

# Map Coverage for Mobile Robot Implemented with Reinforcement Learning

by

Xue Xia

A dissertation submitted to the Graduate Faculty of  
Auburn University  
in partial fulfillment of the  
requirements for the Degree of  
Doctor of Philosophy

Auburn, Alabama  
August 8, 2020

Keywords: environmental complexity, CCPP, reinforcement learning, PPO, GAIL curiosity

Copyright 2020 by Xue Xia

Approved by

Thaddeus Roppel, Chair, Associate Professor of Electrical and Computer Engineering  
John Y. Hung, Professor of Electrical and Computer Engineering  
Shiwen Mao, Ginn Professor of Electrical and Computer Engineering  
Xiaowen Gong, Assistant Professor of Electrical and Computer Engineering

## Abstract

This dissertation introduces a measure of navigation complexity for complete coverage path planning (CCPP). It also introduces a novel approach to CCPP using reinforcement learning (RL) to enable a mobile robot to cover an area, subject to constraints on time and environmental complexity. Target applications are those with many repeated obstacles and hard time constraints, such as cleaning an airline cabin during the gate time between flights.

For navigation complexity measurement, the challenge is to consider various environmental factors together to measure the difficulty of navigating through the bounded space under consideration. For two-dimensional coverage problems, a low-complexity environment could be pictured as an area bounded by a rectangle or circle containing no obstacles, while a high complexity environment might be represented as a region bounded by a complex contour and filled with many randomly placed obstacles of random shapes and sizes.

The size of environments and the numbers of corners on maps are commonly taken into consideration. However, the size of robots to conduct CCPP navigation, the number of obstacles and the location of obstacles on maps are factors that cannot be ignored.

To address the aforementioned challenge, I propose a method consistent with the Shannon Entropy Formula. Four environmental factors are considered as inputs in our navigation complexity measurement. These four inputs include the spatial area to be covered, robot size, number of obstacles, and obstacle locations.

The experimental results show that our approach enables the computation of navigation complexity in a variety of environments, and that the results are intuitively consistent with human observation. This approach provides a comprehensive complexity measurement as a reference for CCPP performance analysis.

For map coverage using RL, the framework trains the robot in a simulated environment to move to uncovered areas and to avoid frequent collisions using rewards. Additionally, it encourages the robot to complete map coverage missions efficiently and quickly.

I select the Machine-Learning Agent provided by Unity3D to build a fragment (sample cell) of an airline cabin environment in which to train the robot. I implement Proximal Policy Optimization as the main training network, and added curiosity functions (i.e., intrinsic rewards) to encourage the robot to explore uncovered areas during training. I use Generative Adversarial Imitation Learning to guide the training policy’s convergence close to the expert data.

Experimental results show that the optimal policy enables complete map coverage in complicated environments. I provide demonstrations comparing random motion methods to reinforcement learning networks to show differences in map coverage, trajectory length, and time-cost.

## Acknowledgments

This work is supported in part by the US NSF under Grant ECCS-1923163, and through the RFID Lab and the Wireless Engineering Research and Education Center (WEREC) at Auburn University, Auburn, AL, USA.

## Table of Contents

Abstract . . . . .	ii
Acknowledgments . . . . .	iv
List of Figures . . . . .	vii
List of Tables . . . . .	x
1 Introduction . . . . .	1
2 Related Work . . . . .	5
2.1 Map Coverage Standards . . . . .	5
2.2 CCPP approaches . . . . .	6
2.2.1 Random Motion . . . . .	6
2.2.2 Decomposition . . . . .	7
2.2.3 SLAM based CCPP . . . . .	8
2.2.4 Machine Learning . . . . .	9
3 Software Introduction . . . . .	12
3.1 Robot Operating System . . . . .	12
3.2 Unity3D . . . . .	13
3.3 Tensorflow . . . . .	15
3.4 Matlab . . . . .	16
4 Theory . . . . .	17
4.1 Environmental Complexity Measurement . . . . .	17
4.1.1 Shannon Entropy . . . . .	17
4.1.2 Environmental Complexity Measurement . . . . .	18
4.1.3 The Process of Environmental Complexity Measurement . . . . .	21
4.2 Reinforcement Learning . . . . .	23

4.2.1	Cabin Environment Analysis . . . . .	23
4.2.2	Reinforcement Learning Selection from CCPP . . . . .	25
4.2.3	Reinforcement Learning . . . . .	27
4.2.4	Policy Gradient . . . . .	28
4.2.5	Generalized Advantage Estimator . . . . .	30
4.2.6	Proximal Policy Optimization . . . . .	31
4.2.7	Generative Adversarial Imitation Learning . . . . .	33
4.2.8	Curiosity-Driven Exploration . . . . .	36
4.3	Simulation Design . . . . .	39
4.3.1	Environmental Setting . . . . .	39
4.3.2	Simulation Setting . . . . .	42
4.3.3	Rewards Design . . . . .	48
4.3.4	Imitation Demo Recording . . . . .	51
5	Experimental Results . . . . .	55
5.1	Environmental Complexity Measurement . . . . .	55
5.1.1	Experimental results . . . . .	58
5.2	CCPP for Cabin Area using Reinforcement Learning . . . . .	61
5.2.1	Training Parameter Setting . . . . .	61
5.2.2	Scalar Analysis using TensorBoard . . . . .	64
5.2.3	Comparison between Reinforcement Learning and Random Motion Approaches . . . . .	65
5.2.4	Demo in Changed Environments . . . . .	73
6	Conclusion . . . . .	81
	Bibliography . . . . .	83

## List of Figures

3.1	Unity3D editor . . . . .	14
3.2	ML-Agent structure . . . . .	14
4.1	Shannon entropy vs. event probability . . . . .	18
4.2	Adjacent Cells . . . . .	20
4.3	Simulated Normalized Maps (a) 5 obstacles are set at the edge of the map. (b) 5 obstacles are set randomly on map. . . . .	23
4.4	Cabin Area in Simulation . . . . .	24
4.5	Decomposition cleaning area . . . . .	27
4.6	Clip Function Optimization . . . . .	32
4.7	The flow chart of GAIL using PPO . . . . .	36
4.8	The flow chart of ICM . . . . .	37
4.9	Cabin area simulation map . . . . .	40
4.10	Two layers of the simulation environments . . . . .	42

4.11	Training cells built by Unity3D . . . . .	43
4.12	Difference of training and testing plane coordinate range . . . . .	46
4.13	Training plane with medium coordinate range . . . . .	47
4.14	Narrow and Expanded testing plane coordinate range . . . . .	47
4.15	The flow chart of map coverage process . . . . .	49
4.16	Cleaning cell demo . . . . .	52
4.17	Cleaning cell demo . . . . .	53
5.1	Robot Platform . . . . .	56
5.2	Mock Mall . . . . .	56
5.3	Meeting Area . . . . .	57
5.4	RFID Office . . . . .	57
5.5	Mock mall maps . . . . .	59
5.6	RFID lab maps . . . . .	60
5.7	Meeting area maps . . . . .	60
5.8	Training statistics for clean cell using Tensorflow . . . . .	66
5.9	Training statistics for track Cell cell using Tensorflow . . . . .	67



5.10	Cleaning robot trajectories in simulation . . . . .	69
5.11	Map coverage and trajectories for one episode . . . . .	71
5.12	Cleaning robot trajectories in simulation . . . . .	72
5.13	Demos in Shape Changing Environment . . . . .	76
5.14	Demos in Expand Environment . . . . .	79
5.15	Demos in Expand Environment . . . . .	80

## List of Tables

4.1	Environmental Complexity of Matrix A and Matrix B . . . . .	22
4.2	Environmental Complexity for the Cabin Map in Simulation . . . . .	24
4.3	Training inputs . . . . .	48
4.4	Rewards for cleaning training . . . . .	49
4.5	Rewards for tracking training . . . . .	50
5.1	Environmental Complexity for Real-World Maps . . . . .	59
5.2	Training hyperparameters using PPO . . . . .	63
5.3	Training hyperparameters using curiosity and GAIL . . . . .	63

## Chapter 1

### Introduction

Complete coverage path planning (CCPP) is widely needed in various applications, such as geophysical surveying, and agriculture [1]. In recent years, CCPP is also used by a robot vacuum cleaner to clean floors. CCPP has three challenges for algorithm design. The first challenge is to guide a robot to traverse all empty areas within given boundaries (e.g. walls of a room). Algorithms are required to enable a robot to cover all areas. In addition, CCPP algorithms need to generate a path that excludes obstacle locations to prevent collisions. When a robot moves to the vicinity of obstacles, the robot needs to turn around to avoid collisions. However, when a robot turns around, its speed slows down. Hence, the rapid heading changing during map cleaning leads to time cost increasing. Third, CCPP algorithms are designed to optimize the total distance of the cleaning trajectory. Shortening the total distance improves the efficiency of complete coverage path planning.

Two aspects are considered to measure the performance of CCPP algorithms. The first is the coverage ratio Equation 1.1. Coverage ratio is the percentage of covered area to the full area of a given region. A CCPP algorithm is required to enable a robot to gain high coverage ratio. The second aspect is to measure the efficiency of CCPP algorithms. If the optimal trajectory is known for a given region, trajectory efficiency is the ratio of the experimental to the optimal trajectory distance Equation 1.2. However, the optimal trajectory is unknown in many cases. Then the efficiency is the value of the trajectory distance of one CCPP algorithm

to the trajectory distance of other CCPP algorithms that are used for comparison.

$$\text{Coverage Ratio} = \frac{\text{Covered Area}}{\text{Whole Region Area}} \quad (1.1)$$

$$\text{Trajectory Efficiency} = \frac{\text{Trajectory distance of tested approach}}{\text{Optimal trajectory distance}} \quad (1.2)$$

Simulation environments and real-world environments are built up to conduct testing for CCPP approaches. In order to analyze the performance of CCPP approaches, environmental map conditions are taken into consideration [2]. CCPP approaches perform differently on maps with various complexity levels. Previously, the size of testing environments is commonly measured. Area size decides the optimal trajectory length for robots to complete map coverage. Turns or corners are used to measure the complexity of maps. Robots need to avoid serious collisions when facing turns or corners. Hence, turns on maps will cause robots to slow down and turn around. A map with more turns will lead to high time costs for robots to complete map coverage. Map size and number of turns are taken into consideration for different CCPP approach performances. However, more environmental factors exist to influence the complexity of environments.

In this paper, a novel method is presented to measure the complexity of environments. The Shannon Entropy formula is considered. Robot size, obstacle number, and locations are taken into consideration to measure environmental complexity. Occupied maps are implemented in our method, and map regions are divided into grids or cells [3]. For each cell

of empty space on maps, the complexity is computed with its adjacent cells as inputs. Our method generates the complexity value using the Shannon Entropy formula.

In recent years, robot applications have been widely used to clean consumers' homes. To complete cleaning tasks, various algorithms are designed for CCPP in house and apartment environments. In this paper, we focus on solutions of CCPP in specific environments with many, similarly-shaped obstacles, such as industry factories, workshops, and airline cabins. In contrast to consumer applications, the time available for a robot to complete full coverage is limited.

The random motion algorithm is a low-cost, easily implementable method that enables autonomous robot navigation during consumer-orientation tasks, but fails to complete commercial or industrial missions within a limited time frame. For example, robots using the random motion algorithm tend to spend a large amount of their motion exploring in corners. As for robots using external sensors (i.e., Light Detection And Ranging (LIDAR) sensors or cameras), their ability to cover an area is based on a Simultaneous Localization And Mapping (SLAM) function. Environmental features are obtained through LIDAR, and maps are generated by SLAM. However, whenever environmental conditions change, the robot must rebuild a map of the surrounding environmental features before performing navigation. Because SLAM functions do not make good use of old versions of maps or pseudo-maps, it would be impractical to attempt to distinguish and match numerous similar features in a given area. Thus, robots enabled with SLAM functions are not suitable for our project.

In this work we select Reinforcement Learning (RL) methods to control ground robots in highly repetitive environments. We divide each environment into multiple sample cells according to pseudo-maps, e.g. airplane seat maps. We conduct the RL training process in a

single sample cell, and, thus, the problem of distinguishing repeated environmental features is avoided. The main purpose of the RL network design is to enable a robot to complete CCPP navigation, and this method provides positive and negative rewards during a robot's training to facilitate "learning" and optimize the training results.

Several challenges arise, however, while designing the RL training structure to enable CCPP navigation in the target environment. The robot expects to have a high coverage ratio using previously trained networks for navigation, to maintain a short total trajectory length with minimal repeated steps, and to complete missions quickly. Additionally, complicated target environments with many obstacles make robot movement difficult. Carefully selected hyper-parameters of the RL networks enable the robot's training to overcome these challenges.

The main contributions of this dissertation are as follows:

1. Compute navigation complexity using environmental factors, which include area size, robot size, obstacle number, and obstacle locations.
2. Provide a complexity measurement as reference for CCPP performance analysis.
3. Provide an optimal policy RL network that enables the autonomous and efficient navigation of robots that perform complete map coverage in complicated environments.

## Chapter 2

### Related Work

#### 2.1 Map Coverage Standards

According to [4, 2], there exist four main types of CCPP, including random motion approaches, template-based approaches, decomposition area approaches, and reinforcement learning approaches. Despite various algorithms and equipment implemented to robots, the exact environmental conditions have an influence on CCPP performance. From 1997, De Carvalho divided various environmental conditions into several templates and designed the robot to perform corresponding motions in [5]. Luo focused on generating CCPP path around obstacles, specifically corners or turns on maps [6]. Yang designed a CCPP algorithm that the robot was able to achieve collision-free performance around corners on maps [7]. Choi developed a path planning algorithm with obtaining information of adjacent cells [8]. Janchiv and the team conducted CCPP demonstrations using robot vacuum cleaners [9, 10]. The number of turns in trajectories were recorded for experimental analysis.

In previous works, the environmental complexity of mazes were discussed. Anthony focused on the patterns of mazes to study the topology [11]. Nalder introduced the Continuum theory [12]. Then McClendon calculated the complexity of mazes based on the Continuum

Theory [13]. In our paper, we measure the complexity of general environments, which include mazes and more. We applied the Shannon Entropy to our measurement with area size, robot size, obstacle size and obstacle locations as input factors.

## **2.2 CCPP approaches**

In CCPP progress, a robot is about to move to next location at each step. The selection of the next location determines the performance of CCPP algorithm approaches. CCPP approaches are classified into various types according to map representation. Some CCPP approaches rely on grid maps to cover the map. Map regions can be divided into grids or cells [3]. Clear grids stand for empty space and occupied grids stand for obstacle locations. The approaches perform map coverage with a provided map or building a map using SLAM function. Some other CCPP approaches conduct map coverage without map data.

### **2.2.1 Random Motion**

Y. Liu presented the random motion approach in which a robot enabled to complete coverage with random heading changes [14]. The robot moves forwards until collision occurs, then turns its heading with a random degree and continues moving forwards. Hasan selected bumper sensors for the low cost to obtain physical parameters, such as collision force, collision angle, and collision frequency for cleaning robots [15]. Based on the original random motion algorithms, Taylor made improvements that enabled the robot to estimate the size of given area based on the collision frequency [16]. Hence, the improved algorithm increased the region coverage of the cleaning robot implemented with random motion.



### 2.2.2 Decomposition

Trapezoidal decomposition relies on the vertices of polygonal obstacles to divide the given region into sub-areas [17]. In each sub area, a zigzag path is generated to cover the whole area. Each sub-area is represented as a node and all the nodes are connected to construct a topology graph. The robot moves to adjacent nodes based on the topology graph. Oksanen merged small sub-areas of the trapezoidal decomposition to improve the generated path [18]. Choset developed the boustrophedon cellular decomposition to split a given region into a set of sub-areas and enables the robot to complete map coverage in each sub-area [19]. Both trapezoidal decomposition and boustrophedon decomposition have similar sub-area splitting and sub-area coverage processes. Trapezoidal decomposition uses vertices of obstacles to divide sub-areas, while boustrophedon decomposition only uses the critical points of obstacles. Boustrophedon decomposition is unable to perform map coverage in an environment with non-polygonal obstacles. Milnor presented the basic theory of Morse decomposition [20]. Then Canny presented Morse decomposition by using the critical points of obstacle edges [21]. Later, various Morse functions were selected to divide cells into various shapes. H. Choset developed Morse decomposition with boustrophedon [22] while Acar generated spiral pattern cells based on Morse decomposition [23]. Hence, decomposition approaches build a clear structure for the CCP problem with topological nodes and subareas.

### 2.2.3 SLAM based CCPP

SLAM based approaches generate maps of the workspace [24]. Then they generate a trajectory (e.g. zig-zag) to cover the whole region. In [5], template coverage models are applied for different conditions, which include wall following and corners. CCPP algorithms are widely used on a 2D plane, however, in some cases CCPP is applied to 3D spaces [25]. Unmanned aerial vehicles (UAV) achieved developments in recent years and are widely used in 3D spaces [26, 27]. However, challenges still exist due to the complicated 3D motions and the complexity of 3D space [28, 29]. In this paper, we focus on ground robots as our selected platform for 2D plane navigation.

Montemerlo developed FastSLAM algorithm [30] and Bailey presented the extended Kalman filter SLAM (EKF-SLAM) [31]. The selection of external sensors influences the reliability of the generated map. External sensors, such as LIDAR, stereo and RGB cameras, or sonar sensors, are used to obtain information about the surroundings and various tags or patterns, such as image icons and RFID tags, can be applied to SLAM demonstrations as landmarks to guide robot navigation and localization [32, 33, 34, 35, 36, 37, 38, 39]. Many low-cost LIDAR sensors work on 2D plane to obtain obstacle distance and angles with high accuracy [40]. Kohlbrecher presented LIDAR based SLAM in indoor environments [41]. Taking advantage of the stable performance of LIDAR under changing scene illumination, James performed LIDAR based SLAM in forests [42]. Cameras are widely used for Visual SLAM functions. Paz performed visual SLAM using stereo cameras [43]. Microsoft put Kinect into the market as one type of low-cost RGB-D camera. Equipped with the affordable

Kinect, RGB-SLAM was presented by Newcombe [44]. After the stereo camera and RGB-D camera were selected to perform Visual SLAM, LSD-SLAM using a monocular camera was presented [45]. Then state-of-art Visual SLAM algorithms, such as RTAB and ORB-SLAM were developed [46, 47]. Also, in order to take advantage of the two types of sensors, LIDAR and a RGB-D camera were both equipped on a robot to perform SLAM function and generate maps [48].

#### **2.2.4 Machine Learning**

Machine learning is classified into three types: supervised learning, unsupervised learning, and RL (reinforcement learning) [49]. Supervised learning, such as behavior clone [50], requires large amounts of sample data, and those samples are used to measure training performance. Unsupervised learning methods are used for grouping clusters from sets of data [51]. Previous machine learning methods, such as Genetic Algorithm and the Ant Colony Optimization algorithm, were used to generate optimal CCPP routes [52, 53]. Alternatively, deep learning has been used for various applications [54, 55, 56]. RL has been highly developed in recent years, and we applied it for CCPP in this project.

In 2013, DeepMind presented an Atari playing model using Deep Q-learning Network (DQN) training [57]. DQN only supports discrete outputs such as Atari control commands, while more applications in practice require outputs with continuous values. In 2014, DeepMind designed Deterministic Policy Gradient (DPG) algorithm based on Policy Gradient (PG) for continuous action control [58, 59]. In 2015, Schulman of UC Berkeley presented True Region Policy Optimization (TRPO) to limit the update degrees of policy [60]. Radford designed Generative Adversarial networks (GAN) [61]. GAN enables competitive training

to improve the quality of data from a generator to pass the threshold of a discriminator with neural networks. In 2016, DeepMind merged the concept of DQN with continuous action control and named the algorithm Deep Deterministic Policy Gradient (DDPG) [62]. To reduce the time required for training, DeepMind developed asynchronous methods for parallel learning agents. Additionally, they applied these methods on an actor-critic model as Asynchronous Advantage Actor-Critic (A3C) to shorten the training time of the Atari playing model significantly [63]. In the same year, Ho presented Generative Adversarial Imitation Learning (GAIL) based on GAIL [64]. Reinforcement learning algorithms are implemented to generator updating in GAIL. In 2017, Pathak designed intrinsic rewards for training process [65]. The intrinsic rewards provides curiosity driven to the agent to encourage exploration motions. Schulman got inspiration from TRPO and presented Proximal Policy Optimization (PPO) algorithm [66]. PPO simplified the equations of TRPO and improved the performance. Microsoft performed the model Hybrid Reward Architecture (HRA) by training parallel agents with separated reward values and functions to archive the maximum score of Atari PacMan [67]. To move the training environment from the 2D Atari game platform to a 3D real-world environment, Li Fei-Fei from Stanford University performed target-driven robot navigation with visual sensors using A3C [68]. AI2-THOR was developed based on the Unity3D engine as the 3D environment emulator for visual AI simulation. The target-driven model enabled navigation to new targets based on trained targets and to navigate in a new scene while observing trained targets [69, 70]. In 2018, Unity3D company developed a plugin named Machine Learning Agents (ML-Agents) [70]. ML-Agents provides an environment for machine learning training. In the same year, Socially Aware Collision Avoidance with Deep Reinforcement Learning (SA-CADRL) was developed as a

dynamic obstacle avoidance method and made to perform hardware experiments while the robot navigated crowds of people [71].

## Chapter 3

### Software Introduction

Various types of software tools and platforms are used in this project for simulation and analysis. Robot Operating System (ROS) and Matlab are applied to environmental complexity measurement. ROS provides gmapping open-source codes and motion driver open-source codes of Create 2 robot for 2D mapping. Matlab is used to compute pixel values of generated maps to measure environmental complexity. In CCPP using reinforcement learning part, Unity3D Editor is selected as the simulation platform for scene building. Unity3D provides the interface Tensor ow to enable reinforcement training and testing. Tensor ow generates networks as optimal results after training, and networks can be used for performance testing. Matlab computes and plots figures with data recorded in testing.

#### **3.1 Robot Operating System**

Robot Operating System (ROS) is a popular platform for robot application design. As explained in [72], ROS is more like a communication structure that enables multiple hardwares or machines work through network. ROS supports various program languages that includes C++, Python, Octave, and LISP. The two main common used languages on ROS are C++, Python. A large amount of tools are provided to run ROS components. Hence, developers from all over the world are able to provide libraries and packages to support robot

functions. Re-usable packages that include SLAM and navigation and motions drivers can be found on ROS. As an open-source platform, ROS is free and mainly runs on Linux.

## 3.2 Unity3D

Unity3D is a commercial game engine and typically used as a tool for game development. The screenshot of the Unity3D Editor is shown in Figure 3.1. In 2018, the Unity3D company developed a Machine Learning Toolkit named ML-Agent With Unity3D platform [73]. Unity3D provides various simulation sensor functions, such as Raycast, a circle of rays used for LIDAR sensor simulation. Additionally, Unity3D supports image renders and camera objects that can be used for Kinect or stereo-camera simulation. In Unity3D, agents or characters and environmental models are all named "game objects". Based on the size and shape of objects, simulated physical interactions are supported by Unity3D using Rigidbody functions. Nvidia PhysX is a built-in component as a physical engine in Unity3D. Unity3D also provides other third-party plugin physical engines, such as Bullet and Mujoco. Language C# is used for script programming, and various scripts are added to game objects to construct simulation environments. Scripts can send commands to agents and describe complicated interactions to build custom simulation environments. Additionally, Unity3D provides recorder functions using C# scripts to record demos as expert trajectories for imitation learning. In our project, a single agent is set for simulation. However, Unity3D enables multiple agents in an environment for training and testing.

Based on Unity3D environments, ML-Agent provides a SDK to build custom agents and environments for reinforcement learning and interfaces to communicate with Python packages that are outside Unity3D Editor. The SDK includes three components that are

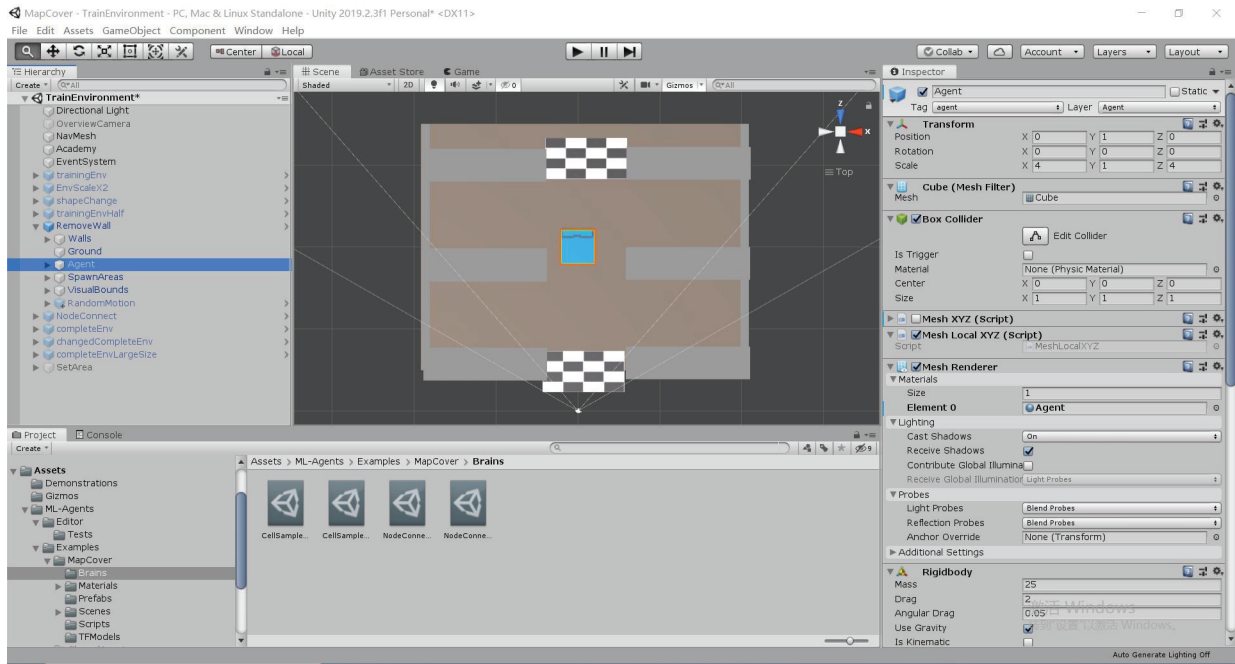


Figure 3.1: Unity3D editor

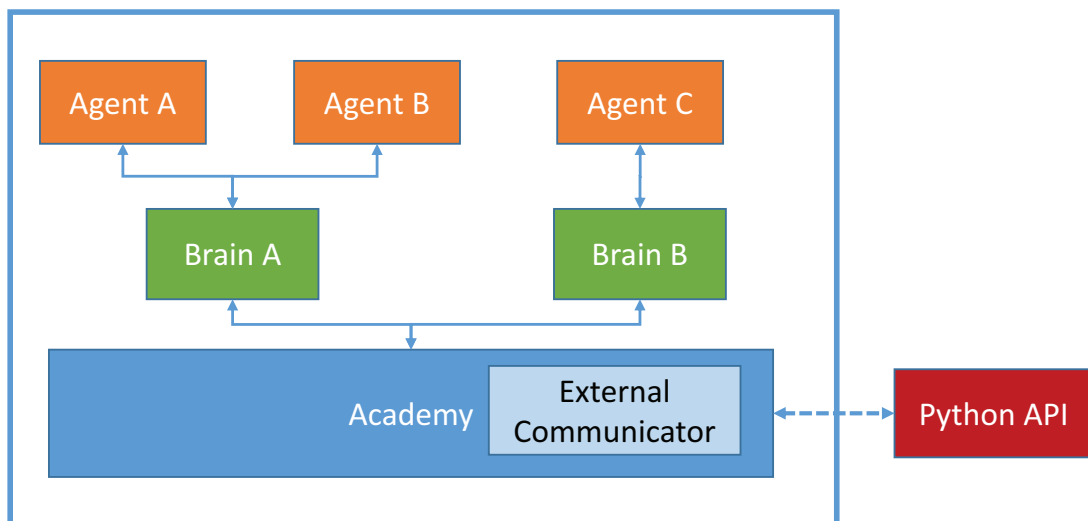


Figure 3.2: ML-Agent structure



agent, brain and academy, and the structure of ML-Agent is shown in Figure 3.2. The agent component is for the agent in simulation environment. Agent scripts send observations that can be vectors or images as inputs for training and receive motion commands as training outputs. When the agent has some specific behaviors to get rewards, agent scripts add reward to networks. Every agent is linked with a brain. As shown in Figure 3.2, multiple agents can be linked to the same brain and agents that are linked with different brains can be trained in the same environment. The agent and the linked brain have the same number of inputs and outputs. The brain component contains policy and makes decisions for agent actions. After training, Tensor ow generates optimal network files and the network files are sent to brains for testing. The academy component controls environmental settings. Simulation speed or framerate are also set by the academy component. Tensor ow provides open-source reinforcement learning packages outside the Unity3D environment with Python. Variables and parameters inside Unity3D are sent through interfaces to Tensor ow. Interfaces connect the Unity3D environment in C# with Tensorflow in Python.

### **3.3 Tensorflow**

Tensorflow is an open source library developed by Google Brain [22]. Machine learning algorithms are built in Tensor ow and are widely implemented by various applications. Tensorflow provides the TensorBoard function that stores statistical data and plots scalar figures for the training process.

### 3.4 Matlab

Matlab provides functions for data analysis and figure plotting and takes advantages of large scale matrix computing [74]. Matlab is user friendly and easy for coding. In addition, Matlab provides interfaces that enable C/C++ codes to be used in the Matlab environment and Matlab code to be used in the C/C++ environment. A large number of 3rd party plugins are implemented in Simulink. Hence, researchers from science and engineering fields select Matlab for model-based design and software simulation.

## Chapter 4

### Theory

#### 4.1 Environmental Complexity Measurement

##### 4.1.1 Shannon Entropy

Shannon Entropy is a fundamental concept of information theory [75]. As explained in [76], information records various events or data and can be described with variables. In other words, variables carry information through data transmission. The word entropy is used for measuring the amount of information. During information transmission, some events are predictable while others tend to be difficult to guess if they will occur or not. When the event is difficult to guess, the amount of information is high. Hence, the events that are hard to predict contain a large amounts of information and the entropy is high.

Equation 4.1, known as the Shannon Entropy Equation, expresses the amount of uncertainty in information. The parameter  $b$  is the logarithm base, and usually  $b$  is set to 2. For each piece of data  $\{d_i = 1, 2, \dots, n\}$ ,  $p_i$  represents the probability that the event  $i$  will occur. In Bernoulli process,  $n = 2$ . When the event is certain to happen, the uncertainty of information is none, so  $p_i = 1$ ,  $H(X) = 0$ . When the event is certain not to happen, the uncertainty of information is none, so  $p_i = 0$ ,  $H(X) = 0$ . In the two cases, the events are predictable and the uncertainty of information does not exist. When the event is uncertain

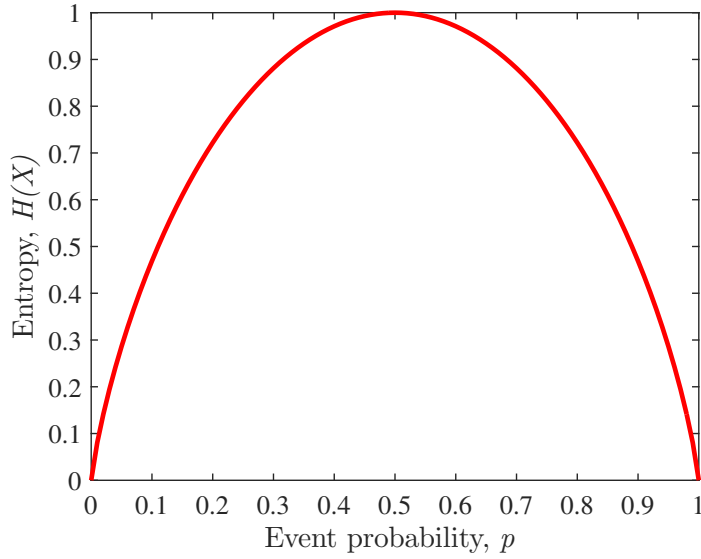


Figure 4.1: Shannon entropy vs. event probability

to occur,  $p_i$  is not equal to 1 or 0, the uncertainty of information grows and entropy value increases. When the event is about to occur randomly, the uncertainty of information reaches the maximum value. When  $p_i = 1 - p_i = 0.5$ ,  $H(X) = 1$ . When  $n = 2$ , the relation between  $p_i$  and  $H(X)$  is shown in diagram Figure 4.1.

$$H(X) = - \sum_{i=1}^n p_i \log_b(p_i) \quad (4.1)$$

#### 4.1.2 Environmental Complexity Measurement

When a robot performs complete coverage in practical environments, different standards exist to measure mission completion. In order to analyze the CCPP performance in complex environments, a measure of complexity is required. In our project, the information is known. Hence, the Shannon Entropy is used to measure the diversity of environmental information.

Shannon Entropy is proposed herein to measure environmental complexity. Grids or cell maps are widely used to represent environments. Clear grids mean empty area which is safe for robots to move around. Occupied grids are the locations of obstacles and walls. Robots are required to move away from occupied areas to avoid collisions. The adjacent area is divided into 8 cells during robot navigation, as shown in Figure 4.2. The red cell at the center of Figure 4.2 represents the location of the robot. The 8 white cells are the adjacent area based on the robot location. The number of obstacles determine the difficulty of robot navigation. If obstacles are present in all 8 adjacent cells, the robot is unable to generate a path to move out. Hence, the complexity of environments is defined as 0 for cases in which there is no path to move. In another case, if no obstacles exist in the adjacent cells, the robot can move in any direction without collision. Therefore the complexity of an empty surrounding environment is defined as 0 as well. In other cases, empty space and obstacles both exist in 8 adjacent cells. The robot is required to move carefully to avoid potential collisions, so the complexity of the environment has evidently increased. So the obstacle distribution in the adjacent area decides the environmental condition with the robot current location. The Shannon Entropy will use the diversity of obstacle location to measure the environmental complexity.

In order to provide a normalizing scale for the calculation of environment complexity, the size of map cells or grids is chosen to be the same as the size of the robots. In addition, the size and locations of the obstacles in an environment is normalized by the size of the robots. Four factors, which include robot size, environment size, obstacle size, and obstacle locations, are taken into account for calculating. To calculate the exact value of environment

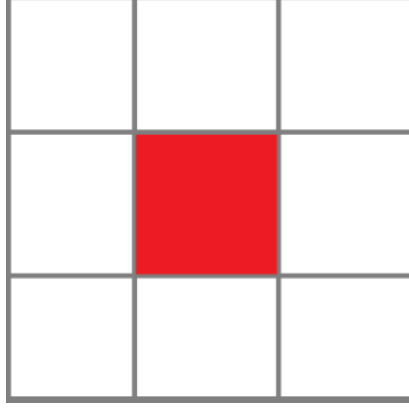


Figure 4.2: Adjacent Cells

complexity, the numbers of empty cells and occupied cells are divided by 8 because, there are 8 grids or cells in an adjacent area.

$$p_{empty} = \frac{\text{Number of empty cells}}{8} \quad (4.2)$$

$$p_{occupied} = \frac{\text{Number of occupied cells}}{8} \quad (4.3)$$

$$H_{adjacent\ cell} = -p_{empty} * \log_2(p_{empty}) - p_{occupied} * \log_2(p_{occupied}) \quad (4.4)$$

The variable  $p_{empty}$  represents the percentage of empty cells to all the 8 adjacent cells in Equation 4.2 and  $p_{occupied}$  represents the percentage of occupied cells to all the 8 adjacent cells in Equation 4.3. The sum of  $p_{empty}$  and  $p_{occupied}$  of the same adjacent cells equals to 1. Hence, the complexity value of adjacent area is computed as Equation 4.4.

### 4.1.3 The Process of Environmental Complexity Measurement

The left and right figures in Figure 4.3 simulate maps with the same robot size, same area size, same obstacle number, but different obstacle locations. The robot size is assumed as the same size of map cells. The total number of cells are 25 on both Figure 4.3.(a) and Figure 4.3.(b). In the Figure 4.3.(a) all obstacles located at the edge of the map. In the Figure 4.3.(b) the obstacles located randomly on the whole map.

The maps of Figure 4.3.(a) and Figure 4.3.(b) are abstracted to  $Matrix_A$  and  $Matrix_B$  in Equation 4.5 and Equation 4.6. In addition, the complexity value to each free space cell is computed using Equation 4.4 with its 8 adjacent cells. The results are shown in Equation 4.7 and Equation 4.8. Both maps have the same size, so that the same type of robots focus on the same size of CCPP missions. The complexity value which are the sum of all elements in  $Matrix_{Adjacent A}$  and  $Matrix_{Adjacent B}$  are shown in Table 4.1. According to Table 4.1, the complexity value of  $Matrix_B$  is significantly higher than  $Matrix_A$ , which means that the environmental condition for  $Matrix_B$  is more complicated than  $Matrix_A$ .

$$Matrix_A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.5)$$

Table 4.1: Environmental Complexity of Matrix A and Matrix B

<b>Matrix A</b>	<b>Matrix B</b>
4.4859	11.1065

$$Matrix_A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.6)$$

$$Matrix_{Adjacent\ A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.8113 & 0.9544 & 0.9544 & 0.9544 & 0.8113 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.7)$$

$$Matrix_{Adjacent\ B} = \begin{bmatrix} 0 & 0.5436 & 0 & 0.5436 & 0 \\ 0.8113 & 0.8113 & 0.8113 & 0.8113 & 0.8113 \\ 0.5436 & 0 & 0.8113 & 0 & 0.8113 \\ 0.5436 & 0.5436 & 0.8113 & 0.8113 & 0 \\ 0 & 0 & 0 & 0.5436 & 0.5436 \end{bmatrix} \quad (4.8)$$



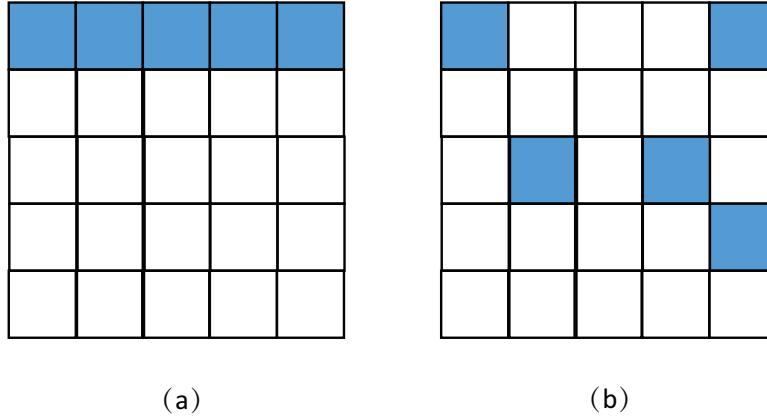


Figure 4.3: Simulated Normalized Maps (a) 5 obstacles are set at the edge of the map. (b) 5 obstacles are set randomly on map.

## 4.2 Reinforcement Learning

### 4.2.1 Cabin Environment Analysis

Unlike the environmental settings of houses or offices, the cabin area has a more specific environmental condition. In the previous chapter, we analysis the factors to generate a complicate environment and we define the concept of environmental complexity. Then we applied the measurement to measure the environmental complexity of the cabin area.

In the previous chapter, the 2D map building was conducted in real-world. However, the cabin environment for this project is built using Unity3D. The coordinates were recorded in .txt files and Matlab was used to plot the cabin map. The Figure 4.4 is the figure of the simulation cabin area. The blue area represents the obstacles and the white area is the free space. The size of 4.4(a) is  $708 \times 108$ . The robot size in pixel units is  $12 \times 12$ . Then the cabin map was normalized by the  $12 \times 12$  cell and the normalized map is shown as 4.4(b). The normalized map was used to compute the environmental complexity. 4.4(c) is the complexity map. The sum of the environmental complexity is 257.8176. After divided by the normalized

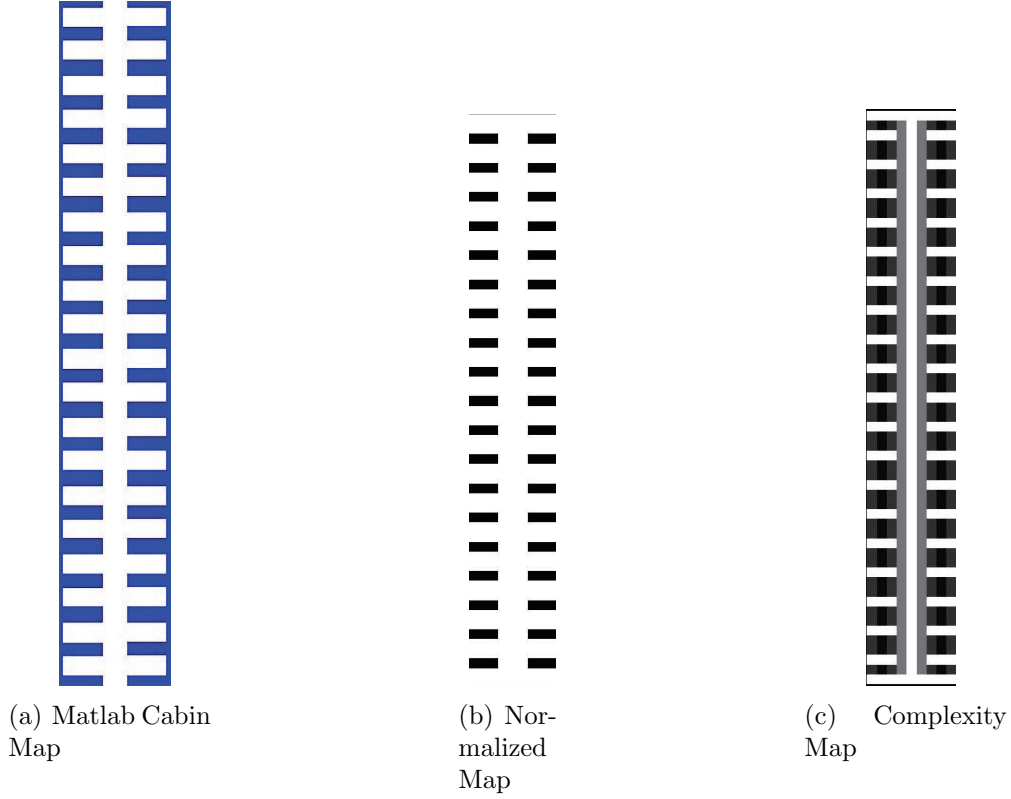


Figure 4.4: Cabin Area in Simulation

map size  $59 \times 9$ , the environmental complexity value of the simulating cabin area is 0.4855.

All the map parameters and values are stored in the Table 4.2.

The environmental complexity values show the a large difference of the environmental conditions between the offices and the cabin area. As mentioned in the later parts, the 3 real-world maps gain complexity value no more than 0.3. The complexity value of the cabin

Table 4.2: Environmental Complexity for the Cabin Map in Simulation

	<b>Cabin Area in Simulation</b>
<b>Original Size</b>	708x108
<b>Normalized Size</b>	59x9
<b>Total sum</b>	257.8176
<b>Total sum/ Normalized Size</b>	0.4855

area is much higher and the environmental condition of the cabin area is more complicate. Hence, we need to select proper approaches to accomplish the map coverage mission of this project.

#### **4.2.2 Reinforcement Learning Selection from CCP**

Random motion approaches are low-cost and easily deployed in consumer applications. When a robot uses random motion algorithms, it generates trajectories randomly to completely cover the map. To cover the whole area of the map, trajectories repeat patterns several times. Hence, the cost in time for random motion algorithms to complete coverage is high. Additionally, there are usually no external sensors mounted to these robots in order to reduce cost to the consumer. However, when these robots are trapped in a corner or small, crowded area, it is difficult for them to find a way out without obtaining additional information about their surrounding environment. For consumers using robots in the home, these two shortcomings are generally not regarded as problems. These robots often perform tasks that are not time-critical to the consumer, so it is acceptable that a robot requires several hours to complete its tasks. Additionally, furniture placement in most apartments and houses does not usually create environments with a lot of corners that trap robots. However, when robots are used in other environments, such as warehouses and airline cabins, these navigational shortcomings cannot be ignored. For this project, we focused on reducing coverage time and we explored a suitable approach for obtaining complete map coverage.

SLAM-based approaches for autonomous robot navigation are generally superior to random motion approaches. The improved performance relies on the quality of map building, as a reliable map enables the robot to separate covered areas from uncovered areas within

a given environment and to complete map coverage with just a few repeating trajectories. However, reliable maps require information about environmental features. For this project, we used an airline cabin, which has many repeated features, as the target environment. Repeated features contribute time-costs and produce unrecognizable information and disturbances during map building and robot localization. Because time is limited to perform the cleaning tasks, a robot needs to cover the map with a suitably high coverage ratio, but possibly less than 100% coverage. For SLAM-based approaches, the robot covers all the edges and corners of irregular obstacles in order to complete coverage as perfectly as possible. While SLAM-based approaches have high performance, airline cabin cleaning requires an approach that balances acceptable coverage and time-costs.

The method described in this dissertation is inspired by decomposition based approaches. Figure 4.5 shows the divided sub-areas using decomposition approaches. Decomposition approaches split area according to vertex or points of obstacles, such as when there is furniture in a room. In 4.5(a), the black edges are four walls and the blue circle and the rectangles represent obstacles. For many house cleaning cases, the distribution of furniture is sparse in a room, and the number of sub-areas remains low. The robot is capable of performing cleaning in one sub-area and moving to the next. However, the decomposition process is impacted adversely in complex environments with high-density obstacles. Figure 4.5(b) is a sample for cabin area decomposition. The sub-areas in 4.5(a) are tiny. It is meaningless for a robot to perform cleaning within each sub-area in 4.5(b). Hence, we plan to define the size of decomposition sub-areas.

Reinforcement learning algorithms train a robot to perform CCPP. A training environment (i.e., airline cabin) is selected for a robot to gain specific policies and strategies for

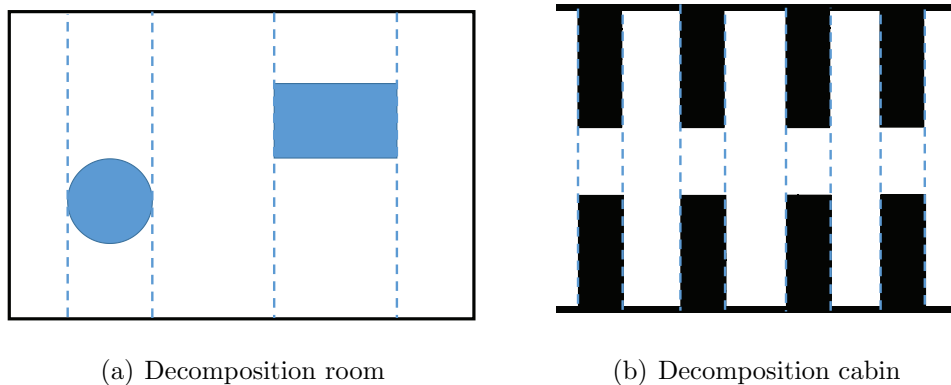


Figure 4.5: Decomposition cleaning area

navigation. A corresponding behavior model is obtained from the training process. During training iterations, rewards guide network convergence to achieve complete coverage; positive rewards are given for cleaning new areas, while negative rewards are given for obstacle collisions and inefficient trajectories. RL algorithms build CCPP strategies in training rather than during testing. The training process of RL takes on the computing burden for CCPP performance to generate optimal policy networks. When the robot performs CCPP in testing, motion commands are sent from these networks without large computational resources.

### 4.2.3 Reinforcement Learning

As one of the branches of machine learning approaches [77], RL focuses on generating specific outputs for a given mission while using rewards to improve performance during training. There are several elements in RL approaches [78]:

Environment: The place built for training or testing.

Agent: The players in the environment that complete tasks.

Observation: The players observe the environment to get information about the surroundings.

Action: The agent takes action while in the environment. The agent takes random actions at the beginning of the training, while the efficiency of actions improves during training. At time  $T = t$ , the agent takes action  $a_t$ .

State: For each step, the states include environmental conditions and agent actions. State information is used by RL approaches to improve performance during training. At time  $T = t$ , the state of agent is  $s_t$ .

Reward: Positive rewards are given to an agent for actions that contribute towards mission completion, while negative rewards are given for actions that should be avoided. At time  $T = t$ , the agent gains current reward  $r_t$ .

#### 4.2.4 Policy Gradient

Policy gradient (PG) methods are a type of reinforcement learning approaches. Actions are selected through probabilities and the probabilities of actions are generated by gradients. In order to gain optimal policy through training, PG approaches use the gradient of the loss function Equation 4.9 to update the policy. Gradient estimators calculate gradients by differentiating loss functions shown in Equation 4.9. Expectation  $E_t[\cdot]$  computes the mean value of samples. The policy  $\pi_{\theta_P}$  for action probabilities converges after numbers of training iterations and  $\theta_P$  is the corresponding parameter. The policy  $\pi_{\theta_P}(a_t|s_t)$  represents the policy when the agent takes actions  $a_t$  at state  $s_t$ .  $A_t$  is the advantage function at time  $T = t$  and  $A_t$  values futures reward sum in Equation 4.10. In Equation 4.10  $V(s_t)$  is the value function and  $Q(s_t, a_t)$  is the Q function from Q-learning.  $V(s_t)$  calculates the sum of future rewards at state  $s_t$  from time  $T = t$  in Equation 4.11.  $\gamma$  is the reward discount and  $R_t$  is the gained reward at time  $T = t$ . The equation of Q function is similar to value function in

Equation 4.12. Despite, Q function computes the sum of future rewards at the state  $s_t$  with action  $a_t$  taken by the agent. Advantage function is the difference from that  $Q(s_t, a_t)$  minus  $V(s_t)$ . Advantage function is used to measure the future trends after current actions. When the training performance is better than the estimation, the gradients are positive and the policy updates to increase the probabilities of corresponding actions. When the training performance goes worse than the estimation, the gradients are negative and the policy updates to decrease the probabilities of corresponding actions. The policy converges to optimal results through training iterations. Policy gradient enables continuous actions as outputs for players. Continuous actions include ground robot headings. Ground robots can have headings from 0-360 degrees. In contrast, TV game main characters move around using UP, DOWN, LEFT, and RIGHT keys. The four keys correspond to limited number of actions as an example of discrete actions. Hence, PG methods were selected for ground robot map coverage.

$$Loss = E_t[\log \pi_{\theta_P}(a_t|s_t)A_t] \quad (4.9)$$

$$A_t = Q(s_t, a_t) - V(s_t) \quad (4.10)$$

$$V(s_t) = E_t\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t\right] \quad (4.11)$$

$$Q(s_t, a_t) = E_t\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t, a_t\right] \quad (4.12)$$

### 4.2.5 Generalized Advantage Estimator

Generalized advantage estimator (GAE) improves the advantage function to decrease the variance of gradient estimators [79]. Hence, the number of training samples are reduced for policy gradient methods. From the original advantage function in Equation 4.10,  $\hat{A}_t^{GAE(\lambda,\gamma)}$  is shown in Equation 4.13, where  $\delta$  is defined in Equation 4.14. When  $\lambda = 0$ ,  $\hat{A}_t^{GAE(0,\gamma)}$  is shown in Equation 4.15 and equal to  $\delta_t$ .  $\hat{A}_t^{GAE(0,\gamma)}$  has a low variance but the estimation bias can not be ignored. When  $\lambda = 1$ ,  $\hat{A}_t^{GAE(1,\gamma)}$  is shown in Equation 4.16.  $\sum_{k=0}^{\infty} (\gamma)^k \delta_{t+k}$  leads to high variance of  $\hat{A}_t^{GAE(1,\gamma)}$  and precise prediction of  $V$ . Hence, the parameter  $\lambda$  is at a range of  $(0, 1)$ , and the value of  $\lambda$  keeps a balance between sample variance and prediction bias.

$$\hat{A}_t^{GAE(\lambda,\gamma)} = \sum_{k=0}^{\infty} (\lambda\gamma)^k \delta_{t+k} \quad (4.13)$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (4.14)$$

$$\hat{A}_t^{GAE(0,\gamma)} = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (4.15)$$

$$\hat{A}_t^{GAE(1,\gamma)} = \sum_{k=0}^{\infty} (\gamma)^k \delta_{t+k} - V(s_t) \quad (4.16)$$



### 4.2.6 Proximal Policy Optimization

Proximal policy optimization (PPO) is a state-of-the-art policy gradient algorithm which is improved from the original policy gradient method. It outperforms other policy gradient approaches [66]. For previous policy gradient approaches, training failures occur when the policy change is large. In this case, policy updating leads to lack of convergence. Trust region policy optimization (TRPO) is focused on decreasing the updating steps to avoid convergence failures during the training process [60]. TRPO restrains updating steps to ensure that updated policy does not move far away from old versions of policy. Following the same concepts of updating steps of TRPO, PPO simplified the equations from TRPO and uses a clipping function to achieve better performance than TRPO.

$$r_t(\theta_P) = \frac{\pi_{\theta_P}(a_t|s_t)}{\pi_{\theta_{P_{old}}}(a_t|s_t)} \quad (4.17)$$

PPO defines the probability ratio in Equation 4.17 to measure the policy update degree.  $\pi_{\theta_{P_{old}}}(a_t|s_t)$  represents for old version of policy and  $\pi_{\theta_P}(a_t|s_t)$  stands for new version of policy. When  $\theta_P = \theta_{oldP}$ , which means the new version of policy is the same as the old version,  $r_t(\theta_P) = \frac{\pi_{\theta_P}(a_t|s_t)}{\pi_{\theta_{P_{old}}}(a_t|s_t)} = 1$ . Inspired by TRPO, the  $\log()$  function in PG loss function in Equation 4.9 is replaced by the probability ratio and modified to Equation 4.18, where CPI represents conservative policy iteration [80]. Then PPO adds clip constraints to  $L^{CPI}$  to avoid large steps of policy updating in Equation 4.19.

$$L^{CPI} = E_t\left[\frac{\pi_{\theta_P}(a_t|s_t)}{\pi_{\theta_{P_{old}}}(a_t|s_t)}A_t\right] = E_t[r_t(\theta_P)A_t] \quad (4.18)$$

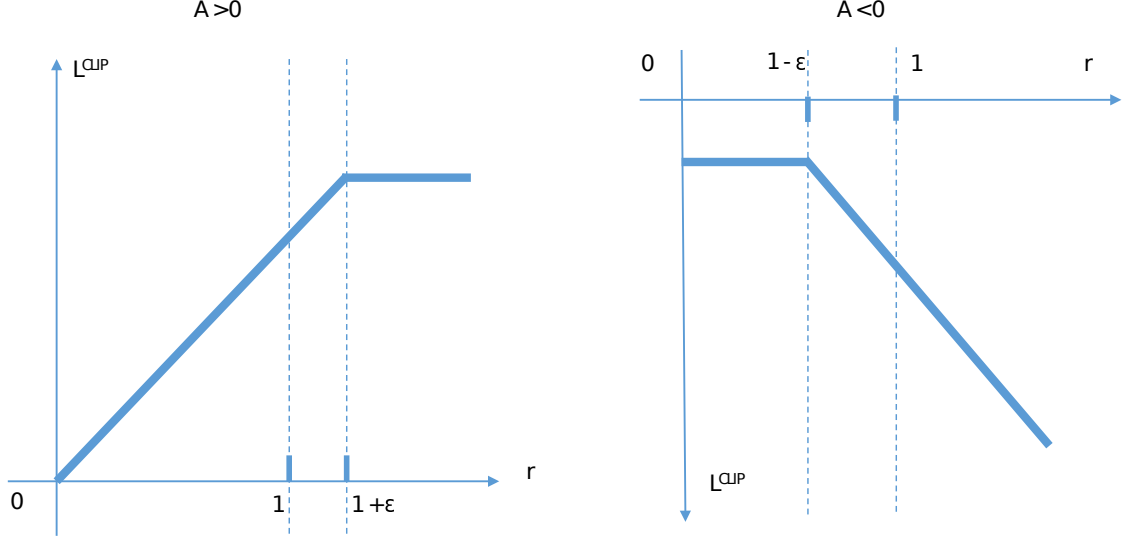


Figure 4.6: Clip Function Optimization

$$L^{CLIP} = E_t[\min(r_t(\theta_P)A_t, \text{clip}(r_t(\theta_P), 1 - \epsilon, 1 + \epsilon)A_t)] \quad (4.19)$$

In  $L^{CLIP}$  function,  $\epsilon$  is a hyperparameter and set  $\epsilon = 0.2$ . In the bracket of  $\min()$ , the two elements are  $r_t(\theta_P)A_t$  and the clip function. The first element comes from the  $L^{CPI}$  function and the second term provides boundaries using  $1 - \epsilon$  and  $1 + \epsilon$  to clip  $r_t(\theta_P)$  value. The minimum output is selected between  $r_t(\theta_P)A_t$  and the clipping function to restrain large policy updating. In Figure 4.6 the diagrams show the relation between  $r_t(\theta_P)$  and clipping function. The left part is for  $A_t > 0$  and the right part is for  $A_t < 0$ .

When  $A_t > 0$ , the boundaries of clipping function are  $[0, 1 + \epsilon]$ . When the probability ratio  $r_t(\theta_P) < 1 + \epsilon$ , the minimum function selects  $r_t(\theta_P)$  as outputs and  $\min(r_t(\theta_P)A_t, \text{clip}(r_t(\theta_P), 1 - \epsilon, 1 + \epsilon)A_t) = r_t(\theta_P)A_t$ . When the probability ratio  $r_t(\theta_P) > 1 + \epsilon$ , the clipping function clips the value of  $r_t(\theta_P)$  to  $1 + \epsilon$ . Then the minimum function selects  $1 + \epsilon$  as outputs and  $\min(r_t(\theta_P)A_t, \text{clip}(r_t(\theta_P), 1 - \epsilon, 1 + \epsilon)A_t) = (1 + \epsilon)A_t$ . The largely positive policy update has

been avoided by clipping. When  $A_t < 0$ , the boundaries of clipping function are  $[1 - \epsilon, +\infty]$ . When the probability ratio  $r_t(\theta_P) < 1 + \epsilon$ , the clipping function clips the value of  $r_t(\theta_P)$  to  $1 + \epsilon$  and the minimum function selects  $1 - \epsilon$  as outputs and  $\min(r_t(\theta_P)A_t, \text{clip}(r_t(\theta_P), 1 - \epsilon, 1 + \epsilon)A_t) = (1 - \epsilon)A_t$ . When the probability ratio  $r_t(\theta_P) > 1 + \epsilon$ , the minimum function selects  $r_t(\theta_P)$  as outputs and  $\min(r_t(\theta_P)A_t, \text{clip}(r_t(\theta_P), 1 - \epsilon, 1 + \epsilon)A_t) = r_t(\theta_P)A_t$ . The probability ratio for negative policy update has been kept a remain value of  $1 - \epsilon$ . According to Figure 4.6, clipping functions restrain positive policy update with an upper bound  $1 + \epsilon$ , while clipping functions allow largely policy update with negative  $A_t$ . Equation 4.20 builds the loss function for PPO, where  $L_t^{CLIP}(\theta_P)$  is the CLIP loss function,  $L_t^{VF}(\theta_P)$  is a squared-error loss and  $S$  represents entropy bonus. The squared-error loss is defined as  $L_t^{VF}(\theta_P) = (V_{\theta_P}(s_t) - V_t^{target})^2$ .  $c_1$  and  $\beta$  are coefficients.

$$L_t^{CLIP+VF+S}(\theta_P) = E_t[L_t^{CLIP}(\theta_P) - c_1 L_t^{VF}(\theta_P) + \beta S[\pi_{\theta_P}](s_t)] \quad (4.20)$$

#### 4.2.7 Generative Adversarial Imitation Learning

Reinforcement learning approaches gain optimal policy through training process. However, sometimes iterations are unable to convergence to optimization results when complicated missions are assigned to training process. In this case, the fact that the success rate during training process is low leads to training failures. In order to accomplish training with complicated missions, imitation learning methods are considered. As the name implies, imitation learning provides expert demonstrations as reference, and the training process of reinforcement learning converges to the expert demonstrations.

Unlike the behavioral cloning method, which requires large amounts of data for training [81], generative adversarial imitation learning (GAIL) performs successful training with only several or dozens of expert demonstration episodes. Improved from inverse reinforcement learning (IRL), which generates cost functions from expert trajectories and runs reinforcement learning using the cost function as rewards [82], GAIL gains optimal policy directly from expert demonstrations. GAIL implements generative adversarial nets (GAN) to generate optimal policy based on expert demonstrations [64].

GAN gains optimal models through adversarial process [83]. GAN selects  $n$  example samples  $x_i = \{x_1, x_2, \dots, x_n\}$  from dataset and generates  $n$  random samples  $z_i = \{z_1, z_2, \dots, z_n\}$  from noise using generator  $G$ . A discriminator  $D$  is applied to distinguish whether the sample comes from data or from the generator  $G$ . The generator  $G$  updates to generate samples more similar to the dataset, and the discriminator updates to distinguish samples. At the end of training, the probability for the discriminator  $D$  to distinguish samples converges to  $1/2$ , which means the discriminator  $D$  recognises the samples randomly. In this case, the generator  $G$  generates samples the same as the dataset samples to complete missions.

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (4.21)$$

$$g_D = E_{\tau_{agent}} \nabla_{\omega} [\log D_{\omega}(s, a)] + E_{\tau_E} \nabla_{\omega} [\log(1 - D_{\omega}(G(s, a)))] \quad (4.22)$$

The value function  $V(D, G)$  presents the updating of the discriminator  $D$  in Equation 4.21. The variable  $\max_D$  maximizes the capability of the discriminator  $D$  to distinguish the data from source data or from the generator  $G$ . The variable  $\min_G$  enables the generator  $G$  to generate data that has minimum difference comparing with source data. After a number of iteration process,  $G$  is able to generate data that are close to the source data. Following the same method of GAN, GAIL uses the generator and discriminator to generate optimal policy as well. The agent during training acts as the generator to take actions. The discriminator is used to distinguish the trajectories from experts and from the agent in Equation 4.22. The variable  $\omega$  stands for the parameters of discriminator  $D$ . Expert trajectories and generated trajectories are represented by  $\tau_E$  and  $\tau_{agent}$  separately. Originally, the authors applied TRPO for generator  $G$  policy updated in GAIL for the publication date of GAIL was earlier than PPO. In this dissertation, I selected PPO for GAIL to restrain large policy updating. When the capability of generator and discriminator largely improve during iterations, the agent acts close to the the expert trajectories. The Figure 4.3 shows the flow chart of GAIL process using PPO.

At the initiating step,  $N_{max}$  is set as the maximum iteration. The GAIL process ends when iterations increase beyond  $N_{max}$ . The records of expert trajectories are imported and the parameters of discriminator and generator for agent are initialized. For each GAIL iteration, the agent as generator  $G$  runs to generate actions. With generated trajectories and expert trajectories as inputs, the discriminator  $D$  updates its parameters according to Equation 4.22. Then the agent as generator  $g$  updates policy by differentiating PPO loss function. Gradually, the policy  $\theta_P$  convergences to optimization after iterations of GAIL process.

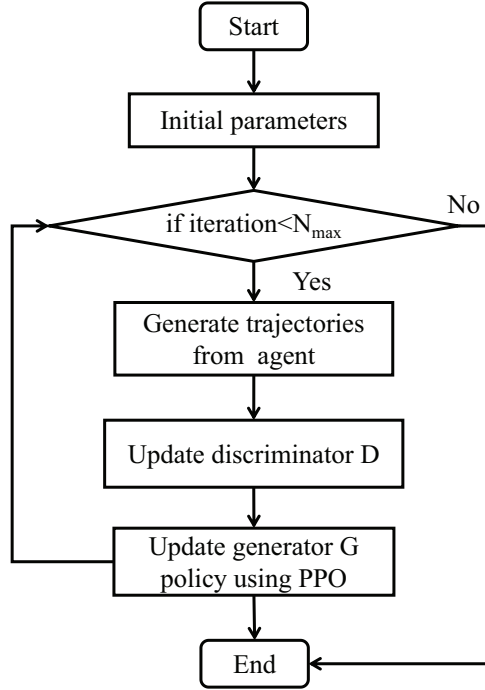


Figure 4.7: The flow chart of GAIL using PPO

#### 4.2.8 Curiosity-Driven Exploration

In order to gain optimal results in complicated environments, rewards for curiosity-driven exploration is implemented to our projects. As explained in [65], reward functions for reinforcement learning training are designed into two parts, namely the extrinsic rewards and intrinsic rewards. The extrinsic rewards are the rewards received from the environment. When the agent accomplishes missions, the agent receives positive rewards. When the agent causes damages, negative rewards are given as punishments to decrease the same motions from agent. However, the extrinsic rewards have difficulty to lead training process to convergence to optimal results in complicated scenarios. In these kinds of cases, intrinsic rewards are introduced to training as curiosity to encourage the agent to explore to gain new

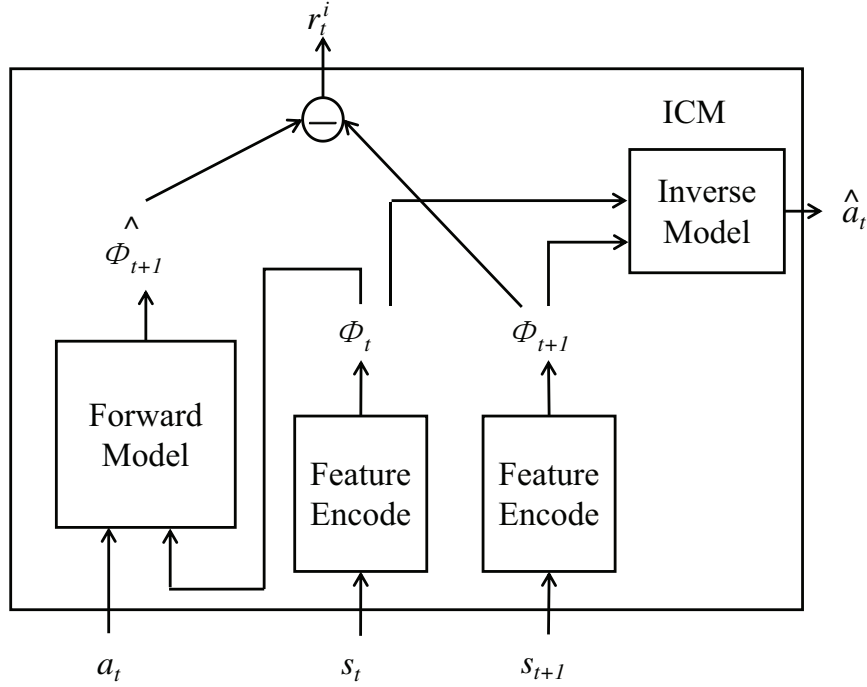


Figure 4.8: The flow chart of ICM

information and learn new skills that can be potentially useful in the future. At time step  $t$ ,  $r_t^e$  represents extrinsic rewards and  $r_t^i$  represents intrinsic rewards.

Intrinsic Curiosity Module (ICM) is designed to compute intrinsic rewards with actions  $a_t, a_{t+1}$ , states  $s_t$ , and  $s_{t+1}$  at time step  $t$  as inputs. ICM contains a forward model and an inverse model separately. The flow chart of ICM is shown in Figure 4.8. The forward model is used to compute environmental impacts which the agent does not conduct but can have an influence to the agent. The inverse model uses environmental data to predict agent actions. Vector  $\phi()$  is used to extract feature from environments. Feature  $\phi(s_t)$  and  $\phi(s_{t+1})$  encode the environmental conditions at state  $s_t$  and  $s_{t+1}$ .

The forward model has function  $f()$  and the network parameters  $\theta_F$ . The function  $f()$  take action  $a_t$  and feature  $\phi(s_t)$  as inputs to perform prediction of  $\phi_{s+1}$  in Equation 4.23, and

$\hat{\phi}_{s+1}$  is the predicting result of function  $f(\cdot)$ . The loss function  $L_F$  is designed as Equation 4.24 to optimize parameter  $\theta_F$ . Then the intrinsic rewards are computed as Equation 4.25 and  $\eta$  is a positive scale factor.

$$\hat{\phi}_{s+1} = f(\phi_s, a_t; \theta_F) \quad (4.23)$$

$$L_f(\phi_s, \hat{\phi}_{s+1}) = \frac{1}{2} \|\hat{\phi}_{s+1} - \phi_{s+1}\|_2^2 \quad (4.24)$$

$$r_t^i = \frac{\eta}{2} \|\hat{\phi}_{s+1} - \phi_{s+1}\|_2^2 \quad (4.25)$$

The inverse model has function  $g(\cdot)$  in Equation 4.26, where  $\theta_I$  is the network parameter. The inverse model are used to estimate action  $a_t$  using feature  $\phi(s_t)$  and  $\phi(s_{t+1})$ , and  $\hat{a}_t$  is the predicting results of function  $g(\cdot)$ . Then the loss function  $L_I$  in Equation 4.27 is used to optimize parameter  $\theta_I$  by minimizing  $L_I$ .

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I) \quad (4.26)$$

$$\min L_I(\hat{a}_t, a_t) \quad (4.27)$$

$$\min_{\theta_P, \theta_I, \theta_F} [-\lambda E_t \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t, a_t; \theta_P \right] + (1 - \beta) L_I + \beta L_F] \quad (4.28)$$



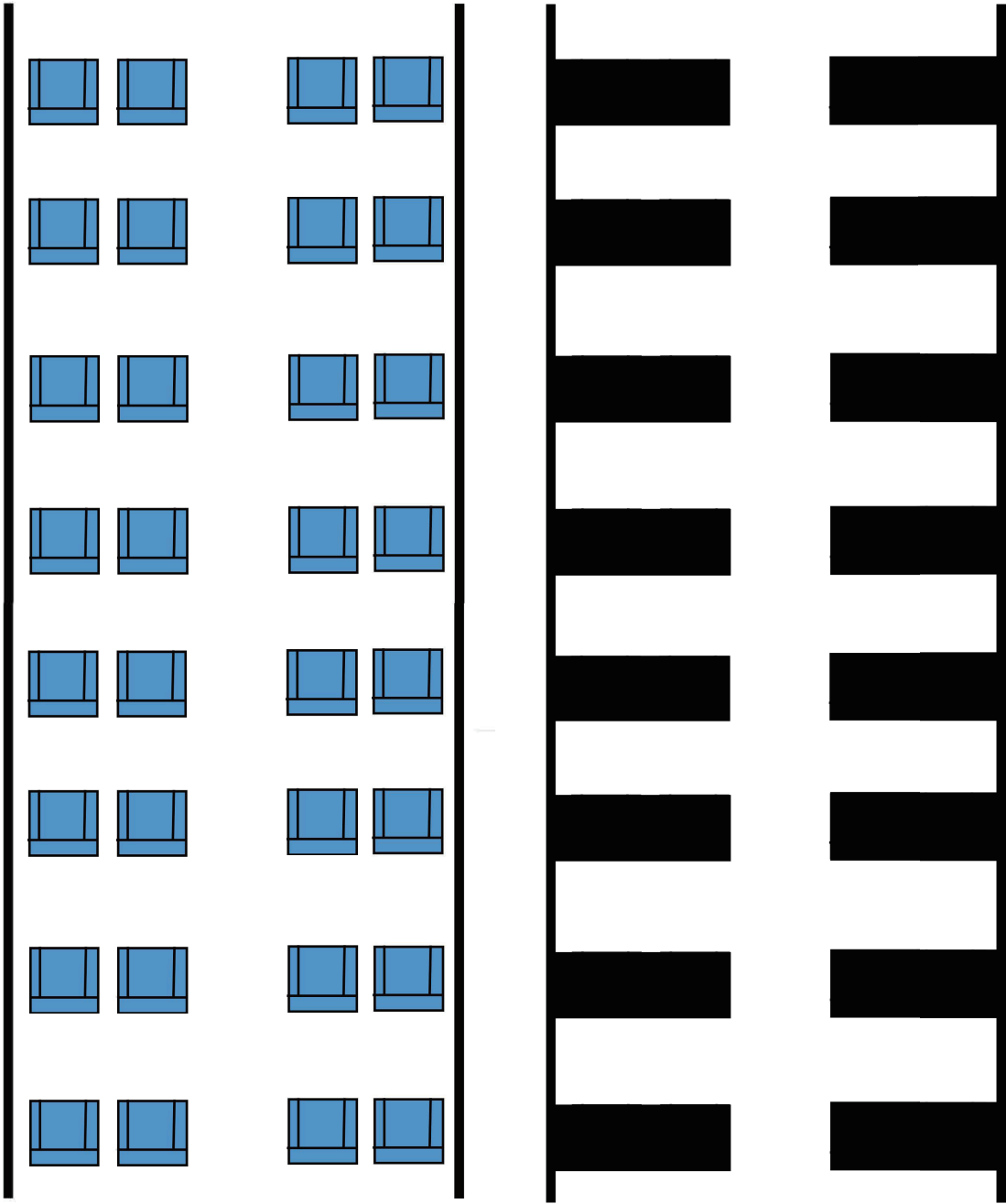
The optimization for the whole network is built using Equation 4.12, Equation 4.24 and Equation 4.27 in and  $a_t \sim (s_t, \theta_P)$ , where  $\theta_P$  will convergence to the optimal policy. The variable  $\beta$  is the scale factor to value the importance of  $L_F$  and  $L_I$ , and  $\beta$  is at a range of  $(0, 1)$ . The variable  $\lambda$  is the scale factor to decide the importance between extrinsic rewards and intrinsic rewards.

### 4.3 Simulation Design

#### 4.3.1 Environmental Setting

For this project, we created a framework that enables a mobile robot to autonomously navigate a room for tasks, such as floor cleaning, using a pseudo-map. Our framework design is based on a deep RL network, and takes the raw input from sensors (such as LIDAR) on the robot to create an optimal strategy that guides the robot while it fully and safely scans a given environment (i.e., factory, warehouse, airplane cabin, etc.).

Figure 4.9 shows both the airline cabin area map and the simplified environmental map. In 4.9(a) a common passenger seat map is used to describe the cabin environment. The two vertical black lines are the bulkheads. Blue passenger seats are set on the map line by line. Between lines of seats, the free spaces are the spaces for passenger legs. In the middle of the passenger seat map, the aisle provides space to connect the whole cabin area. In order to simplify environmental setting up, black blocks in 4.9(b) are used to represent lines of passenger seats. Cabin area is one type of specific environments for flights, the robot is required to avoid large amounts of hard collisions when it conducts CCPP cleaning tasks.



(a) Passenger seat map

(b) Simplified top-view of passenger seat map

Figure 4.9: Cabin area simulation map

During the cleaning process, the entire target area is split into smaller sub-areas. Two layers of training environments are shown in Figure 4.10. In Fig. 4.10(a) each blue node represents one sub-area, and the purple dash lines are the boundaries of each sub-area. The vertical blue line in the middle is the aisle space, and the blue line connects all nodes to construct the entire target space, which in this example is an airline cabin. To complete the cleaning task in each sub-area, the robot is required by our framework to clean the free space. After the area has been cleaned, the robot then moves to the next sub-area through the aisle. Combined with Generative Adversarial Imitation Learning (GAIL), we selected Proximal Policy Optimization (PPO) for simulating the airline cabin cleaning process and to give the robot continuous motion commands [66, 64]. Additionally, we used curiosity functions to generate intrinsic rewards for the agent (robot) during training [65]. The two training environments were a node guidance map and a cabin cell with three aisles of passenger seats and two areas of free space, as shown in Figure 4.10(b). During training, the robot learns the skills to cover the map with a high coverage ratio and to avoid frequent collisions. After the cleaning task is finished within the target cells, the robot moves to the next uncovered cell following the node guidance map. For an area that contains many repeated structures and where the environment is built by multiple cells that are exactly the same, the robot that learned to navigate during training is expected to repeat the behaviors. Once that area is complete, it should move to the next cell until it cleans the entire cabin area. For this project, our testing environments included a whole aircraft cabin space in a simulation setting and two other maps with altered scale and obstacle locations.

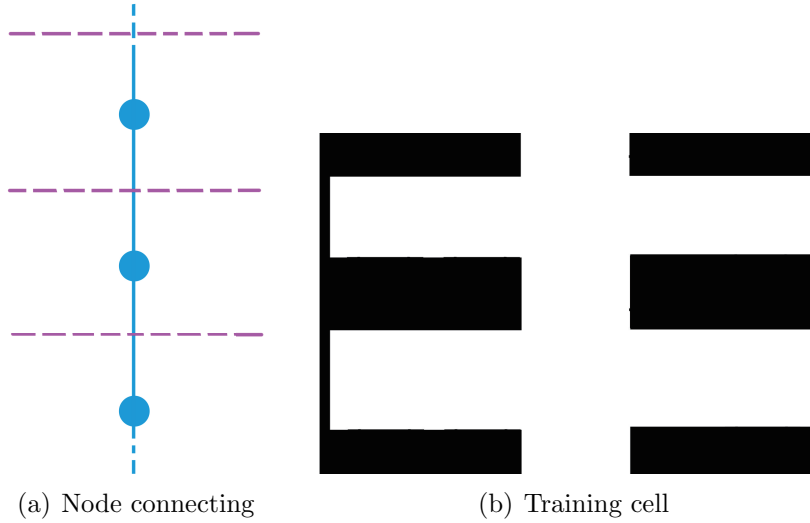
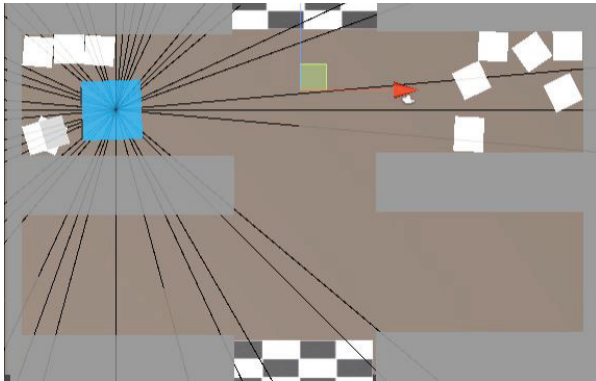


Figure 4.10: Two layers of the simulation environments

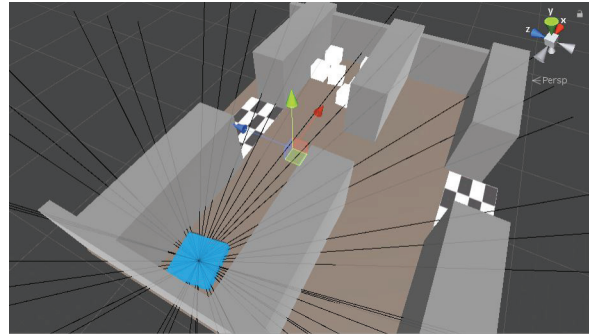
### 4.3.2 Simulation Setting

The training process is divided into two parts, cleaning and tracking. During cleaning, the robot focuses on cleaning every corner of the given area to reach high map coverage. During tracking, when the cleaning mission for an area is complete, the robot moves towards the next uncovered cell and the direction of movement is determined by surrounding landmarks. Both the cleaning and tracking cells were built by Unity3D, as shown in Figure 4.11.

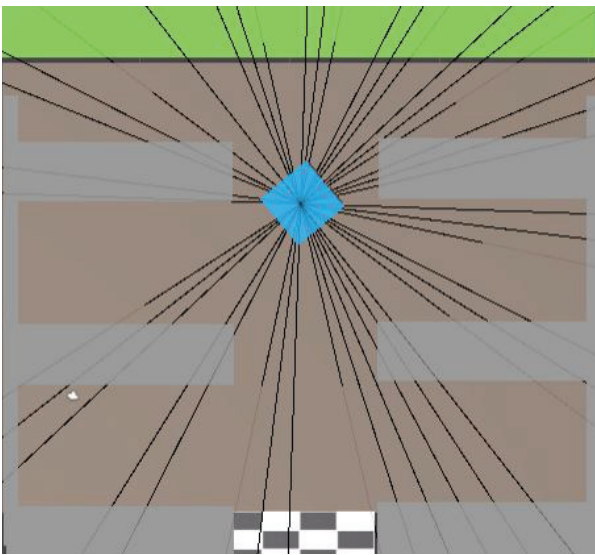
Figure 4.11(b) and Fig. 4.11(a) are screenshots for the cleaning cell, and Fig. 4.11(a) is the top-view. Figure 4.11(c) and Fig. 4.11(d) are screenshots for the tracking cell, and Fig. 4.11(c) is the top-view. In both training cells, the brown area represents the floors where the robot moves freely. Gray blocks represent obstacles such as walls and furniture. The blue box is the robot or agent. The robot receives an action probability distribution that provides linear and angular velocities and enables the robot to rotate 360 degrees with continuous motion. The circle of black lines radiating from the robot simulates the functioning of a LIDAR sensor. LIDAR is transmitted from the center of the robot to its surroundings to



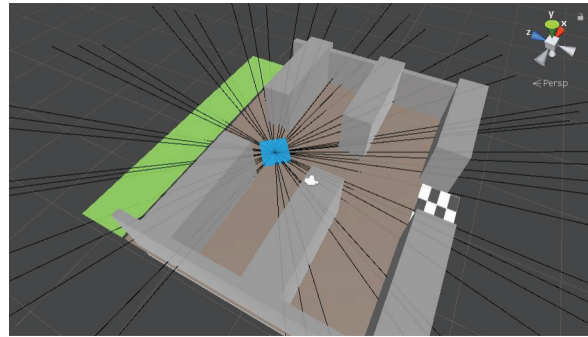
(a) Top-view cleaning cell



(b) Cleaning cell



(c) Top-view tracking cell



(d) Tracking cell

Figure 4.11: Training cells built by Unity3D

detect environmental information. The landmarks with black-and-white patterns represent the bounds of each cell, and the robot is required to move within the given cell without touching the bounds. In Fig. 4.11(a) and Fig. 4.11(b), the white cubes represent locations of uncovered areas and they are set at the corners of the cleaning cells. When the robot reaches a white cube, the white cube is removed from the map. When the robot reaches every white cube on the map, the cleaning mission is regarded as complete. When the training steps of each iteration reach the maximum value, the training cell will be reset even if there are white cubes left on the map. In Fig. 4.11(c) and Fig. 4.11(d), because one bound of the tracking cell remains, it is replaced with a green landmark. The remaining bound has the function of splitting the current cell like the previously visited cell. The green landmark is set in the direction of the uncovered area, and when the robot detects the green landmark with its LIDAR, it moves to the landmark and arrives at the next cleaning cell.

The inputs that were imported into the network during the training and testing process are listed in Table 4.3. The simulated LIDAR function obtains the distance of detected objects, and the LIDAR rays are transmitted with constant angles. Hence, when the rays detect objects, the heading angles of objects are obtained by the robot. During cleaning, the rays can sense objects with tags that include walls, goals, exits, and bounds. The tag "wall" and "bound" represent both walls and obstacles that sit in the middle of the cell. The tag "goal" is for white cubes that denote uncovered areas. The tag "exit" represents the green landmark that guides the robot to leave its current cell and enter a new cell. The velocity of the robot is sent to the network, and 3D vectors  $(x, 0, z)$  are used to store this information. On the Unity3D platform, the  $y$  axis is the vertical axis, and a robot moving on the floor has velocity of  $y = 0$ .

However, the RL training process meets challenges when trying to utilize global coordinate values of grid or cell maps as inputs, while traditional path planning approaches generate routes for robot navigation with global coordinates provided by grid maps. Grid maps are widely used to represent free space and obstacles of environments. Local coordinates describe the distance and directions of obstacles to the robot. Global coordinates record the exact location values of free space and obstacles on maps.

One challenge is that when RL uses global coordinates of maps as inputs for training, the testing performs well if the training and testing environments are the same. When the testing input values are out of the ranges of training, testing fails. Another is that the coordinates of grids or cells are imported as a set of vectors. The number of vectors is unchangeable during training process. If the input coordinates in testing cases are fewer than training, the remained input space can be set as constant values (e.g.  $(x, y) = (999, 999)$ ). However, if the number of coordinate vectors in testing are larger than training environment, the RL network fails to work.

The global coordinate inputs in the training process are in the boundaries of the simulation plane. The difference of coordinate value range is shown in Figure 4.12. Figure 4.12(a) is an example for the training plane and Fig. 4.12(b) is the corresponding testing plane. The global coordinates on the training plane in Fig. 4.12(a) are within the range of  $(-5, 5)$ , which means the inputs values in the training process have the same bounds  $(-5, 5)$ . In Fig. 4.12(b) the testing environment shifts from the training environment and keeps the same shape, so the global coordinates in the test environment will be out of the range  $(-5, 5)$ .

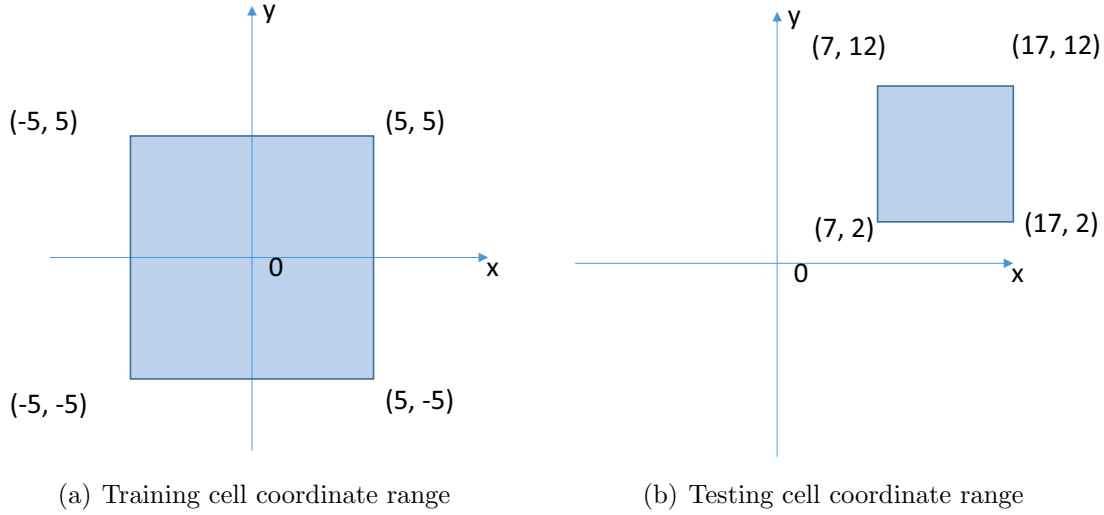


Figure 4.12: Difference of training and testing plane coordinate range

Then the trained RL networks will receive input values that are out of its training experience. In this case, the testing fails. One solution is to normalize global coordinates with map boundary length, then the coordinate value will not be out of range.

As mentioned above, Figure 4.13 and Figure 4.14 shows the second challenge. The training environment is divided into grids or cells. In Figure 4.13, the size of the training environment is  $6 \times 6$  and there are 36 cells on map. The structure of the RL network is decided at the beginning of training, so the number of global coordinate inputs stays the same at 36. When the robot performs map coverage in the environment with a smaller size in 4.14(a), there are only 16 cells for RL inputs. Then the cells beyond those 16 will be filled as constant values (e.g. (999; 999)) to keep the RL network working properly. However, when the robot is required to perform map coverage in 4.14(b), there are more cells on the simulation plane. The plane size in 4.14(b) is  $8 \times 8$  and there are 64 cells on the map. The extended numbers of inputs will cause errors to conduct testing.



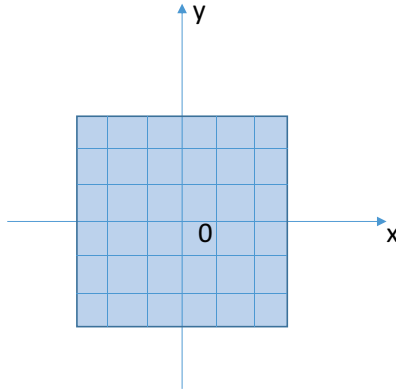
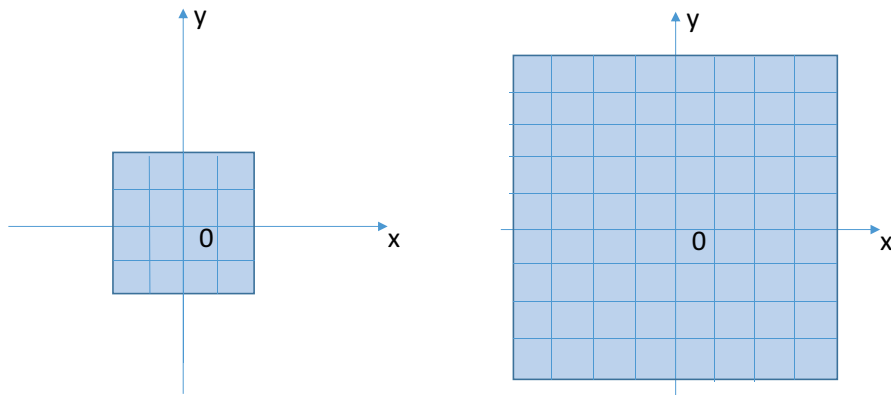


Figure 4.13: Training plane with medium coordinate range



(a) Narrow testing coordinate range      (b) Expanded testing coordinate range

Figure 4.14: Narrow and Expanded testing plane coordinate range

Hence, in order to enable the robot to learn the behaviors in cleaning and tracking process rather than limit the robot in one exact environment, the position  $(x, y, z)$  of robot and the locations of white cubes are not used. In our simulation, only local surrounding information observed by rays and the robot velocity are considered.

After training is finished, both the cleaning process and tracking process alternate when the robot performs map coverage for the given area. The whole map coverage process is shown in Figure 4.15. When the process initiates, the robot performs cleaning process at the

Input	Parameter	Description
Ray observations (cleaning process)	distance angle "wall", "goal", "bound"	The circle of rays detect surroundings and receive information
Ray observations (tracking process)	distance angle "wall", "bound", "exit"	The circle of rays detect surroundings and receive information
robot velocity	Vector3 ( $x, 0, z$ )	The 2D plane in Unity3D is x0z and the y axis is the vertical axis

Table 4.3: Training inputs

current cell by reach all white cubes that set on map. After the robot completes the cleaning task of the current cell, the black-and-white bound that segregates the next uncovered cell will be removed, and the green landmark for guidance will show up. Then the robot behavior shifts to tracking process. The robot senses the location of the green guidance using the set of rays and moves towards to it. After the robot arrives at the green guidance, the guidance is removed and the bound behind the robot shows up to segregate the new arrived cell to previous cells. Then the robot repeats perform the cleaning process and tracking process in this new uncovered cell. When all the uncovered cells have been visited, the whole map coverage process ends.

### 4.3.3 Rewards Design

To complete the full coverage task of this project, we set reward functions during the training process. Reward functions are designed to train cleaning and tracking tasks separately. To train the cleaning process, there are two basic requirements the robot must meet. First, the robot needs to avoid frequently colliding with obstacles. Second, the cleaning cell must be highly covered by the robot's cleaning trajectory. Therefore, we set the

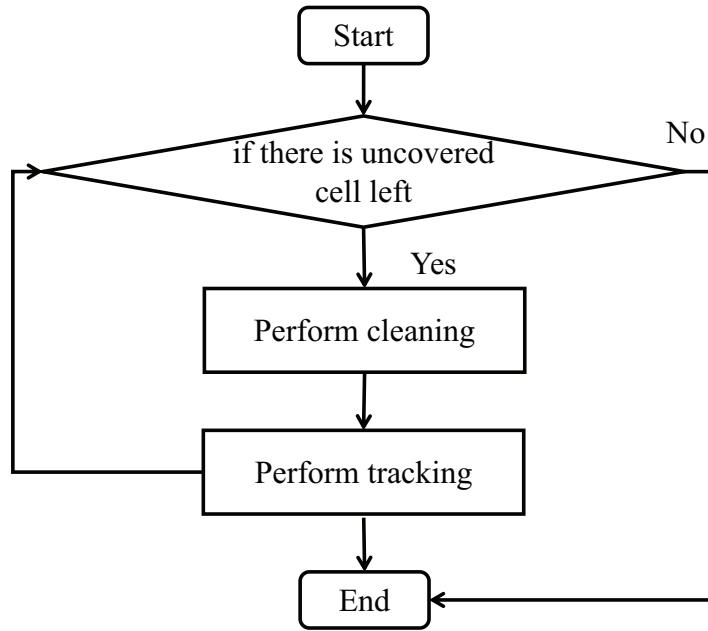


Figure 4.15: The flow chart of map coverage process

Action	Reward	Description
Cover new area	0.5	The robot reaches one white cube
Hit obstacles	-0.3	The robot hits gray blocks for once
Reach bounds	-10	The robot moves cross the black-and-white patterns for once
Step cost	-0.01	The cost accumulates when the process is running
Clear	set 3	The robot covers the cleaning cell

Table 4.4: Rewards for cleaning training

<b>Action</b>	<b>Reward</b>	<b>Description</b>
Hit obstacles	-0.3	The robot hits gray blocks for once
Reach bounds	-10 Current iteration ends	The robot moves cross the black-and-white patterns for once
Step cost	-0.01	The cost accumulates when the process is running
Reach guidance	2	The robot reaches the green landmark and arrives the next uncovered cell

Table 4.5: Rewards for tracking training

reward functions for cleaning training as listed in Table 4.4. Uncovered areas use white cubes as landmarks for training and testing. When the robot arrives at the uncovered area, the corresponding cube is removed, the uncovered area updates to a covered area, and a positive reward of 0.5 is added to the training network. The algorithm records the total number of existing and removed landmarks and computes the coverage ratio. If the robot hits an obstacle, a negative reward of  $-0.3$  is added to the training network, and the robot’s corresponding behaviors will be diminished in future training scenarios. In this way, the robot learns to prevent possible damage caused by collisions during testing. When the robot reaches a boundary, a negative reward of  $-10$  is added to the training network. Additionally, when the current iteration ends, the environment is reset to make the robot move within a cell’s area. A negative reward of  $-0.01$  is continuously added to the network while the training process is running, as this step-cost punishment encourages the robot to complete cleaning using minimal steps. When all uncovered areas are cleared, the cleaning mission is complete, and the total reward for this iteration episode is set to a positive value of 3.


As in the cleaning training process, to train the tracking process, the robot must also avoid frequent collisions. Additionally, the robot must arrive at the green guidance landmark to exit the recently completed cell and to enter the next uncovered cell. We set the reward

functions for tracking training as listed in Table 4.5. The negative rewards for a robot to avoid collisions and to keep moving efficiently within a given cell have the same value as those in Table 4.4; the negative reward for collisions is  $-0.3$  and for crossing bounds is  $-10$ . When the robot reaches a bound, the current iteration ends. The step-cost is  $-0.01$ , and it accumulates while the training process is running. When the robot reaches the green landmark, a positive reward of 2 is added to the training network, the tracking missions are considered complete for this iteration, and the environment is reset.

#### 4.3.4 Imitation Demo Recording

Unity3D provides a recorder script that records episodes of demos (demonstrations) as expert trajectories for imitation learning. Demos are the expert trajectories of imitation learning functions in Unity3D, and the environments for the demos are the same as the environments used for training. Keyboards or control pads are used by developers to control the agents during recording. Demos record episodes, total steps, mean rewards, and the value of all inputs and outputs for every step.

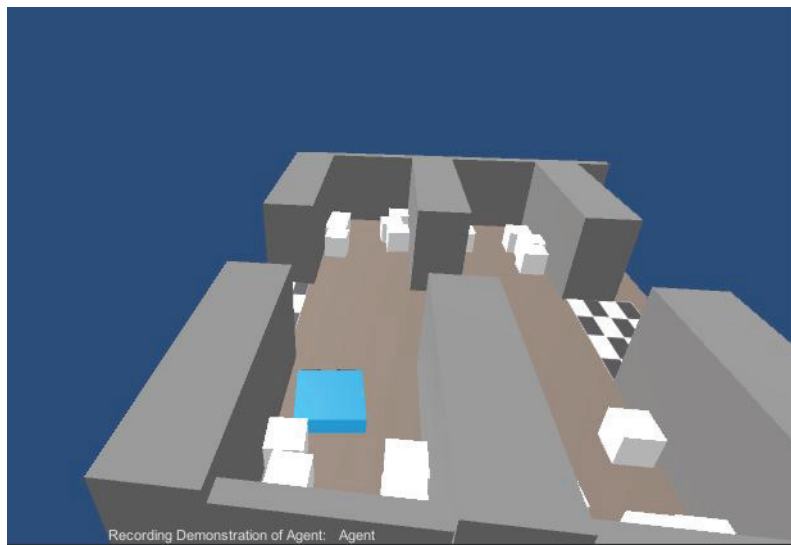
Figure 4.16 shows parameters and a screenshot of demo records for the cleaning cell training. In Fig. 4.16(b), the agent was controlled manually to move to each corner of the cleaning cell and reach all the white cubes that are landmarks for uncovered area. In Fig. 4.16(a), 34 episodes are recorded as demo and 4827 total steps are recorded as total experiences. The mean reward of all episodes of the demo is  $-0.1211876$ . The agent has 244 vectors as inputs, including black rays for environmental observations and its own velocities. The agent acts as a ground robot, moves on 2D plane, has a vector action size of 2, and has a

 CleanCellDemo

**Meta Data**  
Demonstration Name: CleanCellDemo  
Number Experiences: 4827  
Number Episodes: 34  
Mean Reward: -0.1211876

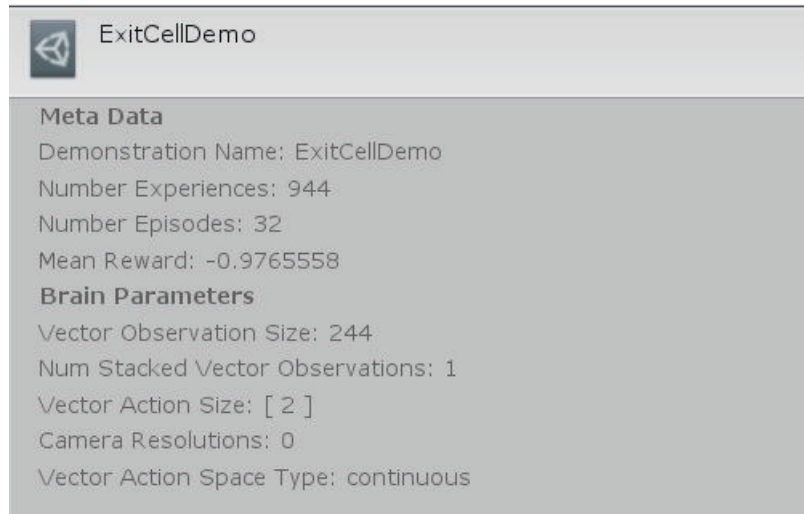
**Brain Parameters**  
Vector Observation Size: 244  
Num Stacked Vector Observations: 1  
Vector Action Size: [ 2 ]  
Camera Resolutions: 0  
Vector Action Space Type: continuous

(a) Parameters for cleaning cell demo

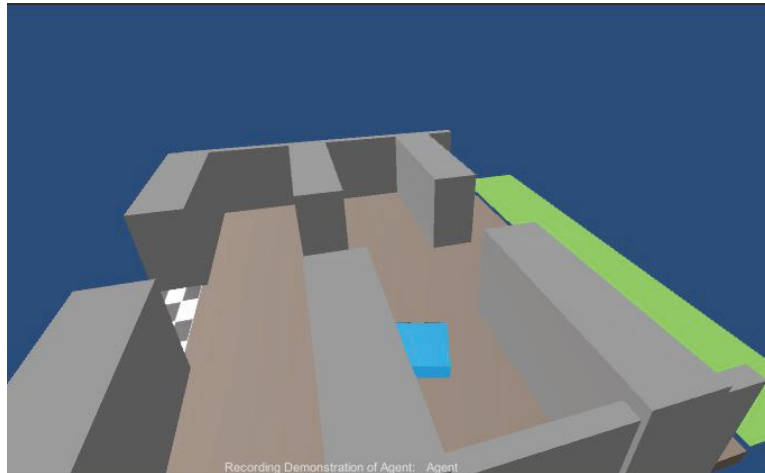


(b) A screen shot for cleaning cell demo

Figure 4.16: Cleaning cell demo



(a) Parameters for tracking cell demo



(b) A screen shot for tracking cell demo

Figure 4.17: Cleaning cell demo

continuous motion type. We used no image inputs during this project, so camera resolution is 0.

Figure 4.17 shows parameters and a screenshot of demo records for the tracking cell training. The agent was controlled manually to the green landmark that are the exit for current cell in screenshot Fig. 4.17(b). In Fig. 4.17(a), 32 episodes are recorded as demo and 944 total steps are recorded as total experiences. The mean reward of all episodes of the demo is  $-0.9765558$ . The agent has the same number of observations and acts in the same continuous motion. Thus, the inputs and outputs for the tracking process training are the same as those used in the cleaning process.



## Chapter 5

### Experimental Results

#### 5.1 Environmental Complexity Measurement

*Robotic platform:* The iRobot Create® 2 was selected as the hardware platform for our experiment. Two driver motors for the left and right wheels enabled the robot to move and rotate for 360-degrees. Two ABS plastic boards were set on the top of the robot to form a chassis. The on-board controller, battery and a LIDAR sensor were set on the chassis. The on-board controller was connected with the robot to send velocity commands to the left and right motors and receive odometry data to locate itself. The on-board controller was also connected with LIDAR. For the sensor of our robot, we chose a RPLIDAR A1 Low Cost 360-Degree laser range scanner to observe surroundings and build maps through Robot Operating System (ROS). The photo of our robot platform is provided in Figure 5.1.

*Environmental setting:* To present the results of our proposed approach, we selected three regions to conduct real-world 2D map building at our lab, and the three regions are the mock mall, meeting area and RFID office. As shown in Figure 5.2 furniture and clothes shelves were set in mock mall area to simulate a mall environment. In the meeting area shown in Figure 5.3, the office table, and chairs located at the center of the room. Shelves and tools were placed against the wall. The main region of RFID office was empty as shown in Figure 5.4. Office tables and chairs were set to one side wall.



Figure 5.1: Robot Platform



Figure 5.2: Mock Mall



Figure 5.3: Meeting Area



Figure 5.4: RFID Office

Experiments were conducted to generate 2D maps to measure environmental complexity. Maps of given regions in RFID lab were built up. We sent commands to the robot through keyboard during the mapping process. The map was established manually, and we controlled the robot to navigate the whole area.

### 5.1.1 Experimental results

The mapping process was conducted in the three areas as described above. The Figure 5.5, Figure 5.6 and Figure 5.7 contain the original 2D maps, the corresponding normalized maps, and complexity maps. Part (a) of all the three figures are the original maps constructed by on-board controller and RPLIDAR (Fig. 5.5(a), Fig. 5.6(a) and Fig. 5.7(a)). The white area is free space and is safe for a robot to move around. The black area is the location of obstacles and walls. The gray area represents unexplored areas, where the obstacle distribution is unknown. Part (b) of all the three figures are the normalized maps (Fig. 5.5(b), Fig. 5.6(b) and Fig. 5.7(b)). Before normalizing the original maps, we replaced the value of the gray area to white. The diameter of the robot is 0.35m. According to the map resolution in (5.1), the robot diameter on maps is 7 pixels, which means the normalization parameter of the original maps is 7. The original maps were divided into  $7 \times 7$  size cells. If there are obstacles within the  $7 \times 7$  cell, the cell space occupied and the pixel is set to 1. Otherwise, the cell is empty and the pixel is set to 0. For each empty pixel, its complexity value was computed using (4.4) with adjacent pixels as inputs. The complexity maps are shown in (c) of all the three figures (Fig. 5.5(c), Fig. 5.6(c) and Fig. 5.7(c)). Different grayscale values

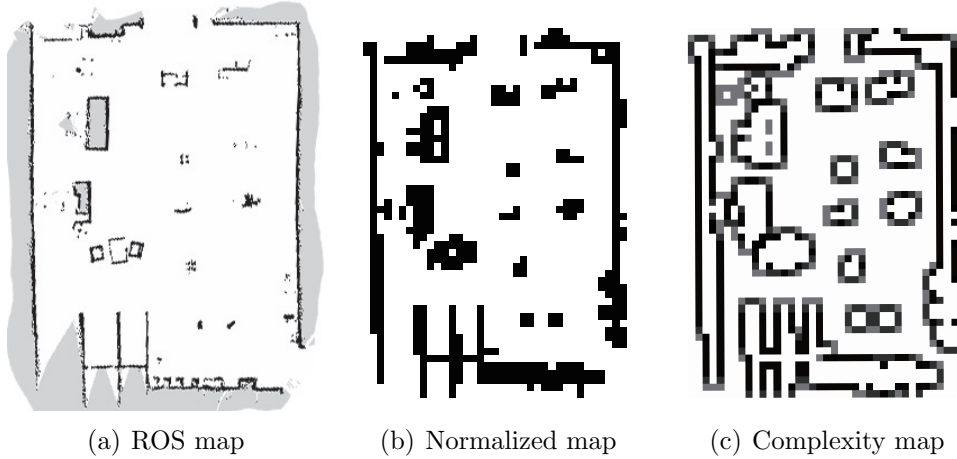


Figure 5.5: Mock mall maps

Table 5.1: Environmental Complexity for Real-World Maps

	Mock Mall	RFID Office	Meeting Area
<b>Original Size</b>	364x280	392x175	252x413
<b>Normalized Size</b>	52x40	56x25	36x59
<b>Total sum</b>	554.5350	607.1892	338.7164
<b>Total sum/ Normalized Size</b>	0.2666	0.2419	0.2859

of each pixel represent the complexity of its surroundings. The sum of complexity values for all pixels was computed and shown in Table 5.1.

$$\frac{\text{meter}}{\text{pixel}} = 0.05 \quad (5.1)$$

According to Table 5.1, the three given regions have different sizes. Finally, the complexity value is defined as the sum divided by the normalized map size. Hence, the complexity value enables us to measure the whole condition of maps. The meeting area has the highest complexity value, while the RFID lab area has the lowest complexity value. The complexity value of the corresponding maps reflects the true complexity of the environments.

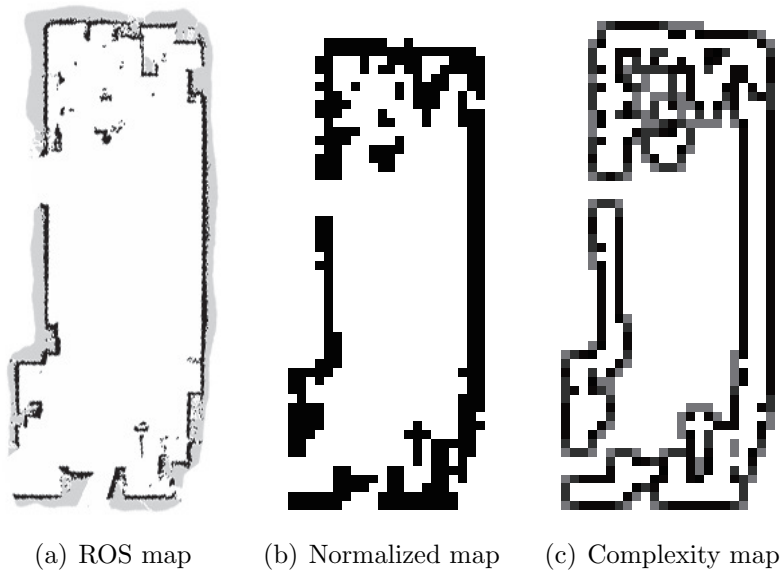


Figure 5.6: RFID lab maps

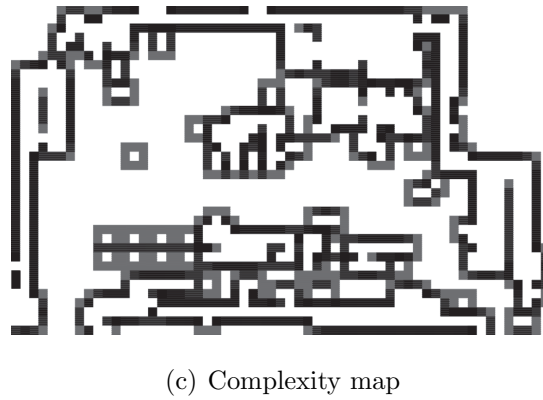
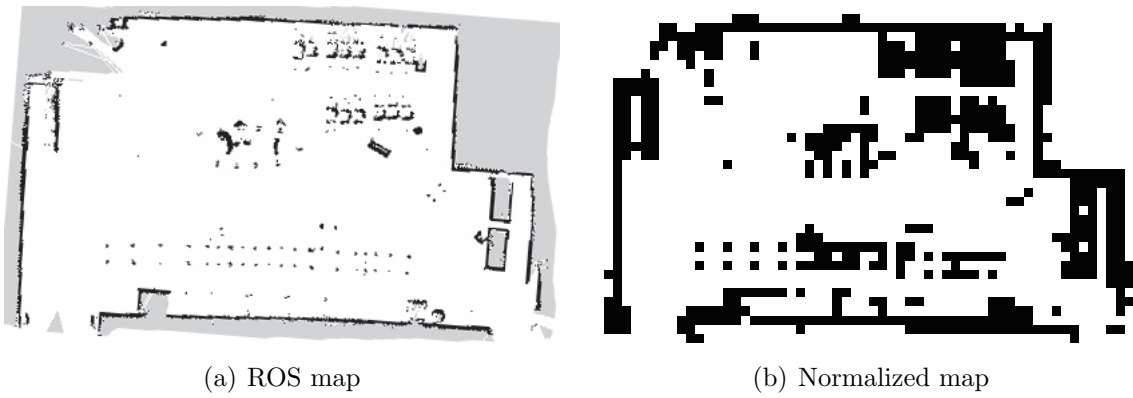


Figure 5.7: Meeting area maps

## 5.2 CCPP for Cabin Area using Reinforcement Learning

### 5.2.1 Training Parameter Setting

The PPO algorithm was used for cleaning cell training and tracking cell training. Before training begins, hyperparameters of PPO are as shown in Table 5.2. The `batch_size` is `minbatch` that is the number of samples for each policy descent and was set to 128. The parameter  $\beta$  in Equation 4.20 as coefficient of entropy was set to 0.01. Before policy updating starts, the number of collected training samples are stored to 2048 as `buffer_size`. The parameter  $\epsilon$  in Equation 4.19 is set to 0.2 as the scale for policy updating. The parameter `hidden_units` controls the number of neuron units in the hidden layer and was set to 512. The parameter  $\lambda$  in Equation 4.13 to balance the bias and variance was set to 0.95. The initial learning rate of PPO as `learning_rate` was set to  $3 \times 10^{-4}$ . The variable `max_steps` represents the maximum steps for all iterations of training. The optimal policy of cleaning cell training requires more steps than tracking cell training. Hence, `max_steps` in cleaning cell training was set to  $6 \times 10^6$  and `max_steps` in tracking cell training was set to  $2 \times 10^6$ . The `normalize` function is used to normalize inputs for training. In cleaning cell training and tracking cell training, the `normalize` function was turned off. An epoch represented using all training samples for policy optimization for one time. The times of epoch as `num_epoch` was set to 3. The `num_layers` was set to 2 and the number of layers in the neuron network was 2. The parameter `time_horizon` is the number of steps as a frequency to collect samples to sample buffer for each agent. The `time_horizon` was set to 128. The `summary_freq` was the frequency to record statistics data. Tensorflow provides TensorBoard to generate scalar plots. In cleaning cell training, `summary_freq` was set to 20000. In tracking cell training,

summary\_freq was set to 10000. The use\_recurrent means to implement recurrent neural networks for training. In both cleaning cell training and tracking cell training, use\_recurrent was turned off.

Curiosity-Driven Exploration and GAIL were applied for both cleaning cell training and tracking cell training. The cleaning cell training and tracking cell training shared the same parameters as listed in Table 5.3. Extrinsic represents environmental rewards signals and curiosity represent intrinsic reward signals as mentioned in [65]. The strength of extrinsic is the parameter that multiplies the raw environmental rewards. This was set to 1.0, which means environmental reward had a large influence for training. The parameter  $\gamma$  of the extrinsic signal is the discount factor to compute the sum of future rewards in Equation 4.11 and Equation 4.12. The extrinsic  $\gamma$  was set to 0.99. The strength of curiosity is the parameter that multiplies the raw intrinsic rewards. The curiosity strength was set to 0.02 to make sure that the intrinsic rewards would not overwhelm the environmental rewards. The curiosity  $\gamma$  is the discount factor to compute the sum of future intrinsic rewards and was set to 0.99. Encoders from ICM framework in Figure 4.8 generate vectors from observations. The size of encoders as encoding\_size was set as 256. The GAIL rewards can be combined with environmental rewards for training. The strength of GAIL is the parameter that times with raw GAIL rewards and the GAIL strength was set to 0.3. The  $\gamma$  of GAIL is the discount factor to compute the sum of future GAIL rewards and was set to 0.99. The GAIL encoding\_size represents the unit size of discriminator in hidden layer and encoding\_size was set to 128.



Table 5.2: Training hyperparameters using PPO

Setting	Cleaning Cell Training	Tracking Cell Training
batch_size	128	128
$\beta$	0.01	0.01
buffer_size	2048	2048
$\epsilon$	0.2	0.2
hidden_units	512	512
$\lambda$	0.95	0.95
learning_rate	$3 \times 10^{-4}$	$3 \times 10^{-4}$
max_steps	$6 \times 10^6$	$2 \times 10^6$
normalize	FALSE	FALSE
num_epoch	3	3
num_layers	2	2
time_horizon	128	128
summary_freq	20000	10000
use_recurrent	FALSE	FALSE

Table 5.3: Training hyperparameters using curiosity and GAIL

Setting	Parameter	Value
Extrinsic	strength	1.0
	$\gamma$	0.99
Curiosity	strength	0.02
	$\gamma$	0.99
	encoding_size	256
GAIL	strength	0.3
	$\gamma$	0.99
	encoding_size	128

### 5.2.2 Scalar Analysis using TensorBoard

Tensorflow provides the TensorBoard function that stores statistical data and plots scalar figures for the training process. TensorBoard generated 6 separate statistical plots for the cleaning cell and tracking cell trainings in Figure 5.8 and Figure 5.9, respectively. In each statistical plot, the  $x$  axis represents the number of training iterations and the  $y$  axis has various meanings. The Cumulative Reward figure shows the cumulative environmental rewards from every iteration in the training process (5.8(a)). Gradually, cumulative rewards grow as the training process is conducted successfully. The GAIL Loss figure (5.8(b)) represents the level at which the learning model imitates the expert trajectories. GAIL Loss decreases when a successful training continues. The Curiosity Forward Loss (5.8(c)) and Curiosity Inverse Loss (5.8(d)) figures represent the value of the loss function in the forward and inverse models, respectively. The loss function of the forward model measures the accuracy of estimating the feature that represents next step's state. The loss function of the inverse model measures the accuracy of estimating the action between two states. The Policy Loss figure represents the average magnitude of each time policy update in PPO (5.8(e)). The Value Loss figure measures the difference between the exact and predicted state values (The value loss shown in Fig).

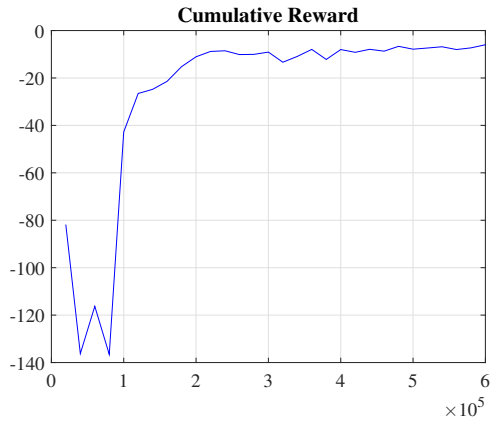
In cleaning cell training Figure 5.8(a), the cumulative reward value increases rapidly from 0k to 100.00k iterations and then slightly from 100.00k to 500.00k iterations. The training process convergence becomes stable after 500.00k iterations. During training, GAIL Loss continued to decrease in Figure 5.8(b), meaning that training networks followed the expert trajectories more closely over time. In both Figure 5.8(c) and Figure 5.8(d), the

curiosity forward loss and curiosity inverse loss decreased when the number of iterations increased. The reduction in curiosity loss shows that the prediction accuracy of state vectors and actions improved throughout the training process. Figure 5.8(e) plots the magnitude at every time the policy updated during the training process. Magnitude values fluctuated throughout the training but decreased over time. The value loss shown in Figure 5.8(f) was high when the training process was learning using reward feedback; however, value loss fluctuated throughout the training but converged to stable towards the end of the process.

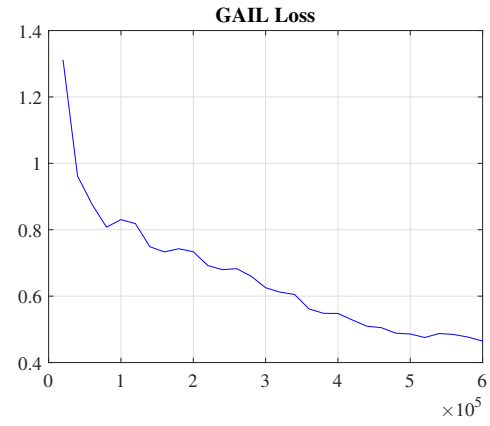
In tracking cell training Figure 5.9(a), the cumulative reward value grew dramatically between 0k to 60.00k iterations, then increased slightly and converged to stable towards the end of the training. During training, GAIL Loss in Figure 5.9(b) decreased quickly between 0k to 40.00k iterations and then fluctuated slightly to imitate the expert trajectories. Curiosity forward loss in Figure 5.9(c) and curiosity inverse loss in Figure 5.9(d) both decreased, but prediction accuracy increased throughout the training process. Figure 5.9(e) shows the magnitude of policy updates during every iteration. Policy loss fluctuated throughout the training process but decreased as compared to the loss value in earlier iterations. In Figure 5.9(f), the value loss was high at the beginning of the training process, indicating changes to the learning policy. Over time, the training somewhat converged to stable but the value loss fluctuated.

### **5.2.3 Comparison between Reinforcement Learning and Random Motion Approaches**

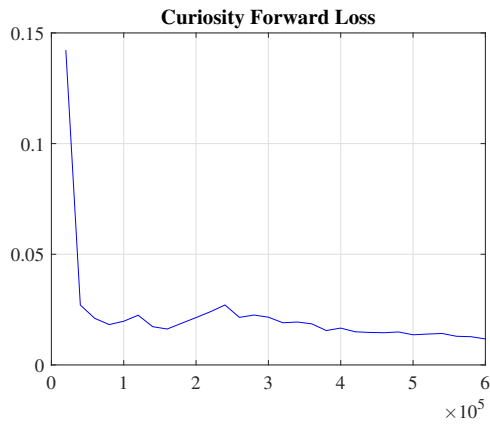
Figure 5.10 are screenshots to record the trajectories for a simulated cleaning robot. In 5.10(a) the robot was implemented random motion algorithms. In 5.10(b) the robot



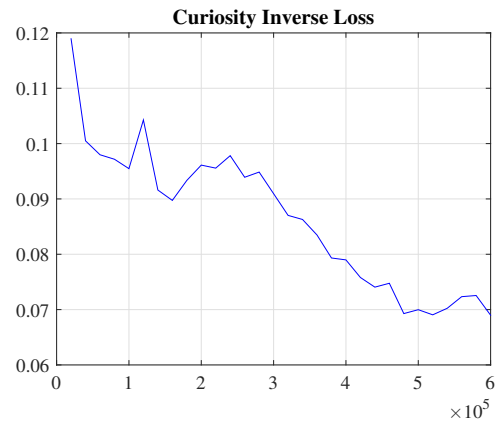
(a)



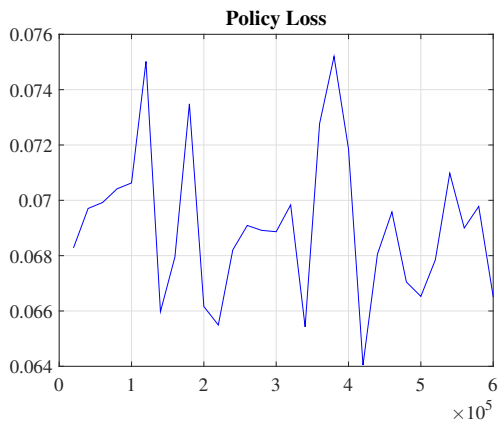
(b)



(c)



(d)

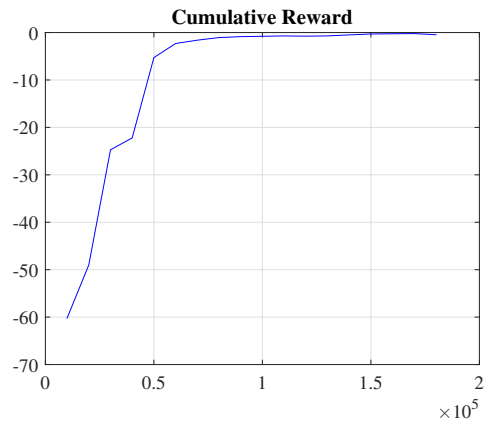


(e)

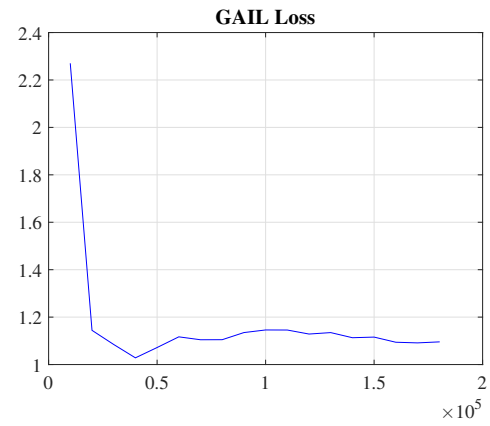


(f)

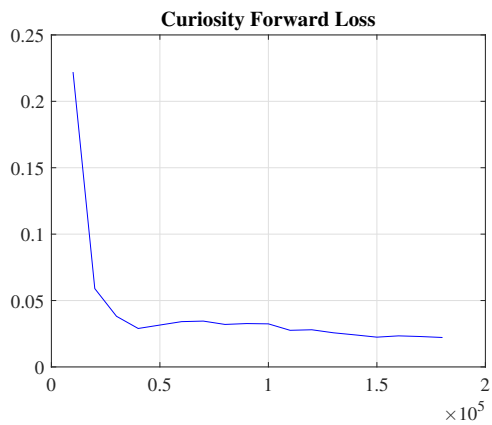
Figure 5.8: Training statistics for clean cell using Tensorflow



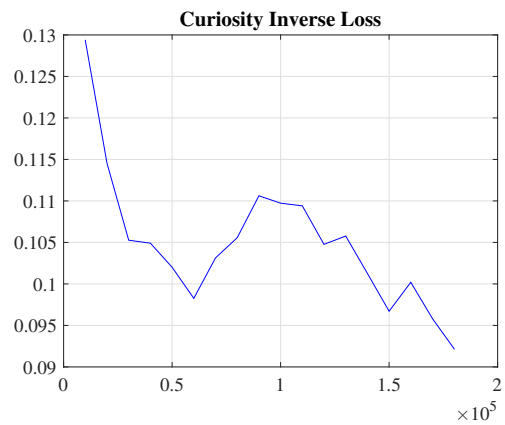
(a)



(b)



(c)



(d)



(e)



(f)

Figure 5.9: Training statistics for track Cell cell using Tensorflow

was implemented with optimal policy that was generated from reinforcement learning. As mentioned before, brown floor represent free space and gray blocks represent obstacles. The blue box is the cleaning robot. Both robots in 5.10(a) and 5.10(b) were required to perform the same CCPP missions to gain high coverage ratio of the map with less time cost and trajectory length. Hence, when the robot reaches the location of all the white cubes that are set at the corners, the CCPP missions for one episode are regarded as completed. The purple lines on Figure 5.10 are the trajectories left by the robot cleaner. In 5.10(a) the trajectories are denser at the corners of the map. The robot moved back and forth repeatedly and kept hitting on walls. Without simulated external sensors provided, the robot was unable to gain observations from surrounding area. In other words, the robot was trapped at corners and had difficulties to move out. Even if the robot moved around at the corner area with repeated trajectories, the robot ignored two corners on the map. After the maximum steps (3000) to end the current episode, the corners remained unvisited. In other words, the CCPP mission was not completed successfully using random motion algorithms. In 5.10(b), the robot used the learned policy from reinforcement learning to cover corners and moved around. The robot reached all the white cubes to complete cover the map area. Compared with 5.10(a), the trajectories in 5.10(b) had fewer repeated lines. The maximum speed for both robots were set to be the same. Hence, reinforcement learning enabled the robot to complete map coverage with in less time.

The robot's position, rotation, and timestamps were recorded while cleaning missions were performed. In Figure 5.11, the  $x$  and  $y$  axis represents position coordinates and the  $z$  axis represents the corresponding timestamp. The testing sessions using RL were recorded for 300 episodes, and the testing sessions using random motion were recorded for 50 episodes.

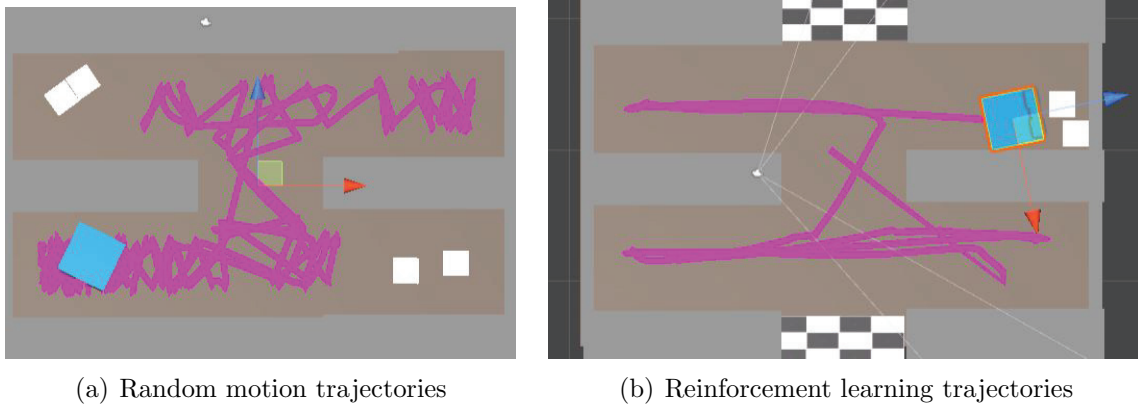


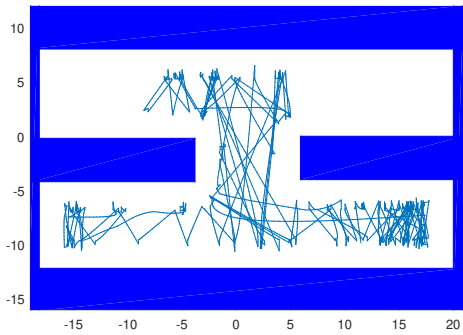
Figure 5.10: Cleaning robot trajectories in simulation

During the simulation, testing sessions using random motion algorithms took longer to collect trial data, hence, only 50 episodes of random motion testing were recorded for comparison and analysis. Figure 5.11 shows the cleaning map coverage area, robot trajectories, and time cost of random motion algorithms as compared to RL methods for one episode of testing. This episode was selected randomly from the testing records. Both methods enabled the robot's movement within the space to cover all map areas, but the differences in trajectory length and time-costs between the methods were significant. The trajectories for one episode are plotted in 5.11(a) and 5.11(b). The center of the robot was recorded and is indicated by blue lines. In Figures 5.11(c) and 5.11(d), the robot's trajectories are plotted with corresponding timestamps on the  $z$  axis. The cleaning map coverage area is plotted in 5.11(e) and 5.11(f). Figure 5.11(e) shows the coverage area using random motion algorithms, and Fig. 5.11(f) shows the coverage area using RL networks. The coordinates of the vertices of gray walls and blocks were recorded from simulation environments in Unity3D, and we used Matlab to plot blue polygons to represent the location of obstacles based on trails. The red area indicates previously covered areas. The center positions of the square-shaped robot

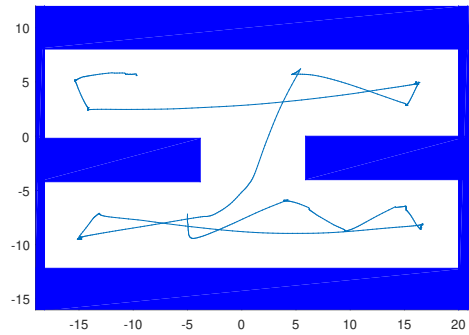
and heading angles were used to plot its location and rotation. The white area indicates uncovered areas. The coverage ratio was computed by comparing pixels of the red area to pixels of the free space. According to 5.11(e), the robot covered the bottom aisle area fully but with many repeated trails. However, the robot left two corner areas in the top aisle uncleaned before it reached its step limitation. Comparatively, the robot in 5.11(b) covered every corner area of both aisles. Note that the area around obstacle edges were left uncovered, as the robot learned to avoid frequent collisions during the training process.

To compare the performance of random motion testing against RL testing, we considered three aspects: coverage ratio, trajectory length, and time-cost. Histograms of these aspects were plotted in Figure 5.12. Figure 5.12(a) and 5.12(b) show the coverage ratio obtained using random motion and RL, respectively. The coverage ratio using random motion was mainly in the range of 55% - 65% as shown in Figure 5.12(a), and the coverage ratio using RL was primarily in the range of 65% - 75%, as shown in 5.12(b). The coverage ratio range of random motion was 10% lower than that of RL. The trajectory length using random motion was mainly in the range of 650 - 800, as shown in 5.12(c), while the trajectory length using RL was mostly in the range of 120 - 145, as shown in Figure 5.12(d). The trajectory length using random motion was more than 5 times longer than that of RL. Figures 5.12(e) and 5.12(f) show the distributions of time-cost for random motion and RL methods. If the robot reached all of the targets within 3000 steps (the set maximum number of steps), the current episode of testing ended. If the robot failed to reach all of the targets within 3000 steps, the current episode also ended. For random motion, each of the 50 episodes reached the maximum number of steps and failed to complete the cleaning missions. Therefore, the total time-cost of the random motion method equalled the time it took for the simulation to run

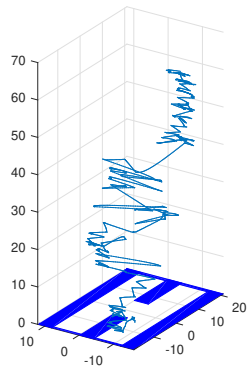




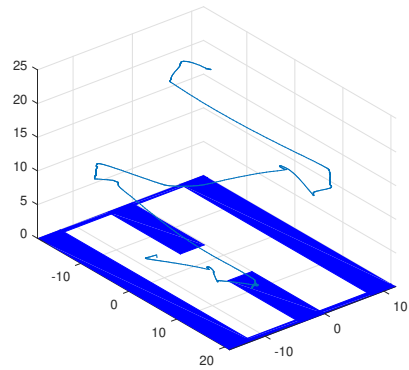
(a) Map trail using random motion



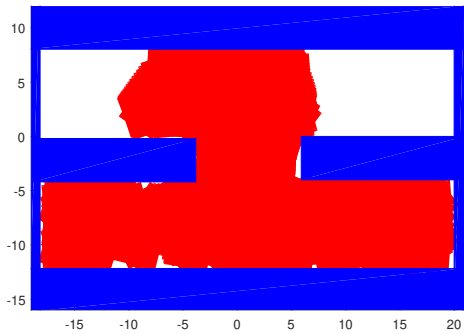
(b) Map trail using reinforcement learning



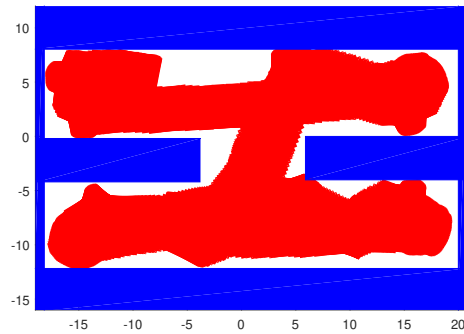
(c) Corresponding time cost using random motion



(d) Corresponding time cost using reinforcement learning

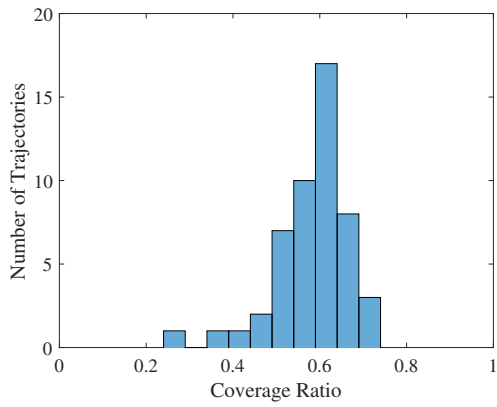


(e) Map coverage using random motion

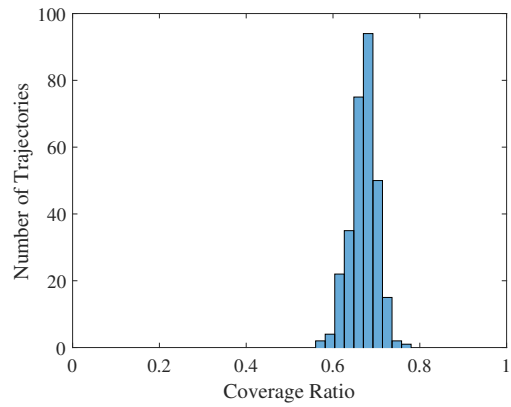


(f) Map coverage using reinforcement learning

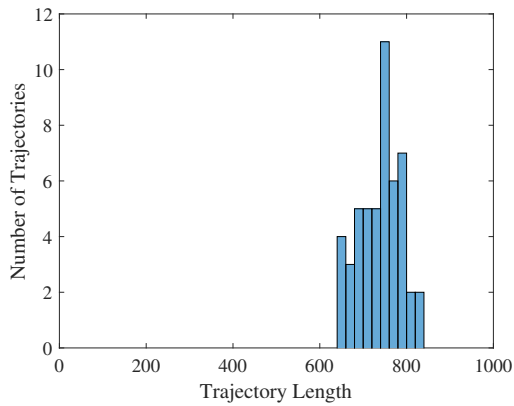
Figure 5.11: Map coverage and trajectories for one episode



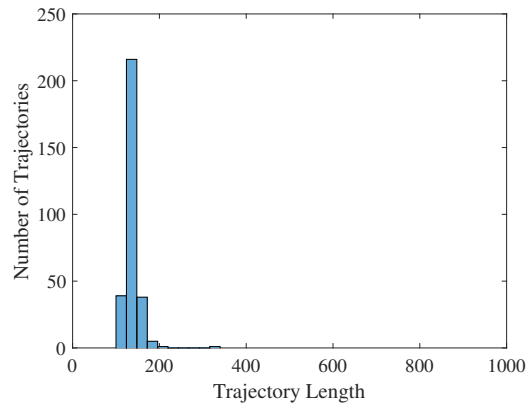
(a) Coverage ratio using random motion



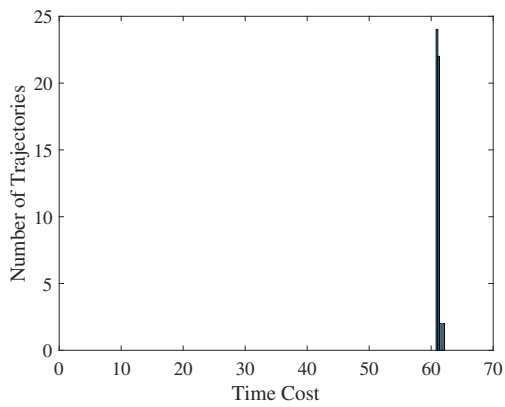
(b) Coverage ratio using reinforcement learning



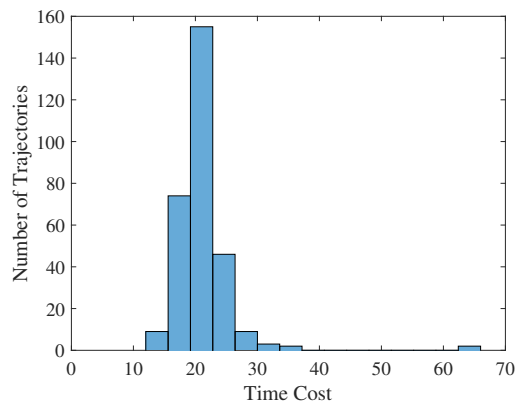
(c) trajectory length using random motion



(d) trajectory length using reinforcement learning



(e) Time cost using random motion



(f) Time cost using reinforcement learning

Figure 5.12: Cleaning robot trajectories in simulation

3000 steps. In 5.12(e) the time it took a robot using random motion to reach the step limit was around 60s. However, RL networks enabled the robot to complete tasks within the step limit. The exact time-cost is plotted in Figure 5.12(f). The range in time taken to complete the cleaning task using RL was mainly between 17 – 27s. The time-cost of random motion was around three times as much as RL according to the histograms in Figures 5.12(e) and 5.12(f).

In this work, we focused on completing cleaning map coverage within a limited time frame due to practical applications in and requirements for an airline cabin environment. RL networks completed the coverage task three times faster and with

ve times less movement than random motion. Therefore, RL networks show superior performance in autonomous robot navigation through environments that are complex and where time to complete the task is limited.

#### 5.2.4 Demo in Changed Environments

The training process of reinforcement learning requires a lot of time. Hence, reinforcement learning is considered to have difficulty to adapt to changed environments. Because the optimal policy that was trained for one case becomes invalid in other cases. To explore this concern, we conducted map coverage experiments in changed environments. We present demonstrations with optimal policy that enable the robot to complete coverage missions in a changed environment. The efficiency of cleaning performance decreases, but the robot can still finish missions, albeit with longer time and trajectory cost.

The expert trajectories of GAIL helped the robot to learn specific behavior patterns during training successfully. In the expert trajectories, the robot was controlled to move

straight into one corner to cover the corner area. The robot turned around and moved out. Then the robot was controlled to move to the next row to repeat the same behaviors. Hence, the expert trajectories taught the robot to sense the uncovered corners. When the robot was on the aisle. When the robot found one uncovered corner, the expert trajectories provided the behavior patterns that the robot moved into the corner. When the robot was in a corner, the expert trajectories taught the robot to move out. The robot learned the behavior models for the structure of aisle and corners based on the expert trajectories of GAIL.

When the environment has been changed, the aisle and corners are still exist on the map but no longer at the same location. When the robot meets the same structures on a changed map, the robot conducts proper actions based on the behavior models from training. When the same cell has a larger scale, the robot can meet familiar structures on an expanded map. The corners become larger, but more free space bring no difficulties to the robot. With the efficiency losing, the robot can still complete the map coverage.

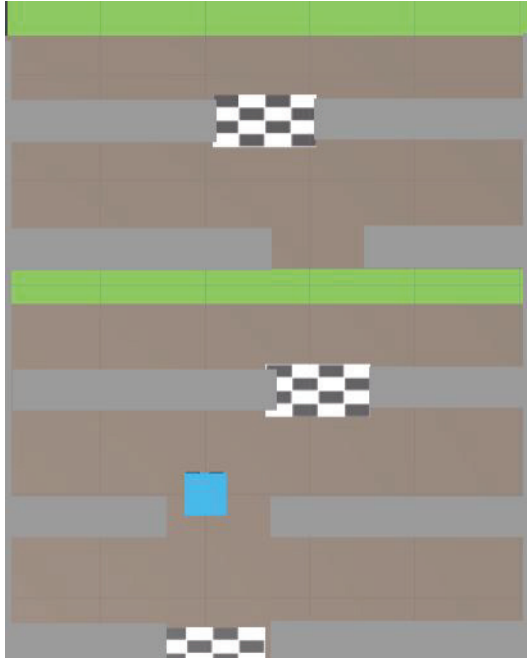
Specific behavior models were gained from training. Additionally, the robot also learned the total action trends from the expert trajectories. I divided the whole cabin area into multiple cells. I tended to control the robot to clean each row first and move to the next row. I did not choose to complete the left side or the right side first. In the testing, the robot had the same trajectories. The robot moved horizontally in one row first, then moved to the next one.

## **Shape Changed Environment**

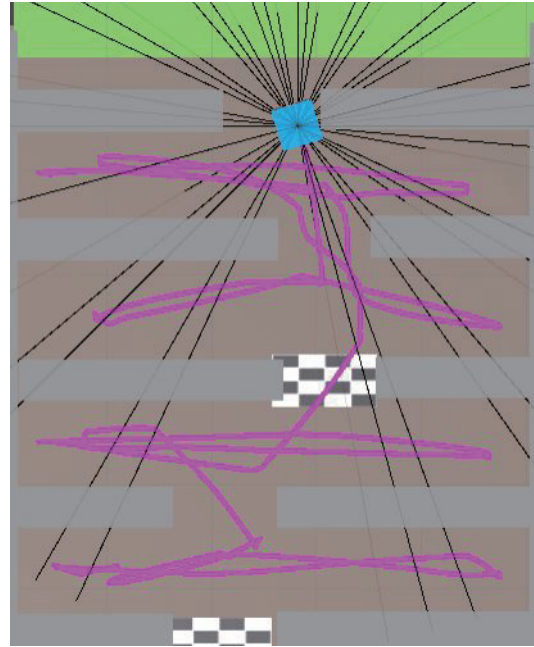
Shape modifications are applied to the testing environment in 5.13(a). The testing environment contains two cells for cleaning, and the two cells are divided by black-and-white

bounds. As shown in 5.13(a), one side of the blocks were extended to change the shape of testing cell. The total width length is extended to 1.25 times of the training clean cell width. The 5.13(b) is a screenshot for testing. The purple lines recorded the trails left by the robot. The robot covered the bottom cell then moved to the unvisited cell to perform cleaning. The 5.13(c) is a Matlab figure that recorded the trajectories of robot in testing environment. The figure was selected randomly from 100 episodes of records. The blue areas were blocks and the red areas were covered area the same as testing in the original clean cell. The 5.13(d) shows that in most cases the robot are able to complete the cleaning mission and keep the coverage ratio around 60% – 70%. However, the robot sometimes failed the mission in testing. Sometimes the performance became unstable due to environmental modifications. The trajectory length was shown in 5.13(e) and the time cost was shown in 5.13(f). Both trajectory length and time cost increased when comparing with the same data of the original clean cell.

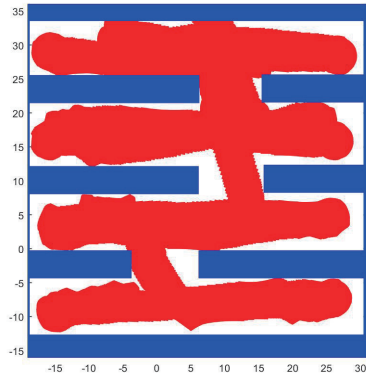
In this shape modified environment, the robot reaches all corners to cover unvisited area. In other words, the policy still works in this testing environment to provide cleaning functions. However, the optimal trajectories and corresponding minimum time cost were not generated in this new area. When the robot moves into corners, we observed that the robot slowed down. When the robot moves into one corner, the rays that detect the back side of the robot will receive a larger value that represents the distance of blocks or walls. However, the robot received small values of the original cleaning cell in training process. The different ranges of observation values caused by environmental modifications lead to unstable performance in testing. Also, the modified environments increased the trajectory length and time costs in testing.



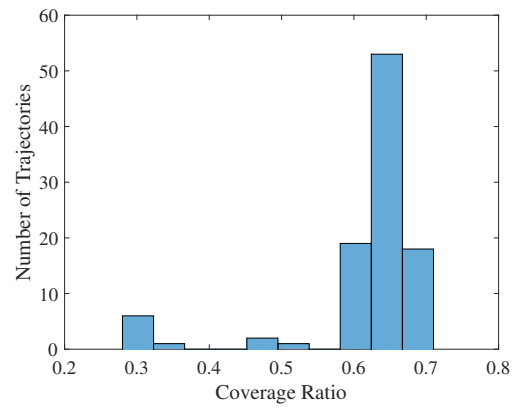
(a) Changed environment for testing



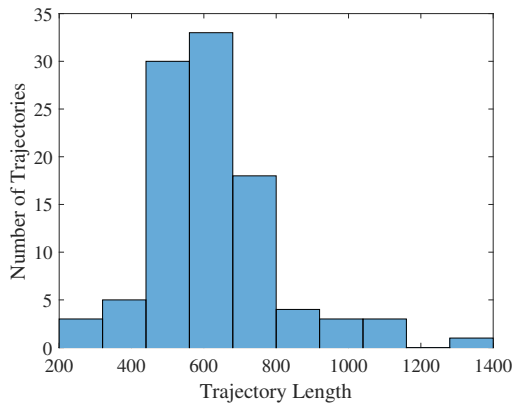
(b) Trajectories in changed environment



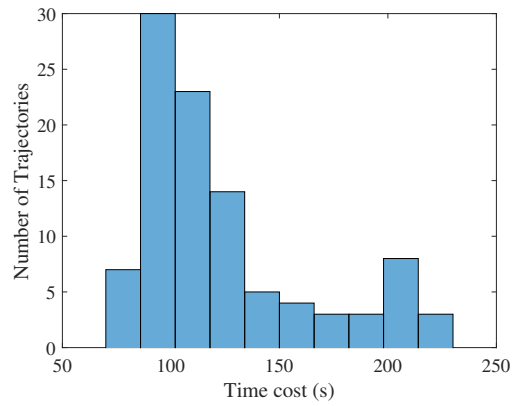
(c) A screenshot for changed environment



(d) Coverage ratio for changed environment



(e) Trajectory length for changed environment



(f) Time cost for changed environment

Figure 5.13: Demos in Shape Changing Environment

## Cabin Cell with Three Column Seats

The testing environment was modified according to the passenger seat setting of large planes. In training process, the cleaning cell was built according to small planes, such as airbus. The testing environment is built based on Boeing 767 or 777 types that are for long time flights.

Figure 5.14(a) shows the top-view of testing environment. The environment contains two cells as well. Unlike the cleaning cell, two black-and-white bounds are set in the middle of the environment to divide the two cells. When the robot finished the cleaning in current cell, the two bounds disappeared and the robot chose the near exit to move to the next cell. According to 5.14(a), the two columns of blocks were added to three columns. The total width length is extended by a factor of two. The 5.14(e) is a screenshot of the testing. The purple lines are the trajectory of robot during testing. Figure 5.14(c) is a figure that generated by Matlab showing the coverage area within the testing environment. The testing continued for 100 episodes and Fig. 5.14(c) was chosen randomly from the 100 recorded testing. Figure 5.14(d) shows that the coverage ratio for the expand environment is in the range of 55% – 65%. The robot was able to complete missions in most episodes. The failed cases still exist due to environmental modifications. Two exits are set to connect with the next cell. However, when the robot finished cleaning, it moved across only one exit to the next cell and never returned. Hence, the empty area of the remaining exit was as unvisited during testing. Hence, the coverage ratio dropped. Figure 5.14(e) shows the trajectory length and Fig. 5.14(f) shows the time costs of testing episodes. Both trajectory length and time costs increased when compared with the same size of original cleaning cells.

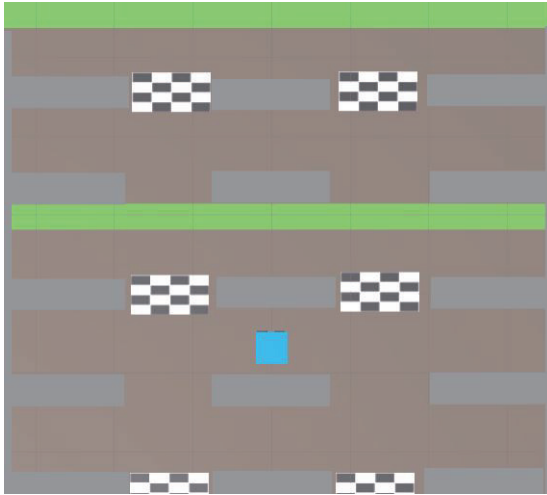
In these expanded environments, when the robot moves to one corner, the rays that provided the simulated LIDAR function will detect nothing at the opposite directions. In the previous testing environment, the rays are able to reach the opposite walls when robot is at one side. However, the length of rays are limited, and the rays are unable to touch walls in the opposite directions. The empty detection caused the robot to wander around without reaching uncovered landmarks directly. Hence, the robot using reinforcement learning policy was able to perform cleaning missions in the three column block environment, but efficiency was reduced.

## Multiple Cell Build

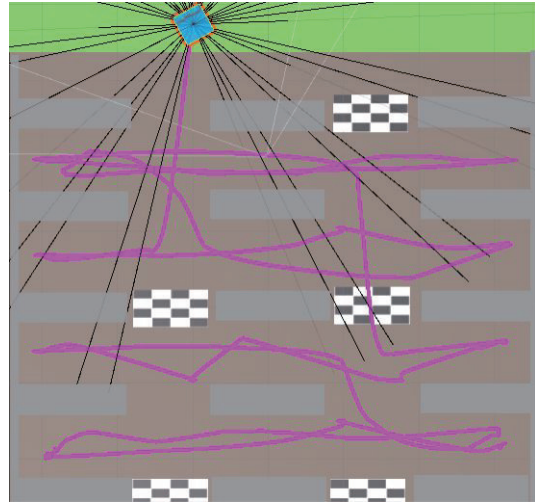
Multiple cleaning cells were connected to build a whole cabin. Experiments were conducted to test whether the robot can complete the cleaning mission of the whole cabin environment. Figure 5.15(a) is a top-view of the whole cabin environment. The previous experimental environments contained two cleaning cells which were divided by black-and-white boundaries. The robot only needed to clean the current cell and move to the other. However, the cabin environment contains 10 cells to be cleaned. In the simulation, 20 lines of passenger seats were created and 10 exits landmarks were used to guide the robot to move to the next cleaning cell. This environment is used to observe if the robot is able to repeat its behaviors with trained with a policy to complete the whole given area.

Figure 5.15(b) is a screenshot of robot motions during testing. Purple lines represent the trajectory followed by the robot. it can be seen that the robot finished cleaning the current cell then moved to the next one. it can be seen that, the robot cleaned each cell one by one to conduct the cleaning process. The robot followed the framework that we designed.

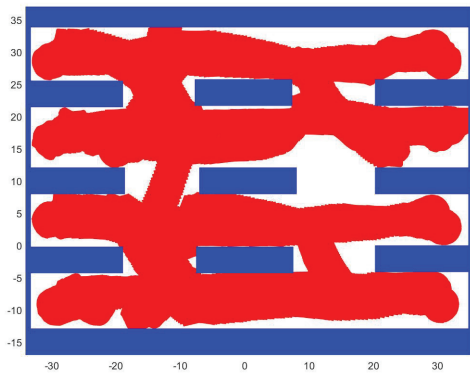




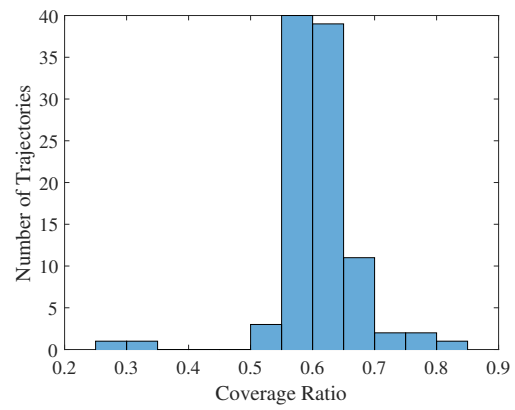
(a) Expand environment for testing



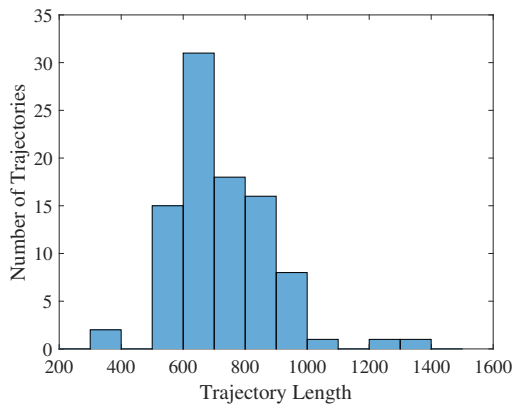
(b) Trajectories in Expand environment



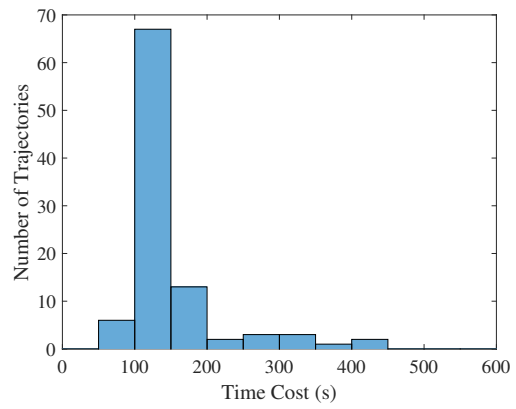
(c) A screenshot for expand environment



(d) Coverage ratio for expand environment



(e) Trajectory length for expand environment

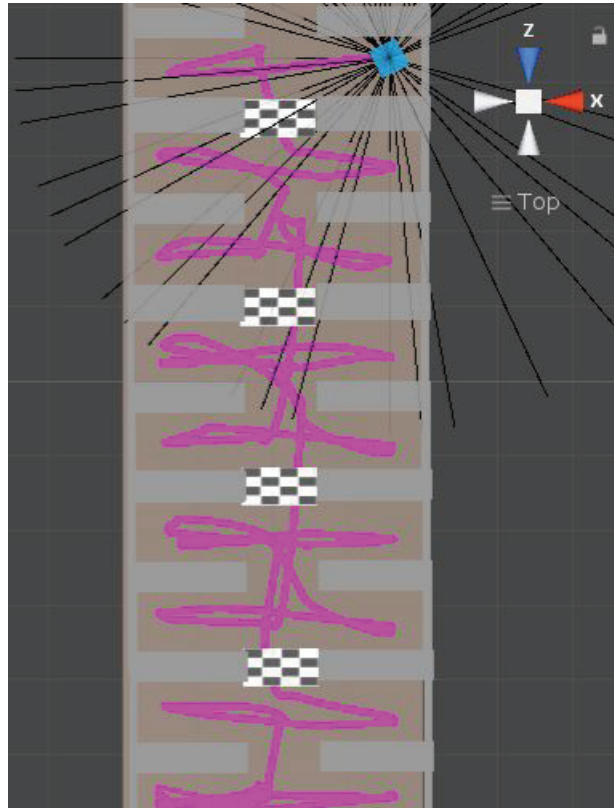


(f) Time cost for expand environment

Figure 5.14: Demos in Expand Environment



(a) Trajectory length for expand environment



(b) Trajectory length for expand environment

Figure 5.15: Demos in Expand Environment

The policy using reinforcement learning training generated robot motions that enable the robot to complete our project task.

## Chapter 6

### Conclusion

This paper presents a novel approach for environmental complexity measurement and CCPP demonstrations with a high coverage ratio using reinforcement learning. In order to measure environmental complexity, it takes robot size, environmental size, obstacle number, and obstacle locations as inputs for measurement. This approach provides a more comprehensive environmental complexity measurement. Lastly, our method gains complexity value for 2D environmental maps and reflects a reliable complexity level of each map. Thus, our approach is a promising measurement method to provide environmental complexity reference for CCPP approaches.

For the CCPP demonstrations using reinforcement learning, the policy takes simulated LIDAR functions as observations and robot actions as outputs. We designed a framework that divided the whole environment with repeated obstacles into multiple sample cells. For a single sample cell, we conducted training using reinforcement learning and GAIL with intrinsic curiosity rewards. Scalar figures were generated to record statistic data of training. After the training process, our robot using reinforcement learning were tested in training environment to cover map area. Comparing with random motion methods, our application keeps a balance between high coverage ratio and time costs. Then, we tested the performance of the policy in modified environments. The robot was able to complete map coverage missions, while the efficiency dropped due to environmental modifications. Also, we built

a whole cabin area with multiple cleaning sample cells as a testing environments. In this environment, the robot was able to clean each sample cell one by one to cover the whole environmental area. Hence, our CCPP application is promising to apply to environments where repeated obstacles exist.

## Bibliography

- [1] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.
- [2] A. Khan, I. Noreen, and Z. Habib, “On complete coverage path planning algorithms for non-holonomic mobile robots: Survey and challenges,” *J. Inf. Sci. Eng.*, vol. 33, no. 1, pp. 101–121, 2017.
- [3] H. Choset, “Coverage for robotics—a survey of recent results,” *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.
- [4] P. Zhou, Z.-m. Wang, Z.-n. Li, and Y. Li, “Complete coverage path planning of mobile robot based on dynamic programming algorithm,” in *2nd International Conference on Electronic & Mechanical Engineering and Information Technology*, Atlantis Press, 2012.
- [5] R. N. De Carvalho, H. Vidal, P. Vieira, and M. Ribeiro, “Complete coverage path planning and guidance for cleaning robots,” in *ISIE’97 Proceeding of the IEEE International Symposium on Industrial Electronics*, vol. 2, pp. 677–682, IEEE, 1997.
- [6] C. Luo, S. X. Yang, D. A. Stacey, and J. C. Jofriet, “A solution to vicinity problem of obstacles in complete coverage path planning,” in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1, pp. 612–617, IEEE, 2002.
- [7] S. X. Yang and C. Luo, “A neural network approach to complete coverage path planning,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 718–724, 2004.
- [8] Y.-H. Choi, T.-K. Lee, S.-H. Baek, and S.-Y. Oh, “Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5788–5793, IEEE, 2009.
- [9] A. Janchiv, D. Batsaikhan, G. hwan Kim, and S.-G. Lee, “Complete coverage path planning for multi-robots based on,” in *2011 11th International Conference on Control, Automation and Systems*, pp. 824–827, IEEE, 2011.
- [10] A. Janchiv, D. Batsaikhan, B. Kim, W. G. Lee, and S.-G. Lee, “Time-efficient and complete coverage path planning based on flow networks for multi-robots,” *International Journal of Control, Automation and Systems*, vol. 11, no. 2, pp. 369–376, 2013.

- [11] A. Phillips, “The topology of roman mosaic mazes,” *Leonardo*, pp. 321–329, 1992.
- [12] S. Nadler, *Continuum theory: an introduction*. CRC Press, 1992.
- [13] M. S. McClendon *et al.*, “The complexity and difficulty of a maze,” in *Bridges: Mathematical Connections in Art, Music, and Science*, pp. 213–222, Bridges Conference, 2001.
- [14] Y. Liu, X. Lin, and S. Zhu, “Combined coverage path planning for autonomous cleaning robots in unstructured environments,” in *2008 7th World Congress on Intelligent Control and Automation*, pp. 8271–8276, IEEE, 2008.
- [15] K. M. Hasan, K. J. Reza, *et al.*, “Path planning algorithm development for autonomous vacuum cleaner robots,” in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, pp. 1–6, IEEE, 2014.
- [16] C. E. Taylor, S. F. Lau, E. C. Blair, A. Heninger, and E. Ng, “Robot cleaner with improved vacuum unit,” Aug. 7 2008. US Patent App. 11/574,290.
- [17] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.
- [18] T. Oksanen and A. Visala, “Coverage path planning algorithms for agricultural field machines,” *Journal of field robotics*, vol. 26, no. 8, pp. 651–668, 2009.
- [19] H. Choset and P. Pignon, “Coverage path planning: The boustrophedon cellular decomposition,” in *Field and service robotics*, pp. 203–209, Springer, 1998.
- [20] J. W. Milnor, M. Spivak, and R. Wells, *Morse theory*, vol. 1. Princeton university press Princeton, 1969.
- [21] J. F. Canny and M. C. Lin, “An opportunistic global path planner,” *Algorithmica*, vol. 10, no. 2-4, pp. 102–120, 1993.
- [22] H. Choset, “Coverage of known spaces: The boustrophedon cellular decomposition,” *Autonomous Robots*, vol. 9, no. 3, pp. 247–253, 2000.
- [23] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, “Morse decompositions for coverage tasks,” *The international journal of robotics research*, vol. 21, no. 4, pp. 331–344, 2002.
- [24] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.
- [25] A. Bircher, M. S. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, “Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots,” *Autonomous Robots*, vol. 40, 11 2015.

- [26] J. Zhang, Z. Yu, X. Wang, Y. Lyu, S. Mao, S. C. Periaswamy, J. Patton, and X. Wang, "Rfhui: An rfid based human-unmanned aerial vehicle interaction system in an indoor environment," *Digital Communications and Networks*, 2019.
- [27] J. Zhang, X. Wang, Z. Yu, Y. Lyu, S. Mao, S. C. Periaswamy, J. Patton, and X. Wang, "Robust rfid based 6-dof localization for unmanned aerial vehicles," *IEEE Access*, vol. 7, pp. 77348–77361, 2019.
- [28] J. Zhang, Z. Yu, X. Wang, Y. Lyu, S. Mao, S. C. Periaswamy, J. Patton, and X. Wang, "Rfhui: An intuitive and easy-to-operate human-uav interaction system for controlling a uav in a 3d space," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 69–76, 2018.
- [29] J. Zhang, Z. Yu, X. Wang, Y. Lyu, S. Mao, S. C. Periaswamy, J. Patton, and X. Wang, "Rfhui: An rfid based human-unmanned aerial vehicle interaction system in an indoor environment," *Digital Communications and Networks*, vol. 6, no. 1, pp. 14–22, 2020.
- [30] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.
- [31] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the ekfslam algorithm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3562–3568, IEEE, 2006.
- [32] J. Zhang, Y. Lyu, J. Patton, S. C. Periaswamy, and T. Roppel, "Bfvp: A probabilistic uhf rfid tag localization algorithm using bayesian filter and a variable power rfid model," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 10, pp. 8250–8259, 2018.
- [33] J. Zhang, Y. Lyu, J. Patton, S. C. G. Periaswamy, and T. Roppel, "Bfvp: A probabilistic uhf rfid tag localization algorithm using bayesian filter and a variable power rfid model," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 10, pp. 8250–8259, 2018.
- [34] X. Wang, Z. Yu, and S. Mao, "Deepml: Deep lstm for indoor localization with smartphone magnetic and light sensors," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.
- [35] X. Wang, J. Zhang, Z. Yu, E. Mao, S. C. Periaswamy, and J. Patton, "Rfthermometer: A temperature estimation system with commercial uhf rfid tags," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2019.
- [36] X. Wang, Z. Yu, and S. Mao, "Indoor localization using smartphone magnetic and light sensors: A deep lstm approach," *Mobile Networks and Applications*, pp. 1–14, 2019.
- [37] X. Wang, J. Zhang, Z. Yu, S. Mao, S. C. Periaswamy, and J. Patton, "On remote temperature sensing using commercial uhf rfid tags," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10715–10727, 2019.

- [38] X. Wang, X. Wang, and S. Mao, “Resloc: Deep residual sharing learning for indoor localization with csi tensors,” in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, IEEE, 2017.
- [39] X. Wang, X. Wang, S. Mao, J. Zhang, S. C. Periaswamy, and J. Patton, “Deepmap: Deep gaussian process for indoor radio map construction and location estimation,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2018.
- [40] Y. Lin, J. Hyypä, and A. Jaakkola, “Mini-uav-borne lidar for fine-scale mapping,” *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 3, pp. 426–430, 2010.
- [41] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, IEEE, 2011.
- [42] L. A. James, D. G. Watson, and W. F. Hansen, “Using lidar data to map gullies and headwater streams under forest canopy: South carolina, usa,” *Catena*, vol. 71, no. 1, pp. 132–144, 2007.
- [43] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, “Large-scale 6-dof slam with stereo-in-hand,” *IEEE transactions on robotics*, vol. 24, no. 5, pp. 946–957, 2008.
- [44] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, IEEE, 2011.
- [45] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *European conference on computer vision*, pp. 834–849, Springer, 2014.
- [46] M. Labbé and F. Michaud, “Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.
- [47] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “Orb-slam: a versatile and accurate monocular slam system,” *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
- [48] J. Zhang, Y. Lyu, T. Roppel, J. Patton, and C. Senthilkumar, “Mobile robot for retail inventory using rfid,” in *2016 IEEE international conference on Industrial technology (ICIT)*, pp. 101–106, IEEE, 2016.
- [49] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.
- [50] P. Abbeel and A. Y. Ng, “Inverse reinforcement learning,” 2010.
- [51] Y. Chen, J. Z. Wang, and R. Krovetz, “Clue: cluster-based retrieval of images by unsupervised learning,” *IEEE transactions on Image Processing*, vol. 14, no. 8, pp. 1187–1201, 2005.



- [52] T. Roppel, Y. Lyu, J. Zhang, X. Xia, *et al.*, “Corrosion detection using robotic vehicles in challenging environments,” in *CORROSION 2017*, NACE International, 2017.
- [53] X. Xia, T. Roppel, J. Zhang, Y. Lyu, S. Mao, S. C. Periaswamy, and J. Patton, “Enabling a mobile robot for autonomous rfid-based inventory by multilayer mapping and aco-enhanced path planning,”
- [54] X. Wang, X. Wang, and S. Mao, “Cifi: Deep convolutional neural networks for indoor localization with 5 ghz wi-fi,” in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017.
- [55] X. Wang, X. Wang, and S. Mao, “Deep convolutional neural networks for indoor localization with csi images,” *IEEE Transactions on Network Science and Engineering*, 2018.
- [56] X. Wang, X. Wang, and S. Mao, “Rf sensing in the internet of things: A general deep learning framework,” *IEEE Communications Magazine*, vol. 56, no. 9, pp. 62–67, 2018.
- [57] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [58] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” 2014.
- [59] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, pp. 1057–1063, 2000.
- [60] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, 2015.
- [61] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [62] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [63] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, pp. 1928–1937, 2016.
- [64] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in neural information processing systems*, pp. 4565–4573, 2016.
- [65] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

- [66] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [67] H. Van Seijen, M. Fatemi, J. Romoff, R. Larocche, T. Barnes, and J. Tsang, “Hybrid reward architecture for reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 5392–5402, 2017.
- [68] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, 2017.
- [69] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, “Ai2-thor: An interactive 3d environment for visual ai,” *arXiv preprint arXiv:1712.05474*, 2017.
- [70] R. H. Creighton, *Unity 3D game development by example: A Seat-of-your-pants manual for building fun, groovy little games quickly*. Packt Publishing Ltd, 2010.
- [71] Y. F. Chen, M. Everett, M. Liu, and J. P. How, “Socially aware motion planning with deep reinforcement learning,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, IEEE, 2017.
- [72] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “Ros: an open-source robot operating system,” in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, (Kobe, Japan), May 2009.
- [73] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” *arXiv preprint arXiv:1809.02627*, 2018.
- [74] MATLAB, *version 9.4.0.813654 (R2018a)*. Natick, Massachusetts: The MathWorks Inc., 2018.
- [75] C. E. Shannon, “A mathematical theory of communication,” *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [76] S. Vajapeyam, “Understanding shannon’s entropy metric for information,” *arXiv preprint arXiv:1405.2061*, 2014.
- [77] P. Lison, “An introduction to machine learning,” *Language Technology Group (LTG)*, 1, vol. 35, 2015.
- [78] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [79] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.

- [80] S. Kakade and J. Langford, “Approximately optimal approximate reinforcement learning,” in *ICML*, vol. 2, pp. 267–274, 2002.
- [81] D. A. Pomerleau, “Efficient training of artificial neural networks for autonomous navigation,” *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.
- [82] A. Y. Ng, S. J. Russell, *et al.*, “Algorithms for inverse reinforcement learning,” in *Icml*, vol. 1, p. 2, 2000.
- [83] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.