2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 979-8-3503-7770-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/IROS58592.2024.10802363

Camera-Based Belief Space Planning in Discrete Partially-Observable Domains

Janis Eric Freund[1], Camille Phiquepal[2], Andreas Orthey[1,3], Marc Toussaint[1]

***Abstract*—Robots often have to operate in discrete partially observable worlds, where the state of the world is only observable at runtime. To react to different world states, robots need contingencies. To find contingencies, prior work developed the path tree optimization (PTO) method, which computes motion contingencies by constructing a tree of motion paths in belief space. In this paper, we extend upon PTO by enabling camera-based belief space planning through an extension of the open motion planning library (OMPL). By leveraging this extension, we develop an improved camera-based state sampler and an efficient open-source implementation of PTO. This version of PTO supports a virtual camera, non-euclidean state spaces, and different state samplers. We evaluate this improved version of PTO on four realistic scenarios with a virtual camera in up to 10-dimensional state spaces. In our evaluations, we compare PTO both with a default and with the new camera-based state sampler. The results indicate that the camera-based state sampler improves success rates in 3 out of 4 scenarios while having a significant lower memory footprint. Our work thus makes an important step in advancing belief-space planning and provides researchers with an open source tool to use, modify, and benchmark belief-space planning methods.**

I. Introduction

Motion planning is a prerequisite to use robots in search and rescue missions, and for autonomous driving. However, in real-world environments, a robot might have limited sensing ability and can only observe the world partially. While robots might have a precise map of an office building, they might not know if doors are open or closed. This requires robots to make observations to figure out the state of the world. However, once a robot makes an observation, it needs to potentially replan its path to fulfill its task. This is often costly and sub-optimal. Fig. 1 illustrates such a problem, wherein a robot needs to find a box that can be located at three different locations.

In prior work [13], this problem was addressed by computing a path tree to capture contingencies when observing different world states. The resulting path tree optimization method [13] (Original-PTO) showed promising results in realistic scenarios. However, Original-PTO uses a simple observation model which is not based on a realistic camera image. Original-PTO also does not support non-euclidean configuration spaces, which makes it not applicable to a wide set of robots. Original-PTO is also not implemented in the open motion planning library (OMPL) [18], which makes it difficult to compare to other planners and to disseminate the method to the wider research community. Furthermore, samples in Original-PTO are generated randomly, which makes it hard to find the crucial observation points.

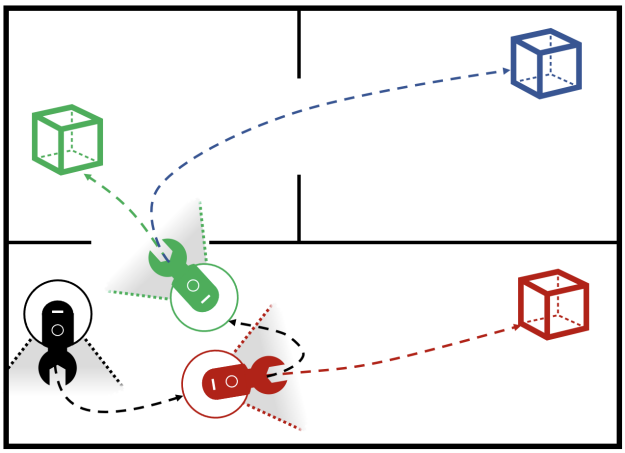


Fig. 1: A planning problem, where the robot (black) needs to find a partially observable box which can be located at three different locations (green, red, blue). The robot can only observe the locations when they are in its field of view (grey). Our planner PTO computes contingencies for robot motions, such that the robot can act optimally, even when the robot either observes the object at a location (green, red robot poses), or if the object is not present.

To address those problems, we develop an extension of OMPL, and leverage this extension to provide an improved version of the PTO planner. PTO differs from Original-PTO by being fully implemented in OMPL, supporting both non-euclidean configuration spaces, and a virtual camera model. Furthermore, a new camera-based state sampler is developed, which generates higher quality samples by sampling configurations where the camera points into the general direction of partially-observable objects. In summary, our contributions are:

1) An extension of OMPL to support a virtual camera model for camera-based belief-space planning.
2) A new camera-based state sampler, which enables more efficient belief space planning through a bias towards configurations at observation points.
3) An implementation of PTO using the OMPL extension, which supports camera-based belief space planning and supports non-euclidean state spaces.

Those contributions are evaluated on four scenarios by integrating the OMPL implementation into the PyBullet [3] simulator.

[1]Technical University of Berlin, Germany
[2]Machine Learning & Robotics Lab, University of Stuttgart, Germany
[3]Realtime Robotics Inc., Boston, MA, USA

II. Related Work

Most work on motion planning in belief space builds on classic sampling-based planners like the rapidly exploring random trees (RRTs) [12], and probabilistic roadmaps (PRMs) [11]. RRTs grow a random tree by sampling the configuration space iteratively. PRMs instead construct a roadmap graph of the free configuration space. While the base variants of RRTs and PRMs are not asymptotically optimal [10], other variants like rapidly exploring random graphs (RRGs) [10], optimal RRTs (RRT*) [10], and batch informed trees (BIT*) [4] are asymptotically optimal. Our approach differs to sampling-based planners by building upon RRGs and using them as the foundation to conduct planning in discrete belief space.

Belief spaces are often introduced because of uncertainty about the world. Motion planning under uncertainty can be formalized using partially observable Markov decision processes (POMDPs) [9]. POMDPs extend Markov decision processes (MDPs) by including situations in which the robot does not know about the state of the world. There are algorithms that try to solve these problems using Gaussian belief spaces [2], [8], [15], [21]. The goal of these algorithms is to minimize the uncertainty of the robot's location. To solve POMDPs, roadmaps in belief space can be planned to deal with uncertainties of motions and observations [15], [21]. Roadmap-based planning can be combined with dynamic replanning to react to changing environments and deviations from the position of the robot [1]. POMDPs have also been combined with a model predictive control step to account for uncertainties in the world state [19]. Another area where POMDPs have been successfully applied are task and motion planning (TAMP) problems, where POMDPs have been used to reduce uncertainties [5], [7], and to create paths across different modes corresponding to specific tasks [20]. Those approaches are complementary to our approach in that they plan in continuous belief spaces. Instead, we consider problems involving discrete beliefs about the state of the environment and finite observations.

The closest work is the Original-PTO method by Phiquepal et al. [13]. Based upon earlier works [14], this planner first introduces the idea of planning path trees in belief space. Our PTO method builds directly upon this work and adds substantial improvements as stated in our contributions.

III. Discrete Belief Space Planning

A belief space planning problem for a robot R is defined by a robot state space $X = Q \times B$, consisting of the configuration space Q, and a belief space B, together with a world W, and a hypothesis space H. The robot configuration space C is a fully-observable n-dimensional space representing robot configurations $q \in Q$. The world W is an environment in which objects and goals can have different states, as depicted in Fig. 2. It is initially unknown in which state the world W happens to be. The hypothesis space H is the finite, discrete set $h \in H$ representing all possible states in which the world W can be. Finally, the belief space B represents probability distributions $b \in B$ over the world hypothesis space H with B being the set of all probability distributions on H given an initial belief $b_0 \in B$.

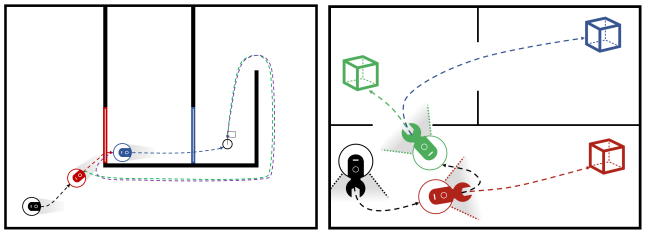


Fig. 2: We support two world states. **Left:** A world with partially observable objects, namely two doors (red, blue). The robot starts at the black state and needs to reach the white flag. The resulting path tree has two observation points (red, blue), at which the robot observes the state of the doors (open or closed) and plans accordingly. **Right:** A world with a partially observable goal state (red, green, or blue box is present). The resulting path tree has two observation points (green, red), at which the robot updates its path according to the observation made (box exists or not).

The robot R can explore the world through observations $o \in O$. An observation at state $x \in X$ is a function from the camera image and the belief of the robot to a discrete output if an object state has been detected. We assume that observation outputs are binary, i.e. we either detect an object state or we do not, which leads to a finite belief space B.

The goal of discrete belief space planning is to create a path tree ψ. A path tree ψ is a directed tree, where edges are paths in the state space X having either the same belief state (movement edges), or the same configuration (observation edges). Nodes are observation points at which the belief state of the robot changes. This path tree represents contingencies for the robot, such that for every observation outcome (e.g. an object is present or not), the robot has a possible option of how to move in the world W.

A. Optimization Objective

We are not only interested in finding some path tree, but in finding an optimal path tree ψ^* over all possible path trees [13]. Our optimization objective is constructed as follows:

$$\psi^* = \underset{\psi}{\operatorname{argmin}} \sum_{(u,v)\in\psi} C(u,v)p\left(v \mid \psi, b_0\right), \tag{1a}$$

s.t.

$$\forall h \in H, \exists l \in \mathcal{L}(\psi) \mid G(l), \tag{1b}$$

$$\mathcal{V}(u,v), \forall (u,v) \in \psi, \tag{1c}$$

$$b_v(h) = \frac{p(o \mid h) b_u(h)}{\sum_{h'} p\left(o \mid h'\right) b_u\left(h'\right)}, \forall h \in H, (u,o,v) \in \psi \tag{1d}$$

whereby $C(u,v)$ is the cost of the edge that connects the nodes u and v. The objective defined by Eq. (1a) is to find an optimal path tree ψ^* that minimizes the sum of the costs $C(u,v)$ over all pairs of nodes in a path tree ψ, while also

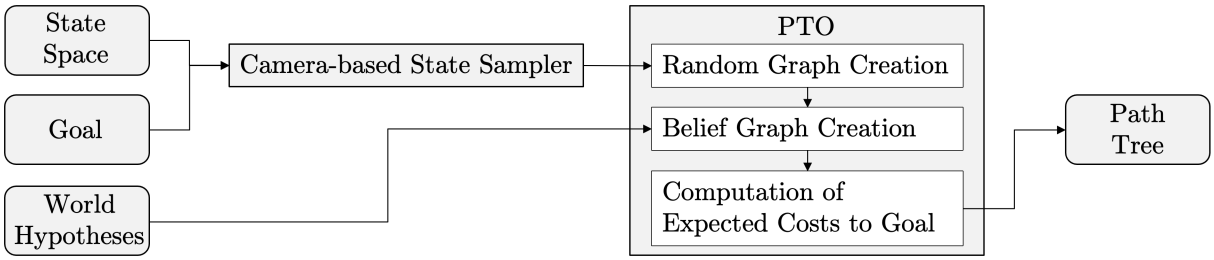


Fig. 3: Structure of the PTO algorithm. The respective algorithm parts of PTO are explained in sections IV-A, IV-B, and IV-C.

taking into account the conditional probability of reaching the node v given an initial belief b_0. Also, some constraints apply: Constraint (1b) makes sure that for each state $h \in H$, it must exist at least one leaf node $l \in \mathcal{L}(\psi)$ that satisfies the goal condition given by $G(l)$. $\mathcal{L}(\psi)$ gives the set of all leaf nodes of the path tree ψ. This condition ensures the completeness of the path tree. Furthermore, the constraint (1c) checks if all edges $(u, v) \in \psi$ are valid as given by $\mathcal{V}(u, v)$. The constraint (1d) specifies that the belief state $b_v(h)$ of the node v should be consistent with the belief $b_u(h)$ of its parent node u when making the observation o while transitioning from u to v. $p(o|h)$ is the observation model.

IV. Path Tree Optimization Method

The PTO planner is composed of three steps as shown in Fig. 3. First, a random graph G_{random} on the configuration space is created. This graph contains information about validity of nodes and edges in different world states. Second, a belief graph G_{belief} on the state space (configuration space plus belief space) is created which adds edges on nodes where a belief change occurs. Finally, we use a dynamic programming search on the belief graph to compute optimal expected cost-to-go values for all nodes. Using those cost-to-go values, we eventually extract the path tree.

A. Random Graph Creation

In the first step, a rapidly exploring random graph [10] G_{random} on the robot configuration space is created. This graph is annotated with additional information: For each configuration, we store a list of all partially observable objects which can be seen. For each edge, we store a list of world states in which this edge is valid.

The process of the random graph creation is illustrated in Fig. 4a. It consists of two phases. First, we sample a random configuration in the robot configuration space, and we sample a random world state. It is then checked if the configuration is valid in this particular world state. If it is valid, we add it to the graph. Afterwards, we search for all observable objects in the world from the current configuration. This information is stored for use in the belief graph creation (Sec. IV-B).

Second, a neighborhood search is conducted to add edge connections to all viable neighbor configurations in the graph. This is accomplished by iterating over all world states, and getting the nearest configurations for each world state in a specific radius R. For each nearest configuration, we check if the edge is valid in the world state. If it is valid, we add the edge to the graph, or update an existing edge by adding the valid world state to the edge itself.

This iterative process is stopped once a planner terminate condition is reached. This can be a number of iterations, or a timeout. The final random graph consists of nodes and edges on the configuration space, together with the information about valid world states.

B. Belief Graph Creation

In the second step, we use the random graph G_{random} to create a belief space graph G_{belief}. This is illustrated in Fig. 4b. The belief graph G_{belief} extends the random graph to the state space by creating nodes and edges, which both contain configurations and beliefs.

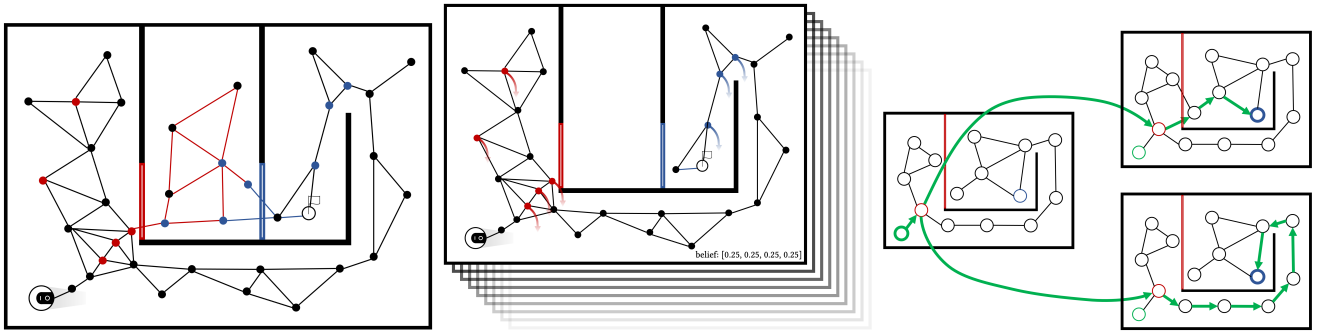
The process of creating a belief graph consists of three phases. First, we iterate over all vertices v in G_{random}, and over all belief states $b \in B$. For each vertex belief pair (v, b) we check if the vertex is reachable in the given belief b. This is accomplished by checking if there is at least one edge in which b is valid. If there is at least one, we add the pair (v, b) to the belief graph.

In the second phase, nodes of the belief graph are connected with edges. Those edges are created by iterating over all edges on the random graph G_{random}. For each edge, we compute the compatible beliefs for the valid world states. For each compatible belief, we add one edge to the belief graph, which connects the nodes of the same belief. After the second phase, there is one graph for each belief, and those graphs are disconnected.

In the third phase, all individual belief graphs are connected to each other. This means we add observation edges, where only the belief, but not the configuration changes. For this, we iterate over all vertices in the current belief graph G_{belief}. For each vertex, we analyse the observable objects from this configuration, and we compute the beliefs that result if the objects are observed at the given belief. For each of those beliefs, we then add an observation edge. For example, if there is an observable object at the configuration, we would add one connection to the belief state where the door is observed as closed, and one edge to the belief state where the door is observed to be open.

C. Extraction of Path Tree

Once the belief graph G_{belief} is constructed, we use a Dijkstra-like algorithm [17] to compute the optimal cost-to-go values for each vertex, from which we extract the path tree



(a) Creating a random graph G_{random} **(b)** Creating a belief graph G_{belief} **(c)** Extracting the path tree ψ

Fig. 4: The three steps of the PTO method. **Left:** A random graph is constructed on the configuration space, with information in which world states (■ [1,2,3,4], ■ [1,3], ■ [1]) an edge is valid. **Middle:** The random graph is extended to the belief space by adding edges whenever an observation changes the belief. **Right** Expected costs are calculated and the path tree is extracted from the belief graph.

ψ. The cost-to-go is computed using dynamic programming by Bellmann updates for each vertex in the graph to all goal vertices.

The path tree extraction is done recursively. Given the belief graph G_{belief} with cost-to-go values, we first compute a list containing all goal vertices. Those vertices are added to the path tree. Then, for each goal vertex, we add the next best node with the lowest cost-to-go value. This process is continued until we either reach the start vertex, or we reach a branching point, where two paths are joined along a single vertex. Once the method terminates, we return the path tree which can then be used by the robot to move through the world towards the goal.

V. Camera-based State Sampler

In Original-PTO, the random graph is constructed through uniform sampling. However, uniform sampling depends on chance whether a partially observable object is seen. Since the potential locations of all partially observable objects are known to the planner, we can take advantage of this, by developing a *camera-based state sampler*. This sampler creates configurations in which the robot's camera is directly looking towards a partially observable object that exists in the current world state.

The camera-based state sampler is shown in Alg. 1. First, the algorithm calls the function GetRandomObject (Line 1), which returns a randomly selected object as the target object, i.e. the object that should be seen by the camera. Then the position of the target object is queried (Line 2), and a random workspace point is sampled (Line 3), at which we like our camera to be positioned.

Given this information, we compute a camera frame. We first compute a normalized direction pointing towards the selected objects from the workspace point (Line 4, 5). Afterwards, a frame represented by a rotation matrix is computed, which points its z-axis towards the object, while keeping the camera horizontal (Line 6-9). This is done by first computing

Algorithm 1: Camera-based State Sampler

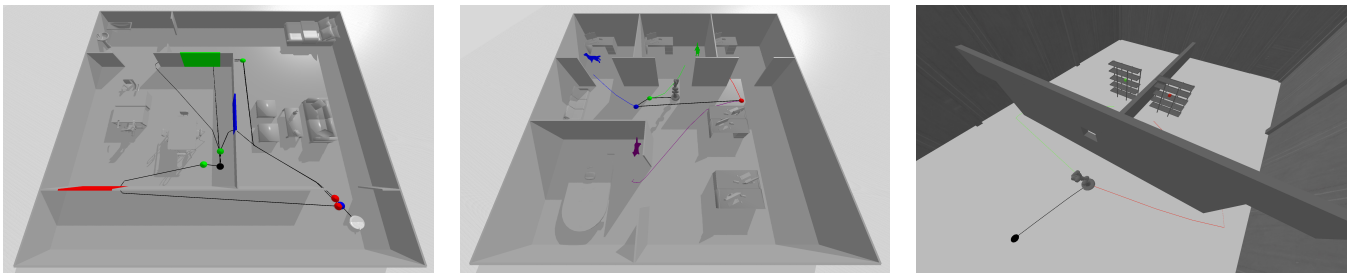
Data: World W
Result: State q
1 $object \leftarrow$ GetRandomObject(W);
2 $T_{object} \leftarrow$ GetObjectCenter(object);
3 $T_{camera} \leftarrow$ GetRandomWorkspacePoint();
4 $c_z \leftarrow T_{object} - T_{camera}$;
5 Normalize(c_z);
6 $e_z \leftarrow$ [0, 0, 1];
7 $c_x \leftarrow$ CrossProduct(e_z, c_z);
8 $c_y \leftarrow$ CrossProduct(v, c_x);
9 $R_{camera} \leftarrow$ RotationMatrix([c_x, c_y, c_z]);
10 $q \leftarrow$ CalculateInverseKinematics($\{T_{camera}, R_{camera}\}$);
11 **return** q;

the x-axis of the frame as the cross-product of the direction c_z and the unit vector that points in the positive z-direction (Line 6, 7). The y-axis of the frame consists of the cross-product of the direction and the x-axis of the frame (Line 8). This frame is represented as a rotation matrix located at the random workspace point (Line 9).

Finally, this camera frame is used in an inverse kinematics (IK) solver (Line 10). Based on the camera frame, the IK solver computes a joint configuration that leads to the camera pointing towards the target object. This configuration is eventually returned (Line 11).

This camera-based sampler can sample states from which the camera can directly observe partially observable objects. This leads to an increase in states in the random graph that are associated with observations. By combining this sampler with a uniform sampler, we can guarantee asymptotic optimality of PTO.

The camera-based sampler also implements a method that replaces the uniform sampling of the base position with values that can be passed to the method as arguments. This can be used in the random graph creation described in section



(a) Door Environment **(b)** Search and Rescue Environment **(c)** Warehouse Environment

Fig. 5: The door, search and rescue, and warehouse environments used for benchmarking.

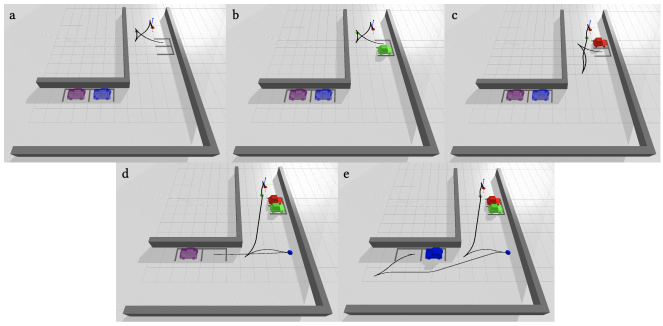


Fig. 6: Result on parking environment with a Reeds-Shepp car.

IV-A. Before sampling randomly, new states can be sampled at the corresponding start position of the base while the robot observes the existing objects in this configuration.

VI. Experiments

We evaluate our implementation of PTO with the default sampler against PTO with the camera-based state sampler on four scenarios. Each run is repeated 12 times and we report on average success rate and cost. The belief for all experiments is initialized with an uniformly distributed initial belief. The experiments are performed on a laptop with 16 GB of RAM using Ubuntu 20.04. PyBullet is used in version 3.2.5. The OMPL library is extended using version 1.5. The code is open source and can be accessed via github[1]. For each scenario, a different timeout has been used, which reflects realistic requirements, and makes the differences in solutions better visible. We showcase solution costs only after 50% of scenarios have been solved.

A. Door Environment

The first environment represents a living area with three doors (Fig. 5a). All doors are partially observable, i.e. the three-dimensional robot needs to observe if they are open or closed.

A resulting path tree is shown in Fig. 5a. Following the trajectory, the robot first observes the blue door. If the door is open, the robot takes a path through the door and reaches the goal. If the blue door is closed, the robot next observes the red door. If the door is open, the robot directly moves to the goal. If the red door is closed, the robot instead moves until it can observe the green door. If it is open, the robot takes the path going through the green door. Otherwise, the robot takes the path avoiding all doors.

Figure 7 shows the performance of the PTO planner using the default sampler versus using the camera-based sampler. For the camera-based state sampler, the robot reaches a 100% success rate after around 500 seconds, which is a 60% improvement to the around 800 seconds needed for the default sampler. Both solution costs improve over time with a slight advantage for the camera-based state sampler.

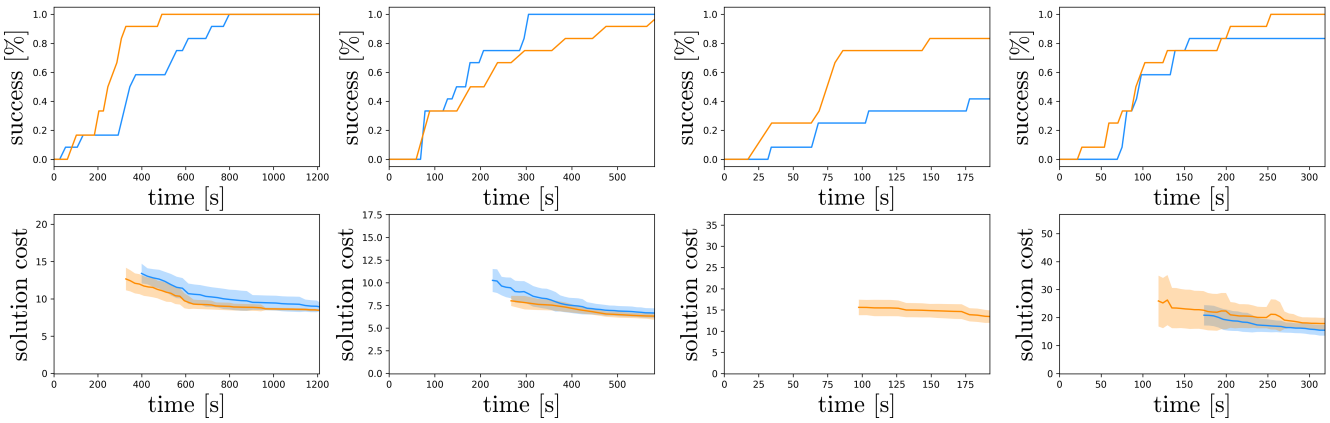
B. Search and Rescue Environment

The second experiment is a search and rescue scenario. It consists of planning a path for a Franka robot arm [6], mounted on a mobile base. A camera is attached to the end-effector of the robot. Together, the robot has ten dimensions, three corresponding to the base and seven corresponding to the arm. In the scenario, a dog must be rescued from an office. Potential locations of the dog are known in advance, but the actual position of the dog is unknown at planning time.

A path tree, planned by the PTO planner, is visualized in Figure 5b. Since the dog can only be at one location at once, there are exactly four different world states. First, the robot observes the potential locations of the dog in the following order: green, blue, purple, and red. If the observed location reveals the presence of the dog, the robot moves directly towards the site. Otherwise, it continues the path tree to the next observation point.

The results are shown in Fig. 7. The default state sampler reaches a success rate of almost 92% after approximately 600 seconds, but the camera-based sampler finds all solutions after 300 seconds. The camera-based sampler converges relatively quickly with the standard deviation approaching zero, while the costs using the default sampler are higher and associated with a greater standard deviation. This can be explained by the 10-dimensional configuration space, making it less likely to sample a state with an informative camera position.

[1]https://github.com/janisfreund/path-tree-optimization



(a) door environment (b) search and rescue (c) warehouse environment (d) parking environment

Fig. 7: We compare PTO with a uniform random state sampler ■ against PTO with the camera-based state sampler ■.

C. Warehouse Environment

The third experiment is an item retrieval task in a warehouse as shown in Fig. 5c. A path is planned for the Franka robot arm which is mounted on a mobile base, as in the second experiment. The location of the item is not known at planning time, but the planner knows about two potential positions. Both item locations are occluded by a wall, so the robot is not able to directly observe them.

A path tree planned by PTO is shown in Fig. 5c. There are only two possible states of the world: Either the object is at the red or the blue location. In the planned path tree, the robot directly moves towards the window. From this state, it is able to observe the blue item. If it exists, the robot moves through the left door and picks up the box. If the blue object does not exists, the robot moves through the right door.

The benchmark results are shown in Fig. 7. In the allocated time frame, the camera-based state sampler solves 80% of the cases, while the default sampler solves 40%. Only the solution cost for the camera-based state sampler is shown (after 50% cases were solved). It can be seen that both the mean cost and the deviation decrease over time, indicating that the the planner converges to a low-cost solution. This scenario demonstrates the benefits of using the camera-based sampler, since finding the window location is beneficial to allow the robot to quickly make the correct decision.

D. Parking Environment

In the final environment, we evaluate PTO on a parking scenario using the non-euclidean Reeds-Shepp car's state space [16] as shown in Fig. 6. The task is to park in one of four parking lots. The parking lots can be free or occupied.

Fig. 6 also shows a planned path tree. For better visibility, the branches of the path tree are shown in different images. First, the car observes the parking lot associated with the red car. If it is empty, the car drives towards this parking space (Fig. 6 a and b). If the red car exists, the car observes the parking lot of the green car next. If the green car does not exists, it directly parks in that lot (Fig. 6 c). If it exists, the car observes the spot of the blue car. It parks there if it cannot detect the blue car (Fig. 6 d). Otherwise, it drives to the last remaining spot (Fig. 6 e).

The benchmark results are shown in Fig. 7. The samplers are comparable, although the default state sampler only find 80% of the solutions after 300s. It is noticeable that the standard deviation of the cost is relatively high for both samplers, but converges over time to a similar value.

VII. Conclusion

We presented PTO, an improved planner based on prior work by Phiquepal et al. [13]. PTO has been shown to solve problems with multiple partially observable objects, and problems with multiple partially observable goal regions. We implemented PTO in the open motion planning library (OMPL), added support for simulated cameras, and for non-euclidean state spaces. An additional innovation is the novel camera-based state sampler, which biases sampling towards configurations at which important observations can be made. In our evaluations, we showed PTO to converge to near optimal solutions.

While we believe this to be a significant improvement, there are two remaining limitations. First, partially observable objects can only take exactly two states. However, objects might be associated with more than two states, like reconfigurable tools, or different robot attachments. PTO could accommodate this by adding an additional detection method of object states from the camera image. Second, the PTO method uses internally three sequentially executed phases. However, this can increase runtime because belief space sampling is decoupled. This could be remedied by developing a unified version to directly sample in belief space.

Despite limitations, PTO was shown to be a well-suited planner for motion planning in discrete partially observable environments. PTO has strong guarantees like asymptotic optimality, and is implemented in OMPL, such that the robotics community can benefit from and build upon this work.

REFERENCES

[1] Ali-akbar Agha-mohammadi, Saurav Agarwal, Aditya Mahadevan, Suman Chakravorty, Daniel Tomkins, Jory Denny, and Nancy M. Amato. Robust online belief space planning in changing environments: Application to physical mobile robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 149–156. IEEE, May 2014.

[2] Adam Bry and Nicholas Roy. Rapidly-exploring Random Belief Trees for motion planning under uncertainty. In *2011 IEEE International Conference on Robotics and Automation*, pages 723–730. IEEE, May 2011.

[3] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[4] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees (BIT$*$): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. *arXiv*, May 2014.

[5] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online Replanning in Belief Space for Partially Observable Task and Motion Problems. *arXiv*, November 2019.

[6] Sami Haddadin, Sven Parusel, Lars Johannsmeier, Saskia Golz, Simon Gabl, Florian Walch, Mohamadreza Sabaghian, Christoph Jähne, Lukas Hausperger, and Simon Haddadin. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29(2):46–64, 2022.

[7] Dylan Hadfield-Menell, Edward Groshev, Rohan Chitnis, and Pieter Abbeel. Modular task and motion planning in belief space. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4991–4998. IEEE, 2015.

[8] Qi Heng Ho, Zachary N. Sunberg, and Morteza Lahijanian. Gaussian Belief Trees for Chance Constrained Asymptotically Optimal Motion Planning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11029–11035. IEEE, May 2022.

[9] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

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

[12] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.

[13] Camille Phiquepal, Andreas Orthey, Nicolas Viennot, and Marc Toussaint. Path-tree optimization in discrete partially observable environments using rapidly-exploring belief-space graphs. *IEEE Robotics and Automation Letters*, 7(4):10160–10167, 2022.

[14] Camille Phiquepal and Marc Toussaint. Control-Tree Optimization: an approach to MPC under discrete Partial Observability. In *International Conference on Robotics and Automation (ICRA)*, pages 9666–9672. IEEE, May 2021.

[15] Sam Prentice and Nicholas Roy. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In *Robotics Research: The 13th International Symposium ISRR*, pages 293–305. Springer, 2011.

[16] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[17] Moshe Sniedovich. Dijkstra's algorithm revisited: the dynamic programming connexion. *Control and cybernetics*, 35(3):599–620, 2006.

[18] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. https://ompl.kavrakilab.org.

[19] Carl Hynén Ulfsjöö and Daniel Axehill. On Integrating POMDP and Scenario MPC for Planning under Uncertainty – with Applications to Highway Driving. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 1152–1160. IEEE, June 2022.

[20] William Vega-Brown and Nicholas Roy. Asymptotically optimal planning under piecewise-analytic constraints. In *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 528–543. Springer, 2020.

[21] Dongliang Zheng, Jack Ridderhof, Panagiotis Tsiotras, and Ali-akbar Agha-mohammadi. Belief Space Planning: A Covariance Steering Approach. *arXiv*, May 2021.