Inverse Reinforcement Learning via Maximum Entropy Formulation

Tuhina Tripathi
ASEN 5519 Spring 2022
Group 12
University of Colorado, Boulder
tuhina.tripathi@colorado.edu

Alexa Reed
ASEN 5519 Spring 2022
Group 12
University of Colorado, Boulder
alexa.reed@colorado.edu

Sourav Chakraborty
ASEN 5519 Spring 2022
Group 12
University of Colorado, Boulder
sourav.chakraborty@colorado.edu

***Abstract*—The purpose of this report is to summarize Group 12's final project for ASEN 5519: Decision Making Under Uncertainty. This project explores the use of Inverse Reinforcement Learning, via Maximum Entropy Formulation, in a Markov Decision Process. The concepts explored in this project were demonstrated using a grid world environment.**

I. INTRODUCTION

Reinforcement Learning is a subset of Machine Learning that optimizes the sequence of decisions an agent makes in an environment. Reinforcement Learning attempts to determine the best set of actions, commonly referred to as a policy, that maximizes a reward function for a given environment. The reward function quantifies behavior and/or states in an environment, indicating to the agent that their actions are positive, negative, or neutral. This information is used to determine the policy, or policies, that will maximize the reward function and reinforces the behavior leads to the highest reward. In most reinforcement learning techniques, an agent will explore an environment and determine what actions correspond to the highest reward. This results in an optimal policy, or policies, that maximize the reward function.

Inverse Reinforcement Learning (IRL) is a subset of Reinforcement Learning that is useful in scenarios where the reward function is difficult to describe, or is unknown. IRL utilizes a "professional" agent that demonstrates ideal behavior in a given environment. The professional demonstrations are used to derive a reward function that describes the objective of the agent. Because this reward function is founded on ideal behavior, it can simplify decisions that are otherwise difficult for an untrained agent to make. In this way, the demonstrations are essentially providing a solution to the reinforcement learning problem and the reward function is derived accordingly.

A common example of IRL is training an autonomous driving system. There are several possible behaviors of interest in an autonomous car driving scenario, but one could determine the most important behaviors would be speed control, avoiding collisions with other cars or people, and obeying traffic laws. It would be difficult to describe a comprehensive reward function for each of these behaviors that prepares an agent for diverse environments or complex scenarios. It may also be difficult to describe a reward function that allows for flexibility in the solution. In this scenario it may be beneficial to accommodate different driving styles. Using IRL, a "professional" agent that is able to perform to all of the desired objectives in several environments will demonstrate ideal behavior that is used to determine a reward function. The information gathered from these demonstrations can be used to understand the objectives of the professional agent and is then used to train agents for the environment.

In this project, we implemented IRL on a Markov Decision Process (MDP) applied to a grid world scenario. The grid world was created with sparse rewards that were both positive and negative. We first used Value Iteration to understand the reward of each state in the grid world, and determined a policy for the optimal action at each state. This policy was used to create trajectories in the grid world that were considered the "professional" agent. We then used Maximum Entropy IRL to determine a reward function that was compared to the sparse rewards that the grid world was created with.

II. RELATED WORK

The concept of the IRL was introduced in [1]. They formulated the problem as a linear programming procedure with constraints corresponding to the optimal condition. Along with [1], [2] considered the cases of (a) Finite-state MDP with known optimal policy, (b) Infinite-state MDP with known optimal policy and (c) Infinite-state MDP with unknown optimal policy, but with demonstrations. For the infinite state space problems, they approximated the reward function using linear combination of all useful features [3] or using feature expectations [2].

[2] introduced apprenticeship learning with sample trajectories, which handled case (c), and tackles the problem of autonomous driving system mentioned in section I. The series of applications of this method was seen in the helicopter demonstrations [4], [5], [6]. Similarly, [7] developed a navigation controller that enabled a real-sized robotic car to learn different parking styles. [8] applied the formulation by [1] to imitate a pedestrian's behavior by a robot. IRL was also used for building dialogue systems [9].

The basic formulations mentioned above had several assumptions and hard constraints that made them unsuitable for most practical applications. The most prominent couple of them being (a) reward function assumed to be a linear combination of features, and (b) assuming that the expert's demonstrations were always optimal. New refinements were published to relax the basic assumptions and to look IRL from a new perspective. [10] introduced maximum margin planning (MMP) which uses quadratic programming formulation and introduces loss functions. Since, MMP still assumes the reward function to have a linear form, it was later extended to learn non-linear rewards by [11]. Bayesian IRL was introduced by [12]. [13] generalized this method to a prefence elicitation formulation in which the goal is to determine the experts posterior distribution of preferences, and also extending it to a multitask setting. Maximum entropy model was also introduced in the similar vein, which is discussed in section III.

III. INVERSE REINFORCEMENT LEARNING

Inverse Reinforcement Learning (IRL) attempts to determine the reward function of an environment through observed behavior of an optimally performing agent. IRL can be utilized in **imitation learning** where an agent replicates behavior rather than learning from it. In imitation learning, also referred to as **behavioral cloning**, the agent will replicate actions that may be irrelevant. This process does not necessarily derive a reward function as it can be accomplished with supervised learning. In this project we pursued **apprenticeship learning**, which is the application and learning process from the demonstrated behavior through determining the reward function. In apprenticeship learning, the agent can disregard actions that are irrelevant to the determined objective when trained on sufficient data by the professional.

One of the main challenges faced when using IRL is reward function multiplicity. The observed behavior of an agent is the basis to a policy, π, for which there may be many applicable reward functions that are optimized by π. Maximum Entropy Inverse Reinforcement Learning is one method that attempts to resolve reward function multiplicity and select a single stochastic policy. This approach heavily favors the trajectory or policy distribution that does not exhibit any additional preferences beyond matching feature expectations. This results in using a reward weight parameter, θ, to create a parameterized distribution over possible paths in the environment. Maximum Entropy formulation results in trajectories that have a higher total reward being selected exponentially more often than paths that yield a lower total reward, due to the reward weight parameter. We will expand more on our application of maximum entropy formulation, as well as applicable equations, in the following sections.

Another challenge in IRL is quality of data from the professional or demonstrator. This challenge is often faced when the demonstrator is a human and the data is recorded behavior in the form of videos or other sensors. Humans can also create data irrelevant to the goal they are demonstrating, such as scratching their face when demonstrating objective behavior for cooking an omelet, or clicking on an ad when demonstrating a behavior online. In this project, because the demonstrated trajectories can be perfectly observed, we were able to avoid this challenge. If however, there was an interest in doing live demonstration as the goal trajectories in which a human would navigate the grid world through an interface, any mistakes from the human could present this challenge in the IRL process.

IV. TECHNICAL APPROACH

A. PROBLEM FORMULATION

We make use of a $5X5$ grid world environment to implement the proposed Inverse Reinforcement Learning approach.

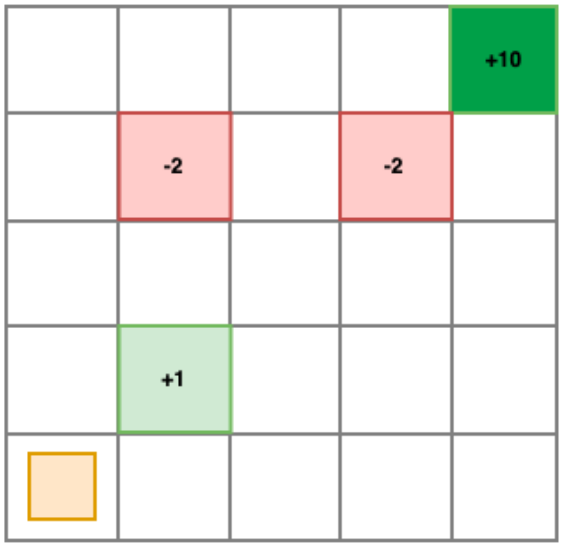


Fig. 1: 5x5 grid world with some sparse and negative rewards to test the implemented IRL algorithm

The grid world is formulated as an MDP (S, A, T, R, γ), where:

- **State Space:** The states $s \epsilon S$ are the cells in our grid world, given by the cell coordinates. The cell coordinates for our implementation start with $(0, 0)$ upto $(gridsize - 1)$. Each cell also has a state index associated with it.

- **Action Space:** We have limited our action space to four discrete actions : $\{RIGHT, UP, LEFT, DOWN\}$. There actions correspond to moving one cell in the given direction and is implemented by adding $\{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ to the coordinate of the state. That gives the next state that results from taking the particular action.

- **Transitions:** T is the set transition probabilities where $T(s, a, s')$ gives the probability of transitioning from state s to s' by taking action a. We assume that transitions in our grid world are deterministic. Additionally, we check if the resulting state s' is a neighbour of the current state s, as the only transitions allowed in our MDP are to the adjacent cells. We also check for edges and corners of the grid world.

- **Reward:** The reward function $R(s, a)$ describes how much reward is obtained by taking an action a in the state s. The rewards for different states in our $5X5$ grid world are given by Figure 1. For this implementation, the reward is primarily dependent on the state, irrespective of the action taken. Moreover, there are no negative rewards for taking steps in the environment.

- **Discount factor:** γ defines how much worth is a given reward one step into the future as compared to getting the same reward now. For our implementation, we chose $\gamma \leq 0.9$ for all the three grids.

B. Finding expert policy

In order to learn the rewards for a given environment, our agent would need to observe the optimal behaviour for that environment. The optimal behaviour can be described using a policy. A policy gives the action the agent should take when it is in state s in order to achieve the highest reward, or complete a given objective. We call this policy that displays the optimal behaviour the **expert policy**. In IRL, this expert behaviour is generally learned from a demonstrator acting in the environment. For example, for the driving scenario, this behaviour can be learned from the human driver.
In a real-world scenario, we would try to learn the reward function the demonstrator is trying to optimise. To simplify things for our implementation, we rely on a different technique to replicate the demonstrator. To calculate the optimal policy, we use an algorithm called **Value Iteration**. It gives the value estimate of each state, denoted by $V(s)$. We can calculate the best action from the given values for every state; the action that maximises the value of the given state. Value Iteration computes the optimal values for each state by iteratively improving upon the estimate of the state-values until the value converges to a given tolerance.

$$V(s, a) = max_a(R(s, a) + \sum_{s'} T(s'|s, a) * \gamma V(s'))$$

The value iteration equation above gives the optimal value for each state, taking into account the reward of the current state and the expectation of the future reward. We additionally keep track of the policy i.e. the best action at each state, to help generate trajectories. This is generated from the action that maximizes the value function at each state.

C. Generating Trajectories

The path taken by the agent in the environment is called a trajectory and is denoted by τ. A trajectory is a set of (s, a, s') tuples, which gives the state, action and next state at each point in the path. The **value iteration** algorithm stated above gives an optimal policy and is used to generate a collection of such trajectories, which are equivalent to the paths taken by an expert demonstrator in the environment.

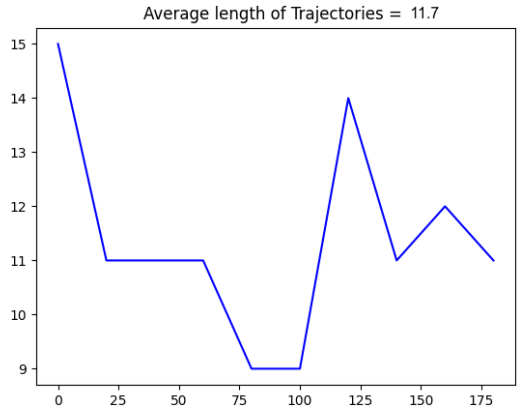


Fig. 2: Plot showing the average length of trajectories

D. Learning the reward structure

IRL takes into consideration an MDP which does not have a reward specification. This is denoted by $(S, A, T, \gamma) \backslash R$. The reward is assumed to be some function of the state values. We consider a vector of features ϕ, and say that there is some true reward $R^*(s) = w^*.\phi(s)$, where w^* denotes the weighing of the mentioned features. The reward, therefore, is a linear combination of the features. Consider an autonomous driving scenario, in which the agent trying to learn the human's reward function. The "features" mentioned could be certain properties of driving styles such as velocities and distances to other vehicles, distance to the lanes, and a preferred speed. A policy π is defined as a mapping from the states to the probability of taking the different actions in the given state. The value of a policy is defined as the expectation with respect to a state sequence starting from s_0 and picking actions according the policy. This can be written as:

$$E_{s_0}[V^\pi(s_0)] = E[\sum_{t=0}^{\infty} R(s_t)|\pi]$$

We already know that the reward is a dot product of the weights and the feature vector. We can then write the expected value as:

$$E_{s_0}[V^\pi(s_0)] = E[\sum_{t=0}^{\infty} \gamma^t w.\phi(s_t)|\pi]$$

$$E_{s_0}[V^\pi(s_0)] = w.E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$$

The expected value for the features discounted over time are called feature expectations. These can we written as:

$$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$$

And according to the definition above, we can now write the expected value as follows:

$$E_{s_0}[V^{\pi}(s_0)] = w.\mu(\pi)$$

The reward can now be learned from a set of demonstrations. We assume that these demonstrations are generated by the expert policy given by π_E and the corresponding reward is $R^* = W^{*T}\phi$. For this implementation, we can get the feature expectations for the expert given the demonstrations.

E. Maximum Entropy Formulation

One of the biggest problems with learning the rewards from this type of formulation is the multiplicity of reward function. Given expert policy π_E, there can be multiple reward functions for which the policy π_E is optimal. We tackle this problem by using Ziebart et al. [3] approach of Maximum Entropy. We make use of a maximum entropy distribution over the different paths. The particular trajectory which gives the maximum reward is exponentially more probable to be picked.
For this approach, we require the state visitation frequencies which represent the probability of being in the particular state s. We simply count all the possible paths to calculate the state visitation frequencies. This approach is certainly not feasible in large environments. But for our use, this works fine as our number of states does not go over 100.
The maximum entropy algorithm is given by:

Maximum Entropy Inverse Reinforcement Learning

1- Initialize: Feature matrix (ϕ) and Weights (w)
2- Calculate feature expectations: Averaging the features over states in given trajectories
3- $i = 0$ **to** $iterations$
a. $r = \phi^T.w$
b. Calculate state visitation frequencies (f_τ)
c. $gradient = feature_expectations - \phi^T.f_\tau$
d. Update weights: $w+ = \alpha * gradient$
4- Return: $\phi^T.w$

V. RESULTS

The algorithm is experimented on two versions of a $3X3$ grid and one version of a $5X5$ grid world. The $3X3$ grid world was used to run initial tests for the algorithm. The rewards are fairly simple in this set up, with just one positive reward for the top left cell, which is the terminal state.The grid world and results are shown in Figure 3.

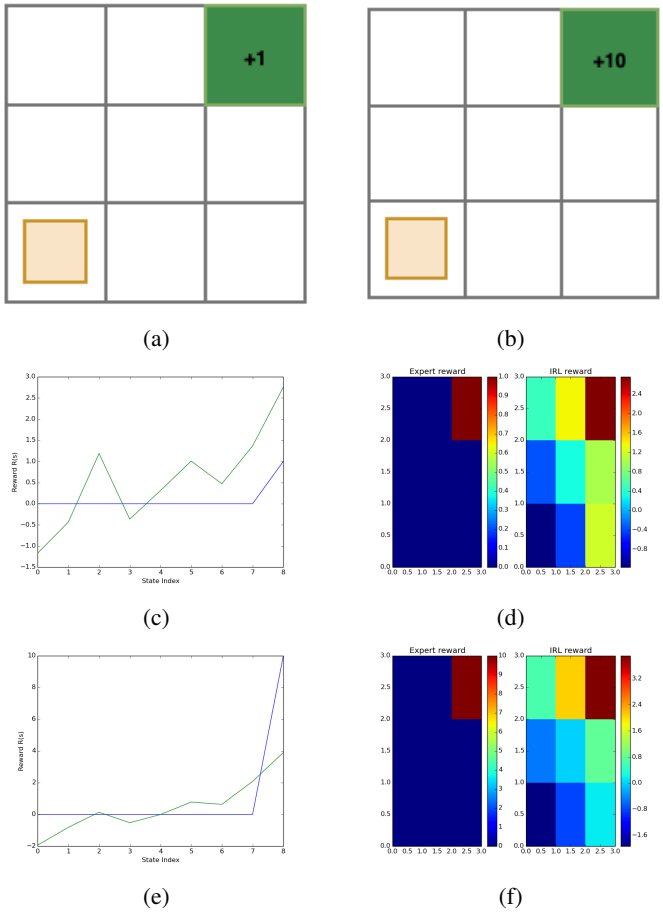
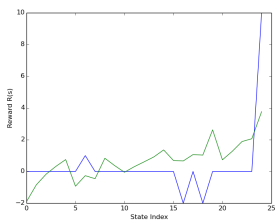


Fig. 3: Environment and results from applying the given IRL algorithm to a 3x3 grid world. (a) 3x3 grid world with $R = 1$ (b) 3x3 grid world with $R = 10$ (c) Comparison of the expert and retrieved rewards for $R = 1$ (d) Ground truth reward and IRL reward distribution for $R = 1$ (e) Comparison of the expert and retrieved rewards for $R = 10$ (f) Ground truth reward and IRL reward distribution for $R = 10$

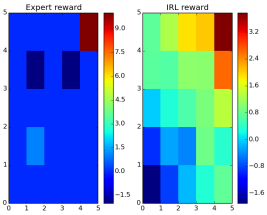
We ran another set of experiments on a 5x5 grid world MDP (Figure 1). We first fixed the number of iterations to 200 and tested for different lengths of the trajectories. We experimented with trajectory lengths of 20, 50 and 100. We notice that the recovered reward is not sensitive to the number of trajectories for less number of iterations such as 200. The plots for the same are given in Figure 4.

We increase the number of epochs for our next set of trials. We fix the number of trajectories to 50, this seemed to work for most of the cases. We notice that the recovered reward increases dramatically with increasing the number of epochs. We start with the previous results for 50 trajectories and 200 epochs. Then we increase the number of epochs to 500 and then 1000. We were able to retrieve the maximum reward in 1000 iterations. The plots are shown in Figure 5.

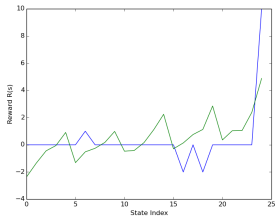
For the final set of experiments, we vary both the parameters in order to reach a middle point for the optimal number of trajectories and epochs. We find that generating 150 trajec-



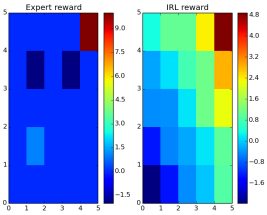
(a) Trajectories = 20



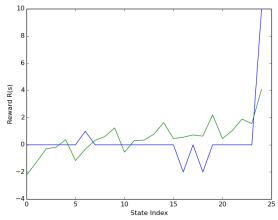
(b) Trajectories = 20



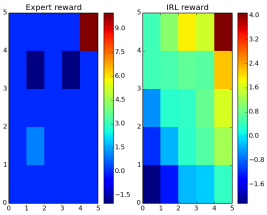
(c) Trajectories = 50



(d) Trajectories = 50

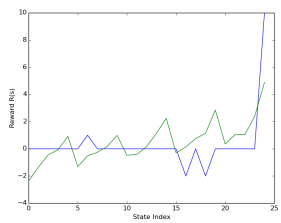


(e) Trajectories = 100

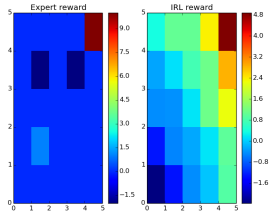


(f) Trajectories = 100

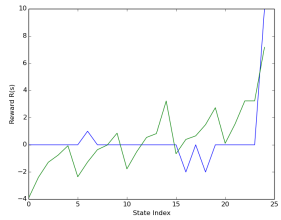
Fig. 4: Comparison plots and distribution of rewards for different number of trajectories with fixed epochs = 200



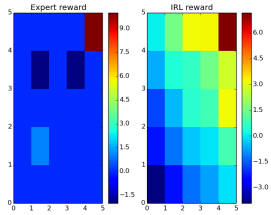
(a) Epochs = 200



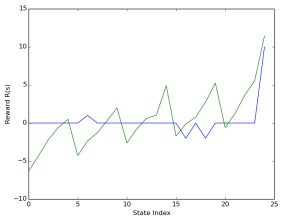
(b) Epochs = 200



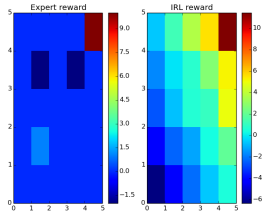
(c) Epochs = 500



(d) Epochs = 500



(e) Epochs = 1000



(f) Epochs = 1000

Fig. 5: Comparison plots and distribution of rewards for different number of epochs with fixed number of trajectories = 50

tories and running the algorithm for 750 iterations gives a middle ground for the parameters. The final plots and results are shown in Figure 6.

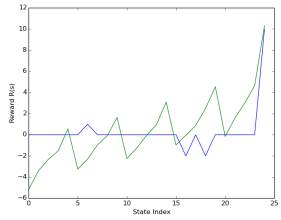
VI. DEMO & INTERFACE

The third level of this project was to create a web-based interface for the end-users to enter and generate expert trajectories which can be fed to the IRL algorithms to learn the reward function of the grid world. The user will have the option of either (a) adding a trajectory, (b) viewing all the recorded trajectories so far, or (c) downloading the recorded trajectories in a CSV file.

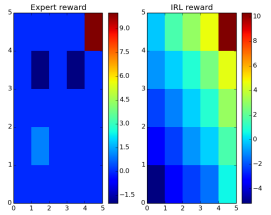
Github (https://github.com/souravchk25/dmu-interface/tree/master) contains the code and the demonstration video of the interface. The code in written in Django framework with HTML.

VII. CONCLUSION AND FUTURE WORK

We experimented with two variations of the grid world ($3X3$ and $5X5$). We varied the different parameters - Epochs, Number of trajectories, and plotted the obtained reward against the true rewards. We also studied the reward distribution across the grid. The experiments with the $3X3$ grid world showed a striking property of Inverse Reinforcement Learning algorithms. We saw that IRL algorithms only learn a structure



(a)



(b)

Fig. 6: Comparison plots and distribution of rewards for number of epochs = 750 with number of trajectories = 150

of the reward distribution. In both the $R = 1$ and $R = 10$ case, the maximum reward averaged around 2 and 4.8 respectively. The algorithm could retrieve a general positive reward for the terminal state but could not differentiate between the actual values (1 and 10).

Building upon this general intuition, we experimented with a slightly complicated grid world. The $5X5$ grid world had both positive and negative rewards distributed across the grid. We saw that the algorithm was not very sensitive to the number of trajectories with less number of epochs. This result is important as it is indicative of the amount of data

that might be required for extending this work to larger environments. Our final plots show that **150 trajectories with 750 iterations** provided a decent trade-off to retrieve the maximum reward of 10 from the environment.
Since the MDPs we used were smaller, we were able to run the tests for the different variations in lesser time. One trial with a $10X10$ grid world was taking around 6 minutes so we decided to downsize the grid. This approach will certainly be inefficient for a larger MDP, specifically the calculation of state visitation frequencies. We simply iterate over all states in the trajectory to get the frequencies which makes computation slower. Moreover, larger MDPs would also require more trajectories to learn the reward.

Future iterations of this project could improve upon our work by making use of a neural network approximation for the rewards. Then the problem can be scaled up by a large factor. This work will particularly be useful for real-world scenarios where learning the reward is difficult. In such cases, demonstrations given by an agent can be used to generate trajectories and learn the reward the agent is trying to optimise. This could be an interesting interaction and allow a user to understand what actions they take either improve or degrade the results derived through IRL based off of their demonstration. We could also use deep networks for both the expert policy and for learning the reward.

VIII. CONTRIBUTIONS AND RELEASE

All members participated in research for formulation approaches and designing the grid world environment for testing out the approaches.
Tuhina wrote the code for grid world and implemented the maxEnt IRL algorithm. She also ran the test cases and generated the results and plots for different trials. She contributed in writing the approach and results for the report.
Alexa wrote 100% of the Value Iteration code and tested the values generated by the expert policy thoroughly. She greatly contributed to writing the report, wording the preliminaries and approach for Value Iteration.
Sourav implemented the user interface in its entirety to generate some user demonstrations. He helped in troubleshooting and also contributed greatly to formulating the related work and writing the report in general.
The code for this implementation can be found on Github
The authors grant permission for this report to be posted publicly.

ACKNOWLEDGMENT

Thank you Professor Sunberg for a great class! We'd also like to thank Qi Heng Ho for all his help throughout the class!

REFERENCES

[1] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000.

[2] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[3] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, pages 1433–1438, 2008.

[4] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006.

[5] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[6] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In Marcelo H. Ang and Oussama Khatib, editors, *Experimental Robotics IX*, pages 363–372, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] Pieter Abbeel, Dmitri Dolgov, Andrew Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. pages 1083–1090, 09 2008.

[8] Andrey Rudenko, Luigi Palmieri, Michael Herman, Kris M Kitani, Dariu M Gavrila, and Kai O Arras. Human motion trajectory prediction: a survey. *The International Journal of Robotics Research*, 39(8):895–935, 2020.

[9] Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefèvre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. pages 1025–1028, 08 2011.

[10] Nathan Ratliff, J. Bagnell, and Martin Zinkevich. Maximum margin planning. pages 729–736, 06 2006.

[11] Nathan D. Ratliff, David Silver, and J. Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27:25–53, 2009.

[12] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. pages 2586–2591, 01 2007.

[13] Christos Dimitrakakis and Constantin Rothkopf. Bayesian multitask inverse reinforcement learning. 06 2011.