Conflict in the World of Inverse Reinforcement Learning

Investigating Inverse Reinforcement Learning with Conflicting Demonstrations

Petar Koev [1]

Supervisor(s): Luciano Cavalcante Siebert[1], **Antonio Mone**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Petar Koev
Final project course: CSE3000 Research Project
Thesis committee: Luciano Cavalcante Siebert, Antonio Mone, Wendelin Böhmer

Abstract

Inverse Reinforcement Learning (IRL) algorithms are closely related to Reinforcement Learning (RL) but instead try to model the reward function from a given set of expert demonstrations. In IRL, many algorithms have been proposed, but most assume consistent demonstrations. Consistency is the assumption that all demonstrations follow the same underlying reward function and near-optimal policy, without any contradictions. This, however, is not always the case. This study investigates the effect of conflicting demonstrations on IRL algorithms. For our experiments, the Lunar Lander environment and a grid-world environment are used in combination with a state-of-the-art IRL algorithm. To obtain the expert demonstrations, agents were trained using RL algorithms with explicit differences in the reward functions to achieve optimal policy. Then these demonstrations were used in training IRL in a variety of different configurations of hyperparameters. Our results show that IRL algorithms can be trained using demonstrations with varying levels of conflict. In conclusion, we demonstrate that IRL can learn even when provided with a set of conflicting demonstrations.

1 Introduction

In recent years, Inverse Reinforcement Learning (IRL) algorithms have been used to learn the underlying reward function of complex tasks [1, 2]. This is achieved given a set of expert demonstrations (human or robot) that an autonomous agent uses to mimic or even learn the intentions behind a human-performed task. This approach has significantly advanced the development of autonomous vehicles [3], personalized medical treatment plans [4] and many other domains, since it eliminates the need for explicit specifications of the reward functions, that would be too strenuous to write out or even impossible for such complex tasks. However, just like in the real world, expert demonstrations are not always consistent, which leads to challenges in efficiency and effectiveness when training and evaluating such algorithms. Understanding how IRL acts when given conflicting demonstrations is crucial in the advancement of autonomous agents.

To understand the inner workings of IRL, first, we need to look into regular Reinforcement Learning (RL). The main goal of an RL agent is to continuously interact with the environment, learning a policy through trial and error. IRL, in contrast, works by inferring a reward function based on a set of given demonstrations. This allows for the use of IRL in complex scenarios where we cannot easily define an explicit reward function.

Prior research into IRL has methodically documented the strengths and limitations of the capabilities of an autonomous agent to learn the true reward function. Many different algorithms have been invented and honed to improve the effectiveness of the task [1, 2, 5], but most of them assume compatible expert demonstrations. Some studies have investigated the effect of noise in the data of the demonstrations [2]. Conflicting demonstrations, however, are derived from differing or even malicious reward functions, unlike sub-optimal demonstrations where the agent is still trained on a single reward function. In conflicting demonstrations, the observed behaviours are based on multiple reward functions, leading to inconsistencies. While these demonstrations could still be optimal, malicious demonstrations deliberately promote detrimental behaviours leading the agent to learn harmful policies. The specific effects of such conflicting demonstrations remain relatively unexplored.

This paper aims to fill in this gap and explore how different degrees of conflict in the demonstrations affect the IRL's ability to learn. The main research topic is "***To what extent can IRL learn rewards from conflicting demonstrations?***". The paper will look into the following sub-questions:

- *How does the degree of conflict between demonstrations affect IRL's ability to learn the reward function?*
- *Does the ratio of conflicting demonstrations influence IRL's ability to learn the reward?*
- *Does the complexity of the task influence IRL's ability to handle conflicting demonstrations?*
- *How do malicious expert demonstrations affect IRL?*

The main contributions include a thorough analysis of the impact of conflicting demonstrations on IRL, insight into what factors affect the ability of the algorithms to learn from conflicting demonstrations and how malicious demonstrations affect the learning abilities of IRL.

The paper is structured as follows: Section 2 provides the background information for Inverse Reinforcement Learning and related algorithms. Section 3 explains the methodology used. Then, Section 4 outlines the environments used when training the algorithms. Section 5 investigates the results of the experimentation. In Section 6, the results are discussed and we delve into which factors helped the ability of IRL algorithms to learn the reward function. Section 7 summarizes the findings of this paper and suggests areas to explore in future studies. Finally, Section 8 touches upon ethical considerations made when conducting this study.

2 Background

To understand the techniques used in this paper, it is crucial to know how the used algorithms work. In this section, we provide a brief explanation of the algorithms used, namely Proximal Policy Optimization (PPO) and Adversarial Inverse Reinforcement Learning (AIRL).

2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [6] is a policy-based reinforcement learning algorithm that directly optimizes the policy that the agent uses when choosing actions instead of modelling a reward function. We use this technique to generate expert demonstrations because collecting them from humans can be expensive and complex. By using synthetic demonstrations, we ensure consistency in our data, allowing for more reproducible results of our study. The main strengths of PPO are its simplicity and efficiency, allowing it to quickly generate optimal demonstrations with low computational needs. This is achieved through its update function that limits the degree of change each generation can bring, ensuring no abrupt changes are made.

To understand PPO, we need to look into the basics of RL, namely the Markov Decision Process (MDP) [7]. In RL, an agent interacts with an environment described by states (s), actions (a), and rewards (r). The agent's objective is to learn a policy (π) that maximizes the expected cumulative reward. An MDP is defined by a tuple (S, A, P, R, γ), where S is the set of states, A is the set of actions, P is the state transition probability, R is the reward function, and γ is the discount factor.

Many different algorithms were created to optimize this process and PPO is an improvement over the Trust Region Policy Optimization (TRPO) [8] algorithm. This is achieved by simplifying the optimization process while keeping stable performance.

Mathematically PPO is presented as:

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

with

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2}$$

In these equations $\mathbb{E}_t$ denotes the expectation, $r_t(\theta)$ is the probability ratio between the new policy $\pi_\theta(a_t|s_t)$ and the old policy $\pi_{\theta_{\text{old}}}(a_t|s_t)$. $\hat{A}_t$ is the estimated advantage at time step t, representing how much better an action is compared to the agent's average action, and ϵ is a hyperparameter that controls the clipping range.

2.2 Adversarial Inverse Reinforcement Learning

Adversarial Inverse Reinforcement Learning (AIRL) is the main algorithm that this paper focuses on. This is due to its great performance in complex environments and its notable generalization to new environments [1].

It follows the same methodology as Generative Adversarial Networks (GAN) [9], but builds on top of it by directly modelling the reward function. Similarly to GANs, AIRL has two networks - a learner and a discriminator. The learner's role is to model the agent's policy, while the discriminator models the reward function of the environment. In this paper, PPO is used to optimize the learner's policy because of its good balance between performance and training time needed.

The discriminator uses the provided expert trajectories and differentiates them from the learner's trajectories. Unlike a traditional GAN discriminator, AIRL's discriminator further updates the learner's policy based on this distinction. This iterative process involves generating new trajectories with the updated learner's policy and honing the discriminator's ability to distinguish between expert and learner trajectories.

The goal of AIRL is to retrieve the reward function that most accurately explains the demonstrations passed, making this method flexible when learning rewards and more generalizable when applied to unseen environments.

Algorithm 1 shows the whole process in pseudo-code, while a more formal definition of the objective of the reward function modelling is equations 3 and 4.

$$D_{\theta,\varphi}(s,a,s') = \frac{\exp\{f_{\theta,\varphi}(s,a,s')\}}{\exp\{f_{\theta,\varphi}(s,a,s')\} + \pi(a|s)} \tag{3}$$

where

$$f_{\theta,\varphi}(s,a,s') = g_\theta(s,a) + \gamma h_\varphi(s') - h_\varphi(s) \tag{4}$$

In these equations, $D_{\theta,\varphi}(s,a,s')$ is the output of the discriminator, representing the probability that the state, action, next state triplet (s,a,s') comes from the expert. $f_{\theta,\varphi}(s,a,s')$ is the function that is used when scoring the triplets and is composed of a learned reward function $g_\theta(s,a)$ and a value function difference $\gamma h_\varphi(s') - h_\varphi(s)$. θ and φ are parameters of the reward and value functions, and $\pi(a|s)$ is the policy's probability of taking action a in state s. Then, the reward function used to update the policy is derived from the discriminator's output as

$$r_{\theta,\varphi}(s,a,s') = \log D_{\theta,\varphi}(s,a,s') - \log(1 - D_{\theta,\varphi}(s,a,s')) \tag{5}$$

Algorithm 1 Adversarial Inverse Reinforcement Learning

1: Obtain expert trajectories τ_E^i
2: Initialize policy π and discriminator $D_{\theta,\varphi}$
3: **for** step t in $\{1, \ldots, N\}$ **do**
4: Collect trajectories $\tau_i = (s_0, a_0, \ldots, s_T, a_T)$ by executing π
5: Train $D_{\theta,\varphi}$ via binary logistic regression to classify expert data τ_E^i from samples τ_i
6: Update reward $r_{\theta,\varphi}(s,a,s') \leftarrow \log D_{\theta,\varphi}(s,a,s') - \log(1 - D_{\theta,\varphi}(s,a,s'))$
7: Update π with respect to $r_{\theta,\varphi}$ using any policy optimization method
8: **end for**

3 Methodology

In the following section, we look into the methodology used when conducting this study. Moreover, we look at the generation of expert trajectories used when training the AIRL algorithm, our definition of conflict in the trajectories,

and the notion of malice.

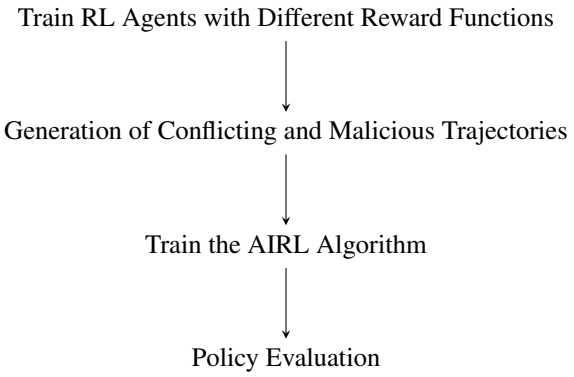

Figure 1: Methodology overview.

As seen in Figure 1, we begin by training agents using RL with various reward functions, generating conflicting and malicious policies. Following this, we generate trajectories based on the agents' policies and pass these trajectories to the AIRL algorithm. Once the algorithm is trained, we evaluate its new policy by comparing it against our baseline to assess its performance.

3.1 Expert Trajectories

In order to train AIRL, we need to feed it expert demonstrations. To have good results, these demonstrations need to be efficient enough to complete the given task. This does not mean that the agents must perform the optimal actions in each state though, just that the overall policy achieves above a predefined score when evaluated, making it considered a solution.

To achieve this, we used PPO on three different environments and we trained until convergence to optimality. Because the environments range in complexity, the agents needed various amounts of training, but this will be elaborated on further in Section 4.

3.2 Conflict

The core research topic of this paper is about conflict in the expert demonstrations. This means that there should be a clear definition of conflict. Although in recent years a lot of literature concerns itself with solving multi-objective environments [10, 11], there is a lack of investigation into how multi-intentional demonstrations affect the IRL's ability to learn a reward function.

Thus, there is a lack of a clear definition of what it means for two trajectories to be conflicting. For this study, we use a time-efficient approach where we compare two trajectories based on the reward functions they were trained on [12]. This is a conceptually oriented approach that looks at the goal of the agent rather than the exact actions taken at each state. Mathematically, we express this in Equation 6.

$$R_1(s, a, s') \neq R_2(s, a, s') \tag{6}$$

One drawback of this approach is its lack of a precise metric to quantify the differences between two conflicting trajectories. Furthermore, it allows trajectories to have the same steps for a part of their path while still being considered conflicting. This will be very important when defining the degree of conflict in each environment in Section 4.

Another approach would be to compare each state of the trajectories and see what actions were taken [13]. This is a more statistical method, which gives exact results. Then, the degree of conflict could be defined by the number of different actions taken at each state. This, however, poses a lot of challenges when taken to a continuous environment. In continuous environments, the state space is infinite [14] and an agent's actions vary vastly, making it impossible to match state-action pairs of two trajectories precisely. Furthermore, minor variations in states could amount to big differences in the action taken, making this method very sensitive to noise in the data.
One approach to fix these issues is to use similarity ranking algorithms [15] between states that allow for small differences. This means that precise thresholds that will vary based on the specific environment need to be set. Due to this issue, we decided not to use this approach in our paper.

Instead, by opting for reward function comparison, we aim to provide a clear conceptual definition of conflict. This method not only simplifies the reward comparison but also aligns better with the main goal of this study.

3.3 Malice

To further push the idea of conflict, in this study, we look deeper into a special subset of conflicting demonstrations, namely malicious demonstrations. These are demonstrations that intentionally try to harm the learning algorithm. Such demonstrations go for as many penalties as possible in order to trick the learner agent into harmful or sub-optimal behaviour. Since IRL algorithms depend solely on expert demonstrations, the effect that the malicious demonstration could have is detrimental. By incorporating such flawed demonstrations into the training process, we aim to show how much performance is lost when training AIRL and what could be an acceptable amount of malicious demonstrations.

For the sake of simplicity, we define malicious demonstrations in a straightforward manner: simply inverting the reward function of the environment as shown in Equation 7.

$$R_{\text{mal}}(s, a, s') = -R(s, a, s') \tag{7}$$

Other methods, such as using reward poisoning attacks performed during the training phase, have been explored [16], but we consider them outside the scope of this study due to their complexity.

4 Experimental Setup

In this section, we are going to take a look at the three environments used in this study. A short explanation about how each environment works will be given as well as the reasoning behind each choice. When deciding on which environments to pick, two main criteria were used: the ability to generate conflicting demonstrations and computational expenses. Each environment constitutes a different level of conflict while still being simple enough to be trained locally. All environments are provided by Gymnasium [17] and MO-Gymnasium [18].

4.1 Lunar Lander

The first environment we look at is the LunarLander-v2 [19] environment shown in Figure 2. We chose this environment as it is a good starting point for our study. It is simple enough to train agents relatively quickly, while still being complex enough to have conflicting demonstrations. Other environments, such as Cartpole-v1, were also taken into account, but there were difficulties in generating conflicting demonstrations since the expert policy, trained on different reward functions, consistently performed the same actions.

Lunar Lander is a simple environment where an agent is trying to land on a platform in the middle of the environment. The agent can utilize their main booster to slow down and their side boosters to rotate. At the beginning of each run, a random force is applied to the agent in an arbitrary direction and the agent's goal is to land safely on the platform in the middle.

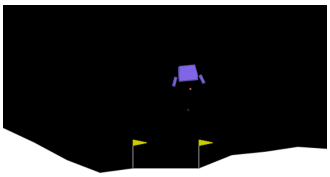

Figure 2: Lunar Lander environment showing an agent trying to land on the landing platform.

While training our expert agents, we extended the default environment's reward function by adjusting the rewards at each time step. The new reward functions were designed with specific objectives: one for approaching the platform from the left, as described by Equation 8, another for approaching from the right, as detailed in Equation 9, and a third for maintaining a central position as much as possible, described by Equation 10.

$$r = r - 10 \cdot (x_{\text{position}} - 0.5) \tag{8}$$

$$r = r + 10 \cdot (x_{\text{position}} - 0.5) \tag{9}$$

$$r = r - 10 \cdot |x_{\text{position}}| \tag{10}$$

This constitutes our first level of conflict. All of the agents are trying to achieve the same task but take different approaches since they were trained on different reward functions. Following our definition of conflict from Section 3.2, we assume that there is a degree of conflict when comparing a left-approaching agent with a right-approaching one. Furthermore, we can logically conclude that comparing left- and right-approaching agents has a higher degree of conflict, than when comparing left- or right-, and centre-approaching ones.

When generating the malicious demonstrations for both the lunar lander environment and the multi-objective one, we used the approach described in section 3.3. We then trained an agent using the environments' inverted reward functions and trained agents using PPO until convergence.

4.2 Multi-objective Lunar Lander

While the method of taking different approaches was able to yield conflicting trajectories, to further increase the degree of conflict, we chose to explore the mo-lunar-lander-v2 [20]. This step was taken to make the difference in the reward functions even more explicit.

The multi-objective lunar lander environment builds on top of the base one, with the main difference that now firing the main engine or the side engines gives a penalty of -1. This enables us to clearly define two distinct reward functions: one that minimizes the penalty of the main engine, and another that minimizes the penalties from the side engines.

This, in turn, takes the degree of conflict one step further, since now we optimize the policies for explicitly different reward functions. Despite that, there is still some overlap in the trajectories since the agent cannot only use their side boosters to slow down enough to land.

4.3 Resource Gathering

Finally, we look at the resource-gathering-v0 [21] environment, shown in Figure 3. This choice was made since it had clearly defined objectives that an agent could be optimized for, without an overlap in the trajectories. Furthermore, it has enemies which will be useful when generating malicious demonstrations.

The resource gathering environment is a grid-world environment where the agent's goal is to collect gold and gems and return home while avoiding the dragons, which act as enemies. The gold and the gem contribute with a +1 reward upon successfully returning home while being killed by the dragons gives a -1 reward. Dragons have a 10% chance of killing the agent when it steps into their square.

Figure 3: Resource Gathering environment showcasing the home position, agent, gold, gem and two enemies.

This environment will simulate the highest degree of conflict that this paper will explore. The agents were trained with reward functions focusing only on one of the two collectable objects. Because of their locations on the map, the trajectories have as little overlap as possible.

A similar approach to the lunar lander environments was taken when generating the malicious demonstrations. The environment's reward function was inverted as described in Section 3.3. This causes the agent to go up to the top-right dragon and continue moving upwards. Due to the environment's boundary constraints, the agent stays in the dragon's square and continues trying to move upwards until it eventually dies.

5 Results

In this section, we will systematically explore the findings based on training the AIRL algorithm on each of the three environments and interpret the implications of the achieved results.

The evaluation method for all environments was based on the average result of 100 episodes using the evaluation method provided by stablebaselines3 [22]. This evaluation was performed based on the standard environment's reward function. Furthermore, we record the mean training reward and standard deviation when training the algorithm every 100,000 timesteps.

5.1 Lunar Lander

Starting with the LunarLander-v2 environment, we see that the smallest degree of conflict didn't affect the learning performance of the AIRL algorithm much. Here we only look at agents trained with left- and right-approaching demonstrations. All agents achieved comparable results, as shown in Figure 4.

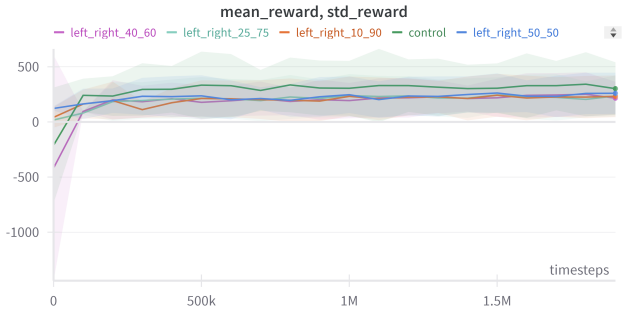

Figure 4: Graph showcasing the learning of AIRL agents in the LunarLander-v2 environment.

In Table 1, we see that all the agents performed similarly to the control version when evaluated, except for the *left_right_40_60* agent. The *left_right_50_50* agent even manages to outperform our control agent.

Run	Final Reward	Final Std
control	263.9	57.3
left_right_50_50	274.3	52.2
left_right_40_60	179.8	112.6
left_right_25_75	214.5	94.9
left_right_10_90	225.4	88.2

Table 1: Comparison of final mean reward and final mean standard deviation for the LunarLander-v2 environment.

When it comes to malicious demonstrations, in Figure 5, we see a trend that will continue in the other environments. The control agent and the agent that has only 10% of malicious demonstrations manage to learn well, while the 25% and 50% agents cannot.

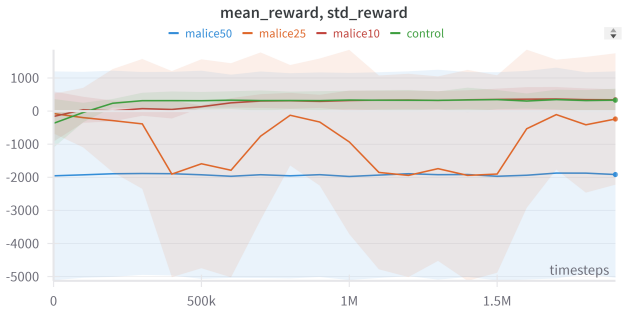

Figure 5: Graph showcasing the training performance of AIRL with varying ratios of malicious demonstrations

5.2 Multi-objective Lunar Lander

Next, we explore the mo-lunarlander-v2. We see that increasing the degree of conflict poses challenges to the learning of the algorithm. In Figure 6, we see that the standard deviation is higher in the beginning stages of the training phase but all agents converge to the same result.

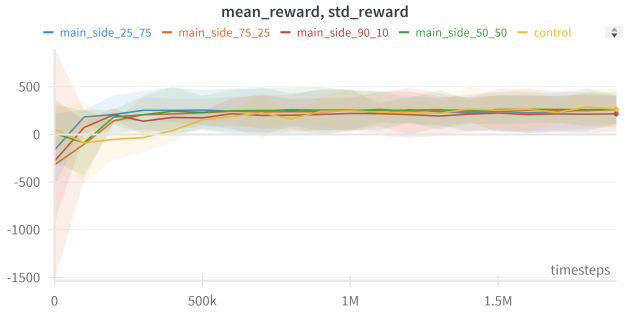

Figure 6: Graph showcasing the learning of AIRL being passed various ratios of main engine and side booster demonstrations.

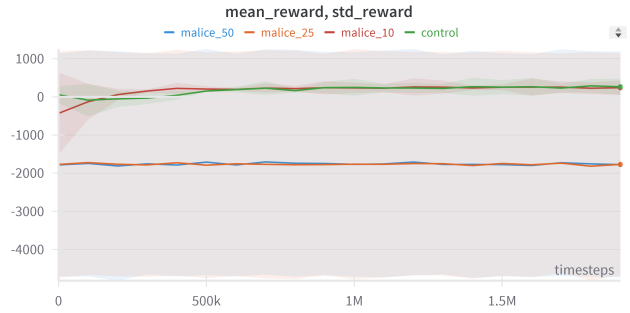

Figure 7: Graph showcasing the learning of AIRL being different ratios of control and malicious demonstrations.

Again in Table 2, we see that the agents perform similarly to the control agent. An interesting observation is that as the number of main engine demonstrations increases, the final reward also increases and the standard deviation goes down.

Run	Final Reward	Final Std
control	177.4	89.8
main_side_90_10	192.0	25.7
main_side_75_25	180.3	74.8
main_side_50_50	168.7	112.1
main_side_25_75	186.7	63.0

Table 2: Comparison of final mean reward and final mean standard deviation for the mo-lunar-lander-v2 environment.

We only look in-depth at the multi-objective lunar lander behaviour of the learned policies. Table 3 shows that AIRL tries to average out the two conflicting reward functions. We can see that the number of demonstrations passed highly influences the number of engine usage.

# Main Engine Use	# Side Engines Use	Run Name
70	21	control
100	14	main_side_90_10
75	48	main_side_75_25
86	54	main_side_50_50
70	38	main_side_25_75

Table 3: Engine usage statistics for different runs.

Figure 7 shows us that, similarly to the lunar lander environment, the agent with only 10% of malicious demonstrations manages to converge to the results of the control agent. Again, the 25% and 50% agents don't learn the reward function.

5.3 Resource Gathering

Finally, we look at the environment with the highest degree of conflict. Here, we achieved results that surprised us. We expected that the behaviour of the agents would be similar to the agents from the previous two environments - averaging out the behaviours of the demonstrations. However, as seen in Figure 8, none of the agents trained with conflicting demonstrations managed to beat the score of the control agent.

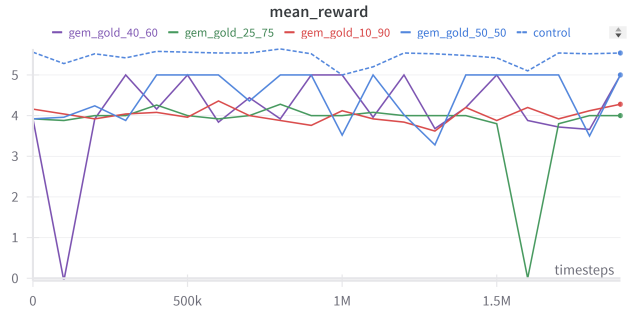

Figure 8: Graph showcasing the mean reward during the training phase.

In Table 4, we see that some agents manage to achieve a 1.0 final reward and 0 standard deviation. This means that they have found a safe path where they don't go through any enemies but only pick up one of the objectives. We see that none of the conflicting agents pick up both objectives.

Run	Final Reward	Final Std
control	1.8	0.8
gem_gold_50_50	1.0	0.0
gem_gold_40_60	1.0	0.0
gem_gold_25_75	1.0	0.0
gem_gold_10_90	0.8	0.5

Table 4: Comparison of final mean reward and final mean standard deviation for the resource-gathering-v1 environment.

To understand this behaviour, we plotted out the reward function of the *gem_gold_50_50* agent and added or subtracted

the rewards that the reward function predicts for 100 trajectories. The result can be seen in Figure 9. The reward function prefers the gold objective, giving it a positive reward, and it gives a negative reward for the gem.

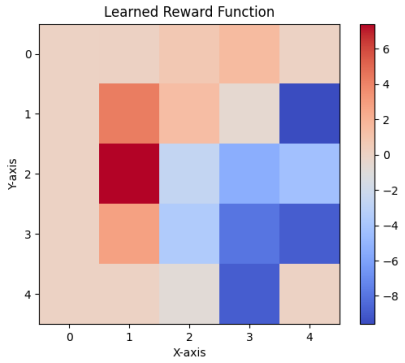

Figure 9: Plot of the rewards predicted by the reward net of the gem_gold_50_50 agent.

Similarly to the lunar lander environments, we see in Figure 10 that runs with over 25% of malicious demonstrations cannot learn. In the case of 10%, it achieves the same results as the control version.

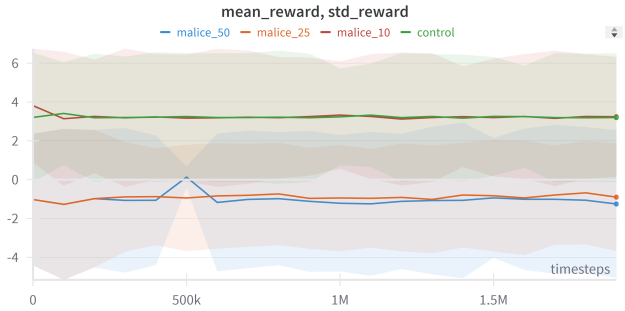

Figure 10: Graph showcasing the learning of AIRL being passed different ratios of malicious demonstrations in the resource-gathering-v1 environment.

6 Discussion

Based on the results of Section 5, there are still some problems to be interpreted and studied further. We saw that in Table 1, one of the agents did not perform as well as the control agent. This could be due to the environment's seed, as all agents underwent only one training run.

Furthermore, the lunar lander environments give a reward based on the time taken to land. Since in our observations AIRL tries to average out the reward functions of the demonstrations, this could potentially also explain the lower score of the *left_right_25_75* and *left_right_10_90* agents, since it takes them more time to land on the right side of the landing pad when compared to going straight down to the centre of the landing pad.

Nonetheless, having similar training metrics and similar evaluation metrics causes us to believe that for smaller levels of conflict, IRL algorithms can successfully learn the reward function.

When it comes to the multi-objective lunar lander, a behaviour that we observed is that the agent cannot slow down enough to land while only using the side boosters. This explains the relatively high usage of the main engine for the *main_side_25_75* agent. Moreover, it explains the results that we obtained in Table 2. By increasing the number of main engine demonstrations, the agent can learn to slow down fast enough, thus the standard deviation gets lower.

Overall, we can see that when an environment has a single reward function, AIRL can learn even when given conflicting demonstrations. However, with the increase of conflict, the distance between the trajectories grows larger and it is harder to stay in a good direction. This was especially showcased when malicious demonstrations were introduced.

As for environments that have more than one reward function, we saw that AIRL trained with conflicting demonstrations chooses only one of them. This is showcased in Figure 9, where the discriminator gives a positive reward for the gold and a negative for the gem. This indicates that the discriminator considers the trajectories leading to the gold to be consistent with the expert demonstrations, while the trajectories leading to the gem are not.

Further study is required to understand more complex environments. Based on our results with malicious demonstrations, we believe that as the complexity of the environments increases, fewer malicious demonstrations will be needed to harm the learning of AIRL. However, additional research is necessary to confirm this.

7 Conclusions and Future Work

In this paper, we explored how conflicting demonstrations affect IRL's ability to learn the true reward function. We found that even with conflicting demonstrations we successfully trained optimal policies in the three environments that were explored.

We observed that as the degree of conflict intensified, it became more challenging for the algorithm to learn. Furthermore, malicious demonstrations had a great impact on performance even when they constituted only a small portion of the demonstrations.

Future work could explore applying conflicting demonstrations to environments with higher levels of complexity. Furthermore, a hybrid combination of the two definitions of conflict from section 3.2 could be explored to see if defining conflict more strictly makes the algorithm perform better. Another interesting topic that would be highly

impactful is whether we can protect the IRL algorithms from malicious demonstrations. This could be done by either detecting the malicious trajectories and removing them before training starts, or by pruning trajectories that have a negative impact during learning.

8 Responsible Research

Although this paper doesn't use human-generated data, we find it important to mention the measures taken to ensure that ethically sound and reproducible research was done.

8.1 Reproducibility

The expert demonstrations were exclusively trained by RL models provided by the imitation library [23]. All environments used were taken from OpenAI's Gymnasium and MO-Gymnasium. This means that everyone can easily reproduce all of the work provided in this paper.

Furthermore, all required code to train the algorithms locally is provided. Additionally, the weights zip files of the already trained models that were used in the generation of the results of this paper are also given.

8.2 Ethical Considerations

It is important to stress that all of the malicious demonstration work was done locally in a controlled environment. Even though we show that malicious demonstrations can harm the performance of IRL algorithms we strongly condemn using this information to cause harm to any real-world organisations or individuals.

We hope that any organisation that sees this research and uses IRL algorithms in their operational works will consider the effects of contradictory demonstrations, malicious or not, and employ measures to ensure their algorithms' safety and ethical soundness.

References

[1] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *CoRR*, vol. abs/1710.11248, 2017. [Online]. Available: http://arxiv.org/abs/1710.11248

[2] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, M. M. Veloso, Ed., 2007, pp. 2586–2591. [Online]. Available: http://ijcai.org/Proceedings/07/Papers/416.pdf

[3] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," *Robotics and Autonomous Systems*, vol. 114, pp. 1–18, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889018302021

[4] C. Shen, Y. Gonzalez, P. Klages, N. Qin, H. Jung, L. Chen, D. Nguyen, S. B. Jiang, and X. Jia, "Intelligent inverse treatment planning via deep reinforcement learning, a proof-of-principle study in high dose-rate brachytherapy for cervical cancer," *Physics in Medicine Biology*, vol. 64, no. 11, p. 115013, may 2019. [Online]. Available: https://dx.doi.org/10.1088/1361-6560/ab18bf

[5] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, "Modeling interaction via the principle of maximum causal entropy," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 1255–1262. [Online]. Available: https://icml.cc/Conferences/2010/papers/28.pdf

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] M. L. Puterman, "Chapter 8 markov decision processes," in *Stochastic Models*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1990, vol. 2, pp. 331–434. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801720

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, p. 139–144, oct 2020. [Online]. Available: https://doi.org/10.1145/3422622

[10] A. Bighashdel, P. Meletis, P. Jancura, and G. Dubbelman, "Deep adaptive multi-intention inverse reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds., vol. 12975. Springer, 2021, pp. 206–221. [Online]. Available: https://doi.org/10.1007/978-3-030-86486-6_13

[11] J. Choi and K. Kim, "Nonparametric bayesian inverse reinforcement learning for multiple reward functions," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, P. L. Bartlett, F. C. N. Pereira,

C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 314–322. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/140f6969d5213fd0ece03148e62e461e-Abstract.html

[12] S. Balakrishnan, Q. P. Nguyen, B. K. H. Low, and H. Soh, "Efficient exploration of reward functions in inverse reinforcement learning via bayesian optimization," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4187–4198. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/2bba9f4124283edd644799e0cecd45ca-Paper.pdf

[13] K. Toohey and M. Duckham, "Trajectory similarity measures," *SIGSPATIAL Special*, vol. 7, no. 1, p. 43–50, may 2015. [Online]. Available: https://doi.org/10.1145/2782759.2782767

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[15] Z. Wang, C. Long, G. Cong, and Y. Liu, "Efficient and effective similar subtrajectory search with deep reinforcement learning," *CoRR*, vol. abs/2003.02542, 2020. [Online]. Available: https://arxiv.org/abs/2003.02542

[16] K. Cai, X. Zhu, and Z. Hu, "Reward poisoning attacks in deep reinforcement learning based on exploration strategies," *Neurocomputing*, vol. 553, p. 126578, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223007014

[17] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[18] F. Felten, L. N. Alegre, A. Nowé, A. L. C. Bazzan, E. Talbi, G. Danoy, and B. C. da Silva, "A toolkit for reliable benchmarking and research in multi-objective reinforcement learning," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/4aa8891583f07ae200ba07843954caeb-Abstract-Datasets_and_Benchmarks.html

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[20] W. Hung, B. K. Huang, P.-C. Hsieh, and X. Liu, "Q-pensieve: Boosting sample efficiency of multi-objective RL through memory sharing of q-snapshots," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=AwWaBXLIJE

[21] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 41–47. [Online]. Available: https://doi.org/10.1145/1390156.1390162

[22] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[23] A. Gleave, M. Taufeeque, J. Rocamonde, E. Jenner, S. H. Wang, S. Toyer, M. Ernestus, N. Belrose, S. Emmons, and S. Russell, "imitation: Clean imitation learning implementations," arXiv:2211.11972v1 [cs.LG], 2022. [Online]. Available: https://arxiv.org/abs/2211.11972