Online Informative Path Planning for Drone Mapping with Reinforcement Learning

Harry Freeman (hfreeman), David Russell (davidrus)

***Abstract*—In this work we tackle the problem of exploring an unknown region with a scalar field. We train a variety of agents to choose samples based on the previous observations. We find that a variety of agents perform better than random on this task.**

I. Introduction

The overall goal of this work is to simulate an agent that is trying to map a quantity of interest in an unknown environment. We are motivated by the problem of a drone exploring an agricultural field to try to sense specific traits, for example, the presence of pests. It is possible to plan an exhaustive trajectory that covers the entire environment, but this can be expensive in practice. For example, drones have a limited battery that precludes exhaustive surveys of large areas. Since many natural phenomena such as pests are heavily spatially correlated, an intelligent sampling of the environment can often lead to results that are significantly better than a naive strategy.

The field of informative path planning deals with this exact problem of having an agent collect information about an environment by figuring out where to sample next in an **online** manner. A number of approaches have been proposed for IPP [1, 3], but they are often only evaluated in one environment and may not translate easily to other domains. They often also have a number of tunable parameters that must be explored to give good performance. Finally, existing IPP methods often require computationally expensive optimizations or tree searches, which can limit the feasibility for fast-moving agents. We propose to use reinforcement learning as an approach for informative path planning. A recent work that applies RL to IPP is [5] and we will use this as inspiration, though it tackles the problem differently.

II. Environment

In this work, we are focusing on a simple environment that we designed from the ground up during previous research. It is not meant to be fully realistic, but rather serve as a testbed that we can use to generate intuitions for the real world.

A. World

We represent our simplified world as a 2D plane that we wish to explore. On this plane, we have a continuous function that represents some quantity of interest, like the degree of pest infestation. The goal of our toy example is to mimic the spatial correlations that are common in natural phenomena. One way to generate this map is to use a mixture of gaussians to represent the density of some quantity of interest. We begin by choosing a number of gaussians to use. In this case, we use four. Then, for each gaussian, we sample a mean within the bounds of the world and axis-aligned variances uniformly between an upper and lower range. In this case we used a 20x20 unit world and the gaussian axis variances were sampled between 3 and 15. After creating each guassian PDF with the appropriate mean and variance, we sum these PDFs and use min-max normalization so the map is between 0 and 1. Then a scaling and shifting is applied, so the values are between 0.5 and 1, which improves learning later one. An example map can be seen in Figure 1.

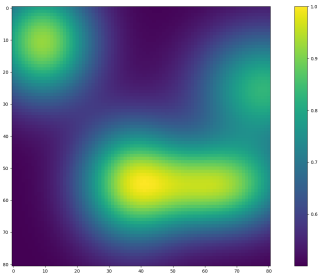

Fig. 1. Sum of four randomly-selected Gaussian maps. This is one example of an environment we are trying to explore.

B. Agent and Action Space

In our simplified problem, our agent represents the drone and its x and y coordinates are the location of the drone in 2D space. In our baseline approach, the action space of the agent is discrete. We represent the world by divided it into an NxN grid where N is a controllable parameter by the user. At each step, the agent is free to move to any position in the world on this NxN grid. Once it moves, it samples the value of the environment at the location. The represents the measuring of the value of the scalar field at the location of interest. We assume that both the measurement value and the position are free from noise, however our code supports introducing noise into these measurements if desired.

C. Episodes

We consider our episode to run for a fixed number of timesteps. This represents the situation where a drone's battery limits the flight duration. Given the agent can move to any discretized location in the world at each step, we understand this is not a true representation as travelling larger distances

requires more time and effort. Therefore, in the future we will adjust the episode length to consider distance travelled.

D. Belief Modeling and Observation Space

The overall goal of our agent is to model the world with a limited number of samples. In our initial attempt, we sought to model locations which we had not visited using Gaussian Process regression, as is common in the IPP literature. However, after initial experimentation, we found this was too computationally expensive to be feasible in the inner loop of the training procedure. Therefore, we reverted to a simple model of the environment. The observation space is discretized into an MxM grid and each location has a mean and variance, where M is a user-controlled parameter. M was configured to be the same as N, although our code supports using different values. The reason we chose this implementation was so the agent can effectively observe each grid cell it can move to. Once the agent moves to a new cell on the grid, the variance of that cell is reduced from 1 to 0. The mean is also updated according to the update belief model as discussed in Section II-E. A more complex model could be considered in the case of sensor noise or more complex prior beliefs about the distribution, but we found this sufficient for initial experiments. For more-effective model training, the observation space outputs are scaled and clipped to lie in the range of [-1, 1].

E. Processing Observations

We wish to give the agent information about the previous samples it has observed, so it can choose where to go next. Rather than providing the raw samples to the agent, we use our belief model as an intermediate representation to simplify the problem. We update our belief model every time a new sample is drawn, and feed the agent a grid sampled from it covering the whole world.

This simplifies our observation to something that can be thought of as a two-channel image, one representing the predicted mean of our scalar field and the other representing the predicted variance. Since the observation is always centered around the agent, we do not need to encode the global position, since we assume that we can only reason on this local neighbor. Exactly how large this context should be is a design decision we will need to address experimentally.

Note that in deployment, this belief model would be maintained by the agent. However, for implementation reasons, it is part of the environment. This is because, in addition to being used to generate the observation, the belief model is used to compute the reward. Since the environment handles the reward computation in the `gym` abstraction, the belief model must be part of the environment.

III. Reward function

At a high level, we want our agent to learn policies that allow it to plan a trajectory that samples informative regions of the environment. The end goal is to have a predicted map that matches the real map. This can be measured by sampling a grid of points from both the real and predicted maps and taking the mean squared error between the samples.

An additional consideration we introduce is that the agent should prioritize regions with a high value. This models a situation where something in your environment is more interesting than another part. Going back to our motivating example, it is more important to accurately assess the presence of pests in a heavily infested region than in a lightly infested one.

A simple approach was proposed by Popovic et al [3], where they only consider the mean squared error on the 40% of the ground truth map that has the highest values. We began with this approach, but realized that it was unneeded in our setting. Because we have a prior belief that every cell has a value of zero, we reduce our error more by sampling a new high-valued cell than a low-valued one. The discussion of the thresholded reward function is provided for completeness, but we can consider this threshold to be set at evaluating 100% of points.

To formalize our reward function, consider X to be a dense set of 2D points sampled from our environment. The number of sampled points in X is greater than M and N used to sample the action and observation space. We will evaluate the quality of our predictions at these points. Let y be the vector of ground truth values corresponding to our map at these locations. Our predicted values are taken from our belief model at the same locations, $\hat{y} = B(\theta, X)$, where θ is the parameters of the belief model.

The overall loss of our prediction can be assessed as follows

$$L = ||\hat{y} - y|| \tag{1}$$

More explicitly, this can be noted as

$$L = ||B(\theta, X) - y|| \tag{2}$$

To give our agent more feedback, we consider our loss to be incremental rather than episodic. Therefore, our reward is the amount that our prediction on the areas of interest improves due to the most recent action. If we index the belief model parameters by timestep as θ_t, the reward for the tth timestep is

$$R_t = -[||B(\theta_t, X) - y|| - ||GP(\theta_{t-1}, X) - y||] \tag{3}$$

More compactly, this is

$$R_t = -[||\hat{y}_t - y|| - ||\hat{y}_{t-1} - y||] \tag{4}$$

Note the leading negative because the reward is the negative of our cost.

IV. Methods

A. On-policy Policy-gradient RL:

For our on-policy gradient algorithm, we utilize PPO as it is a state of the art approach very commonly used in practice. While one of the benefits of gradient-based are that they require fewer hyperparameters to tune, this comes at a cost of sample efficiency. However, we designed our world model and observation space to allow for fast sampling to allow for PPO to learn a policy within a reasonable number of rollouts.

B. Off-policy Q-function-based RL:

For our off-policy gradient algorithm, we use DQN. This is because it is simple and supports our discretized action space as input.

C. Model-based RL:

For our model-based RL implementation, we train a network to predict state changes given observation-action inputs. At runtime, the agent samples actions and passes them to the network along with the current observation to determine the action that leads to the most optimal next state. Because there is no access the ground-truth map at runtime, we cannot use the reward to determine the best action. Instead, we use a pseudo-reward of

$$P_{R_t} = \frac{1}{M^2} \sum_{y=0}^{M-1} \sum_{x=0}^{M-1} u(y,x) - \sigma^2(y,x) \tag{5}$$

where u and σ^2 are the mean and variance values of the predicted next state's observation. This is because we want to reach states with higher values and lower variances. The sampled action that leads to the next state with the highest pseudo-reward is selected.

D. Modification 1: Behavioural Cloning and DAgger from expert:

For our first modification, we would like to use Behaviour Cloning and DAgger to reproduce behaviour demonstrated by an expert. Our expert, which we denote as the Perfect agent, has full knowledge about the environment, and can always determine the next best location to move to. As a result, for each action the expert takes, the maximum possible reward is returned, and the final predicted map is as close to ground-truth as possible. The Perfect agent's performance is an upper bound on any other agent that acts in the environment.

We train both Behavioural Cloning and DAgger agents using observation-action pairs produced by the Perfect Agent. We suspect the Behavioural Cloning will not be successful. This is because of the wide distribution of observations, especially in the initial states, that will result in the Behavioural Cloning agent to perform poorly early on and from which it will be unable to recover. However, with the ability to query demonstrations from the expert online, we suspect that DAgger should be able to perform reasonably well.

E. Modification 2: Continuous Action Spaces

While our discretized action space approach simplifies the problem, it does not truly represent the real world. For example, a drone is not limited to traveling to only a fixed set of grid locations, but should be able to fly to any location of choice. As a result we explore the performance of continuous action spaces. We use the policy to learn the exact horizontal and vertical world coordinates the agent should to move to. To achieve this, we adapt our model-based design to work in the continuous space.

For model-free methods, it required a fair bit of experimentation and hyperparameter tuning to find an algorithm that would work in the continuous space. We tried PPO, SAC, and DDPG, all with no success. Eventually, we were able to get an implementation of TD3 to learn with a continuous action space. DQN was not evaluated as it only support discrete action spaces.

For our continuous action space implementation, actions outputted by the networks were scaled between [-1, 1] and unscaled inside the environment.

F. Other modifications

There are other modifications that we would potentially like to explore, time permitting. These include different observation space encodings, removing the need for ground truth by designing a reward function using information gain, extending our environment into 3D, and introducing a cost of movement or time of flight.

V. Experiments and Results

A. Training Details

1) PPO: To train PPO, we used Stable-Baselines3 [4]. The selected network architecture was their default MLPPolicy to which we passed our flattened observation space. The default hyperparameters from https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html were used. The network was trained for 300000 steps.

2) DQN: Stable-Baselines3 was also used to train DQN. Once again, we used the default MLPPolicy. Hyperparameter tuning for DQN required signifcantly more effort to get reasonable results compared to PPO, which is expected for Q-function-based algorithms. We ended up using the hyperparameters from https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html, with the following modifications:

- learning_starts: 10000
- batch_size: 128
- target_update_interval: 250
- exploration_fraction: 0.2
- exploration_final_eps: 0.1
- tau: 0.01

The network was trained for 100000 steps.

3) TD3: TD3 was also trained using Stable-Baselines3 with the default MLPPolicy. Hyparparameter tuning was quite involved. Our final implementation used the hyperparameters from https://stable-baselines3.readthedocs.io/en/master/modules/td3.html along with the following modifications:

- learning_rate: 5e-4
- buffer_size: 10000
- learning_starts: 10000
- batch_size: 128
- tau: 0.01

The network was trained for 300000 steps.

4) Model-Based RL: We implemented our own model-based RL training framework with code inspired from hw4. The network was a 5-layer MLP with hidden-layer size of 256. The input to the network was the observation and action spaces described in Section II and the output was the predicted delta in observation space. Unlike in hw4, we did not normalize the inputs to the network because they observation and action spaces were already scaled to within a reasonable range and we saw no noticeable effect on performance. To train the network, the agent acted randomly in the environment and the observation, action, and next observations were recorded in a replay buffer. The discrete and continuous networks were trained for 40 iterations, where each iteration ran 10000 steps and used 1000 gradient updates per iteration with samples selected from the replay buffer.

During runtime, the trained model was used to determine the best action. The way this was acheived depending if a discrete or continuous action space was used. For the discrete case, all actions were evaluated and the action that led to the predicted next state with the highest pseudo-reward as described in IV-C was selected. For the continuous case, 5000 random actions were sampled, and similarly the action that resulted in the highest pseudo-reward state was chosen. In the future we would like to explore longer planning horizons.

5) Behavioural Cloning: For behavioural cloning, we used the imitation [2] library. We collected a rollout of size 50000 from the expert policy and the network was trained for 100 epochs with a batch size of 256.

6) DAgger: For DAgger, we once again used the imitation library. Itaratively, rollouts of step size 2000 were collected by the expert and performed 100 gradient updates per iterations with a batch size of 256. A total of 10000 steps were performed by the expert policy.

B. Experiments

All experiments have a world size of 20 continuous spaced units and a discretized observation space of 7x7. For the discretized action space environmetns, the action space is divided into a 7x7 grid. The number of actions per episode is fixed at 20, just under 50% of the size of the action space. This was chosen because we felt a learned agent should be able to learn more about the world when exploring just under half of it compared to a random agent. We also compare the performance of each agent against a random agent, which randomly selects actions from a uniform distribution. We run 100 episodes for each agent and average the results across all experiments.

We provide results on the average reward per step and the map error represented in Eq. 2. We report both because the reward is what is used to train the PPO and DQN agents, but the minimizing the map error is the overall objective. Because our reward is incremental, if a learned agent explores efficiently early on then the rewards are expected to decrease over time, as shown by the Perfect Agent in Fig. 4. In such a case it is possible for the random agent to have higher rewards

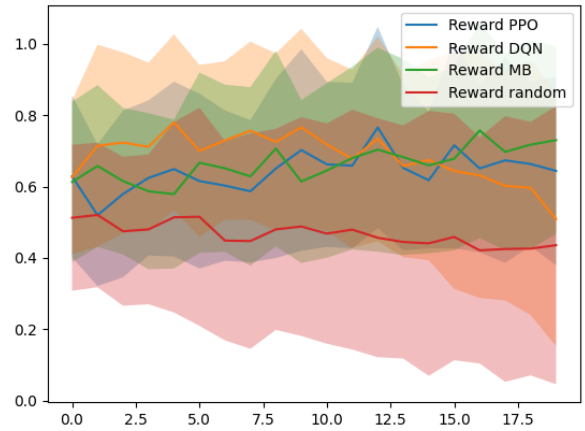

Fig. 2. Rewards for base methods averaged over 100 trials versus iterations. All methods perform better than the random baseline.

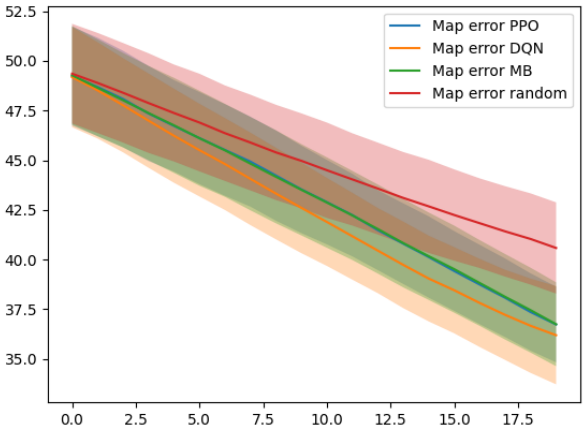

Fig. 3. Map errors per iteration for base methods averaged over 100 iterations.

near the end of the episode. However, a learned agent should have a lower map error by the end of training.

We also report the distribution of all final map errors in Fig. 8 and Fig. 9.

C. Results

1) Base Method Experiments : We evaluate the performance of our PPO agent, DQN agent, and model-based (MB) agent, in addition to the random agent. The averaged rewards and map errors across the 100 runs can be seen in Fig. 2 and Fig. 3.

All three of our learned agents outperform the random agent in terms of both average reward and average map error. The DQN agent performs the best, while our PPO and model-based agents perform quite similarly. One reason for this could be because a lot more effort went in to tuning the DQN hyperparameters compared to the PPO and model-based

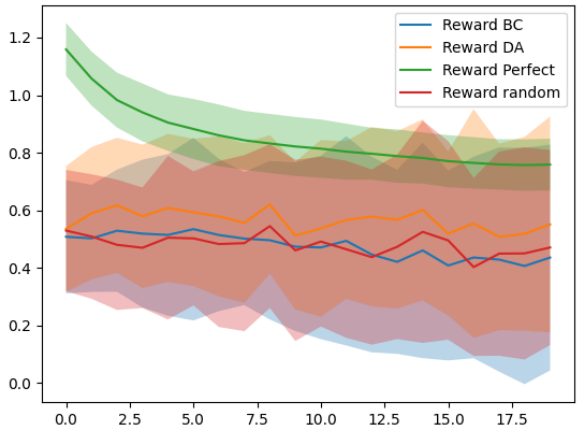

Fig. 4. Rewards for the model-based agents.

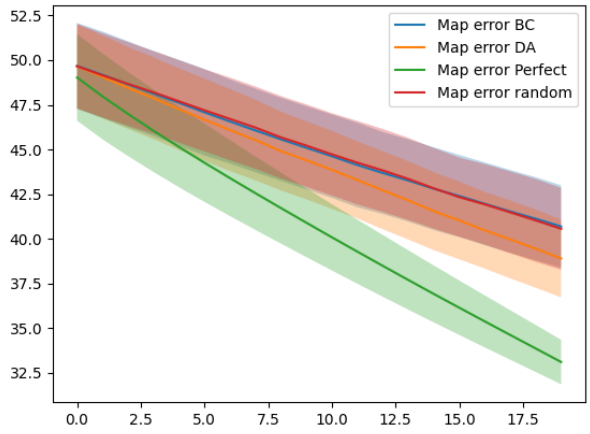

Fig. 5. Final map errors for the model-based agents over 100 iterations. Note that a perfect agent is presented to show the maximum possible values. DAGGER performs slightly better than random, while Behavior Cloning fails to learn.

agents. It is important to note that at the end of the episodes DQN has a lower reward than PPO and model-based. This is because it has learned more about the world and there is less incremental reward to improve upon, which is indicated by the lower map error.

As shown in Fig. 8, the PPO, DQN, and model-based agents had similar error distributions. DQN actually had a wider distribution, despite producing the best averaged results. If a minimum performance threshold is required, PPO or model-based would may more desired alternatives.

2) Modification 1: Behavioural Cloning and DAgger: We compare the performance of our Behavioural Cloning (BC) and DAgger (DA) agents against the random agent. We also provide the results for the Perfect Agent to demonstrate the upper and lower bounds of reward and map error. The results can be seen in Fig. 4 and 5.

As expected and described in IV-D, the Behavioural Cloning is unable to effectively learn and performs as well as the random agent. This is because of its inability to recover from error in new environments that have data outside of the distribution it was trained out. DAgger, on the other hand, is able to outperform the random agent. This is because during training it was able to query behaviour from the Perfect Agent. The DAgger agent does not perform as well as the agents in V-C1.

As seen in Fig. 8, DAgger has a similar distribution range to the base methods but with a higher mean final map error. Behavioural Cloning has a similar map error distribution to the random agent, with a smaller standard deviation. As expected, the perfect agent has the lowest mean and narrowest final map error distribution.

3) Modification 2: Continuous Action Spaces: We run our continuous model-based agent and TD3 agent and compare the performance to the random agent. The results can be seen in Fig. 6 and 7.

Our model-based agent and TD3 agent both outperform the

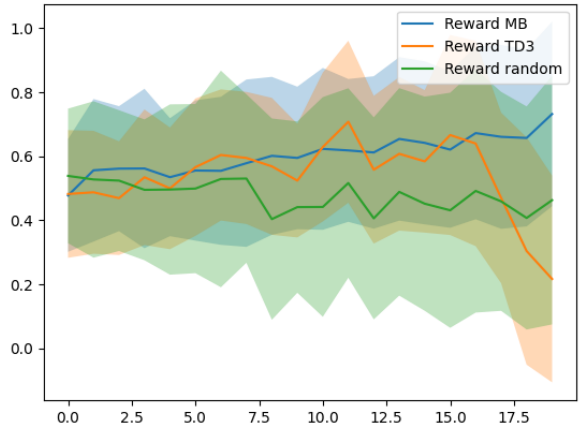

Fig. 6. Rewards for the agents on the continuous environment. TD3 does well until the last few iterations when performance stagnates. The model-based agent does consistently better than random.

random agent. While the random agent has higher rewards near the end of the episode than the TD3 agent, this is because of the incremental rewards as TD3 is still able to achieve a lower final map error. What is interesting is the model-based agent achieves a higher final map error with a continuous action space compared to a discretized one, which are 38.3 and 36.7 respectively. As well, TD3 had higher final map errors compared to the PPO, DQN, and DAgger experiments. This suggests that there is value to discritizing the action space and limiting the range of possible actions. This could be because it enables the network to better generalize and learn. This makes us curious to explore how performance varies when modifying the discretization of the action space to larger and smaller sizes.

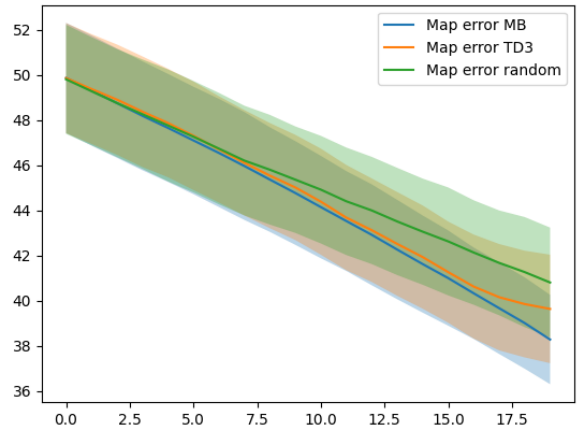

Fig. 7. Final map errors for the agents on the continuous environment.

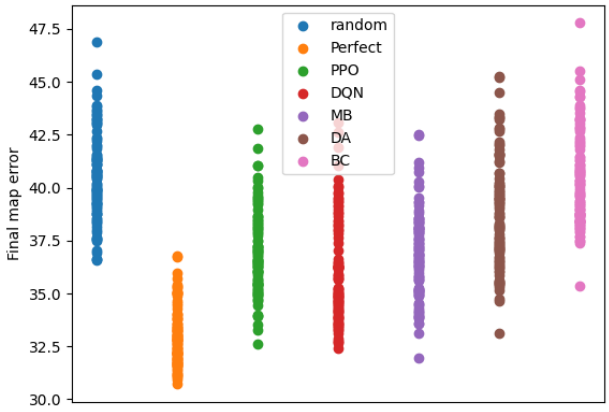

Fig. 8. Final errors for the discrete agents.

As demonstrated in Fig. 9, the continuous model-based agent has a lower averaged final map error but a wider distribution compared to TD3. The relationship is similar to that of the TD3 agent and the random agent. Directly comparing Fig. 8 and Fig. 9, the discrete agents have smaller mean and narrower error distributions compared to the continuous agents (apart from the behavioural cloning agent).

D. Conclusion

For this project, we simulated a drone navigating a simplified 2D world with the objective of collecting as much meaningful information from the world as possible. We demonstrated that a variety of reinforcement learning-based approaches outperform taking random actions. These approaches include policy gradient-based, q-value based, and model-based. As well, we showed that imitation learning with DAgger also provides improved results, as well as utilizing both discretized and continuous action spaces. In the end, DQN with a discretized action space achieved the best results. One open question that remains is if DQN is the optimal approach, or if dedicating more time to tuning hyperameters and training other implementation could result in greater improvements.

E. Future Work

Our next step is to understand what the agent is learning. In Figure 10, we show the final map result from a PPO agent. It seems like it may have done well either because it explored near other high-value regions, avoided sampling regions multiple times, or tended toward the center, where reward will be expected to be higher. Teasing apart these considerations will help us extend this work to more complex environments.

We would also like to add cost to movement to make navigating more realistic. In the real-world, a drone cannot simply travel far distances as easily as shorter ones. We could

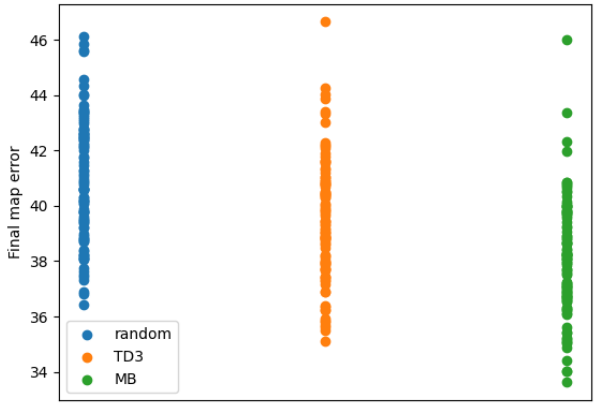

Fig. 9. Final errors for the continuous agents.

achieve this by either adding a cost of movement or terminating the episode after a certain distance has been travelled. Alternatively, instead of allowing the agent to move anywhere, we could restrict it to moving within a certain distance per step. This would lead to additional interesting implementations as horizon planning would have to be considered.

Other ideas we have include integrating sensor noise to make sampling more representative of the real-world, and incorporating interpolation of known sensor measurements into our predictions, which we think will reduce final if integrated properly, with the cost of adding difficulty of learning how to reason about the observation space.

Lastly, we would like to add temporal reasoning into our learned policies, which we can implement by utilizing consecutive observations.

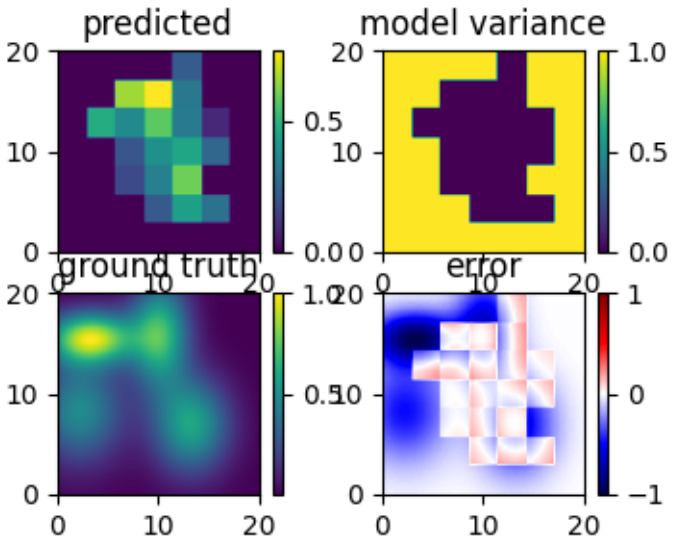

Fig. 10. A qualitative result from PPO.

F. Video Links

Videos can be found here: https://drive.google.com/drive/folders/1fTVB4RdCZFvNKls3c6dHxlpkzdnKyZ2K

References

[1] Alberto Candela, Kevin Edelson, Michelle M. Gierach, David R. Thompson, Gail Woodward, and David Wettergreen. Using Remote Sensing and in situ Measurements for Efficient Mapping and Optimal Sampling of Coral Reefs. *Frontiers in Marine Science*, 8(September):1–17, 2021. ISSN 22967745. doi: 10.3389/fmars.2021.689489.

[2] Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell. imitation: Clean imitation learning implementations. arXiv:2211.11972v1 [cs.LG], 2022. URL https://arxiv.org/abs/2211.11972.

[3] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Jen Jen Chung, Inkyu Sa, Roland Siegwart, and Juan Nieto. An informative path planning framework for UAV-based terrain monitoring. *Autonomous Robots*, 44 (6):889–911, 2020. ISSN 15737527. doi: 10.1007/s10514-020-09903-2.

[4] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268): 1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[5] Julius Rückin, Liren Jin, and Marija Popovic. Adaptive informative path planning using deep reinforcement learning for uav-based active sensing. *CoRR*, abs/2109.13570, 2021. URL https://arxiv.org/abs/2109.13570.