Dual-Robot Path Planning with geometries through narrow passages with Max Entropy Inverse Reinforcement Learning

Hoang-Dung Bui,
George Mason University
Fairfax, USA
hbui20@gmu.edu

I. Introduction

Multi-Robot Path planning (MRPP) plays a fundamental role in robotics, with various applications including transportation, self-driving, and search and rescue. The objective is to enable a robot group to move to desired locations while avoiding collisions with obstacles and other robots. In this project, we consider a problem where two robots with geometries need to go through a narrow passage which is fit for only one robot (Fig. 1). The two robot in (red and green squares) plan to move to their goals (red and green circles). To reach the goals, the robots must pass through a narrow corridor. The space at the middle are expanded and can fit for two robots. If the robots are at the two corridor ends at the same time, collision will occurs. The action space and state space are both continuous. The valid paths for the robots can be found by search-based algorithm as A^*, however, its runtime is a disadvantage. We would like to speed up the runtime, then combine with robot dynamics, which is a critical step to apply to real world application. Therefore, we would like to develop an machine learning framework which can help the robots planning faster in those scenarios. Furthermore, the framework can be applied in distributed robot systems, which then ease to apply in practice.

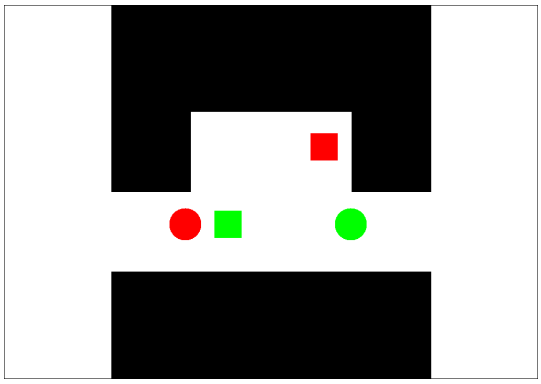

Fig. 1. **Planning for two robots going through a narrow passage**. Collision will occurs if two robot goes into the passage in the same same and without any collision avoidance techniques.

The recent progress in Inverse Reinforcement Learning (IRL) reveals a potential to solve complex scenarios [1], [2]. That wonders me triggers my interest to use IRL to solve the mentioned problem. The basic idea behind IRL is to determine a reward function from the expert demonstrations that could explain the expert behavior. The reward function is then to guide robot's moving in complex situations which make hand-writing the function challenging.

II. Related Work

Path planning is a well-known research field in robotics. There are variety of methods to solve the problem. One popular planning family is A^*-based motion planners [3], which converts the environments into discrete spaces, then representing them as a graph. The planners then perform path-finding on the graphs. However, with multi-robots, the search space is high dimensional, so the runtime of A^* is an disadvantage. Another approach is the sampling-based planners such as RRT [4] or PRM [5], which rely on sampling from collision-free spaces. Those planners require knowledge of the environment such as geometry and location of obstacles. In this project, we modify RRT^* to plan for two robots as an expert to generate demonstrated trajectories.

With huge progress in IRL recently, Sartoretti et al. [2] integrated reinforcement learning and Imitation Learning into the planners, which help them to handle various challenging scenarios in grid-world environments. A problem in Imitation learning is the unstable performance with states which do not appear in the expert demonstration. IRL [6] can solve this problem by developing a reward function which can generalize the expert behavior. To avoid the bias and suboptimal solutions from demonstration, Ziebart et al. [7] use a probability distribution to represent the expert demonstration by using maximum entropy principals. Wulfmeier et al. [8] avoids the manual design of the reward function features by utilizing a deep neural network to approximate the reward values. However, the work cannot expand in continuous space due to using visited frequency for all state space. Another frameworks [9], [10] expanded Maximum Entropy IRL to continuous action space and state space by using sampling trajectories to build the probability distribution. However, the features of the reward functions are manual design.

In this work, we would like to build a Max Entropy Deep IRL framework working on continuous action space and state space. It is unexpected for me to realize that the desired solution for the proposed problem is quite novel and cannot solve it within several weeks.

III. Maximum Entropy IRL

Let a Markov Decision Process (MDP) be defined as $M = \{S, A, T, r\}$, where S denotes the state space, A denotes the set of possible actions, T denotes the transition model and r denotes the reward structure. Given an MDP, an optimal policy π^* is one which, when adhered to, maximizes the expected cumulative reward. Furthermore, a discounted factor $\gamma \in [0, 1]$ may be considered in order to discount future rewards.

IRL considers the case where a MDP specification is available but the reward structure is unknown. Instead, a set of expert demonstrations $D = \{\zeta_1, \zeta_2, ..., \zeta_N\}$ is provided which are sampled from a user policy π, provided by a demonstrator. Each demonstration consists of a set of state-action pairs such that $\zeta_i = \{(s_0, a_0), (s_1, a_1), ..., (s_K, a_K)\}$. The goal of IRL is to uncover the hidden reward r from the demonstrations.

To solve the problem, Ziebart et al. [7] proposed Maximum Entropy IRL, in which the probability of user preference for any given trajectory between specified start and goal states is proportional to the exponential of the reward along the path.

$$P(\zeta|r) \propto \exp\{\sum_{s,a\in\zeta} r_{s,a}\} \tag{1}$$

Due to high dimensionality and huge state space in real world application, it is impossible to construct rewards for every state explicitly. Therefore, we need a reward function approximation which works on state feature representation: $g : \mathbb{R}^N \to \mathbb{R}$, with N being the dimensionality of the feature space such that:

$$r = g(f, \theta) \tag{2}$$

A feature representation, f, is usually hand-crafted based on preprocessing and manually defined distance metrics.

There are several choices of model to approximate the reward function. One common way is mapping from state to reward by a weighted linear combination of features values. This choice can be appropriate in some scenarios, however, is suboptimal choice if the true reward consists of non-linear factor. Especially in collision avoidance, the distance between the agent to agent, and agent to obstacle should be over a threshold, and should push faraway if the distance is shorter than it.

To scale well with large feature spaces with complex situations, a deep architecture is a natural choice for the reward function with non-linear elements. The reward function is obtained as:

$$r \approx g(f, \theta_1, ..., \theta_n) \tag{3}$$

We solve the IRL problem by maximizing the joint posterior distribution of observing expert demonstrations, $\mathcal{D}$, under a given reward structure with the model parameters θ. The loss function as follow:

$$\mathcal{L}_\mathcal{D} = logP(\mathcal{D}, \theta|r) = log\frac{1}{Z(\theta)}e^{\sum_{s_j\in\zeta_i} r(f_{s_j},\theta)} \tag{4}$$

where $Z(\theta)$ is *partition function* for all state space.

The gradient is the sum of the gradients with respect to θ, and can be separated as following:

$$\frac{\delta\mathcal{L}_\mathcal{D}}{\delta\theta} = \frac{\delta\mathcal{L}_\mathcal{D}}{\delta r}\cdot\frac{\delta r}{\delta\theta} = (\mu_\mathcal{D} - \mathbb{E}[\mu])\cdot\frac{\delta}{\delta\theta}g(f, \theta) \tag{5}$$

$\mu_\mathcal{D}$ is the state visited frequencies on from the expert demonstration $\mathcal{D}$, while $\mathbb{E}[\mu]$ is the state visited frequencies of the state space. For continuous action and state space, $\mathbb{D}[\mu]$ is intractable, thus we need to find a way to approximate it.

A practical way to determine $\mathbb{E}[\mu]$ is we can generate a set of trajectories $\mathcal{D}_s$ from the current policy for each demonstration ζ_i, then build a state visited frequencies distribution. That is where value iteration comes into the framework. However, there are one huge problem with this approach: it only works well with discrete and action space, and becoming intractable in continuous state space.

IV. Methods

We would like to combine IRL and RL to work under one framework to solve our problem. The pipeline of our project is shown in Fig. 2

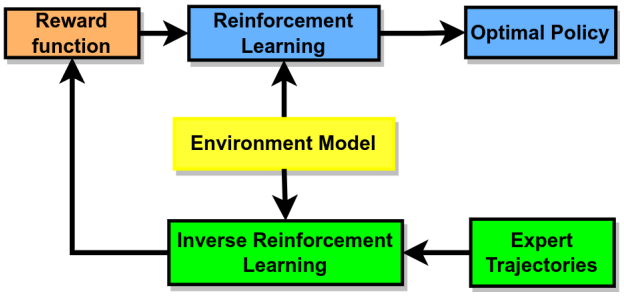

Fig. 2. **RL-IRL framework**. Expert trajectories are generated by an RRT-based planner. In IRL block, we would like to apply Maximum Entropy IRL to determine the reward function. For Reinforcement Learning, PPO algorithm is used to train the policy

For most parts, we have done the coding, which are described in detail in section V, except the IRL algorithm. We have reviewed multiple papers, and surprisingly, we can not find any work solving for continuous action state space with well-defined start and goal states. Therefore, we describe the desired algorithm in pseudo code in Alg.1, which are still in developing process.

Algorithm 1 IRL algorithm

INPUT: The demonstration dataset $D_M = \{\zeta_i^1, \zeta_i^2\}_{i=1:M}$;
OUTPUT: optimized reward function parameters $\theta*$;

$\theta_0 =$ initialize_weights()
Iterative model refinement
for i=0:N **do**
  Generate the sample set $D_s^0 = \{\tau_m^i\}_{m=1:K,i=1:M}$
  $r_i = \frac{1}{M}\sum_{m=0}^{M}$ nn_forward(ζ_m, θ^i)
  $r_i^s = \sum_{k=0}^{K}\frac{1}{M}\sum_{m=0}^{M}$ nn_forward(ζ_k^m, θ^i)
  $L_i^D = log\frac{e^{r_i}}{e^{r_i^s}}$
  $\theta_{i+1} = \theta_i + \alpha(\frac{\delta L_i^D}{\delta\theta^n})$
return θ

The system has two agents coordinating to going through a narrow passage without collision, so the demonstration dataset $\mathcal{D}$ consist of M couple trajectories for the two agents. To calculate the maximum entropy, we need to approximate the state visited frequencies in all in state space, and it is intractable in continuous state space. We will use an approximation by generating sample trajectories set for each expert trajectory, and calculate the probability ratio between the expert trajectory and sample trajectories. All the sample trajectories are stored in $\mathcal{D}_s$ (Alg. 1:3). This is the most challenging step which make our work being stuck, because how to determine the trajectories while the policy is still training.

Assuming we get the sample trajectories, then they are fed into the reward function to get the average reward for trajectories in $\mathcal{D}_M$ and sum of reward for all trajectories in $\mathcal{D}_s$ (Alg. 1:4–5). The loss function is defined in Alg. 1:6, and the update step for the reward parameter θ in Alg. 1:7.

V. Implementations

A. Building the environments

We build an environments with dimension of $10\ m \times\ 7\ m$ with a narrow passage (Fig. 1). The goals and starts points are set on two sides of the map, and two robots need to pass through the passages to reach their goals. The environment is represented as an occupancy grid with resolution of $6\ cm$. The obstacles inside the environment will have value 1 in the occupancy grid, and the free-space have value of 0.

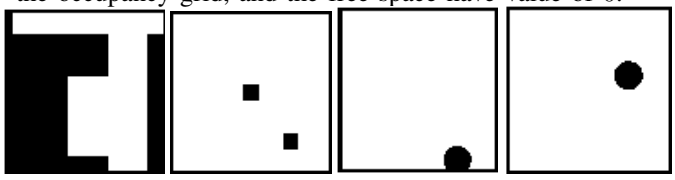

Fig. 3. **An observation returned from the environment** (corresponding to the situation in Fig. 1). The observation consists of four view channels from the FOV of current robot: Obstacle, robots within the view of the current robot (at the center), the current robot's goal, and the goal of other robots.

a) Observation space: We consider a partially-observable world, where agents can only observe the state of the world in a limited FOV centered around themselves (Fig. 3). The observation is a square with dimension of $6\ m \times 6\ m$, which is converted into 100×100 pixels view in practice. Assuming a fixed FOV can allow the policy to generalize to arbitrary world sizes and also helps to reduce the input dimension to the neural network. However, an agent needs to have access to information about its goal, which is often outside of its FOV. To this end, it has access to both a unit vector pointing towards its goal and Euclidean distance to its goal at all times. In the limited FOV, we separate the available information into 4 channels to simplify the agents' learning task. Specifically, each observation consists of binary matrices representing the obstacles, the positions of other agents, the agent's own goal location (if within the FOV), and the position of other observable agents' goals. As agents are close to the edges of the world, obstacles are added at all positions outside the world's boundaries.

b) Action Space: Agents take continuous actions in both x and y direction with values within the range of $[-1.0 : 1.0]\ m/s$ in the simulated world. At each timestep, certain actions may be invalid, such as moving out of the boundary, into a wall or another agent. During training, actions are sampled only from valid actions.

c) Collision Avoidance: To perform the collision avoidance with agent geometries, we build several collision checking functions. Assuming the agents have square shape with dimension *size*, so, the collision will occurs if any of agent's corners lie inside obstacles or outside the boundaries.

- *wall_collision*: checks whether any agent's corner lie outside the map/environment's boundaries.
- *obs_collision*: checks whether any agent's corner lies inside the obstacles. It is performed by projecting the position of the agent's shape into the occupancy grid. If there is any value in the intersection array greater than 0, so a collision occurs.
- *h2h_collision*: To simplify the calculation, if two robots are close than a distance limit, we set the robot-robot collision to *True*.

B. Building RRT as the expert

We deploy *RRT* as a centralized planner to plan for the two robots. We reuse the class *Link* from one assignment to represent the state of one agent at one position. A tree is developed and its nodes is a composite of two links corresponding to two agents. The cost of the tree nodes is the sum of two cost from the sub-nodes. The collision detection in the RRT-map are similar to the IRL environments, in which we need to consider the agent shape. There are three types of collision: agent-agent, agent-obstacle, and agent-boundaries.

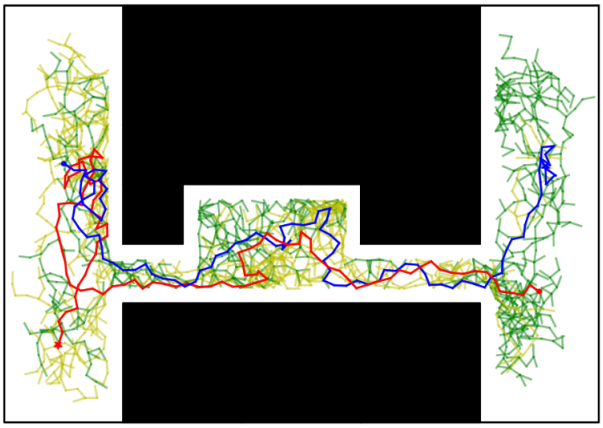

Fig. 4. **One solution of RRT results.** The *RRT* can determine paths for the two robots going through the narrow passages.

The *RRT* works well to generate the valid paths ζ_1, ζ_2 for the two agents, however, that paths are not optimal. If we have time, I think we should try *RRT** instead.

To generate the expert trajectories to feed to the IRL framework, we need to modify the output trajectories ζ_1, ζ_2 into the form of $\{s_0, a_0, s_1, a_1, ..., s_n, a_n\}$, and the state s_i consists four binary image channels and orientation to goal.

The actions are defined as:

$$a^i(x) = \zeta^{i+1}(x) - \zeta^i(x)$$
$$a^i(y) = \zeta^{i+1}(y) - \zeta^i(y)$$

The pair of ζ^i, a^i is then feed into the *step()* function in the environments, and we save all the returns as the expert trajectories.

C. Building a reward function

The algorithm to update the reward function is shown in Alg. 1. To avoid a potential collision as the multiple agents moving, we would like to add potential field force into the reward function, which is a quadratic function. Therefore, the reward function has a non-linear elements, and we would like to use a simple fully-connect neural network to approximate it.

D. Building a RL pipeline

After we have the reward function, it is integrated into the environment to return the reward after a robot make a move. To develop a RL framework, we rely on *PPO* algorithm [11] which exhibited its efficient performance in my experiments.

a) Proximal Policy Optimization: The advantage of PPO is it stabilizes the learning progress by preventing the overshoot of updating the neural network's parameters. It calculates the ratio $r_t(\theta)$ between the current and the previous policy.

$$r_T(\theta) = \pi_\theta(s_t|s_t)/\pi_{\theta_k}(a_t|s_t)$$

If the ratio is outside the range $(1-\epsilon, 1+\epsilon)$ - *Clipped Objective* $\mathcal{L}$, it will be trimmed to the closest limit.

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathbf{E}_{\tau\ \pi_k}[\sum_{t=0}^{T}[min(r_t(\theta)\hat{A}^{\pi_k}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_k}_t]$$

The selected limit and the ration are multiplied with the *advantage* value, and the smaller one between them will be used to update the neural networks.

$$\theta_{k+1} = argmax_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$$

b) Network's Structures: There are two CNN networks which estimate the policy and value function for PPO algorithm. Their structures are similar except the output layer (Fig. 5 and Fig. 6).

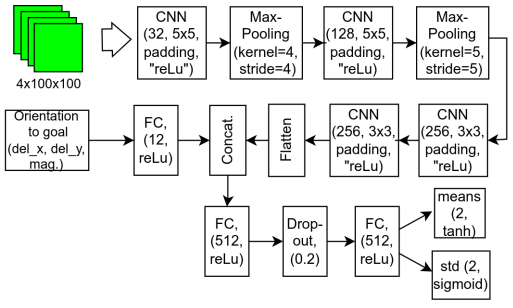

Fig. 5. Policy Network

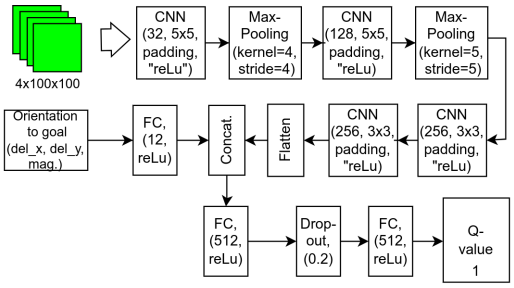

Fig. 6. Value Network

The loss function for the networks are:

$$value\_loss = (R + \gamma\hat{v}(S', \omega) - \hat{v}(S, \omega))^2$$

$$policy\_loss = \gamma\nabla log(\pi(a_t|s_t))(R + \gamma\hat{v}(S', \omega) - \hat{v}(S, \omega))$$

VI. Results and Discussion

As proposing this work, we would like to find a new approach to improve the performance of the existing planners for a typical situation in multi-agent planning. I have done the following work:

- Developing a distributed RL-IRL framework to work with two agents in mentioned context.
- Building an environment for two agents with their geometries in continuous action and state space; the environment returns 4 binary images channels, and an orientation vector of a robot to its goal; several collision detection are developed to handle the collision in this environments.
- developing a RRT-based planner for dual agents in continuous state space to generate collision-free paths.
- converting the paths from the RRT-planner into expert demonstrations, by feeding the each position on the trajectory into the environments step function of the environment.
- Constructing a PPO algorithm with CNN networks to estimate the policy and value function.
- Reviewing state-of-the-art papers about inverse reinforcement learning with maximum entropy using deep neural network, and continuous action state space. There are some limitations in the existing work, and I can see the room for improvement for future work.
- Moreover, we are on the progress on developing an IRL algorithm, which can work on continuous action and state space while can handle the non-linear factor in reward function structure.

However, the work is still no done due to several reasons:

1) To deploy maximum entropy IRL, we need to determine the state visited frequencies for all the states in state space - *partition function* - $Z(\theta)$. For discrete state space [8], we can develop a policy from the value iteration function, and perform a number of roll-outs, and use that to determine the *partition function*. However, the problem becomes intractable in continuous state space due to the unlimited states.

2) There is some ideas to approximate the *partition function* - $Z(\theta)$ by generating sample trajectories $\mathcal{D}_s$ [9], [10] and calculate the reward sum along the trajectories. Then the maximum entropy is calculate based on the expert demonstration $\mathcal{D}$ and $\mathcal{D}_s$. However, the approach is suitable to autonomous vehicles with safety requirements within a horizon T and no particular target in the movement. For our problem, we need to the robot reach its goal. However, the policy is still learning from the reward function, and it cannot assure that the agent can reach its goal. We need a new approach to solve this problem.

The code of this work can be found in this link. It is still actively developing.

REFERENCES

[1] Xiaonan He, Xiaojun Shi, Jiaxiang Hu, and Yingxin Wang. Multi-robot navigation with graph attention neural network and hierarchical motion planning. *Journal of Intelligent & Robotic Systems*, 109(2):25, 2023.

[2] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[3] Wei Zeng and Richard L Church. Finding shortest paths on real road networks: the case for a. *International journal of geographical information science*, 23(4):531–543, 2009.

[4] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[5] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[6] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[7] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[8] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[9] David Sierra González, Ozgur Erkent, Víctor Romero-Cano, Jilles Dibangoye, and Christian Laugier. Modeling driver behavior from demonstrations in dynamic environments using spatiotemporal lattices. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3384–3390. IEEE, 2018.

[10] Zheng Wu, Liting Sun, Wei Zhan, Chenyu Yang, and Masayoshi Tomizuka. Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving. *IEEE Robotics and Automation Letters*, 5(4):5355–5362, 2020.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.