Path-Tree Optimization in Discrete Partially Observable Environments using Rapidly-Exploring Belief-Space Graphs

Camille Phiquepal[1], Andreas Orthey[2], Nicolas Viennot[3] and Marc Toussaint[2]

***Abstract*— Robots often need to solve path planning problems where essential and discrete aspects of the environment are partially observable. This introduces a multi-modality, where the robot must be able to observe and infer the state of its environment. To tackle this problem, we introduce the Path-Tree Optimization (PTO) algorithm which plans a *path-tree* in belief-space. A path-tree is a tree-like motion with branching points where the robot receives an observation leading to a belief-state update. The robot takes different branches depending on the observation received. The algorithm assumes a deterministic observation model and is composed of three main steps. First, a rapidly-exploring random graph (RRG) on the state space is grown. Second, the RRG is expanded to a belief-space graph by querying the observation model. In a third step, dynamic programming is performed on the belief-space graph to extract a path-tree. The resulting path-tree combines exploration with exploitation i.e. it balances the need for gaining knowledge about the environment with the need for reaching the goal. We demonstrate the algorithm capabilities on navigation and mobile manipulation tasks, and show its advantage over a baseline using a task and motion planning approach (TAMP) both in terms of optimality and runtime.**

I. Introduction

Motion planning problems often assume that the environment is fully observable. However, in realistic scenarios, a robot will only have limited access to the environment through its sensors and can therefore only partially observe the state of the world. An example would be a robot, which has a blueprint of a building, but without knowing which doors are open or close. Another example would be a robot in a warehouse, where it has to get a package, but does not know in which location the package resides. In such environments, the robot has a countable number of hypotheses about the environment, which it can observe only in close proximity to certain objects. We say that such environments are partially observable and *multi-modal*.

Multi-modal environments require novel, more general approaches to planning. First, this requires a plan, which takes all possible outcomes into account, such that we have *contingencies* to control the robot irrespective of what observations were made. If a door is closed, the robot should know how to proceed. Second, this requires *optimal motions*, which minimize the expected cost to solve the problem—not only contingencies, but optimal behavior over all modes.

Those requirements are difficult to model with existing frameworks based on task and motion planning (TAMP) or optimization-based methods. Instead, we develop an integrated solution based on belief-space planning, which we call path-tree optimization (PTO). PTO extends previous research work [1][2] which introduce tree-like motions computed with optimization based-methods in the respective sub-fields of Task and Motion Planning (TAMP) and Model Predictive Control (MPC). The concept of tree-like motions is extended here to sampling-based path planning, and leverages the strong guarantees of asymptotic optimality to tackle problems challenging for pure optimization-based methods (e.g. due to a high number of local minima). In addition, unlike [1], in which the observation actions are planned at the task level, the presented approach incorporates the observation model and the belief-state inference on the motion planning level directly, leading to a unified algorithm. Accordingly, the main contributions of the paper are:

- a general sampling-based algorithm planning optimal path-trees for partially observable multi-modal problems.
- the demonstration of the applicability of the method on low-dimensional navigation tasks, as well as on high-dimensional mobile manipulation problems.

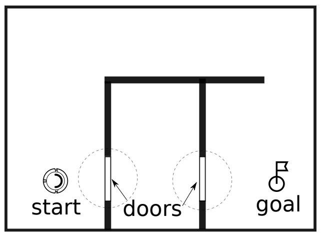
II. Related work

Planning over multi-modal beliefs of the environment has been investigated in the context of active perception. This includes algorithms for SLAM [3] or active object classification [4]. Those methods concentrate primarily on finding good viewpoints, but don't address the motion planning problem. In contrast, with PTO, we concentrate instead on the path planning challenges.

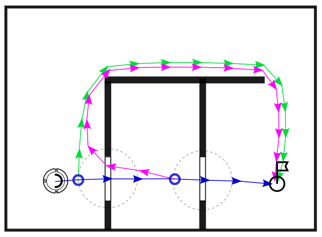
Path planning under partial observability of the environment is related to the broader topic of perception-aware path-planning for which adaptations of classical sampling-based planners [5][6] have been developed. For example, [7][8][9] optimize paths on a transition-system grown in a sampling-based fashion. The optimization is w.r.t. an information gathering objective, but does not explicitly model belief-states.

Another group of algorithm considers POMDP-like problems with continuous observations and gaussian belief-states, often to account for localization uncertainty. In particular, our approach shares ideas with BRM [10], FIRM [11][12], or CS-BRM [13]. The main idea we share is the sampling of a roadmap on which belief-space planning is performed. The main difference lies in the kind of problems we tackle: environment multi-modality in our case vs. continuous uncertainty. This impacts the assumptions made to ensure tractability: countable number of belief states with a deterministic

[1]Machine Learning & Robotic Lab, University of Stuttgart, Germany `camille.phiquepal@ipvs.uni-stuttgart.de`
[2]Learning and Intelligent Systems Lab, TU Berlin, Germany `toussaint@tu-berlin.de`
[3]`nicolas.viennot@cs.columbia.edu`



(a) Example of planning problem



(b) Reactive Path-tree

Fig. 1: The robot has to reach the goal region. The doors state (open vs. closed) is only observable in their vicinity. PTO plans path-trees with observation points (blue circles) and path-contingencies (blue, green, magenta) reacting to the observations.

observation model vs. gaussian assumption. From this also follows the different nature of the computed solutions: contingent reactive path-tree vs. paths for [10] or sequences of controllers [11][12][13].

In the field of object manipulation ConCERRT [14] plans policies reacting to contacts, which reduce the uncertainty over the relative position of the object to manipulate. The contingent-planning aspect is similar to our approach. However, we aim for optimality, whereas [14] does not consider the path costs. The use cases also differ: mobile manipulator with vision based observations vs. table-top object manipulation. In addition, we aim for short planning times (a few seconds) compared to ten minutes or more in [14].

Finally, an analogy can be drawn between PTO and Task and Motion Planning (TAMP). In particular, for roadmap-based approaches like [15] or [16], where a transition system is built across modes. The main difference resides in the meaning of the modes: in PTO modes are belief-states and mode-switches happen when an observation changes the belief-state, whereas modes typically represent different tasks in TAMP. The difference in planning problems also lead to the optimization of contingent path-trees, optimized over all modes simultaneously, instead of paths.

III. Problem formulation

We optimize policies in a context of mixed-observability. The robot state is fully observable, but discrete and essential parts of the environment are only partially observable. In the example of the Fig. 1a this models the fact that each door can be open or closed.

To capture this mixed observability, we adopt a compound state representation where a state is composed of 2 parts:

- $x \in \mathbb{R}^n$ is the robot state from a continuous space, it is fully observable.
- $s \in \mathcal{H}$ is a discrete state from a finite set of world hypotheses $\mathcal{H}$.

In the problem introduced in Fig. 1a the variable s can take four possible values corresponding to the possible states of the environment as illustrated on Fig. 2.

The robot is not oblivious about the likelihood of each state hypothesis. On the problem introduced on Fig. 1a, when the robot is in the vicinity of a door (symbolized with the circles), it receives an observation indicating whether the door is open or not.

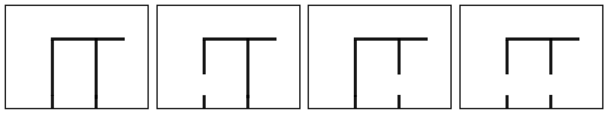


Fig. 2: Partially observable discrete state: With 2 partially observable doors, there are 4 possible states of the environment.

Planning is performed in belief-space, where a belief-state b is a probability distribution over the possible states.

A. Path-tree

The algorithm consists in planning a path-tree. A path-tree consists of paths that branch into multiple possible paths where the robot receives an observation leading to a belief-state update. As schematically shown on Fig. 1b, the path-tree starts from a single root node and finishes with leaf nodes satisfying the goal condition. The different branches of the path-tree are the different planned contingencies. To be complete, a path-tree must end inside the goal region for all $s \in \mathcal{H}$.

In addition, we assume the observation model to be binary, meaning that $p(o|x,s) \in \{0.0, 1.0\}, x \in \mathbb{R}^n, \forall s \in \mathcal{H}, \forall o \in O$, where o is an observation from the observation space O. This property is important. It guarantees that the number of belief-states is finite and can be enumerated which is needed for the graph expansion to belief-state presented in IV-C. On Fig. 1a, it means that observations indicate whether the door is open or not without uncertainty. In addition, it typically implies that the number of branchings N_o stays small compared to the total number of states N on the path-tree. In other words $N_o \ll N$. It can be understood easily on the example: once an observation has been received for a door, the agent knows with certainty if the door is open or not, such that the next observations of the same door do not lead to an update of the belief-state.

B. Optimization objective

We note ψ a path-tree, and (u,v) consecutive nodes on the tree ψ. Nodes are associated with a robot configuration and a belief state. The motion cost between two nodes u and v is given by a cost function $C(u,v)$. In addition, we note $p(u|\psi, b_0)$ the probability to reach a node u. It depends on the initial belief-state b_0 and the observation model.

In the presence of uncertainty we minimize the expectation of the motion costs, such that the partially observable multi-modal optimization problem is defined as follows.

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

s.t.

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

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

$$b_v(s) = \frac{p(o|s)b_u(s)}{\sum_{s'} p(o|s')b_u(s')}, \forall s \in \mathcal{H}, \forall (u,o,v) \in \psi, \tag{1d}$$

where $\mathcal{L}(\psi)$ gives the leaf nodes of ψ, and $G(l)$ is the goal predicate, indicating whether a node fulfills the goal conditions. Eq. (1b) states that ψ shall be complete i.e. there is a leaf node satisfying the goal condition for each state s.

Eq. (1c) expresses that ψ shall be composed of feasible motions (e.g. collision free, feasible given the robot motion model), and $\mathcal{V}(u, v)$ is the predicate encoding the validity of the transition between u and v.

Finally, Eq. (1d) corresponds to the Bayesian updates of the belief state. The symbol o is the observation received when transitioning from u to v. The belief-state of the node v is noted b_v, and $p(o|s)$ is the observation model.

The problem formulation does not contain any term explicitly incentivizing the robot to explore its environment. The balance exploration / exploitation will emerge as a result of the minimization of the expected motion costs (see IV-D).

IV. Path-tree optimization (PTO)

To build a path-tree satisfying the specification of Equation 1, we proceed stepwise. First a transition system is grown in a sampling-based fashion until the existence of a solution is guaranteed. This corresponds to the first two steps schematized on Fig. 3. In a second step, the optimal path-tree is extracted using dynamic programming.

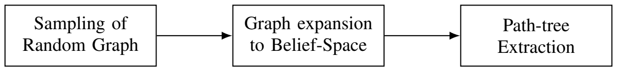


Fig. 3: PTO Algorithm overview. It contains three main steps corresponding to the algorithms described in the sections IV-B, IV-C and IV-D.

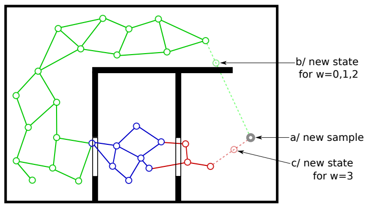
A. Interface between the algorithm and the application layer

The connection between the core of the algorithm and a given planning problem is achieved via 4 functions that the application layer provides:

- StateCheck, which takes a robot configuration x as input and returns the list of worlds in which the configuration is valid.
- TransitionCheck, which takes two robot configurations as inputs and returns the list of worlds in which the robot configurations are valid.
- GoalCheck, which takes a robot configuration as input and returns the list of worlds in which the robot configuration fulfills the goal conditions.
- Observe, which receives a robot configuration and a belief-state as inputs and returns the possible output belief-states.

StateCheck, TransitionCheck and GoalCheck are akin to the functions required by standard path-planning frameworks like OMPL [17], but they are more general: the return type is a list of worlds instead of a boolean.

The Observe function is directly linked to partial observability and is specific to this algorithm. It allows belief-state inference. Going back to the example of Fig. 1a, Observe returns an unchanged belief-state for all robot configurations that are outside of the door visibility zone, because no observation can be received, and therefore no belief-state inference is done. On the other hand, up to two possible belief-states are returned for robot configurations inside the door visibility zone. They correspond to the updated belief-states for the two possible observations.



4: Random Graph expansion: For a new sample *a)*, the new state is computed steering from the closest node having a reachability $\mathcal{R}$ compatible with the sampled world w, see *b)* and *c)*.

The next sections detail how the calls to those functions are orchestrated by the algorithm to build a path-tree.

B. Rapidly-exploring Random Graph

In this first step, a random-graph is grown in a sampling-based fashion. To avoid the curse of dimensionality, sampling is not performed in belief-space directly but in the robot configuration space. The random graph is an intermediate representation and will be expanded to belief-state in a second step as described in the next section.

The nodes of the random graph are associated with:

- a robot configuration x, which is randomly sampled.
- a list of worlds $\mathcal{R}$ in which the node is guaranteed to be reachable. It is obtained by composing the reachability of the node's parents with the results of the TransitionCheck.
- a list of worlds $\mathcal{F}$ in which the robot configuration is fulfilling the goal condition. This is obtained by querying the function GoalCheck.

The edges of the random graph are associated with:

- a list of worlds $\mathcal{W}$ in which the transition is valid. This is obtained by calling the function TransitionCheck.

The random-graph creation is described in Alg. 1. First, the state is sampled (line 4). Then the new state is steered from a neighbor node of the graph (line 7). Unlike RRT where the new state is steered from the nearest neighbor, here, the selected neighbor is the nearest neighbor having world reachabilities $\mathcal{R}$ containing a uniformly sampled world w (line 5), as illustrated on Fig. 4. This additional condition for the neighbor selection is to ensure that the random graph contains paths to the goal for each world.

If the new state is valid for at least one world (line 9), the goal conditions are checked (line 10) and a new node is added to the random-graph (line 11). Finally, the new node is connected to nodes within an adaptive radius r [18], and having world validities containing w (lines 13 to 16).

This procedure is repeated at least until the graph is complete, meaning that it will allow the successful extraction of a solution path-tree (see line 3). To implement the function IsComplete we assume in our examples that the transitions are *symmetrical*, i.e. if a motion exists from a node u to v, then a motion between v and u also exists. Under this assumption, the random-graph contains a solution path-tree as soon as the set of leaf nodes is complete, meaning that it covers every possible world, i.e. $\bigcup_{u \in \mathcal{L}(\mathcal{G})} \mathcal{F}_u \cap \mathcal{R}_u = \mathcal{H}$. Indeed, the fact that the set of leaf nodes is complete implies that for each possible world, a path from the root to a leaf exists. A naive solution is therefore to execute those paths in sequence, and potentially *backtrack* to the root node if a path doesn't reach the goal (hence the required assumption regarding the symmetrical transition). In practice, once the completeness condition is reached, a solution can be extracted which is more optimal than the aforementioned worst-case backtracking strategy. A minimum number of iterations is also specified, to eventually expand the random-graph beyond the completeness threshold to improve the quality of the path-trees.

Algorithm 1 Rapidly-exploring Random Graph

```
1:  function BUILDRRG($q_{start}$, $i_{min}$)
2:      $\mathcal{G}$.INIT($q_{start}$); $i \leftarrow 0$
3:      while $\neg$ ISCOMPLETE($\mathcal{G}$) or $i < i_{min}$ do
4:          $q_{rand} \leftarrow$ SAMPLESTATE()
5:          $w \leftarrow$ SAMPLEWORLD()
6:          $q_{near} \leftarrow$ NEAREST($q_{rand}, w$)
7:          $q_{new} \leftarrow$ STEER($q_{near}, q_{rand}$)
8:          /*get relevant info of the new state and add it to the graph*/
9:          if STATECHECK($q_{new}$) $\neq \emptyset$ then
10:             $\mathcal{F} \leftarrow$ GOALCHECK($q_{new}$)
11:             $\mathcal{G}$.ADDNODE($q_{new}, \mathcal{F}$)
12:             /*get relevant edges info and add them to the graph*/
13:             for $q_{near} \in$ NEARESTS($q_{new}, w, r$) do
14:                 $\mathcal{W} \leftarrow$ TRANSITIONCHECK($q_{near}, q_{new}$)
15:                 if $\mathcal{W} \neq \emptyset$ then
16:                     $\mathcal{G}$.ADDEDGE($q_{near}, q_{new}, \mathcal{W}$)
17:         $i \leftarrow i + 1$
```

Each node is kept connected to multiple neighbors because the best parent at this stage is ambiguous: it is only defined for a given belief-state. The optimal traversal will be determined on the belief-space graph (see IV-D).

C. Graph expansion to belief-space

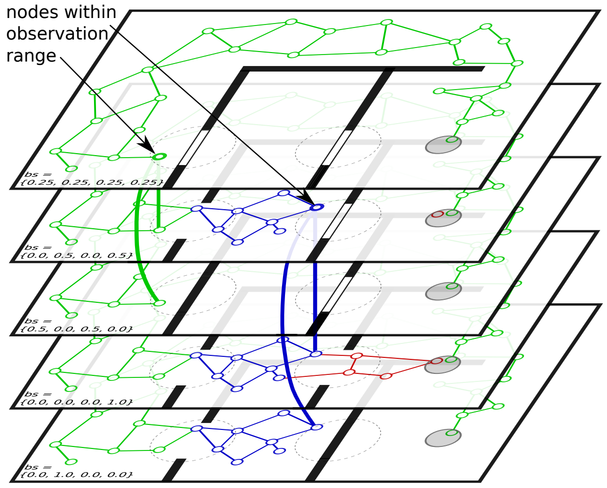
The random graph $\mathcal{G}$ is an intermediate representation that cannot be used directly to optimize path-trees. In this step, a transition system in belief-space, or belief-graph $\mathcal{B}$ is constructed out of $\mathcal{G}$ by querying the observation model. The nodes of the belief-graph are associated with:

- a robot-configuration x.
- a belief-state b.

The edges of the belief-graph are associated with:

- the observation o making the transition between the beliefs on the incoming and outcoming nodes.

The belief-graph can be understood as the random-graph replicated over several layers, where each layer is a different belief-state, as shown on Fig. 5. The transitions within one layer correspond to robot motions, whereas the transitions from one layer to another correspond to the integration of an observation leading to a belief-state update. The motion


nodes within
observation
range
bs = {0.25, 0.25, 0.25, 0.25}
bs = {0.0, 0.5, 0.0, 0.5}
bs = {0.5, 0.0, 0.5, 0.0}
bs = {0.0, 0.0, 0.0, 1.0}
bs = {0.0, 1.0, 0.0, 0.0}
Fig. 5: Random Graph expanded to belief-state: The belief-states are represented as *layers* of the belief-graph. The nodes within the observation range potentially lead to a belief-state update and therefore have edges transitioning to other belief-states (thick vertical edges).

transitions are, in general, not identical across belief-states. For example, on Fig. 5, the transitions crossing through the doors exist only in beliefs compatible with an open door.

The construction procedure is given by the Alg. 2. First, the edges of $\mathcal{G}$ are replicated to the belief-states compatible with the edge's valid worlds (line 3 to 7).

Second, the observation model is called via the function OBSERVE to identify where edges should be created between belief-states (lines 9 to 12). The belief-state-transition edges, due to observations, are represented by vertical lines going from one layer to another on Fig. 5.

Algorithm 2 Creation of the belief-graph

```
1:  function BUILDBELIEFGRAPH($b_{start}, \mathcal{G}$)
2:      /*connect nodes within the same belief*/
3:      for $e \in \mathcal{G}.edges$ do
4:          for $b \in$ BELIEFS($e.world\_validities$) do
5:              $u \leftarrow \mathcal{B}$.ADDNODE($b, e.from$)
6:              $v \leftarrow \mathcal{B}$.ADDNODE($b, e.to$)
7:              $\mathcal{B}$.ADDEDGE($u, v$)
8:      /*create transitions between beliefs due to observations*/
9:      for $u \in \mathcal{B}.nodes$ do
10:         $V \leftarrow$ OBSERVE($u$)
11:         for $v \in V$ do
12:             $\mathcal{B}$.ADDEDGE($u$, $v$)
```

The belief-graph $\mathcal{B}$ is significantly larger than the random graph $\mathcal{G}$ since there are multiple belief-states corresponding to the same robot configuration. However, its construction only calls the observation model (function OBSERVE). In contrast, the random-graph is smaller but its creation calls the expensive collision checks (STATECHECK and TRANSITIONCHECK) in addition to the nearest neighbor search.

D. Policy extraction

Now that $\mathcal{B}$ is built and contains at least one path-tree satisfying the goal conditions (Eq. 1b), the goal is to find the optimal path-tree by minimizing the expected costs (Eq. 1a).

The expected costs to goal are computed using dynamic programming by iteratively applying Bellman updates on each node of $\mathcal{B}$, as described by the Algorithm 3.

The algorithm is similar to the Dijkstra algorithm [19] in the way the nodes are prioritized using a priority queue (lines 2 to 13). For edges corresponding to a robot motion, the Bellman update is also the same as in Dijkstra (lines 15 and 16). However, for edges corresponding to an observation, the Bellman update of the parent node differs: it is the sum of the children's expected costs weighted by their respective branching probabilities (lines 17 to 19).

Algorithm 3 Computation of the expected costs to goal

```
1:  function ComputeExpectedCostToGoal($\mathcal{B}$)
2:      $C \leftarrow$ Array() // Expected costs to goal indexed by node id
3:      $Q \leftarrow$ PriorityQueue()
4:      /*Initialization*/
5:      for $n \in \mathcal{B}.nodes$ do
6:          if IsFinal($n$) then
7:              $C[n] \leftarrow 0.0$
8:              $Q$.Push($n, 0.0$)
9:          else
10:             $C[n] \leftarrow +\infty$
11:     /*The main loop*/
12:     while $\neg$ IsEmpty($Q$) do
13:         $v \leftarrow$ Pop($Q$)
14:         for $u \in$ Parents($v$) do
15:             /*Bellman update dependent on the edge type*/
16:             if IsActionEdge($u, v$) then
17:                 $c \leftarrow$ Cost($u, v$)$+C[v]$
18:             else if IsObservationEdge($u, v$) then
19:                 $\mathcal{W} \leftarrow$ ObservationChildren($u$)
20:                 $c \leftarrow \sum_{\nu \in \mathcal{W}} p(\nu|u) \times C[\nu]$
21:             if $c < C[u]$ then
22:                 $C[u] \leftarrow c$
23:                 $Q$.Push($u, c$)
```

Once the expected costs to goal are known, the optimal path-tree ψ^* can be built straightforwardly starting from the root and recursively appending the best next child, or next best children (on nodes with observation branching).

It is noteworthy, that the expected costs are computed for all nodes of $\mathcal{B}$, such that a path-tree can be extracted from each node (not only from the root node). In practice, this can be used for quick re-planning at execution time.

Optimizing the expected costs to goal naturally results in an exploration vs. exploitation trade-off, i.e. path-trees balance the need to move towards configurations providing informative observations vs. advancing towards the goal.

E. Path-tree refinement

Despite asymptotic optimality guarantee, a path-tree may contain unnatural or jerky motions due to the random nature of the random-graph creation. While it is possible to increase the number of iterations, this can become costly in terms of runtime. To tackle this, we add a refinement step, working piecewise: path-pieces between observation branchings are refined using the partial-shortcut method [20]. Observation nodes connecting path-pieces are not modified by the refinement procedure.

F. Completeness and optimality

We argue that PTO is probabilistically complete and asymptotically optimal, i.e. it will eventually solve Eq. (1) to the optimal solution. This is true under the assumption that our observation model is well behaved and does not introduce zero-measure configurations changing the belief state. Our reasoning is divided into two steps. First, since we use an asymptotically optimal sampling-method [21], [18] on every modality (Alg. 1), together with a uniform modality sampler, PTO will eventually converge to the optimal solution on each mode. Second, by inter-connecting edges between modalities, we guarantee that we account for possible mode-changes which can only happen in observation nodes (Alg. 2). The final belief-graph will therefore contain the optimal solution, which we can then extract using the Dijkstra-like algorithm in our policy extraction step (Alg. 3). This combination guarantees that the optimal solution will eventually be attained in the limit.

V. Experiments

PTO is implemented in the Rust programming language [22]. The application layer is implemented in C++ using MoveIt [23]. The source code is available for reference [1].

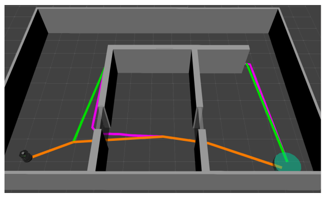
A. Mobile robot navigation

We consider two planning problems similar to the initial example introduced in Fig. 1. The robot is the Kobuki platform. The robot has to reach a predefined goal region, but at planning time, it doesn't know which doors are open. The observation model simulates that the sensor mounted on the Kobuki and the perception pipeline allow the robot to know if a door is open once the robot is close to a door (less than 1.5 m) and has a non-occluded line of sight to the door. We plan for the two-dimensional position of the robot.

The first problem, called problem-A (Fig. 6) has two uncertain doors. The second problem, called problem-B (Fig. 7) is more complex, the map is larger, there are four doors, and more obstacles.

In this example, we see the strong influence of the initial belief-state on the topology of the path-tree: Fig. 6b shows the path-tree obtained with an initial probability of only 50% that the doors are open. In that case, it is not worthy to attempt the way through the doors, the planned path takes the longer, but sure way to the goal. The path-tree boils down to a sequential path without observing branching points.

[1] https://github.com/cambyse/po-rrt


(a) Path-tree obtained with a high likelihood that doors are open (80%). The path-tree has 2 branching points corresponding to the observation of the 2 doors.

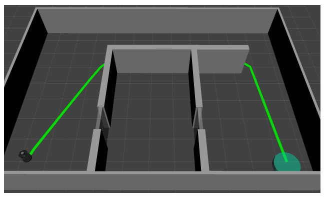

(b) Path-tree obtained with 50% of likelihood that doors are open: It is not advantageous to attempt the direct way through the doors.

Fig. 6: Problem-A: The robot has to reach the green region on the right.

Fig. 6a shows the path-tree obtained with a higher likelihood that each door is opened (80%). In that case, it is advantageous to attempt the direct path through the doors. The path-tree has two branching points corresponding to the observations of the two doors.

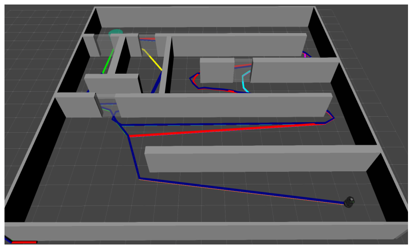


Fig. 7: Path-tree for Problem-B: The map has four doors leading to a more complex path-tree.

Fig. 7 shows the path-tree obtained for the problem-B, and Fig. 8 shows its execution in a subset of the possible worlds. The higher number of doors leads to a path-tree which is much more complex.

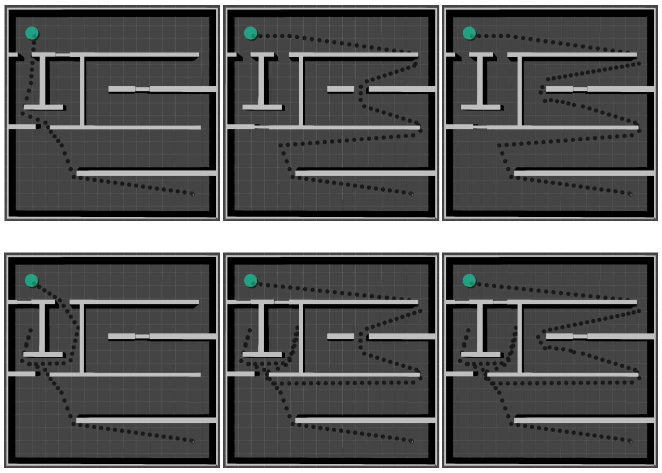


Fig. 8: Path-tree execution: The robot explores the doors on the left of the map. It potentially has to back-track and take the long route if it is blocked.

Table I, gives the expected costs to goal, and the planning times obtained when running PTO 20 times. Each problem is solved using 2 different strategies for the RRG sampling: In the initial variation, the RRG sampling is stopped as soon as completeness is guaranteed (see IV-B), and the second variations (A′ and B′) use a minimum number of 5000 iterations to obtain a more qualitative path-tree.

Generally, the algorithm is fast, and takes less than 1 second even for the most complex map. The longest is the graph expansion to belief-space (see IV-C). The reason is that the number of belief-states grows quickly when the number of doors increases, and the observation model is queried for each belief-space node.

	# of iter	random graph creation	belief-space expansion	policy extraction	partial shortcut	path cost	total planning time (ms)
A	433 (107)	0.72 (0.44)	0.98 (0.51)	0.54 (0.27)	0.49 (0.28)	18.6 (0.37)	2.78 (1.4)
A′	5000 (0)	30.9 (3.7)	29.1 (2.80)	16.7 (1.1)	0.31 (0.05)	18.17 (0.1)	77.1 (6.02)
B	3956 (626)	44.7 (14.3)	345 (102)	173 (60.8)	3.25 (1.07)	45.1 (1.74)	567 (174)
B′	5000 (0)	61.5 (6.49)	538 (32)	270 (22.9)	2.95 (0.38)	43.9 (1.25)	873 (47.1)

TABLE I: Planning time and expected cost: the mean value is given first. The standard-deviation is in parentheses. Times are in milliseconds.

Increasing the number of iterations results in smaller expected costs of the path-tree, at the expense of the overall runtime. In practice, the final refinement step (see IV-E) leads to path-trees that are already near-optimal even without a very high number of iterations, such that it is an efficient strategy to keep a small number of minimal iterations.

B. Robot arm object fetching

Our next scenario is a Panda arm robot mounted on a mobile base. We introduce two examples, where the task is to pick-up an object which is at an unknown location (see problem-A on Fig. 9 and problem-B on Fig. 10). The robot has prior knowledge of several potential locations, but the actual location is unknown at planning time.

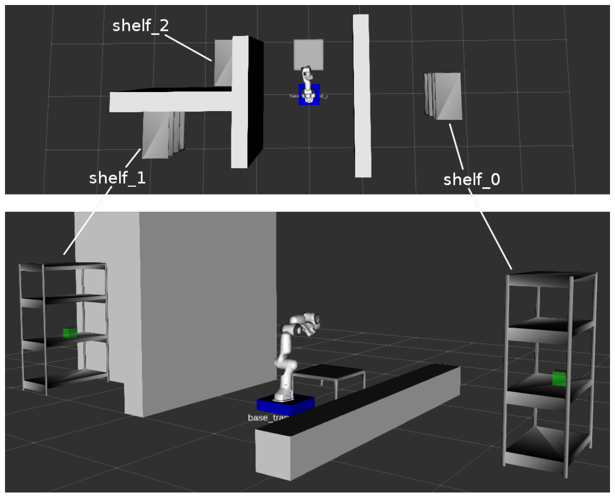


Fig. 9: Problem-A: The robot has to pick the green block. The block location is unknown, it may be on each one of the three shelves.

The observation model simulates that a sensor is placed on the robot gripper and that the perception pipeline detects the block when it is within the sensor field of view (60°), at a distance less than 2 meters from the sensor, and not occluded by other objects (e.g. walls).

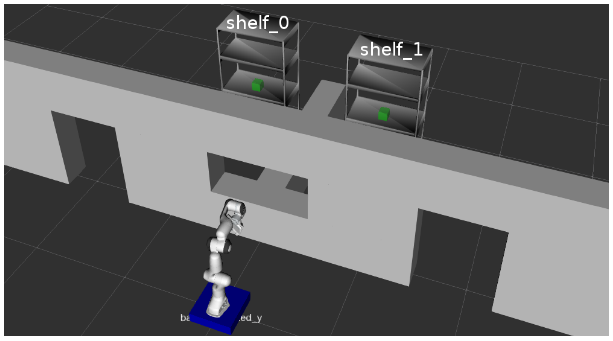


Fig. 10: Problem-B: The block might be on two different shelves. The opening between the two doors potentially allows the robot to observe the shelves without entering the rooms.

Planning is performed in joint space with 9 degrees of freedom (2 for the base, and 7 for the robot arm). Fig. 11 shows the trajectory of the robot base. There are two observation branching points corresponding to the observations of the two shelves.

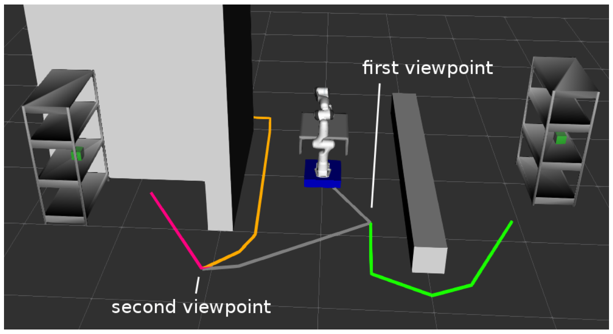


Fig. 11: Path-tree for Problem-A: The path tree first moves towards the shelf_0 to reach the first viewpoint. If the object is on the shelf_0, the green path is executed. Otherwise the robot moves towards the second viewpoint.

Fig. 12 shows the robot configuration at the branching point of the path-tree. The opening in the wall allows the robot to take a look at the shelf.

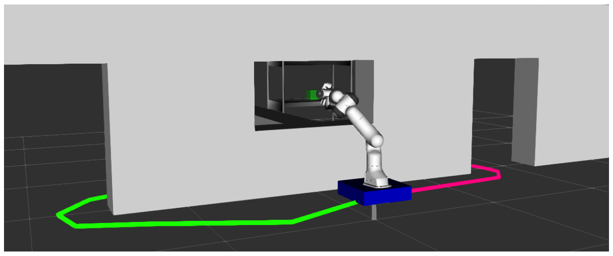


Fig. 12: Path-tree for Problem-B: A first common branch of the path-tree leads the robot to an observation point through the opening. Depending on the observation, the robot will execute the green or the magenta path.

We report in Table II on the average path-tree cost and planning time obtained over 100 planning queries. In contrast to the examples of the previous section V-A, the planning time is dominated by the random graph creation. This is consistent with the fact that the geometric collision checks are more computationally expensive with this robot. In addition, we note that the planning times have a large dispersion around the mean value (see the standard deviation on II), which is due to the higher geometric complexity and dimensionality of the planning problem. Overall, the planning time remains limited to a few seconds.

	# of iter	random graph creation	belief-space expansion	policy extraction	partial shortcut	path cost	total planning time (s)
A	10521 (16481)	5.86 (14.7)	0.31 (0.66)	0.30 (0.66)	0.43 (0.06)	3.27 (0.53)	6.75 (15.8)
B	4578 (3484)	1.28 (2.05)	0.03 (0.04)	0.01 (0.02)	0.26 (0.03)	3.72 (0.46)	1.59 (2.10)

TABLE II: Path-tree cost and planning time obtained over 100 planning queries: It indicates the mean value and the standard deviation in parentheses. Planning times are in seconds.

C. Comparison to baselines

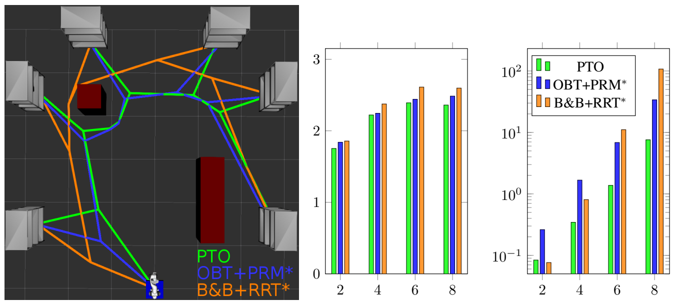
We compare PTO to two baselines on variations of the shelf domain (Fig. 13a), where we plan for the 2D base position only. The number of shelves is varied to compare the scalability.

In the first baseline, planning is hierarchical: it interleaves a symbolic Branch and Bound (B&B) search [24] defining high level plans, and a motion planning phase using RRT*. The B&B search explores the possible sequences for visiting the shelves. The nodes of the search correspond to shelves. For each node/shelf, 2 paths are planned using RRT*: a path to an observation point, and a path to fetch the object (to be executed if the object is actually detected). The path-tree is obtained by gathering the path pieces from the root node to a leaf of the search tree. The B&B search is performed in a depth-first fashion, which leads to a quick first solution. The best path-tree found so far is an upper bound of the cost and allows the pruning of a majority of the search tree.

The second baseline is adapted from the Orbital-Bellman-Tree (OBT) method [15][16]. The algorithm is adapted to belief-space planning by considering that different belief states correspond to different orbits, and by using Algorithm 3 to perform contingent planning and retrace a path-tree (instead of extracting a simple path). It grows different roadmaps for each visited belief state using PRM*. Observation points are explicitly sampled which transition between roadmaps.

We report on Figure 13 on the average expected cost of the path-trees, as well as on the planning times obtained over 100 planning queries. RRT* paths are planned with 2500 iterations. The random graph of PTO is created with a minimum number of 5000 iterations. OBT+PRM* is parameterized to reach an average of 5000 iterations per roadmap.

First, we observe that PTO consistently leads to the lowest path-tree costs. This can be understood compared to B&B+RRT*: the decomposition into piecewise motions leads to path pieces that are optimal w.r.t the sub-problem of reaching one shelf, but taken together, they don't form an optimal path-tree. In contrast, PTO searches for a globally optimal path-tree. This is visible on Fig. 13a: the B&B+RRT*



(a) PTO plans path-trees with lower costs than RRT* and PRM*.

(b) Path cost (m)

(c) Planning time (s) in logarithmic scale

Fig. 13: Comparison of PTO to baseline algorithms: 13b and 13c give the costs and planning times (log scale) for different number of shelves. PTO provides path-trees with lower costs and scales better.

path-tree greedily moves towards the next shelf to explore, whereas PTO takes a less direct path to the observation point, which overall leads to a shorter path-tree. PTO also leads to lower costs than OBT+PRM*. Even though OBT+PRM* is also asymptotically optimal (see [16]), we observed, that for a comparable final number of nodes, the belief-graph of PTO (grown as a RRG with periodic sampling of goal states) is more efficiently sampled in the regions of the optimal path-tree compared to the goal-agnostic PRM* graph.

Second, PTO scales better w.r.t. the number of shelves (see Fig. 13), it outperforms B&B+RRT* by one order of magnitude on the 8-shelves problem. The main reason is that PTO is more sample-efficient: the random graph creation which involves state and transition checks is constructed just once. On the other hand, the B&B+RRT* samples a new random tree for the planning of each path-piece. OBT+PRM* is more efficient but still requires, in general, different roadmaps for each belief-state to account for domains where the partially observable state impacts the state and transition validities (like for the problem of Fig. 1).

In this example, the three methods could be made faster by using, e.g. heuristics for the B&B tree search or by re-using samples across belief-states for the OBT+PRM*. We chose, however, to compare the general approaches, without optimizations tailored to the particular example.

VI. Conclusions

We proposed a new sampling-based path planning algorithm (PTO) for motion planning problems where discrete and critical aspects of the world are partially observable. The algorithm not only optimizes feasible motions but also plans observation points that allow the robot to gain knowledge about its environment to achieve its task. The resulting motions are path-trees in belief-space that react to observations. We showed that PTO is asymptotically optimal, and it compares advantageously to Task and Motion Planning (TAMP) approaches both in terms of optimality and runtime efficiency.

References

[1] C. Phiquepal and M. Toussaint, “Combined Task and Motion Planning under Partial Observability: An Optimization-Based Approach,” *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9000–9006, 2019.

[2] C. Phiquepal and M. Toussaint, “Control-Tree Optimization: an approach to MPC under discrete Partial Observability,” *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9666–9672, 2021.

[3] M. Hsiao, J. G. Mangelson, S. Suresh, C. Debrunner, and M. Kaess, “Aras: Ambiguity-aware robust active slam based on multi-hypothesis state and map estimations,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5037–5044, IEEE.

[4] T. Patten, W. Martens, and R. Fitch, “Monte carlo planning for active object classification,” *Autonomous Robots*, vol. 42, no. 2, pp. 391–421, 2018.

[5] S. M. LaValle *et al.*, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” 1998.

[6] 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*, vol. 12, no. 4, pp. 566–580, 1996.

[7] G. A. Hollinger and G. S. Sukhatme, “Sampling-based Robotic Information Gathering Algorithms,” *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1271–1287, 2014.

[8] D. Levine, B. Luders, and J. How, “Information-rich Path Planning with General Constraints using Rapidly-Exploring Random Trees,” in *AIAA Infotech@ Aerospace 2010*, p. 3360, 2010.

[9] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, “Graph-based Subterranean Exploration Path Planning using Aerial and Legged Robots,” *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[10] S. Prentice and N. Roy, “The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance,” in *Robotics research*, pp. 293–305, Springer, 2010.

[11] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, “Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements,” *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 268–304, 2014.

[12] K. Leahy, E. Cristofalo, C.-I. Vasile, A. Jones, E. Montijano, M. Schwager, and C. Belta, “Control in Belief Space with Temporal Logic Specifications using Vision-based Localization,” *The International Journal of Robotics Research*, vol. 38, no. 6, pp. 702–722, 2019.

[13] D. Zheng, J. Ridderhof, P. Tsiotras, and A.-a. Agha-mohammadi, “Belief Space Planning: A Covariance Steering Approach,” *arXiv preprint arXiv:2105.11092*, 2021.

[14] E. Páll, A. Sieverling, and O. Brock, “Contingent Contact-Based Motion Planning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6615–6621, IEEE, 2018.

[15] W. Vega-Brown and N. Roy, “Asymptotically optimal planning under piecewise-analytic constraints,” in *Algorithmic Foundations of Robotics XII*, pp. 528–543, Springer, 2020.

[16] R. Shome, D. Nakhimovich, and K. Bekris, *Pushing the Boundaries of Asymptotic Optimality in Integrated Task and Motion Planning*, pp. 467–484. 02 2021.

[17] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. `https://ompl.kavrakilab.org`.

[18] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, “Revisiting the Asymptotic Optimality of RRT,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2189–2195, IEEE, 2020.

[19] M. Sniedovich, “Dijkstra’s algorithm revisited: the dynamic programming connexion,” *Control and Cybernetics*, vol. 35, pp. 599–620, 2006.

[20] R. Geraerts and M. H. Overmars, “Creating High-quality Paths for Motion Planning,” *The international journal of robotics research*, vol. 26, no. 8, pp. 845–863, 2007.

[21] S. Karaman and E. Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning,” *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[22] N. D. Matsakis and F. S. Klock II, “The Rust Language,” in *ACM SIGAda Ada Letters*, vol. 34, pp. 103–104, ACM, 2014.

[23] D. Coleman, I. Sucan, S. Chitta, and N. Correll, “Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study,” *arXiv preprint arXiv:1404.3785*, 2014.

[24] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, “Branch-and-Bound Algorithms: A Survey of Recent Advances in Searching, Branching, and Pruning,” *Discrete Optimization*, vol. 19, pp. 79–102, 2016.