Coverage Path Planning in Dynamic Environment through Guided Reinforcement Learning

Ming Lung Raymond Chong

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-123
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-123.html

May 13, 2022

Copyright © 2022, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I'd like to acknowledge my advisor Professor Claire Tomlin for her help with this work, my mentor Dr. Shankar Deka for his insight, conversations and help with this project without whom this wouldn't have been possible. Finally, my parents (CST, YCY) for their support throughout my education.

Coverage Path Planning in Dynamic Environment through Guided Reinforcement Learning

by Ming Lung Raymond Chong

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Claire J. Tomlin
Research Advisor

May 13, 2022

(Date)

* * * * * * *

Shankar Anand Deka

Dr. Shankar A. Deka
Second Reader

May 13, 2022

(Date)

Coverage Path Planning in Dynamic Environment through Guided Reinforcement Learning

by

Ming Lung Raymond Chong

A submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire J. Tomlin, Chair
Dr. Shankar A. Deka

Spring 2022

Coverage Path Planning in Dynamic Environment through Guided Reinforcement Learning

Copyright 2022
by
Ming Lung Raymond Chong

Abstract

Coverage Path Planning in Dynamic Environment through Guided Reinforcement Learning

by

Ming Lung Raymond Chong

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Claire J. Tomlin, Chair

Path planning for mobile robots has been, and remains, a design and development challenge. In addition to well known path planning tasks such as searching for a shortest path from a start point to its end, Coverage Path Planning (CPP) is a task to design a trajectory by which agents can traverse every point in an area of interest. Generating an efficient CPP algorithm is, therefore, far more challenging than calculating a shortest path. In particular, environments with both static and dynamic obstacles require robots to navigate efficiently while simultaneously avoiding obstructions and other potential agents. Traditional approaches to address the challenges that dynamic obstacles introduce often require replanning strategies for coverage as the environment changes. Such replanning mechanisms and computations are both expensive and suboptimal, often resulting in path detours.

In this report, we first investigate several methods for conducting CPP with dynamic obstacles using a graph search algorithm (Spanning Tree Coverage) and on-policy reinforcement learning algorithm (Proximal Policy Optimization) separately. Then, we introduce a guided reinforcement learning approach for CPP which incorporates a unique reward structure as well as additional coverage information generated from Spanning Tree Coverage to help guide the reinforcement learning agent algorithm. We evaluate our proposed algorithm, Coverage Path Planning through Guided Reinforcement Learning (CPP-GRL), across different settings (grid sizes and obstacle locations) and compare with prior research methods. Experimental results show that CPP-GRL generalizes well for both stochastic and deterministic moving obstacles and performing very similarly to other control-based and learning-based algorithms, but with fewer constraints (such as following specific control laws) and lower computational power (such as using Convolutional Neural Network instead).

Contents

Acknowledgments

I'd like to acknowledge my advisor Professor Claire Tomlin for her help with this work, my mentor Dr. Shankar Deka for his insight, conversations and help with this project without whom this wouldn't have been possible. Finally, my parents (CST, YCY) for their support throughout my education.

Chapter 1

Introduction

Advances in reinforcement learning have affected applications ranging from image classification and voice recognition to text translation and robotics. Much literature and research has focused on reinforcement learning techniques in path planning, particularly concerning robotic systems, optimal control, and global minimum cost for multi-agent systems [5], [16], [11].

Motivations of CPP are inspired in applications such as cleaning robots and agriculture. With drones conducting CPP on agriculture farmland, it is not sufficient to find any route that completely covers the land but rather a path that is optimal in order to minimize certain costs (such as batteries in the drones). In the area of precision farming, the CPP problem is presented for tasks such as harvesting, seeding, spraying, applying fertilizer or taking imagery of the land. One particular application is deploying drones onto farmland to take imagery of the land while avoiding the cloud shadows. According to NASA [13], approximately two thirds of the surface of the Earth is covered by clouds at any given time. As a result, clouds cast shadows on the ground can interfere with the interpretation of multi-spectral and hyper-spectral image data. Hence, one can first develop a Long short-term memory (LSTM) neural network model for predicting the cloud movement. From there, the prediction can help assist the trajectory planning and decision making of the drones. More detail of the application can be found at Chapter 6.

In many existing path planning algorithms, both graph-based and learning-based, main objectives include minimizing cost (determined by the weights of the edges) and finding the shortest path. Examples of traditional path planning algorithms, such as Dijkstra's, A* and Sampling-based algorithms (like RRT, etc.) are beneficial but often involve oversimplified real-life scenarios [10]. With a learning-based approach, however, the agent can adapt to a dynamic environment. Currently, within learning-based literature, there is limited research focusing on coverage path planning or, more precisely, coverage path planning with dynamic moving obstacle avoidance given a fixed coverage map.

With all the above in mind, this project aims to develop a reinforcement learning agent that masters coverage path planning while avoiding dynamic moving obstacles along its trajectory. The main objective is to develop an algorithm/agent that can conduct coverage path planning with dynamic obstacles in a fully observable environment with information on the evolution of dynamic obstacles over a fixed time-horizon.

In this work, we introduce a guided reinforcement learning approach, incorporating existing benefits of reinforcement learning algorithms and traditional coverage path planning using graph search algorithms. We use coverage path planning with guided reinforcement learning (CPP-GRL) to address the problem of conducting coverage in an area with dynamic obstacles.

Chapter 2

Background

Past works on coverage path planning algorithms can be broken down into two main types of technique: graphical-based and learning-based approaches. Both present their benefits alongside challenges for coverage path planning tasks.

2.1 Spanning-Tree Coverage (STC)

The STC algorithm [6] is a graphical search-based coverage path planner that computes the optimal robot path in the minimum number of steps, given some assumptions. The STC takes O(N) runtime where N, the number of cells comprising the area, represents the size of an area of interest. It first subdivides the work-area into disjoint cells corresponding to a 2D grid with square cells. Next, the STC converts the cells into nodes and connects the edges of those nodes with adjacent ones. Once connected, the nodes and edges can be represented graphically, and if the graph is connected, a minimum spanning tree can be constructed. In cases where the weights of the edges are equal, multiple minimum spanning trees become possible. By circumnavigating the Minimum Spanning Tree (MST), one can generate a coverage path for the given 2D grid. An example of the STC algorithm is shown in Figure 3.1.

2.2 Proximal Policy Optimization (PPO)

Developed by OpenAI, Proximal Policy Optimization (PPO) [14] is a Deep Reinforcement Learning (DRL) algorithm based on the concept of Policy Gradient. Policy Gradient is an on-policy RL method as it attempts to optimize the agent's policies π directly by applying stochastic gradient ascent to the policy parameters. More specifically, within the DRL framework, a Deep Neural Network (DNN) [12] is used to facilitate the mapping of actions $a \in A$ to states $s \in S$. One of the advantages of Policy Gradient based algorithms is that the training data is used only once and then discarded, which helps limit the amount of memory needed for training. However, this approach also reflects poor data efficiency as compared to other Q-learning methods such as Deep Q-Network (DQN), where sample data can be used multiple times.

The goal of an agent in a reinforcement learning is to maximize the discounted cumulative reward function $R(\tau)$, $\tau = (s_0, a_0, s_1, a_1, ...)$ from the current state up to a terminal state at time T. With a movement budget, the discounted cumulative reward function is defined as a finite horizon sum, which can be represented as

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{2.1}$$

with the discount factor $\gamma \in [0, 1]$ encoding the importance of immediate and future rewards. r_t represents the reward the agent receives for taking action, a_t, at time step, t. The output is maximized by training the agent's policy π. The policy can be deterministic with $\pi(s)$ such that $\pi : S \to A$ or probabilistic with $\pi(a|s)$ such that $\pi : S \times A \to \mathbb{R}$, resulting in a probability distribution over the action space for each $s \in S$.

Policy: A policy in RL focuses on maximizing the expected discounted cumulative reward by mapping states or observations to actions. A policy provides a probability distribution over actions given states, that fully define the behavior of the agent. While most policies can be deterministic, they are often stochastic. $\pi(s, a)$ is the probability of taking action a in state s under policy π.

Q-Function: $Q^\pi(s, a)$ is the expected value of the return of the policy after taking action a in state s and therefore following π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[R(\tau)|s, a] \tag{2.2}$$

Value-Function: $V^\pi(s)$ is the expected value of the return of the policy in state s

$$V^\pi(s) = \mathbb{E}_\pi[R(\tau)|s] \tag{2.3}$$

$V^\pi(s)$ can be described as the weighted probability average of discounted cumulative rewards over all possible actions in state s. Hence, the relationship between Q-function and value function can be expressed as

$$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a) \tag{2.4}$$

Policy Gradient: Building off the foundation of the definition of a policy, a policy gradient method focuses on optimizing a parameterized policy π_θ and θ represents the parameters learned from the data during training. For example, if neural networks were used for training,

θ represents the weights in the neural network. In policy gradient method, one can define the objective function as the expected discounted cumulative reward

$$J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)] \tag{2.5}$$

Therefore, one seeks θ^* which maximizes $J(\theta)$ such that

$$\theta^* = \arg\max_\theta J(\theta) \tag{2.6}$$

Advantage Function: The advantage function, which reflects how advantageous a given action a is compared to the expectation of all actions is defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.7}$$

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient scent algorithm. Let $\hat{A}_t$ as a estimator of the advantage function. Using the definition of $R(\tau)$ and linearity of expectation, one can derive $\nabla_\theta J(\theta)$. A commonly used gradient estimator follows the form

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{i=0}^{T} \nabla_\theta \log(\pi_\theta(s_t, a_t))R(\tau)\right] \approx \mathbb{E}_{\pi_\theta}\left[\sum_{i=0}^{T} \nabla_\theta \log(\pi_\theta(s_t, a_t))\hat{A}^\pi(s_t, a_t)\right] \tag{2.8}$$

Since the policy π_θ is represented by a neural network, the focus is thus on the policy loss which can be defined as

$$L^{PG}(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{i=0}^{T} \nabla_\theta \log(\pi_\theta(s_t, a_t))\hat{A}^\pi(s_t, a_t)\right] \tag{2.9}$$

$L^{PG}(\theta)$ is the general loss function for policy gradient method. PPO, PPO-Penalty, PPO-Clip are specific policy gradient methods. Below is the algorithm used for PPO-Clip.

Algorithm 1: PPO-Clip

```
Input: Initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
Output: A stochastic or deterministic policy $\pi^*$
for $k = 0, 1, 2, ...$ do
    Collection set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the
      environment
    Compute reward-to-go $\hat{R}_t$
    Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_k$
    Update the policy by maximizing the PPO-Clip objective:
        $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t) \right)$$
    *typically via stochastic gradient ascent with Adam
    Fit value function by regression on the mean-square error:
        $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2$$
    *typically via some gradient descent algorithm
end
```

2.3 Prior Work

Although coverage path planning with dynamic obstacles is an unexplored branch of CPP, there have been several previous related works. These focus on coverage path planning with specific applications in UAV and mobile cleaning robots with static obstacles, and existing research is divisible into two different categories: graphical search-based and reinforcement learning-based. All prior work has focused on one of these two approaches.

In 2011, Marija Dakulovic, Sanja Horvatic, and Ivan Petrovic proposed a coverage D* algorithm for path planning in a floor-cleaning mobile robot [4]. In 2014, Zengyu Cai, Shuxia Li, Yong Gan, Ran Zhang and Qikun Zhang also proposed a coverage planning algorithm for cleaning robots [3]. Both graphical techniques demonstrated ability to conduct coverage on deterministic path planning given static obstacles with high coverage rates and low repetition rates using D* and A* searches.

In addition to static obstacles, there has been research on CPP in unknown moving obstacles avoidance where researchers propose a time varying density function, which represents the importance of each point of interest and helps the system to avoid collision with obstacles. In [9], a Lloyd Newton method was proposed to have an optimal coverage control of time varying density function. Kooshkbaghi and Abdollahi [8] later presented a control law that makes the agent follow an unique density function while avoiding unknown moving obstacles for optimal coverage. Although the algorithm has been proven to outperform previous methods, such as Lloyd method, it is worth noting the design of the time varying density function has to be continuously differentiable.

As reinforcement learning increased in popularity with robotic applications, researchers began investigating the application of RL to coverage planning problems. In their respective publications, Boufous and Theile presented coverage path planning solutions that employed Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) [15]. Their algorithms were able to achieve 95% coverage with an agent given a limited number of time steps. As noted, the problem becomes much more challenging with dynamic moving obstacles should the agent need to avoid them while conducting coverage.

In addition to the pure learning-based approach, researchers also investigated hybrid approaches. In [7], the authors presented a hybrid RL approach, BA* wherein the agent has a partially observable view of the entire environment through a 3D sensor and begins covering the free areas of the environment by using a simple zigzag movement. When it reaches a point where there is no free area surrounding itself, the robot re-positions to an available free area as identified while performing the zigzag movement. Since there can be several free areas, as well as multiple possible paths to them, BA* deploys the A* search to find the closest one. However, due to the use of complex image-based observation and the DQN algorithm, the agent needs to be trained for more than 20,000 episodes.

Chapter 3

Methodology

Graphical-Based Search Approach: While Spanning-Tree Coverage (STC) Algorithms are capable of conducting complete coverage on a two-dimensional grid with static obstacles (where the dimensions must be even), we find it difficult to directly adapt such an algorithm to dynamic moving obstacles. In section 3.1, we present three different graph-based approaches.

Learning-Based Approach: Following the three graph-based methods, we present additional reinforcement learning approaches: a naive method and a guided reinforcement learning method in section 3.2.

3.1 Graph-Based: Constant Re-planning STC

The first and simplest solution is to apply the STC algorithm at each time step and re-plan accordingly with respect to the new state of the dynamic obstacles. Additionally, we incorporate the information from previously covered regions to inform the agent that these regions have been dealt with. An example of one iteration of STC is shown Figure 3.1.

More specifically, at time step i, the algorithm first subdivides the area of interest into 2D cell sizes and discards cells that have been previously visited or are currently being covered by obstacles at time step i. Then, a graph structure $G(V, E)$ is defined whereby the nodes V are the center points of each cell and the edges E are line segments that help connect these nodes to adjacent ones. Next, the algorithm constructs a spanning tree for G where a coverage path can then be generated surrounding the spanning tree. The agent then takes one step in the direction of the generated path. The process repeats until all cells have been visited.

Since running the STC algorithm for one agent at each time step takes O(N) runtime, where N represents the number of tiles (the area of interest broken down into square-shaped 1x1 cells) to cover, the maximum number of replannings (assuming no static obstacles) is also N. As a result, the total runtime required using this method is $O(N^2)$. However, we must also be aware of several edge cases and some heuristic designs when picking the path. We discuss this in later sections.

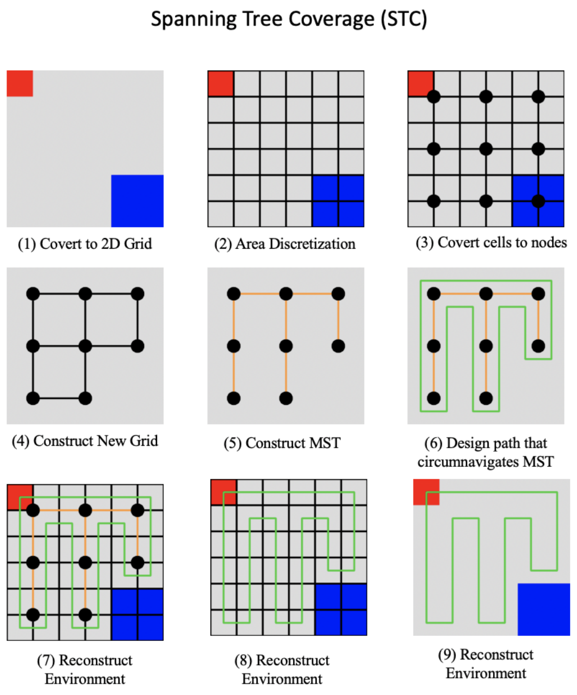

Spanning Tree Coverage (STC)
(1) Covert to 2D Grid
(2) Area Discretization
(3) Covert cells to nodes
(4) Construct New Grid
(5) Construct MST
(6) Design path that circumnavigates MST
(7) Reconstruct Environment
(8) Reconstruct Environment
(9) Reconstruct Environment

Figure 3.1: A simple guide to STC, Red = Agent, Blue = Obstacle

Heuristics Design

As previously mentioned, the STC algorithm can return more than one possible path (given the number of possible spanning trees); hence, it is critical to pick a path that minimizes the number of turns required. Since in actuality, each turn would require the agent to change direction (and thus necessitate the use of more energy/power), we impose a penalty every time the agent makes a turn. Thus, in the proposed algorithm, at each time step during replanning, the agent chooses the path with the minimum number of turns as shown in Figure 3.2.

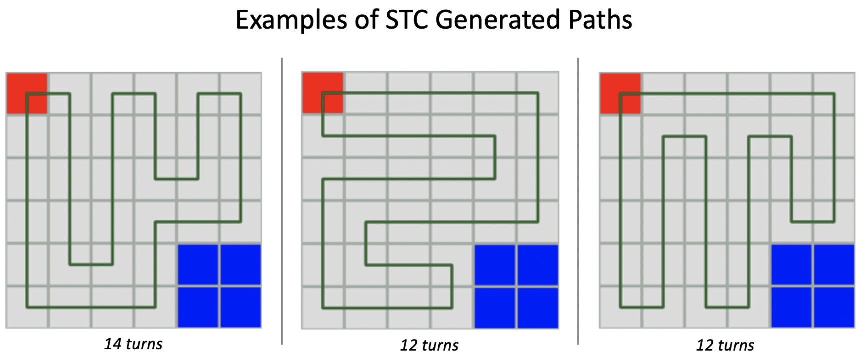


Figure 3.2: Examples of Different STC Generated Paths

Edge Case

The STC algorithm is only capable of conducting complete coverage on a grid that is not obstructed or blocked by obstacles. In the event that the obstacles/covered regions transform the grid into disjointed sets S, the proposed algorithm will run the STC algorithm on the S disjointed sets. The disjointed sets will then be connected via the most inexpensive distance to other sets to ensure connectivity of the entire grid.

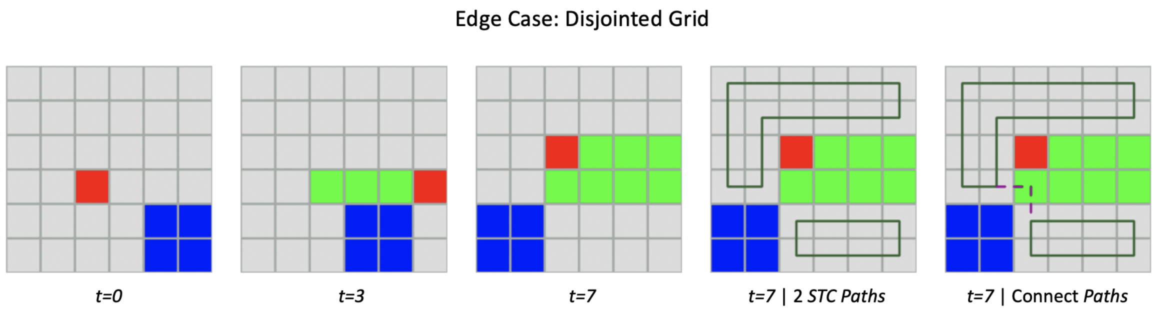


Figure 3.3: Examples of Connecting Two Disjointed Set

Algorithm 2: Constant Re-planning STC Algorithm

```
Input: A geometrical description of the Map, agent's position $a$, obstacles
  movements list $obs$
Output: A coverage paths $P$
Initialize: 2D-Grid $\leftarrow Map$;
$k \leftarrow 0$;
$N \leftarrow size(Map)$;
$(a_x, a_y) \leftarrow a$;
$Visited \leftarrow []$;
while $k < N$ do
    if $len(Map) > 1$ then
        $recursive$ ;      /* If map is not connected (have disjoint set) */
    else
        $G(V, E) \leftarrow obs[k]$ ;                         /* Construct MST */
        $action, step \leftarrow G(V, E)$ ;          /* Takes action from MST path */
        $k \leftarrow k + 1$;
        $visited \leftarrow visited + step$;
    end
    $P \leftarrow visited$;
end
```

3.2 Graph-Based: Longest Path with Time Available Function

Time Available Function: To help ease and simplify the difficulty of CPP with dynamically moving obstacles, we designed a **Time Available Function** which is capable of providing the future movement of the obstacles. As a result, the moving obstacles are no longer stochastic but rather deterministic for a certain time frame into the future. The Time Available Function aids in converting the information regarding a cloud into a value on each tile on the grid. Each value represents the remaining time (in seconds for example) during which the agent can perform coverage before obstacles arrive.

Using the **Time Available Function**, we have derived a **Find Longest Path** algorithm to detect the longest possible path that is valid under the Time Available Function. More specifically, at every time step i, a tile is valid as part of the longest path if the value on the tile is $\geq$ i.. In the event of multiple valid paths with the same lengths, the agent chooses the one with the least number of turns required for coverage.

The algorithm starts by generating and assigning values to tiles on the grid. The agent then follows the above-mentioned rules to find the longest possible path. The algorithm resets when the agent reaches the end of its determined/programmed path. At this point, the tiles that have been visited will be assigned with values of infinity so that the covered tiles remain valid pathways that can still be used to reach those with limited time available for coverage. The process continues until all tiles have been visited.

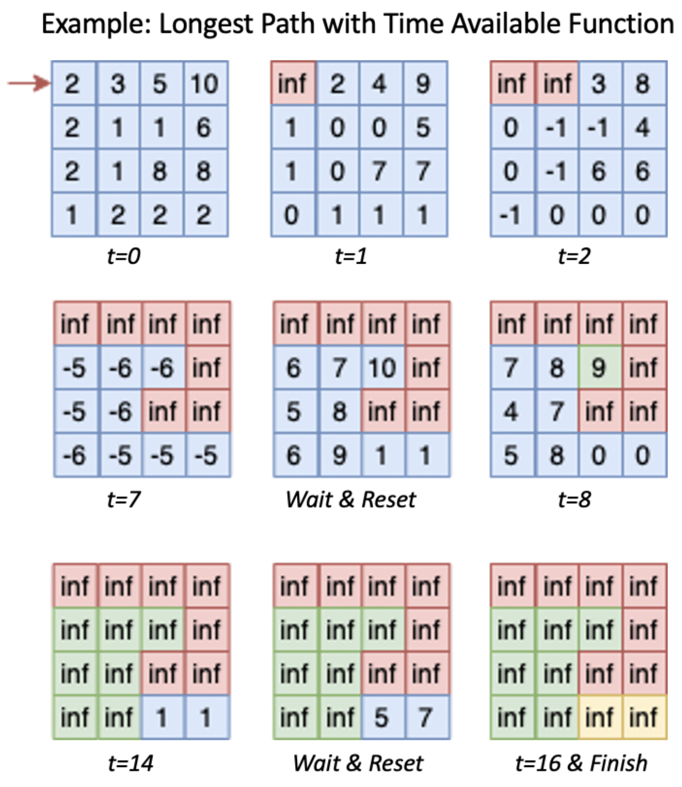


Figure 3.4: A complete example of Longest Path with Time Available Function

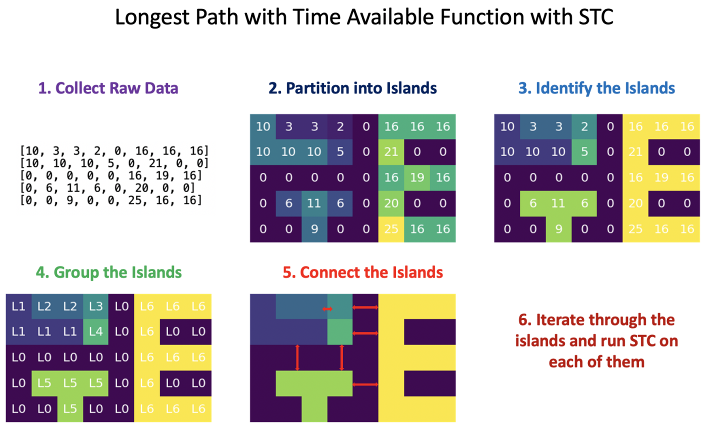
3.3 Graph-Based: Longest Path with Time Available Function with STC

While both of the above approaches are feasible, it is important to note their potential flaws. More specifically, re-planning with STC at every time step is infeasible with a larger dimensional grid, and **Find Longest Path** may lead to a single region being visited repeatedly. With those challenges in mind, we plan to combine these approaches to build a hybrid solution, taking the benefits from both algorithms. Initially, the resulting algorithm will iterate through all the tiles on the grid, breaking it into smaller islands by the following formula and condition (Equation 3.1). Once the grid is broken down, the agent will treat the islands as one singular tile/node and run the Find Longest Path algorithm previously mentioned. Having done so, the agent will iterate through the islands and run STC on each one.

Given a N x N grid, we denote $c_1, c_2, ...c_w$ as the considered clusters of cells in the N x N grid as one island if

$$min(c_1, c_2, ...) > floor(1.5 * w) \tag{3.1}$$

The intuition beside the design of the equation comes from the fact that if the agent happens to start in the middle cells of any given island, it has the ability to traverse to one end and still has time to traverse to the other end of the island.


Longest Path with Time Available Function with STC
1. Collect Raw Data
2. Partition into Islands
3. Identify the Islands
4. Group the Islands
5. Connect the Islands
6. Iterate through the islands and run STC on each of them

```
[10, 3, 3, 2, 0, 16, 16, 16]
[10, 10, 10, 5, 0, 21, 0, 0]
[0, 0, 0, 0, 0, 16, 19, 16]
[0, 6, 11, 6, 0, 20, 0, 0]
[0, 0, 9, 0, 0, 25, 16, 16]
```

Figure 3.5: A simple guide to Longest Path-STC

3.4 Learning-Based: RL Coverage Path Planning on Static Obstacle

To fully comprehend and appreciate the power and benefit reinforcement learning introduces in coverage path planning in an environment with dynamic obstacles, we first establish a baseline model whereby we trained an agent on static obstacles. Results in the static obstacle environment enabled us to better compare and analyze the different methods discussed later with dynamic moving obstacles. With the added benefits of the previously discussed policy gradient methods, we then investigated the performance of well-established DRL Proximal Policy Optimization (PPO) techniques.

In this section, we will present the preliminaries of the considered problem. A completed coverage path planning is characterized by the following attributes: Consider K static point of interest in a 2D grid where the agent is interested in conducting coverage. More specifically, we denote each variable with the following notations

- Movement Budget: B
- Total Area Visited: V
- Obstacles Positions (for K obstacles): $obs = [(obs_{1x}, obs_{1y}), (obs_{2x}, obs_{2y}), ...(obs_{Kx}, obs_{Ky})]$
- Agent Position: $agent = (a_x, a_y)$

An MDP is described by the tuple (S, A, R, P), with the set of possible states S, the set of possible actions A, the reward function R, and the deterministic or stochastic state transition function $P : S \times A \rightarrow S$.

Action Space: The action space A is defined as five discrete actions:

$$A = \{\text{up, down, left, right, stay}\}$$

Observation Space: In a NxN grid, the observation space S has the following dimensions:

$$S = \underbrace{\mathbb{R}^2}_{\text{position}} \times \underbrace{\mathbb{R}^{K\times 2}}_{\text{obstacles}} \times \underbrace{\mathbb{R}^{N\times N}}_{\text{Map}} \times \underbrace{\mathbb{R}^1}_{\text{Coverage}}$$

For obstacles positions and map positions, they are all computed in relative distance from the agent's current position.

Reward Function: The immediate reward function $r_i(s_i, a_i)$ consists of three components: the newly visited reward, previously visited reward, and obstacles penalty. At time i, we denote state as s_i and action as a_i The coverage reward is designed as

$$r_i(s_i, a_i) = \begin{cases} 10 \quad \text{newly visited} \\ -1 \quad \text{collision with obstacles or boundaries} \\ -0.1 * W \quad \text{previously visited (W represents \# of time visited)} \end{cases}$$

Following the formulation of PPO and the above specification, we used a 2-layers multilayer perceptron (MLP) with 64 units per layers with Tanh activation function.

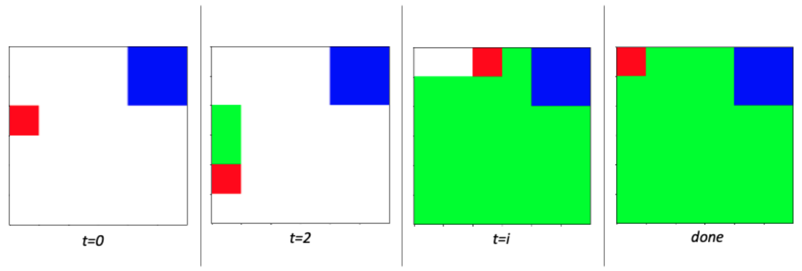


Figure 3.6: An example of coverage on static obstacle

3.5 Learning-Based: RL Coverage Path Planning on Dynamic Obstacle

Once we established a baseline static obstacle policy, we proceeded with dynamic moving obstacles, where they were able to move in any of the four directions uniformly. Additionally, the agent's speed was double that of obstacles' speed. The action space, observation space, and reward function remained the same.

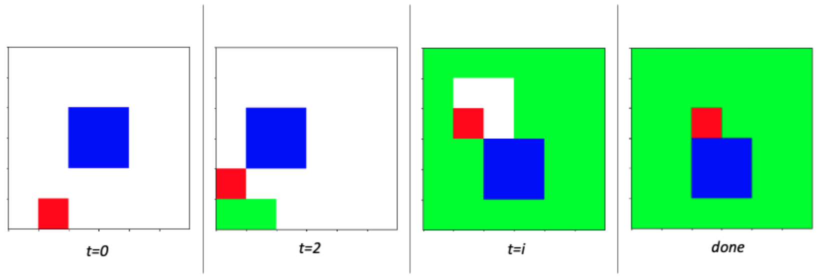


Figure 3.7: An example of coverage on dynamic obstacle

3.6 Learning-Based: RL Coverage Path Planning on Dynamic Obstacle with Memory

To determine the impact moving obstacles have on coverage path planning, we encoded one step of future movement of the dynamic obstacles within the observation space for the agent to better learn. The updated **observation space** in a grid has the following dimensions:

$$S = \underbrace{\mathbb{R}^2}_{\text{position}} \times \underbrace{\mathbb{R}^{K\times 2}}_{\text{current obstacles}} \times \underbrace{\mathbb{R}^2}_{\text{future obstacles}} \times \underbrace{\mathbb{R}^{N\times N}}_{\text{Map}} \times \underbrace{\mathbb{R}^1}_{\text{Coverage}}$$

With the added benefit of one-step future obstacle position information, we hypothesize that the agent can more efficiently conduct coverage path planning.

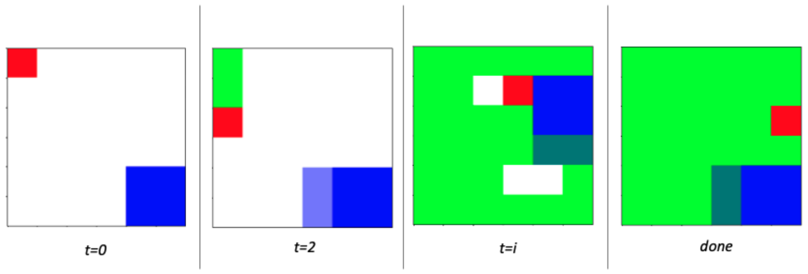


Figure 3.8: An example of coverage on dynamic obstacle with memory

3.7 Learning-Based: Guided RL Coverage Path Planning on Dynamic Obstacle

Although a learning-based approach can conduct coverage path planning in a dynamic environment, the training process was typically longer than that of a static environment as the difficulty of the objective increases. Hence, to help guide the agent, we decided to also incorporate the STC algorithm during the learning process. During training of the agent at each time step, the STC path is computed and fed into the observation space of the agent. More specifically, the observation space:

$$S = \underbrace{\mathbb{R}^2}_{\text{position}} \times \underbrace{\mathbb{R}^{K\times 2}}_{\text{current obstacles}} \times \underbrace{\mathbb{R}^{N\times N}}_{\text{Map}} \times \underbrace{\mathbb{R}^1}_{\text{Coverage}}$$

where the Map, which represents the tiles on the grid, is multiply with a weighted value from the coverage path. With such a concept, we anticipate that the agent can accelerate its learning process.

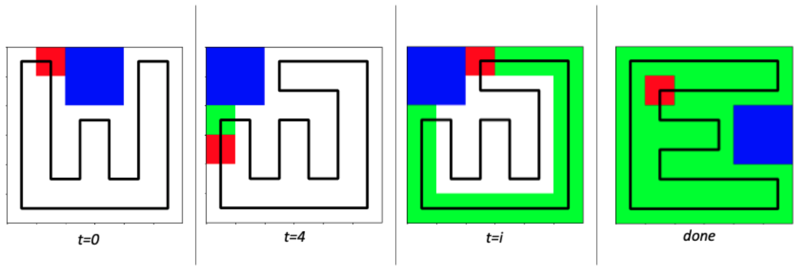


Figure 3.9: An example of coverage on dynamic obstacle with STC guided RL algorithm

Chapter 4

Result

4.1 Graph-Based Approaches

Constant Re-planning STC: Figure 4.1 shows time complexity using the constant STC re-planning algorithm for N x N grid size. Although the time complexity is reasonable, the number of revisited cells shown in Figure 4.2, which is a measurement of inefficiency, increases linearly with respect to the dimensions of the grid. As a result, such a method is not scalable for larger dimensional grids.

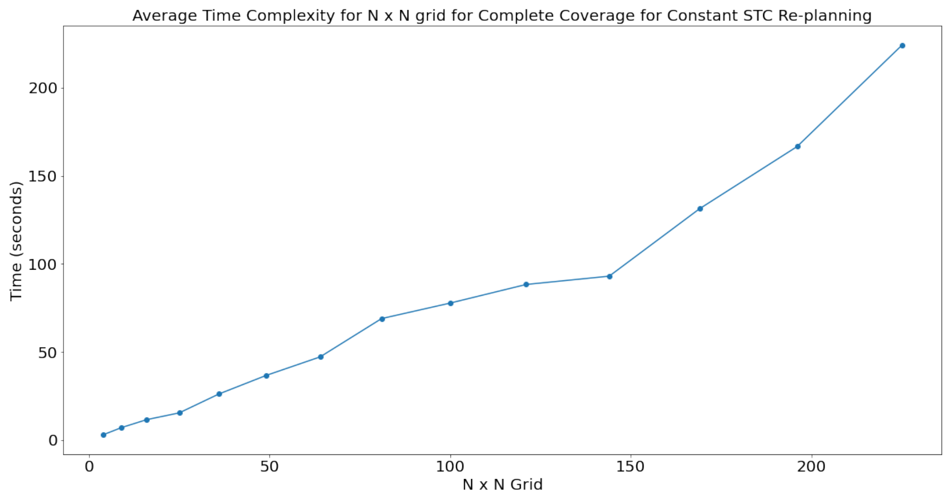


Figure 4.1: Average Time Complexity for N x N grid using Constant STC Re-planning

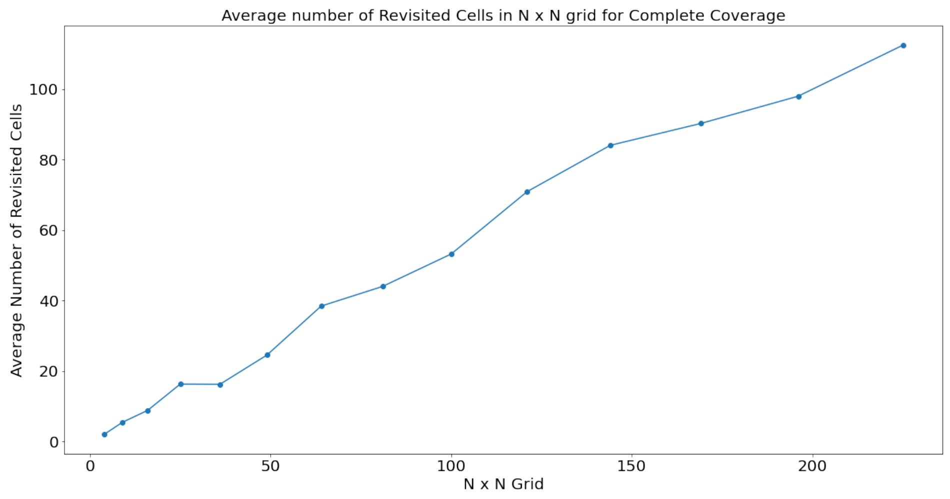


Figure 4.2: Average Number of Revisited Cells in N x N grid

To better explain the cause of the increase in revisited cells, see the example below. (Figure 4.3) As the obstacles and the agent move, all tiles visited are also treated as obstacles – breaking remaining non-visited cells into three disjointed sets. As a result, the agent needs to revisit some cells in order to conduct complete coverage.

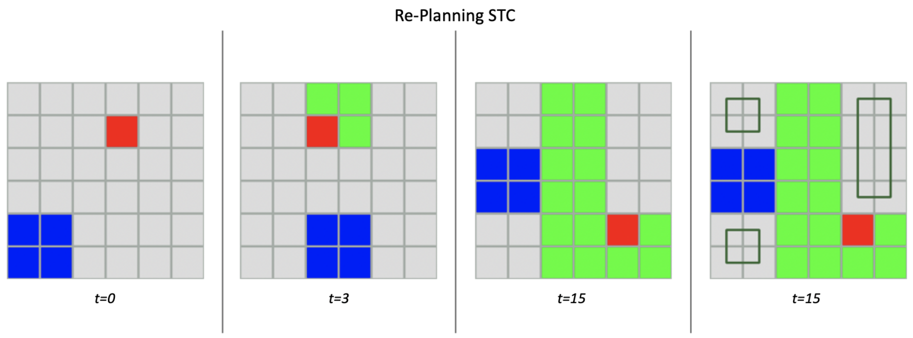


Figure 4.3: Example of STC Re-planning

Longest Path with Time Available Function: Figure 4.4 shows time complexity using the Longest Path with Time Available Function algorithm on the N x N grid. Since the algorithm recursively calls itself until all tiles have been visited, it is interesting to visualize the coverage rate after the first iteration. Figure 4.5 shows the coverage rate after the first iteration of the Longest Path algorithm. Additionally, the range of the Time Available Function is $\{1, 2, 3, ..., 10\}$ – the range of possible values on each tile.

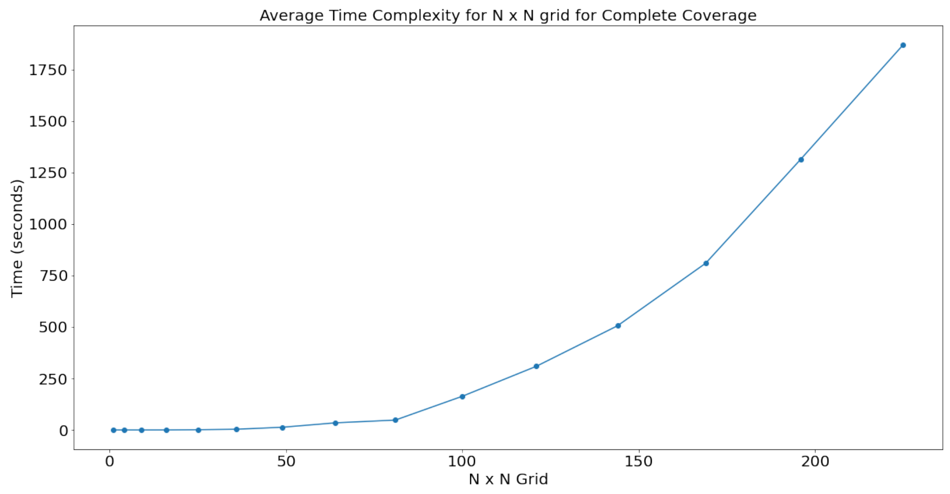


Figure 4.4: Average Time Complexity for N x N grid for Complete Coverage

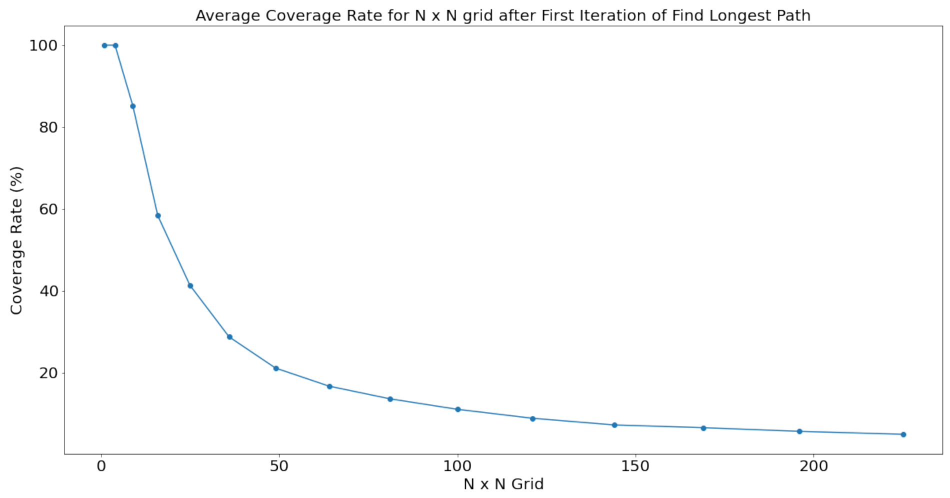


Figure 4.5: Average Coverage Rate for N x N grid after First Iteration of Find Longest Path

Longest Path with Time Available Function with STC: Figure 4.6 shows number of generated islands using the Longest Path with Time Available Function with STC algorithm on the N x N grid.

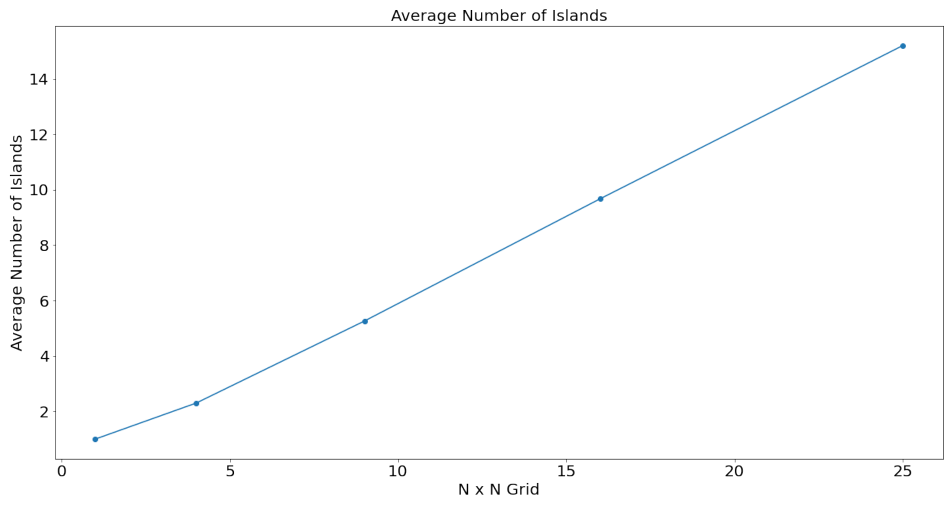


Figure 4.6: Average Number of Islands for N x N grid

4.2 Learning-Based Approaches

In addition to the 3 graph-based approaches, Table 4.1 present the results (total steps to complete coverage for a 6x6 grid) for the for the 4 different learning-based trained models followed by their RL training metrics.

Env: 6x6 Grid	Static Obstacle	Dynamic Obstacle	Dynamic Obstacle & Memory	Dynamic Obstacle & STC
Mean Length	36.934	65.147	65.760	70.548
Min Length	32	37	38	37
Max Length	47	205	237	308
Standard Deviation Length	2.890	21.314	22.340	26.823

Table 4.1: Results using RL-Based Approaches (in Steps to Complete Coverage)

Training RL Metrics

Mean Episode Length: Average number of steps the agent takes to complete one episode.

Mean Reward Length: Average discounted cumulative reward the agent receives per episode.

Approximate Mean KL Divergence: Approximate mean KL divergence between old and new policy (for PPO), it is an estimation of how much changes happened in the update.

Clip Fraction: Mean fraction of surrogate loss that was clipped (above clip range threshold) for PPO.

Mean Entropy Loss: Mean value of the entropy loss (negative of the average policy entropy)

Explained Variance: Fraction of the return variance explained by the value function.

- Explained Variance $= 0$, random prediction
- Explained Variance $= 1$, perfect prediction
- Explained Variance ≤ 0, worse than random prediction

Training Loss: Current total loss value according to cross entropy loss.

Value Loss: Current value for the value function loss for on-policy algorithms, usually error between value function output and Monte-Carlo estimate

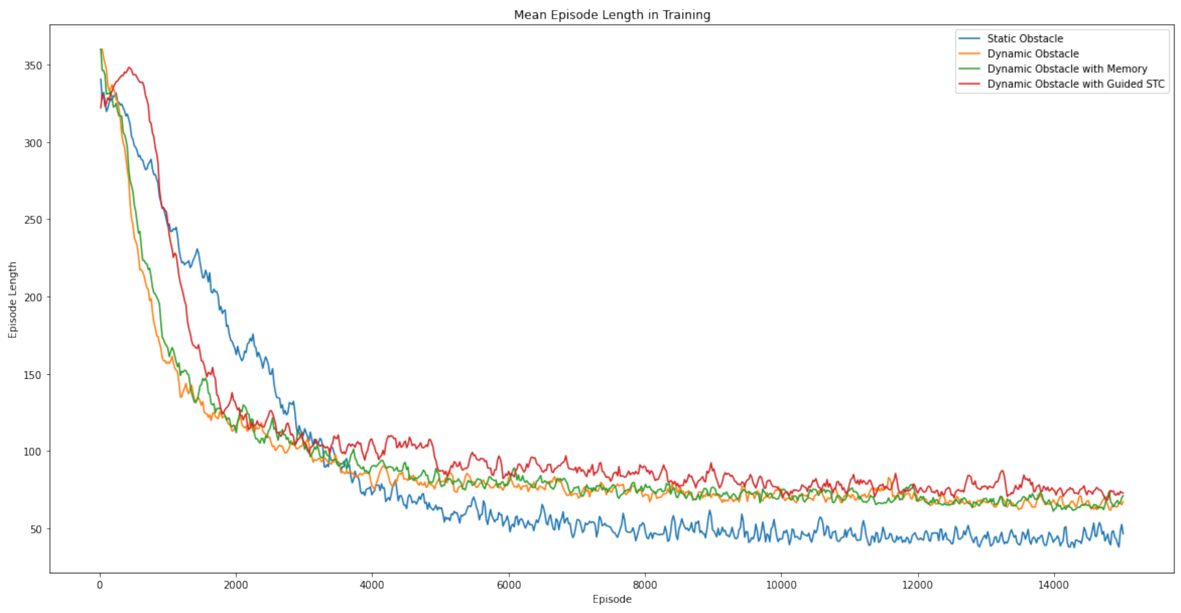


Figure 4.7: Mean episode length

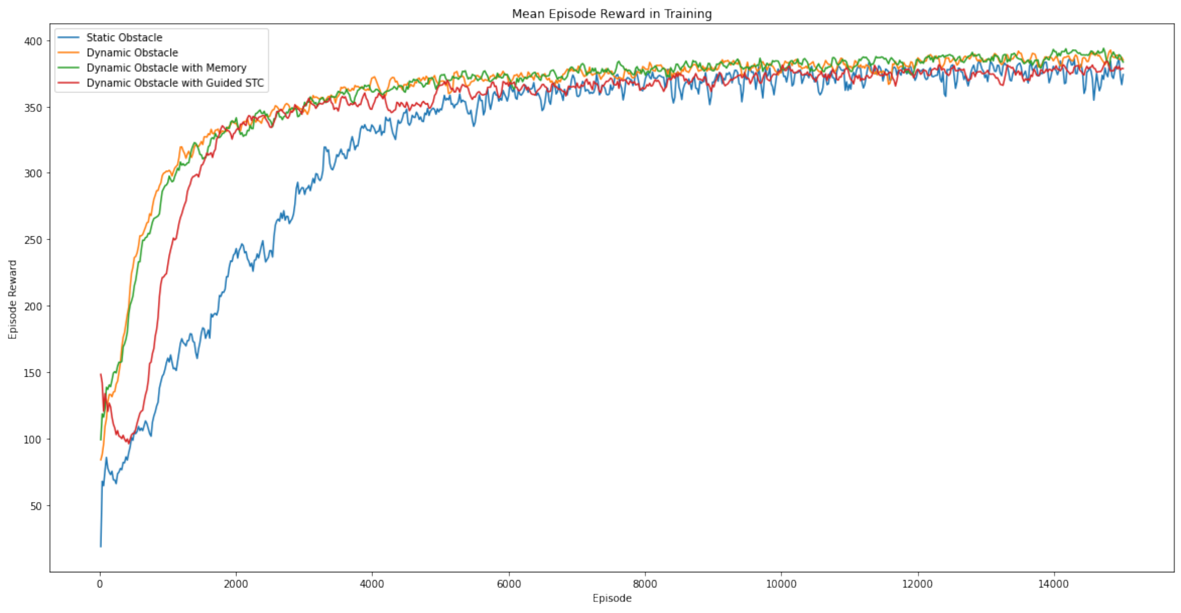


Figure 4.8: Mean Reward length

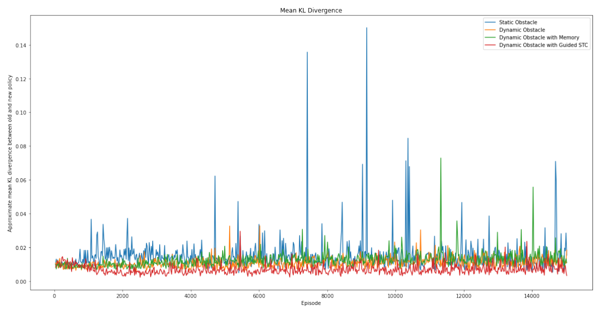


Figure 4.9: Approximate Mean KL Divergence

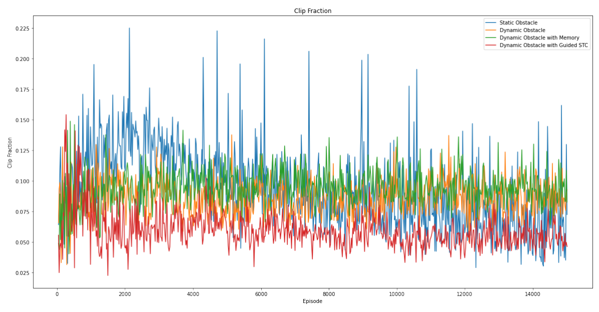


Figure 4.10: Clip Fraction

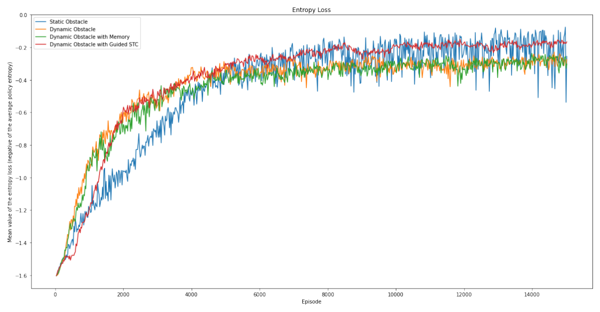


Figure 4.11: Mean Entropy Loss

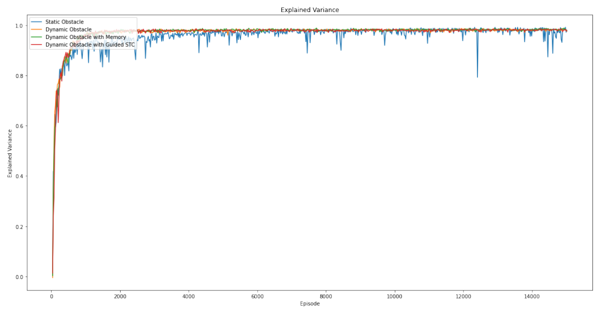


Figure 4.12: Explained Variance

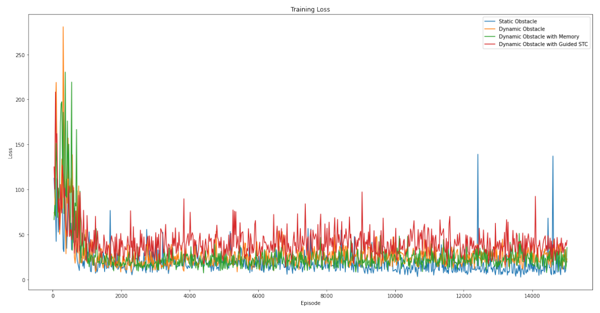


Figure 4.13: Training Loss

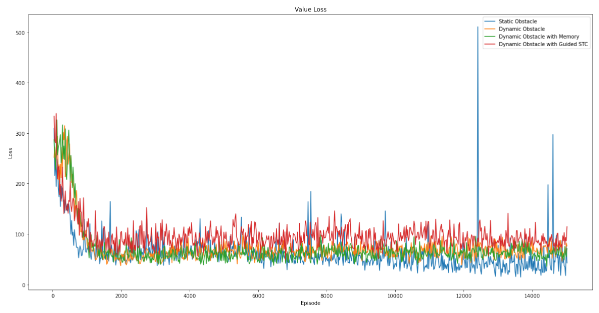


Figure 4.14: Value Loss

Chapter 5

Analysis

The results exhibit significant applicability of the proposed algorithm to coverage path planning tasks.

As evidenced in the first and simplest graph-based approach, Constant Re-planning STC from Section 3.1, although the computation is $O(N^2)$, where N represents the dimensions of the grid, agents are required to recompute the trajectory at every time step. Even when the agent does take information regarding the possible future positioning of the obstacles into consideration, the number of revisited states is still significant as shown in Figure 4.2. And with the current heuristic design, the agent will also have to revisit multiple previously visited states. Such a method, therefore, can guarantee complete coverage but is inefficient.

In the second graph-based approach, Longest Path with Time Available Function from Section 3.2, we applied the Longest Path algorithm conditioned on a valid tile as defined by the time available function and presented our result in Figure 4.4 and Figure 4.5. Although this method allows the agent to account for future movement of obstacles, it fails to deliver the most efficient coverage path. We designed the algorithm using depth-first search (DFS) with memoization, where each function call of the Find Longest Path algorithm results in the time and memory complexities of $O(N^2)$. Since the agent is constantly replanning until the grid has been fully visited, the worst case could result in $O(N^4)$ for both time and memory complexities. It is worth noting that this algorithm can generate a more efficient coverage path as compared to the first method. Unfortunately, it also experiences high computation in time and memory complexity.

The final graph-based approach, Longest Path with Time Available Function with STC from Section 3.3, aimed to combine the previous two methods by (1) breaking the grid into valid islands, (2) finding the longest path on the islands as a singular node, and (3) visiting the islands according to the result of Find Longest Path and running STC on each one before proceeding to the next. The intentions of this method were to (1) increase efficiency and remove any heuristic design compared to the first method of constant replanning, and (2) reduce the average time and space complexities as the Find Longest Path algorithm is used on clusters of tiles defined as islands. Results shown in Figure 4.6, however, indicate that this method may populate too many islands, resulting in essentially the same scenario as method two, above.

Regarding learning-based methods, and using the specific formulation of the coverage problem in PPO discussed in Section 3.4, the agent's performance in static obstacle circumstances was as expected: it visited every tile exactly once in the minimum case and on average requiring 1.1x time steps to visit all tiles. However, the agent still had a standard deviation of 2.8 and the maximum number of steps was 5x more than the coverage requirement. All learning-based methods results are presented in Table 4.1.

In the naive dynamic obstacle setting presented in Section 3.5, the agent was still capable of conducting coverage with an average of 65 steps and a standard deviation of 21 steps. In this setup, we certainly expect the agent to require more time steps to conduct complete coverage as compared to the steps required in static-obstacle circumstances, due to unknown obstacle movement and direction. The high standard deviation also illustrates the agent's instability due to the uncertainty of obstacle movement. To compensate, we designed the agent to move at 2x the speed of the obstacle.

Since the dynamic obstacles were completely stochastic, it is difficult for the agent to plan the optimal path. Hence, we introduced a one-step memory of the obstacle in the agent's observation space hoping to achieve better path planning. Such method was presented in Section 3.6. However, according to the results, the agent is still only capable of achieving similar success to that of the naive setup. We suspect this will require tuning the reward function as the current penalty for obstacles is not high enough.

As discovered in previous methods, the agent takes time to learn the objective and plan for coverage, but fails to find the most efficient path from time to time. Thus, we aim to incorporate the graph-based STC algorithm in the RL training process in hopes that the agent is able to learn faster and perform better (achieving the coverage in fewer time steps compared to the naive approach). The method, Guided RL Coverage Path Planning on Dynamic Obstacle, was introduced in Section 3.7. However, results show that the agent suffers from noise due to the constant replanning path generated from the STC. Another way to visualize the issue is that the STC path generated at every time step differs dramatically making it difficult – for even the human brain – to determine optimal action.

Chapter 6

Application

Recent robots are equipped with auto-navigation systems that incorporate path planning techniques. These robots, through the use of sensors and visions (LiDARs, images), are capable of navigating unknown environments. More specific practical applications involve conducting coverage path planning via UAVs: coverage on farmland with both static and dynamic obstacles, for example (such as shadows generated by clouds). Further, with the rise of aerial robots in precision agriculture, aerial vehicles can be used to better support management of farm inputs by detecting, locating, and identifying nutrient deficiencies, weed densities, disease, pest infestations, and crop water status. Such detection enables early intervention to protect crop yields. UAVs will also enable the application of inputs including fertilizer levels, water management and many more. As a result, growers' yields are expected to increase, while they also reduce fuel expenditure, time, cost, quantity of inputs, and environmental impact.

Agricultural robotic systems, however, frequently face a diverse set of competing objectives, and the greatest obstacle to their wide and effective deployment is the present lack of control designs that guarantee the accomplishment of multiple goals by multiple collaborating robotic vehicles. Currently, research labs from the University of California, Berkeley, Stanford University, and the University of Illinois Urbana-Champaign have come together in an effort to collect shadow-free imagery of farm fields by multi-vehicle robotic systems. Such applications can be formulated as coverage path planning for farmland where cloud shadows are considered as dynamic obstacles. Each of the multiple vehicle objectives will represent a specific task in precision agriculture surrounding the collection of images and crop data where such collection can only be accomplished during five hours of daylight time. This is in order to avoid cloud shadows and ensure the safety of the overall system.

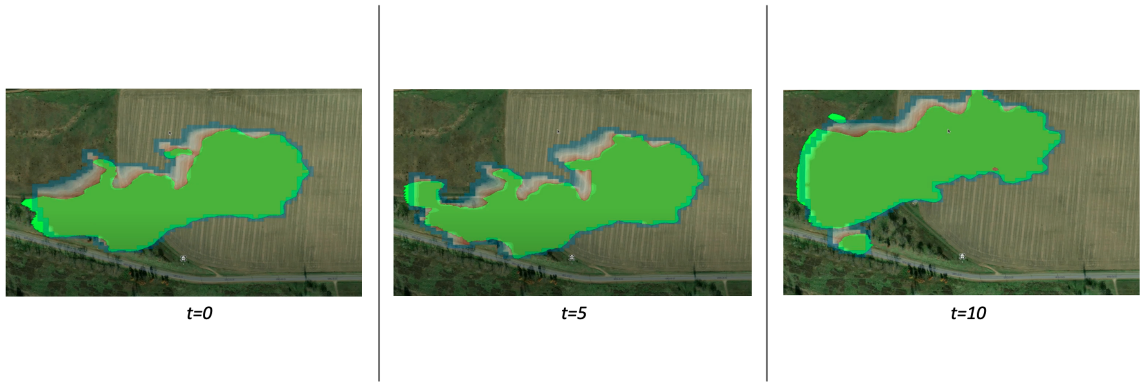


Figure 6.1: Illustration Cloud Movement and Prediction on Farmland

Chapter 7

Conclusion

While there has been extensive research on the topic of path planning in both traditional deterministic graph-based algorithms, there still lack efficiency in both the path planning task and computational time and memory of these algorithms. Furthermore, coverage path planning (CPP) with dynamic obstacles has not been well explored by the community. Hence, with advancement in reinforcement learning, researchers have been trying to use more learning-based approaches (such as neural networks) to help conduct path planning, more specifically coverage path planning. However, it is worth noting that there still exist many unknowns in neural networks and how/why they work from time to time.

Although results from static obstacles using a learning-based approach showcased equalivantly effective results (achieving exacting visiting each tile exactly once), the same cannot be said for dynamic obstacles. Additionally, it is also worth exploring other baseline models to compare dynamic obstacles results. For static obstacles, one can use the deterministic graph-based result (which is the most efficient) compared to that of learning-based methods. For dynamic obstacles, while the learning-based approach removes issues such as memory constraint and runtime complexity, it still has its own challenges. For example, it's difficult to provide a good baseline model to compare dynamic obstacles to. In conclusion, more in-depth investigation can be conducted using learning-based approaches for CPP.

Chapter 8

Future Work

Following this paper, future research may focus on refining and tuning the details of the reinforcement learning model.

Curriculum Learning To further encourage the reinforcement learning agent, aside from guided reinforcement learning through STC, one can also train the model through curriculum learning, an idea first introduced by Bengio et al. [2]. Curriculum learning trains the machine learning model with easier data (or easier subtasks) and gradually increases the difficulty level of the data (harder subtasks) until the entire training dataset (target task) is reached. For CPP within dynamic obstacles, subtasks can be broken down into several "levels": environments with no obstacles, with static obstacles, with deterministic moving obstacles, and with stochastic moving obstacles as illustrated in Figure 8.1.

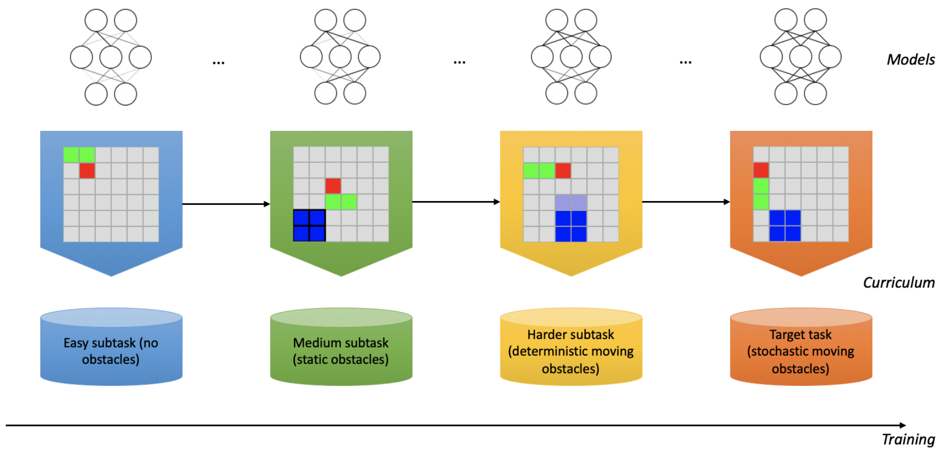

Models
Curriculum
Easy subtask (no obstacles)
Medium subtask (static obstacles)
Harder subtask (deterministic moving obstacles)
Target task (stochastic moving obstacles)
Training

Figure 8.1: Illustration of the Curriculum Learning for Coverage Path Planning

Curse of Dimensionality Another major consideration given the current RL model is whether the input (observation) space for the neural network has high dimensions. Depending on architecture and activation functions [1] such high dimensions may suffer from the "curse of dimensionality". Existing methodology to help circumvent the curse of dimensionality involves using different kernel functions, such as a convolutional network where the last layer is a softmax layer. Here, the architecture conducts a non-linear projection onto lower dimensions which, in turn, provide a compressed representation of data. Though basic multilayer perceptron (MLP) layers do not have such capacity, we can introduce and train an autoencoder to first compress the observation space into lower dimensions and then feed into the neural network within the reinforcement learning algorithm. Once the model has been trained in lower dimensions, we can apply a decoder to reverse into higher ones for interpretation. We hypothesize that such a method can accelerate the learning process and improve the accuracy of the model with lower dimensional input. Figure 8.2 illustrates the autoencoder architecture for dimensionality reduction.

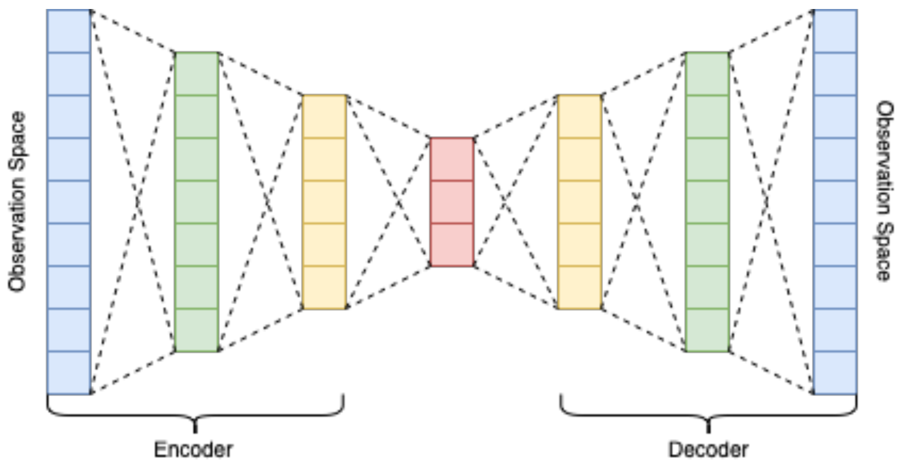


Figure 8.2: Illustration of an Autoencoder

Bibliography

[1] Francis R. Bach. “Breaking the Curse of Dimensionality with Convex Neural Networks”. In: *CoRR* abs/1412.8690 (2014). arXiv: 1412.8690. URL: http://arxiv.org/abs/1412.8690.

[2] Yoshua Bengio et al. “Curriculum Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 41–48. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: https://doi.org/10.1145/1553374.1553380.

[3] Zengyu Cai et al. “Research on Complete Coverage Path Planning Algorithms based on A*Algorithms”. In: *The Open Cybernetics & Systemics Journal* 8 (2014).

[4] Marija Dakulović, Sanja Horvatić, and Ivan Petrović. “Complete Coverage D* Algorithm for Path Planning of a Floor-Cleaning Mobile Robot”. In: *IFAC Proceedings Volumes* 44.1 (2011). 18th IFAC World Congress, pp. 5950–5955. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20110828-6-IT-1002.03400. URL: https://www.sciencedirect.com/science/article/pii/S147466701644557X.

[5] Anthony Davis et al. *Path Planning Algorithms for Robotic Aquaculture Monitoring*. 2022. DOI: 10.48550/ARXIV.2204.09753. URL: https://arxiv.org/abs/2204.09753.

[6] Y. Gabriely and E. Rimon. “Spanning-tree based coverage of continuous areas by a mobile robot”. In: 2 (2001), 1927–1933 vol.2. DOI: 10.1109/ROBOT.2001.932890.

[7] Javad Heydari, Olimpiya Saha, and Viswanath Ganapathy. “Reinforcement Learning-Based Coverage Path Planning with Implicit Cellular Decomposition”. In: *CoRR* abs/2110.09018 (2021). arXiv: 2110.09018. URL: https://arxiv.org/abs/2110.09018.

[8] Marzieh Kooshkbaghi and Farzaneh Abdollahi. “Coverage control considering unknown moving obstacles avoidance”. In: *2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*. 2014, pp. 089–094. DOI: 10.1109/ICRoM.2014.6990882.

[9] Sung. G. Lee and Magnus Egerstedt. "Controlled Coverage Using Time-Varying Density Functions*". In: *IFAC Proceedings Volumes* 46.27 (2013). 4th IFAC Workshop on Distributed Estimation and Control in Networked Systems (2013), pp. 220–226. ISSN: 1474-6670. DOI: `https://doi.org/10.3182/20130925-2-DE-4044.00030`. URL: `https://www.sciencedirect.com/science/article/pii/S1474667015402319`.

[10] Jessica Leu, Michael Wang, and Masayoshi Tomizuka. *Long-Horizon Motion Planning via Sampling and Segmented Trajectory Optimization*. 2022. DOI: `10.48550/ARXIV.2204.07939`. URL: `https://arxiv.org/abs/2204.07939`.

[11] Jessica Leu et al. *Autonomous Vehicle Parking in Dynamic Environments: An Integrated System with Prediction and Motion Planning*. 2022. DOI: `10.48550/ARXIV.2204.10383`. URL: `https://arxiv.org/abs/2204.10383`.

[12] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: `1312.5602`. URL: `http://arxiv.org/abs/1312.5602`.

[13] S.PlatnickandT.Arnold. *Cloudy earth*. URL: `https://earthobservatory.nasa.gov/images/85843/cloudy-`.

[14] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`. URL: `http://arxiv.org/abs/1707.06347`.

[15] Mirco Theile et al. "UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning". In: *CoRR* abs/2003.02609 (2020). arXiv: `2003.02609`. URL: `https://arxiv.org/abs/2003.02609`.

[16] Christos K. Verginis, Yiannis Kantaros, and Dimos V. Dimarogonas. *Planning and Control of Multi-Robot-Object Systems under Temporal Logic Tasks and Uncertain Dynamics*. 2022. DOI: `10.48550/ARXIV.2204.11783`. URL: `https://arxiv.org/abs/2204.11783`.