
Open-loop plans in multi-robot POMDPs

Chih-Han Yu

Department of Computer Science
Stanford University
chihan@stanford.edu

Jason Chuang

Scientific Computing & Comp. Math
Stanford University
jcchuang@stanford.edu

Brian Gerkey

Department of Computer Science
Stanford University
gerkey@robotics.stanford.edu

Geoffrey J. Gordon

School of Computer Science
Carnegie Mellon University
ggordon@cs.cmu.edu

Andrew Ng

Department of Computer Science
Stanford University
ang@cs.stanford.edu

Abstract

It is well known that it is much easier to find open-loop plans in partially-observable environments than it is to compute full policies. It is less well appreciated that open-loop plans can provide good (sometimes even optimal) performance in some real-world robotic planning problems. In this paper we identify several conditions under which open-loop plans provide good performance, and show that multi-agent tag (a recently-popular POMDP benchmark problem) satisfies these conditions. We take advantage of this result to compute plans for a team of three robots searching for an evader in an office environment.

1 Introduction

Robotic planning problems are characterized by *actuator uncertainty* (a given action may have unpredictable effects) and *sensor uncertainty* (the available sensor information is not enough to disambiguate the current state of the world). The standard way to model these types of uncertainty is with a partially-observable Markov decision process, or POMDP. In POMDPs, the robots maintain a *belief*, a posterior distribution over the possible states of the world, and select actions on the basis of their current belief. In other words, the planning problem for POMDPs is to compute an optimal *policy* which maps beliefs to actions in a way that maximizes expected rewards. Unfortunately, the planning problem in POMDPs is quite difficult: it is known to be PSPACE-complete [1], which means that the best known algorithms take time $2^{\text{poly}(n,h)}$ to solve a problem with n states and horizon h .

Part of the reason for the difficulty of planning in POMDPs is the fact that there

are a very large number of distinct policies: each policy is a map $\Delta_n \mapsto A$, where Δ_n is the $(n - 1)$ -dimensional probability simplex on n atoms and A is the set of available actions. So, instead of computing a full policy, we might look for a good *open-loop plan*: a sequence of actions $\langle a_1, a_2, \dots, a_h \rangle \in A^h$ which achieves high reward. Since the number of open-loop plans is much smaller than the number of policies, we might hope that it would be easier to find a good open-loop plan. Indeed, this intuition holds: the problem of finding open-loop plans in POMDPs (also known as the NOMDP, or non-observable MDP, planning problem) is merely NP-complete [2], and it is generally believed that $\text{NP} \subset \text{PSPACE}$.

The defining characteristic of NP-complete problems is that we can solve them with search algorithms. That is, an NP-complete problem can be solved by searching for a short (polynomial-length) *certificate* of solution; once we have the certificate, it is efficient (polynomial time) to check that the certificate solves the problem. For example, in satisfiability of boolean formulas (the prototypical NP-complete problem), the certificate is just the satisfying assignment. In open-loop planning, the decision problem is whether there exists a plan which achieves a certain level of expected reward, and a certificate is a plan which achieves the desired reward.

Typical algorithms which people have used to solve NP-complete robotic planning problems include A* search [3], satisfiability solvers like Walksat [4] and Chaff [5], and rapidly-exploring random trees (RRTs) [6]. In this paper, we use a version of RRTs to solve robotic planning problems efficiently.

Of course, this gain in efficiency comes at a price: not all POMDPs have good open-loop plans. In the rest of this paper, we will identify several conditions under which good open-loop plans will be available, and we will demonstrate how these conditions allow us to apply classical search algorithms to the problem of finding good strategies in the game of multi-robot tag.

2 Background

We use the typical representation of POMDPs: a set of states S , a set of actions A , a set of observations Z , and probability distributions $p_0(s)$, $p(s' | s, a)$, and $p(z | s)$. A NOMDP is just a POMDP where $|Z| = 1$. POMDP belief tracking is accomplished by the equation:

$$b_t(s) = \eta p(z_t | s_t) \sum_{s'} b_{t-1}(s') p(s_t | s', a_{t-1}) \quad (1)$$

RRTs [6] are a common method for planning in continuous deterministic problems. The RRT algorithm maintains a tree of paths rooted at the start state: each node is a state in the planning problem, and its children are possible successors under various actions. Each step of the RRT algorithm consists of selecting a node in the tree and adding a new child to it. To select a node, the RRT algorithm first picks a “target” point at random; the distribution of target points is a parameter of the algorithm. Next it selects the tree node closest to the target point; the distance metric is another parameter of the algorithm. Finally, it chooses an action to apply from the selected node; the method for action choice is a parameter of the algorithm, but usually is designed so that the child node tends to be closer to the target point than the parent node was.

3 Quality of open-loop plans

As mentioned in the introduction, open-loop plans are generally less powerful than full policies, but they can achieve good performance on some POMDPs. In this

section, we will describe four conditions under which open-loop plans are optimal or near-optimal.

No sensors The simplest condition under which open-loop plans are optimal, of course, is when the robots' sensors provide no information about the state of the problem. We will not say much about this condition except to note that it has been used to advantage for the problem of orienting mechanical parts for factory assembly [7]. In the cited paper, the authors describe a system in which a known part is placed in a tray in an initially unknown position and orientation; a sequence of tilts of the tray can then bring the part to a desired position and orientation.

Deterministic actions, known initial state If the robots' initial positions are known and the outcomes of their actions are deterministic, then we can predict their positions perfectly at all times. In this case, no matter how informative our sensors may appear to be, we can afford to ignore them since they tell us nothing new. This condition is a special case of the no-sensors condition, but it is one which often arises in robotics problems. For example, in some problems motion actions are nearly perfectly accurate at the scale on which the POMDP planner operates: the robot described in [8], if operating on relatively level terrain, can move for 100 meters while accumulating less than 1 meter of position error.

Hierarchical planning Even if our robots' actions accumulate error rapidly enough that we must take account of the resulting uncertainty, it is often possible to design a hierarchical planner which deals with uncertainty at a low level and thereby avoids the need to compute full policies at the highest level. For example, suppose that our task requires us to drive a robot along one of two narrow bridges from location A to location B, but the robot's actuators have enough error in them that all open-loop plans have a significant probability of falling off of the bridges. The high-level decision of which bridge to take is fundamentally different from the low-level decisions of what commands to send to the actuators to move the robot along the desired path.

In particular, we can define macro-actions which use a PD controller to servo the robot to a desired pose; so long as the robot starts out near the desired pose, the PD controller will achieve the desired result to high accuracy with nearly 100% probability. These macro-actions use the robot's sensors and take uncertainty into account, but because they remove (almost) all of the uncertainty from the low-level actions, the high-level planning problem reduces to finding an open-loop sequence of poses that the robot should pass through. This sort of hierarchical decomposition is very common in control theory: the low-level reactive problem is referred to as trajectory following, while the high-level open-loop problem is called trajectory generation.

Unlike the other conditions in this section, this sort of hierarchical approach may not produce an optimal plan. However, in many robotic systems, motion actions can be wrapped in this way with almost no loss in plan quality. Generally, the loss in plan quality depends on what type of and how many macro-actions actions are available: in the limit, if we provide one macro-action for each possible POMDP policy, then the resulting open-loop plans are guaranteed to be optimal (but of course are no easier to find than closed-loop plans were for the original problem).

Stopping-time sensor The final condition under which open-loop plans are optimal is if we have a sensor which tells us only when to stop acting. For example, in a problem where we want to insert a peg in a hole, we can design an open-loop sequence of compliant motions which is likely to result in success. To improve the

chances of success, this sequence may contain multiple “tries” in which we move the peg back toward its starting position and approach the hole again. If the peg enters the hole on a particular try, we will stop the robot from executing any further tries, since the act of moving the peg back towards the starting location will undo our success.

The reason this condition allows us to design open-loop plans is that, in choosing actions, we can always assume that we haven’t succeeded yet. Under the assumption of lack of success so far, our sensor history is completely determined, and therefore we can perfectly predict the results of our actions.

The stopping-time sensor can be combined with the other listed conditions, often to good effect. As we shall see below, the problem of multi-robot tag can be reduced to an open-loop planning problem by the combination of a hierarchical planner and the use of a stopping time sensor.

4 Multi-robot tag

In the problem of tag [9, 10, 11, 12, 13, 14] a robot must search through a known environment to find an evader. Once the evader is found, the pursuer must approach the evader and execute a **tag** action. The pursuer pays a cost of 1 for every step until the evader is tagged; it then gets a bonus of 100 points and the game ends.

The environment is specified by a two-dimensional map of free and occupied space (in our case, represented as an occupancy grid—see Figure 2). The robot has a line-of-sight sensor which can detect the evader if it is within range; occupied areas of the map block the robot’s view. In our case the sensor is omnidirectional but has limited range (about 2 meters). We assume that the sensor has some probability of missing the evader even if the evader is within range and line of sight, but no probability of false detection.

The evader is allowed to move while the pursuer is searching for him. So, for example, if we are in an environment with multiple rooms branching off of a common hallway, the evader may slip behind the pursuer while he is searching one room and move to a previously-searched room. To prevent this occurrence, the pursuer must make sure to maintain a view of the hallway at all times (except perhaps for brief periods that are short enough that the evader would be unable to cross to previously-searched areas).

In multi-agent tag, there are several pursuers chasing a single evader. The pursuers must act as a team to find the evader as quickly as possible. In our hallway example, one pursuer could guard the hallway while another searched through a room; in this arrangement, the first pursuer would keep the evader from reaching previously-searched areas, thereby allowing the second pursuer to move away from the doorway and search larger rooms than he could search without a partner.

The robot model for the tag problem differs from paper to paper. In the current paper we will assume that the pursuers are differential drive robots (i.e., they can move forward or backward at bounded velocity while turning at bounded angular velocity) and that the results of their motion commands are noisy. The average speed of motion is assumed to be $1m/s$. The pursuers’ sensors give noisy estimates of their current position by comparing their perceptions to the known map. This model is a good description of typical “trashcan” robots which carry laser rangefinders such as the SICK LMS (see Figure 1, left panel).

The evader model for tag also differs from paper to paper: options include evaders which move by Brownian motion, evaders which run away from nearby pursuers

using long-range sensors to detect pursuer locations, and evaders which have omniscient knowledge of the pursuer locations and planned paths and which move to maximize the time until capture. Previous work also differs on the relative speeds of the pursuers and evaders; in this paper we will allow a range of speeds for the evader and assume that the evader moves by Brownian motion.

5 Tag as an open-loop problem

At first glance, tag appears to be a fully-general POMDP: the pursuers' actions are noisy, their sensors return a wide range of noisy information including position estimates and evader detections, and the evader's strategy contains randomness. However, using the techniques described in Section 3, we can translate multi-robot tag to an open-loop planning problem.

The first transformation will be to eliminate the use of sensor information in motion planning. We will accomplish this using a hierarchical planner: the top-level planner passes waypoints to a lower-level planner, and the lower-level planner servos the robots to these waypoints using the domain's noisy motion actions and position information.

A side effect of the hierarchical planner is that we don't need to consider robot orientations at the top level: since diff-drive robots can turn in place, the robot orientations will affect the time it takes to reach a desired waypoint but not whether it is possible to reach that waypoint. More specifically, at every step we will allow each pursuer to choose among 8 waypoints, to the north, south, east, west, north-east, northwest, southeast, and southwest of its current position; this representation results in 8^3 actions for a 3-pursuer team.

In our implementation, the lower-level planner keeps track of the robot's pose using a particle filter; it then moves to a desired waypoint by servoing the robot orientation to point directly at the waypoint and driving at a fixed velocity until the robot gets close to the waypoint. (Usually, once we are close to a waypoint, we will switch to the next waypoint; if instead we want to stop at a given waypoint, the low-level controller will reduce its forward velocity in proportion to the remaining distance to the waypoint.)

Once we have eliminated the use of position information at the top level, the only remaining sensor information is whether we have seen the evader. Once we have seen the evader, it is relatively easy to design a policy which approaches and tags it. That means that we have a stopping-time sensor: we can plan our paths under the assumption that we have not yet found the evader, then switch to our tagging policy if our sensor ever tells us that we have a sighting.

6 RRTs and tag

Since we have reduced tag to an open-loop planning problem, we can find a solution by searching for a sequence of actions which achieves a high level of reward (that is, which has a high probability of catching the evader in a small number of steps). In our experiments we plan for a team of three robots; since each robot has 8 action choices on each time step, there are 8^{3h} possible plans of length h .

To search for plans, we use the RRT algorithm described in Section 2. Each state of the tag domain contains the (x, y) positions of the three pursuers as well as a belief about where the evader could be hiding; this belief is computed from the history of past pursuer positions according to Equation 1. The parameters which are not

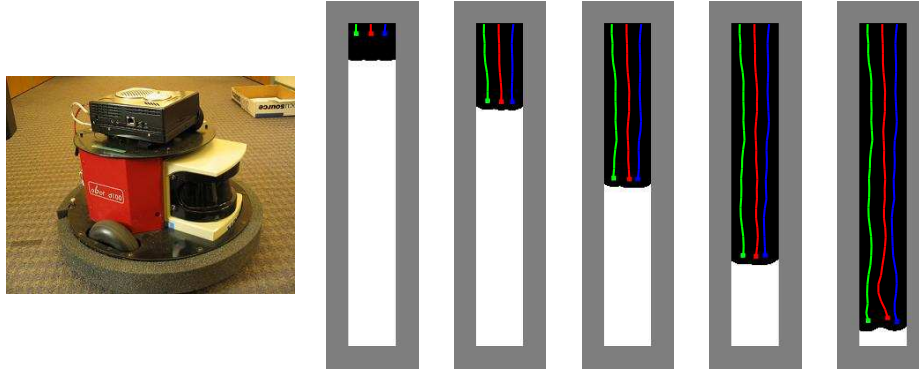


Figure 1: Left: the robot for which we designed our model of tag. Right: multi-robot coordination in tag. Three robots sweep down the hallway in parallel so that no evader can sneak behind them.

specified in Section 2 are how we generate target states, what distance metric we use to associate target states with states in our tree, and how we choose an action from the selected state in the tree.

To generate target states, we pick the three robot locations randomly from the map’s unoccupied space, and choose a belief in which the evader has already been caught. This distribution of target states tends to encourage goal-directed behavior by preferring expansion in directions which increase the probability of catching the evader, but also encourages exploration by drawing the robots towards unexplored areas. The distance metric we use between states is a combination of Euclidean distance for the robots with L_1 distance in beliefs: we match robots between the two states so as to minimize the total distance between matched robots. To this total distance, we add a constant multiple of the L_1 distance between beliefs. Finally, we expand our tree by choosing the action which moves each robot towards its position in the target state. This expansion rule is not the only possible one; for example, we could also pay attention to the instantaneous probability of catching the evader. But, in our experiments, this expansion rule worked acceptably.

7 Experiments

We tried our algorithm on two different types of experiments: in the first experiment we examine our robots’ coordinated behavior in a simple corridor. This experiment demonstrates performance on a fairly simple environment in which the optimal policy is apparent to a human observer. In the second experiment, we tested our planning algorithm in a more complicated office environment. In this environment, the optimal policy is not easy for humans to see, so we compared our algorithm with several heuristic planners to demonstrate its performance.

7.1 Simple Corridor

For this experiment, we placed three robots at one end of $5m$ wide and $40m$ long corridor, and assumed that the evader travels as fast as the controlled robots. As each robot is equipped with a $1.8m$ laser range sensor, one robot cannot clear the space individually; instead, the robots must cooperate to clear the corridor.

This problem is easy for humans to solve: the best solution is for all three robots to line up and form a wall of sensor coverage which stretches across the corridor, then sweep that wall from one end of the corridor to the other. If fewer than three robots line up, their sensor fields will not cover the width of the corridor, and so the evader will be able to slip behind them.

As shown in Figure 1, our algorithm discovers this plan. By doing so, it demonstrates that it has successfully reasoned about coordination among multiple robots: while there is enough overlap in the sensor fields to tolerate one or two incorrect actions over the length of the plan, it would not take very many mistakes to break the wall of sensor coverage, so the planner has rejected a huge number of uncoordinated action sequences in favor of the coordinated one. The planner expanded 1000 RRT nodes in 10 minutes on a 2.8GHz Pentium-IV.

7.2 Office Environment

In the second experiment, we applied our algorithm to a more complicated environment, shown in Figure 2. This environment measures $55m \times 40m$. Since we do not know the optimal policy for this environment, we compared our planner with two heuristics from the literature. These heuristics are the QMDP and Densest policies described by Roy and Gordon [10]. For these heuristics, we tracked a belief about where the evader could be hiding. This belief satisfies $b(s) \geq 0$ and $\sum_s b(s) = 1$.

In the QMDP heuristic, we solved the tag problem as if it were fully observable (that is, as if we could see the evader at all times and merely had to approach it and tag it). This resulted in an action-value or Q function which assigns an expected future reward to each state-action pair. We then chose the action a which maximized $\sum_s b(s)Q(s, a)$; this is the action which would be best if we knew that we were going to take exactly one action before finding out where the evader is.

In the Densest heuristic, we used $b(s)$ to compute the three spots that the robots would want to teleport to in order to have the highest probability of seeing the evader immediately. We assigned each spot to one robot in such a way as to minimize the total distance from the robots to their assigned spots, and had each robot take one step in the direction of its assigned region.

The performance of all three algorithms (our RRT and the two heuristics) depends on how quickly the evader moves as compared to the pursuers. We experimented with 9 different evader speeds, ranging from stationary to 80% of the speed of the pursuers. The right panel of Figure 2 shows the results, while the left panel shows a representative trajectory planned by our algorithm.

In the graph in Figure 2, each line shows the probability of catching the evader by the time the pursuing robots have traveled a total distance of $300m$ between them. The figure shows that, when the evader has a relatively slow moving speed (less than half of the speed of the pursuers), the planned paths generated by RRT have more than 90% probability of catching the evader. This probability gets lower as the evader's speed increases, but the RRT always performs at least as well as the two heuristics. To test the stability of planned paths generated by RRT, we selected 20 fixed sets of starting points and ran RRT to generate 20 paths for each set of starting points. The standard deviation of the success probability was only 2.3% of the mean success probability. A typical run required somewhat less than an hour to expand 5000 nodes on a 2.8GHz Pentium-IV.

In addition to the two heuristic approaches described above, we also attempted to apply Roy and Gordon's [10] approach for approximating POMDPs using E-PCA belief compression. To do so, we randomly took samples of plausible robot positions

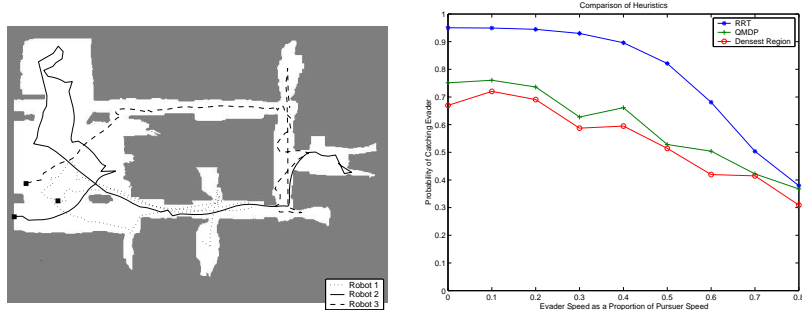


Figure 2: Left: the sample path generated by our RRT in tag. Right: comparison of different heuristics in tag.

and beliefs from paths generated by our RRTs. We compressed the beliefs using an E-PCA with 7 basis functions (not counting a single constant basis function which was included to ensure normalization of the predicted belief distributions). Running the computation on the Stanford distributed computer network, we were unable to process enough samples to achieve good planning performance: with 67,673 samples taken from approximately 8,000 different RRT paths, the algorithm took about 3,000 CPU-hours to complete, but still did not yield a useful value function.

References

- [1] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.
- [2] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, 2000.
- [3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd ed)*. Prentice Hall, 2003.
- [4] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–531, 1996.
- [5] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, pages 530–535, 2001.
- [6] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):278–400, 2001.
- [7] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Transactions on Robotics and Automation*, 4(4):369–379, 1988.
- [8] L. Ojeda, G. Reina, and J. Borenstein. Experimental results from FLEXnav: An expert rule-based dead-reckoning system for Mars rovers. In *IEEE Aerospace Conference*, 2004.
- [9] Matt Rosencrantz, Geoff Gordon, and Sebastian Thrun. Locating moving entities in dynamic indoor environments with teams of mobile robots. In *AAMAS*, 2003.
- [10] Nick Roy, Geoff Gordon, and Sebastian Thrun. Planning under uncertainty for reliable health care robotics. In *FSR*, 2003.
- [11] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *IJCAI*, 2003.
- [12] Matthijs T. J. Spaan and Nikos Vlassis. A point-based pomdp algorithm for robot planning. In *ICRA*, pages 2399–2404, 2004.
- [13] Brian P. Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. In *AAAI*, 2004.
- [14] S. M. LaValle, D. Lin, L. J. Guibas, J. C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *IEEE International Conference on Robotics and Automation*, 1997.