ASEN 6519 Final Project REPORT

Himanshu Gupta, Jackson Wagner

TOTAL POINTS

(not graded)

QUESTION 1

1 Final Project Report

- 0 pts Correct

Tree Search or Deep RL for Solving Belief MDPs?

Himanshu Gupta
University of Colorado Boulder
Boulder, USA
himanshu.gupta@colorado.edu

Jackson Wagner
University of Colorado Boulder
Boulder, USA
jackson.wagner@colorado.edu

Zachary Sunberg
University of Colorado Boulder
Boulder, USA
zachary.sunberg@colorado.edu

I. Introduction and Related Work

As a society, our current vision is to have robots assist us in our day-to-day tasks at home, at work, or while just walking on the road. Unfortunately, a lot of work needs to be done for us to get there. One major reason why we are not there yet is the presence of various kinds of unstructured uncertainties. These uncertainties can come from having sensors that don't always give perfect information, robot motion that is not as accurate as it was expected to be, or from just being in an unstructured environment that changes quickly. The common source of problem in all of them is the inability of the agent to see the true state of the environment, i.e., partial observability. To deal with partial observability and other uncertainties in a sequential decision-making problem, we use the Partially Observable Markov decision process or POMDP framework. Once a problem is formulated as a POMDP, there are lots of ways to figure out the best solution for it. Since the agent doesn't get to see the true state of the environment, it tries to maintain a distribution over the possible states based on the past actions that it took and observations that it received from the environment. The robot then bases its next action on this current distribution over states.

A lot of previous work has focused on solving small and discrete POMDPs [1, 2, 3]. They solve it approximately by making simplifying assumptions. Over the last two decades, researchers have focused their efforts on large and continuous POMDPs which are more difficult to solve. Most of these methods have focused on online planning [4, 5, 6, 7, 8]. The overall system works as follows. The agent maintains a belief over the unobservable state space which is a measure of how likely a particular state is the true state of the environment at that time. Starting from the current belief, the agent builds a tree which helps it reason over the possible things that can happen in the future and choose the best action within a fixed time budget. The agent executes that action, receives a new observation, and updates its belief. This repeats until the termination criterion is met. Although the concept behind building the belief tree to find the best action seems straightforward, its computational complexity increases exponentially with the number of states, actions, observations, and the planning horizon. For large or continuous problems, this is often resolved by using sparse sampling techniques [9, 10].

Partially Observable Monte Carlo planning or POMCP [6] is a really common online planning algorithm. POMCP is good at handling problems with large or continuous state space, but small and discrete action and observation space. It does so by sampling states from the belief and using an unweighted particle filter to represent different belief nodes during the tree search. Another approach called ARDESPOT [7] works well for problems with large or continuous state space and large observation space. It does so by picking a fixed set of likely possibilities and finding the best policy for this reduced set. ARDESPOT has done well in complex tasks like autonomous driving cars [11]. However, it fails to work for problems with continuous observation space and large or continuous action space.

Dealing with continuous spaces presents two big challenges when it comes to searching through decision trees. Firstly, in continuous spaces, the chance of getting the same exact number more than once is basically zero. This means that when we start planning, the tree gets super wide right away. This makes the tree search very shallow and not at all helpful, especially when the problem has sparse rewards. To address this, methods like POMCPOW and PFT-DPW [8] have used the double progressive widening (DPW) technique. Secondly, due to the large or continuous observation space, current methods (particle filter) that are generally used to represent a belief node in a belief tree often collapse down to just one single possibility. This makes them overly confident and doesn't encourage seeking more information. To address this, POMCPOW and PFT-DPW switched to a weighted particle filter representation of every belief node in the tree search. To handle continuous action space problems, VOMCPOW [12] used the Voronoi Progressive Widening (VPW) technique where it samples a new action from the action Voronoi that seems to do well. VPW helps search through options effectively, balancing between local and global action searching. Even though VOMCPOW consistently performed better than other state-of-the-art solvers, the continuous space POMDPs are far from solved.

Another approach that has been gathering a lot of attention for solving these complex POMDPs recently is to use deep reinforcement learning (DRL) techniques. Early DRL approaches to POMDPs involved frame stacking or LSTM layers [13]. Frame stacking is an approach to estimating the POMDP through a k-Markov state, which is highly successful in applications such as the Atari benchmark suite where velocities are not directly observed. To improve on the less principled

k-Markov and LSTM approaches, generative approaches [14, 15] attempt to learn the observation model and generate a latent state or belief. While recent approaches combining tree search techniques guided by learned policy and value functions [16] also show promise, they are somewhat hindered with regard to tree searches over continuous actions.

With this work, we are essentially trying to answer the question if one should be using tree search techniques or deep reinforcement learning techniques to solve complex POMDP problems. If the deep reinforcement learning techniques can do as well as the existing POMDP solution techniques when the problem is formulated as a belief MDP for simpler problems, they likely can be used to solve the more complex large or continuous action space POMDP problems, something tree search techniques are not good at. Also, this can potentially open up a new way to solve information-gathering problems which are often formulated using the ρPOMDP [17] framework. It differs from the traditional POMDP framework in the sense that the reward is no longer state-dependent but is instead belief-dependent which causes a majority of the existing solution techniques to not work right away.

II. Background

This section reviews mathematical formulations for sequential decision problems, some existing POMDP solution techniques, particle filters and a famous deep reinforcement learning technique.

A. MDPs

The Markov Decision Process (MDP) is a mathematical framework for representing a broad class of sequential decision-making problems. An MDP is defined by a tuple (S, A, T, R, γ), where S is the state space, A is the action space, T is the transition model, R is the reward model, and γ is the discount factor. When the system is in state $s \in S$ and takes an action $a \in A$, it reaches state $s' \in S$ with probability $T(s, a, s')$. The reward model R is specified by a function $R(s, a, s')$ which specifies the immediate reward of transitioning from state s via action a to state s'.

A policy for a MDP is a function π that specifies the action $a = \pi(s)$ at any given state in the state space S. The goal is to find a policy that maximizes the expected cumulative future reward, starting from the current state s. The expected future cumulative reward from a state s is often represented using the value function $V(s)$ as:

$$V_\pi(s) = \mathrm{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s\right) \tag{1}$$

B. POMDPs (Belief MPDs)

A POMDP is a generalization of an MDP in which the agent cannot directly observe the underlying state. Instead, it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP.

A POMDP is defined by a tuple $(S, A, Z, T, O, R, \gamma)$, where S is the state space, A is the action space, Z is the observation space, T is the transition model, O is the observation model, R is the reward model, and γ is the discount factor. When the system is in state $s \in S$ and takes an action $a \in A$, it reaches state $s' \in S$ with probability $T(s, a, s')$ and gets an observation $z \in Z$ with probability $O(s', a, z)$. The reward model R is specified by a function $R(s, a, s')$ which specifies the immediate reward of transitioning from state s via action a to state s'.

One method for handling the lack of direct state observability is to maintain a belief over all the possible states. Let b_{t-1} be the belief at time $t-1$. If the system takes an action a_t and gets an observation z_t at the next time step t, then using Bayes' rule, we get the new belief b_t as:

$$b_t(s') = \eta O(s', a_t, z_t) \sum_{s \in S} T(s, a_t, s') b_{t-1}(s) \tag{2}$$

where η is a normalization constant.

A policy for a POMDP is a function π that specifies the action $a = \pi(b)$ at any given belief over the state space b. The expected cumulative future reward starting from a belief b and following the policy π is represented using the value function $V_\pi(b)$ as:

$$V_\pi(b) = \mathrm{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t)) \mid b_0 = b\right) \tag{3}$$

The goal of any solution technique is to find the policy that maximizes the value function for the current belief b.

A POMDP can also be thought of as a special case of a belief MDP where the reward is generally a linear function of the states. -dependent. A BeliefMDP is defined by a tuple (B, A, τ, R, γ), where B is the belief space, A is the action space, τ is the transition model, R is the reward model, and γ is the discount factor. When the system is in belief state $b \in B$ and takes an action $a \in A$, it reaches belief state $b' \in S$ with probability $\tau(b, a, b')$. The reward model R is specified by a function $R(b, a, b')$ which specifies the immediate reward of transitioning from belief state b via action a to belief state b'.

C. Bootstrap Particle Filter

Particle filtering is a commonly used method used for approximating the probability density function in the presence of nonlinear system dynamics and/or non-gaussian noises. It does so by maintaining a set of weighted particles to give an intuition of where the probability mass lies. It can also be used for linear gaussian systems, but other methods are better suited in those cases. There are three major steps for maintaining a belief distribution using the bootstrap particle filter [18] where X_t denotes the set of particles and W_t denotes their weights at time t.

Step 1 - Prediction Step: Move all the particles according to the system's dynamics.

$X_{t+1}[i] = f(X_t[i]) + \epsilon_t[i]$ where $X[i]$ denotes the i^{th} particle, t denotes the time step, f denotes the dynamics function and ϵ denotes the transition noise.

Step 2 - Reweighting Step: Reweight all the particles based on the received observation z_{t+1} from the environment.

$W_{t+1}[i] = W_t[i] * p(z_{t+1}/X_t[i])$ where $W[i]$ denotes the weight of the i^{th} particle, t denotes the time step and $X[i]$ denotes the i^{th} particle.

Step 3 - Resampling Step: Normalize all the weights so that all the weights sum to 1 and draw particles with replacement from the modified set of particles X_{t+1} with probabilities proportional to their weights in W_{t+1}.

D. Moment Generating Functions

The particles in a particle filter make up an empirical distribution, and there are a variety of methods used to process these particles into a state representation for learning. If one were to simply concatenate the particle state, an issue that may arise during learning is that this representation is not permutation invariant which leads to higher variance updates and subsequently longer training times or decreased performance. One approach [15] to resolve the issue of particle permutation is to encode the belief using the moment generating function (MGF).

The MGF of a random vector $x \in \mathbb{R}^n$ evaluated at point $v \in \mathbb{R}^n$ is given by

$$M_x(v) = \mathbb{E}_x \left[e^{v^T x} \right], \tag{4}$$

and serves as an alternative specification of the random vector's probability distribution. By sampling x, assuming samples are independent and identically distributed, the unbiased estimate

$$\tilde{M}_x(v) = \frac{1}{N} \sum_{i=1}^{N} e^{v^T x_i} \tag{5}$$

may be evaluated at various $\{v_j\}$ to provide an encoding of the belief.

E. QMDP

Since obtaining exact solutions to POMDPs are non-trivial, they are often solved approximately. One such technique is the QMDP [19] value method. The QMDP method approximates the Q value function at a given belief b and for action a as follows: $Q_a(b) = \sum_s b(s) Q_{MDP}(s, a)$ where $Q_{MDP}(s, a)$ is the Q value function for the underlying MDP. This estimate essentially assumes that the true state is visible after taking an action and then the POMDP converts into an MDP which can be solved optimally. It chooses the action that maximizes the $Q_a(b)$ value.

QMDP policies have generally shown to work well in many domains, but they fail to work in problems where the agent actively needs to take information-gathering actions to reduce uncertainty. Due to its assumption of no uncertainty after one step, it fails to identify the benefits of information-gathering actions.

F. ARDESPOT

Approximate offline POMDP solvers like QMDP [19] and SARSOP [3] work well for problems with discrete state, action, and observation space POMDPs. However, the majority of the real-life robotics problems that we care about are continuous. Since sampling-based tree search methods [9, 10] have shown to work well for continuous MDPs, it is fair to assume that they will work well for continuous POMDPs as well. ARDESPOT [7] is one such algorithm that shows the effectiveness of methods that combine sparse sampling with tree search techniques. It can handle continuous state space and large observation spaces. DESPOT prevents the tree search from exploding in size by only expanding it under a fixed number of sampled scenarios. It evaluates all possible policies under this fixed set of scenarios and finds the best one. While building and expanding the tree, the algorithm relies on computing a lower bound and an upper bound on the value function at every belief node in the tree. The lower bound is generally obtained by executing a sub-optimal policy from that belief node, while the upper bound is generally obtained by calculating an over-optimistic estimate of the value function by designing a problem-specific function.

G. PPO

Proximal Policy Optimization (PPO) [20] is an on-policy actor-critic deep reinforcement learning algorithm. PPO is a policy gradient-based algorithm that constrains the policy updates using the clipped surrogate objective

$$L(\theta) = \mathbb{E}_t \left[\min(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t) \right] \tag{6}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and ϵ is a hyperparameter.

The advantage A and target returns used for updating the critic are estimated using Generalized Advantage Estimation (GAE) [21] by taking a discounted sum of TD-errors,

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_l \tag{7}$$

where λ is a hyperparameter, with $\lambda = 1$ corresponding to the Monte-Carlo return and $\lambda = 0$ corresponding to the 1-step TD error.

H. H-PPO

Deep reinforcement learning is heavily researched in the context of discrete and continuous action spaces, but somewhat under-developed in the area of hybrid, parameterized, and hierarchical action spaces. Hybrid Proximal Policy Optimization (H-PPO) [22] attempts to address this problem by using multiple actor networks, with each actor network trained on a separate PPO loss (6).

III. Problem Formulation

For empirical evaluation of our proposed approaches, we have used the LaserTag problem and its extensions.

The LaserTag problem is a POMDP for which the goal of the agent is to find and tag a moving target in an obstacle-rich environment, as shown in Figure 1. To keep the problem

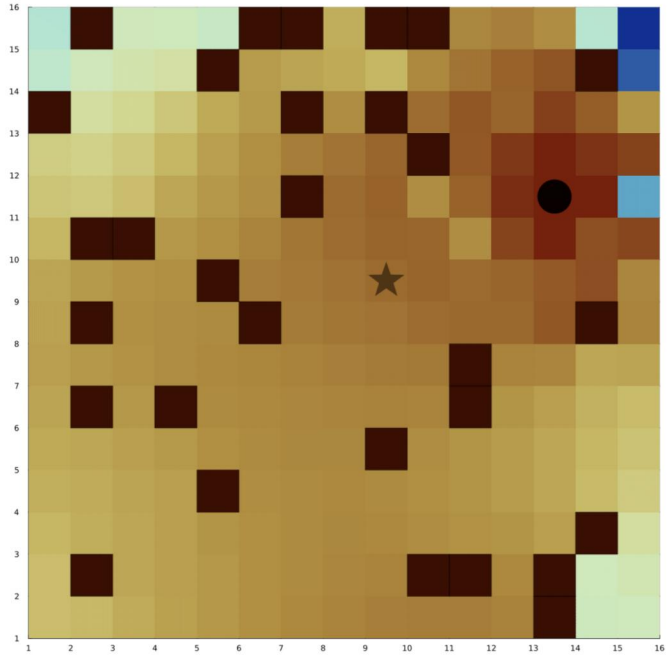

Fig. 1. A 15x15 LaserTag domain where the robot (denoted using a circle) is looking for the target (denoted using a star) in an obstacle rich environment. The black grid cells denote obstacles and the color of the grid cell denote the probability of the target being in that grid, with blue being high and red being low.

simple and tractable, it is assumed that the target moves in a grid world. At every time step the robot can take a ":measure" action which shoots a laser in all four directions (up, down, left, and right) and returns the correct distance of the robot from the boundary wall, target or obstacle (whichever is encountered first) in that direction. In case the robot decides to not take a ":measure" action, but rather move in a certain direction, it gets a noisy observation from the environment. With a high probability of 0.9, it fails to get a meaningful observation (all laser readings are zeros) and with a low probability of 0.1, it gets the same accurate observation that the ":measure" action would have returned. Also, the robot always gets an accurate observation which tells it if the robot and the target are in the same grid cell. The target has a very naive policy for moving in the environment. If the robot is more than two units away from the target, it moves randomly. If the robot is two or less than two units away, then it moves in the direction opposite of the robot. We have chosen a very high discounting factor ($\gamma = 0.997$) for all our variants. In this project, we have evaluated an offline POMDP solver, an online POMDP solver, and a deep reinforcement learning technique on four different variants of the LaserTag problem.

A. Variant 1: Discrete LaserTag with Exact Belief Updates

This variant is called the discrete LaserTag problem with exact belief updates. It is called discrete because the robot's position and action space are discrete. The robot's position is a tuple of integer values (R_x^d, R_y^d) denoting the grid cell it is in. The robot has nine possible actions- (:up, :down, :left, :right, :measure, :left-up, :left-down, :right-up, :right-down). The :measure action has a cost of -2 but gives accurate observations and helps the robot localize the target. When the robot decides to move in any of the four directions, it incurs a cost of -1 and receives noisy observations from the environment as mentioned earlier. If the robot reaches the same grid cell as the target, it gets a positive reward of +100 and the episode terminates. Since the system's transition probability and the robot's observation probability are well-defined for this problem, it is easy to compute the exact belief after every time step. Let $b_{t-1}(s)$ represent the belief of the target being in state s at time $t-1$. If at time step t, the robot takes the action a_t and gets an observation z_t, then using Bayes' rule, the new belief b_t can be computed using equation 2

The robot position and the exact belief over the state space are used by the solution approach to find the robot's best action at any given time step.

B. Variant 2: Discrete LaserTag with Particle Filter Belief Updates

This variant is called the discrete LaserTag problem with particle filter belief updates. The robot's state space and action space, the reward function, and the observation function are the same as Variant 1. However, the robot maintains a particle filter belief over the target position in the environment instead of maintaining a belief for the target being in every grid cell using equation 2. The reasoning behind designing this variant is as follows. In the majority of real-life robotics problems, it is often not possible to specify well-defined transition and observation probability functions. For such problems, computing exact beliefs is thus not feasible. In such cases, one possible way to represent the belief is to use the Kalman filter [23]. However, Kalman filters often diverge from the true belief for problems with nonlinear dynamics or non-gaussian noises or when the belief is multimodal. For such cases, keeping track of belief using a particle filter is generally better. We designed variant 2 since we wanted to compare the performance of different aforementioned techniques when the belief is approximated using the bootstrap particle filter.

C. Variant 3: Continuous LaserTag with Exact Belief Updates

This variant is called the continuous LaserTag problem with exact belief updates. It is called continuous because the robot's position and action space are now continuous. The target is still assumed to move in discrete grid cells in the environment. The robot's position is a tuple of float values (R_x^c, R_y^c) where R_x^c and R_y^c are continuous values that lie inside the environment. The robot has a hybrid action space. It either chooses to take the ":measure" action or it chooses to move (δ_x, δ_y) in the x and y directions respectively. Both (δ_x and δ_y) are bound to lie in the range of -1 to 1. The reward and observation function are the same as for variants 1 and 2. Since we have assumed no noise in robot dynamics and since the target still moves in a grid world, we once again have well-defined system transition probability and the robot observation probability. While generating the observations for

the robot, we observe the grid cell in which the robot lies and generate the observations assuming that discrete position of the robot. This gives us the same observation as is the case for variants 1 and 2. Once again, we can compute the exact belief after every time step. using equation 2.

D. Variant 4: Continuous LaserTag with Particle Filter Belief Updates

This case is called the continuous LaserTag problem with particle filter belief updates. The robot's state space, action space, observation space, and reward function are the same as Variant 3. Also, just like variant 2, the robot maintains a particle filter belief over the target position in the environment instead of maintaining a belief for the target being in every grid cell using equation 2.

As one can observe, the problem gets progressively more difficult and resembles real-life complex robotics problems more and more as we go from variants 1 to 4.

IV. Solution Approach

For this paper, we have compared three different solution techniques for solving the different variants of our LaserTag problem. We have used 1) QMDP, an offline approximate POMDP solver; 2) ARDESPOT, an offline approximate POMDP solver; 3) PPO, a deep reinforcement learning technique. Further details for each technique on all the different variants are explained in their respective sections below.

A. QMDP

QMDP is an offline approximate POMDP solver. For a discrete problem, like in variants 1 and 2, it is run offline to generate alpha vectors α_a corresponding to each action a in the action space. During run time, for any given belief b_t, we find the action a whose alpha vector maximizes the dot product $\alpha_a * b_t$. The robot executes this action, gets an observation from the environment, and updates its belief over the target. This cycle is repeated until the robot has successfully tracked the target.

Unfortunately, the existing QMDP solver in Julia is not capable of solving continuous problems in variants 3 and 4 because the QMDP method relies on solving the underlying MDP which is assumed to be discrete. Ideally, the QMDP solver can be extended to incorporate different techniques to solve the underlying continuous state and/or action space MDP, but we did not focus on that for this project. As a result, we don't have any results on variants 3 and 4 for the QMDP solver.

B. ARDESPOT

ARDESPOT is a state-of-the-art online POMDP solver. To run ARDESPOT for variants 1 and 3, we sample scenarios from the exact belief representation. Every scenario consists of the known robot position, the target position sampled from the exact belief, and a sequence of random number generators. For variants 2 and 4, the scenarios have the same structure, but the target position is now sampled from the particle filter belief. One major limitation of ARDESPOT and other online POMDP solvers is that they don't work well with continuous or large action space POMDPs. To ensure good performance by ARDESPOT in the continuous action domain of variants 3 and 4, we have discretized the action space to a finite number of actions. ARDESPOT plans over the same nine possible actions as is the case for the discrete problem in variants 1 and 2 - (:up, :down, :left, :right, :left-up, :left-down, :right-up, :right-down, :measure).

Another critical feature that often affects the performance of ARDESPOT is the effectiveness of the roll-out policy [11]. To guide the tree search well, we use the QMDP policy as the roll-out policy in the discrete problem of variants 1 and 2 and. In the continuous problem of variants 3 and 4, we use the roll-out policy of moving in a straight line to the grid cell with the highest probability. It is often not trivial to design effective roll-out policies. The upper bound is calculated by finding the minimum Euclidean distance d_m between the robot and the sampled target positions and returning the reward obtained by traveling this distance in a straight line path to the target while assuming that the target doesn't move and that there are no static obstacles in the environment.

The overall pipeline works as follows. At time step t, ARDESPOT samples scenarios from the belief b_t. It uses those sampled scenarios to build a tree and finds the best action a_t within a short period of 0.5 seconds. The robot executes action a_t and receives an observation o_t from the environment. The belief at next time step $b_{t+1} = \tau(b, a, o)$ is obtained either using exact updates or particle filter updates, depending on the variant. The cycle repeats until the robot has successfully tracked the target.

C. Deep Reinforcement Learning

In order to solve the POMDP using deep reinforcement learning, we utilize the belief MDP formulation. PPO is used to solve the belief MDP, where the encoded belief state is the concatenation of the processed robot position and target belief. In all cases, the robot position is scaled by the size of the environment such that the state falls in $\mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$, and then concatenated with the belief embedding.

In the case of the exact belief updates, the belief probabilities are scaled by $f(x) = \max(0, 1 + \log(x)/3)$ such that any $p(s_t|h_t) \leq 10^{-3}$ is equivalent to a belief that the target is not in that state. This transform was chosen such that each entry in the encoded state would fall in the range $[0, 1]$, with uniform distribution of belief taking a value near 1/3.

In the case of the particle filter belief updates the particles are first scaled similarly to the robot position. The scaled particles are used to evaluate the MGF at several learned points, with the outputs used to comprise the belief embedding.

In the case of the discrete LaserTag problems, the actions are sampled from a softmax distribution, $a \sim \pi(\cdot \mid s) = \text{softmax}(g_\theta(s))$ where g_θ is the actor network. For the continuous LaserTag problems, the action space is $A = \{\text{:measure} \cup \mathbb{R}^2_{[-1,1]}\}$ which is represented as the hierarchical action described by the following logic:

TABLE I
EMPIRICAL RESULTS BY RUNNING DIFFERENT SOLVERS. AVERAGE PERFORMANCE WHERE $\pm$ CAPTURES A 95% CONFIDENCE INTERVAL.

Problem	QMDP	ARDESPOT	PPO
Variant 1 : Discrete with Exact Belief Updates	30 ± 12	69.9 ± 4.1	72.6 ± 2.9
Variant 2: Discrete with Particle Filter Belief Updates	34.34 ± 25.12	71.41 ± 8.98	48.5 ± 3.1
Variant 3: Continuous with Exact Belief Updates	-	27.1 ± 8.23	60.2 ± 6.9
Variant 4: Continuous with Particle Filter Belief Updates	-	23.7 ± 9.51	-25.6 ± 7.2

a_c, a_d {continuous and discrete action}
if a_d is :measure **then**
  $a \leftarrow a_d$
else
  $a \leftarrow a_c$
end if

In order to support hierarchical actions the modifications described by H-PPO are used. The discrete actions are sampled using a softmax distribution as in the discrete LaserTag setting, and continuous actions are sampled from a normal distribution parameterized by the actor network's outputs.

The same hyperparameters are used across all experiments where applicable. Parameters of note are: 100 particles used in the particle filter, 64 sample points for the MGF, linearly decaying learning rate, discount factor of 0.997, and 32 actors generating 512 samples each per update.

V. RESULTS

In this section, we present our qualitative results over different variants of the LaserTag problem. We have split our results for different solution techniques.

A. POMDP Solvers

This section presents the results for the two POMDP solvers that we have implemented. The empirical results for the POMDP solvers are the average results over 100 different trials where each trial had a maximum episode length of 100.

1) QMDP: The results obtained by running the QMDP solver are presented in column two of Table I. Since variants 1 and 2 have discrete robot states and robot actions, the QMDP solver could be used for it. As mentioned earlier, the QMDP algorithm works well for many POMDPs but not so well when the agent needs to take active information-gathering actions. The LaserTag problem is one such problem. As a result, the reward obtained by the QMDP algorithm is not so high and has high variance. We observed that there were a lot of trials where the agent just failed to track the target within the fixed episode length of 100. This could be attributed to the fact that the QMDP approach doesn't realize the importance of the ":measure" action.

For variants 3 and 4 which have continuous robot states and robot actions, the existing QMDP solver in Julia could not be used right away as there is no existing solver that integrates the approximate solution for the underlying continuous MDP with the belief.

2) ARDESPOT: The results obtained by running the ARDESPOT solver are presented in column three of Table I. At every time step, the algorithm was run with 50 scenarios and for a time duration of 0.5 seconds. As can be seen from the results table, the ARDESPOT algorithm did much better than the sub-optimal QMDP algorithm for the discrete problems in variants 1 and 2. For the continuous problem in variants 3 and 4, it didn't do very well. We believe this is primarily due to the ineffective roll-out policy for obtaining the lower bound on the value function. The roll-out policy implemented was to try to move in a straight line to the grid cell that has the highest probability of the target being in it. For problems with sparse rewards, an ineffective roll-out policy causes the algorithm to fail in distinguishing good actions from bad actions [11] and this leads to sub-optimal behavior. We also ran the experiments with roll-out policies that are worse than the current one (random policy; no rollout) and got inferior performance which is what was expected. It is often difficult to design effective roll-out policies in such cases and that brings another limitation of these online solvers into light.

B. Deep Reinforcement Learning

The results obtained by running the PPO algorithm are presented in column four of Table I.

Each problem was run for 5 million steps on 3 seeds with the learning curves shown in Figure 2. Note that the learning rate was linearly annealed to obtain stable results, and increased performance may be gained from increasing the training time. Notably with exact belief updates PPO was able to match or exceed ARDESPOT's performance, and in all cases PPO matched or exceeded QMDP's performance. It can be seen that progressing from discrete to continuous robot states (discrete to parameterized actions), or from exact to particle beliefs, results in a decrease in performance and potentially lower sample efficiency. We expect with more training time and a sufficient hyperparameter search the continuous state problems and particle filter belief update problems should at least match the performance of the discrete state problem.

While exploring hyperparameters several results came to light. Perhaps the most sensitive parameter was the number of steps each agent took between updates, with decreased step counts leading do degradation in training such that the

policies are unable to find the target. Unfortunately, increasing the number of steps between updates leads to lower sample efficiency (if the agent can sufficiently train), and this may be a parameter to continue exploring in the future and the cause of problem variant 4's relatively poor performance. Another parameter that had high sensitivity to learning was the Generalized Advantage Estimation parameter λ, with values around 0.95 performing best.

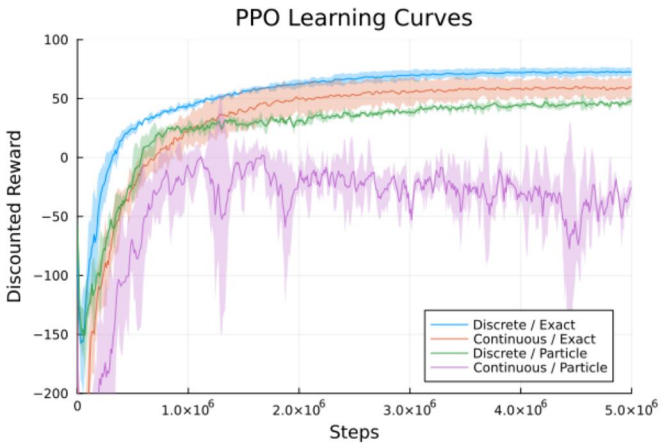

Fig. 2. Average discounted return over 3 seeds with 95% CI bounds.

To explore the learning of the MGF sample points we look at learned points in Figure 3. In addition to the MGF we also trained an encoding using the log of the MGF (also known as the cumulant generating function (CGF)). Sample points were initialized according to a normal distribution with zero mean and possible standard deviations of 0.01, 0.1, and 1.0. For the two smaller valued standard deviations we see that both the MGF and CGF converge to the same distribution, suggesting that the encoding is improving the sample points towards some local minima.

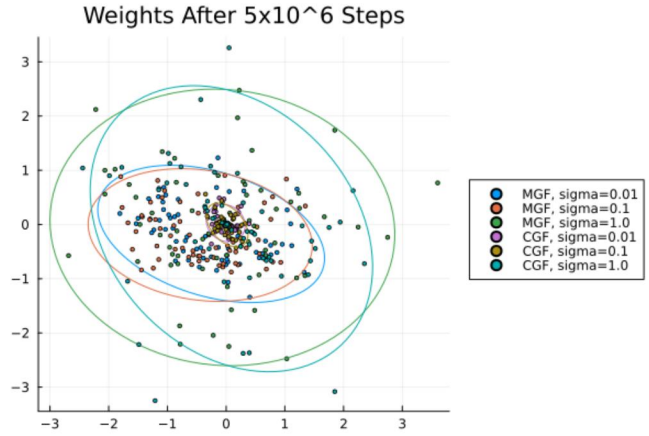

Fig. 3. Learned MGF and CGF weights with 95% CI error ellipses.

Fundamentally, the particle filter with MGF approach depends on the number of particles available to the agent. After training on 100 particles in the discrete state setting we see in Figure 4 the policy can work with a varying number of particles as expected, and increasing the number of particles corresponds to an increase in reward. This is further indication that the MGF evaluation step is appropriately learning some encoding of the belief distribution, without the need for kernels or heuristics.

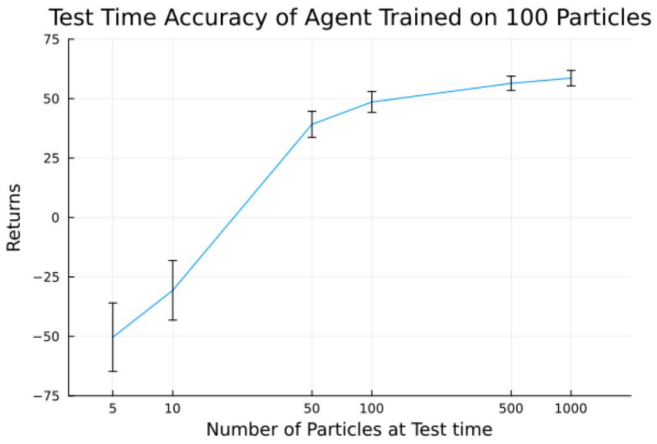

Fig. 4. Performance of learned policy for the discrete state and action problem with particle filter belief over a varying number of particles at test time.

VI. Conclusion and Future Work

This paper explored solving POMDPs as belief MDPs over various representations of belief, state, and action spaces.

The preliminary findings support the use of online planners for partially observable decision-making problems when working with discrete state and action spaces. Incorporating continuous spaces can lead to issues with commonly used POMDP solvers, with QMDP unable to handle continuous states and ARDESPOT unable to handle continuous actions. Conversely, the "offline" solutions provided by a deep reinforcement learning agent were able to account for continuous and parameterized spaces, albeit with increased sample complexity. This leads us to believe more complex solutions involving online tree searches informed by pre-trained policies are worth exploring.

To fully explore the power of DRL in the context of solving POMDPs the problem complexity must be increased. In the example of the LaserTag problem, the sparse reward may become an excessive hindrance to on-policy learning. Off-policy methods used to address exploration should be investigated. Additionally, more experimentation and exploration are needed with regards to moment generating function representations of beliefs. RL agents are shown to extract information from the MGF encoding, but the initialization and number of sample points needed remain unclear. Performance must also be compared to belief representations based only on the mean and covariance of the particles.

VII. Release

The authors grant permission for this report to be posted publicly.

References

[1] Michael L Littman. "The witness algorithm: Solving partially observable Markov decision processes". In: *Brown University, Providence, RI* (1994).

[2] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[3] Hanna Kurniawati, David Hsu, and Wee Sun Lee. "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces." In: *Robotics: Science and systems*. Vol. 2008. Citeseer. 2008.

[4] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. "Point-based value iteration: An anytime algorithm for POMDPs". In: *Ijcai*. Vol. 3. 2003, pp. 1025–1032.

[5] Trey Smith and Reid Simmons. "Heuristic search value iteration for POMDPs". In: *arXiv preprint arXiv:1207.4166* (2012).

[6] David Silver and Joel Veness. "Monte-Carlo planning in large POMDPs". In: *Advances in neural information processing systems* 23 (2010).

[7] Nan Ye et al. "DESPOT: Online POMDP Planning with Regularization". In: *Journal of Artificial Intelligence Research* 58 (Jan. 2017), pp. 231–266. ISSN: 1076-9757. DOI: 10.1613/jair.5328. URL: http://dx.doi.org/10.1613/jair.5328.

[8] Zachary Sunberg and Mykel Kochenderfer. "Online algorithms for POMDPs with continuous state, action, and observation spaces". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 28. 2018, pp. 259–263.

[9] Michael Kearns, Yishay Mansour, and Andrew Y Ng. "A sparse sampling algorithm for near-optimal planning in large Markov decision processes". In: *Machine learning* 49 (2002), pp. 193–208.

[10] John Asmuth and Michael L. Littman. "Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search". In: *CoRR* abs/1202.3699 (2012). arXiv: 1202.3699. URL: http://arxiv.org/abs/1202.3699.

[11] Himanshu Gupta, Bradley Hayes, and Zachary Sunberg. "Intention-Aware Navigation in Crowds with Extended-Space POMDP Planning". In: *arXiv preprint arXiv:2206.10028* (2022).

[12] Michael H Lim, Claire J Tomlin, and Zachary N Sunberg. "Voronoi progressive widening: efficient online solvers for continuous state, action, and observation POMDPs". In: *2021 60th IEEE conference on decision and control (CDC)*. IEEE. 2021, pp. 4493–4500.

[13] Matthew Hausknecht and Peter Stone. *Deep Recurrent Q-Learning for Partially Observable MDPs*. 2017. arXiv: 1507.06527 [cs.LG].

[14] Maximilian Igl et al. "Deep Variational Reinforcement Learning for POMDPs". In: *CoRR* abs/1806.02426 (2018). arXiv: 1806.02426. URL: http://arxiv.org/abs/1806.02426.

[15] Xiao Ma et al. "Discriminative Particle Filter Reinforcement Learning for Complex Partial Observations". In: *CoRR* abs/2002.09884 (2020). arXiv: 2002.09884. URL: https://arxiv.org/abs/2002.09884.

[16] Robert J. Moss et al. *BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations*. 2023. arXiv: 2306.00249 [cs.AI].

[17] Mauricio Araya et al. "A POMDP extension with belief-dependent rewards". In: *Advances in neural information processing systems* 23 (2010).

[18] Neil Gordon, David Salmond, and Craig Ewing. "Bayesian state estimation for tracking and guidance using the bootstrap filter". In: *Journal of Guidance, Control, and Dynamics* 18.6 (1995), pp. 1434–1443.

[19] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. "Learning policies for partially observable environments: Scaling up". In: (1995). Ed. by Armand Prieditis and Stuart Russell, pp. 362–370. DOI: https://doi.org/10.1016/B978-1-55860-377-6.50052-9.

[20] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[21] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv: 1506.02438 [cs.LG].

[22] Zhou Fan et al. "Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space". In: *CoRR* abs/1903.01344 (2019). arXiv: 1903.01344. URL: http://arxiv.org/abs/1903.01344.

[23] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter". In: (1995).

VIII. Contributions

(On a new page so can be removed for peer review.)

A. Himanshu

Himanshu was primarily responsible for the LaserTag environment and POMDP solvers. The LaserTag environment was written from scratch, but derived from Dr. Sunberg's DMUSTudent.jl Julia package. The modified code consists of different variants of the LaserTag problem and something which is very generic, easy to use and easy to extend. It will be interesting if Dr. Sunberg incorporates these changes for his DMU class to modify the assignments. The belief updaters were for both exact updates and particle filter updates were also written from scratch, made to accommodate changes to the observation. QMDP and ARDESPOT solutions come solely from Himanshu, using off-the-shelf solvers and custom-written heuristics for the problem. There is also an evaluation pipeline in place to evaluate different solvers with different updaters.

B. Jackson

Jackson was primarily responsible for the DRL algorithm and training. PPO implementation is fully work done by Jackson, but not entirely from the scope of this project. Primary algorithmic work during this project involved writing the interfaces for and expanding the capabilities of the actor-critic networks. This includes incorporating parameterized action spaces, writing training interfaces for multiple types of critic networks, and hand-coding derivatives for certain networks.

C. Repositories

Detailed code and gifs can be found at the following Github repositories.

- https://github.com/himanshugupta1009/SolveBeliefMDP.jl
- https://github.com/the-one-and-only-jackson/RLAlgorithms.jl

1 Final Project Report

- 0 pts Correct