


Fig. 5: The structure of the SPN implementing our spatial model. The bottom images illustrate a robot in an environment and a robocentric polar grid formed around the robot. The SPN is built on top of the variables representing the occupancy in the polar grid.

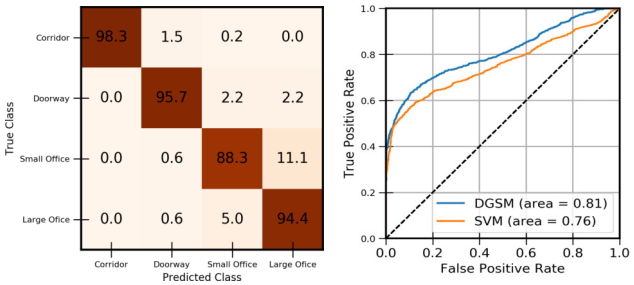
C and is set to the appropriate class label during learning. Overall, such decomposition allows us to use networks of different complexity for representing lower-level features of each view and for modeling the top composition of views into place classes.

6 Experimental Evaluation

Our experimental evaluation consists of two parts. First, we evaluated the ability of the deep default knowledge model implemented with DGSM to perform both top-down and bottom-up inferences across the layers of the representation. Then, we deployed our complete implementation of DASH in order to build representations of large-scale environments.

Experimental Setup

Our experiments were performed on laser range data from the COLD-Stockholm database (Pronobis and Jensfelt



(a) (b)

Fig. 6: Results of experiments with bottom-up inference: (a) normalized confusion matrices for semantic place categorization; (b) ROC curves for novelty detection (inliers are considered positive, while novel samples are negative).

2012). The database contains multiple data sequences captured using a mobile robot navigating with constant speed through four different floors of an office building. On each floor, the robot navigates through rooms of different semantic categories. Four of the room categories contain multiple room instances, evenly distributed across floors. There are 9 different *large offices*, 8 different *small offices*, 4 long *corridors* (1 per floor, with varying appearance in different parts), and multiple examples of observations captured when the robot was moving through *doorways*. The dataset features several other room categories: an elevator, a living room, a meeting room, a large meeting room, and a kitchen. However, with only one or two room instances in each. Therefore, we decided to use the four categories with multiple room instances for the majority of the experiments and designated the remaining classes as novel when testing novelty detection.

To ensure variability between the training and testing sets, we split the samples from the four room categories four times, each time training the model on samples from three floors and leaving one floor out for testing. The presented results are averaged over the four splits.

Bottom-up Inference

First, we evaluated the ability of DGSM to infer semantic place categories given information in the peripersonal layer. As a comparison, we used a well-established model based on an SVM and geometric features (Mozos, Stachniss, and Burgard 2005; Pronobis et al. 2010a). The features were extracted from laser scans raytraced in the same local Cartesian grid maps used to form polar grids of the peripersonal layer. We raytraced the scans in high-resolution maps (2cm/pixel), to obtain 362 beams around the robot. To ensure the best SVM result, we used an RBF kernel and selected the kernel and learning parameters directly on the test sets.

The models were trained with peripersonal representations obtained for locations on three floors in places belonging to four place categories, and evaluated on the fourth floor or using data from rooms designated as novel. The classification rate averaged over all classes (giving equal importance

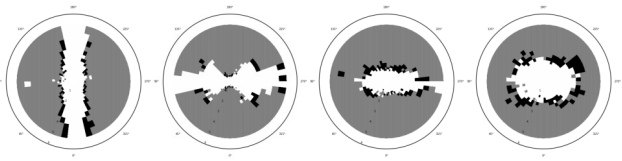
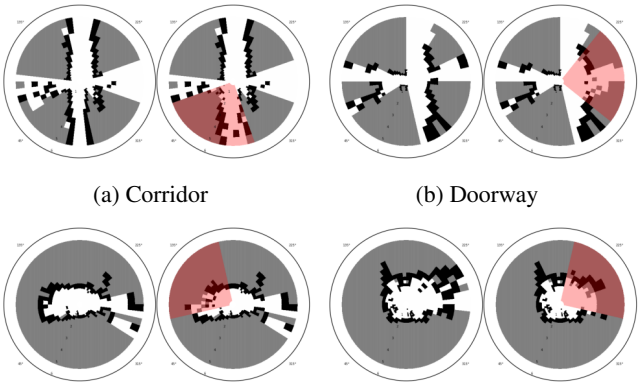


Fig. 7: Prototypical peripersonal representations inferred from semantic place category.



(a) Corridor (b) Doorway

(c) Small Office (d) Large Office

Fig. 8: Examples of completions of peripersonal representations with missing data grouped by true semantic category.

to each class) and data splits was $85.9\% \pm 5.4$ for SVM and $92.7\% \pm 6.2$ for DGSM, with DGSM outperforming SVM for every split. The normalized confusion matrix for DGSM is shown in Fig. 6(a). Most of the confusion exists between the small and large office classes. Offices in the dataset often have complex geometry that varies greatly between room instances.

Additionally, we evaluated the quality of the uncertainty measure produced by DGSM and its applicability to detecting novel concepts. To this end, we thresholded the likelihood of the test peripersonal representations produced by DGSM to decide whether the robot is located in a place belonging to a class known during training. We compared to a one-class SVM with an RBF kernel trained on the geometric features. The cumulative ROC curve for the novelty detection experiments over all data splits is shown in Fig. 6(b). We see that DGSM offers a significantly more reliable novelty signal, with AUC of 0.81 compared to 0.76 for SVM.

Top-down Inference

In the second experiment, we used DGSM to perform inference in the opposite direction, and infer values of cells in the peripersonal representation. First, we inferred complete, prototypical peripersonal representations of places knowing only place semantic categories. The generated polar occupancy grids are shown in in Fig. 7a-d. We can compare the plots to the true examples depicted in Fig. 2. We can see that each polar grid is very characteristic of the class from which it was generated. The corridor is an elongated structure with walls on either side, and the doorway is depicted as a narrow structure with empty space on both sides. Despite the fact that, as shown in Fig. 2, large variability exists between the instances of offices within the same category, the generated observations of small and large offices clearly indicate a distinctive size and shape.

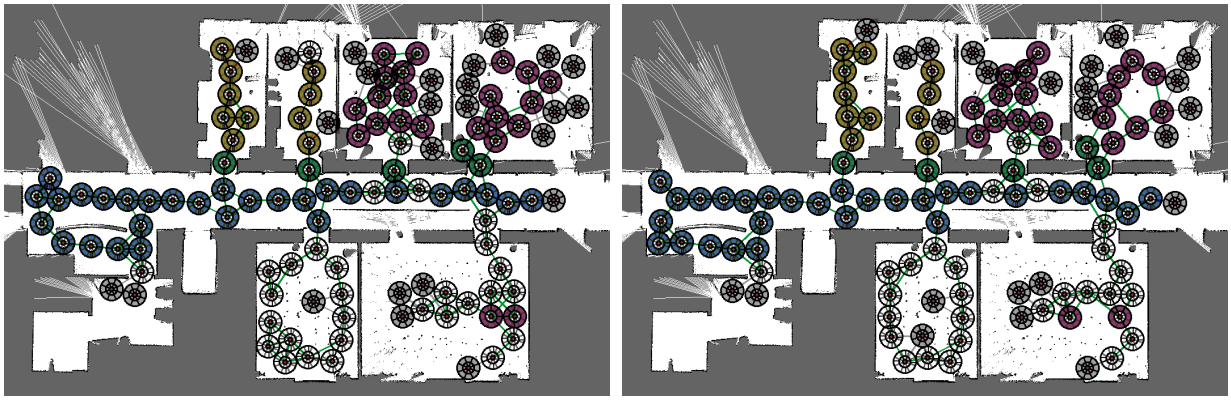
Then, we used DGSM to generate missing values in partial observations of places. To this end, we masked a random 90-degree view in each test polar grid (25% of the grid cells). All indicators for the masked polar cells were set to 1 to indicate missing evidence and MPE inference followed. Fig. 8 shows examples of peripersonal representations filled with predicted information to replace the missing values. Overall, when averaged over all test examples and data splits, DGSM correctly reconstructed $77.14\% \pm 1.04$ of masked cells. This demonstrates its generative potential.

Representing Large-Scale Space

In our final experiment, we deployed the complete implementation of DASH and evaluated its ability to build comprehensive, multi-layered representations of large-scale space. Specifically, we tasked it with representing the 5-*th* and 7-*th* floor of the office building in the COLD-dataset, which measure respectively 298 and 435 square meters. In each case, we incrementally built the representation based on the sensory data captured as the robot navigated through the environment. We relied on the perceptual layer to perform low-level integration of observed laser scans, on peripersonal layer to capture local place information, the topological layer to maintain a consistent topological graph expressing navigability and knowledge gaps related to unexplored space, and finally on the semantic layer to encode information about semantic categories of places, including detections of novel semantic categories.

Fig. 9 illustrates the state of the representation after two completed runs over the 5-*th* floor. The figure presents the final topological graph of places visited by the robot, paths expressing navigability between them, as well as paths leading to placeholders representing possibility of further exploration. For each place, we use color to illustrate the inferred semantic category, or detection of a novel category. First, we can observe that places are evenly distributed across the environment and exist in locations which are relevant for navigation or significant due to their semantics (e.g. in doorways). Moreover, the graphs created during different runs are similar and largely consistent. Second, the semantic place categories inferred by DGSM agree with the ground truth when the category of the place was recognized as known. To detect novel classes, we again thresholded the estimates of the likelihood of the peripersonal representations provided by DGSM. On the 5-*th* floor, the novel category was "meeting room" and two meeting rooms are shown in the bottom part of the map. Although both false positives and false negatives exist, places in both meeting rooms are largely correctly classified as belonging to novel categories.

Fig. 10 shows results for a different environment, the 7-*th* floor. Similar observations can be made as for the 5-*th* floor. However, here the novelty detection is less accurate. DGSM correctly detects the places in the elevator as novel (marked



(a) Run #1 (b) Run #2

Fig. 9: Contents of the topological and semantic layers after two different runs over 5-*th* floor. Gray nodes represent placeholders, while blank nodes indicate places detected as belonging to novel categories. Colors indicate recognized semantic place categories: blue for a corridor, green for a doorway, yellow for a small office, and magenta for a large office. The two large bottom rooms belong to a novel category: "meeting room".

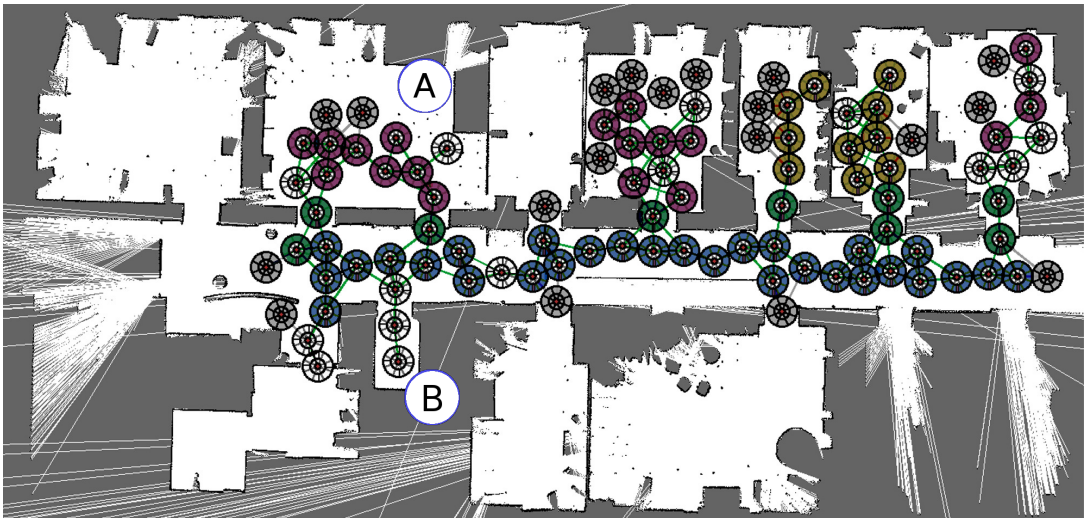


Fig. 10: Contents of the topological and semantic layers after a single run over the 7-*th* floor. Gray nodes represent placeholders, while blank nodes indicate places detected as belonging to novel categories. Colors indicate recognized semantic place categories: blue for a corridor, green for a doorway, yellow for a small office, and magenta for a large office. The rooms marked with letters A and B belong to novel categories: "living-room" and "elevator".

with "B" in the figure), but fails to detect novelty in the living room ("A" in the figure), which instead is misclassified as "large office". While not a desirable outcome, it is not surprising, given the similarity between the living room and large offices in the dataset when observed solely using laser range sensors.

7 Conclusions and Future Work

This paper presented Deep Spatial Affordance Hierarchy, a representation of spatial knowledge, designed specifically to represent the belief about the state of the world and spatial affordances for a planning algorithm on a mobile robot. We demonstrated that an implementation following the principles of DASH can successfully learn general spatial concepts at multiple levels of abstraction, and utilize them to obtain a complete and comprehensive model of the robot environment, even for a relatively simple sensory input. The natural direction for future work is to extend our implementation to include more complex perceptions provided by visual and depth sensors. Additionally, we intend to train the deep model of default knowledge to directly predict complex place affordances related to human-robot interaction. Finally, we are working to integrate our implementation of DASH with a deep hierarchical planning approach to evaluate its capacity to support autonomous robot behavior in complex realistic scenarios.

References

Aydemir, A.; Pronobis, A.; Gbelbecker, M.; and Jensfelt, P. 2013. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics* 29(4):986–1002.

Balaguer, J.; Spiers, H.; Hassabis, D.; and Summerfield, C. 2016. Neural mechanisms of hierarchical planning in a virtual subway network. *Neuron* 90(4):893 – 903.

Chung, M. J.-Y.; Pronobis, A.; Cakmak, M.; Fox, D.; and Rao, R. P. N. 2016. Autonomous question answering with mobile robots in human-populated environments. In *Proc. of IROS*.

Davis, R.; Shrobe, H.; and Szolovits, P. 1993. What is a knowledge representation. *AI Magazine* 14(1).

Gens, R., and Domingos, P. 2012. Discriminative learning of sum-product networks. In *Proc. of NIPS*.

Grisetti, G.; Stachniss, C.; and Burgard, W. 2007. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics* 23(1).

Hanheide, M.; Göbelbecker, M.; Horn, G. S.; Pronobis, A.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; Zender, H.; Kruijff, G.-J.; Hawes, N.; and Wyatt, J. L. 2016. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*.

Hawes, N.; Zender, H.; Sj, K.; Brenner, M.; Kruijff, G.-J.; and Jensfelt, P. 2009. Planning and acting with an integrated sense of space. In *Proc. of the International Workshop on Hybrid Control of Autonomous Systems*.

Holmes, N. P., and Spence, C. 2004. The body schema and multisensory representation(s) of peripersonal space. *Cognitive processing* 5(2).

Kuipers, B. 2000. The spatial semantic hierarchy. *Artificial intelligence* 119(1-2).

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17(39):1–40.

Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B.; and Konolige, K. 2010. The office marathon: Robust navigation in an indoor office environment. In *Proc. of ICRA*.

Mozos, O. M.; Stachniss, C.; and Burgard, W. 2005. Supervised learning of places from range data using AdaBoost. In *Proc. of ICRA*.

Peharz, R.; Tschiatschek, S.; Pernkopf, F.; and Domingos, P. 2015. On theoretical properties of Sum-product Networks. In *Proc. of AISTATS*.

Poon, H., and Domingos, P. 2011. Sum-product networks: A new deep architecture. In *Proc. of UAI*.

Pronobis, A., and Jensfelt, P. 2012. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *Proc. of ICRA*.

Pronobis, A., and Rao, R. P. N. 2017. Learning deep generative spatial models for mobile robots. arXiv:1610.02627 [cs.RO].

Pronobis, A.; Mozos, O. M.; Caputo, B.; and Jensfelt, P. 2010a. Multi-modal semantic place classification. *The International Journal of Robotics Research* 29(2-3).

Pronobis, A.; Sjöö, K.; Aydemir, A.; Bishop, A. N.; and Jensfelt, P. 2010b. Representing spatial knowledge in mobile cognitive systems. In *Proc. of the International Conference on Intelligent Autonomous Systems (IAS-11)*.

Thrun, S.; Bücken, A.; Burgard, W.; Fox, D.; Fröhlinghaus, T.; Henning, D.; Hofmann, T.; Krell, M.; and Schmidt, T. 1998. Map learning and high-speed navigation in RHINO. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.

Zender, H.; Mozos, O. M.; Jensfelt, P.; Kruijff, G.; and Burgard, W. 2008. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems* 56(6). Special Issue "From Sensors to Human Spatial Concepts".

Dynamics-Aware Reactive Planning for Unmanned Ground Vehicles to Avoid Collisions with Dynamic Obstacles on Uneven Terrains

Pradeep Rajendran and **Brual C. Shah** and **Satyandra K. Gupta**
Department of Aerospace and Mechanical Engineering, Viterbi School of Engineering
University of Southern California, Los Angeles, CA 90007, USA

Abstract

Collision avoidance is a key capability for autonomous ground vehicles and must respect the dynamic constraints of the vehicle. Many recent approaches for avoiding collisions with dynamic obstacles respect the dynamic constraints of the vehicle. However, as the terrain changes, dynamic constraints imposed on the vehicle also change. A common practice is to assume conservative dynamics constraints that work on all terrains. This reduces overall mission performance. In this paper, we present a real-time dynamics-aware reactive trajectory generator which produces trajectories that avoid collisions with dynamic obstacles under varying dynamic constraints. The trajectory generator considers modifications to the intended path, generating alternatives in real-time. It also considers regulating speed along the various modified paths, finding a trajectory that avoids collision while minimizing deviation from the intended trajectory. It accounts for uncertainty in the obstacle position and velocity while evaluating the alternative trajectories and uses conservative estimates of the vehicle's dynamic constraints to ensure collision risk is minimized. It also uses "deferred value binding", to exploit more accurate estimates of the dynamic obstacle states as obstacles approach the vehicle. It automatically adjusts the number of options being evaluated based on the estimated time to collision. In order to ensure real-time performance, the trajectory generator handles multiple dynamic obstacles by either grouping them into a single composite obstacle in the configuration space or deals with them sequentially by prioritizing obstacles based on the estimated time to collision. We present simulation results to show that the planner is able to effectively deal with the dynamic obstacles on terrains with varying slopes.

1 Introduction

The ability to successfully avoid collisions with dynamic obstacles is a fundamental capability needed to realize autonomous unmanned ground vehicles (UGVs). The collision avoidance approach must take into account vehicle's performance constraints, such as maximum achievable velocities, accelerations, braking distances, and turning radii. Collision avoidance with dynamic obstacles on uneven terrains is challenging because a vehicle's performance constraints change based on the vehicle state, terrain characteristics, terrain slope, and vehicle's orientation with respect to the terrain slope. Consider the stopping distance constraint as an example. The stopping distance constraint cannot be defined just based on the vehicle characteristics alone. The stopping distance changes based on vehicle's velocity. It also depends on the traction available on the terrain. The slope of the terrain affects this as well. Finally, the stopping distance is shorter if the vehicle is traveling uphill compared to the same vehicle traveling downhill. Thus, the planner must be dynamics-aware and use accurate estimates of vehicle's dynamic constraints based on the current state of the vehicle and the terrain.

Using overly conservative dynamics constraints can avoid collisions by stopping or steering the vehicle far from the obstacles, but lead to significant deviations from the intended trajectories and compromise mission performance. By contrast, using constraints that overestimate vehicle's capabilities leads to an increased risk of collision. Therefore, using accurate constraints is important during the generation of reactive plans to avoid collision with dynamic obstacles.

In order to avoid collision with dynamic obstacles, the vehicle must consider modifications to the intended path by generating path alternatives and consider regulating speed along the alternative paths to find a trajectory that avoids collision and minimizes deviations from the planned nominal trajectory. This trajectory modification must be done in real-time. Based on the available time, the planner should then automatically adjust the number of options that it evaluates to ensure that computation time is tractable and does not reduce the reaction time available to the vehicle.

Finally, the reactive planner needs to account for uncertainty in the obstacle position and velocity when evaluating trajectories for collision risks. The uncertainty in obstacle position and velocity reduces as obstacles approach the vehicle. The planner should delay committing to a specific trajectory until the last possible moment to fully exploit the reduction in uncertainty in obstacle position and velocity.

Dynamic obstacle collision avoidance is a well-studied problem (please see Section 2 for a detailed discussion). Popular methods include Generalized Velocity Obstacles (GVO) (Wilkie, van den Berg, and Manocha 2009) and its variants. In this paper, we present a method that is real-time, dynamics-aware and generates trajectories which avoid collision with dynamic obstacles. It works by blending the path selection and the speed profile selection. Conceptually, our method can be viewed in the same family as GVO. Our method differs from GVO in the way control/input space is

partitioned. In GVO, control-space sampling and collision detection determines the available feasible control actions. These control actions impart changes to both the path and speed of the UGV. So, a dense control-space sampling is required to explore a variety of trajectories. In addition, one way time-varying dynamics constraints can be incorporated into the selection of control actions is to use costly forward simulations of at least a low-fidelity UGV dynamics model to verify feasibility of the control actions. Otherwise, it is customary to assume an overly conservative UGV capability and use conservative control actions resulting in degradation in mission performance in some cases. In our method, we decouple trajectory generation using the standard approach of geometric path selection followed by speed profile selection. As a result, the geometric paths can be more spatially focused for collision avoidance avoiding the need for dense control-space sampling. Furthermore, the variation in dynamics constraints along these paths is easier to incorporate as a function of the position of the UGV on the path. For example, a path crossing a slippery or sloped region imposes acceleration constraints that can easily be incorporated into the spatio-temporal configuration space of the path along with the dynamic obstacles. These paths are chosen with the curvature and velocity limits of the UGV.

Our trajectory generator accounts for uncertainty in the obstacle position and velocities when evaluating alternative trajectories for collision risks and uses conservative values of estimated dynamics constraints to ensure that the risk of collision is minimized. It also uses deferred value binding to utilize improved estimates of obstacles' states as they come closer to the vehicle. Finally, it automatically adjusts the number of options being evaluated based on the estimated time to collision. A vehicle may encounter multiple dynamic obstacles. In order to maintain real-time performance, the approach presented in this paper either groups multiple obstacles into a single composite obstacle in spatio-temporal configuration space or deals with them sequentially (i.e. prioritizing obstacles based on the estimated time to collision). This approach works well for both cases when obstacles are too close or too far apart.

2 Related Work

Trajectory planning for unmanned ground vehicles (UGVs) is an extensively studied problem in robotics. This paper will be focusing on trajectory planning approaches that deal with dynamic obstacles.

Local obstacle avoidance paradigms such as potential field method, vector field histogram (Borenstein and Koren 1991) and nearness diagram method (Minguez and Montano 2000) primarily consider the instantaneous position of the obstacles while computing collision avoidance strategies. Using just instantaneous position and purely reactive strategies result in undesirable oscillatory behavior. These methods are termed as zero-order methods as they only consider position and not the velocity of obstacles.

Inevitable collision states (ICS) (Fraichard and Asama 2003) is an ideal method that provides provable zero collision guarantees. In this method, states of the robot in which no feasible action may be performed to avoid collision are identified and deliberately avoided. Variants of ICS such as (Martinez-Gomez and Fraichard 2009) and (Blaich et al. 2015) differ in how much the three criteria for motion safety (Fraichard 2007) is relaxed (i.e. knowledge of vehicle dynamics, future environment/obstacles and ability to reason over infinite time horizon). The precise characterization of ICS is computationally expensive even for low-dimensional dynamics and hence, prohibitive for real-time applications.

Other methods such as Dynamic Window Approach (DWA) (Fox, Burgard, and Thrun 1997) and Velocity Obstacle (VO) (Fiorini and Shiller 1998) directly work in the velocity space (VS). Hence, these methods are able to incorporate first-order behavior of obstacles and vehicle dynamics to yield viable velocity vectors that avoid obstacles. The original VO formulation models obstacles moving in piece-wise constant velocities between planning windows. Assumptions are relaxed in recent developments of VO such as (Shiller, Large, and Sekhavat 2001; J. van den Berg 2011; Fulgenzi, Spalanzani, and Laugier ; Bareiss and van den Berg 2015). They differ in the representation of obstacles, assumptions related to obstacle trajectories and how they react to impending collision.

Deliberative lattice-based methods used for dynamic obstacle avoidance compute trajectories using a lattice of viable alternatives. This lattice structure is computed using the kino-dynamic model of the vehicle. The approach presented in (Brock, Trinkle, and Ramos 2009) plans the trajectory in a 4D lattice structure. Computational efficiency is retained by adaptive variation of lattice resolution and by using the Anytime Dynamic A* (AD*) (Likhachev et al. 2005) algorithm for graph search and by using environment constrained heuristics to guide the search. Similar methods have been developed for unmanned surface vehicles in (Shah et al. 2015), where the motion primitives used during the graph search are scaled proportionate to the congestion in the environment.

While dynamics constraints are respected in GVO, ICS and some of the recent approaches (Shimoda, Kuroda, and Iagnemma 2005; Spenko et al. 2006; Howard and Kelly 2007; Iagnemma, Shimoda, and Shiller 2008; Werling et al. 2010), they do not explicitly consider (1) reference trajectory deviation, (2) time-varying dynamics constraints due to uneven terrain. Collision avoidance alone without regard to reference path deviation may compromise mission performance. In this work, an obstacle avoidance approach that explicitly reasons about reference trajectory deviation, varying dynamics constraints while minimizing collision risk is presented.

3 System Architecture

Autonomous operation of an Unmanned Ground Vehicle (UGV) requires a path and trajectory planner that respects both the vehicle's kinematic and dynamic constraints (LaValle 2006). Typical UGV missions may span several kilometers and computing the entirety of a dynamically feasible trajectory over such large distances is time consuming and impractical in the face of dynamic obstacles. Thus, a hierarchical planning architecture is used.

Our hierarchical planning architecture is composed of three layers: (i) the Global Path Generator (GPG), (ii) the Trajectory Generator respecting Kinematic Constraints (TGKC) and (iii) the reactive Trajectory Generator respecting Dynamics Constraints (TGDC). Each layer in the architecture has its own planning horizon defined by distance or time. And, each layer sets up a reference trajectory for the layer below. In the case where a reference trajectory imposed by a upper layer is found to be infeasible by the lower layer, the lower layer raises an exception that is handled by the upper layer. In this paper, we employ the GPG described in (Shah and Gupta 2016). It produces a globally optimal, any-angle, geometrically feasible path on a planning horizon which spans several hundred kilometers.

The TGKC computes trajectories respecting the vehicle's kinematic constraints while tracking the geometric path laid out by the GPG. It employs a planning horizon spanning hundreds of meters. This relatively short planning horizon enables it to re-compute plans at higher frequency than the GPG. The TGKC computes a trajectory consisting of a reference path and a reference speed profile along the path. Only static obstacles are taken into consideration in TGKC while the dynamic obstacles are delegated to the TGDC.

The focus of this paper is the development of the third layer of the planning architecture: Trajectory Generation respecting Dynamics Constraints (see Section 5). The TGDC computes trajectories that respect the dynamics constraints of the vehicle and avoids moving/dynamic obstacles over uneven terrains, planning trajectories with a horizon of up to several seconds. Its re-planning frequency is significantly higher than that of the TGKC, which is essential in the presence of dynamic obstacles. While avoiding dynamic obstacles, the TGDC also attempts to follow the trajectory provided by the TGKC as closely as possible, altering the velocity profile on the reference path. Where appropriate, the TGDC also locally alters the reference path prescribed by the TGKC. These alterations are performed in view of vehicle capabilities on uneven terrain. An example is presented in Section 4 to briefly illustrate the trajectory generation process in TGDC.

4 Preliminaries and Notations

Let us assume that TGKC layer has commanded a UGV to reach the top of the ramp in Figure 1. The UGV is on the ramp, attempting to climb it. A dynamic obstacle is poised to cross the intended path of the UGV. In this situation, it is crucial to know if the UGV is capable of accelerating (against gravity) to pass the on-coming obstacle. On the same note, if the UGV happens to be facing down the slope, it is crucial to know if the UGV is capable of slowing down (against gravity) to avoid on-coming obstacles.

Suppose the UGV's throttle actuator is already close to saturation. If we simply assume that the UGV is capable of high acceleration and go full throttle ahead, the UGV may not able to speed up fast enough to avoid the dynamic obstacle and end up colliding with it. Conversely, assuming overly conservative acceleration bounds for the UGV and not accelerating when it is in fact doable results in reference trajectory deviation. This means future commands to the speed

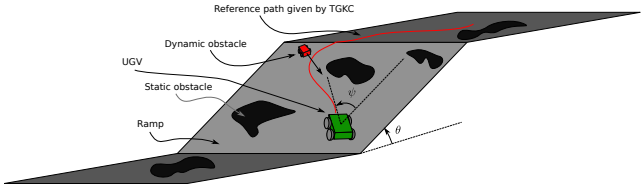


Figure 1: An illustrative example showing the need for vehicle-dynamics aware planning

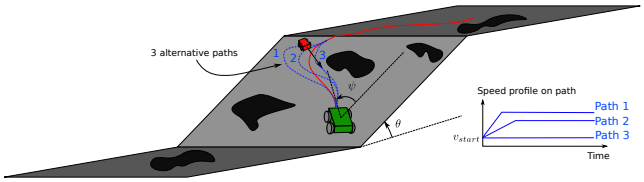


Figure 2: Alternative paths choices are shown. For each path, there can be many associated speed profiles that can circumnavigate the dynamic obstacle in the face of dynamics constraints. Only one speed profile is shown for each path. On each path, a few speed profiles are evaluated in TGDC to yield the best path and speed profile.

of UGV have to be selected carefully. Broadly, the selected speed profiles perform one of three strategies: *pass* obstacles (by rushing ahead along the path before the obstacle crosses the path), *yield* to obstacles (by waiting for obstacles to cross the path) or continue along path with *no change* to speed.

Given that the UGV is already throttle-saturated, what other options are there to reach the top of the ramp quickly and safely? Instead of accelerating, the UGV could try modifying its path slightly to "buy" some time for maneuvering around the dynamic obstacle at the same speed. This means alternative future paths (see Figure 2) have to be selected opportunistically while minimizing the deviation from the reference path provided by TGKC.

The rest of the paper covers the technical details behind the TGDC. Symbols and terms used in the following sections are briefly described below.

- $s \in \mathbb{R}$: Parameter variable in the arc-length parameterization of a curve in 3D space.
- $K(s)$: A curve $K : \mathbb{R} \mapsto \mathbb{R}^3$ in 3D space parameterized by s.
- l_K : Total length of curve K.
- $t \in \mathbb{R}$: Time.
- $v \in \mathbb{R}$: Speed on a curve K.
- Speed profile : This refers to various speeds the UGV travels at along the path. Note that this is slightly misrepresented in the figures as a position vs time function (instead of speed vs time function).
- C_K (or $s-t$ space) : A configuration space C_K defined over s and t. Each point $p_0 = (s_0, t_0) \in C_K \subset \mathbb{R}^2$ in this space represents a particular position s_0 on a certain curve $K(s)$ at a particular time t_0. The subscript K denotes the

association of the space with the curve K. (see Chapter 7.1.3 (LaValle 2006))

- $PW_K(s_0, t_0, l, t_h)$ (or planning window) : A rectangular subset of C_K defined by the rectangle's two diagonal points $p_{lower\_left} = (s_0, t_0)$ and $p_{upper\_right} = (s_0 + l, t_0 + t_h)$ where t_h is the horizon (look-ahead) time and l is the total length of the curve K.
- $\tau_K(t)$ (or $s-t$ trajectory) : A trajectory $\tau_K : \mathbb{R} \mapsto \mathbb{R}$ defined over a planning window PW_K with time t as the independent variable. In other words, this trajectory is a specification of position on the curve K as a function of time t.
- UGV state : A tuple of numbers consisting of the bounding circle radius, position, velocity, orientation and direction (ψ, see Figure 1) with respect to the slope of the terrain.
- $\tau_{accel,K}(t)$: The $s-t$ trajectory on the curve K as a function of time as the UGV accelerates under full throttle along the curve. This function also depends on UGV state.
- $\tau_{decel,K}(t)$: The $s-t$ trajectory on the curve K as a function of time as the UGV decelerates under full braking along the curve. This function also depends on UGV state.
- DO : A tuple of numbers consisting of the bounding circle radius, position and velocity of a dynamic obstacle.
- $x-y-t$ obstacle : This is a projection of a dynamic obstacle onto the local terrain plane (i.e. discarding the z components)
- $C_{K,DO}$ (or $s-t$ obstacle) : This is the set of all points $C_{K,DO} \subset PW_K$ corresponding to a particular $x-y-t$ obstacle moving across the curve K. These points mark the position and time at which the UGV will be in collision with the dynamic obstacle DO. The UGV is in collision if the position of the UGV on the curve is s_{ugv} at t_n and the point $(s_{ugv}, t_n) \in C_{K,DO}$. Thus, $C_{K,DO}$ can computed by sampling points in PW_K and checking if the sampled point corresponds to a collision between the UGV and a dynamic obstacle.
- UL vertex (or upper left vertex) : This is the upper left vertex of a $s-t$ obstacle $C_{K,DO}$. $UL(C_{K,DO}) = (s_{UL}, t_{UL}) \in C_{K,DO}$ where $s_{UL} = \max_{(s,t)\in C_{K,DO}}(s)$ and $t_{UL} = \min_{(s,t)\in C_{K,DO}}(t)$
- LR vertex (or lower right vertex) : This is the lower right vertex of a $s-t$ obstacle $C_{K,DO}$. $LR(C_{K,DO}) = (s_{LR}, t_{LR}) \in C_{K,DO}$ where $s_{LR} = \min_{(s,t)\in C_{K,DO}}(s)$ and $t_{LR} = \max_{(s,t)\in C_{K,DO}}(t)$
- Velocity tuning (or speed regulation) : The process of finding a $s-t$ trajectory τ_K over C_K such that the trajectory does not pass through any $s-t$ obstacle $C_{K,DO}$. (see Chapter 7.1.3 (LaValle 2006))

5 Problem Formulation

Given:

(a) A kinematically feasible, collision free reference path $K_r(s) \in \mathbb{R}^3$ parameterized by the arc-length parameter s and a reference speed profile $v_r(s)$ over the path $K_r(s)$

(b) The current time t_{start}

(c) UGV state consisting of:
- $p_{ugv} \in \mathbb{R}^3$: The current position of the UGV
- $v_{start} \in \mathbb{R}$: The current velocity in direction of motion
- θ : The average slope angle of the terrain under footprint of the UGV
- ψ : The yaw angle with respect to the slope

(d) A look-up table of speed, acceleration and deceleration constraints indexed by θ and ψ

(e) A set of n_{do} dynamic obstacles $D = \{DO_k\}_{k=1}^{n_{do}}$ sensed by the perception system, each defined by:
- Instantaneous position measurement $p_{do} \in \mathbb{R}^3$ and the associated perception variance $\sigma_p \in \mathbb{R}$ following the Gaussian distribution $\mathcal{N}(p_{do}, I_{3\times 3} \cdot \sigma_p)$
- Instantaneous velocity measurement $v_{do} \in \mathbb{R}^3$ and the associated perception variance $\sigma_v \in \mathbb{R}$ following the Gaussian distribution $\mathcal{N}(v_{do}, I_{3\times 3} \cdot \sigma_v)$
- Distance between obstacle and the UGV perception system d
- Noise saturation threshold d_t
- Position uncertainty parameter
  - $\sigma_p(d) = \begin{cases} \sigma_{p,max} \cdot \frac{d}{d_t}, & d < d_t \\ \sigma_{p,max}, & d \geq d_t \end{cases}$
- Velocity uncertainty parameter
  - $\sigma_v(d) = \begin{cases} \sigma_{v,max} \cdot \frac{d}{d_t}, & d < d_t \\ \sigma_{v,max}, & d \geq d_t \end{cases}$

(f) A set of kinematically feasible paths (computed offline) A, which locally modify $K_r(s)$ by connecting p_{ugv} to a common point $K_r(s_{end})$ on the path K_r
- An alternative path: K_m of length l_{K_m} where $K_m(0) = p_{ugv}$ and $K_m(l_{K_m}) = K_r(s_{end})$
- $A(p_{ugv}, s_{end}) = \{K_m(s)\}_{m=1}^{n_{alt}}$

(g) Full-throttle acceleration trajectory for the current UGV state $\tau_{accel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ and full braking deceleration trajectory $\tau_{decel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ defined over C_K space. s_0 denotes the current position of the UGV on $K(s)$. t_0 denotes the current time. Additionally, the tuple (t_0, v_0, s_0) defines an initial condition over $s-t$ space. $\theta(s)$ is the local slope angle of the terrain at the point $K(s)$ and $\psi(s)$ is the angle the tangent at $K(s)$ makes with the slope.

Compute: A dynamically feasible trajectory, which minimizes collision risk and minimizes reference path and reference speed deviation.

6 Approach

The approach used in the TGDC to compute a dynamically feasible plan is outlined in Algorithm 1. The approach is described with reference to the motivational example.

Algorithm 1 Trajectory planning with dynamics constraints

procedure COMPUTEDYNAMICALLYFEASIBLEPLAN
  Depending on time to collision with the nearest obstacle, choose the number of alternative paths n_{alt} to generate.
  Generate the path set $A(p_{ugv}, s_{end})$ based on the current vehicle position and look-ahead distance.
  Obtain the dynamic obstacle state measurements $D = \{DO_k\}$ from the perception system
  for each path $K_m \in A$ **do**
    Construct a planning window PW_{K_m}
    Compute vehicle dynamics constraints τ_{accel,K_m}, τ_{decel,K_m} based on θ and ψ along the path K_m and the UGV state
    Pre-process the reference speed profile on the path using vehicle dynamics constraints to form a dynamically feasible reference trajectory
    for each $DO_k \in D$ **do**
      Obtain n_s samples of DO_k (from the Gaussian distribution) and compute the $s-t$ obstacle samples $\{C_{K_m,DO_k,n}\}_{n=1}^{n_s}$
    end for
    If combining multiple obstacles is appropriate, coalesce $s-t$ obstacles $\{C_{K_m,DO_k,n}\}_{k=1}^{n_{do}}$ to form n_s samples of a single composite $s-t$ obstacle $\{C_{K_m,DO,n}\}_{n=1}^{n_s}$
    for each strategy $\in \{PASS, YIELD, NO\ CHANGE\}$ **do**
      Compute $p_{coll,strategy}$, $c_{strategy}$ using the $s-t$ obstacle samples $\{C_{K_m,DO,n}\}_{n=1}^{n_s}$ using the method in Section 6.4
    end for
    Append the cost tuple $p_{coll,pass}, c_{pass}, p_{coll,yield}, c_{yield}, p_{coll,no\ change}, c_{no\ change}$ for each path option K_m and each of the strategies to the cost matrix
  end for
  Generate a symbolic plan $(K_{best\ path}, c_{best\ strategy})$ consisting of the best path option and best strategy on that path by using criteria in Section 6.5
end procedure

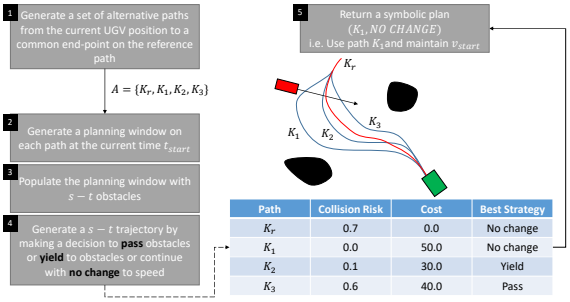


Figure 3: Graphical overview of the approach

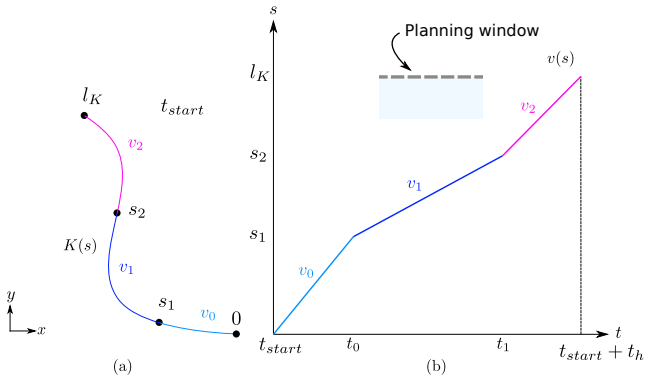


Figure 4: (a) Path K and (b) its corresponding planning window $PW_K(0, t_{start}, l_K, t_h)$ showing a reference speed profile $v(s)$ to follow

6.1 Construction of a Planning Window

Line 3 of Algorithm 1: The kinematically feasible path $K_r(s)$ and speed profile $v_r(s)$ provided by TGKC is used as a reference by the TGDC. They are termed reference path and reference speed profile respectively.

Each time the TGDC is invoked, the TGDC generates a set of paths A. This path generation can be done online or offline. In this work, A is constructed by picking a few paths from a library of pre-computed paths described in (Howard and Kelly 2007). Some of the these paths may be in collision with static obstacles when rotated, translated and placed in front of the UGV. While some other paths may not satisfy the curvature constraints imposed by the vehicle kinematics. These paths are removed from A. The number of alternative paths n_{alt} is scaled based on the time to collision with the next dynamic obstacle. This allows for fast reactive collision avoidance even when available reaction time is limited. In this work, n_{alt} ranges between 2 and 10. Each path option starts at p_{ugv} and ends at $K(s_{end})$. It deviates slightly from reference path midway. Speed profile on the alternative path could be specified using the dynamic trajectory space (Spenko et al. 2006) or adapted from the reference speed profile.

For example, in Figure 1 and 3, $A = \{K_r, K_1, K_2, K_3\}$.

K_r will not be included if the UGV happens to be significantly away from K_r. Each path in A is such that it starts at the current position of the UGV and ends at a common point $K_r(s_{end})$ where $s_{end} = s_{start} + l_{deviation}$. s_{start} is parameter value on K_r such that $K_r(s_{start})$ is the closest point on K_r to the current UGV position. $l_{deviation}$ is the distance over K_r in which the UGV will be deviating from the reference path K_r. $l_{deviation}$ typically has to scale with the obstacle size. In this work, $l_{deviation}$ was fixed to 10 m as obstacles were of a fixed size. Note that $K_m(l_{K_m}) = K_r(s_{end})$. For each path $K_m \in A$, a planning window $PW_{K_m}(0, t_{start}, l_{K_m}, t_h)$ is constructed (see Figure 4b). Thus, the planning window is t_h wide starting at the current time t_{start} and l_{K_m} tall starting at 0.

So far, the planning windows are all blank. The next step is to populate each window with dynamic constraints.

6.2 Construction of Dynamics Constraints

Line 7: The terrain type (i.e. the local slope at the vehicle footprint) is used to compute the associated dynamics constraints. In this work, a simple look-up table (Table 1) is used to determine parameters describing the acceleration and deceleration capabilities (1) of the vehicle in a particular terrain type. Acceleration and deceleration constraints are specified via trajectories in the planning window. Terrain parameters and the orientation of the vehicle on the terrain affect the local acceleration and deceleration constraints. In this work, only the local slope θ under the vehicle's footprint and the direction ψ with respect to the slope are used to form equations of acceleration and deceleration. However, other parameters affecting the powertrain can be easily incorporated into these equations (Spenko et al. 2006).

The upper bound on acceleration a^+ and the upper bound on deceleration a^- used in this work are shown in (1) and Table 1. For a certain initial state (s_0, t_0, v_0) and a path K, an acceleration trajectory $\tau_{accel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ and a deceleration trajectory $\tau_{decel,K}(t, t_0, v_0, s_0, \theta(s), \psi(s))$ can be computed (see Figure 5) as the double integral of the equations describing upper bounds on acceleration and deceleration along the path. Note that θ and ψ change along the path K.

Table 1: Scaling parameters for each terrain type

Parameter	Slope angle θ (°)		
	-10	0	10
v_{max}(m/s)	2.0	1.8	1.6
k	$0.3 + 0.2c_\psi$	0.3	$0.3 - 0.2c_\psi$
a_{min}(m/s^2)	$-0.2 + 0.1c_\psi$	-0.2	$-0.2 - 0.1c_\psi$

$$a^+(v) = k(v_{max} - v)$$
$$a^- = a_{min} \quad (1)$$

The acceleration trajectory (blue) shows the resulting $s-t$ trajectory if maximum throttle were to be applied from the initial state (s_0, t_0, v_0) onwards. Similarly, the deceleration trajectory (red) shows the resulting $s-t$ trajectory if maximum braking were to be applied from the initial state. Thus,

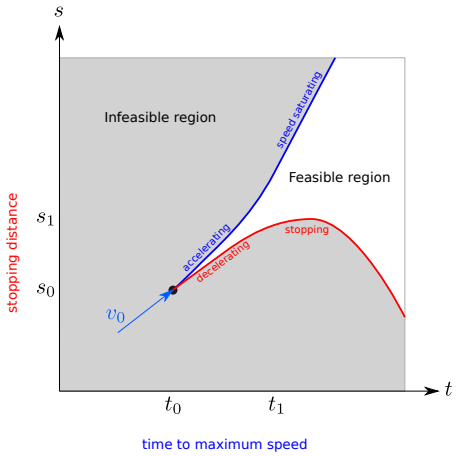


Figure 5: Maximum speed, acceleration and deceleration constraints are trajectories in the planning window corresponding to a certain curve.

the region above the acceleration curve and the region below the deceleration curve are not *reachable* from the initial state. In this context, a target point in $s-t$ space is considered reachable if there is a continuous function between the initial state and the target point that does not pass through $s-t$ obstacles or the infeasible region.

At this point, planning windows have dynamic constraints. The next step is to populate them with dynamic obstacles.

6.3 Construction of $s-t$ Obstacles

Line 8: A given reference speed though kinematically feasible, may be dynamically infeasible when vehicle is on certain types of terrain (e.g. high-speed over uphill terrain). In such circumstances, the reference speed profile is capped at the maximum speeds allowed by the local terrains in the planning window. This pre-processing step ensures that reference speed profile is reasonably altered for the terrain conditions but not necessarily collision free.

Line 10: Collision avoidance using the velocity tuning method requires dynamic obstacles to be translated from $x-y-t$ space to the planning window PW_{K_m}. Dynamic obstacles that cut across the reference path are termed $x-y-t$ space obstacles henceforth. Figures 7 and 6 collectively illustrate the relationship between $x-y-t$ space and $s-t$ space. For example, the $s-t$ obstacle $C_{K,DO}$ corresponding to the dynamic obstacle cutting across the path K in Figure 6 is computed by moving the inflated obstacle time step by time step from $t = t_0$ to $t = t_1$ and noting the intersection between the inflated obstacle and the path. Thus, in this case, for the path K, $C_{K,DO}$ is $[s_a, s_b] \times [t_0, t_1]$ as shown in Figure 7. Note that the subscript m is omitted from K_m for notational simplicity. However, it is to be understood that K has to be replaced with each of the paths in A to yield path specific $s-t$ obstacles such as $C_{K_m,DO}$.

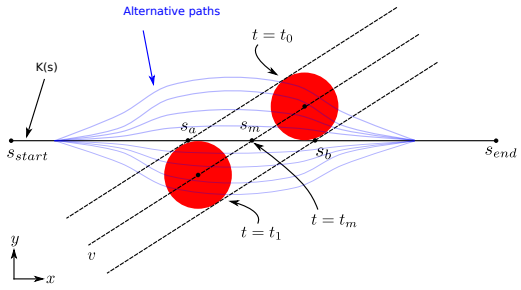


Figure 6: Snapshots of an inflated (vehicle radius + obstacle radius) $x-y-t$ space obstacle as it passes diagonally across a few paths at velocity v. (also see Figure 7)

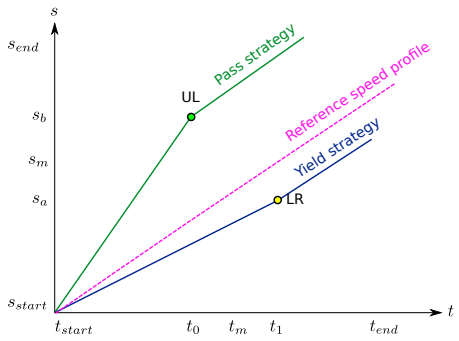


Figure 7: A planning window in which a $s-t$ space obstacle (dark red) and its bounding box approximation (light red) are shown. The reference speed profile is infeasible as it intersects the $s-t$ obstacle. Two trajectories start at (s_{start}, t_{start}) in the planning window. The passing trajectory (green) accelerates in front of the obstacle using the upper left (UL) vertex. The yielding trajectory (purple) decelerates and yields to the obstacle using lower right (LR) vertex. (also see Figure 6)

6.4 Evaluating Strategies for Obstacle Avoidance

Given a $s-t$ obstacle, a schematic view of two possible trajectories is illustrated in Figure 7. The two possible trajectories, *pass* and *yield*, are computed over the time range $[t_{start}, t_{end}]$. The two classes of speed profiles maneuvering around the $s-t$ obstacles are termed *strategies*. When the UGV executes the *pass* strategy, the UGV passes in the front of the dynamic obstacle before tracking reference speed. When the UGV executes the *yield* strategy, it slows down to pass the dynamic obstacle before tracking reference speed. There is a third strategy *no change* that is not shown. This strategy is applicable when reference speed profile does not intersect with the $s-t$ obstacle and hence, the UGV can follow the speed profile with no changes.

Multiple dynamic obstacles cutting across the path are either dealt with sequentially one after the other or coalesced into a composite obstacle depending on how these obstacles are spaced apart in the planning window. Figure 8 shows how two $s-t$ obstacles are combined into one single $s-t$ obstacle by obtaining the bounding box of C_{K,DO_1} and C_{K,DO_2}. Section 7.4 specifies a condition to determine if multiple obstacles should be merged depending on Δs and Δt. Either way, multiple dynamic obstacles are reduced to a single $s-t$ obstacle to be dealt within a planning window. This way, the pass and yield strategies developed so far still work the same way even with multiple obstacles in the planning window.

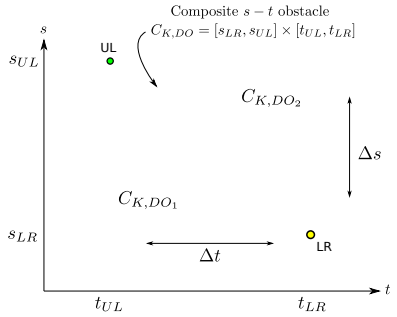


Figure 8: An example of how two $s-t$ obstacles C_{K,DO_1} and C_{K,DO_2} are merged into one single $s-t$ obstacle $C_{K,DO}$

Before any informed decision can be made regarding the strategy to choose, perception uncertainty associated with the dynamic obstacle state needs to be incorporated. In order to incorporate dynamic obstacle state (position and velocity) uncertainty and assess the risk associated with each strategy, the uncertainty model of a $x-y-t$ obstacle has to be utilized. Multiple realizations of the dynamic obstacle state is sampled from the Gaussian distribution yielding samples of position and velocity. In this work, a sample size n_s of 50 is used. For each sample $p_{do,n}$ and $v_{do,n}$, a corresponding $s-t$ obstacle $C_{K,DO,n}$ can be computed. A few samples of $s-t$ obstacles corresponding to a dynamic obstacle is illustrated in Figure 9. In order to circumnavigate the obstacle, there are two strategies that can be employed - *passing* and *yielding*. A passing strategy involves accelerating and passing in front of the obstacle, which requires targeting and reaching the upper left (UL) vertex from the initial state (s_0, t_0, v_0). Similarly, a yielding strategy involves decelerating and letting the obstacle pass, requiring targeting and reaching the lower right (LR) vertex from the initial state. For a passing strategy, the risk of collision can be estimated as the ratio of the number UL vertices that are unreachable to the total number of UL vertices. Similarly, for a yielding strategy, the risk of collision can be estimated as the ratio of the number of LR vertices that are unreachable to the total number of LR vertices. For the passing strategy shown in Figure 10, the collision risk is $1/6$ since there is only one UL vertex that is situated in the infeasible region. In other words,

$$R(p) = \begin{cases} 1, & \text{if point } p \text{ is not reachable} \\ 0, & \text{if point } p \text{ is reachable} \end{cases} \tag{2}$$

$$p_{coll,pass} = \frac{1}{n_s} \sum_{n=1}^{n_s} R(UL(C_{K_m,DO,n})) \tag{3}$$

$$p_{coll,yield} = \frac{1}{n_s} \sum_{n=1}^{n_s} R(LR(C_{K_m,DO,n})) \tag{4}$$

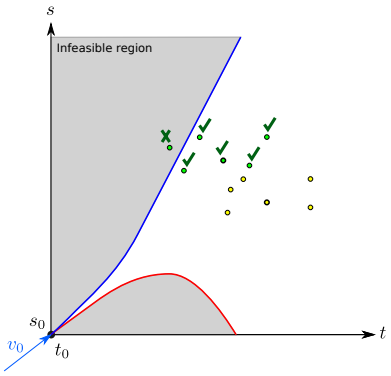


Figure 9: Qualitative example of risk assessment for a passing strategy where 6 samples of a st obstacles on a particular path are used. One of the samples has the upper left (UL) vertex in the infeasible region. Thus, the collision risk is 1/6 for the passing strategy

For the no change strategy, the ratio of $s-t$ obstacles instances intersecting the reference speed profile to the total number of $s-t$ obstacle instances is taken as the collision risk. This way perception uncertainty and satisfaction of dynamics constraints are taken care of. Note that the notion of collision employed here not only involves a physical collision but also counts inability to satisfy dynamics constraints as a collision.

In addition to collision risk, each strategy has an associated reference trajectory deviation cost. Section 6.6 describes the cost function used to evaluate a strategy. Once collision risk and trajectory deviation cost are computed for each strategy, they are assembled into a 6-tuple containing collision risk p_{coll} and reference deviation cost c for each of the three possible strategies that can be attempted over the reference path. An example of such a tuple is shown in the first row of Table 2.

Table 2: An example of the cost matrix for 4 path choices

Path choice	Strategy choice					
	Pass		Yield		No change	
	p_{coll}	c_{pass}	p_{coll}	c_{yield}	p_{coll}	$c_{no\,change}$
K_r	0.5	37.2	0.0	43.3	0.9	0
K_1	0.1	45.2	0.0	108.6	0.1	40
K_2	0.0	77.2	0.0	103.3	0.0	50
K_3	0.2	63.2	0.1	93.3	0.0	60

6.5 Generating Dynamically Feasible Plans

Line 14: For each path in A, each strategy is evaluated. And, a cost matrix is populated to form the entire table shown in Table 2.

Line 18: A combination of path option and strategy pair is termed as a *symbolic* plan. For example, $(K_3, YIELD)$ is a symbolic plan. The risk assessment method implicitly handles the dynamics constraints. Hence, picking symbolic plans with zero risk satisfies dynamics constraints. In some scenarios, there might be multiple symbolic plans that are risk free. In these cases, the expected cost is used to favor one symbolic plan over the other. Conversely, in some scenarios, there might be no symbolic plan that is risk free. In such cases, the symbolic plan exhibiting the least risk is chosen without regard for costs. An exception is raised and fed to the higher-level layer to trigger a replan.

Note that the symbolic plan does not specify any specific speed or specific coordinates on the planning window that has to be targeted and reached. Rather, it only specifies an abstract class of trajectories over the planning window. A value binding operation that converts the symbolic plan to a *numerical* plan occurs only when the actual obstacle is observed at a closer range during execution. Being at closer range, more accurate $s-t$ obstacles are observed and this accuracy is exploited to minimize reference trajectory violation while retaining safety. This value binding operation is needed because the perception system may be running at a higher frequency and thus, provide fast updates to dynamic obstacle measurements. During the value binding operation, new samples of $s-t$ obstacles are used to construct a worst case $s-t$ obstacle by obtaining a bounding box around the samples (see blue rectangle in Figure 11). If the strategy decision was to *pass*, the planner defaults to following the reference speed profile (i.e. no change strategy) till a critical time after which value binding operation is triggered. This critical time depends on the acceleration curve as shown in Figure 11. For the pass strategy, the value binding operation uses the UL vertex of the worst case $s-t$ obstacle and computes the command velocity along the chosen path as $v_{command} = s_{UL}/(t_{UL} - t_{start})$. Similarly, if the strategy decision was to *yield*, the planner defaults to following the reference speed profile till a critical distance after which value binding operation is triggered. This critical distance depends on the stopping distance associated with the deceleration curve. For the yield strategy, the value binding operation uses the LR vertex of the worst case $s-t$ obstacle and computes the command velocity along the chosen path as $v_{command} = s_{LR}/(t_{LR} - t_{start})$. The rationale behind such triggering is to incur as little reference trajectory deviation as possible and wait till it is absolutely necessary to act to avoid collision. Value binding triggered this way minimizes reference speed deviation. After the *pass* or *yield* maneuver, the planner resumes following the reference speed profile.

6.6 Trajectory deviation cost function

A distance based metric is used to evaluate symbolic plans consisting of a path choice and a strategy choice. The trajectory deviation cost consists of path deviation penalty c_{path} incurred while using alternative paths and speed deviation penalty incurred when passing and yielding strategies are used . The cost of pass, yield and no change strategies on a

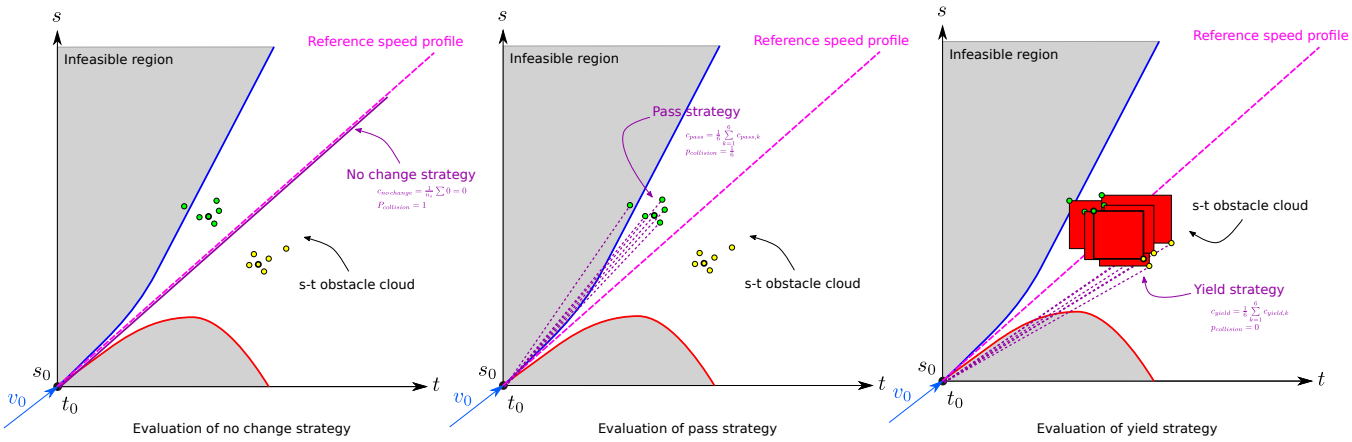


Figure 10: Evaluation of strategies

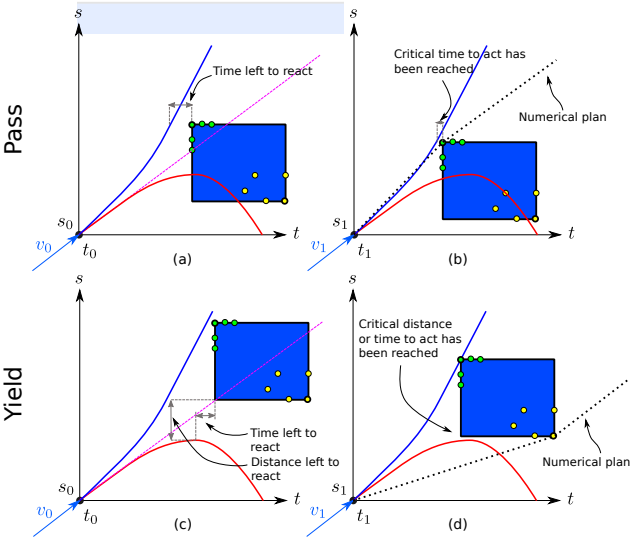


Figure 11: Triggering of value binding operation depends acceleration/deceleration curve. (a) and (c) illustrate the scenario where pass and yield strategies have been decided but reference speed profile is followed. (b) and (d) illustrate the scenario a little later at t_1 and just after value binding has occurred. The computed numerical plan is shown as a dotted black line. The blue rectangle shows the worst-case $s-t$ obstacle corresponding to the red $s-t$ obstacle samples used in value binding process

certain path are computed as follows:

$$c_{path} = \begin{cases} l_{K_m}, & \text{if path is } K_m \\ 0, & \text{if path is } K_r \end{cases} \tag{5}$$

$$c_{pass} = \frac{s_{UL} + s_{LR}}{2} \frac{|v_p - v_r|}{v_r} + c_{path} \tag{6}$$

$$c_{yield} = \frac{s_{UL} + s_{LR}}{2} \frac{|v_y - v_r|}{v_r} + c_{path} \tag{7}$$

$$c_{no\,change} = 0 + c_{path} \tag{8}$$

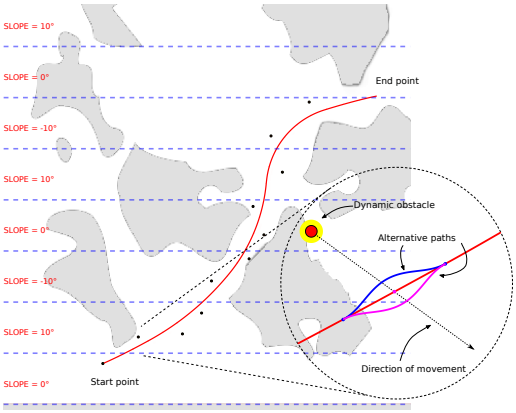


Figure 12: Top view of the scenario used for generating the results. Dynamic obstacles are poised to cut across the reference path from either the left or the right of the reference path. Inset: a magnified view of the dynamic obstacle (red with yellow buffer region), reference path (red) and alternative paths (blue, pink).

The cost function captures the distance over which speed violation happens and also weights it by the degree of speed deviation from the reference speed v_r (Figure 10). Note that the cost computed by this cost function corresponds to only one sample of the $s-t$ obstacle. Over n_s samples selected from a $s-t$ obstacle distribution, an expected cost can be computed for each of the strategies as follows:

$$E[c_{strategy}] = \frac{1}{n_s} \sum_{k=1}^{n_s} c_{strategy,k} \tag{9}$$

7 Results and Discussion

The following performance metrics are used to evaluate the executed trajectory:

- Number of collisions
- Execution time: the time taken in moving from start point to end point

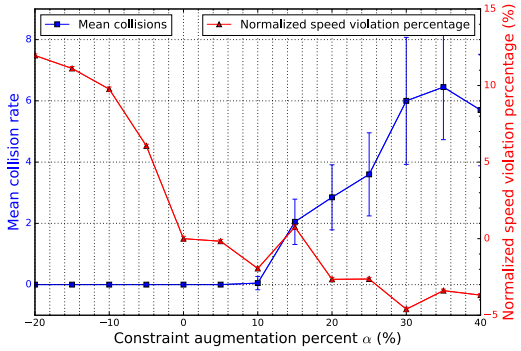


Figure 13: Relationship between collision rates and speed violation as the constraint augmentation factor is varied.

- Speed violation ratio: the ratio of execution time spent in violation of the reference speed profile to the overall execution time
- Path deviation distance: the distance over which reference path was not followed.

The TGDC was implemented on the scenario shown in Figure 12. The scenario features terrain strips with varying slopes. A point mass dynamic model of a vehicle was used. The capability of the vehicle was artificially capped at the limits allowed by the terrain. For example, if the terrain only allowed a certain maximum speed $2.0\,m\,s^{-1}$, any further throttle will have no effect after the vehicle reached $2.0\,m\,s^{-1}$. The vehicle was placed at the start point and commanded to reach the end point. Then, a kinematically feasible trajectory was supplied to the TGDC. Dynamic obstacles were positioned along the reference path at various points and with their trajectories crossing the path at various angles.

Each dynamic obstacle was assigned a trigger region, as the vehicle entered the trigger region, the dynamic obstacle was set in motion and assigned a specific speed. This speed was such that collision would occur if the vehicle continued traveling at the reference speed. Such triggering emulates the appearance of dynamic obstacles within the vehicle's sensing radius. Once triggered, a dynamic obstacle's state information is available to TGDC. Dynamic obstacle speed was also randomly varied around the nominal obstacle speed.

7.1 Impact of Inexact Dynamics Constraints on Performance

The TGDC's sensitivity to inexact dynamics constraints was measured by deliberately supplying a range of dynamics constraints around the correct dynamics constraints. Each dynamics constraint parameter in Table 1 was inflated by constraint augmentation factor δ to yield new parameters that were either more conservative or aggressive than the correct values. Figure 13 shows the collision rate and speed violation as the dynamics constraints are altered. Speed violation ratio has been normalized with respect to that of the exact dynamics constraints ($\delta = 0$). Figure 13 shows that conservative ($\delta < 0$) dynamics constraints results in zero collision at the cost of increased speed violations. On the other hand, aggressive ($\delta > 0$) dynamics constraints result in lower speed violations at the cost of a higher collision rate.

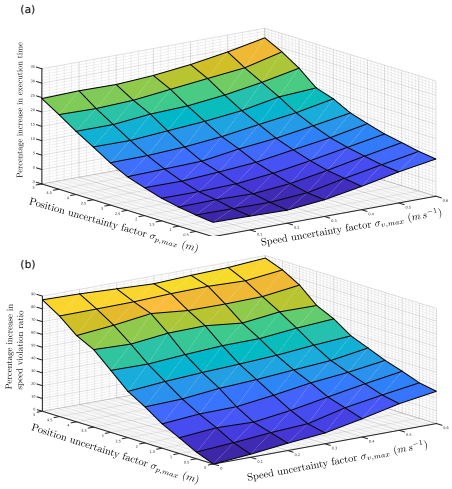


Figure 14: (a) Percent increase in execution time and (b) percent increase in speed violation ratio as uncertainty parameters $\sigma_{p,max}$ and $\sigma_{v,max}$ are varied.

7.2 Impact of Perception Uncertainty on Performance

To observe the effect of perception uncertainty on TGDC performance, the dynamic obstacle noise model parameters (position uncertainty parameter, $\sigma_{p,max}$ and speed uncertainty parameter $\sigma_{v,max}$) were varied, while the TGDC was supplied conservative dynamics constraints ($\delta = -20\%$). These noise model parameters control the interval width of the uniform distribution used in the obstacle position and velocity measurement. Note that $\sigma_{v,max}$ scales with the velocity of the dynamic obstacle. For each combination of noise parameters, 50 independent experiments were performed. The noise saturation distance threshold d_t (see Section 5) was set to 10 meters.

The normalized mean execution time and normalized speed violation ratio are shown in Figure 14. These quantities were normalized with respect to the zero noise case. No collisions were observed in these experiments. However, as noise increased, the execution time and speed violation ratio increased by up to 31.7% and 88.1% respectively. This result shows that increased dynamic obstacle state measurement noise does not compromise the safety of our method.

7.3 Need for both speed-regulation and path-variation

Relying on speed-regulation alone (with no alternative paths) to avoid dynamic obstacles results in longer execution time and a larger reference speed deviation. This is undesirable when dynamic obstacles stop along the reference path, which causes the vehicle to stop on the reference path indefinitely. Similarly, avoiding dynamic obstacles

Table 3: Comparison of speed-regulation, path-variation against the hybrid approach

(a) Pre-critical value binding (dynamic obstacles detected on time)

Normalized metric	Collision avoidance approach		
	Path-variation only	Speed-regulation only	Hybrid
Collision rate	0	0	0
Execution time	0.95	1.05	1.0
Path deviation	1.89	-	1.0
Speed violation ratio	-	1.3	1.0

(b) Post-critical value binding (dynamic obstacles detected too late)

Normalized metric	Collision avoidance approach		
	Path-variation only	Speed-regulation only	Hybrid
Collision rate	6.34	2.65	1.0
Execution time	0.77	0.96	1.0
Path deviation	0.38	-	1.0
Speed violation ratio	-	1.2	1.0

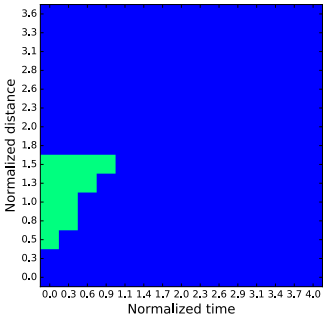


Figure 15: When handling two obstacles sequentially, collisions (green) happen when $s-t$ obstacles are spaced less than 1.5 stopping distances and 1 stopping time

by solely relying on moving on alternative paths (without speed-regulation) results in a larger reference path deviation. This is especially undesirable when no alternative paths exist (e.g. in a narrow passage). Table 3 compares the execution time, path deviation and speed violation ratio of the path-variation and speed-regulation approaches against the hybrid approach. No collisions were observed when dynamic obstacles were sensed sufficiently early (Table 3). These results show that the hybrid approach used in TGDC, combining speed-regulation and path-variation incurs a lower path-deviation and speed violation ratio at the cost of a marginal increase in execution time.

The sudden appearance of dynamic obstacles impacts collision avoidance. The value of the hybrid approach is more apparent when dynamic obstacles appear suddenly and hence, are sensed late (after the critical point). In such a situation, speed-regulation alone may not avoid collisions due to the UGV's inability to quickly decelerate or accelerate. Similarly, swerving to the extreme left or right without modifying speed, may cause vehicle to overturn. In contrast, the hybrid approach allows for these actions whilst decelerating. Allowing such actions results in lower collision rate (Table 3).

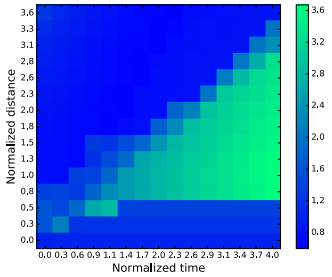


Figure 16: When handling two obstacles as a single composite obstacle, speed violation ratio increases when $s-t$ obstacles are spaced apart further than 0.5 stopping distances

7.4 Sequential Processing of Dynamic Obstacles

The spacing between multiple $s-t$ obstacles in the planning window affects TGDC performance. If the $s-t$ obstacles are well spaced, it may be better for the TGDC to consider each $s-t$ obstacle sequentially. Otherwise, it is better to coalesce them into a single composite $s-t$ obstacle as shown in Figure 8. To quantitatively study the effect of spacing between two $s-t$ obstacles, a grid of $s-t$ points is used (see Figure 16). Each point in the grid corresponds to the centroid-to-centroid vector $(\Delta t, \Delta s)$ between two $s-t$ obstacles (e.g. $(0,0)$ implies both obstacles are coincident, see Figure 8). The s and t axes of Figure 15 are normalized by vehicle stopping distance (corresponding to maximum speed) and vehicle stopping time, respectively. Figure 15 shows collisions when $s-t$ obstacles separated by 1.5 stopping distances (Δs) and 1 stopping time (Δt). This arises due to sequential processing of obstacles, where overtaking one obstacle put the vehicle on a collision course with the next. From the examination of Figure 15, $s-t$ obstacles with centroids less than one stopping time away from each other or less than approximately 2 stopping distances can be safely handled by creating a composite $s-t$ obstacle.

However, Figure 16 shows an increase in speed violation ratio up to a factor of approximately 3.7 when these composite $s-t$ obstacles are used. This trade-off must be taken into consideration while processing multiple dynamic obstacles within a single planning window.

8 Conclusions and Future Work

A method to generate collision risk-aware, dynamically feasible trajectories for a UGV operating over uneven terrain has been presented. This method has been shown to produce collision-free trajectories when the correct dynamics constraints are used even in the presence of sensor noise.

The overall trajectory generation architecture does not make assumptions regarding sensing modalities and vehicle models. Though the trajectory generation method is presented in the context of UGVs over uneven terrain, it is widely applicable to a variety of mobile robot platforms and with some modifications, even unmanned surface vehicles operating in dynamic sea states encountering dynamic obstacles.

For each terrain type, the dynamics constraints were as-

sumed to be known before hand. A natural extension would be to perform online learning of dynamics constraints, starting with conservative estimates and converging to the correct constraints. While effective, this approach has some limitations. Opportunities to explore multiple homotopy classes while navigating around $s - t$ obstacles are eliminated due to the way multiple obstacles are handled. Intuitively, this limitation is not particularly crippling since robots are typically agility-constrained, thus unable to weave through $s - t$ obstacles.

References

Bareiss, D., and van den Berg, J. 2015. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research* 34(12):1501–1514.

Blaich, M.; Weber, S.; Reuter, J.; and Hahn, A. 2015. Motion safety for vessels: An approach based on inevitable collision states. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1077–1082.

Borenstein, J., and Koren, Y. 1991. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on* 7(3):278–288.

Brock, O.; Trinkle, J.; and Ramos, F. 2009. *Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles*. MIT Press. 214–221.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *I. J. Robotic Res.* 17(7):760–772.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE* 4(1):23–33.

Fraichard, T., and Asama, H. 2003. Inevitable collision states. a step towards safer robots? In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, 388–393 vol.1.

Fraichard, T. 2007. A short paper about motion safety. In *Robotics and Automation, 2007 IEEE International Conference on*, 1140–1145.

Fulgenzi, C.; Spalanzani, A.; and Laugier, C. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *IEEE International Conference on Robotics and Automation, 2007*, 1610–1616. IEEE.

Howard, T. M., and Kelly, A. 2007. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research* 26(2):141–166.

Iagnemma, K.; Shimoda, S.; and Shiller, Z. 2008. Near-optimal navigation of high speed mobile robots on uneven terrain. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4098–4103. IEEE.

J. van den Berg, J. Snape, S. G. D. M. 2011. Reciprocal collision avoidance with acceleration-velocity obstacles. In *IEEE Int. Conf. on Robotics and Automation*.

LaValle, S. M. 2006. *Planning algorithms.* Cambridge, U.K.: Cambridge University Press. Available at http://planning.cs.uiuc.edu.

Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, 262–271.

Martinez-Gomez, L., and Fraichard, T. 2009. Collision avoidance in dynamic environments: An ics-based solution and its comparative evaluation. In *2009 IEEE International Conference on Robotics and Automation*, 100–105.

Minguez, J., and Montano, L. 2000. Nearness diagram navigation (nd): a new real time collision avoidance approach. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, 2094–2100 vol.3.

Shah, B., and Gupta, S. K. 2016. Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles. In *Internation Conference on Automated Planning and Scheduling (ICAPS' 16), London, UK, June 12 - 17, 2016*.

Shah, B. C.; Švec, P.; Bertaska, I. R.; Sinisterra, A. J.; Klinger, W.; von Ellenrieder, K.; Dhanak, M.; and Gupta, S. K. 2015. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots* 1–25.

Shiller, Z.; Large, F.; and Sekhavat, S. 2001. Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, 3716–3721 vol.4.

Shimoda, S.; Kuroda, Y.; and Iagnemma, K. 2005. Potential field navigation of high speed unmanned ground vehicles on uneven terrain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2828–2833.

Spenko, M.; Kuroda, Y.; Dubowsky, S.; and Iagnemma, K. 2006. Hazard avoidance for high-speed mobile robots in rough terrain. *Journal of Field Robotics* 23(5):311–331.

Werling, M.; Ziegler, J.; Kammel, S.; and Thrun, S. 2010. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, 987–993.

Wilkie, D.; van den Berg, J.; and Manocha, D. 2009. Generalized velocity obstacles. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 5573–5578.

Towards an Architecture for Discovering Domain Dynamics: Affordances, Causal Laws, and Executability Conditions

Mohan Sridharan and Ben Meadows
Department of Electrical and Computer Engineering
The University of Auckland, New Zealand

Abstract

Robots assisting humans in complex domains often have to reason with different descriptions of incomplete domain knowledge. It is difficult to equip such robots with comprehensive knowledge about the domain and axioms governing the domain dynamics. This paper presents a combined architecture that enables interactive and cumulative discovery of axioms governing action capabilities, and the preconditions and effects of actions in the domain. Specifically, Answer Set Prolog is used to represent the incomplete domain knowledge, and to reason with this knowledge for planning and diagnostics. Unexpected outcomes observed during plan execution trigger reinforcement learning to interactively discover specific instances of previously unknown axioms and to revise the existing axioms. Furthermore, a decision tree induction approach based on the relational domain representation constructs generic versions of the discovered axioms, which are then used for subsequent reasoning. The architecture's capabilities are illustrated and evaluated in a simulated domain of a robot moving objects to specific places or people in an indoor domain.

1 Introduction

Consider a robot[1] assisting humans by finding and moving desired objects to particular locations in an office building. In such complex, dynamic, and potentially resource-constrained environments, the robot will require a significant amount of domain knowledge, e.g., about domain objects, events and its own capabilities. At the same time, it will be challenging for humans to equip the robot with comprehensive domain knowledge, or to possess the time and expertise to interpret raw sensor input and provide detailed feedback. Domain knowledge may include commonsense knowledge, including default knowledge such as "books are usually in the library" that holds in all but a few exceptional circumstances, e.g., cookbooks are in the kitchen. The robot also extracts information from its sensors and actuators, which is typically associated with numerical representations, e.g., probabilistic representations of uncertainty such as "I am 90% sure the robotics book is on the table". In addition, a robot is typically equipped with axioms governing domain dynamics. Such domain axioms typically describe the preconditions and expected outcomes of actions that can be executed in the domain. The axioms also include knowledge of action capabilities, also known as *affordances*. With reference to an action, we define an affordance as a combination of attributes of object(s) and agent(s) involved in the action (Gibson 1986), e.g., the affordance of a person climbing a stair is described in terms of the stair's height with reference to the person's leg length (Warren 1984).

A fundamental challenge with these different types of knowledge possessed by a robot is that the knowledge is usually incomplete, and often needs to be revised over time. For instance, if the floor of a room has just been polished, a robot's movement in this room will produce unexpected outcomes in the absence of an accurate description of the robot's movement on such surfaces. To truly assist humans in complex domains, robots thus need the ability to augment and revise the different types of knowledge. Towards addressing this challenge, the architecture described in this paper seeks to enable interactive and cumulative discovery of domain axioms. Although our architecture can be used to discover both axioms governing actions performed by the robot and axioms governing exogenous actions, we focus on the former in this paper and assume that any knowledge of exogenous actions is limited to that encoded a priori—we leave the exploration of exogenous actions as a direction for further research. We discuss the following key characteristics of the architecture:

- For planning and diagnostics, an action language-based descriptions of transition diagrams of the domain are translated to an Answer Set Prolog (ASP) program for non-monotonic logical reasoning, and to a partially observable Markov decision process (POMDP) for probabilistic reasoning.
- Unexpected observations during plan execution are considered to indicate the existence of previously unknown knowledge about domain axioms. The discovery of these axioms and the corresponding action capabilities, is formulated as a Reinforcement Learning problem that is informed by ASP inference.
- Decision tree-based regression with the relational representation encoded in the ASP program, and a sampling-based approach, are used to identify candidate axioms,

[1] We use the terms "robot", "agent" and "learner" interchangeably, although an embodied agent is not essential for the learning task described in this paper.

and to generalize across these candidates. These generic axioms are included in the ASP program and used for subsequent reasoning.

Given the focus on the ability to discover axioms corresponding to different types of knowledge, we abstract away the uncertainty in perception and do not describe the probabilistic reasoning component of the architecture. We illustrate the architecture's non-monotonic logical reasoning and axiom discovery capabilities in a simulated domain that has a robot assisting humans by delivering desired objects to particular locations or people in an indoor domain.

The remainder of this paper is organized as follows. We first review related work in Section 2, and describe our architecture's components in Section 3. Experimental results are discussed in Section 4, followed by conclusions and a discussion of future work in Section 5.

2 Related Work

This section reviews some related work in logic programming, probabilistic reasoning, and relational learning, in the context of robotics.

Probabilistic algorithms are used widely for tasks such as reasoning and learning in robotics and AI, but these formulations, by themselves, make it difficult to reason with commonsense knowledge. In parallel, research in classical planning has provided many algorithms for representing and reasoning with commonsense knowledge. For instance, approaches based on first-order classical logic have been used for applications in robotics and AI, but they do not support desired capabilities such as non-monotonic logical reasoning and default reasoning. The logic programming community has developed many formalisms for non-monotonic logical reasoning, e.g., ASP is used by a growing international research community (Erdem and Patoglu 2012). These logical reasoning approaches, however, often require complete knowledge about the domain and the agents' capabilities. Also, these approaches, by themselves, do not support probabilistic reasoning, whereas quantitative reasoning about the uncertainty related to sensing and actuation on robots is often based on a probabilistic representation. Approaches have been developed to support both logical and probabilistic reasoning (Baral, Gelfond, and Rushton 2009; Milch et al. 2006; Richardson and Domingos 2006). The subset of these approaches based on first-order logic are often not expressive enough for certain types of knowledge, e.g., they model default knowledge and uncertainty by associating logic statements with numbers that may not be meaningful, whereas approaches based on logic programming do not support reasoning with large probabilistic components. For all these approaches, interactive discovery of knowledge continues to be an open problem.

Different approaches have been developed for representing and reasoning about action capabilities. Research in psychology indicates that humans can make accurate judgments about others' action capabilities using simple representations, without actually observing the subject perform the action(s) of interest (Ramenzoni et al. 2010). Many computational approaches have also been developed for representing and reasoning about affordances, often building on the knowledge representation and reasoning algorithms summarized above (Griffith et al. 2012; Sarathy and Scheutz 2016). Despite the existing research, open questions remain regarding the suitable definition and representation of affordances (Horton, Chakraborty, and Amant 2012).

In complex domains, agents often have to start with incomplete domain knowledge, and learn from repeated interactions with the environment. Different algorithms and architectures have been developed to support this capability. For instance, a first-order logic representation and the observed effects of actions have been used to learn causal laws (Shen and Simon 1989). This approach used axioms as working hypotheses to be revised through discriminant learning when predictions fail, but only the encoded preconditions and effects of actions could be monitored. Another approach incrementally refined operators encoded in first-order logic by making any unexpected observations the preconditions or effects of operators (Gil 1994). This work focused on augmenting existing knowledge and not on revising incorrect axioms, and did not allow for the same action to lead to different outcomes in different contexts. Furthermore, these (and other such) approaches do not support generalization of acquired knowledge as described in this paper, and also have the (known) limitations of approaches based on first-order logic.

Researchers have used inductive logic with ASP to monotonically learn causal rules (Otero 2003). A maximum satisfiability framework has also been used with plan traces for refining incomplete domain models (Zhuo, Nguyen, and Kambhampati 2013). Interactive learning has also been posed as a Reinforcement Learning (RL) problem with an underlying Markov decision process (MDP) formulation (Sutton and Barto 1998). Approaches for efficient RL include sample-based planning algorithms (Walsh, Goschin, and Littman 2010), and Relational Reinforcement Learning (RRL), which combines relational representations with regression for Q-function generalization (Tadepalli, Givan, and Driessens 2004). However, existing interactive relational learning algorithms focus on planning, only generalize over the states and actions corresponding to a given planning task (Driessens and Ramon 2003), or do not support the desired commonsense reasoning.

In this paper, we present an architecture that supports automatic, interactive discovery of previously unknown knowledge governing action capabilities and the preconditions and effects of actions. We build on and extend our architectures that (a) combined declarative programming and probabilistic graphical models for planning and diagnostics in robotics (Sridharan and Gelfond 2016; Sridharan et al. 2017); and (b) integrated declarative programming with relational reinforcement learning for interactive discovery of previously unknown axioms governing action execution (Sridharan and Meadows 2016) and the agent's action capabilities (Sridharan, Meadows, and Gomez 2017).

3 Architecture Description

Figure 1 shows a block diagram of the overall architecture. For any given goal, ASP-based non-monotonic log-

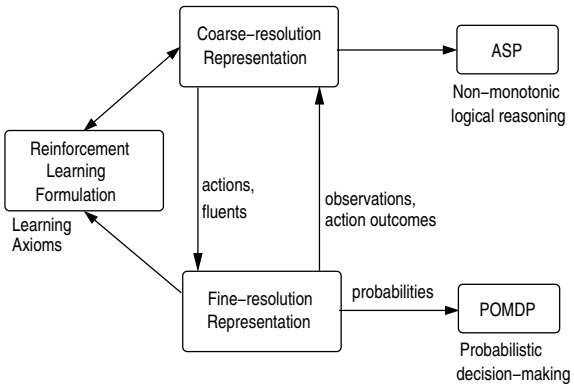


Figure 1: Architecture combines complementary strengths of declarative programming, probabilistic graphical models, and relational reinforcement learning.

ical reasoning with a coarse2-resolution domain description provides a sequence of abstract actions. Each abstract action is implemented as a sequence of concrete actions, using a POMDP to reason probabilistically with the relevant part of the corresponding fine-resolution system description. For complete details about representing and reasoning with tightly-coupled transition diagrams at these two resolutions, please see (Sridharan and Gelfond 2016; Sridharan et al. 2017). Here, we focus on the new component of the architecture for interactively discovering domain axioms. We thus abstract away the uncertainty in perception and do not discuss probabilistic planning. Instead, we describe ASP-based reasoning for planning and diagnostics, and relational reinforcement learning for axiom discovery. We illustrate these capabilities of our architecture in the context of the following domain.

Example 1. *[Robot Assistant (RA) Domain]*
Consider a robot that has to deliver objects to particular rooms or people. Attributes include:

- Different sorts (classes) such as *entity*, *person*, *robot*, *object*, *book*, *desk* etc.
- Static attributes such as a human's *role*, which can be $\{engineer, manager, sales\}$; the robot's *armtype*, which can be $\{electromagnetic, pneumatic\}$; an object's *surface*, which can be $\{hard, brittle\}$; and an object's *weight*, which can be $\{light, heavy\}$.
- Fluents such as location of humans and the robot, which can be one of *office*, *kitchen*, *library* and *workshop*; *status* of an *object*, which can be $\{damaged, intact\}$; whether an object is being held by the robot; and whether an *object* has been labeled.

As a partial illustration, consider a scenario with two rooms, three humans, one robot, three movable objects and five immovable objects—it has $6,946,816$ physical (object) configurations in a standard RL/MDP formulation and $55,296$ static attribute combinations that can be explored during the axiom discovery phase. In this domain, the robot may not know, for instance, that:

1. A brittle object is damaged when it is put down.
2. Delivering an unlabeled object to a sales person causes it to be labeled.
3. A damaged object cannot be delivered to a person, except to an engineer.
4. An object with a brittle surface cannot be labeled.
5. A heavy object cannot be picked up by a robot with an electromagnetic arm.
6. A damaged object cannot be labeled by a robot with a pneumatic arm.

These statements correspond to different types of knowledge encoded as causal laws, affordances etc, as described later. The objective is to construct and include suitable axioms in the ASP program.

3.1 Knowledge Representation

In our architecture, the transition diagrams of any given domain are described using an *action language* AL_d (Gelfond and Inclezan 2013). Action languages are formal models of parts of natural language used for describing transition diagrams. Action language AL_d has a sorted signature containing three *sorts*, namely *statics*, *fluents* and *actions*. Statics are domain properties whose truth values cannot be changed by actions, whereas fluents are domain properties whose truth values can be changed by actions. Fluents are of two types, *basic* and *defined*. Basic fluents obey the laws of inertia and are changed directly by actions. Defined fluents, on the other hand, do not obey the laws of inertia and may not be changed directly by actions—they are changed based on other fluents. Actions are defined as a set of elementary operations. A domain property p or its negation $\neg p$ is a *literal*. In AL_d, three types of statements are supported:

a **causes** l_b **if** $p_0, \ldots, p_m$ (Causal law)
l **if** $p_0, \ldots, p_m$ (State constraint)
impossible $a_0, \ldots, a_k$ **if** $p_0, \ldots, p_m$ (Executability condition)

where a is an action, l is a literal, l_b is a basic literal, and $p_0, \ldots, p_m$ are domain literals.

Domain description The representation of any domain is given by the system description $\mathscr{D}$, a collection of statement of AL_d, and history $\mathscr{H}$. The system description $\mathscr{D}$ has a sorted signature Σ and axioms that describe the transition diagram τ. The basic sorts in the signature Σ for the RA domain in Example 1 include *place*, *robot*, *entity*, *person*, *object*, *role*, *armtype*, *weight*, *surface*, *cup*, *book* etc. Sorts that are subsorts of other sorts, e.g., *cup* and *book* are subsorts of *object*, and *person* and *robot* are subsorts of *entity*, are arranged hierarchically. Furthermore, the signature includes specific instances of sorts, e.g., we have robot rob_1, places $\{office, workshop, kitchen, library\}$, and roles (of people) $\{engineer, programmer, manager\}$.

Domain attributes and actions are described in terms of the sorts of their arguments. The Σ of the RA domain in-

cludes fluents such as:

$$loc(entity, place)$$
$$obj\_status(object, status)$$
$$in\_hand(entity, object)$$

statics such as:

$$obj\_surface(object, surface)$$
$$obj\_weight(object, weight)$$
$$person\_role(person, role)$$

and actions such as:

$$move(robot, place)$$
$$serve(robot, object, person)$$
$$affix\_label(robot, object)$$

The signature Σ also includes the sort *step* for temporal reasoning, and the relation $holds(fluent, step)$ to state that a particular fluent holds true at a particular timestep.

The axioms of the system description $\mathscr{D}$ include causal laws such as:

$$move(rob_1, Pl) \textbf{ causes } loc(rob_1, Pl)$$
$$serve(rob_1, O, P) \textbf{ causes } in\_hand(P, O)$$
$$pickup(rob_1, O) \textbf{ causes } in\_hand(rob_1, O)$$
$$affix\_label(rob_1, O) \textbf{ causes } has\_label(O)$$

where the second axiom implies that when the robot executes the *serve* action in the context of a specific object and person, the object is in the person's hand. Although we do not describe it here, it is also possible to encode non-deterministic causal laws (Sridharan et al. 2017).

Examples of state constraints of the RA domain include:

$$\neg loc(O, L_2) \textbf{ if } loc(O, L_1),\ L_1 \neq L_2$$
$$\neg in\_hand(E, O_2) \textbf{ if } in\_hand(E, O_1),\ O_1 \neq O_2$$
$$loc(O, L) \textbf{ if } loc(E, L),\ in\_hand(E, O)$$

where the second axiom implies that any entity (robot or person) can only hold one object at a time.

Examples of executability conditions of the RA domain include the following:

$$\textbf{impossible } move(rob_1, L) \textbf{ if } loc(rob_1, L)$$
$$\textbf{impossible } pickup(rob_1, O) \textbf{ if } loc(rob_1, L_1), loc(O, L_2), L_1 \neq L_2$$
$$\textbf{impossible } serve(rob_1, O_1, P) \textbf{ if } in\_hand(P, O_2), O_1 \neq O_2$$

where the first axiom implies that a robot cannot pick up an object unless the robot and the object are in the same location.

The recorded history $\mathscr{H}$ of a dynamic domain is usually a record of fluents observed to be true/false at a time step, i.e., $obs(fluent, boolean, step)$, and the occurrence of an action at a time step, i.e., $hpd(action, step)$. We expand this notion of history to construct a model that supports the representation of (prioritized) defaults describing the values of fluents in their initial states. For instance, we can encode a statement such as "books are usually in the library and if it not there, it is normally in the office" as follows:

$$\textbf{initial default } loc(X, library) \textbf{ if } textbook(X)$$
$$\textbf{initial default } loc(X, office) \textbf{ if } textbook(X), \neg loc(X, library)$$

We can also encode exceptions to this default statement such as "cookbooks are in the kitchen". Any inconsistencies introduced by observations or acquired (and encoded) knowledge, is addressed using consistency-restoring (CR) rules, as described later in this section.

Affordance representation Next, consider the representation of affordances, which can be of two types. Positive (i.e., enabling) affordances describe permissible uses of object(s) and agent(s) in actions, whereas negative (i.e., forbidding) affordances, also known as disaffordances, describe unsuitable combinations of object(s) and/or agent(s) in the context of specific actions. In this paper, we introduce the following generic definition of forbidding affordances:

$$aff\_forbids(ID, A) \textbf{ if } not\ fails(ID, A), forbidding\_aff(ID, A)$$
$$\neg occurs(A, I) \textbf{ if } aff\_forbids(ID, A)$$

where the "not" in the first statement represents default negation (on which we provide more details later). The second statement implies that action A cannot occur if it is not afforded, which depends on whether suitable conditions have been defined to arrive at this conclusion. Any action can have one or more such relations defined with unique *ID*s. For instance, if we know that a robot with an electromagnetic arm cannot pick up a heavy object, the following statements may be included in $\mathscr{D}$:

$$forbidding\_aff(id1, pickup(R, O))$$
$$fails(id1, pickup(R, O)) \textbf{ if } not\ obj\_weight(O, heavy)$$
$$fails(id1, pickup(R, O)) \textbf{ if } not\ arm\_type(R, electromagnetic)$$

where the *pickup* action is not afforded if the object is heavy and the robot's arm is electromagnetic.

The representation of knowledge in our architecture brings up some subtle issues. First, for any given action, the axioms for both the executability conditions and the forbidding affordances imply that, when the body of these axioms are true, the desired outcomes (i.e., effects) will not be achieved because not all of the action's preconditions are satisfied—the action should then not be included in a plan. However, there are key differences in the type of knowledge encoded by executability conditions and affordances, and how this knowledge is represented. Affordance relations, once discovered, either specify preconditions that when true will lead to the corresponding action not having the desired outcomes (negative affordance), or specify preconditions that when true will lead to the successful execution of an action that may not have been considered possible (so far). In other words, these relations remove or add elements from the set of actions available for consideration to

achieve any given goal. Also, for any particular action, each affordance is defined in terms of the attributes of the objects operated by an agent, or of an agent and an object involved in this action. An executability condition does not have to meet these requirements, e.g., when an executability condition is discovered and in use, the plans computed for any given goal are a subset of the plans obtained in the absence of this condition. Second, the representation of affordances as relations between domain properties and actions, similar to the representation of actions, is distributed, e.g., we can define multiple affordance relations for any action. The advantages of this representation, e.g., information reuse and ease of plan explanation, are discussed in Section 4.2.

ASP-based inference The domain representation is translated into program $\Pi(\mathscr{D},\mathscr{H})$ in CR-Prolog, a variant of ASP that allows us to represent and reason with defaults and their exceptions, and incorporates CR rules (Balduccini and Gelfond 2003). We will use the terms CR-Prolog and ASP interchangeably in this paper. ASP is based on stable model semantics and non-monotonic logics, and includes *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that states *a is believed to be false*, "*not a*" only implies that *a is not believed to be true*, and unlike "$p \vee \neg p$" in propositional logic, "*p or* $\neg p$" is not tautologous (Gelfond and Kahl 2014). ASP can represent recursive definitions, defaults, causal relations, and constructs that are difficult to express in classical logic formalisms. The program Π thus consists of causal laws of $\mathscr{D}$, inertia axioms, closed world assumption for defined fluents, reality checks, and observations, actions, and defaults, recorded in $\mathscr{H}$. Every default is turned into an ASP rule and a CR rule that allows the robot to assume that the default's conclusion is false, under exceptional circumstances, so as to restore program consistency under exceptional circumstances. For instance, Π could include prioritized defaults encoded as ASP rules:

$$\begin{aligned} holds(loc(B,library),0) \leftarrow\ & \#textbook(B), \\ & not\ \neg holds(loc(B,library),0) \\ holds(loc(B,office),0) \leftarrow\ & \#textbook(B), \\ & \neg holds(loc(B,library),0), \\ & not\ \neg holds(loc(B,office),0) \end{aligned}$$

and the CR rules:

$$\begin{aligned} -holds(loc(B,library),0) \stackrel{+}{\leftarrow}\ & \#textbook(B). \\ -holds(loc(B,office),0) \stackrel{+}{\leftarrow}\ & \#textbook(B), \\ & \neg holds(loc(B,library),0). \end{aligned}$$

where the first CR rule implies that under exceptional circumstances, to restore program consistency, the robot can assume that a textbook is not in the library in the initial state. For planning, Π also includes the definition of a goal, a constraint stating that the goal must be achieved, and a rule generating possible future actions of the robot. Furthermore, although we do not discuss it in this paper, Π includes relations and axioms for explaining observed outcomes and partial scene descriptions.

Algorithms for computing the entailment, and for planning and diagnostics, reduce these tasks to computing the *answer sets* of the program $\Pi(\mathscr{D},\mathscr{H})$. The ground literals in an answer set represent the beliefs of an agent associated with program Π. An ASP solver is used to compute the answer sets of any given program Π. We use SPARC, a language that expands CR-Prolog to provide explicit constructs for specifying objects, relations, and their sorts (Balai, Gelfond, and Zhang 2013).

In the absence of comprehensive domain knowledge, appropriate plans may not be found and the execution of plan steps may have unexpected outcomes. In the RA domain, a robot moving a brittle cup to the kitchen may not know that putting the cup down is going to damage it—the unexpected outcome will only be observed after the action is completed. In this paper, *we focus on discovering such axioms that encode knowledge corresponding to causal laws, executability conditions, and forbidding affordances*. Including these axioms in Π will improve the quality of plans computed for achieving any given goal.

3.2 Axiom Discovery

This section describes the steps of the axiom discovery process. We begin with the formulation of axiom discovery as a reinforcement learning (RL) problem, followed by the decision-tree regression approach to construct a relational representation of the experiences obtained during RL. We then describe the construction of candidate axioms from the tree, and the validation of the candidate axioms to produce generic axioms to be included in the CR-Prolog program.

RL and Tree Induction When executing an action produces an unexpected transition, i.e., it does not produce the expected observations and/or produces new unexpected observations, the state description described by the action's effects becomes the goal state in a relational reinforcement learning (RRL) problem. The objective of this formulation is to find state-action pairs that are likely to lead to analogous "error" states. First consider the standard RL formulation and the underlying Markov decision process (MDP) defined by the tuple $\langle S,A,T_f,R_f\rangle$:

- S is the set of states;
- A is the set of actions;
- $T_f : S \times A \times S' \rightarrow [0,1]$ is the state transition function;
- $R_f : S \times A \times S' \rightarrow \Re$ is the reward function.

In the RL formulation, functions T_f and R_f are unknown to the agent. Each element in S grounds the domain attributes, and whether the last action to be executed had the expected outcome(s). Such a formulation mimics the experiences that a robot acquires incrementally and interactively as it performs the assigned tasks, and provides a principled approach to assign credit to current or previous state-action combinations for the observed transition(s). Note that the definition of the reward functions used in this approach has to be different when discovering axioms corresponding to the different types of knowledge. For instance, high immediate reward is to be provided:

- When an action's expected effects are not observed, if the focus is on discovering executability conditions.
- When effects in addition to those expected for an action are observed, if the focus is on discovering causal laws.
- When the expected and unexpected effects of an action are observed, if the focus is on discovering affordances.

One key benefit of ASP-based reasoning is that we can define and automatically compute the states and actions relevant to a given (unexpected) transition, eliminating parts of the search space irrelevant to the discovery of the desired knowledge. This identification of the relevant search space is equivalent to constructing the system description $\mathscr{D}(T)$, which is the part of $\mathscr{D}$ relevant to the transition T of interest. To do so, we first define the object constants relevant to the unexplained transition—this is a revised version of the definition in (Sridharan and Gelfond 2016; Sridharan et al. 2017).

Definition 1. *[Relevant object constants]*
Let a_{tg} be the *target action* that when executed in state σ_1 did not result in the expected transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$. Let $relCon(T)$ be the set of object constants of signature Σ of $\mathscr{D}$ identified using the following rules:

1. Object constants from a_{tg} are in $relCon(T)$;
2. If $f(x_1,\ldots,x_n,y)$ is a literal formed of a domain property, and the literal belongs to σ_1 or σ_2, but not both, then $x_1,\ldots,x_n,y$ are in $relCon(T)$;
3. If body B of an executability condition of a_{tg} contains an occurrence of a term $f(x_1,\ldots,x_n,Y)$ whose domain is ground, and $f(x_1,\ldots,x_n,y) \in \sigma_1$, then $x_1,\ldots,x_n,y$ are in $relCon(T)$.

Constants from $relCon(T)$ are said to be *relevant* to transition T. For instance, if the target action in RA domain is $a_{tg} = serve(rob_1, cup_1, person_1)$, with $loc(rob_1, office)$, $loc(cup_1, office)$ and $loc(person_1, office)$ in state σ_1, the relevant object constants will include rob_1 of sort *robot*, cup_1 and $person_1$ of sort *thing*, and *office* of sort *place*.

Once we know the relevant object constants, we can define the relevant system description $\mathscr{D}(T)$ as follows.

Definition 2. *[Relevant system description]*
The system description $\mathscr{D}(T)$ relevant to the transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$ that resulted in the unexplained transition, is defined by the signature $\Sigma(T)$ and axioms. The signature $\Sigma(T)$ is constructed as follows:

1. Basic sorts of Σ that produce a non-empty intersection with $relCon(T)$ are in $\Sigma(T)$.
2. For basic sorts of $\Sigma(T)$ that correspond to the range of a static attribute, all domain constants are in $\Sigma(T)$.
3. For basic sorts of $\Sigma(T)$ that correspond to the range of a fluent, or domain of a fluent or a static, domain constants that are in $relCon(T)$ are in $\Sigma(T)$.
4. Domain properties restricted to the basic sorts of $\Sigma(T)$ are also in $\Sigma(T)$.

The axioms of $\mathscr{D}(T)$ consist of those of $\mathscr{D}$ restricted to the signature $\Sigma(T)$. Building on our example of a state transition with $a_{tg} = serve(rob_1, cup_1, person_1)$, $\mathscr{D}(T)$ would not include other robots, cups or people that may exist in the domain. It can be shown that each transition in the original system description $\mathscr{D}$ maps to a transition in the system description $\mathscr{D}(T)$ relevant to the unexpected transition of interest—see (Sridharan et al. 2017) for complete details about establishing this relationship between transition diagrams. States of $\mathscr{D}(T)$, i.e., literals formed of fluents and statics in the answer sets of the corresponding ASP program, are states in the RL formulation for axiom discovery. Actions included in the RL formulation are (in a similar manner) the ground actions of $\mathscr{D}(T)$. Furthermore, it is possible to pre-compute or reuse some of the information used to construct the system description relevant to any given transition.

Coming back to our example of the target action $a_{tg} = serve(rob_1, cup_1, person_1)$, the relevant system description $\mathscr{D}(T)$ (with a small set of domain objects) includes thirteen atoms formed of relevant fluents:

$in\_hand(rob_1, cup_1)$, $item\_status(cup_1, damaged)$,
$item\_status(cup_1, intact)$, $labelled(cup_1, false)$,
$labelled(cup_1, true)$, $loc(person_1, kitchen)$,
$loc(person_1, library)$, $loc(person_1, office)$,
$loc(person_1, workshop)$, $loc(rob_1, kitchen)$,
$loc(rob_1, library)$, $loc(rob_1, office)$,
$loc(rob_1, workshop)$

eight possible (relevant) ground actions:

$affix\_label(rob_1, cup_1)$, $move(rob_1, kitchen)$,
$move(rob_1, library)$, $move(rob_1, office)$,
$move(rob_1, workshop)$, $pickup(rob_1, cup_1)$,
$putdown(rob_1, cup_1)$, $serve(rob_1, cup_1, person_1)$

and nine atoms formed of relevant static attributes:

$arm\_type(rob_1, electromagnetic)$, $surface(cup_1, hard)$,
$obj\_weight(cup_1, heavy)$, $obj\_weight(cup_1, light)$,
$role(person_1, engineer)$, $role(person_1, manager)$,
$role(person_1, sales)$, $surface(cup_1, brittle)$,
$arm\_type(rob_1, pneumatic)$

The system can then restrict its discovery process to focus on the possible combinations of relevant domain attributes.

Restricting exploration to just the relevant system description still leaves some open problems. For instance, Definitions 1 and 2 may not capture deeper relationships in the construction of the axioms. Also, in domains with complex relationships between objects, the space of relevant states and actions may still be so large that exploration may need to be limited to a fraction of this space during the RL trials. Furthermore, Q-learning (by itself) does not generalize to relationally equivalent states, making it computationally expensive to conduct RL trials in complex domains.

We exploit the relational representation encoded in the ASP program to address some of these problems. Specifically, we use a relational representation to generalize to relationally equivalent states. After one or more episodes of

Q-learning, all visited state-action pairs and their estimated Q-values are used to incrementally update a binary decision tree (BDT)—the motivation for constructing this tree is to relationally represent the robot's experiences. The path from the root to a leaf node corresponds to a partial description of a state-action pair (s,a). Internal nodes correspond to boolean *tests* of specific domain attributes or actions, and determine the node's descendants. The remainder of the state description is stored at the leaf—some of this may be transferred to a new node (for variance reduction) when the BDT is updated. The revised tree is used to compute a new policy, eliminating the need to completely rebuild the tree after each episode. The incremental inductive learning of the BDT draws on the algorithm by Driessens and Ramon (2003). In each Q-learning episode, the system stochastically decides to attempt either a random action or the one preferred by the current policy, ignoring actions currently invalidated by known axioms. Each action application also updates the information stored at a relevant leaf. Over time, the system assigns a higher value to outcomes perceived to be similar to the originally encountered unexpected transition. Since these errors may appear in the context of different combinations of domain attributes, these combinations are varied during RL trials, and the BDT reflects the exploration of different, but similar, MDPs. Q-learning episodes terminate when the Q-values stored in the BDT converge. For large, complex domains, we assume that when the number of explored attribute-value combinations reaches a fraction of the total number of possible combinations, the RL trials will be halted.

Constructing Candidate Axioms The next step constructs candidate axioms for causal laws, executability conditions and forbidding affordances, each of which has a known structure, as described in Section 3.1. For instance, any executability condition has the non-occurrence of an action in the head, with a body of (pre)condition(s) that prevent this action from being included in a plan. In a similar manner, causal laws have the occurrence of an action in the body and specific effects of executing the action in the head.

To construct these axioms, the system examines each leaf from the induced BDT, extracts a partial state-action description using its path to the root, and aggregates the stored information about domain attributes from this description. Branches with low Q-values or corresponding to an action that did not result in the observed transition are eliminated. The resulting structures include information on the mean and variance of the stored Q-value, based on the different samples clustered under each leaf. Each structure's statements about specific attributes holding or not holding are partitioned into two subsets, and all possible pairwise combinations of those subsets are elicited, producing unique tuples, each of which is the basis of a candidate. These store (a) the amassed Q-value; (b) the total variance; and (c) the number of training samples that influenced the candidate.

The system estimates the quality of the candidate from the Q-values of relevant samples it has experienced. It makes a number of random sample draws from the BDT, proportional to the size of the tree and the number of attributes not used as tests, without replacement. Each sample is a full state description of information at the leaf and along the path to the root. Each such state description that matches a candidate axiom adds to its Q-value, variance and count. Consider the axiom corresponding to a disaffordance relation in the RA domain, which prevents a robot with an electromagnetic arm from trying to pick up a heavy object. Assume that a candidate axiom has been constructed from an example leaf whose path to the root represents a partial state description that includes:

$loc(book_1, workshop)$, $loc(rob_1, workshop)$,
$obj\_weight(book_1, heavy)$, $arm\_type(rob_1, electromagnetic)$

and that a Q-value of 9.5 is associated with this example. One resulting candidate can be written as:

---

positive: [$obj\_weight(book_1, heavy)$,
$arm\_type(rob_1, electromagnetic)$]
negative: []
Q-sum: 9.5, Count: 1, Mean: 9.5

Of the random samples drawn during candidate quality estimation, only some will match the candidate's partial state description, e.g.:

---

positive: [$loc(book_1, workshop)$,
$obj\_weight(book_1, heavy)$, $obj\_status(book_1, intact)$]
negative: []
Q: 9.9

The system uses this sample to update the candidate:

---

positive: [$obj\_weight(book_1, heavy)$,
$arm\_type(rob_1, electromagnetic)$]
negative: []
Q-sum: 19.4, Count: 2, Mean: 9.7

When all the candidates have been found, the system can then choose the final set of axioms to be added to the system description, i.e., the CR-Prolog program.

Filtering Candidate Axioms The final step of the axiom discovery process is generalization, i.e., the identification of the most generic form of candidate axioms with a sufficiently high likelihood of representing correct knowledge about the domain. First, candidates not refined by additional training samples after their construction are removed. Then, the candidates are ranked by the number of samples used to adjust them, and any candidates that elaborate other, higher-ranked candidates are removed. For instance,

The remaining candidates undergo validation tests in simulation. For instance, a candidate executability condition, if true, should describe a case where an action will not provide the desired effects. If we can find a case that should imply a "failure" (i.e., unexpected transition) based on this axiom, but meets with "success" (i.e., expected transition) when the action is executed, the candidate axiom is incorrect. To this end, the simulation takes a random state where the target action is known to succeed, and makes minimal changes to the

domain attributes necessary to make the state match the partial state description of the candidate axiom. If executing the action only provides the expected outcomes in this adjusted state, the candidate axiom is discarded. Note that these validation tests are guaranteed *not* to retract any correct axioms, but are not guaranteed to retract all incorrect ones. The remaining candidate axioms, after suitably replacing the object constants with variables, are included in the CR-Prolog program that is used for reasoning in the subsequent steps.

4 Experimental Setup and Results

In this section, we first state the claims about our architecture's capabilities (Section 4.1). Next, we illustrate some of these capabilities using an execution trace, and discuss some key advantages of the representation of knowledge in our architecture (Section 4.2). We then summarize and discuss the results of experimental evaluation in a simulated domain (Section 4.3).

4.1 Claims

We posit and evaluate the following six central claims about our architecture's capabilities:

1. The distributed representation of affordances and other types of knowledge supports efficient inference, information consolidation and information reuse;
2. The architecture can learn symbolic knowledge structured as affordances, executability conditions, and causal laws;
3. During axiom discovery, automatically limiting search to the space relevant to any given unexpected transition improves computational efficiency;
4. Our approach to discovering different types of knowledge is robust to perceptual noise;
5. Introducing validation tests in the loop of learning, planning, and execution, significantly improves the accuracy of the discovered axioms; and
6. The discovered axioms help improve the quality of plans generated for any given goal.

We discuss the first claim qualitatively, and evaluate the other five claims quantitatively. We consider six target axioms—two each of affordances, executability conditions, and causal laws—including those discussed in Example 1. We conducted trials of 1000 repetitions apiece, providing them to the domain but removing them from the system's domain model. We examined the output axioms which the system discovered in each trial. Although each of these tests focused on a single target axiom, we have demonstrated the ability to discover axioms simultaneously elsewhere (Sridharan, Meadows, and Gomez 2017). We also conducted trials in which we explored different fractions of the search space, ignoring relevance, comparing 300 trials for each of these explorations (50 repetitions for each of the six target axioms). We used precision and recall as measures of accuracy. Furthermore, we allowed the system to test each discovered axiom in simulation, and report the resulting effects of this checking on initial accuracy.

4.2 Execution Trace and Discussion

As an illustration of the architecture's working, consider the robot in the RA domain that does not know that a brittle object will be damaged if it is put down. Suppose also that $obj\_surface(cup_1, brittle)$ and $obj\_status(cup_1, intact)$, and that the domain characteristics are otherwise as described earlier. Let the initial state therefore include the following literals:

$$in\_hand(cup_1, rob_1)$$
$$loc(cup_1, workshop)$$
$$obj\_surface(cup_1, brittle)$$
$$obj\_status(cup_1, intact)$$

and let the goal state description include:

$$loc(C, kitchen)$$

where C is a variable of sort *cup*, i.e., the objective is to deliver a cup to the kitchen. One possible plan to achieve this goal has two actions:

$$move(rob_1, kitchen)$$
$$putdown(rob_1, cup_1)$$

However, if this plan is executed, the second action results in an unexpected outcome, potentially triggering the discovery process. Over a period of learning, the robot is able to add the following generic axiom to its system description:

$$putdown(R, O) \textbf{ causes } obj\_status(O, damaged)$$
$$\textbf{if } obj\_surface(O, brittle)$$

preventing the robot from (in the future) including a *putdown* action in a plan involving a brittle object.

Let us now also consider the first claim about the benefits of the representation of knowledge in our architecture. Recall that action capabilities, and the preconditions and effects of actions, are encoded in a distributed manner using one or more axioms, which provides the following advantages:

- First, it will be possible to provide more meaningful explanations of plans and inferences. Recall that affordances are statements about an action with respect to the attributes of object(s) and/or agent(s) involved in the action. Each affordance also relates a partial state description to the actions that can (or cannot) be performed in states that build on this state description. When grounded, such a representation is close to language structures likely to be used for responses, for instance in diagnostics and plan explanation. It will thus be easy to translate and use this knowledge to make statements of the form "lifting this large cylinder with this small robot could work as long as the cylinder is not heavy", or "Affixing a label to the coffee cup will not work when the cup is known to be brittle and the label applicator is known to work on hard surfaces".
- Second, it will be possible to efficiently respond to queries that require consolidation of knowledge across attributes of robots or objects, by directing attention to the relevant

knowledge. For instance, assume that the domain knowledge includes affordance relations that describe the ability of individual robots, with different strength levels, to pick up (or not pick up) objects of different weights and surfaces. Questions of the form "what objects can weak robots pick up?", or "which robots can pick up cups?" can be answered quickly by expanding Definition 2 to automatically consider only the affordance relations and attributes relevant to such questions. Furthermore, it will also be possible (although we do not consider it here) to develop composite affordance relations, e.g., a hammer may afford an "affix objects" action in the context of a specific agent because the handle affords a pickup action and the hammer affords a swing action, by the agent.

- Third, the distributed representation will simplify inference and information reuse. For instance, if a hammer has a graspable handle, this relation also holds true of the parent object class and for all other objects with graspable handles. In a similar manner, a forbidding affordance (i.e., disaffordance) that prevents the *pickup* action when the type of the robot's arm does not match the weight of the object, can also be used to infer the robot's inability to perform similar actions such as opening a heavy door or closing a large window. This relates to research in psychology which indicates that humans can judge the intent and action capabilities of others without actually observing them perform the target actions (Ramenzoni et al. 2010).

Initial results of experimental evaluation do support the benefits listed above—future work will design and conduct extensive experimental trials to gather quantitative results in support of the first claim.

4.3 Experimental Results

Our prior work proposed different architectures for discovering forbidding affordances (Sridharan, Meadows, and Gomez 2017) and for executability conditions (Sridharan and Meadows 2016). Here, we evaluated whether a single architecture can discover these forms of knowledge as well as the knowledge of causal laws.

Discovering different types of knowledge: In our experiments, the average recall and precision over the entire set of target axioms, were 0.98 and 0.72 respectively. The system took a mean 5.95 time units to perform decision tree induction and a mean 0.35 time units to extract a set of axioms. We found that axioms with more clauses were more difficult to learn. Also, if axioms were structured to include specific exceptions, there were more logical over-specifications, e.g., an axiom stating that "a light, brittle object cannot be labeled" was discovered instead of "a brittle object cannot be labeled". We treat these over-specifications as false positives, and the (overall) worst case recall and precision were 0.91 and 0.64 respectively. This supports our second claim, that our architecture can discover knowledge corresponding to affordances, executability conditions, and causal laws.

Effect of directed exploration: Next, we conducted trials to examine the benefit of limiting exploration, for any given unexpected transition, to just the relevant portion of the search space. The mean time required to compute this relevant space was 533 time units, and the mean time to discover the axioms in this reduced search space was 6.3 time units, resulting in a mean total time of 539.3 units for discovering axioms in the search space relevant to any given transition. It is possible to precompute and cache the relevant space to be explored in response to any given unexpected transition in a given domain. Recall that the RA domain has $55,296$ static attribute combinations and $6,946,816$ physical (object) configurations. For our target axioms, this space can be reduced to $8-128$ static combinations (mean $= 54$) that are relevant to any given action.

Next, we conducted trials that explored only a fraction of the original space, ignoring relevance, to discover the target axioms. At 0.01% exploration of the original space, a mean total time of 608.4 units was required to discover the axioms; the corresponding precision and recall were 0.28 and 0.86 respectively. The time taken to explore 0.01% of the original search space is therefore similar to the length of time taken to explore just the space relevant to any given transition. However, exploring only the relevant search space provides significantly higher values of recall and precision. Next, at 0.1% exploration of the search space (again ignoring relevance), the discovery of axioms took a mean total time of 800.4 units; the corresponding precision and recall were 0.53 and 0.99 respectively. In this case, although the recall is similar to that obtained when only the relevant search space is explored, the precision is still significantly lower and the computation time requirement is significantly higher. Furthermore, our implementation of the construction of the relevant search space can also be made more efficient. These results thus support the third claim, that limiting exploration to the space relevant to any given unexpected transition improves the computational efficiency of axiom discovery.

Robustness to perceptual noise: Next, we evaluated the fourth claim, i.e., whether axiom discovery is robust to perceptual noise, which we interpreted as the noise having a negative but non-catastrophic impact on performance. We introduced noise in the form of a fixed chance for an action to have an unexpected outcome in the form of the removal or addition of a single literal formed of a random fluent of the desired resultant state. We performed 500 repetitions for each of the six target axioms in the RA domain, and repeated this for 10 different levels of noise between $0-20\%$. The corresponding recall and precision scores are summarized in Figures 2 and 3. We observe that the addition of noise affects both precision and recall, with a more significant effect on precision. However, note that errors were predominantly false positives corresponding to over-specifications of the target axioms, e.g., they included executability conditions that would correctly prevent an action from being considered during planning but were not in the most general form possible. In addition, these false positives were incrementally eliminated as the robot performed a series of validations tests on the discovered axioms, as described below.

Condition	Causal laws		Executability conditions		(Dis)Affordances	
	Axiom 1	Axiom 2	Axiom 3	Axiom 4	Axiom 5	Axiom 6
Without axioms	101.4	0	37.9	164.4	29.7	121.3
With axiom	110.7	11.2	24.8	75.7	23.0	86.5

Table 1: Number of plans found without and with each of the target axioms (see Example 1) under consideration. On average, discovering causal laws increases the number of feasible plans to achieve any given goal, whereas discovering executability conditions and disaffordances decreases the number of plans that can be used to achieve any given goal.

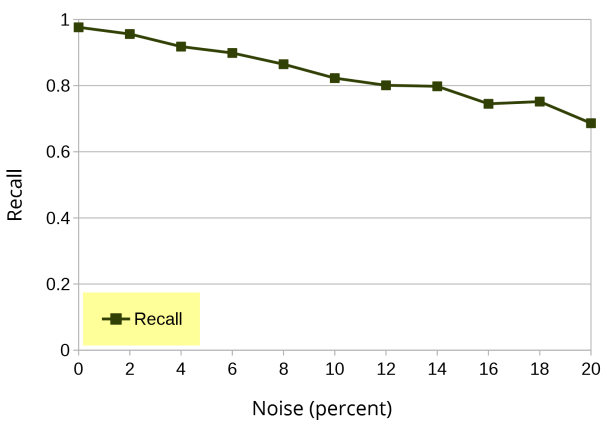


Figure 2: Recall scores as a function of added noise; although added noise reduces accurate recall of axioms, many errors correspond to over-specifications that are filtered by validation tests.

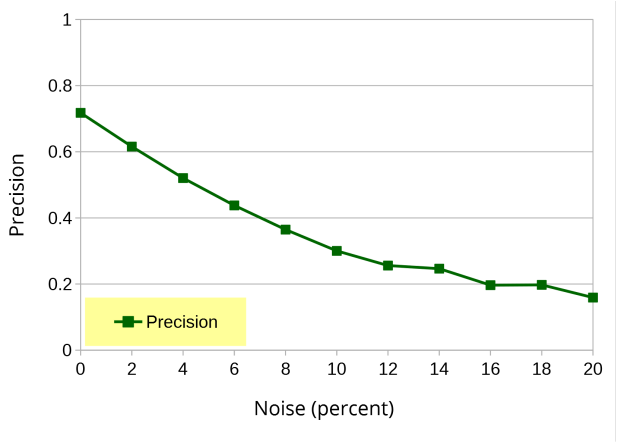


Figure 3: Precision scores as a function of added noise; although added noise affects precise discovery of axioms, many errors correspond to over-specifications that are filtered by validation tests.

Effect of validation tests: Our recent work showed that precision increased with the number of tests conducted to validate the discovered axioms (Sridharan, Meadows, and Gomez 2017). In our current work, we observed that there is a similar improvement in precision with our architecture that supports the discovery of axioms corresponding to different types of knowledge in a more complex domain. These validation tests improved precision by filtering incorrect axioms. For instance, when only the search space relevant to any unexpected transition is explored, precision improved (on average) from 0.72 to 0.91 after ten validation tests. Even for the 0.01% and the 0.1% exploration of the original space (i.e., ignoring relevance), ten validation tests improved precision from 0.28 to 0.90 and from 0.53 to 0.95 respectively. These results support our fifth claim, that including validation in the loop of planning, execution, and learning, improves the accuracy of the discovered knowledge.

Effect on plan quality: Finally, we explored the effect of the discovered axioms on the quality of plans generated. We conducted 1000 paired ASP-based planning trials for each axiom, and for all the axioms, with and without the corresponding target axiom(s) in the system description. Table 1 summarizes the results for each of the six axioms, and displays some interesting trends. For instance, the set of plans found after including axioms that represent knowledge corresponding to a causal law (e.g., axiom 1 or 2 in this study) was a *superset* of the plans found without including these axioms in the system description. In other words, discovering previously unknown knowledge corresponding to a causal law (on average) increases the number of possible plans that can be constructed to achieve an assigned goal. On the other hand, the set of plans found after including axioms that represent knowledge of an executability condition or a forbidding affordance (axioms $3-6$ in this study) was a *subset* of the plans found without including these axioms. In other words, discovering knowledge corresponding to a previously unknown executability condition or forbidding affordance (on average) reduces the number of plans that can be executed to achieve an assigned goal. However, when axioms corresponding to different types of knowledge are considered together, the set of plans found after including these axioms is no longer a subset or superset of the plans found without including the axioms. For instance, when all six target axioms are considered together, all we can say is that 29.2 is the average magnitude of the difference in the number of plans found with and without including these axioms. Furthermore, we verified that all the plans that were computed after including all the target axioms were correct.

5 Conclusions and Future Work

This paper described an architecture for interactive and cumulative discovery of axioms corresponding to causal laws, executability conditions and forbidding affordances. We used Answer Set Prolog to represent and reason with incomplete domain knowledge for planning and diagnostics, and used decision tree induction and relational reinforcement learning to identify specific candidate axioms and generalize across these specific instances. Experimental results (in the context of a robot assisting humans in an indoor domain by moving particular objects to particular places or people) indicate that our approach:

- Supports reliable and efficient reasoning and discovery of the axioms corresponding to different types of knowledge, especially when search is limited to the space relevant to the unexpected transition that triggered axiom discovery;
- Provides some robustness to perceptual noise—although noise degrades the accuracy of axiom discovery, including validation tests in the loop of planning, execution and learning helps recover from these errors; and
- Results in the discovery of axioms that when included in the system description, improves the quality of the plans found for any given goal.

The architecture opens up multiple directions for research that we seek to investigate in the future. More specifically:

- We will explore the problem of automatically determining when to discover different types of knowledge, and thoroughly investigate the benefits of the underlying distributed representation;
- We will investigate the discovery of affordance relations that enable the execution of specific actions by specific agents;
- We will explore active interactive discovery of axioms instead of limiting discovery to situations corresponding to unexpected state transitions during plan execution; and
- We will evaluate the architecture on physical robots, which will require the use of the component of the architecture that reasons about perceptual inputs probabilistically.

The long-term objective is to enable robots assisting humans to represent, reason with, and interactively revise different descriptions of incomplete domain knowledge.

Acknowledgements

The authors thank Pat Langley for discussions related to the representation of affordances described in this paper. This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-13-1-0766, Asian Office of Aerospace Research and Development award FA2386-16-1-4071, and US National Science Foundation grant CHS-1452460. All opinions and conclusions expressed in this paper are those of the authors.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*, 9–18.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Driessens, K., and Ramon, J. 2003. Relational Instance-Based Regression for Relational Reinforcement Learning. In *International Conference on Machine Learning*, 123–130. AAAI Press.

Erdem, E., and Patoglu, V. 2012. Applications of Action Languages to Cognitive Robotics. In *Correct Reasoning*. Springer-Verlag.

Gelfond, M., and Inclezan, D. 2013. Some Properties of System Descriptions of AL_d. *Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming* 23(1-2):105–120.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gibson, J. J. 1986. *The Ecological Approach to Visual Perception*. Psychology Press.

Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *International Conference on Machine Learning*, 87–95.

Griffith, S.; Sinapov, J.; Sukhoy, V.; and Stoytchev, A. 2012. A Behavior-Grounded Approach to Forming Object Categories: Separating Containers From Noncontainers. *IEEE Transactions on Autonomous Mental Development* 4:54–69.

Horton, T. E.; Chakraborty, A.; and Amant, R. S. 2012. Affordances for Robots: A Brief Survey. *Avant: Journal of Philosophical-Interdisciplinary Vanguard* III(2):70–84.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2006. BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Otero, R. P. 2003. Induction of the Effects of Actions by Monotonic Methods. In *International Conference on Inductive Logic Programming*, 299–310.

Ramenzoni, V. C.; Davis, T. J.; Riley, M. A.; and Shockley, K. 2010. Perceiving Action Boundaries: Learning Effects in Perceiving Maximum Jumping-Reach Affordances. *Attention, Perception and Psychophysics* 72(4):1110–1119.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine Learning* 62(1-2):107–136.

Sarathy, V., and Scheutz, M. 2016. A Logic-based Computational Framework for Inferring Cognitive Affordances. *IEEE Transactions on Cognitive and Developmental Systems* 8(3).

Shen, W.-M., and Simon, H. 1989. Rule Creation and Rule Learning through Environmental Exploration. In *International Joint Conference on Artificial Intelligent*, 675–680.

Sridharan, M., and Gelfond, M. 2016. Using Knowledge Representation and Reasoning Tools in the Design of Robots. In *IJCAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS)*.

Sridharan, M., and Meadows, B. 2016. Should I do that? Using Relational Reinforcement Learning and Declarative Programming to Discover Domain Axioms. In *International Conference on Developmental Learning and Epigenetic Robotics (ICDL-EpiRob)*.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2017. A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. Technical report, `http://arxiv.org/abs/1508.03891`.

Sridharan, M.; Meadows, B.; and Gomez, R. 2017. What can I not do? Towards an Architecture for Reasoning about and Learning Affordances. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Sutton, R. L., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational Reinforcement Learning: An Overview. In *Relational Reinforcement Learning Workshop at International Conference on Machine Learning*.

Walsh, T. J.; Goschin, S.; and Littman, M. L. 2010. Integrating Sample-Based Planning and Model-Based Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Warren, W. H. 1984. Perceiving Affordances: Visual Guidance of Stair Climbing. *Journal of Experimental Psychology: Human Perception and Performance* 10(5):683–703.

Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *International Joint Conference on Artificial Intelligence*, 2451–2457.

Approximating Reachable Belief Points in POMDPs with Applications to Robotic Navigation and Localization

Kyle Hollins Wray and **Shlomo Zilberstein**
College of Information and Computer Sciences
University of Massachusetts, Amherst, MA 01002
{wray,shlomo}@cs.umass.edu

Abstract

We propose an algorithm called σ-approximation that compresses the non-zero values of beliefs for partially observable Markov decision processes (POMDPs) in order to improve performance and reduce memory usage. Specifically, we approximate individual belief vectors with a fixed bound on the number of non-zero values they may contain. We prove the correctness and a strong error bound when the σ-approximation is used with the point-based value iteration (PBVI) family algorithms. An analysis compares the algorithm on six larger domains, varying the number of non-zero values for the σ-approximation. Results clearly demonstrate that when the algorithm used with PBVI (σ-PBVI), we can achieve over an order of magnitude improvement. We ground our claims with a full robotic implementation for simultaneous navigation and localization using POMDPs with σ-PBVI.

Introduction

Automated planning domains have been steadily growing in complexity, especially for partially observable Markov decision processes (POMDPs) (Kaelbling, Littman, and Cassandra 1998). They now encapsulate problems ranging from water reservoir control (Castelletti, Pianosi, and Soncini-Sessa 2008) to autonomous driving (Wray and Zilberstein 2015a; Wray, Pineda, and Zilberstein 2016). The growing number of possible states and observations in these problem domains requires POMDP solvers to handle a large space of agent's beliefs over domain states. The complexity of planning has inspired the development of numerous approximate planning algorithms.

One approximation method that proved particularly effective is point-based value iteration (PBVI) (Pineau, Gordon, and Thrun 2003), which restricts value function computations to a subset of the belief space, thereby accelerating *value iteration* techniques (Smith and Simmons 2004; Spaan and Vlassis 2005; Pineau, Gordon, and Thrun 2006; Shani, Brafman, and Shimony 2007; Poupart, Kim, and Kim 2011; Shani, Pineau, and Kaplow 2013). We propose an algorithm called σ-approximation that exploits a bounded quantity of zero-values over the set of beliefs to greatly improve belief operations in POMDP algorithms.

The σ-approximation method addresses an orthogonal issue from PBVI; both methods can, in fact, be used together or separately. PBVI concerns itself with the number of reachable beliefs and the selection of an approximate subset. Our algorithm focuses on the number of non-zero values *within* each belief point. Specifically, we construct a new set of beliefs to use for updates given a non-zero value constraint r_z (e.g., $r_z \approx \log n$, where n is the number of states). For each belief, we sort the belief values and select only the top r_z values, then normalize these values to create a new belief. These are then used in update equations, allowing for dot products with beliefs to be computed much faster based on this constraint r_z. We formally show that this simple routine is the optimal projection given the r_z constraint. Then, we prove a strong bound on the error for the σ-approximation used in point-based algorithms. Finally, we demonstrate its vast performance gains with low error in six larger domains.

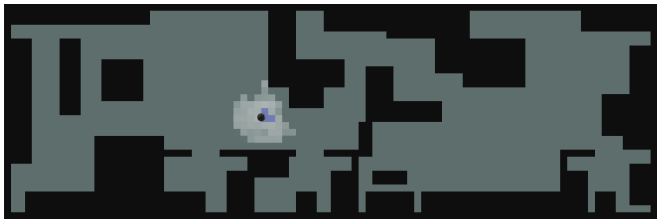


Figure 1: Example POMDP navigation in a real world laboratory map (2914 states; $\sim$28m-by-8.8m). The black circle is the robot. Gray and black cells are free space and obstacles, respectively. Blue and white cells visually depict a single belief point; their opacity is a log-probability of the robot's location. Blue highlights the top $k=3$ probability masses in the belief. For *planning*, the σ-approximation uses the fixed top k weights for each belief.

To our knowledge, this is a new form of *belief compression* for POMDPs, with theoretical guarantees in conjunction with PBVI. A similar method was briefly suggested for the separate Bayes-adaptive POMDP model (Ross, Chaib-draa, and Pineau 2008). They did not, however, provide any theoretical or empirical analysis, nor the general algorithm presented here. Value directed belief state compression (Poupart and Boutilier 2003) performs intelligent state space compression to only discard (mostly) irrelevant parts of the belief state, yielding the smallest invariant Krylov subspace. They use a distinct linear lossy compression method that approximates the original POMDP. Exponential

family principle components analysis (E-PCA) has also been used to compress beliefs into a low-dimensional belief space (Roy, Gordon, and Thrun 2005). They instead solve the compressed POMDP, then map the policy back to the original POMDP. This operates over all beliefs at once, whereas ours operates on individual beliefs. Both compression methods and their numerous variants differ markedly from our fixed non-zero values, sort-based algorithm.

The σ-approximation exploits *sparse beliefs*. While few algorithms leverage this fact, such as sparse stochastic finite state controllers (Hansen 2008), it has been suggested as a measure of POMDP complexity (Lee, Rong, and Hsu 2008). Other related work includes algebraic decision diagrams (ADDs) used to solve large factored POMDPs, and approximate belief points in the process (Shani et al. 2008), albeit in a very different manner from our approach.

Our paper begins with a review of the POMDP model (Section 2), followed by our σ-approximation algorithm (Section 3). Additionally, we present two main propositions (correctness and an error bound) as well as two supporting lemmas. Then, we present experiments on standard benchmark domains, and a full robot implementation for navigation and localization, that demonstrate our approximation vastly improves performance with minor error (Section 4). We conclude with a discussion of our approach and potential future work (Section 5).

Background

A partially observable Markov decision process (POMDP) is represented by a tuple $\langle S, A, \Omega, T, O, R \rangle$. S is a set of n states, A is a set of m actions, and Ω is a set of z observations. $T : S \times A \times S \to [0,1]$ is a state transition function mapping a state s and action a to a successor state s' with probability $T(s,a,s') \equiv Pr(s'|s,a)$. It is common in practice, however, to define T with a successor function that returns only the non-zero valued successor states and their probabilities. Let the maximum number of possible successor states be denoted as $n_s \leqslant n$. $O : A \times S \times \Omega \to [0,1]$ is an observation function that stochastically emits an observation ω given action a led to state s' with probability $O(a,s',\omega) \equiv Pr(\omega|a,s')$. $R : S \times A \to \mathbb{R}$ is a reward function, denoted $R(s,a)$ for state s and action a.

The agent does not necessarily know the true state of the POMDP at any given time. Instead noisy observations are made and the agent is able to maintain a *belief* over the true state. We denote a set of r beliefs as $B \subseteq \triangle^n$, with $\triangle^n$ denoting the standard n-simplex. The agent updates a current belief $b \in B$ after taking an action a and making an observation ω to a new belief b' for a state $s \in S$ following:

$$b'(s'|b,a,\omega) = \eta O(a,s',\omega) \sum_{s \in S} T(s,a,s') b(s) \qquad (1)$$

with normalization constant $\eta = Pr(\omega|b,a)^{-1}$. Importantly, let r_z denote the maximum number of non-zero values over all belief vectors $b \in B$.

Agents operate for a number of discrete time steps called the *horizon* $h \in \mathbb{N}$. The agent's reward is reduced by a *discount factor* $\gamma \in (0,1)$ per time step. Infinite horizon ($h = \infty$) POMDPs can often be approximated by some finite horizon. A *policy* $\pi : B \to A$ describes how the agent acts based on its beliefs. We also define the *value function* $V : B \to \mathbb{R}$ as the expected reward at each belief, which is piece-wise linear and convex in this space (Lovejoy 1991). This fact enables us to represent the value function as a collection of α-*vectors* $\Gamma = \{\alpha_1, \ldots, \alpha_x\}$ with each $\alpha_i = [V(s_1), \ldots, V(s_n)]^T$ and $V(s_j)$ denoting the value of state s_j. We record a policy by marking an action with each α-vector, so we have the compact notation: $V(b) = \alpha \cdot b$ and $\pi(b) = a_\alpha \in A$.

Point-Based Solution Methods

Point-based value iteration (PBVI) (Pineau, Gordon, and Thrun 2003) and other belief point-based approaches, such as heuristic search value iteration (HSVI2) (Smith and Simmons 2004) and Perseus (Spaan and Vlassis 2005), do not expand all reachable beliefs from an initial seed belief. Instead, they operate on a different set (e.g., a subset) to avoid the exponential growth of reachable beliefs over the horizon. In PBVI, we have an initial *expand* step (denoted as $expand(\cdot)$ in Algorithm 1) which produces a set of beliefs $B \subseteq \triangle^n$. Then, we apply value iteration over these beliefs, producing α-vectors at each time step t denoted as Γ^t. Formally, this procedure is applied h times (denoted as $update(\cdot)$ in Algorithm 1), given Γ^{t-1}, to produce Γ^t, is given by:

$$\Gamma^t_{a\omega} = \{[V^t_{s_1 a\omega\alpha}, \ldots, V^t_{s_n a\omega\alpha}]^T, \forall \alpha \in \Gamma^{t-1}\}, \quad \forall a \in A, \omega \in \Omega$$

$$\Gamma^t_b = \{r_a + \sum_{\omega \in \Omega} \operatorname*{argmax}_{\alpha \in \Gamma^t_{a\omega}} \alpha \cdot b, \forall a \in A\}, \quad \forall b \in B$$

$$\Gamma^t = \{\operatorname*{argmax}_{\alpha \in \Gamma^t_b} \alpha \cdot b, \forall b \in B\}$$

with variables $V^t_{sa\omega\alpha} = \gamma \sum_{s' \in S} O(a,s',\omega) T(s,a,s') \alpha(s')$, $r_a = \sum_{s \in S} b(s) R(s,a)$, and initial α-vectors be $\alpha(s) = \underline{R}/(1-\gamma)$, for all $s \in S$, with $\underline{R} = \min_{s \in S} \min_{a \in A} R(s,a)$ guaranteeing α-vectors increase (Lovejoy 1991).

The σ-Approximation Method

Our inspiration comes from the realization that: (1) belief dot products are nested throughout PBVI and other algorithms, (2) zero-multiplied values may be skipped, (3) a similar definition of n_s for beliefs might be exploitable, and (4) there is a significant performance improvement in practice when $r_z \ll n$ as opposed to $r_z \approx n$. With these insights, we designed a variant that can be applied to *any* belief-based algorithm that reduces the beliefs from an expand step to be of size $\hat{r}_z \leqslant r_z$ for use within an update step. For the sake of clarity, we focus here on PBVI applications only; however, the algorithm can be easily applied to commonly used value iteration (VI) methods such as HSVI2 or Perseus in a natural way. We call this general algorithm the *σ-approximation*. For brevity, we denote the use of our algorithm on any point-based algorithm with the prefix 'σ' (e.g., σ-PBVI, σ-HSVI2, σ-Perseus, etc.). The σ denotes the measure of approximation, a value that can be computed, with a guarantee that $\sigma \in [1/n, 1]$.

The algorithm separates the true set of beliefs used in the expand step B from the (approximate) set used in the

update step $\hat{B}$. Importantly, each expand step continues to use the true beliefs B. Since our method removes non-zero beliefs, which are small in belief vectors, if we used $\hat{B}$ for expansions, then algorithms that explore *reachable* beliefs might never explore the full set of reachable beliefs. By preserving B for expand, we are able to explore the full set of reachable beliefs, and then approximate these with a bounded size of non-zero values for beliefs in $\hat{B}$ for updates. *Thus, how should we best approximate beliefs in B given the $\hat{r}_z$ constraint*?

Optimal Selection in the σ-Approximation

Let $b \in B$ be any belief point from the expanded set of beliefs B. Let $N = \{1, \ldots, n\}$. Assume we are given a constraint $\hat{r}_z \leqslant r_z \in N$ that denotes the desired maximum number of non-zero belief point values in any belief. Let $\hat{B}$ denote the approximated beliefs of B given the $\hat{r}_z$ constraint. Formally, this constraint guarantees that for $\hat{b} \in \hat{B}$:

$$|\{i \in N | \hat{b}_i > 0\}| \leqslant \hat{r}_z \tag{2}$$

The σ-approximation operates in the following manner. For all beliefs $b \in B$, $b = [b_1, \ldots, b_n]^T$. We sort the belief's values in $O(n \log n)$ time (denoted $sort(\cdot)$ in Algorithm 1). Optionally, this is much faster if: (1) we cleverly expand so the beliefs are already sorted, and/or (2) if we sparsely store beliefs. Let $o_r : N \to N$ denote the resulting *descending* ordering (rank index) of the belief vector's indices after sorting. Let $\hat{I} = \{i \in N | o_r(i) \leqslant \hat{r}_z\}$ be the reduced set of indices of only the top $\hat{r}_z$ with respect to their probabilities. We define the new approximate belief $\hat{b}$, to be added to $\hat{B}$, of the original b, for $i \in N$ as:

$$\hat{b}_i = \begin{cases} \frac{b_i}{\sigma_b}, & \text{if } i \in \hat{I} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

with $\sigma_b = \sum_{i \in \hat{I}} b_i$. This also ensures Equation 2 holds. We let $\sigma = \min_{b \in B} \sigma_b$ denote the overall worst-case approximation error using our method. Interestingly, the definition of $\hat{I}$ implies that the worst-case approximation error is bounded to an interval $\sigma \in [1/n, 1]$. This only arises with $\hat{r}_z = 1$ and a uniform belief b. The procedure is shown in Algorithm 1.

Theoretical Analysis of the σ-Approximation

First, we prove in Proposition 1 that the σ-approximation algorithm yielding $\hat{b}$ from Equation 3 returns the correct optimal approximate belief given the fixed $\hat{r}_z$.

Proposition 1 (Correctness)**.** *For belief $b \in \triangle^n$ and $\hat{r}_z \in N$, for all other beliefs $b' \in \triangle^n$ with the same $\hat{r}_z$ constraint: $|\{k \in N | b'_k > 0\}| \leqslant \hat{r}_z$, we have the property that $\hat{b} \in \triangle^n$ produced by the σ-approximation:*

$$\|\hat{b} - b\|_1 \leqslant \|b' - b\|_1 \tag{4}$$

Proof. Assume by contradiction there exists a $b' \in \triangle^n$ with the $\hat{r}_z$ constraint (Equation 2) such that $\|\hat{b} - b\|_1 > \|b' - b\|_1$. Let $K' = \{k \in N | b'_k > 0\}$. By definition of 1-norm we have:

$$\sum_{i \in \hat{I}} |\hat{b}_i - b_i| + \sum_{i \notin \hat{I}} |b_i| > \sum_{k \in K'} |b'_k - b_k| + \sum_{k \notin K'} |b_k|$$

Algorithm 1 The σ-Approximation Method for basic PBVI.

Require: $\langle S, A, \Omega, T, O, R \rangle$: The POMDP.
Require: $\hat{r}_z$: The desired maximum number of non-zero values.
Require: b^0: The initial belief.
$B \leftarrow expand(b^0)$
$\hat{B} \leftarrow \varnothing$
for $b \in B$ **do**
  $\hat{b} = [0, \ldots, 0]^T$
  for $i \in \{1, \ldots, n\}$ **do**
    $o \leftarrow sort(b_i)$
    $\hat{I} \leftarrow \{i \in N | o_r(i) \leqslant \hat{r}_z\}$
    $\hat{b}_i \leftarrow \begin{cases} \frac{b_i}{\sigma_b}, & \text{if } i \in \hat{I} \\ 0, & \text{otherwise} \end{cases}$
  end for
  $\hat{B} \leftarrow \hat{B} \cup \{\hat{b}\}$
end for
$\Gamma \leftarrow update(\hat{B})$

By rearranging and the definition of $\hat{b}$ in Equation 3:

$$\sum_{i \in \hat{I}} \left| \frac{b_i}{\sigma_b} - b_i \right| + \sum_{k \in K'} |b'_k - b_k| > \sum_{k \notin K'} |b_k| - \sum_{i \notin \hat{I}} |b_i|$$

By Equation 2, $\hat{I} = \{i \in N | o_r(i) \leqslant \hat{r}_z\}$, which by the descending ordering o_r, we guarantee $\hat{b}$ selected the largest $\hat{r}_z$ values from b. Thus, $\forall X \subseteq N$ such that $|X| \leqslant \hat{r}_z$, $\sum_{i \in \hat{I}} b_i \geqslant \sum_{x \in X} b_x$. By rearranging and applying probability normalization requirement: $\sum_{i \notin \hat{I}} b_i \leqslant \sum_{x \notin X} b_x$. With this fact and properties of absolute values, we obtain:

$$\left| \frac{1}{\sigma_b} - 1 \right| \sum_{i \in \hat{I}} b_i - \sum_{k \in K'} |b'_k - b_k| > 0$$

By the definition of σ_b, rearranging, and subadditivity:

$$\left| \frac{1}{\sigma_b} - 1 \right| \sigma_b > \sum_{k \in K'} |b'_k - b_k| \geqslant \left| \sum_{k \in K'} b'_k - b_k \right|$$

By definition of b' and that probabilities sum to 1:

$$\left| \frac{1}{\sigma_b} - 1 \right| \sigma_b > \left| 1 - \sum_{k \in K'} b_k \right| = 1 - \sum_{k \in K'} b_k$$

Rearrange, apply the definitions of $\hat{I}$, K', and σ_b, as well as the properties of absolute values with $\sigma_b \in (0, 1]$ to obtain:

$$1 < \left| \frac{1 - \sigma_b}{\sigma_b} \right| \sigma_b + \sum_{k \in K'} b_k \leqslant \left| \frac{1 - \sigma_b}{\sigma_b} \right| \sigma_b + \sum_{k \in \hat{I}} b_k = \frac{1 - \sigma_b}{\sigma_b} \sigma_b + \sigma_b$$

This implies that $1 < 1 - \sigma_b + \sigma_b = 1$, hence a contradiction is reached. Therefore, $\hat{b}$ is optimal following Equation 4. □

Next, we would like to know how much error (in terms of value at a belief) this approximation adds to PBVI and the other point-based methods. First, Lemma 1 provides an upper bound on the distance from any approximate belief $\hat{b} \in \hat{B}$ to an arbitrary belief $b' \in \triangle^n$. Importantly, this bound is only in terms of the corresponding $b \in B$ for which $\hat{b}$ was an approximation and σ_b.

Lemma 1. *For any belief $b' \in \triangle^n$, and belief $\hat{b} \in \triangle^n$ produced by the σ-approximation of belief $b \in B$, we have:*

$$\|b' - \hat{b}\|_1 \leqslant \|b' - b\|_1 + 2(1 - \sigma_b) \tag{5}$$

Proof. Take any belief $b' \in \triangle^n$ and σ-approximate belief $\hat{b} \in \triangle^n$ for belief $b \in B$. We apply the triangle inequality (using b_i), the definition of $\hat{b}$ (Equation 3), rearrange, apply the definition of σ_b, and simplify.

$$\begin{aligned}
\|b' - \hat{b}\|_1 &= \sum_{i=1}^{n} |b'_i - \hat{b}_i| \leqslant \sum_{i=1}^{n} |b'_i - b_i| + \sum_{i=1}^{n} |b_i - \hat{b}_i| \\
&= \|b' - b\|_1 + \sum_{i \in \hat{I}} \left| b_i - \frac{b_i}{\sigma_b} \right| + \sum_{i \notin \hat{I}} |b_i| \\
&= \|b' - b\|_1 + \left| 1 - \frac{1}{\sigma_b} \right| \sum_{i \in \hat{I}} |b_i| + (1 - \sigma_b) \\
&= \|b' - b\|_1 + \frac{1 - \sigma_b}{\sigma_b} \sigma_b + (1 - \sigma_b)
\end{aligned}$$

which implies $\|b' - \hat{b}\|_1 \leqslant \|b' - b\|_1 + 2(1 - \sigma_b)$. □

We use this result in Lemma 2 and Proposition 2, which proves a bound on σ-PBVI's value error in terms of the density of the *original* belief points $\delta_B = \max_{b' \in \triangle^n} \min_{b \in B} \|b - b'\|_1$ (Pineau, Gordon, and Thrun 2003) and the worst-case approximation error σ. The bound also utilizes $\overline{R} = \max_{s,a} R(s,a)$ and $\underline{R} = \min_{s,a} R(s,a)$. Importantly, this proof extends the original by Pineau *et al.* (Pineau, Gordon, and Thrun 2003) and contains components of it.

Lemma 2 (σ-PBVI One Step Error Bound)**.** *The error ϵ introduced in σ-PBVI when performing one iteration of value backup over $\hat{B}$ instead of B or $\triangle^n$, is bounded by:*

$$\epsilon \leqslant \frac{\overline{R} - \underline{R}}{1 - \gamma} (\delta_B + 2(1 - \sigma)) \tag{6}$$

Proof. We start with the belief $b' \in \triangle^n$ that had the largest error after a σ-PBVI update, and the closest $\hat{b} \in \hat{B}$ (which σ-approximates belief $b \in B$) to b' via a 1-norm, with maximal α-vector α' for b' and would be maximal α-vector $\hat{\alpha}$ at $\hat{b}$.

$$\begin{aligned}
\epsilon &\leqslant \alpha' b' - \hat{\alpha} b' \leqslant \|\alpha' - \hat{\alpha}\|_\infty \|b' - \hat{b}\|_1 && \text{By Pineau } \textit{et al.} \\
&\leqslant \|\alpha' - \hat{\alpha}\|_\infty (\|b' - b\|_1 + 2(1 - \sigma_b)) && \text{By Lemma 1} \\
&\leqslant \frac{\overline{R} - \underline{R}}{1 - \gamma} (\delta_B + 2(1 - \sigma_b)) && \text{By Pineau } \textit{et al.} \\
&\leqslant \frac{\overline{R} - \underline{R}}{1 - \gamma} (\delta_B + 2(1 - \sigma)) && \text{By } \sigma = \min_{b \in B} \sigma_b
\end{aligned}$$

□

Proposition 2 (σ-PBVI Error Bound)**.** *For any set of beliefs $B \subseteq \triangle^n$, σ-approximation $\hat{B}$ of B, and horizon t, the error of the σ-PBVI algorithm $\epsilon_t = \|V_t^{\hat{B}} - V_t^*\|_\infty$ is bounded by:*

$$\epsilon_t \leqslant \frac{\overline{R} - \underline{R}}{(1 - \gamma)^2} (\delta_B + 2(1 - \sigma)) \tag{7}$$

with $V_t^{\hat{B}}$ and V_t^* denoting the estimate and optimal value functions, respectively.

Proof. Again by Pineau *et al.* we have the error ϵ_t at time t bounded as:

$$\begin{aligned}
\epsilon_t &\leqslant \|\tilde{H} V_{t-1}^{\hat{B}} - H V_{t-1}^{\hat{B}}\|_\infty + \gamma e_{t-1} && \text{By Pineau } \textit{et al.} \\
&\leqslant \frac{\overline{R} - \underline{R}}{1 - \gamma} (\delta_B + 2(1 - \sigma)) + \gamma e_{t-1} && \text{By Lemma 2} \\
&\leqslant \frac{\overline{R} - \underline{R}}{(1 - \gamma)^2} (\delta_B + 2(1 - \sigma)) && \text{By geometric series}
\end{aligned}$$

with $\tilde{H}$ and H above above denoting the PBVI and exact update operators, respectively. Note that σ-PBVI has the same value update operator just on a different belief set. □

An interesting facet of this bound is the relation between δ_B and $2(1-\sigma)$. Since beliefs are probabilities, $\delta_B \in [0, 2]$. Similarly, $\sigma \in [1/n, 1]$ implies the other term is on the same range $2(1-\sigma) \in [0, 2(n-1)/n] \to [0, 2]$ as $n \to \infty$. We call this term the *σ-error*. Both also measure an approximation and are orthogonal considerations. In other words, one could have dense beliefs with high σ-error (σ-VI), sparse beliefs with low σ-error (PBVI), sparse beliefs and high σ-error (σ-PBVI), or dense beliefs and low σ-error (VI).

The best-case scenario that will yield the largest performance gains using our σ-approximation consists of domains in which beliefs are almost all collapsed to a few states, but have a lot of very small spread out beliefs over other states. The σ-approximation will then replace these beliefs and efficiently perform updates on most of the denser parts of the belief vector's space.

The theoretical complexity of our PBVI's update equation is $O(n^2 m z r^2)$ in the worst case with $n_s = \hat{r}_z = r_z = n$. In comparison, the σ-approximation has a reduced complexity of $O(mzrn(n + r\hat{r}_z))$ in the worst case with $n_s = n$. Note that the absolute worst-case cost of sorting, $O(rn \log n)$, is greatly overshadowed by the update cost. Additionally, this reduces memory requirements. PBVI requires $O(rn)$ space to store all belief points, whereas σ-PBVI requires $O(r\hat{r}_z)$. While this may not seem like much for smaller problems, larger problems can have beliefs that are spread out over many states. Thus, we can approximate large belief vectors with the σ-approximation, while maintaining the original size of smaller ones. This largely preserves the accuracy of PBVI with a minor modification that vastly improves overall runtime performance, especially if $\hat{r}_z \approx \sqrt{n}$ or $\hat{r}_z \approx \log n$. This observation is empirically supported by our experiments, described in the next section.

Furthermore, parallel implementations of PBVI (multi-core CPU, GPU, or cluster) eliminate the major bottleneck: number of belief points r (Shani 2010; Wray and Zilberstein 2015b). With $n_s \ll n$, one of the remaining major bottleneck variable becomes r_z, which a parallelized σ-PBVI addresses. Finally, communication overhead is one of the biggest factors for parallel algorithms, particularly on clusters. σ-PBVI enables belief points to be transferred over a network on a cluster much faster because of its tunable bounded memory size $O(r\hat{r}_z)$.

Domain							PBVI			σ-PBVI								
							$\hat{r}_z{=}r_z$			$\hat{r}_z{=}\lceil r_z/3 \rceil$			$\hat{r}_z{=}\lceil r_z/10 \rceil$			$\hat{r}_z{=}\lceil r_z/30 \rceil$		
Name	n	m	z	r	n_s	r_z	T	$V(b_0)$	σ	T	$V(b_0)$	σ	T	$V(b_0)$	σ	T	$V(b_0)$	σ
Aloha-10	30	9	3	64	25	10	1.3	106.0	1.0	0.6	105.8	0.64	0.3	101.1	0.36	0.18	98.3	0.18
Aloha-30	90	29	3	128	27	30	82.0	787.4	1.0	34.4	787.3	0.83	13.5	784.5	0.38	7.6	769.1	0.19
Fourth	1052	4	28	256	3	1052	186.4	-60.5	1.0	187.3	-60.5	1.00	183.4	-60.5	1.00	87.3	-60.5	1.00
Hallway2	92	5	17	128	88	88	80.6	0.28	1.0	25.4	0.26	0.34	7.9	0.23	0.10	3.3	0.16	0.03
Rock Sam.	12545	13	2	512	1	256	142.0	-147.1	1.0	71.9	-148.0	0.34	50.2	-145.3	0.10	42.7	-146.9	0.04
Tag	870	5	30	256	5	841	158.7	-25.8	1.0	131.4	-27.7	0.33	131.9	-30.6	0.10	118.8	-30.2	0.03
Tiger Grid	36	5	17	64	5	36	5.04	-0.79	1.0	2.32	-1.06	0.99	0.85	-1.09	0.78	0.48	-1.11	0.69

Table 1: Computation time T (in seconds) for $h{=}50$, initial belief's value $V(b^0)$, and σ averaged over 10 trials for each domain.

Experimentation

We begin with a comparison of σ-PBVI over six standard POMDP benchmark domains, varying the levels of the approximation. Then, we experiment with σ-approximation on a real robot performing simultaneous navigation and localization.

Performance of σ-Approximation on Benchmarks

We implement σ-PBVI to investigate its performance improvements and solution quality. Table 1 shows the results over six larger well-known domains using ranges of $\hat{r}_z$ values. In particular, we compute the base r_z without our σ-approximation, then vary $\hat{r}_z$ to be r_z, $r_z/3$, $r_z/10$, and $r_z/30$. Importantly, this version of PBVI is already much more efficient than a naive implementation that stores all n probabilities for each belief point, even with $\hat{r}_z = r_z$.

Aloha-30, Hallway2, and Tiger Grid all obtain over an order of magnitude improvement. Even the largest domain, Rock Sample (7x8), results in over three times improvement with almost zero error in value $V(b_0)$. Results can be further improved by the user, in terms of time or quality, using the *tunable* parameter $\hat{r}_z$.

Overall, there is a clear trend that larger domains benefit more from this than smaller domains. This is due in part to large spread out belief vectors being relatively rare after expand steps; most reachable beliefs in large domains are actually dense with a few near-zero belief values. Thus, these introduce very small overall error when approximated with smaller belief vectors. Additionally, more complex expand steps (e.g., PEMA) might improve the standard PBVI beliefs, but recall that we are still σ-approximating those beliefs. Thus, the σ-approximation result will also further improve. In summary, our σ-approximation worked well in large domains, introducing low error for greatly reduced computation time.

Application to Robotic Navigation and Localization

We construct a real robotic navigation and localization experiment similar to those found in the few previous real applications of POMDPs (Brooks et al. 2006; Spaan and Vlassis 2004; Pineau et al. 2003). Here, we define a 56 state POMDP: an 8-by-7 abstracted grid. There are 9 actions: all 8 neighboring cells and a stop action. Furthermore, there are 2 observations: "bump" or "no bump". Note that this results in the POMDP's actions and observations allowing for both navigation and localization. The probability of successful forward motion is 0.9, with a slight uniform chance of deviating left and right, as well as not moving. The probability of observing a "bump" is proportional to the average number of obstacles over all possible successor states. The reward is a small -0.05 for non-goal states and 0.0 for the goal. Belief is therefore over the location of the robot as it moves around the world. We assign the initial beliefs to be collapsed with 1.0 probability mass over each state and perform original PBVI expansions afterward selecting maximally "distinct" beliefs (Pineau, Gordon, and Thrun 2003). The σ-approximation is applied on these beliefs.

Figure 2 shows the real world execution of σ-PBVI ($k{=}4$) and PBVI in a maze on a robot platform: the base Kobuki made by Yujin Robot Co., Ltd. with an Nvidia Jetson TX1 made by Nvidia Corporation. As we observe, the actual real-world performance (i.e., the paths and actions taken by the robot shown in Figure 2) is almost the same between σ-PBVI and PBVI. The maze itself was designed to spread belief over the straight "hallways" prior to entering each "room". In practice, the belief spreads out over much more than $k{=}4$ states; however, as observed, the final performance is quite similar.

Conclusion

We provide an approximation algorithm that compresses the non-zero values in belief vectors, solving larger problems faster with bounded additional error. We provide two propositions, and two related lemmas, proving that our σ-approximation is optimal and has bounded error. This is demonstrated in our experiments on six standard domains. Additionally, we implement a POMDP on a real robot in a simultaneous navigation and localization domain, comparing σ-PBVI and PBVI, showing only minor policy differences.

The main contribution of the σ-approximation its applicability to *all algorithms that operates over beliefs*. We envision its use in many other algorithms beyond σ-PBVI, including σ-HSVI2 and σ-Perseus. Also, the σ-approximation is much simpler to implement over other approaches, such as value directed belief state compression (Poupart and Boutilier 2003) or E-PCA methods (Roy, Gordon, and Thrun 2005). We plan to explore broader use of σ-approximation in future work with this foundation established. Finally, we will provide our source code so that others could easily build faster approximate POMDP solvers.

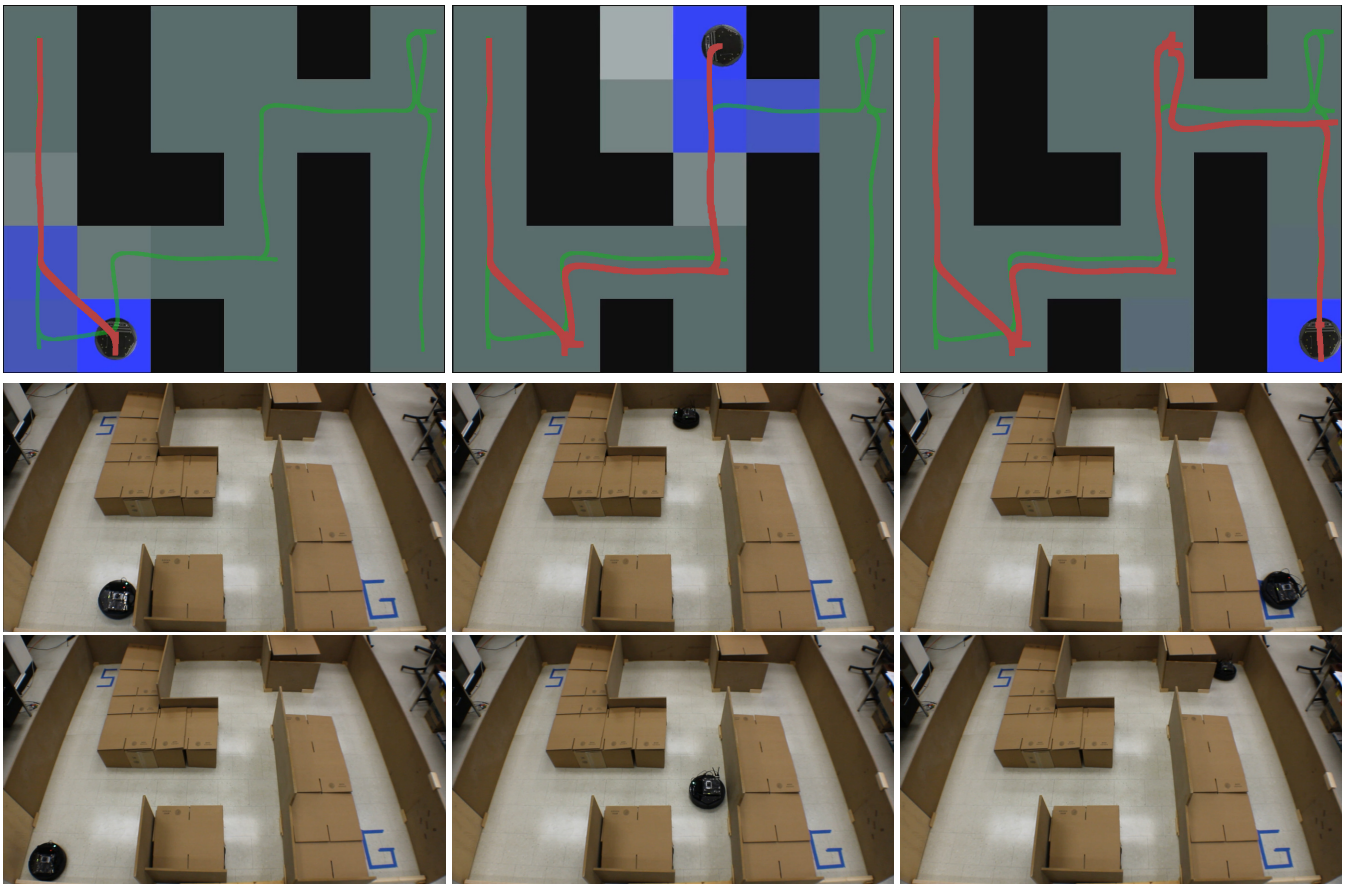


Figure 2: Demonstration of our σ-approximation used on a real robot. Each column of images denotes the ROS output (top) and corresponding real world pictures for σ-PBVI (middle) and normal PBVI (bottom) over time (left to right). The black circle is the robot. Blue and white denote log-probability belief regarding the robot's physical location. Blue visually highlights only the top three highest weights for reference. The red line denotes the σ-PBVI path. The green line denotes the normal PBVI path. (Both paths are from *odometry*.) The start and goal are marked as "S" and "G", respectively. Note the localization attempts in the paths in which the robot intentionally "bumps" the wall to confirm its location and collapse belief.

Acknowledgments We thank the reviewers for their helpful comments. Also, we thank Dirk Ruiken and Samer Nashed for their help with our robot in the experiments.

References

Brooks, A.; Makarenko, A.; Williams, S.; and Durrant-Whyte, H. 2006. Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems* 54(11):887–897.

Castelletti, A.; Pianosi, F.; and Soncini-Sessa, R. 2008. Water reservoir control under economic, social and environmental constraints. *Automatica* 44(6):1595–1607.

Hansen, E. 2008. Sparse stochastic finite-state controllers for POMDPs. In *Proceedings of the 24th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 256–263.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1):99–134.

Lee, W. S.; Rong, N.; and Hsu, D. J. 2008. What makes some POMDP problems easy to approximate? In *Proceedings of Advances in Neural Information Processing Systems 20 (NIPS)*, 689–696.

Lovejoy, W. S. 1991. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research* 39(1):162–175.

Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; and Thrun, S. 2003. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems* 42(3):271–281.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 3, 1025–1032.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research* 27:335–380.

Poupart, P., and Boutilier, C. 2003. Value-directed compres-

sion of POMDPs. In *Proceedings of Advances in Neural Information Processing Systems 15 (NIPS)*, 1579–1586.

Poupart, P.; Kim, K.; and Kim, D. 2011. Closing the gap: Improved bounds on optimal POMDP solutions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 194–201.

Ross, S.; Chaib-draa, B.; and Pineau, J. 2008. Bayes-adaptive POMDPs. In *Proceedings of Advances in Neural Information Processing Systems 20 (NIPS)*. 1225–1232.

Roy, N.; Gordon, G. J.; and Thrun, S. 2005. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research (JAIR)* 23:1–40.

Shani, G.; Poupart, P.; Brafman, R. I.; and Shimony, S. E. 2008. Efficient ADD operations for point-based algorithms. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 330–337.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2007. Forward search value iteration for POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2619–2624.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27(1):1–51.

Shani, G. 2010. Evaluating point-based POMDP solvers on multicore machines. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 40(4):1062–1074.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI)*, 520–527.

Spaan, M. T., and Vlassis, N. 2004. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, 2399–2404.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24:195–220.

Wray, K. H., and Zilberstein, S. 2015a. Multi-objective POMDPs with lexicographic reward preferences. In *Proceedings of the 24th International Joint Conference of Artificial Intelligence (IJCAI)*, 1719–1725.

Wray, K. H., and Zilberstein, S. 2015b. A parallel point-based POMDP algorithm leveraging GPUs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (SDMIA)*, 95–96.

Wray, K. H.; Pineda, L.; and Zilberstein, S. 2016. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 517–523.

Integrated Commonsense Reasoning and Probabilistic Planning

Shiqi Zhang[†] **and Peter Stone**[‡]
[†]Department of Electrical Engineering and Computer Science, Cleveland State University
[‡]Department of Computer Science, The University of Texas at Austin
s.zhang9@csuohio.edu; pstone@cs.utexas.edu

Abstract

Commonsense reasoning and *probabilistic planning* are two of the most important research areas in artificial intelligence. This paper focuses on Integrated commonsense Reasoning and probabilistic Planning (**IRP**) problems. On one hand, commonsense reasoning algorithms aim at drawing conclusions using structured knowledge that is typically provided in a declarative way. On the other hand, probabilistic planning algorithms aim at generating an action policy that can be used for action selection under uncertainty. Intuitively, reasoning and planning techniques are good at "understanding the world" and "accomplishing the task" respectively. This paper discusses the complementary features of the two computing paradigms, presents the (potential) advantages of their integration, and summarizes existing research on this topic.

Introduction

Robots that operate in the real world frequently need to work on complex tasks that require more than one action. Two planning paradigms have been developed for robots that work on such complex tasks: *task planning* and *probabilistic planning*. Task planning algorithms focus on computing a *sequence* of actions, implicitly assuming perfect action executions in a deterministic domain. Probabilistic planning algorithms aim at, in stochastic domains, computing an action *policy* that suggests an action from any state under the uncertainty from the non-deterministic outcomes of robot actions. Examples of non-deterministic action outcomes include opponent moves in chess and results of grasping an object using an unreliable gripper. This paper focuses on *probabilistic planning* in stochastic domains.

The Markov assumption states that the next state only relies on the current state and is independent of all previous states (the first-order case). Accordingly, Markov decision processes (MDPs) and partially observable MDPs (POMDPs) have been developed as probabilistic planning frameworks under full and partial observabilities respectively (Kaelbling, Littman, and Cassandra 1998). When the current world state is not directly observable, the robot needs to make observations to estimate the current state, where the observations are frequently local and unreliable. Accordingly, a belief distribution over all possible states is maintained as the state estimation representation. MDP and POMDP algorithms, e.g., value iteration (Sutton and Barto 1998), Monte Carlo tree search (Kocsis and Szepesvári 2006) and SARSOP (Kurniawati, Hsu, and Lee 2008), help compute a *policy* that enables planning toward maximizing long-term rewards.

Orthogonal to planning, commonsense knowledge is used to refer to the knowledge that is normally true but not always. Such knowledge can be represented in different forms, e.g., as defaults and using probabilities. Commonsense reasoning is concerned with drawing conclusions (or generating new knowledge) using the existing commonsense knowledge. Generally speaking, all knowledge is commonsense knowledge and can be represented in very different forms, such as First-Order Logic (FOL) (Smullyan 1995), Markov Logic Networks (MLNs) (Richardson and Domingos 2006), and Answer Set Programming (Gelfond and Kahl 2014). Such reasoning paradigms are good at drawing (deterministic, probabilistic, or both) conclusions within a static world, but is ill-equipped for planning to achieve long-term goals in dynamic, stochastic domains.

The difficulty of solving MDP and POMDP problems comes from the two major computational challenges of "curse of dimensionality" (a complex robotic task generates a high-dimensional state space) and "curse of history" (a robot often needs to take many actions to reach the goal, resulting in a long planning horizon) (Kurniawati et al. 2011).

> *The main objective of Integrated commonsense Reasoning and probabilistic Planning (IRP) algorithms is to decompose a robot planning problem into two sub-problems: commonsense reasoning and probabilistic planning. Then a commonsense reasoner and a probabilistic planner can be used to focus on the sub-problems of high-dimensional reasoning and long-horizon planning respectively. .*

In what follows, we first present a state space decomposition strategy that paves the way of IRP methods, and then summarize existing research related to this topic.

State Space Decomposition

State space decomposition plays an important role in IRP algorithms. We first define **endogenous** and **exogenous** domain variables for the sake of easier discussion. Endogenous variables are the variables whose values the robot wants to *actively* change or observe (or both). Exogenous variables

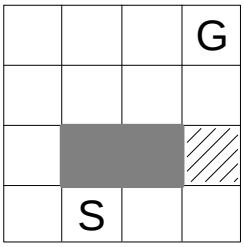


Figure 1: An illustrative example: the robot needs to navigate from its start location (*S*) to the goal (*G*). The hatching area on the right is a near-window area where the robot can be trapped (probabilistically) under sunlight.

are the variables whose values the robot only wants to *passively* observe and adapt to as needed.

Consider a robot navigation problem in a fully-observable 2D grid world shown in Figure 1. The robot can take actions (*North*, *East*, *South*, and *West*) to move toward one of its nearby grid cells, and such actions succeed probabilistically. The hatching cell is a dangerous area to the robot, because, in the mornings, sunlight there can blind its range-finder sensor, causing it unrecoverably lost (probabilistically). In this example, the robot's current location should be modeled as an *endogenous* variable, because its value change needs to be modeled in the planning process, i.e., its value needs to be *actively* changed. Current time (morning or not) should be modeled as an *exogenous* variable, meaning that the robot does not need to change its value in the planning process. However, it is indeed necessary to keep an eye on (*passively* observe) its value, and adjust the probabilistic planner as needed, e.g., reducing the success rate of navigating though the near-window cell when current time is morning.

In principle, all domain variables should be modeled in (PO)MDPs. However, in practice, we usually do not do that, because there is always the trade-off between model completeness and computational tractability. The goal of maintaining two sets of variables is to enable the robot to focus on planning over a long horizon in a relatively small state space (partial space) and reasoning within a relatively large state space (full space). Given full and partial state spaces where the robot reasons and plans respectively, the question will be how the reasoning and planning in two different spaces are connected, which will be discussed next.

Existing Research on IRP Problems

Logical commonsense reasoning has been incorporated into probabilistic planning to compute an informative prior (Zhang, Sridharan, and Bao 2012; Zhang, Sridharan, and Wyatt 2015). In that work, a target search problem was used as the application domain. The robot's noisy observations were modeled using a POMDP, and the belief distribution of the POMDP represents the estimate of the target's position, as the single endogenous variable. The robot moves to different areas in a large office domain to "uncover" the position of the target object. A categorical tree that includes a large number of exogenous variables (such as scanners and printers are office electronics) was constructed using a logical reasoner. As a result, the probabilistic planner is able to focus on a very small partial space that includes only the variable of the target's position, while being able to reason about the target's likely positions within a much larger state space. The gap between commonsense reasoning and probabilistic planning was bridged by using a set of heuristics (such as printers are usually collocated with scanners) to convert deterministic conclusions into a distribution for a POMDP.

In order to better bridge the gap between commonsense reasoning and probabilistic planning, some IRP algorithms have used reasoners that are able to reason about both logical and probabilistic commonsense knowledge. These algorithms and implementations include CORPP (Zhang and Stone 2015) and OpenDial (Lison 2015) that use P-log (Baral, Gelfond, and Rushton 2009) and MLN (Richardson and Domingos 2006) for commonsense reasoning respectively. Their commonsense reasoners are able to directly output a probability distribution for the planner. For instance, a spoken dialog problem was used as the application domain in (Zhang and Stone 2015), where the robot uses unreliable speech recognition to identify the human's request. In that work, the state space decomposition enables the probabilistic planner to focus on only the endogenous variables that are needed for specifying the requests (such as delivering *coffee* for *alice*). All other variables, such as time – people prefer buying coffee in the *mornings*, are modeled as exogenous variables and handled by the commonsense reasoner.

There are other ways of integrating commonsense reasoning and probabilistic planning, where full and partial state spaces are not explicitly differentiated. A refinement-based architecture has been developed for robot reasoning and planning (Sridharan et al. 2015). At the high level, an action language is used for computing a sequence of symbolic actions to deterministically guide the robot behaviors. At the low level, a probabilistic model (a POMDP) is used for physically implementing these actions. As a result, in that work, the high level reasoning layer is able to conduct complicated reasoning tasks, such as explaining history behaviors, that are impossible for probabilistic planners. In another line of research, commonsense reasoning was used for diagnostic tasks and generating explanations, and a hybrid planner allows switching between deterministic and probabilistic planners (Hanheide et al. 2015). POMDP-based planning has been integrated with commonsense learning, where the agent learns from a set of example traces and commonsense knowledge refers to the knowledge based on which a reference policy generates the example traces (Juba 2016).

Probabilistic planning frameworks and algorithms assume a known world model (including world dynamics and robot capabilities). In case of an unknown world, reinforcement learning (RL) algorithms can be used to help an agent learn an action policy by interacting with the environments (Sutton and Barto 1998). Existing research has studied the integration of commonsense reasoning and RL. For instance, relational RL has been used for learning robot action precon-

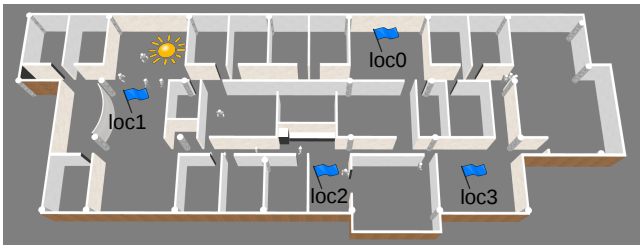


Figure 2: The robot navigation domain that includes four possible navigation goals. Human pedestrians might block the hallway (probabilistically), and sunlight can blind the robot's range-finder sensors (probabilistically).

ditions (affordances), as a kind of commonsense knowledge about robot capabilities (Sridharan, Meadows, and Gomez 2017). In order to reduce the space of exploration in RL, a commonsense reasoner has been used to help the agent to focus on only the *reasonable* actions, significantly increasing the learning rate (Leonetti, Iocchi, and Stone 2016).

In what follows, we summarize our IRP algorithm called iCORPP that dynamically constructs (PO)MDPs to shield exogenous variables from (PO)MDPs while still enabling probabilistic planning to adapt to the exogenous events.

A Summary of iCORPP, an IRP Algorithm

A general description of interleaved CORPP (iCORPP) is available in our recent paper (Zhang, Khandelwal, and Stone 2017). In this section, we directly present an instantiation of iCORPP on a robot navigation problem, and compare it against CORPP, which is similar except that CORPP requires the planner to consider any exogenous variables that could change its transition dynamics. Figure 2 shows the domain map, where the robot needs to visit the four locations that are connected through a corridor. However, human pedestrians can block the way (probabilistically) in the corridor and sunlight can blind the robot's range-finder sensors. It is also known that sunlight only exists in near-window areas when the time is morning and the weather is sunny.

We assume the values of all domain variables are fully observable, so we can use an MDP to construct the planner. If we model only thirty locations in the corridor, there will be ten states in the state space. When we consider each of the locations can be either occupied or unoccupied by humans, the number of states becomes 30×2^{30}. When we further consider each of the locations can be either under sunlight or not, the number of states becomes $30 \times 2^{30} \times 2^{30}$, which is a huge number. This is a small toy domain, and we still have not considered the domain variables of time, weather, and each area is near-window or not.

The whole idea of iCORPP in this domain is to *model only robot position as the endogenous variable for probabilistic planning and all others as exogenous variables to be considered only by the commonsense reasoner.*

Next, we very briefly describe our commonsense reasoner, where the probabilistic transition system of MDP is described in P-log (Baral, Gelfond, and Rushton 2009). In case of exogenous events, our commonsense reasoner dynamically constructs a new MDP that captures the effects of the exogenous variables on the transition dynamics of the endogenous variables.

The navigation domain shown in Figure 2 is defined using sorts `row` and `col`, and predicates `belowof` and `leftof`. We then introduce predicates `near_row` and `near_col` used for specifying if two grid cells are next to each other, where `R`'s (`C`'s) are variables of row (column).

$$\texttt{near\_row}(\texttt{RW}_1,\texttt{RW}_2) \leftarrow \texttt{belowof}(\texttt{RW}_1,\texttt{RW}_2).$$
$$\texttt{near\_row}(\texttt{RW}_1,\texttt{RW}_2) \leftarrow \texttt{near\_row}(\texttt{RW}_2,\texttt{RW}_1).$$
$$\texttt{near\_col}(\texttt{CL}_1,\texttt{CL}_2) \leftarrow \texttt{leftof}(\texttt{CL}_1,\texttt{CL}_2).$$
$$\texttt{near\_col}(\texttt{CL}_1,\texttt{CL}_2) \leftarrow \texttt{near\_col}(\texttt{CL}_2,\texttt{CL}_1).$$

We use predicates `near_window` and `sunny` to define the cells that are near to window and the cells that are actually under sunlight. The rule below is a default stating that: in the mornings, a cell near window is believed to be under sunlight, unless defeated elsewhere.

$$\texttt{sunny}(\texttt{RW},\texttt{CL}) \leftarrow \texttt{near\_window}(\texttt{RW},\texttt{CL}),\ \texttt{not}\ \neg\texttt{sunny}(\texttt{RW},\texttt{CL}),\ \texttt{curr\_time} = \texttt{morning}.$$

While navigating in areas under sunlight, there is a large probability of becoming lost (0.9), which deterministically leads to the end of an episode.

$$\texttt{pr}(\texttt{next\_term} = \texttt{true} \mid \texttt{curr\_row} = \texttt{RW},\ \texttt{curr\_col} = \texttt{CL},\ \texttt{sunny}(\texttt{RW},\texttt{CL}),\ \texttt{curr\_term} = \texttt{false}) = 0.9.$$
$$\texttt{pr}(\texttt{next\_term} = \texttt{true} \mid \texttt{curr\_term} = \texttt{true}) = 1.0.$$

The robot can take actions to move to a grid cell next to its current one: $\texttt{action} = \{\texttt{left},\texttt{right},\texttt{up},\texttt{down}\}$. For instance, given action `up`, the probability of successfully moving to the above grid cell is 0.9, given no obstacle in the above cell.

$$\texttt{pr}(\texttt{next\_row} = \texttt{RW}_2 \mid \texttt{curr\_row} = \texttt{RW}_1,\ \texttt{curr\_col} = \texttt{CL}_1,\ \texttt{belowof}(\texttt{RW}_1,\texttt{RW}_2),\ \neg\texttt{sunny}(\texttt{RW}_2,\texttt{CL}_1),\ \neg\texttt{blocked}(\texttt{RW}_2,\texttt{CL}_1),\ \texttt{curr\_a} = \texttt{up}) = 0.9.$$

iCORPP significantly reduces the complexity of probabilistic planning compared to its one-shot solution, while enabling robot behaviors to adapt to exogenous changes. As an example on complexity, the MDP constructed by iCORPP (thirty positions, five weather conditions and three times) includes only 60 states, whereas the traditional way of enumerating all combinations of attribute values (Boutilier, Dean, and Hanks 1999), produces more than 2^69 states, which cannot be solved (accurately or approximately) in practice.

Experimental Results

Experiments in simulation were conducted using GAZEBO (Koenig and Howard 2004). We used a solver introduced in (Zhu 2012) for P-log programs (except that reasoning about reward was manually conducted) and value iteration for MDPs (Sutton and Barto 1998).

We limit the number of random walkers to be 1 and its speed to be one fifth of the robot's. A goal room is randomly selected from the four flag rooms. Reasoning happens only after the current episode is terminated (goal room is reached). The walker's position is the only exogenous domain change (by temporarily setting the time to be "evening"). We cached policies for both CORPP as the baseline (4 policies) and iCORPP (56 policies).

The walker moves slowly between $loc0$ and $loc2$. Without adaptive planning developed in this work, the robot follows the "optimal" path and keeps trying to bypass the walker for a fixed length of time. If the low-level motion planner does not find a way to bypass the walker within the time, the robot will take the other way to navigate to the other side of the walker and continues executing the "optimal" plan generated by the outdated model. When the robot navigates between $loc0$ and $loc2$, iCORPP reduces the traveling time from about 250 seconds to about 110 seconds, producing a significant improvement.

A comprehensive description of the experimental results is available in our iCORPP paper (Zhang, Khandelwal, and Stone 2017) and this web page includes videos of real-robot experiments. [1]

Conclusions

In this paper, we present the motivation of Integrated commonsense Reasoning and probabilistic Planning (IRP) within the context of robot planning. We summarize existing research on this topic and present our recent work, called iCORPP, that dynamically constructs MDPs and POMDPs for adaptive robot planning. The general idea of IRP algorithms is to decompose the original probabilistic planning problems into the sub-problems of commonsense reasoning and probabilistic planning that respectively focus on "understanding the world" and "accomplishing the task". iCORPP demonstrates that this decomposition significantly reduces the state space where planning is conducted and enables robot to adapt to the value change of exogenous variables without including these variable in planning models.

Acknowledgments

A portion of this work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), AFOSR (FA9550-14-1-0087), Raytheon, Toyota, AT&T, and Lockheed Martin. Peter Stone serves on the Board of Directors of, Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11(1):94.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press.

Hanheide, M.; Göbelbecker, M.; Horn, G. S.; Pronobis, A.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; et al. 2015. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*.

Juba, B. 2016. Integrated common sense learning and planning in pomdps. *Journal of Machine Learning Research* 17(96):1–37.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1):99–134.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Koenig, N., and Howard, A. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. S. 2011. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* 30(3):308–323.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*.

Leonetti, M.; Iocchi, L.; and Stone, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence* 241:103–130.

Lison, P. 2015. A hybrid approach to dialogue management based on probabilistic rules. *Computer Speech & Language* 34(1):232–255.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2):107–136.

Smullyan, R. M. 1995. *First-order logic*. Courier Corporation.

Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2015. A refinement-based architecture for knowledge representation and reasoning in robotics. *arXiv preprint arXiv:1508.03891*.

Sridharan, M.; Meadows, B.; and Gomez, R. 2017. What can i not do? towards an architecture for reasoning about and learning affordances. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press Cambridge.

Zhang, S., and Stone, P. 2015. CORPP: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, 1394–1400.

Zhang, S.; Khandelwal, P.; and Stone, P. 2017. Dynamically constructed (po)mdps for adaptive robot planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

Zhang, S.; Sridharan, M.; and Bao, F. S. 2012. ASP+POMDP: Integrating Non-monotonic Logic Programming and Probabilistic Planning on Robots. In *International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob)*.

Zhang, S.; Sridharan, M.; and Wyatt, J. L. 2015. Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics* 31(3):699–713.

Zhu, W. 2012. *PLOG: Its Algorithms and Applications*. Ph.D. Dissertation, Texas Tech University, USA.

[1] http://eecs.csuohio.edu/~szhang/corpp/