AN ADVERSARIAL INVERSE REINFORCEMENT LEARNING APPROACH TO AN INDUSTRIAL ROBOTIC PICK-AND-PLACE

by

JACOB BOWDEN

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTERS OF SCIENCE IN MECHATRONICS AND ROBOTICS ENGINEERING

2026

Oakland University
Rochester, Michigan

Thesis Advisory Committee:

Dr. Jun Chen, Ph.D., Chair
Dr. Diego DAntonio, Ph.D.
Dr. Khalid Mirza, Ph.D.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Dr. Jun Chen, Associate Professor at Oakland University, for his guidance, encouragement, and thoughtful feedback throughout this work. His direction and support were instrumental in shaping the technical approach and maintaining academic rigor from initial concept to final manuscript.

I would also like to thank the faculty and staff at Oakland University for providing the resources and learning environment that supported my graduate studies and this research.

Finally, I am grateful to my family and friends for their continued encouragement and support throughout my time at Oakland University.

JACOB BOWDEN

ABSTRACT

AN ADVERSARIAL INVERSE REINFORCEMENT LEARNING APPROACH TO AN INDUSTRIAL ROBOTIC PICK-AND-PLACE

by

Jacob Bowden

Adviser: Dr. Jun Chen, Ph.D.

Industrial automation cells are often engineered to be highly repeatable, yet many pallet-handling tasks remain over-constrained because key sources of variability are difficult to model explicitly. In production environments, pallets may remain in service for years and become worn, chipped, warped, or cracked. These changes introduce unpredictable contact behavior during grasping and placement through variations in friction, compliance, and seating. While machine vision is commonly used to estimate pallet pose and compensate for offsets, vision-based solutions can be sensitive to lighting and surface appearance and can increase cycle time due to image capture and processing.

This thesis investigates an alternative approach that reduces reliance on non-value-added sensing by learning robust pick-and-place behavior from demonstrations. A physics-based palletizing work cell is developed in simulation, and expert trajectories are generated using a traditional state-machine strategy. Adversarial Inverse Reinforcement Learning (AIRL) is then used to learn a reward function from these demonstrations using a discriminator/reward network coupled with a policy generator. The AIRL stage demonstrates the ability to learn a reward function from demonstrated trajectories. A second stage of learning is applied, in which AIRL's learned reward and initial policy are refined and added too with forward reinforcement learning using additional rewards to improve final state action and robustness under pallet worn pallets.

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RL Reinforcement Learning

IRL Inverse Reinforcement Learning

BC Behavior Cloning

GAIL Generative Adversarial Imitation Learning

AIRL Adversarial Inverse Reinforcement Learning

MDP Markov Decision Process

POMDP Partially Observable Markov Decision Process

GAN Generative Adversarial Network

MPC Model Predictive Control

PnP Pick-and-Place

EE End Effector

EV Explained Variance

CHAPTER ONE

INTRODUCTION

1.1 Motivations

Industrial automation cells are often engineered to be highly repeatable, yet many tasks are still avoided or heavily over-constrained because key variables are difficult to model explicitly. In pallet handling and stacking operations, for example, the effective starting pose of a pallet can drift by millimeters due to upstream variation, and the contact conditions at placement can change over time as pallet surfaces wear. These sources of uncertainty—small pose offsets, varying surface finish, and contact dynamics—make it challenging to design reward functions or deterministic logic that reliably achieves millimeter-level placement without extensive tuning, sensing, or fixturing. This thesis explores an alternative to the traditional deterministic approach

1.2 Problem Formulation

1.2.1 Robotic Palletizing using AIRL

Industrial robotic palletizing cells are typically engineered to a high degree of precision, but many production environments rely on pallets that remain in service for years. Over time, pallets can become worn, chipped, warped, and cracked, making their effective dimensions and surface conditions difficult to predict. These changes alter contact behavior during grasping and placement, through inconsistent friction, compliance, and seating, creating small but consequential deviations that can accumulate into misalignment, poor seating, or failed placements when millimeter-level accuracy is required.

A common approach to managing variability is to add sensing, most often machine vision, to estimate the location of the pallet and correct for offsets introduced by staging variation or motion during pick-up and placement. Vision can reduce some geometric

uncertainty, it introduces practical limitations in industrial settings. Lighting variation, glare, and color variations can reduce reliability. The biggest issue is the time required to capture and process images . As a result, there is ongoing pressure to remove non–value-added sensing steps and instead achieve robust performance through control and decision-making that tolerates variation without excessive sensing, fixturing, or conservative motion constraints.

The challenge is that robust, high-precision pick-and-place behavior is difficult to specify explicitly using traditional rule-based logic or hand-crafted reward functions. Conventional automation solutions often require extensive tuning, added sensing, and rigid constraints to handle edge cases. Similarly, learning-based methods trained with sparse success signals or poorly shaped rewards frequently converge to policies that are "almost correct" but remain biased by a few millimeters or fail to generalize across the range of pallet wear and contact conditions encountered in practice.

Therefore, the problem addressed in this thesis is to obtain a control policy that reliably performs pallet pick-and-place with millimeter-level final accuracy across realistic pallet wear, pose drift, and contact variability, while reducing reliance on non-value added sensing and over-constrained automation assumptions.

1.3 Contributions

This thesis makes the following contributions toward robust, high-precision robotic palletization:

- **Physics-based palletizing simulation environment:** Developed a PyBullet-based simulation workcell representing a simplified industrial pallet pick-and-place task, including contact interactions between the gripper, parts, and pallet surfaces.
- **Expert demonstration generation:** Implemented expert trajectories using a traditional robotic state-machine approach to generate consistent demonstrations.

- **Adversarial inverse reinforcement learning reward learning:** Designed and trained an AIRL formulation that learns a reward model from expert demonstrations using a discriminator/reward network paired with a policy generator, enabling reward acquisition without manual reward shaping as the primary learning signal.
- **Two-stage learning strategy for precision:** Proposed and validated a staged approach in which AIRL provides a learned reward representation and an initial policy, followed by a forward-RL second stage that combines the learned AIRL reward with task-specific environment rewards to achieve proper termination states and account for additional variability.

1.3.1 Outline

This thesis investigates learning-based pick-and-place control for industrial pallet handling under variability caused by pallet wear, pose drift, and uncertain contact conditions. The remainder of this manuscript is organized as follows. Chapter 2 reviews the reinforcement learning and inverse reinforcement learning background required for this work, including maximum-entropy IRL and adversarial formulations (GAIL/AIRL). Chapter 3 describes the proposed two-stage learning methodology and the physics-based simulation environment, including the state/action design, reward-learning strategy, and evaluation criteria. Chapter 4 presents the experimental results, including AIRL reward learning outcomes, tuning lessons learned, the rationale for splitting the reward into pick and place phases, and the final Stage 2 PPO fine-tuning performance. Finally, Chapter 5 summarizes key conclusions from the study and outlines directions for future work toward improved robustness and deployment in real palletizing cells.

CHAPTER TWO

BACKGROUND

2.1 Background

2.1.1 Reinforcement learning and MDP structure

When traditional rule-based programming struggles to produce adequate results, a popular approach for learning a desired behavior is reinforcement learning [1]. A common formulation for these problems is a Markov Decision Process (MDP), defined by the tuple (S, A, P, r, γ) [1, 2], where S is the state space, all states that exist in the environment, A is the action space, all available actions the agent can take, $P(s_{t+1} \mid s_t, a_t)$ is the transition distribution that defines the probability of moving to state s_{t+1} given the current state s_t and action a_t, $r(s_t, a_t)$ is the reward received for taking action a_t in state s_t, and $\gamma \in (0, 1]$ is a discount factor. The discount factor is used to discount future rewards, the lower the discount factor, the less the RL model values futures rewards. This discourages the model from staying alive and exploiting the reward structure. When reinforcement learning is structure this way, the objective is to find a policy $\pi(a \mid s)$ that maximizes the expected discounted return.

$$J(\pi) \;=\; \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t)\right], \qquad \gamma \in (0, 1]. \tag{2.1}$$

Robotic manipulation often involves continuous state and action spaces, is partial observability, and rewards can be sparse or difficult to shape [3]. Policy gradient methods address continuous actions by directly optimizing parameterized policies to maximize the expected discounted return, Modern deep RL commonly uses actor–critic structures, where an actor represents the policy $\pi_\theta(a \mid s)$ and a critic estimates the expected return

through a value function,

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r(s_{t+k}, a_{t+k}) \,\middle|\, s_t = s\right], \tag{2.2}$$

The value function is an estimate for the value of each state. Similarly to the value function, there is a $Q^{\pi}(s_t, a_t)$ function. The difference is that the Q function is a function of state and action.While the value function is only a function of state. The Q function is the value of the action. The difference between the Q function and the value function, is the advantage. It is a measure of how much better was this action than the average action I would normally take in this state. for advantage function greater than zero, the action is better than expected. less than zero, the action is worse than expected. It is practical to think in terms of the advantage function, because this difference compare the new action and state to the baseline.

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \tag{2.3}$$

On-policy algorithms such as Proximal Policy Optimization (PPO) optimize a surrogate objective with constrained updates to improve stability and practical performance, which has made PPO a common baseline in continuous control [4]. PPO measures how much the updated policy differs from the previous policy using the probability ratio

$$r_t(\theta) = \frac{\pi_{\theta}(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.4}$$

and maximizes the clipped surrogate objective

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right], \tag{2.5}$$

which discourages excessively large policy updates while still allowing improvement. This is critical to the functionality of PPO, if updates to the policy are left unchecked, chaotic behavior can happen.

Implementing reinforcement learning on a physical system is often not feasible due to the chaotic learning process and the physical constraints of these systems. Physics engines are used to model and simulate these systems, but as mentioned earlier the sparse nature of the reward system can be difficult to produce desirable results, and simplifying these systems with traditional RL does not always scale to the fully complex system. This paper supplements the traditional approach by using a two staged approach where a dense smooth reward is learned and traditional RL is used to improve the behavior on the more complex realistic system.

2.1.2 Inverse Reinforcement Learning (IRL)

Inverse reinforcement learning (IRL) reverses the reinforcement learning (RL) problem described in the previous subsection. Rather than learning a policy from a user-defined reward function, IRL seeks to infer a reward function that explains expert behavior. The expert is provided as a set of demonstration trajectories, D, (state $\rightarrow$ action sequences) sampled from an MDP,

$$D = \{\tau_i\}_{i=1}^{N}, \qquad \tau_i = (s_0^i, a_0^i, s_1^i, a_1^i, \ldots, s_{T_i}^i, a_{T_i}^i) \tag{2.6}$$

where the demonstrations may come from a human demonstration, an existing process, a simplified environment/controller, or other sources. The expert policy is assumed to be optimal under an unknown reward function. IRL assumes that the reward is linear in a set of features

$$r_w(s,a) = w^\top \phi(s,a) \tag{2.7}$$

where w are unknown feature weights to be learned, and $\phi(s,a)$ are the features. In simple environments, this is just a count of state and actions the agent visits along the trajectory. In more deep learning IRL algorithms a neural net learns the features, this also allows for non-nonlinearities. Under this parameterization, the expected discounted return, that was

introduced in the RL section, becomes

$$J_w(\pi) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_w(s_t, a_t) \right] = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t, a_t) \right]$$
$$= w^\top \mu(\pi) \tag{2.8}$$

where

$$\mu(\pi) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t, a_t) \right] \tag{2.9}$$

is the expected discounted feature count for policy π. The expert feature count can be estimated directly from demonstrations as

$$\mu_E = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T_i} \gamma^t \phi(s_t^i, a_t^i) \tag{2.10}$$

where, N is the number of expert transitions, and T is the time horizon of a single episode. therefore, the inner sum of Eq. (2.10) is the average feature count across the episodes. and the outer loops averages that over the entire expert trajectory. Thus, IRL can be posed as finding weights w such that the optimal policy under r_w produces feature counts that match the expert.

$$\mu(\pi_w^\star) \approx \hat{\mu}_E, \qquad \pi_w^\star \in \arg\max_\pi J_w(\pi) \tag{2.11}$$

While this establishes the foundation for IRL, it is inherently an under defined problem. examining (2.10), there are more than one set of weights that would make the expected return of the policy equal to that of the expert. This ambiguity needs to be managed.

2.1.3 Max Entropy IRL

Maximum Entropy IRL introduces a probabilistic model over trajectories to address ambiguity in demonstrations. Rather than assuming the expert follows a single deterministic optimal trajectory τ . a probability distribution $\rho(\tau)$ is defined over the

expert trajectory, and the entropy H is calculated. By maximizing the entropy, the trajectory that imposes the least amount of constraints that still explain the behavior of the expert is chosen.

$$H(p) = -\sum_{\tau} p(\tau) log(p(\tau)) \tag{2.12}$$

the equation for H is the definition for entropy in information theory, however we negate it to make it concave, allowing us to find the max entropy. Similarly to the base IRL function, the discounted feature counts and the expert feature counts is defined as

$$\mu(\tau) = \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t) \tag{2.13}$$

$$\mu_E = \frac{1}{N} \sum_{i=1}^{N} \mu(\tau_i) \tag{2.14}$$

With that in mind, the Max Entropy is framed as.

$$\max_{p(\tau)} H(p) \quad \text{s.t.} \quad \mathbb{E}_{p(\tau)}[\mu(\tau)] = \mu_E, \quad \sum_{\tau} p(\tau) = 1, \quad p(\tau) \geq 0 \tag{2.15}$$

The Max entropy introduction paper [5] shows that the solution of Eq. (2.15) problem takes the form of Eq. (2.16).

$$p(\tau \mid \theta) = \frac{1}{Z(\theta)} \exp\Big(\theta^\top \mu(\tau)\Big) \qquad Z(\theta) = \sum_{\tau} \exp\Big(\theta^\top \mu(\tau)\Big) \tag{2.16}$$

Where θ are the Lagrange multipliers, which also act as the feature weight. $Z(\theta)$ is a normalization parameter which ensures the probability sums to one over all trajectories. To tie this all together, is we rewrite the standard linear reward function in terms of θ instead of W from (2.7).

$$r_w(s,a) = \theta^\top \phi(s,a) \tag{2.17}$$

and combine that with the discounted feature count shown in (2.14). The max entropy distribution can be rewritten as

$$p(\tau \mid \theta) = \frac{1}{Z(\theta)} \exp(R_\theta(\tau)) \tag{2.18}$$

2.1.4 Generative Adversarial Imitation Learning (GAIL)

Generative Adversarial Imitation Learning builds on the Maximum-entropy IRL framework for learning from demonstrations. Max entropy IRL typically requires repeatedly solving an MDP under trial reward vectors, which can be expensive in large or continuous settings [6]. Generative Adversarial Imitation Learning (GAIL) addresses this by learning an imitation policy directly from demonstrations using an adversarial objective, without requiring an explicit dynamics model. [6]

2.1.4.0.1 Occupancy measure matching. Let $\rho_\pi(s,a)$ denote the (discounted) state–action occupancy measure induced by a policy π, This is the count that the policy π visists a specific state and takes a specific action (s,a). The goal of Imitation learning is matching the expert occupancy measure ρ_{π_E}, if $\rho_\pi \approx \rho_{\pi_E}$ then the policy visits the same regions of the state–action space as the expert.

2.1.4.0.2 Adversarial objective. GAIL introduces a discriminator $D_\psi(s,a) \in (0,1)$ trained to distinguish expert state-action pairs from policy-generated pairs. The policy is trained to "fool" the discriminator, yielding the saddle-point problem

$$\min_\pi \max_\psi \mathbb{E}_{(s,a)\sim\rho_\pi}\left[\log D_\psi(s,a)\right] + \mathbb{E}_{(s,a)\sim\rho_{\pi_E}}\left[\log\left(1 - D_\psi(s,a)\right)\right] - \lambda H(\pi), \tag{2.19}$$

where $H(\pi)$ is a causal-entropy regularizer and $\lambda \geq 0$ controls the degree of stochasticity encouraged in the learned policy. [6]

For a fixed policy, the optimal discriminator has the usual GAN form [7], and substituting it back into the objective shows that GAIL minimizes a Jensen-Shannon divergence [8] between the expert and policy occupancy measures, optionally trading off with the entropy term. [6]

The Jensen-Shannon divergence is a measure between distributions to determine the difference from one distribution to the next. This provides an intuition that the policy improves by making its visitation distribution the same as the expert's.

2.1.4.0.3 Policy optimization via discriminator-derived reward. The discriminator output is converted into a dense learning signal for an RL optimizer. Typocially a surrogate reward is defined such as

$$r_{\text{gail}}(s,a) = -\log\big(1 - D_{\psi}(s,a)\big), \tag{2.20}$$

which increases when (s,a) appears more expert-like to the discriminator. Training alternates between updating D_{ψ} using labeled expert vs. policy samples and updating π to maximize the expected return under r_{gail} (plus entropy regularization).

2.1.4.0.4 Limitations and motivation for AIRL. While GAIL can produce imitation policies, the discriminator signal is primarily a tool for matching the expert occupancy measure and does not correspond to a transferable, reward function. In particular, the discriminator (and thus R_{gail}) is non-stationary during training and can encode potential-based shaping terms that reproduce behavior without recovering the underlying task reward. This motivates Adversarial Inverse Reinforcement Learning (AIRL), which preserves the adversarial training structure of GAIL but constrains the discriminator/reward parameterization to recover a reward function that is more robust to changes in dynamics and more suitable for reuse in downstream reinforcement learning.

2.1.5 Adversarial Inverse Reinforcement Learning (AIRL)

In GAIL and AIRL, two neural networks are pit against each other. The first, is called the discriminator. The second is called the generator. The discriminators job is to identify whether the data input into it is that of the expert or of the generator. The discriminator scores the data, and uses it to calculate a reward. that reward is sent to the generator. Then the generator, uses that reward to optimize a policy. The generator's job is to fool the discriminator. This back and fourth trains both the discriminator and the generator and as a result produces more robust rewards.

While MaxEnt IRL provides a baseline framework for learning rewards from demonstrations, applying IRL at scale can be challenging in large, high-dimensional domains with unknown or unmodeled dynamics [9]. Adversarial methods such as GAIL address scalability by learning a policy that matches expert behavior via an adversarial discriminator, but the discriminator signal in GAIL is primarily optimized to match the expert's state action behavior and frequency and does not recover a usable reward. In GAIL, the discriminator can encode reward-shaping terms that reproduce behavior without recovering the underlying objective. This is key in AIRL as well as to the two stage nature of this thesis.

Adversarial Inverse Reinforcement Learning (AIRL) builds on the adversarial training structure of GAIL while explicitly targeting the recovery of a reward function that can be reused and can generalize under changes in environment dynamics or scaled to similar problems. [5, 6, 9]

2.1.5.0.1 Adversarial discriminator with policy dependence. Let (s,a,s') denote a transition sample. AIRL trains a discriminator of the form

$$D_\theta(s,a,s') \;=\; \frac{\exp\big(f_\theta(s,a,s')\big)}{\exp(f_\theta(s,a,s')) + \pi(a \mid s)}, \tag{2.21}$$

where $\pi(a \mid s)$ is the current policy and f_θ is a learned scoring function. This function restricts the domain of the discriminator function between 0 to 1. The closer to 0, the more the state action pair is from the generator. Similarly, the closer to 1 it is, the more expert like the state action pair is. Lastly, if it is equal to 1/2, than the discriminator is unsure, whether it is generator or expert like.

2.1.5.0.2 Scoring funtion to reward AIRL parameterizes f_θ using a the relationship shown in Eq. (2.22)

$$f_\theta(s,a,s') = g_\theta(s,a) + \gamma h_\theta(s') - h_\theta(s), \tag{2.22}$$

where $g_\theta(s,a)$ represents the reward, this is what we are trying to recover, and $h_\theta(s)$ is a potential function that captures potential-based shaping. This function accounts for reward ambiguity. By enforcing this structure on the scoring function we can account for the effects of the environment. This structure allows us to recover rewards that are a function of state only. [9]

$$\hat{r}(s,a) = g_\theta(s,a), \tag{2.23}$$

and can then be used as a stationary reward function for future control like reinforcement learning or MPC. [9]

2.1.5.0.3 Training objective and loop. Given expert demonstrations and rollouts from the current policy, AIRL alternates between updating the discriminator parameters θ to differentiate between the expert and current policy, and updating the policy π using an RL optimizer (PPO in the case of this thesis) with a reward signal derived from the discriminator and the recovered g_θ. This adversarial procedure provides dense gradients similar to GAIL while producing an explicit reward model that can be carried into next steps. [6, 9]

CHAPTER THREE

METHODOLOGY

This experiment was motivated by a reliability issue in an existing industrial pick-and-place (PnP) process. The cell was functional and productive, but it experiences frequent robot collisions during normal operation. The main root cause was variability in the condition and geometry of pallet bottom surfaces, which introduced motion of the pallets when the robot would pick and an inability to nest the pallets during the placement. Conventional rule based strategies (e.g., pallet jostling and roll-in alignment behaviors) were tested but produced only marginal improvements.

The working hypothesis of this thesis is that a learning-based policy can achieve better performance under pallet variability than a strictly rule-based approach. Because the physical process already existed, the trajectories generated by the legacy controller were leveraged as demonstrations to reduce setup and it allowed us to test a second hypothesis, IRL can recover a reward. To safely iterate and scale training, the learning pipeline was developed and evaluated in simulation, with variability modeled to reflect the real-world noise.

3.1 Approach

The experimental structure follows the Markov Decision Process (MDP) formulation defined by the tuple (S, A, P, γ, R). The goal is to learn a control policy $\pi(a \mid s)$ that produces continuous end-effector motion commands for pallet PnP while remaining robust to pallet pose variation and contact uncertainty.

A key design objective was to minimize state dimensionality while retaining the information needed to perform the task. Early prototypes included additional state components (e.g., full pallet/EE orientation, multiple contact signals, and extended history), but these higher-dimensional representations required substantially more training

and hyperparameter tuning before meaningful progress was observed. More importantly, in these high dimensional spaces the agent would easily diverage from wanted behavior.Therefore, the final state representation was intentionally compact.

3.1.1 State Representation

At each decision step, the environment state $s_t \in S$ is defined as:

$$s_t = \left[x_t,\, y_t,\, z_t,\, \phi_t,\, k_t,\, e_t^{pick},\, e_t^{place},\, a_t\right] \tag{3.1}$$

where (x_t, y_t, z_t) is the end-effector position in Cartesian coordinates and ϕ_t is a single-axis orientation component used to monitor end-effector rotation about the "palm" of the robot arm. Full 3-axis orientation was not required for the task and was excluded to reduce complexity. Although may be interesting for future work to see alternative approaches the agent would attempt to seat the pallet. Finally a_t is the active pallet represented by an integer

The scalar stage flag $k_t \in \{0, 1, 2\}$ encodes progress through the task:

- $k_t = 0$: approach and align to the pick target,
- $k_t = 1$: pallet is successfully gripped / in contact with the gripper,
- $k_t = 2$: pallet is placed within a predefined tolerance.

This stage flag provides minimal memory of the task context and encourages the policy, telling it that it is on the right path.

The next two state components are distance-to-goal error terms for pick and place. The pick error is defined as the Euclidean distance between the end effector and the current pallet pick target:

$$e_t^{pick} = \|p_t^{ee} - p^{pick}\|_2, \tag{3.2}$$

and the place error is defined similarly relative to the desired placement location:

$$e_t^{place} = \|p_t^{ee} - p^{place}\|_2. \tag{3.3}$$

Only one of these error terms is active at a time based on k_t; the other term is set to held constant to avoid injecting irrelevant signals into the policy during each phase. The final state variable is the active pallet, a_t that represents the current pallet that needs to be picked.

3.1.2 Action Space

The action $a_t \in A$ is a continuous end-effector motion command applied at each timestep. In this work, actions that connect to the simulation controller are represented as Cartesian commands, and an orientation. The final action command is a grip command. It is binary, zero representing open and one representing close. If the close command is given but the pallet is not aligned with the end effector, then the grip fails. That failure is encoded in the stage flag.

$$a_t = [x_t, y_t, z_t, \phi_t, g_t], \tag{3.4}$$

The cartesian commands are subject to saturation limits consistent with the robot's safe operating envelope. The environment executes the commanded motion over a fixed control interval, producing the next state s_{t+1}. The actions were normalized using (3.5)

$$\tilde{\mathbf{a}} \;=\; 2\,\frac{\mathbf{a} - \mathbf{a}_{\min}}{\mathbf{a}_{\max} - \mathbf{a}_{\min}} \;-\; 1 \tag{3.5}$$

3.1.3 Reward and Learning Strategy

Rather than hand-designing a reward function, this thesis uses Adversarial Inverse Reinforcement Learning (AIRL) to infer a reward model from demonstrations. Demonstrations were collected from the deployed rule-based controller and used as expert

trajectories. AIRL trains a discriminator that distinguishes expert from generated behavior and a reward network that produces a shaped reward signal for policy learning.

Training was performed in two stages. First, AIRL was used to learn an imitation reward model and an initial policy. This training was completed using a simplified environment. The pallet edges were square and smooth and the generation of the pallets were in a constant location. Second, the resulting policy was refined using forward reinforcement learning (RL) to improve robustness and task completion under randomized conditions. This included generating spheres at the mating/nesting surfaces of the pallet to simulate wear on the pallets. The pallets where also generated with random noise on there location. The placement of the pallet could fluctuate by $\pm$ 10 mm. Tis fluctuation was not fed into the RL, it had to learn it by trial and error. For stage 1, the environment reward was ignored, but for stage 2 and engineered reward was calculated and fed back into the forward RL algorithm (PPO) to augment the previously learned AIRL reward. This improved the reliability of the policy. The final reward that was fed back into the RL algorithm was a weighted sum of the AIRL and the enviroment reward shown in (3.6)

$$r_{\text{total}}(s,a,s') \;=\; w_{\text{env}}\, r_{\text{env}}(s,a,s') \;+\; w_{\text{AIRL}}\, r_{\text{AIRL}}(s,a,s') \tag{3.6}$$

The stage 2 environment reward was designed to encourage progress toward the current cartesian setpoint (Pick/Place) and penalize undesired contact that correlates with knocking pallets or robot collisions. Let p_t denote the end-effector position at time t, and let $p_t^\star$ denote the current desired location (pick or place setpoint depending on the task phase). The position error is decomposed into a planar component $e_{xy,t}$ and vertical component $e_{z,t}$, and the reward is computed using a progress term that rewards reductions in error from the previous step. The error terms were split, for two main reasons. One the gripping component on the pallet is a cylinder, therefore the xy position is more critical than the z position. Second, it made it easier to curriculum the learning. AIRL did a great

job at learning xy from the expert but since the observation space didn't include anything relating to pallet motion or knocking over of the pallets, AIRL didn't learn this. So for place, the z motion was de-emphasized until the agent learned to move to x, y first without knocking over pallets then the weight of $e_{z,t}$ was increased so the agent learns to move down. This relationship was the same for entering pick but departing pick, the relationship was flipped.

$$e_{xy,t} = \left\| (p_t - p_t^\star)_{xy} \right\|_2, \qquad e_{z,t} = \left| (p_t - p_t^\star)_z \right|. \tag{3.7}$$

$$r_{\text{prog},t} = w_{xy}(k)\left(e_{xy,t-1} - e_{xy,t}\right) + w_z(k)\left(e_{z,t-1} - e_{z,t}\right) + b(k), \tag{3.8}$$

besides the error the next main goal of the stage 2's, augmented reward was to discourage contact/collision with anything other than the active pallets. This was done through a normal force N_t and again scaled with w weight W_C to not overpower the other signals.

$$r_{\text{contact},t} = -w_c N_t, \tag{3.9}$$

Finally, there were stage depended bonus, the RL algorithim received small bonus ever step after transitioning from pick to place to signal it was on the right track. It also helped overpower the collision with pallets, because in the beginning, the normal force was significant as the agent learned how to pick the pallets without colliding in others. Without the bonus term, it appeared more beneficial to avoid the pick area all together to reduce pallet collisions.

3.2 Simulation Environment

A physics-based simulation environment was developed to reproduce the PnP task while enabling safe large-scale training. The environment itself is based in Pybullet [10] and contains (i) an industrial robot model (Fanuc M-20iD/25 [11], (ii) a Destaco

robot-hand gripper (RTH-4M-L) [12] with custom fingers, (iii) and custom pallets. Figure 3.1 illustrates the overall workcell layout. In both stages, the pallet the agent is instructed to pick is randomly generated. For this Thesis, the right side pallets are the pick pallets and the left side are the place. The correct layers of pallets are auto-generated in the environment based on the pallet number that was randomly generated.

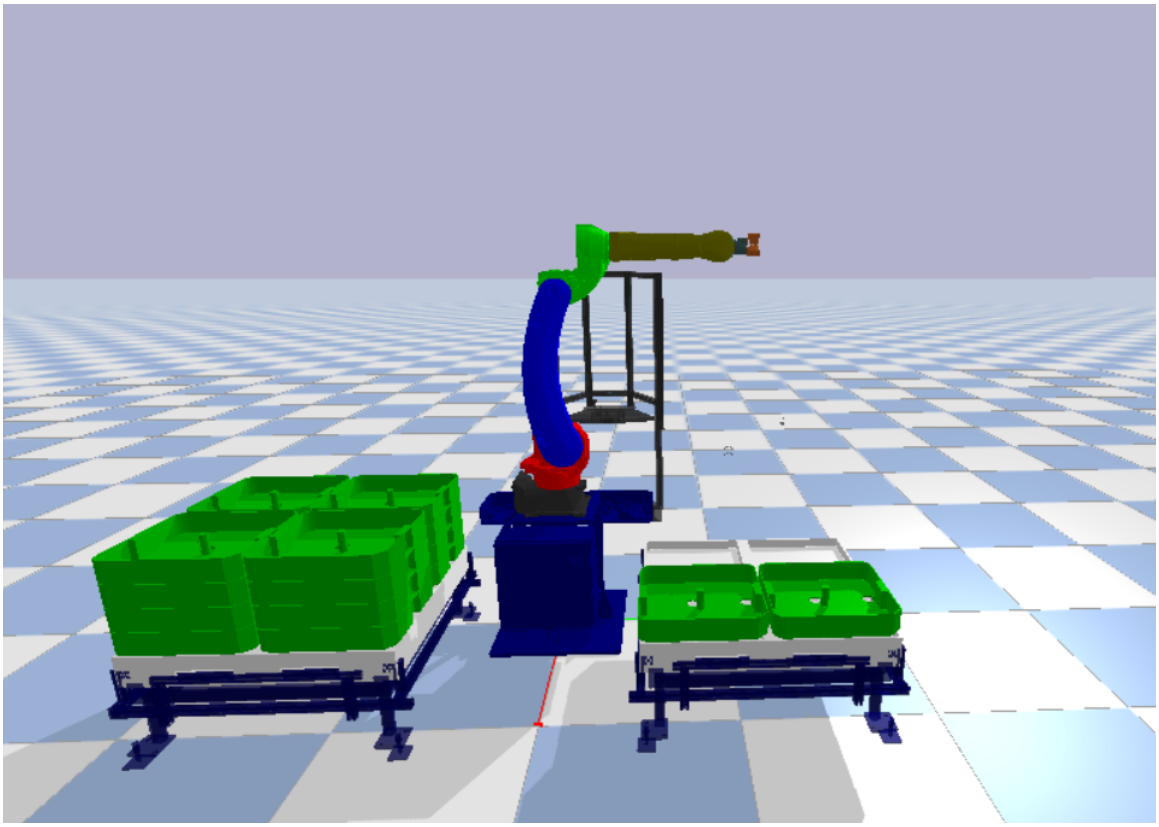


Figure 3.1: Basic project layout of the physics-based pick-and-place simulation environment.

The simulator exposes the MDP interface required for training, reset initializes a randomized scenario, and feeds the inputs based on the current state. step applies the action from the agent, advances physics, computes observations and environmental reward, for stage 1 the environment reward is ignored, and checks termination conditions.

Episode termination is different for each stage of training. For stage 1, the termination is simply a time limit (600 time steps). This is done this way, to avoid leaking information about which trajectory is the expert. Otherwise the discriminators job becomes trivial, it can tell which trajectories have different time steps and identify experts this way, thus not properly learning the reward. In stage 2, its more efficient to stop training after placing a pallet or the time limit is met. Some of the learning was curriculumed (given easier success/rewards to learn desired behavior), and therefore some conditions varied during tuning which will be discussed later.

As discussed earlier, small collision spheres were randomly generated and added to the underside of the active pallet to introduce un-controlled contact variation during stacking/unstacking. This "nesting noise" element is collision-only geometry that slightly perturbs the effective support surface, preventing perfectly idealized flat-on-flat contact and encouraging the policy to tolerate minor misalignment and surface irregularities. By injecting a disturbance at the contact interface, the simulation better reflects real pallets where wear, debris, and manufacturing tolerances influence how parts settle during placement. Figure 3.2 shows the collision sphere used to generate this nesting variability. In Pybullet, the collision geometry is shown in wireframe. There are two pallets, that are attempting to be nested, And Figure 3.3 shows a Stage 1 pallets without the collision spheres. It also shows the ideal nested position.

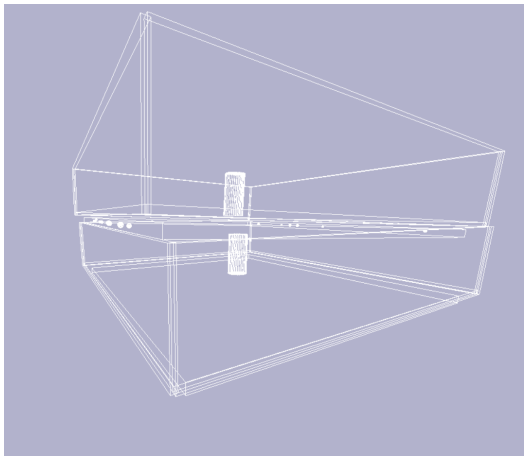


Figure 3.2: Collision sphere used to introduce pallet nesting noise at the contact interface.

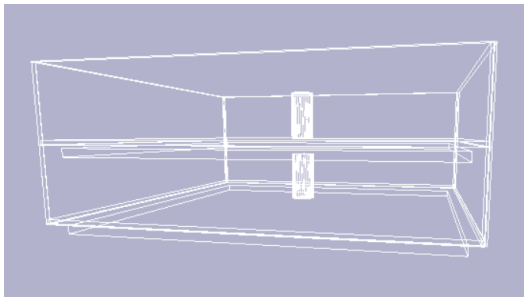


Figure 3.3: smooth and nested stage 1 pallets to show the simplified environment.

3.2.1 Software Stack and Interfaces/libraries

The environment is implemented using the Gymnasium API [13] to provide standard observation/action spaces and a consistent reset/step interface for reinforcement learning. Policies are trained using Stable-Baselines3 (SB3) [14], while Adversarial Inverse Reinforcement Learning (AIRL) is implemented using the imitation library [15], which integrates with SB3 and PyTorch [16]. This stack enables reproducible training, standardized logging (episode return, episode length, and value-function diagnostics such

as explained variance), and straightforward swapping of algorithms during experimental studies.

3.3 Evaluation Criteria

Performance was evaluated using metrics that reflect both task success and robustness to variability:

- **Task completion rate:** fraction of episodes that successfully reach the placement tolerance.
- **Pick success rate:** fraction of episodes where a stable grasp is achieved.
- **Pallet seated:** calculation of weather a pallet is sitting square and nested inside the one below it. If there is no pallet below it, than just square and the error from where it should be is within tolerance.
- **Episode efficiency:** mean episode length (or mean time-to-completion) for successful trials.
- **Safety violations:** frequency of collisions or constraint violations (simulation-based proxy for real-world safety).

The learned policy was compared against the baseline rule-based controller under identical simulation randomization to quantify improvements attributable to the learning-based approach.

CHAPTER FOUR

RESULTS

4.1 Results Summary

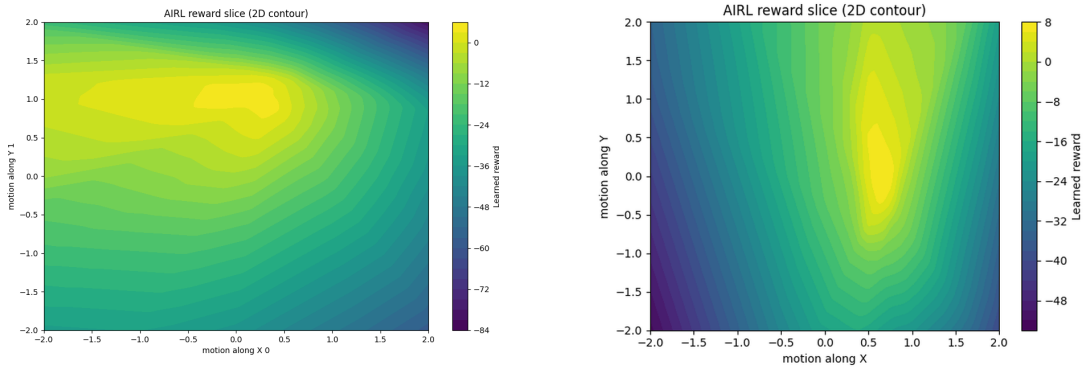
Stage 1 used Adversarial Inverse Reinforcement Learning (AIRL) to learn reward functions directly from expert demonstrations. In the final configuration, AIRL produced two dense reward networks: one specialized for the pick phase and one specialized for the place phase. AIRL is well-suited to this setting because it provides a smooth reward signal that can be optimized with standard deep RL in Stage 2, reducing the need for manual reward shaping [9].

Because the true reward is partially unobserved, reward quality was evaluated indirectly using three complementary techniques. First, Stage 2 learning diagnostics were used to assess whether the learned reward produced a consistent learning signal. In particular, the PPO explained variance approached 0.998 for pick and 0.992 for place, indicating that the value function predicted the returns with very small residual variance. In Stable-Baselines3, EV ≈ 1 corresponds to near-perfect return prediction, while EV ≤ 0 indicates a value function no better (or worse) than predicting zero [17]. Although explained variance is not a direct measure of task success, consistently high explained variance in Stage 2 is a strong indicator that the learned reward is dense and contains little noise for the policy updates.

Second, the learned rewards were evaluated through Stage 2 sample efficiency. Using the AIRL reward networks and the augmented engineered reward, Stage 2 reproduced the expert-like behavior in fewer than 5×10^5 environment steps for pick and 2×10^6 for place. For context, PPO-based pick-and-place training in the literature commonly requires on the order of millions of time steps to converge even with engineered rewards (e.g., 8.3 million steps reported for a Panda pick-and-place PPO

baseline) [18]. This comparison highlights that AIRL substantially reduced the amount of forward RL interaction needed by providing a shaped reward landscape aligned with the demonstrations.

Third, the learned reward functions were visualized by sweeping the end-effector (x, y) inputs over a grid (holding other inputs fixed) and plotting the scalar reward output as a heatmap. As shown in Fig. 4.1, both pick and place rewards form smooth gradients around their respective goal regions, without the sharp discontinuities typical of sparse success-only rewards. This qualitative visualization complements the learning-based metrics above by confirming that reward gradients exist over a wide region of the workspace, enabling local policy improvements throughout training.



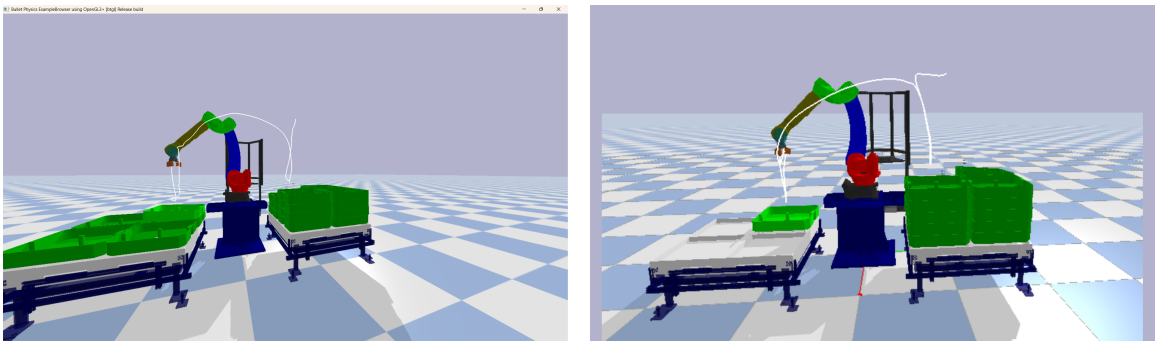
(a) Pick (b) Place

Figure 4.1: Heatmaps of pick and place results shown side-by-side.

Despite partial improvements, the project did not fully satisfy the original hypothesis: the learned IRL reward plus forward RL fine-tuning did not produce a pick-and-place controller that was more reliable than the existing rule-based baseline. However, the approach was successful in two important ways. First, Stage 1 AIRL produced reusable, task-specific reward networks that captured meaningful structure of expert behavior. Second, when these learned rewards were leveraged in Stage 2 with PPO,

Table 4.1: Evaluation summary for Stage 1 AIRL policies and Stage 2 PPO using the learned rewards. The episodes listed are training episodes. The percentages are based on 1000 post-training samples.

Method	Episodes (train)	Pick success	Place success	Seated success
Stage 1 (AIRL policy, Pick task)	6×10^5	57%	N/A	N/A
Stage 1 (AIRL policy, Place task)	2.25×10^6	N/A	42%	N/A
Stage 2 (PPO + learned rewards)	2.75×10^6	75.8%	75.3%	67.7%



(a) learned trajectory (b) Expert trajectory

Figure 4.2: End-effector path comparison between the expert state-machine controller and the learned policy (post-training).

the resulting policy achieved near-expert-like trajectories and substantially higher end-to-end performance over 1000 evaluation episodes. These results indicate that while the learned policy did not surpass the engineered baseline in reliability, the IRL stage successfully recovered a transferable reward representation that can be reused for further training, domain randomization, and future controller refinement. Table 4.1 displays a summary of the final results relative to the success rates. The stage 1 results were generated by taking the best performing policy, and running the environment recording successful picks/places depending on which policy was taken. Figure 4.2 shows the final poly relative to the expert trajectory used to train Stage 1.

As for the safety requirement, The AIRL is well suited to prevent the AIRL from diverging far from the expert without the need to engineer a punishment for this type of

behavior. Because of this, the policy stays significantly far from dangerous zones and therefore the end effector only made contact with the pallets and no other tooling.

4.1.1 Stage 1: Trials and improvements

Why a Single Unified AIRL Reward Failed: In practice, AIRL did not reliably learn the full pick-and-place behavior as a single unified reward. The primary failure mode occurred around termination conditions and the pick-to-place transition. In a combined formulation, completing the pick forces the agent to leave a region of high pick reward and immediately enter a region where the place objective is still far away, producing an abrupt drop in reward. This creates a local optimum where the policy learns to approach and "almost pick," but avoids committing to the grasp/transition because the immediate post-transition returns become negative. For this reason, the AIRL problem was segmented into two phases (pick and place), and the two learned rewards were only combined during Stage 2.

Several mitigation were explored. The first was replacing a single blended error feature with separate pick-error and place-error features. This was retained because AIRL is often implemented with reward models that are approximately linear in the chosen features; a single blended error introduces an implicit nonlinearity precisely at the transition boundary. A second mitigation was augmenting the expert demonstrations by varying dwell time near the transition so AIRL observed more examples of the boundary region. Finally, AIRL was sensitive to pallet disturbance during the grasp transition: the observation included pallet motion terms that were penalized, while the expert demonstrations contained near-zero pallet disturbance. This made even small incidental motion strongly "non-expert" to the discriminator, further discouraging committing to the grasp/transition.

Injecting Noise into Expert Demonstrations: Although the expert controller used to generate demonstrations is fundamentally deterministic, the training pipeline intentionally injects noise (e.g., small perturbations to initial conditions, observation noise, and bounded action jitter) to produce a set of expert rollouts rather than a single repeatable trajectory. This idea mirrors the historical motivation behind behavior cloning (BC), a purely deterministic demonstrator can concentrate the dataset on a narrow manifold of the state–action space, causing a learned policy to overfit and to fail when it drifts even slightly off the demonstrated path. By augmenting demonstrations with realistic disturbances, the expert occupancy measure covers more of the surrounding region, exposing not only the nominal behavior but also the expert's recovery to deviations. It also tells the discriminator which points in space are inherently more important than others. This is accomplished by fluctuating the noise of certain points more than others.

In the context of AIRL, this added variability makes the discriminator's classification problem less trivial. It can no longer rely on memorizing a single simple trajectory to separate expert and generator samples. Instead, the discriminator is forced to learn decision boundaries that are tied to the underlying task progress. This should yield a reward function that generalizes better, remains smooth and dense across a broader set of states, and provides a more informative learning signal during forward RL fine-tuning.

AIRL Tuning and Normalization: AIRL alternates between (i) updating a discriminator/reward network to separate expert vs. generator transitions, and (ii) updating the generator policy (PPO) to maximize the learned reward [9, 15]. The most impactful tuning parameters were those that control discriminator strength and feedback delay. Increasing `n_disc_updates_per_round` improves how quickly the reward adapts, but if set too high the discriminator can become over-confident (low entropy), leading to saturated rewards and poor learning signals. Batch sizes (`demo_batch_size`, discriminator minibatch size) traded off gradient noise vs. overfitting: larger batches

stabilized training, while smaller batches helped avoid discriminator collapse early. On the generator side, PPO parameters such as learning rate, clip range, entropy coefficient, and rollout length governed update stability: overly aggressive updates (high learning rate or wide clip range) produced oscillatory behavior, while too much entropy regularization slowed convergence.

Action normalization was one of the best improvements for both AIRL and Stage 2 PPO. Following common practice for continuous control, actions were rescaled to a symmetric range $[-1,1]$ (Eq. (3.5)). Stable-Baselines3 explicitly recommends symmetric action normalization because it improves conditioning for neural network policies and reduces sensitivity to raw actuator units [19]. Empirically, this reduced early training instability and shortened the time required to reach consistent progress. In contrast, observation normalization produced the opposite effect in this system. A running mean/variance normalization was tested to standardize the observation vector, but it slowed learning and occasionally destabilized training.

Stage 1 AIRL Hyperparameter Tuning via Adversarial Training Diagnostics: In stage 1 the AIRL, proved to be very sensitive to hyperparameter. Because of this, Stage 1 required significant tuning. During training, logs were produced an tuning was performed primarily by monitoring whether discriminator/generator statistics stayed near their expected equilibrium behavior. In the idealized adversarial optimum, the generated distribution matches the expert distribution and the discriminator becomes maximally uncertain, outputting 0.5 everywhere [20]. Under the balanced discriminator batches used in the imitation AIRL implementation (one generated sample per expert sample), the "normal" steady-state signatures are therefore: (i) discriminator accuracy near chance ($\texttt{disc\_acc} \approx 0.5$), (ii) predicted-label entropy near its maximum for a Bernoulli classifier ($\texttt{disc\_entropy} \approx \ln 2 \approx 0.693$), (iii) entropy loss near $\ln 2$ when logits imply $p(\text{expert}) \approx 0.5$, and (iv) predicted expert proportion close to the true expert proportion

(`disc_proportion_expert_pred` $\approx$ `disc_proportion_expert_true` ≈ 0.5) [15]. In reality, it didn't matter how weak the discriminator was made by increasing the batch replay buffer capacity (increases the amount of variation the discriminator sees from the expert) or the decreasing the number of updates per round or demo batch size. The discriminator never got near the ideal 0.5 accuracy. It would sit around 0.7-0.8 in healthy training. If the discriminator accuracy ever got above 0.9 for more than just a couple of rollouts, the training would not progress and in most cases diverge. Since it was never possible to get the values to the ideal range, the generator hyperparameters were set at the weakest values and left there.

- `demo_batch_size` = 256
- `n_disc_updates_per_round` = 1
- `gen_replay_buffer_capacity` = 50,000

As for the generator a significant amount of tuning was required. the generator (PPO) using its standard stability diagnostics because generator update size directly affects discriminator balance. PPO is designed to avoid overly large policy updates [21, 22]. We monitored $approx\_kl$ and $clip\_fraction$ were monitored because they represent step size, consistently large values indicated overly aggressive updates and motivated reducing learning rate, epochs, or tightening the clip range [23]. The KL target, was anything on the order of 10^{-2} (consistent with PPO's reported adaptive KL targets such as 0.01 and 0.03) [21, 22]. Finally, we tracked value-function fit using explained variance, expecting it to increase as policy learning stabilized [17]. No single value was used for training beginning to end, hyperparameter that corresponds to large aggressive updates were used to start training and once PPO started to exhibit desired behavior, the hyperparameters were lowered. If the parameters were lowered to quickly, training stalled and needed to be raised again.

4.1.2 Stage 2: Trials and improvements

Forward RL Fine-Tuning Setup: Stage 2 performs forward reinforcement learning (RL) to train a control policy directly in the task environment using the learned reward models from Stage 1. A policy-gradient agent (PPO) interacts with the PyBullet-based pallet stacking environment, collecting rollouts and updating the policy and value networks from on-policy trajectories. Unlike Stage 1, where the objective is to infer a reward function from expert behavior, Stage 2 treats the reward as known and focuses on learning a robust policy that can reliably execute the full pick-and-place sequence under the stochasticity and domain randomization that was discussed earlier.

Reward Composition and Stage-Conditioned Selection: The Stage 2 reward is constructed as a weighted combination of (i) the environment-provided shaping reward and (ii) the AIRL reward networks learned for pick and place. Because the task contains distinct phases with different objectives, the AIRL contribution is selected conditionally based on a discrete stage indicator in the observation. The active AIRL network is switched using the stage flag so that the policy receives a dense reward aligned with the current subtask while still optimizing the overall episode objective.

Calibration of Reward Scales and Weighting: A key step in Stage 2 is calibrating the relative magnitudes of the reward components so that no single term dominates learning. In particular, the environment shaping reward and the AIRL rewards can differ in scale due to differing architectures, training dynamics, and input normalization. To address this, the reward terms are scaled by tunable weights (e.g., w_{env}, $w_{\text{airl,pick}}$, $w_{\text{airl,place}}$) and optionally passed through simple stabilizers (e.g., clipping to bounded ranges or normalizing by running statistics). The calibration goal is to ensure that each component contributes a comparable learning signal throughout training, while

preserving the dense structure of the AIRL rewards and the information encoded in the environment reward.

Action Interface and Normalization Calibration: Stage 2 uses an action interface consistent with the robot's safe operating envelope. Actions produced by the policy are represented in a normalized space (typically $\tilde{\mathbf{a}} \in [-1, 1]^m$) and then mapped back to physical units via a transformation followed by saturation, enforcing actuator limits and preventing unrealistic commands. This calibration improves numerical conditioning for PPO, reduces sensitivity to coordinate scaling, and makes exploration more uniform across action dimensions. The same mapping is applied consistently in both training and evaluation to ensure the learned policy is optimized under the exact command constraints that will be used at deployment.

Observation/Model Consistency and Reward-Net Calibration: Because the AIRL reward networks consume state (and optionally action) features, Stage 2 must maintain strict consistency between the observation definition used to train the reward nets and the observation presented during policy learning. This includes matching feature ordering, dimensionality, and any preprocessing (e.g. normalization). Prior to training, Stage 2 validates that the reward networks can be loaded and evaluated on live environment observations without mismatch, and that the stage flag used for reward selection is well-defined at every timestep (including transitions). This calibration step prevents silent reward corruption and ensures that the policy is trained against the intended learned objective.

Training Protocol, Evaluation, and Instrumentation: Stage 2 training is executed using separate training and evaluation environments with controlled seeding to support repeatability. Rollouts are collected under the current policy, and periodic evaluations are performed without exploration noise to track behavioral progress under identical environment settings. Logging is instrumented to record not only the total reward

but also the individual components (environment reward, AIRL pick reward, AIRL place reward) and the active stage indicator. This decomposition enables targeted calibration of weights and normalization, and provides a clear diagnostic view of how each reward source shapes learning across the pick-to-place transition.

Action filter to improve Stage 2 motion: Post training, the raw actions coming out of the policy were shaky and at certain stages very large while at others very low, and while they actually accomplished the tasks given to them better than the final policy, knocking over pallets was in almost every episode and deployment on a physical system would be impossible. To improve this, an low-pass action filter was implemented to reduce the large jerky motions of the policy. The action filter implements a two-stage smoothing that converts a the jittery cartesian target command into a bounded, physically reasonable end-effector position command. At each control step, the raw target $\mathbf{x}_t$ is first passed through a deadband relative to the previously issued command $\mathbf{x}_{t-1}^{cmd}$, suppressing very small command changes that typically correspond to a dithering noise. In practice, if a per-axis change satisfies $|x_{t,i} - x_{t-1,i}^{cmd}| < \varepsilon$, that axis is held at the previous command for that step. This mimics more of how a human would program the robot, were the policy always outputs a varying command, even if updates are small.

To attenuate high-frequency action noise while preserving responsiveness during large motions, the filter then applies an adaptive low-pass smoother in the style of a "one euro" filter [24]. A derivative estimate is formed from the difference between the deadbanded target and the current filtered state, $\dot{\mathbf{x}} \approx (\mathbf{x}_t - \mathbf{x}_t^{filt})/\Delta t$, and this derivative is itself exponentially smoothed with a fixed cutoff. The magnitude of the filtered derivative increases the effective cutoff frequency of the position filter according to

$$f_c = f_{min} + \beta\,|\dot{\mathbf{x}}|,$$

so that slow motions are strongly smoothed (reducing jitter), while fast intentional movements experience a higher cutoff (reduced lag). This makes the filter dynamic, it has an apdative cutoff frequency. For each axis, the cutoff f_c is mapped to an exponential moving average gain α using the relationship

$$\alpha = 1/(1+\tau/\Delta t) \text{ with } \tau = 1/(2\pi f_c) \tag{4.1}$$

and the filtered position is updated as

$$x^{filt} \leftarrow \alpha \mathbf{x}_t + (1-\alpha) x^{filt} \tag{4.2}$$

Finally, to prevent large step-to-step jumps, a slew-rate limiter constrains the maximum per-step change in the commanded action, $\Delta \mathbf{x}^{cmd}$, to a user-defined bound. The resulting command is clipped to workspace limits before being sent to the controller. Lastly a freeze *xy* mode is provided for grasping phases, the commanded *x* and *y* position is held exactly at the current end-effector value, and the filter internal states are updated consistently to avoid windup or transient drift when unfreezing.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This thesis studied whether a learning-based controller can improve robustness in an industrial pick-and-place palletizing task where variability in pallet condition and contact dynamics causes failures for traditional logic. A physics-based simulation workcell was developed and used to safely scale training under modeled pallet wear and pose noise, and expert demonstrations were generated from a conventional state-machine controller to provide a practical baseline and training signal.

The original hypothesis—that an IRL + forward-RL policy would outperform the existing rule-based controller in reliability under pallet variability—was not fully supported. In the final experiments, the learned controller did not surpass the engineered baseline in overall robustness. Nevertheless, two major outcomes were demonstrated. First, Stage 1 AIRL successfully recovered reusable reward models from demonstrations, producing task-specific reward networks for pick and place and achieving intermediate imitation success (Pick: 57%; Place: 42%). Second, when these learned rewards were used in Stage 2 with PPO and augmented with an engineered environment reward, the resulting policy produced near expert-like trajectories and achieved substantially higher post-training success in simulation (Pick: 75.8%; Place: 75.3%; Seated: 67.7% over 1000 evaluation episodes). These results suggest that AIRL is valuable in this setting primarily as a reward-recovery mechanism that yields a dense learning signal and a transferable reward representation, even when end-to-end reliability remains bounded by contact disturbances and partial observability.

5.2 Future Work

Several directions could improve reliability and better test the central hypothesis:

(1) Improve state observability for contact-induced failures. A key limitation is that the policy is expected to avoid knocking/dragging pallets while the observation space contains limited information about pallet motion and contact dynamics; as noted during development, AIRL did not learn these effects when the observation did not encode them. Future work should add compact contact-aware features (e.g., pallet motion estimate, contact normal-force, or pallet pose change over a short horizon) to reduce partial observability without greatly increasing state dimension. One could add vision to the system and generate a feedback loop in regards to pallet location.

(2) More realistic pallet wear and contact randomization. The current domain randomization injects nesting disturbances via collision spheres on the pallet interface. Future work should extend this to richer variability such as spatially-varying friction, compliance, and heightfield/mesh wear models, and randomized gripper friction/contact parameters to better match long-term pallet degradation and improve sim-to-real transfer. Pybullet has little support for this type of randomization and therefore the use of an alternative simulator may be necassary.

(3) Systematic comparison to the rule-based baseline. While the learned policy was compared under identical simulation randomization, future studies should expand evaluation to include stress tests (worst-case wear, worst-case pose offsets), time-to-completion, collision counts, and isolating the contribution of AIRL rewards vs. engineered shaping. This would clarify where learning helps and where deterministic logic is better. In addition from this, drawing inspiration from behavioral cloning, it would be interesting to randomize the initial condition, this has the potential to significantly improve action when the policy enters states that the expert never visit.

(4) velocity based action space or delta based. A common design choice in learning-based robotic manipulation is to parameterize the action as an incremental Cartesian command (e.g., $\Delta x, \Delta y, \Delta z$ and sometimes a small orientation delta) rather than

issuing absolute Cartesian waypoints. This formulation appears frequently in modern pick-and-place / manipulation systems, where policies output end-effector pose deltas along with a gripper open/close command [25, 26]. One advantage is that delta commands pair naturally with a lower-level Cartesian controller that converts small increments into smooth joint motion while enforcing magnitude and rate limits, which reduces the likelihood of large discontinuous jumps in the commanded end-effector motion [27]. A closely related alternative is to use a velocity-based action space, since many Cartesian controllers are designed around end-effector velocity servoing and can explicitly saturate the commanded speed to produce approximately straight-line motion with bounded magnitude [27]. In palletizing, constraining motion through delta-pose or velocity commands is a practical way to reduce impulsive movements that can excite pallet dynamics and lead to knock-overs. It also would make punishing large action easier; for these reasons, future work with alternative actions spaces may be interesting.

(5) Hardware validation. The most important next step is deploying the recovered reward (and/or a fine-tuned policy) on a real palletizing cell. A practical pathway is to keep the existing state-machine controller as a safety supervisor while using the learned reward for offline evaluation and incremental policy refinement, then gradually increase autonomy as reliability is demonstrated.

5.2.1 GitHub Link for Code

github.com/jlbowden-9/AIRL-Robotic-PnP

REFERENCES

[1] K. De Asis, J. Hernandez-Garcia, G. Holland, and R. Sutton, "Multi-step reinforcement learning: A unifying algorithm," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[2] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[3] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, vol. 22, no. 30, pp. 1–82, 2021.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[6] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[8] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 2002.

[9] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2018.

[10] PyBullet contributors, "Pybullet quickstart guide and documentation." `https://pybullet.org`, 2026. Accessed: 2026-03-01.

[11] FANUC America Corporation, "M-20iD/25 Data Sheet." PDF datasheet. Accessed 2026-02-28.

[12] DESTACO, "RTH-4M-L - RTH Series - 3-Jaw Pneumatic Parallel Gripper (Long Stroke): Technical Details." Product page, 2025. Accessed 2026-02-28.

[13] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[14] 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.

[15] Center for Human-Compatible AI Lab and Contributors, "imitation documentation: Adversarial inverse reinforcement learning (airl)." `https://imitation.readthedocs.io/en/latest/algorithms/airl.html`. Accessed: 2026-02-16.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1912.01703.

[17] Stable-Baselines3 Contributors, "Stable-baselines3 documentation: explained_variance." `https://stable-baselines3.readthedocs.io/en/v1.0/common/utils.html`. Accessed: 2026-02-16.

[18] A. Lobbezoo and H.-J. Kwon, "Simulated and real robotic reach, grasp, and pick-and-place using combined reinforcement learning and traditional controls," *Robotics*, vol. 12, no. 1, p. 12, 2023.

[19] Stable-Baselines3 Contributors, "Stable-baselines3 documentation: Reinforcement learning tips and tricks." `https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html`. Accessed: 2026-02-16.

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] "Stable-baselines3 ppo documentation (update size control and target_kl)." Online documentation, 2026. Accessed 2026-03-01.

[23] "Stable-baselines3 logger documentation (ppo)." Online documentation, 2026. Accessed 2026-02-27.

[24] G. Casiez, N. Roussel, and D. Vogel, "1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems," in *Proceedings of the SIGCHI Conference on human factors in computing systems*, pp. 2527–2530, 2012.

[25] J. Borja-Diaz, O. Mees, G. Kalweit, L. Hermann, J. Boedecker, and W. Burgard, "Affordance learning from play for sample-efficient policy learning," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6372–6378, 2022.

[26] R. Hoque, A. Mandlekar, C. Garrett, K. Goldberg, and D. Fox, "Intervengen: Interventional data generation for robust and data-efficient robot imitation learning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2840–2846, 2024.

[27] O. Khatib, "Advanced robotic manipulation." Lecture Notes (CS327A), Stanford University, 2005. Accessed: 2026-03-01.