MIT Thesis Template in Overleaf

by

Tim Beaver

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1990

Author .....
Department of Electrical Engineering and Computer Science
May 18, 1990

Certified by .....
William J. Supervisor
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Chairman
Chairman, Department Committee on Graduate Theses

MIT Thesis Template in Overleaf

by

Tim Beaver

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1990, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

Printed Circuit Board(PCB) Synthesis is pivotal in ensuring new circuits get designed automatically considering the complexity of board designs are getting exponentially more difficult. Automating such procedures will require machine learning algorithms (ML) and they often make mistakes. For this project, I utilize Adversarial Inverse Reinforcement Learning (AIRL) to approach a small toy-subset of this problem, in the scope of robotic manipulation. I showcase that this model can help robotic-manipulators have a better large-scale planner that is generalized to find *offending pins* and move them out of the way. I conduct the reinforcement learning via creation of a logical python based PCB along with various logical states and actions - as well as probabilistic environment generation and expert trajectory generation. I then combine this logical solution with inverse kinematics and Drake-body physics, simulating 'pin-manipulation' as a PCB-designer robot might have to do when encountering different kinds of pins. I represent all these abstractions in the physical world with the YCB dataset. This experiment was able to show that AIRL can certainly be useful within the scope of PCB problems and helps *generalize* for offending pin removal in this simple example. The future of Printed Circuit Boards will certainly benefit from further experimentation into more detailed methodologies utilizing similar tools as the ones this toy experiment used.

Thesis Supervisor: William J. Supervisor
Title: Associate Professor

Acknowledgments

Thank you to the staff of 6.843 for a wonderful semester! I learned a lot.

Contents

List of Figures

List of Tables

Chapter 1

6.843 Project

1.1 Introduction

The need for PCBs is more important than ever before. Most modern technologies are leveraging the increased capabilities provided by faster processors, multi-threaded designs and improved power efficiency among other things - and these machines can't function without PCBs (see Figure 1-1). The field of automating PCB design is an important one to improve upon, as it can take a single individual expert well over a hundred hours to meticulously design a PCB and ensure all components are properly aligned[14]. However humans, while requiring more time, ensure the board satisfies design requirements. Automated methods do exist to speed up the process, but they make mistakes. Automated solutions that do not make mistakes are either too computationally intractable, or there aren't many straightforward approaches to ensure that learning how to solve the present board will necessarily generalize to future designs.

A Classic PCB

[width=.5height=keepaspectratio]Media/pcbImage.jpg

Figure 1-1: A classical PCB - one that you may find cousins of if you peek into any compute-capable machine near you![1]

1.1.1 Aiming for Generalized Solutions

Therefore, it's in everyone's best interest to start using newer modern machine learning approaches that consider generality over a larger problem scope. Utilizing classification techniques are only a part of the problem. To solve an entire board is to **play an entire single player game**. In other words, the solution is optimizing over a horizon - that of the time it takes to build a PCB. Thus, it's natural to consider *reinforcement learning (RL)*. Combining both of these techniques (classification and RL) has resulted in **Deep Reinforcement Learning (DRL)**. However, in problems that require significant generalization, there can be even more achieved via smart objective functions, and by taking information from simulators or human demonstrators. Some methods that are making significant progress in this field typically revolve around models which have objective functions that involve entropy[3] and expert demonstrations. This is a field of DRL known as *imitation learning* - and this subset is called *inverse reinforcement learning (IRL)*. These approaches have been modernized to *AIRL*, which utilizes generative adversarial neural networks(GANs) to train a policy generator which learns by **separating the dynamics of the problem from the goals**, becoming more robust to changing environments and dynamics[8]. This is the model I explore in this paper as I apply them to a simple toy problem in the PCB space, as a step in exploring the applicability of these newer RL approaches to generalizing across **different distributions of goals and actions with the same over-arching intent** of various PCB States.

1.1.2 Application to Manipulation

In reality, PCBs are always going to be developed by robotic manipulators working at micro or nano scales. Thus, on top of the RL-methodology based planner above, I also need to explore how this work might come about in physical reality (even if my simulation is at a much larger size). I aim to do this via utilizing inverse kinematics (IK) and simple force application on the pins chosen by the AIRL model discussed above. This allows us to physically manipulate the objects and simulates what it

might be like to move around differing pins. Thus, this will allow showcasing a manipulation-system that is able to utilize newer machine learning models to have an 'improved brain' that is able to generalize to solving a toy problem in differing distributions. The brain will be more 'robust' thanks to the experimental application of AIRL to this problem scope, and the physics will be integrated with the brain's decision. The ability to mix modern DRL methods with the power of physics in manipulation definitely shows the power that such approaches can have. Personally, this combination has certainly taught me a lot about modern robotics![16][1]

1.1.3 Our Problem Setup

To solve this toy-problem, a couple of pieces are pivotal - first the architecture must ensure that we have objects which are accurate enough to simulate and randomly generate PCB environments. Then, it must also ensure that it's robustly able to connect to both Physics within Drake and also with OpenAI's custom gym environments, baseline's modern DRL models, and Berkeley's Human Compatible AI's imitation models[18]. The systems should be able to scale to generate thousands of trajectories, and allow a nice, possibly parallel pathway to a trained model which can be showcased as the decision maker on a physical robot. This was solved via generation of a Pythonic PCB (called a 'PCBStation') as a 'logical system' (LS), combined with an ActualSimulationEnvironment(ASE) - which allows porting the logical system into Drake[17] See Figure 1-2 for a visual on the LS. The architecture was then infused with the various RL libraries mentioned above via adapting and prep-rocessing data generated from the LS and ASE. Various problems, such as random environment generation from probability distributions, functionality of the PCB, coding practices such as deep copying (which is important when there are transitioning objects), working linear algebra and physical motion, connecting to DRL, and more were all solved with this architecture. However, you might still be wondering what the actual 'problem' is.

[1]This class was eye-opening and I'm glad I was able to take it! I have the utmost thanks to Russ (aka Prof. Tedrake) and team for making it!

An Example of my Logical State PCB

Figure 1-2: If I had generated a 'viewer' for the Logical PCB, it would look something like this. Note that the actual str() function does exist and it is far more comprehensive than but similar to the example shown here - regardless, this image gives an idea of what the logical system object is like.

The abstract problem is as follows - there are **6** pins on some abstract PCB Board class. These pins represent possible pins to 'fix'. Out of the 6 pins, *up to 2* can be moved in some order to solve the problem. This may be considered simply 'object-manipulation', but under the abstract notion of application to PCBs. This is because it is assumed that this is a PCB where all the other pins are properly connected, except for 2, which are by themselves okay, but they block pins on opposing sides. In fact, those two specific pins **are blocking important traces** that the other two pins require - so they **must** move. The idea here is that the AIRL and Physics combination must learn to **un-block the pins on the opposite sides in any way possible while learning the 'concept' or 'intent' of the problem *even under differing distributions from the experts environment***. Here's a few examples of randomly generated distributions of these pins already put into 'Drake-space' - see Figure 1-3. They are created on a 'floating table'[2] - an abstract table atop which the abstract PCB object appears with its YCB object visualizations[3]. If this problem were to be solved, in this toy-case, that PCB board would be *functional* in the sense that it would be capable of being powered.

Two Randomly Sampled States

Figure 1-3: Shown after placing it into Drake-space via monogram notation - there are two distributions here, with the first picture showing another angle of the 'floating PCB' in the '3rd' picture.

[2]Let's say a robot on the International Space Station! Or if you're into virtual reality, the 'Space Station Home Base' in Oculus VR - where there appears to be a manipulation robot!

[3]Maybe hardware companies could also use mustard bottle shaped pins? Jokes aside, the pins are varied and these YCB objects **do** represent that.

1.1.4 Testing and Evaluation

The idea is that with this simple toy example, it will allow showing promise that methods like AIRL and future variants (ie better-than-demonstrator[12] methods are starting to appear) are a good starting place to tackle these increasingly important problems in the age of computers. Seeing that manipulation can be applied to physically work under the hood of these modern RL-based planning techniques also showcases that the robots that design the PCBs of today can also benefit from these modern deep RL 'expert based' intent learning systems. The results of this experiment definitely align with this view, especially considering that results indicate AIRL appears to understand intent acrss **vast numbers of unique start states (where the floats ensure each new state is likely to be different from the last)**. This was seen through utilizing a metric of 'Steps to Termination' and comparing to a random choice baseline.

For the reader, here is information on the sections to allow easy parsing. Section 2 showcases **related work** to this field. Section 3 **describes background material - a mini-survey (so it's a bit long) of RL up to AIRL**, Section 4 **describes all that I built for this problem**, Section 5 **discussed the results of my experiment**, and Section 6 **concludes and discuses improvements and further work.** References can be found past that.

1.2 Related Work

The project in consideration is being done thanks to novel results over the past half decade. From Alpha Go's transformation to AlphaGo Zero and beyond([5, 6]), the world has seen novel DRL methods working with algorithms such as Monte Carlo Tree Search (MCTS)([4, 7] can have considerable successes that *don't even require human information.* Building on this, we also notice new methods in reinforcement learning([8]) that seem to be perfect at handling large action spaces that require a well planned action-path (*essentially single player games).*

There have also been various other projects that have used methods in adjacent or

separate parts of the machine learning field in parts of the PCB Synthesis process[19, 13, 2]. These methods have led to promising outcomes, indicating that these sorts of approaches can have success. In my personal thesis work outside this project I am utilizing **MCTS** with **offline RL methodology**[10], to form a self-improving system for a larger scope PCB-synthesis problem. This project is adjacent to that research but has improved my conceptual understanding of the field.

These past efforts showcase there are a novel number of techniques the world can utilize from recent literature in order to solve these upcoming practical problems in PCB Synthesis. This project's toy experiment, utilizing AIRL in the application of PCBs with manipulation, as far as I can tell, is a novel but simple exercise that aligns with this push for more robust models - especially since it's being applied in the setting of a robotic manipulator.

It is important to note trade-offs with this approach - for example, we forego utilizing other recent tools like meta-learning or more recent models. I claim this is okay because we are attempting to understand how well an algorithm from recent times performs when applied to a robotic manipulator in hopes of 'intent' learning. Results here may merge with or inform of how to handle working with other methods. Another trade-off is the reliance on experts, and how I generate them - the experts may not be ideal. However, I claim this is okay because, given our scope, results with this attempt can inform future exploration.

1.3 Approach - Background of AIRL, Imitation Learning and Manipulation Physics

Applying machine learning directly as a perception tool, while useful for cat-dog identification, or object-mask identification, can only go so far. It is the combination of this perception in wide scale systems that truly provide meaning. For example, perception may guide human decision makers or simple robotic algorithms. However, these perception systems never optimize for a longer-term solution - and if they do,

then they are not merely perception tools. We cross into the territory of reinforcement learning when we think about optimizing for a longer-term solution. For any meaningful problem with large amounts of data or complexity in combinatorial space[11][4], we must focus on **DRL**, or utilizing neural networks within reinforcement learning systems. RL methodology gives the perception systems meaning - they are a part of a system optimizing for a much larger and longer-term goal in many cases. There are many RL models and systems[9] - I will focus on AIRL for this paper. But first, I'd like to build up the background of RL and how it lead to AIRL.

1.3.1 Abstract Framework

In RL, problems often work with MDPs. An MDP can be framed as a long chain of states and actions, as well as the results that occur with those said states and actions. In some models, this formulation involves a transition probability - the *underlying transition network* of the environment at hand. In other problems (namely **model free** RL problems), we are able to marginalize out the transition probability due to high sample count and focus more on the state and action pair and do not see all future states (hence the model is focused on the distribution of the $(A|S_i)$ random variable). Since the models I use have gradients that remove the transition probabilities due to their lack of reliance on the neural network parameter (often called θ) that makes the actions, the abstract MDP I define here will not utilize the transition probabilities but I write this to note of their existence. Thus, we can define our MDP as a 4 vector of **(S,A,R,S')**[5] where each element is itself a tensor relating to the trajectory formed by the MDP. Note I do not list some other terms in the tuple, such as the discount factor, the initial state, etc... - they can be thought of as implicitly being formed or used within this MDP. **S** refers to the states/observations (the former is for full observance, the latter is for partially viewed observance) along the trajectory. It's a tensor that is as long as the trajectory, and per each index contains a tensor related to

[4]Combinatorial Problems are very interesting as improving abilities to solve them often map often to finding and merging RL techniques with clever ideas!

[5]Yes, this acronym does not vibe with the times

the observation - i.e a multi-channeled image, or a simple vector. **A** is a tensor that contains per each i'th index, the action performed at the i'th state. **R** is the reward tensor, which contains at every i'th state, the immediate reward received for taking **action** A_i **at state** S_i. **S'**, which may be seen as redundant (but in some datasets where the buffer is not in order, may be important) is merely the state reached after taking $A_i from S_i$.

It's important to know that the distribution is impacted upon how this is viewed - viewing S' as the result of a distribution across all A_i possible from state S_i gives a distribution of possible actions that led to S' - namely that S' may be different if in the future the same action were taken from the same state. Again, remember if this was **model based**, we'd have some way to gauge those transition probabilities and 'see' all the future possible states (S') and the probability to them[6] In other cases, viewing S' as the surefire transition from $S_i with action A_i$ implies that the underlying transition probability is 1. Note that both formulations (model-based and model-free) still involve the **probability of the actions** as predicted by the machine learning model - θ - which is different from the environment's transitions. In the objective function of deep RL networks, it's **this theta output distribution** that often matters most - but the other distribution (transitions) is typically marginalized out and thus inferred in the underlying optimization. Other important concepts revolve around the reward itself - such as discounting (via parameter of failure, γ) the reward from state S_i to future states, and summing up what is called the **discounted value**. This is a key part in training models, because this helps identify **exploration vs exploitation** opportunities for an agent, which is very important to maximize long-term results. This MDP formulation is often the core to understanding most RL problems.[9] You can see an example of a *winning MDP* from a model we discuss later in results in Figure 1-4.

Where AIRL changes is that the underlying reward function (R) is **unknown**. Thus, a part of the formulation becomes not just finding the best action (also known

[6]The MDP would look like **(S,A,R,P)** at this point, where P would be a transition function that dictates the probability of going to state S' from state S.

An MDP from the Model Developed for this Paper

[width=.5height=keepaspectratio]Media/WinIn4.PNG

Figure 1-4: This MDP showcases **4** MoveActions that lead from State S1 to S2, and so on until the game is 'won' (the pins are cleared - you may have to zoom in to see their final locations). The iiwa gets a little help between 'MDP Frames' in case it bumps things it shouldn't or doesn't push it all the way. This could surely be improved with more time but for now the purpose of the desired paper is achieved this way.

as a policy) from every state available in the environment, but actually understanding the underlying reward function from the expert demonstrator. This 'understanding' is tested by comparisons to the sampled trajectories from improved REINFORCE like algorithms (advanced on-policy algorithms such as Proximal Policy Optimization(PPO) - see more below)[16]. We will discuss other aspects of AIRL further down, but this is how, in the base MDP formulation, AIRL is different.[8].

1.3.2 RL Overview to Modern Imitation Learning

Behavioral Cloning is often a start concept taught to those studying RL, because it has a natural intuitive way around it's framing. The idea is that we want to get an agent to mimic some expert trajectory. Unfortunately, things are bound to go bad as slight deviations cause the agent to deviate far off on it's goals as it's model hits spaces the demonstrations did not show. Solutions with manual fixes (i.e. replace the automated action choice with a human - i.e. DAgger) can only go so far.

More automated solutions arise when we consider gradients of future rewards - we end up at the Policy Gradient formulation (classically always taught starting with REINFORCE, an algorithm that embodies this). It requires the agent to sample future trajectories *by itself*, gather rewards, and optimize itself to travel in the direction of the best average discounted value. We say 'average' because, as we mention above, there are probability distributions - both by the underlying transition dynamics of the environment, and also by the agent's own choices - which requires the usage of expectations in the reward function. Many methods are implemented here to reduce variance - but luckily bias is naturally reduced as the model samples *real* rewards in

its *collection trajectories* N number of times (i.e. for all trajectories). This is known as an 'on-policy' model because the agent doing the sampling is also the one containing the neural net (θ) being trained. A reformulation of this is an actor-critic model (i.e. another agent does the training, while one does the sampling). This eventually leads to algorithms that are model free, where the distribution changes to not know which future states, s', are reachable, but instead *focuses on predicting optimal actions* and hopes to land at good states.

This leads to 'off-policy' algorithms - where one model collects data and the other learns from it. Further improvements allow **bootstrapped** neural networks, where the learner has a target function generated from older versions of itself (see DQNs). All these methods heavily consider variance and bias - variance being how the traveled trajectories can vary in output rewards, and bias being how off a neural-network might *consistently* be with it's answer relative to the unknown truth. The goal is to reduce variance - for example in on-policy methods, the usage of reward-to-go (only future) values and baselines (i.e. advantages) reduces the variability of the data without changing the model's ability to predict. Similarly, to reduce bias involves techniques such as reducing how often incorrect/not yet fully trained neural networks are used, or ensure constantly changing networks in bootstrapped settings lag by some amount of timesteps. Where possible, these models attempt to also increase convergence capability, even if not guaranteed (which often happens due to the differing spaces of the optimization - i.e. l2 norm vs the infinity norm from the bellman operator). These improvements are more practical in nature.[9][16].

Recent developments have led to Offline RL, which is one step further than off-policy RL where it's assumed that **all the data** in the buffer is all the data we'll ever see. The model is trained in a supervised learning (SL) approach where the training is sampled randomly i.i.d. from the data-buffer in batches and mini batches. These models typically require a lot of data, and typical SL approaches have to be adapted to the RL formulations[10]. There are also other approaches - such as applying transformers to RL, meta-learning, curriculum learning, etc.. that are all angled at getting more out of existing data and improving model learning. Other approaches improve

the 'REINFORCE' policy gradient above with improved 'trust' regions or entropy in the objective in forward RL - such as Trust Region Policy Optimization(TRPO) and previously mentioned PPO. There are improvements happening yearly - with new combinations of algorithms commonly tested (ie as mentioned before, MCTS with Policy Networks, etc..) or improvements to these newer algorithms mathematically in their objectives.

However, much of these formulations require **a reward signal**. There are a class of **imitation learning algorithms** that work without a reward signal. These have been most commonly seen as Generative Adversarial Imitation Learning (GAIL) and AIRL. These are able to take better advantage of their experts **to infer the underlying reward signals from many possible solutions**. Once that is known, forward RL can be run. This is an interesting approach, and is the approach I wanted to apply to this problem. The idea of forming the reward signal within the imitation RL framework felt interesting due to it's potential to generalize. This approach involves using Generative Adversarial Networks (GANs) - a policy generator and a expert-vs-generator discriminator. Furthermore, the entropy-based objective is baked into the discriminator. This works by attempting to maximize understanding the experts 'intention features' (important points of their trajectories that are commonly seen across varied experts). This is done by comparing those commonly seen 'expert features' to the samples run on the environment by on-policy policy-gradient based algorithms (such as, for example, PPO, which is what I use). This makes both GAIL and AIRL able to generalize very well to goals. However, GAIL, unlike AIRL, is not robust to changing environments or changing dynamics - which may be important in the PCB setting as pins can change locations (our main focus), wires can break, etc.... This is why this project wishes to use AIRL.[8]

1.3.3 AIRL

Sharing many similarities with GAIL, the most important pieces of this algorithm have been discussed above. However, what truly separates it is their concept of 'reward shaping'. GAIL has 'entangled' rewards.[8] The reward function cannot be

pulled out as it's entangled with the environment it was trained with. That's because the discriminator relies on the (S, A) pairs in the trajectory. Thus, **there is - even in this advanced form of imitation learning - only mimicking going on**. If the environment changes slightly (ie differing locations, broken gripper finger, etc...), these models will be prone to collapse easier. AIRL, on the other hand, by questioning reward shaping:

$$\hat{r}(s, a, s') = r(s, a, s') + \gamma * \Phi(s') - \Phi(s)$$

is able to provide a possibe solution to dis-entangle the reward. This reward shaping structure is seen throughout RL as the idea that the **true reward** (and thus by relation the optimal policy) will not change when the semi-optimal reward is computed by **any function of states** relative to the truth in the form seen above.

In AIRL, the reward is **disentangled** from the actions that got it (i.e. the **dynamics**), leaving us with a Q function which *does rely on the state, actions* but the reward function does not.[8] This can have pros and cons - pro in the sense that now changing environment dynamics or agent dynamics (to an extent) can still result in a similar goal being achieved - and **the 'concept' of the goal being understood rather than mimicry**. However, the cons are that the rewards may be more 'jumpy' since it no longer contains the (state,action) reward which may be the ground truth reward in some situations[15][7].

I should quickly note that there are now better than demonstrator models such as Hybrid AIRL[12] and newer ones - however for this problem and just to start applying this to PCBs, we focus on AIRL.

1.3.4 Physics - IK and Simple Force Grasping and Pushing

As shown in 6.843[16], real world applications don't always need reinforcement learning everywhere. It's enough to use RL for the higher level motion/action planning or goal planning, and leave the low level to the physics engines and systems. The combination of the two can result in interesting results, and improvements in one

[7]This[15] Variational AIRL paper claims to fix this where they essentially conduct AIRL and are able to pull out the proper ground truth reward while still keeping it disentangled

often translate to an overall improvement in the combined system. That is a part of what this experiment is showing. Utilizing more generalized and robust reinforcement learning tools, such as AIRL, I wish to know if it can properly generalize to a changing set of randomly generated environments with similar goals in the domain of PCB pin blockage removal. Thus, by utilizing the newer IRL model for the larger level motion planning, then using the lower-level physics controllers to actually conduct the manipulation, this experiment will showcase an improved manipulation system that is more robust than random across *differing boards from a distribution.* In other words, it allows us to conduct **'Pin Blockage Manipulation'**.

There are two main components in use here for low level physics. The first is Inverse Kinematics (IK). IK is a low level physics planner that allows moving the joints of the robot based on a goal. The goal in our case is the final location within which to move a selected pin towards. The generalization is that the pins will now **change locations** each time, so solving the problem requires **understanding the goal - not just mimicking the experts**. Once the AIRL model above does this, the IK solves for robotic joint configurations to lead it to the goal via keyframe interpolation. This is done by converting the final desired position of the pin specified by the chosen action to the 3D RigidTransform pose. Then, from that final location, and the current location of our gripper and the pin, we interpolate positions within a certain timeframe. Once that's done, *per each position keyframe*, we utilize IK, which given a set of goals in position and velocity space, solves for the underlying configuration space of the **joints of the robot space**. This is done via using the **inverse Jacobian** to figure out what the configuration space of the robot should do (the joints) in order to achieve the desired velocity and travel through the key-frames to the desired goal.

The other main component in use is Simple Force Grasping and Pushing. During the 6.843 lectures, especially in the latter half, the extent of this sort of manipulation is very far. There are solutions here from antipodal grasps with darboux frames to PID force/position control. For our use case it's sufficient that the pins are manipulated 'clear' of the goal. Thus we rely on utilizing force while the arm is in motion -

ie 'closing the gripper' or 'leaving it open' and pushing on the object away from the location of blockage (applying force). In case any deviations occur, I utilize monogram notation and work in 'frames' - where frames indicate a completed action. I then help reset the position for deviations by clipping to the desired final location.

As briefly mentioned above, we must heavily rely on the **monogram** notation learned in class, which has been a good lesson in general 3D space linear algebra manipulations. They are also heavily used. Much of the background for this section can be found within the lecture notes for 6.843[16].

1.4 Approach - Problem Setup

Much of the setup will borrow heavily from the technical RL foundations and Physics described above in [sec:airl]Section 3. I'll also dive into the logical objects generated and the technicalities of what was developed.

1.4.1 Logical State and Actions

In order to even have this problem in the context within which it was desired ('Pin Blockage Manipulation'), we have to setup a PCB within the Drake system. To do that, we require some abstract notation of a PCB that is able to change state - a 'Logical System'(LS). The first creations were the **PCBStation Object**, then **the Pin Object**, and the **IAction** interface along with a few **Action** implementations (specifically the **MoveAction** implementation which uses angular parameters to move pins away within a 360 degree circle split into arcs). Any number of actions can be added on but for this project I heavily rely on my implementation of MoveAction.

The **PCBStation** object contains methods to automatically generate distributions for the PCB, based on a random distribution of possible float positions sampled from a uniform distribution. Certain pairs of pins contain diversity in the sampling, leading to many random values of possible boards. We also contain **strategies** to greedily-identify the expert per each **random value environment toy-problem from the distribution**. Generating these experts involve knowing the setup of

the problem and greedily moving pins out of the way by comparing collisions and locations, which the PCBStation is also capable of conducting with simple bounds checking. This is possible because while we test **random distributions (and thus new boards)** each time, the boards all come from a common **board class** - i.e. there are a million raspberry pi's with various defects (some of which may match this toy problem!), but one 'logical' class of the raspberry pi. Deep Copies were also pivotal and were implemented meticulously as MDPs by existence constantly change state.

Moving on to actions, the **IAction** interface gives a method to apply - 'applyM' - this method will for any state it gets passed in, return the updated state (hence again why deep-copies are important as this may lead to unforseen consequences!). We focus on "MoveAction", one of the implementations I created for the IAction interface. This action, when given a pin, the board (PCBStation) of that pin, an angle, and a radius, will move the pin via the radius and angle **iff** it doesn't collide with another object in the new location. All possible action combinations for the entire state space are generated **and each is given an one-hot integer**. We utilize a **canonical** form of storage for an action involving the **(ActionClass, CanonicalPinUUID, Angle, Radius): Integer** and vice versa. This allows easily converting between RL network outputs and the PCBStation logical objects (and by the ASE the physical objects) and vice versa - even across distributed network connections if needed. See Figure 1-5 for an example of a MoveAction.

An Example of an MDP Action

[width=.5height=keepaspectratio]Media/ActionConcept.PNG

Figure 1-5: Here's a MoveAction that's identifying 2 180 degree arcs to move the pin, for example.

1.4.2 The ASE

So we have a logical object - thus the next step is to 'glue' this to the physics of Drake. The way that works is as follows - we implement an **ASE object** which takes any logical object as input. From this, it generates all the basic physical renditions of the

pins. Feeding from the logial object, the ASE converts all the logical pin positions (which are in form (x,y) within the size of the 2D logical PCB Board) into various data within the ASE. This data contains the 3D location, identify the physical object (i.e. Drake's '*model instance*'), and so on. From this, between adding manipulands - which are the YCB Objects set *per pin* in the logical object pre plant finalization - to properly utilizing monogram notation and doing the linear algebra to setup the pins in the world post plant finalization, we arrive at a initialized physics environment. See Figure 1-6 for a visual. The setting is the **ManipulationStation** for bin-motion - where the iiwa moves a foam brick from one bin to another. We ignore that foam brick movement, and instead generate *floating PCB Table* above that bin (imagined as a virtual 'table' in zero gravity).

Logical System to the ASE

[width=.5height=keepaspectratio]Media/L2Phys.PNG

Figure 1-6: Here's a LogicalState that gets converted and initialized in the Drake-space via Monogram Notation and Linear Algebra of relative positions.

Thus we have converted the LS to physics in Drake. The ASE contains various modular functionality to utilize Drake as seen through 6.843 lectures. From controllers to do inverse kinematics with Jacobians and classes that handle interpolations, to methods and functions that generate rigid transforms or give the possibility for improved gripping (some ready but not utilized due to time constraints), lots of functionality was implemented. A particularly important method computes and physically pushes any pin in the robust and changing PCBStation data type during MDP transitions. This is done via utilizing IK - and is simplified by the nature that this is a **simulation - i.e. we don't need to use convolution networks to classify objects since we know correspondences by initialization.** Another important method converts the **logical (x,y) embedding** into the world's X_{WPin}.

This physics system now follows the 'brain' - aka the generator's output, where the generator was the neural network within the PPO algorithm that was trained by the AIRL GAN loop. If the simulation misses the default position by a bit, the rendering step will automatically fix it via monogram notation algebra, as long as the intention

of the inverse kinematics was correct. Once more the concept is to showcase that AIRL + Low Level Physics in the context of a toy PCB problem of 'pin-blockage-manipulation' for the purpose of 'pin clearing' to prepare board improvements can properly function and generalize to different distributions. The work done showcases this is possible, and so slight mis-rendering's are given the ability to be quickly fixed (ie reset/clipped to the proper next position) as long as the overall model was clearly conducting the right intentions.

1.4.3 The MDP Generated

The MDP is similar to the abstract one described in [sec:airl]Section 3. **S, the state** is the PCBStation object represented as a tensor of pin positions (which for our use case is a good enough representation. though **images** were another possibility). Thus per trajectory (ie one run), this is a (N,2*P_n,1) tensor (where P_n is the number of pins we have, and multiplying by 2 allows us to encapsulate their **coordinate points** in the PCB-space in a manner the AIRL Library accepted). The action is a (N,Angles*P_n*Radii) tensor of the one hot action, which can be in-versed in a dictionary via one-hot canonical keys. We also store if it's **terminal or not** (N,1), and the **reward** - these are all also (N,2*P_n,1) tensors. This is for a **single trajectory** - per experience tuple (a single transition), the external 'N' term can be removed.

Now, **for AIRL, I will not require this full MDP dataset**. However, it's interesting to note that the dataset I create can be applicable for offline RL, or for various other forms of RL. For my problem, however, I generate the HumanCompatibleAI (built above stable baselines) trajectory using just states and actions to match their **state-state-dense or state-action-dense models (ie (S,S') or (S,A))**. This trajectory provides a core part of the AIRL model we see below. For **our particular experiment, where we do not *fine tune the model nor use other experts outside our greedy experts,*** we generate **10,000** random experts across 10,000 unique randomly generated environments from the same board class.

1.4.4 The AIRL Model

With the above logic setup for MDPs, a logical PCB Board, and a Physics Controller, we have done most of the pre processing and generation work. The AIRL Model of the HumanCompatibleAI codebase, which is built using the formulation described above in [sec:airl]Section 3, works with my embeddings and within my framework. The only additional requirement is that this model needs an OpenAIGym Environment. Due to my design, **Drake is connected to my logical PCB and Actions**. Thus, I can run the OpenAI Environment using **only my logical PCB and Actions**, then take the final outputs, and map it into the Drake physical space. Therefore, I built my OpenAI Gym Environment as a custom environment utilizing just the logical state and action pairs. The AIRL model also requires a forward-sampler - for which I used StableBaseline3's default PPO implementation - this is an advanced policy gradient on-policy method described earlier. Since **parallelization** is commonly used in baselines, I had to convert the environment into a **vectorized** environment which has methods such as **step async** and **step wait** that allow parallel parsing of the state space in forward sampling. This allows *more* data to be sampled to compare with the experts.

1.5 Evaluation and Discussion - Results

The desired experiment outcome - which was generalization of 'intent' across changing environment distributions used on a physical manipulator - was achieved. I found that the model **was properly able to generalize** to varying environments and find the right pins to move out of the way. It **wasn't perfect** but the data seems to suggest that **it clearly learned the 'concept' or 'intent' of the experts**. The data that indicates this robust 'intent' understanding was based on my experimentation and interpretation of the statistical output across large numbers of runs. The model was capable of generalization, and the physics allowed for the lower level manipulation, creating a overall 'manipulation thinker' that seems to be robust.

1.5.1 Evaluation Protocol

The goal of this experiment is *to test the 'intent-learning' of AIRL within a PCB Pin Manipulation problem.* The hope is that AIRL will allow broader generalization to changing dynamics under a singular intent - clear the blocking pins. The evaluation protocol to test this is as follows: **Compare the average number of moves taken for a random/un-trained PPO network to solve the toy problem with the number of moves taken after training to solve the same problem (where every single tested instance varies in start/end states)**. The reason this is a good indicator is because of the experiment's reliance on **changing distributions of the environment**. Every single new instance is fundamentally different than any seen before it, because of the near-continuous nature of floats. Thus, a model that learns the 'intent' of the problem will have improved performance to random even if it is given thousands of new samples. As far as the lower level physics go, it's job is to manipulate the pins clear out of the way. The AIRL model's evaluation will correspond to the lower level physics evaluation.

1.5.2 Tests and Outcomes

Before we train any model, we generate 10000 expert trajectories. These are required to train the AIRL model, and in our case are generated via a greedy strategy that works under various distributions of the environment.

Using the ability to generate a new random sampled environment, we then run 10,000 increments of testing on a *brand new model* with no training to get a baseline statistic for the model's capabilities. We do this while tracking the number of **'steps taken to solve the board per each randomly generated board within the same board class'**. We also compute the standard deviation of these runs. We then repeat this experiment with the **post AIRL trained** PPO-generator model. Assuming the **correct fine tuning - or the right seed of tensorflow during runtime,** the results are indeed suggestive of the 'intent' of the solution being learned. There could be more done, such as more robust experts, or more fine-tuned parameters for

the model - but the results obtained with what was built are enough to suggest intent-learning in this problem scope of 'pin-blockage-manipulation'. It's likely that these future possible improvements would only showcase the results with more strength.

We find that **there is an improvement of 1.9 moves** over those 10,000 'test' runs when comparing the baseline random model to the trained one (a total of 20,000 runs). This indicates that **even on changing** environments, different than that of the experts, AIRL is able to robustly 'learn the underlying intent' from the showcased experts and solve the boards better. This is unique because while the overall 'goal' **concept** stays the same, the various changes in the 'dynamics' (ie the mini-goals - aka state-action pairs) are different. And yet, the model seems to pick up an 'understanding' of what to do as suggested by these results. A typical RL model or net might not be able to generalize as much due to the dynamics of the action choices being related to the state where they were taken from. Now that all states look different, the underlying optimization would be tougher for those models - but AIRL's formulation likely prevented collapse and instead indicated intent learning. Typically for those problems, **the 'initial condition' varies but the final goal is the same**. In our case, **only the intent - or the larger meta-goal - is the same.** Both initial and final conditions are different. See Figure 1-7 for the data before training and after training. We see that on average, the number of moves decreases by a large margin whereas the deviation stays similar, indicating that it truly did improve. The training data is also shown in 1-8 which describe the AIRL model improving it's generator post-training with a good seed.

Regarding the lower level physics side, the performance achieved the desired result of a lower-level physics controller acting on generalized higher level model commands to create a manipulation system. The interpolation, joint movement, and overall result of the physics in following the AIRL model's output worked to the extent that it was able to 'force clear' the PCB pins out of the blocked zones. With more time or in future work, this can certainly have improved results, but what was done was enough to showcase the desired outcome. Utilizing monogram notation and 'frame by frame' (where a frame is a step in the MDP space, not time-steps) adjustments

Number of Moves to solve Board on a Good Seed across 20,000 Unique Samples

State	μ	σ
PreTrained	9.0745	6.023
PostTrained	7.2036	4.8492

Figure 1-7: Given a random seed, this was the result of running PPO, non-trained, on 10,000 samples - each sample different. As you can see, **on average it took it about 9 moves to solve the board pin clearing problem.** With the same seed (which happened to luckily be a good one as we haven't done much fine tuning), it took around 7.2 moves. That's a clear improvement, indicating over **10000 new** random boards, the 'intent' was learned. The σ - standard deviation - seems to imply improvement as well, as it tightened around the new value.

An AIRL Model with a good PPO Generator that fools the Discriminator

[width=.5height=keepaspectratio]Media/GoodSeed.PNG

Figure 1-8: Here we see a particularly good model generated from the random seeds within the stochastic nature of machine learning models. We know it performs well because it's able to have strong results over *many thousands* of samples. The could be further shown as significant with p-value testing, but for now we can clearly see the numbers shown earlier in Figure 1-7 indicate it's an improvement over the random model.

if required with the physics allowed this experiment to showcase the desired concept. See Figure 1-9 for this in action.

In Action! The IIWA starting to free the middle.

[width=.5height=keepaspectratio]Media/PushedOut.PNG

Figure 1-9: Here's the IIWA following an AIRL command to move the 2nd pin in the middle. Re-adjusting of frames will re-fix the final locations, but the iiwa had the right intent and manipulated the object away to clear the middle in 0-Gravity ISS conditions!

This showcases that going down this route of expert-trajectory based inverse RL in the field of PCBs is definitely a good path to follow for more robust results within this space of PCBs.

1.5.3 Anomalies and Issues

Much of the development was done in Deepnote, since that was the initial starting point. This led to issues for installing HumanCompatibleAI Library later on and caused me to form, download, re-upload, and manipulate (no pun intended) various Docker images. Deepnote would also often crash leading to tons of re-run moments. On top of this, I found *build inconsistencies* that caused me to spend lots of time - such as Mesh-cat working improperly offline vs on Deep-note **even though they ran the same code**, or Python's paths containing the right module but it wouldn't work on Deepnote's notebook (but would work on it's terminal). Thus I wasted some precious time on these build issues, and the results of the experiment could likely have been better if I had started with a local tested docker container.

Something to note is that the **seed** matters - fine tuning of the model was not done, so the results, to be replicated from pure code, **will require fine tuning parameters.** If that is done, the results may very likely improve beyond what was discovered. I do note that **the model discussed in this result section *is saved and replicable if purely loaded in.*** Even without fine tuning, a few runs can generate a good seed. This model discussed here was from one of these seeds. *It is replicable for any unique numbers of new instances (i.e. it truly has the desired*

performance).

Another interesting anomaly was that given the generating experts from a greedy policy across different distributions for this experiment, it appears that 20 experts (almost one-shot) was enough to teach it - though we stuck with 10000 for the main experiment. This might be interesting to explore one-shot learning with approaches similar to this, although that was not the goal of this experiment. Of course utilization of more robust algorithms over greedy for expert collection, such as MCTS or other policies, may generate more interesting experts that don't abide by this anomaly. Also, the problem scope, if increased in size, may reduce this anomaly as well.

1.5.4 Limitations

Due to the limitations of time, there are certainly improvements that could be made. Starting with simple ones, such as improving the lower level physics code with even more concepts covered in 6.843. Even simple changes, such as making all pins different variations of a brick, would allow this to stylistically have better manipulation properties. We also helped the iiwa in case it threw **other pins** off course when it didn't mean to (so as long as the 'desired' positions made sense, we would 'help' the iiwa by re-positioning objects frame by frame based on those desired positions). This could definitely be improved. Other improvements could involve having **a better expert generation scheme** - my greedy strategy, while able to generate experts in a variety of distributions, may have not shown **all** the best ways to solve pin blocking removal - i.e. it may have missed on other parts of the solution space. Utilizing MCTS, or other policies to generate experts might give even more promising results. Yet another improvement could deal with the fine tuning of the AIRL model - this was not done in detail, and if more fine tuning could be done it's likely the model's performance could improve. The **HumanCompatibleAI Team** also argue that *variable sample sizes* are not good and utilizing static length samples that match the experts are desired[18] - this could be looked into as well. And of course the problem scope, size and impact could be extended. For example, the problem of varying action sizes, large changes in distribution (far larger than my changes in this toy problem), etc..

all are challenges of larger scope that I can see could continue to be tackled.

1.5.5 Connection to Related Work

As much of the recent literature has shown, DRL applications have been put to use in simulations and games in unique ways - such as with AlphaGo or with similar work in PCB Design - i.e. PPO applied to place PCB devices in a specific shape[2]. However, we have explored that AIRL can be added to this continuing list of approaches, as it seems to suggest that it can help generalize *across similar distributions*. This may provide a new avenue of research into its applicability into PCB Design.

1.6 Conclusions and Further Work

This work, which involved the combination of logical PCB objects within Drake's Physics, and the application of AIRL, was able to showcase that these newer **inverse RL technqiues** do seem to learn 'intent' *across differing distributions of PCB Boards*. This was showcased when models trained with experts of varying pin dimensions were able to solve the board **in less steps** than a random policy, *even as the boards were slightly different - both in initial position and in goal* from the experts that taught them. This indicated that the model had learned the 'intent' (the abstract problem goal) and was able to help generate a better **action planner** for the underlying **low level manipulation physics**. This led to a 'manipulation brain' that was robustly able to solve the 'PCB Pin Blocking Manipulation' toy problem that was setup for this paper in an improved form over random.

Time lost to build issues, as well as generally considering the scope as a final project, definitely left some possible future improvements. From improving the lower level physics to improving the expert generation beyond greedy strategies, tuning the AIRL model better, or handling variable paths, there's more legwork that could be done which may improve these results even more. On top of this there are even more challenging problems such as varying action sizes that still need to be tackled to continue the trend towards finding highly generalized models that are applicable in

the PCB Space. Tackling these problems will certainly require utilizing more models that are able to generalize in the way AIRL promises - to various dynamics and even varying action sizes. Unique combinations of approaches may be required, such as possibly applying transformers to predict sequences of possible actions that are larger than - but contain - the current known actions. Then combining methods like that with models such as AIRL - or improved variations of it like HAIRL or Variational AIRL.

At the very least, however, the work done showcases that this sort of model **can** be applied to PCB generation/modification and **these models** do seem to learn the 'intent' or idea. It also showcases how nicely *new and improving RL models* can continue to improve the 'brain' of the robots that utilize them. At the very least for me, it was a great learning experience to put it all together, and it showcased how exciting the field of RL and robotics can be when put to the test in real world problems!

Appendix A

Tables

Table A.1: Armadillos

Armadillos	are
our	friends

Appendix B

Figures

Figure B-1: Armadillo slaying lawyer.

Figure B-2: Armadillo eradicating national debt.

Bibliography

[1] What's the difference between pcb potting and conformal coating?, sep 2020.

[2] Peter Crocker. Physically constrained pcb placement using deep reinforcement learning. Jun 2020.

[3] Brian D. Ziebart et. al. Maximum entropy inverse reinforcement learning. *Twenty-Third AAAI Conference on AI*, 2008.

[4] Cameron Browne et al. A survey of monte carlo tree search methods. *IEEE transactions on computational intelligence and ai in games vol 4 no 1*, mar 2012.

[5] David Silver et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, jan 2016.

[6] David Silver et al. Mastering the game of go without human knowledge. *nature 550*, sep 2017.

[7] Maciej Swiechowski et al. Monte carlo tree search: A review of recent modifications and applications. *arXiv*, mar 2021.

[8] Sergey Levine Justin Fu, Katie Luo. Learning robust rewards with adversarial inverse reinforcement learning. *ICLR 2018 Conference and arXiv*, aug 2018.

[9] Sergey Levine. Cs 285 - deep reinforcement learning, 2021.

[10] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[11] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing, 2021.

[12] Yi Chen Mingqi Yuan, Man-on Pun. Hybrid adversarial inverse reinforcement learning. 2021.

[13] John R. Murphy. Neural network fitness function for optimization-based approaches to pcb design automation. *MIT Graduate Thesis*, 2020.

[14] Taylor Hogan Naveid Rahmatullah, Xiao-Ming Gao. Applications of ai and machine learning in interposer pcb design, mar 2019.

[15] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. Adversarial imitation via variational inverse reinforcement learning, 2019.

[16] Russ Tedrake. Robotic manipulation: Perception, planning, and control (course notes for mit 6.843), 2021.

[17] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[18] Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. https://github.com/HumanCompatibleAI/imitation, 2020.

[19] Zachary Zumbo. Genetic optimization applied to via and route strategy. *MIT Graduate Thesis*, 2020.