ABSTRACT

MECHANICAL IMPLEMENTATION OF REINFORCEMENT LEARNING ALGORITHMS

Nathan Stallé, M.S.
Department of Mechanical Engineering
Northern Illinois University, 2018
Brianno D. Coller, Director

Reinforcement learning is a type of machine learning technique that can be used to solve classical control problems. One key difference between reinforcement learning and classical methods like PD or PID control is that reinforcement learning does not necessarily need a model of the system that is being controlled. Reinforcement learning uses a type of trial and error approach in which a reward function is implemented. The reinforcement learning algorithm focuses on maximizing the reward through a series of actions. In recent years there has been a renewed interest in reinforcement learning as it is a major component in new and even more sophisticated techniques such as “Deep Learning”.

This thesis aims to implement reinforcement learning in a mechanical system where the goal is to control the angular position and angular velocity of an arm that can rotate freely in a horizontal plane. Forces are applied to the arm via propeller and a pair of rudders controlled by a servo motor. The reward system makes use of an Inertial Measurement Unit to determine the orientation and rotation rate of the arm. A BeagleBone Blue single board computer is used to implement the reinforcement learning algorithm, store and process information about the system’s states, and receive information from the onboard sensors.

While the basic reinforcement learning algorithm has been around for a long time, implementing it into a physical problem is always a unique challenge. Throughout this thesis the process of transitioning between exploratory and exploitive phases of the machine learning process as well as the discretization of the system's state space will be examined. Finally, unique control objectives will be achieved through the development of reinforcement learning control policies.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2018

MECHANICAL IMPLEMENTATION OF REINFORCEMENT LEARNING ALGORITHMS

BY

NATHAN STALLÉ

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

Thesis Director:
Brianno D. Coller

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

CHAPTER 1
INTRODUCTION

The aim of this thesis is to show the implementation of reinforcement learning algorithms of the Q-Learning style in a physical mechanical system. A simulation of the experimental apparatus is constructed, and a visualization of the system's discretized state space is examined. Simulated and physical behavior of the system is explored and a comparison between the two is made. Finally, ideas for future improvements to the system are briefly discussed.

1.1 Motivation

Reinforcement learning can be used as a type of machine learning to solve classical control problems. One specific style of reinforcement learning is called Q-Learning, which is a type of temporal-difference learning. Temporal-difference learning techniques, including Q-Learning, can learn directly from experience without a model of the system's dynamics [1]. With an increase in excitement in machine learning and artificial intelligence over recent years it would be a worthwhile challenge to implement Q-Learning techniques into a custom physical system to experience the effects of reinforcement learning first hand.

1.2 Material Review

Many central ideas that were used in the creation of this thesis came from Richard S. Sutton and Andrew G. Barto's book *Reinforcement Learning: An Introduction* [2]. Dr. Dan Klein released a series of videos on the UC Berkeley webpage explaining theory and conceptual ideas behind reinforcement learning and Markov Decision Processes [3] that were used throughout this thesis. The Robotics Cape Library is a piece of open source software that was developed by Strawson Design [4] and was used when integrating the Q-Learning algorithms to the experimental apparatus. Finally, a piece of Q-Learning software created by Northern Illinois Universities' Brianno D. Coller was used as a base skeleton for the experiment and simulation.

CHAPTER 2
REINFORCEMENT LEARNING

This chapter discusses the details of Reinforcement learning, specifically the Q-Learning style. Reinforcement learning is an area of machine learning which processes information from a semi-random exploration process and "learns" how to achieve a pre-defined reward [5]. Q-Learning is a specific style of reinforcement learning that is built around a Markov Decision Process.

2.1 Markov Decision Process

The Markov Decision Process (MDP) defines a finite set of states *S*, a finite set of actions A_s that can be performed at each set of states, the probability $P_a(s, s')$ that a specific action a at state *s* will lead to state s' after a single time-step, a reward $R_a(s, s')$ for transitioning from state *s* to state s' after performing a specific action *a*, and a discount factor γ which represents the weight between future rewards relative to the current received reward [6]. Markov Decision Process techniques can be used to discover a "policy" that specifies which action to take at each set of states to maximize a cumulative reward function [7] per Equation 2.1 for which a specific control objective is achieved.

$$R = \sum_{t=0}^{t_{end}} \gamma^t R_{a,t}(s_t, s_{t+1}) \quad (2.1)$$

After performing an action *a* in state *s* at time *t* the system will experience state s_{t+1} one time-step later and will receive a reward value $R_{a,t}(s_t, s_{t+1})$. The cumulative reward function *R* is the discounted sum of all rewards received for all actions taken throughout an episode ranging from $t = 0$ to $t = t_{end}$.

2.2 Q-Learning

Q-Learning is a form of reinforcement learning that was developed by Christopher Watkins in 1989 [8] which makes use of an MDP. The simplest form of Q-Learning assigns a Q-Value to each possible action which is defined by Equation 2.2 [9].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma\, maxQ(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.2)$$

Where $Q(s_t, a_t)$ is the action-value function, or Q-Value, for action a_t in the state cell s_t, α is the algorithm's learning rate, r_{t+1} is the reward value in a state cell after an action has occurred, γ is the discount factor described by the MDP, and $maxQ(s_{t+1}, a)$ is the maximum Q-Value out of all possible actions for state s_{t+1} after an action has occurred. $Q(s_t, a_t)$ directly approximates $Q^*(s_t, a_t)$, which is the optimal action-value function [9] in accordance with Bellman's Principle of Optimality.

Every action *a* described by the MDP has a Q-Value associated with it. An exploration phase of the state space updates each action's Q-Values per Equation 2.2. By performing a specific

action in each set of states with a maximum Q-Value associated with it an MDP based policy can be defined.

The Q-Learning algorithm makes use of a reward system to update Q-Values for all possible actions while exploring a finite discrete state space. When deciding which action to take the algorithm implements an Epsilon Greedy strategy [10]. The Q-Learning algorithm goes through an exploration phase in which it performs actions to fill in Q-Values across the entirety of the state space. Depending on the overall size of the discretized state space the time to complete the exploration process may vary. After the state space has been sufficiently explored the Q-Learning algorithm can generate a policy based off the previously learned Q-Values.

2.3 Rewards

The reward system is necessary for the Q-Learning process to function properly per Equation 2.2. When implementing a reward system, the algorithm examines the state space before and after an action has been taken. Depending on the change in states before and after the action has occurred the algorithm may receive a numerical reward. This numerical reward can have a positive value, a negative value or a value equal to zero. The Q-Learning algorithm focuses on receiving a maximum cumulative reward per Equation 2.1 by performing a specific set of actions based off previously learned Q-Values and rewards.

2.3.1 Positive and Negative Rewards

Reward values are associated with the transition from one set of states at a specific time-step to another set of states at the next time-step and can have numerical values that are positive, negative or zero. When deciding what type of policy should be generated it is very important to pay close attention to the reward values assigned to certain types of state changes. The reward system holds significant power over what type of policy will be generated, therefore assigning positive reward values to favorable state changes may hold benefits. Assigning negative reward values to unfavorable state changes will encourage the algorithm to perform actions that avoid these types of state changes. Ultimately, the reward system should be specific to the problem which must be solved, as the reward system will help generate a unique policy.

2.4 State Space

To generate a sufficient policy the state space should be explored sufficiently so that the Q-Learning algorithm's states can converge towards a solution in accordance with a pre-defined control objective. Each dimension of the state space is a unique and measurable feature of the system being observed. Any states that may be considered continuous with no finite upper or lower bounds should be converted to a discrete form due to MDP's being a discrete time stochastic control process [11]. The total size of the discretized state space can directly affect the amount of time it takes for the algorithm to converge towards a solution as well as the resolution of the state

space. Each set of states can be thought of as a state cell, and multiple different actions can be taken at every one of these state cells.

2.4.1 Exploration

When beginning a training session for a new policy, all Q-Values for every action over the state space are set to zero. As the system performs actions and consequentially experiences new states the Q-Values associated with those actions are modified per Equation 2.2. Visual representation of the early exploration process can be seen in Figure 2.1 and Figure 2.2.

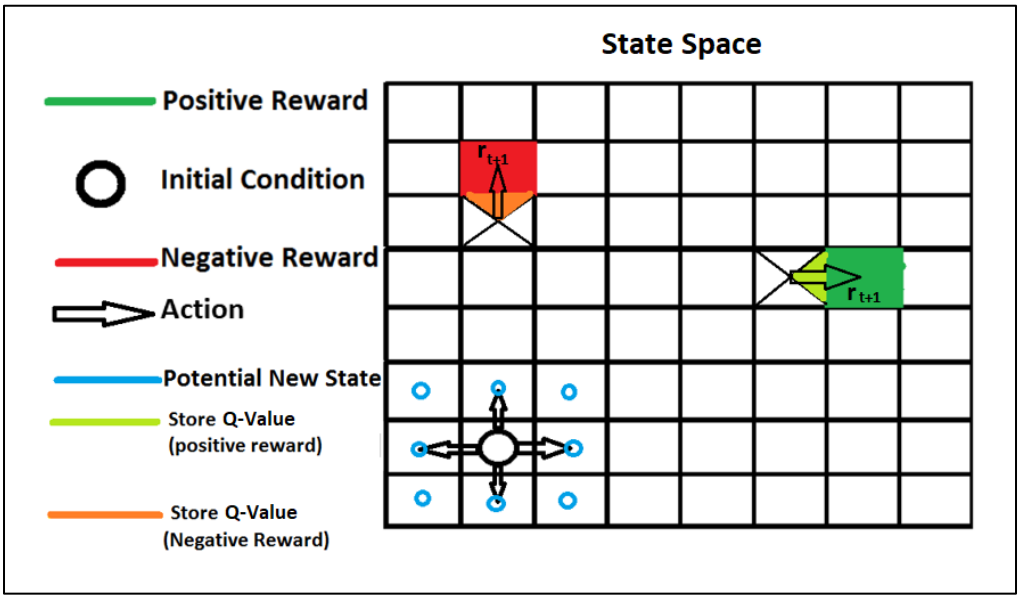

Figure 2.1 Graphical display of the early exploration process over the full state space

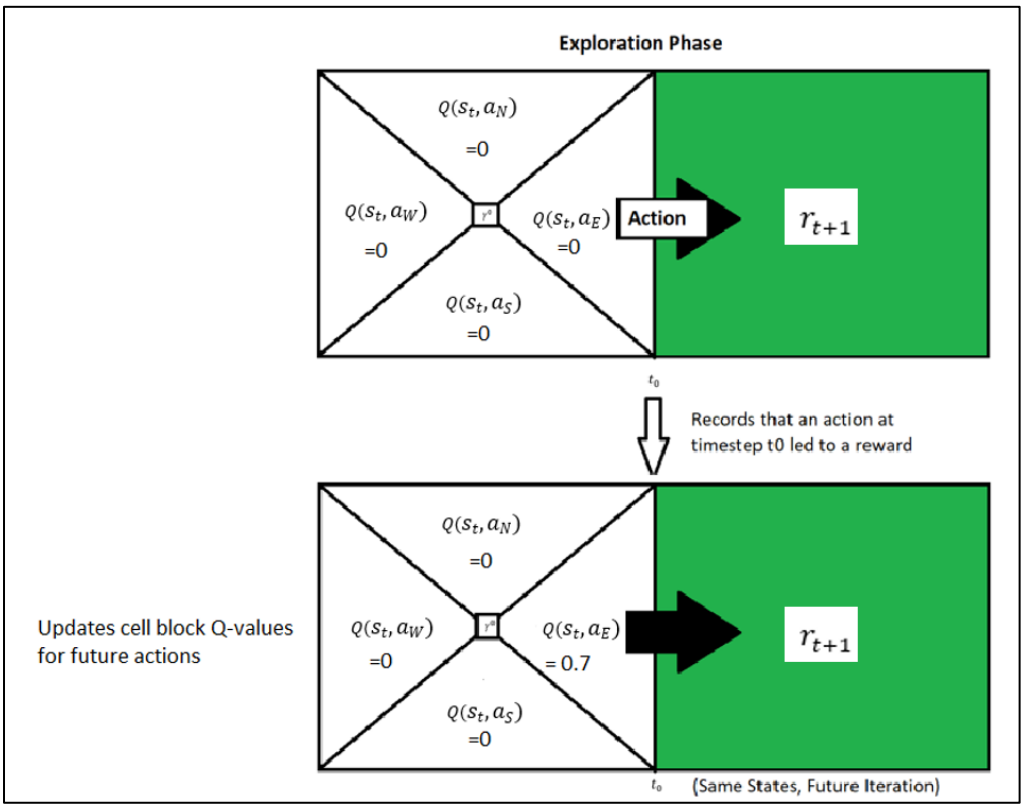

Figure 2.2 Graphical display showing exploration of a previously unexplored state cell

2.5 Possible Actions

The Q-Learning algorithm has the choice to perform one of multiple different actions at each state cell in accordance with Equation 2.2. When deciding which action to take the Q-Learning algorithm can make use of the Epsilon Greedy sub algorithm. If the Q-Learning algorithm decides to perform a "greedy" action according to the Epsilon Greedy decision process it will perform the action with the maximum Q-Value associated with it. These Q-Values are modified through the exploration process in accordance with Equation 2.2.

2.5.1 Epsilon Greedy

The Epsilon Greedy decision process can be used to help the Q-Learning algorithm decide whether to perform a randomly selected action or an action with a maximum Q-Value associated with it. The Epsilon Greedy algorithm creates a variable called Epsilon and assigned it a value between zero and 1.0 in accordance with Equation 2.3

$$0.0 \leq \varepsilon \leq 1.0 \qquad (2.3)$$

A random number is then generated which also ranges between zero and 1.0. If the randomly generated number is less than epsilon, then the Q-Learning algorithm will perform a randomly chosen action. If the randomly generated number is greater than epsilon, then the Q-Learning algorithm will perform the action with the maximum Q-Value associated with the system's current set of states.

2.6 Define a Policy

After the state space has been sufficiently explored and the system's states have converged towards the desired control objective the Q-Learning's Q-Values can be saved as a policy. The saved set of Q-Values can be reuploaded to the system after learning has finished and the system will perform the uploaded policy. Different policies can be learned by performing modifications to the reward system as discussed in Chapter 2.3.

CHAPTER 3
EXPERIMENTAL APPARATUS AND SOFTWARE DEVELOPMENT

In this chapter both hardware and software aspects of the experimental apparatus will be examined. A system consisting of a 3-Dimensional state space capable of performing three distinct actions was chosen for the implementation of reinforcement learning algorithms.

3.1 The Blotator

A custom machine which was designed and built by Dr. Brianno D. Coller can be seen in Figure 3.1. This machine was given the name "The Blotator" and will be referred to as such throughout the remainder of this thesis. The Blotator is essentially a rigid arm that is capable of rotating in a horizontal plane. A propeller which is attached to a small brushless AC motor can produce a constant outward force. Two rudders are attached to a micro servo and can be used to redirect the air flow which is generated by the propeller creating a moment about the central axis of rotation. This causes either a clockwise or counterclockwise angular acceleration.

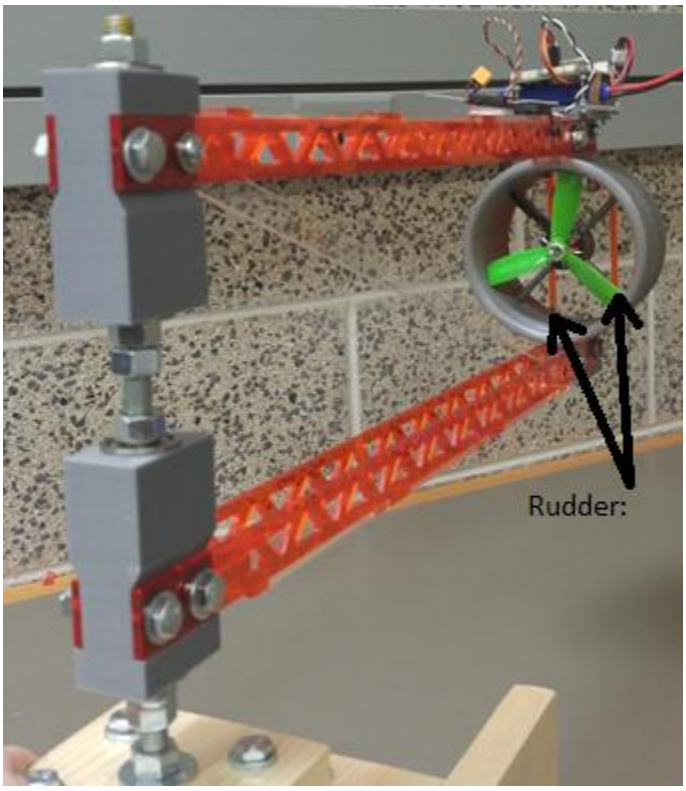

Figure 3.1 The Blotator system

3.1.1 Blotator State Space

The dynamics of the Blotator reside on a 3-Dimensional, continuous state space. Two of the states are the angle of the Blotator arm, θ, and the rotation rate, $\dot{\theta}$. The third state, ϕ, is the angle of the rudder which is altered through the output of the servo dynamics. When $\phi = 0$, the air flow from the propeller is directed outward, causing no net moment on the Blotator. The states are depicted visually in Figure 3.2.

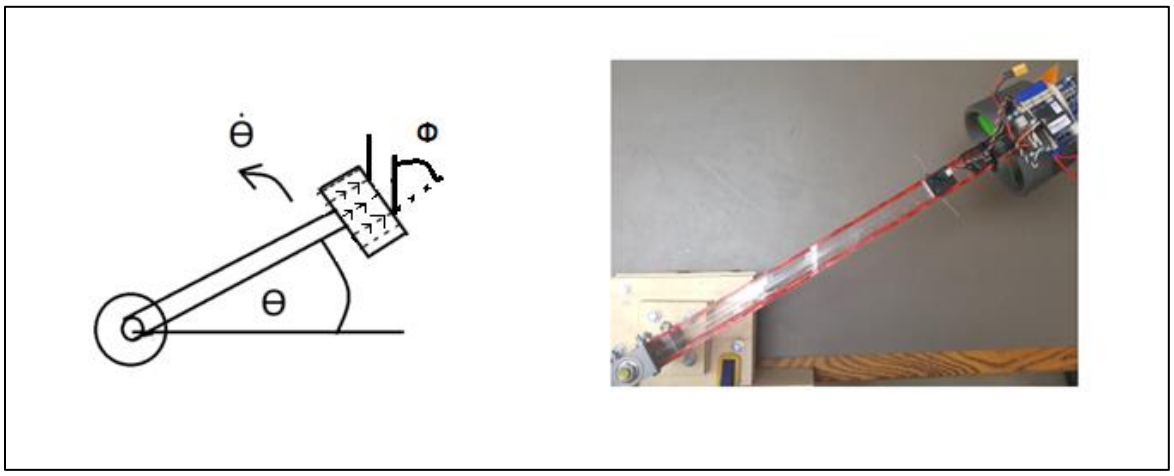

Figure 3.2 Visual top view of Blotator showing all three states (left) and experimental Blotator apparatus (right)

3.1.2 Discretized State Space

The Blotator's rudders can only experience three possible angles, with a short transition time of roughly 0.1 seconds between rudder angle changes. These three rudder angles can be easily translated to a discrete state space. However, the arm angle θ and the rotation rate $\dot{\theta}$ both take a continuous form. Because application of Q-Learning requires a discrete state space the continuous values for θ and $\dot{\theta}$ were discretized.

Each dimension of the state space was assigned minimum and maximum values and were discretized into a finite number of equally spaced points in the form of a discrete grid. If a state's recorded value was lower than that state's minimum value, then a grid index of zero was assigned. If the state's recorded value was greater than or equal to that state's maximum value, then a grid

index equal to one less than the number of discretized points for that state dimension was assigned. If the state's recorded value was between the minimum and maximum value, then a grid index which corresponds with that state value depending on the number of discretized grid points was assigned, where the distance between each grid point relative to the state value was equal to the maximum state value minus the minimum state value divided by one less than the discretized number of grid points. This process is shown visually in Figure 3.3.

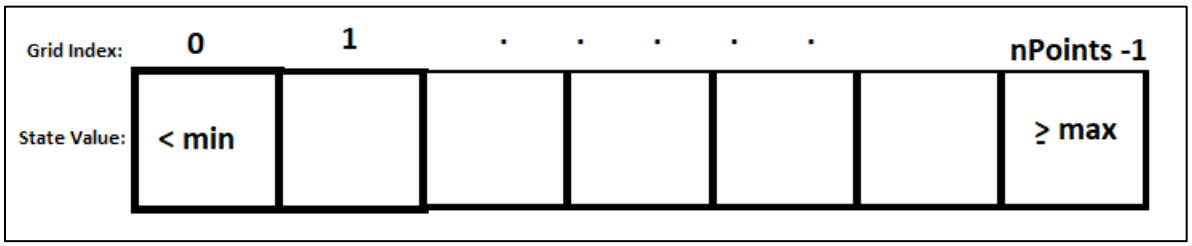

Figure 3.3 Discrete form of a continuous state dimension's grid indexing

3.1.3 Blotator Actions

The Blotator can perform 3 different actions which are achieved by adjusting the current rudder angle. The reinforcement learning algorithm offers the options of either increasing the rudder angle, decreasing the rudder angle or leaving the rudder angle the same. The Blotator can only experience three possible rudder angles which are defined as a maximum rudder angle, a minimum rudder angle, and a middle rudder angle that is achieved when $\phi = 0$. Because there are only three possible rudder angles the discrete state space for dimension ϕ only consists of three

elements. When ϕ is equal to the minimum or maximum rudder angles a grid index of zero or two was assigned respectively for that state dimension. When ϕ takes on a value of zero a grid index of one was assigned for that state dimension. A visual representation of the rudder angle's state space grid indexing can be seen in Figure 3.4

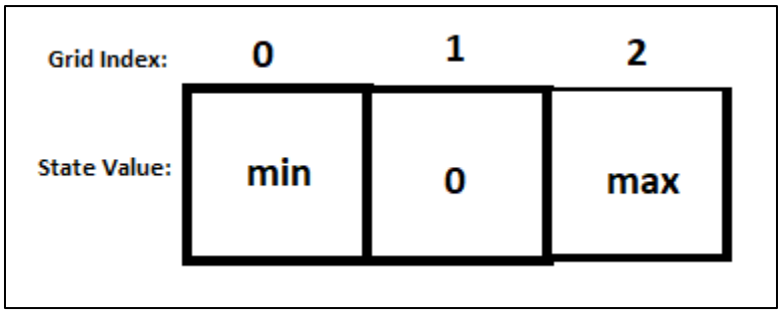

Figure 3.4 Discrete form of rudder angle grid indexing

3.2 Hardware

In addition to the physical structure shown in Figure 3.1, the Blotator also possesses an inertial measurement unit (IMU) to sense the angle θ and its time derivative $\dot{\theta}$. It has a servo motor to actuate the rudder and a brushless AC motor paired with an electronic speed controller (ESC) to power the propeller. All sensors and motors are coordinated by a BeagleBone Blue single board computer which also provides the computational power and data storage necessary to implement the Q-Learning algorithms. The BeagleBone Blue can communicate wirelessly with a laptop computer.

The Blotator apparatus is equipped with a BeagleBone Blue single board computer along with the Robotics Cape Library which is developed by Strawson Design [4]. BeagleBone Blue comes equipped with 512MB of DDR3 RAM and a 1GHz processor and uses a Linux operating system [12]. The device's IMU consists of a three-axis gyro sensor, three-axis accelerometer, and a magnetometer, which are used to measure angular position and velocity in real time. The board also contains a separate processor and circuitry for sending PWM signals to the servo motor to actuate the rudder, and the ESC running the propeller's brushless motor. All hardware attached to the Blotator was powered by a 2-cell 7.4V LiPo battery for full wireless control. Real time monitoring of the Blotator was done through wireless communication between the BeagleBone Blue and a laptop computer using a web server port running the Cloud9 IDE.

3.2.1 Sensors (IMU)

The BeagleBone Blue has access to an onboard 9 degree of freedom IMU containing a 3-axis accelerometer, a 3-axis gyro, and a 3-axis magnetometer [12]. These components of the IMU are used to acquire data related to both the angular position and angular velocity of the Blotator. The information from these sensors is received at a frequency of roughly 10 Hz through access to the Robotics Cape Library's gyro method for direct measurement of the Blotator's angular velocity, and an integration of the gyro's input was performed to approximate the Blotator's angular position. This integration was done due to insufficient readings from the accelerometer because the Blotator experienced pure rotation in a horizontal plane. Because the value for θ is an approximation through integration of $\dot{\theta}$ there is a numerical "drift" in the estimated value for θ.

Because sensor information is taken 10 times per second the drift is minimized, however roughly 1° of drift was experienced per second. A rate of roughly 10 sensor samples per second was also sufficient to explore the size of the experimentally discretized state space.

3.2.2 Motors

An HXT900 servo motor was used to control the Blotator's rudder angle and an LDPower M2204 – 2300KV brushless AC motor paired with an ESC was used to spin the Blotator's 4-inch diameter propeller. The servo motor can supply up to 1.6kg-cm of torque [13] which is more than enough to control the Blotator's rudder angle. The brushless AC motor can withstand a maximum current of 18.1Amps. Throughout the experiment roughly 30% maximum throttle was ever used at any given time, which requires less than 3Amps of current when using a 7.4V 2S LiPo battery. Maximum estimated power at these specifications was less than 20Watts [14].

3.3 Software

Custom software was written that can implement Q-Learning. The software sets up appropriate data structures, allows one to specify different reward schemes, establishes training episodes and stepping protocols, records information that is "learned" from each Q-Learning step, and provides visualization of the learned Q-Values throughout the system's state space.

A majority of the code was written using object-oriented C++. One version was compiled with Gnu g++ for use on the BeagleBone. This version connected the Q-Learning algorithm to the board's sensors and motors. A second configuration of the software was created using Visual Studio on a PC running Windows. This second software configuration was connected to a computational simulation of the Blotator. A third program was written in Python which provided visualization of the discretized state space with learned Q-Values.

3.3.1 Blotator Simulation

A numerical simulation of the Blotator was created to test Q-Learning implementation quickly and inexpensively. The simulation can be used to test different forms of discretizing the state space, different reward functions, different hyperparameters such as the learning rate and discount factors, and different state-sampling intervals. The simulation can also be used to attain a rough estimate on how much time it may take to train the physical system.

The simulation makes use of a 4th order Runge Kutta (RK4) integrator that estimates the states θ and $\dot{\theta}$ by integrating the differential equations seen in Equation 3.1 and Equation 3.2 where F is the force supplied by the propeller, L is the length of the Blotator arm, ϕ is the simulated rudder angle and I is the moment of inertia.

$$\omega = \dot{\theta} \tag{3.1}$$

$$\dot{\omega} = \ddot{\theta} = \frac{FLsin(\phi)}{I} \tag{3.2}$$

The RK4 integrator estimates the states θ and $\dot{\theta}$ using a simulated time-step of 0.01 seconds. A total of 10 RK4 time-steps are executed before a simulated state value is sent to the Q-Learning algorithm to simulate 0.1 seconds of dynamic motion. The Q-Learning algorithm takes place on a time-step that is ten times larger than the RK4 integrator's time-step.

CHAPTER 4
SIMULATION RESULTS

As stated in Chapter 3, the purpose of creating a simulation of the Blotator is to confirm conditions under which Q-Learning is viable. This chapter is focused on the results of the simulation-based learning for which promising results worthy of being testing in the physical system were achieved.

4.1 Simulation

As previously discussed in Chapter 3.3 a simulation of the Blotator apparatus was created using Object Oriented programming in C++. This simulation made use of an RK4 integrator to estimate the system's future states after an action had been performed. Three different policies were generated simply by modification of the reward function. Simulated episodes each consisting of 200 Q-Learning time-steps were chosen to be performed to sufficiently explore the state space by simulating a 20 second exploration phase. After a 200-action episode has finished the values for each state are randomized and a new episode begins. Simulated values for the angle θ and the rotation rate $\dot{\theta}$ were plotted to show the system's behavior.

4.1.1 RK4 Integrator

A 4^{th} Order Runge Kutta integrator was developed for the simulation. Because each episode consisted of 200 actions minimizing the error in each state estimation became critical. It was found that some policies required over 1,000 episodes to fully explore the state space, which consequentially required over 200,000 simulated actions to perform. To minimize the error as much as possible the RK4 integrator was chosen for this simulation.

The RK4 integrator used in the simulation integrated Equation 3.1 and Equation 3.2 to estimate state values for θ and $\dot{\theta}$. Simulated parameter values for L and I were held constant while the simulated propeller force F was adjusted until the simulation produced noticeable numerical changes in RK4 estimated values for state θ to generate rewards over each simulated 200-action episode.

4.2 Reward Function

The control objectives for the system are defined through the reward function. To create a unique policy the reward system must be defined. Throughout the simulation process three separate reward functions were defined each of which created its own unique policy which describe a specific control objective through Equation 2.2. Each unique control objective specifies a unique style of behavior to be simulated. Control objectives which were investigated consisted of spinning the Blotator in either a clockwise or counterclockwise motion at a reasonably slow

rotation rate, as well as performing a stopping action when the state value for θ was reasonably close to zero. State information related to the system's current states and the previous states were sent to the reward function every 0.1 seconds to simulate a 0.1 second time-step between state readings. Depending on the transition between the previous and current state for the Blotator's angular position and angular velocity the reinforcement learning algorithm may receive a positive reward, a negative reward, or a reward value of zero.

The first policy was created based only off previous and current information related to the Blotator's angular velocity and used a learning rate of 0.15 to incorporate small updates in learned Q-Values after each action. A discount factor of 0.98 was chosen so that the earliest actions in each episode have significantly more weight than the final actions per Equation 2.2. The continuous states were discretized into 20 evenly spaced segments ranging between -π to π for state θ, and 7 evenly spaced segments ranging from -1.5 Radians Per Second (RPS) to 1.5RPS for state $\dot{\theta}$ in accordance with Figure 3.3. Positive reward values of 5.0 were received when the Blotator simulation experienced relatively low rotation rates producing negative $\dot{\theta}$ values ranging between -0.3RPS to -0.7RPS, where the maximum and minimum possible values for $\dot{\theta}$ were 1.5RPS and -1.5RPS respectively. If the simulated values for $\dot{\theta}$ were ever greater than 1.5RPS or less than -1.5RPS then a negative reward value of -10.0 was produced. The system was encouraged not to perform a stopping action so if state values for $\dot{\theta}$ ever switch signs over the course of a 200-action episode a large negative reward value of -1000.0 was produced. The system was also encouraged not to speed back up after a relatively slow value for $\dot{\theta}$ was achieved. If $\dot{\theta}$ begins speeding back up beyond ±0.5RPS a negative reward value of -100.0 was produced. Graphical display of the explored state space showing each cell's maximum Q-Values were

produced at 200 episodes and 400 episodes respectively and can be seen below in Figure 4.1 and Figure 4.2. Red state cells indicate that the maximum Q-Value for that state cell is a negative value, while green state cells indicate a positive maximum Q-Value. Yellow State cells indicate that the maximum Q-Value for that state cell is equal to zero. The state space seems to be sufficiently explored after roughly 400 episodes.

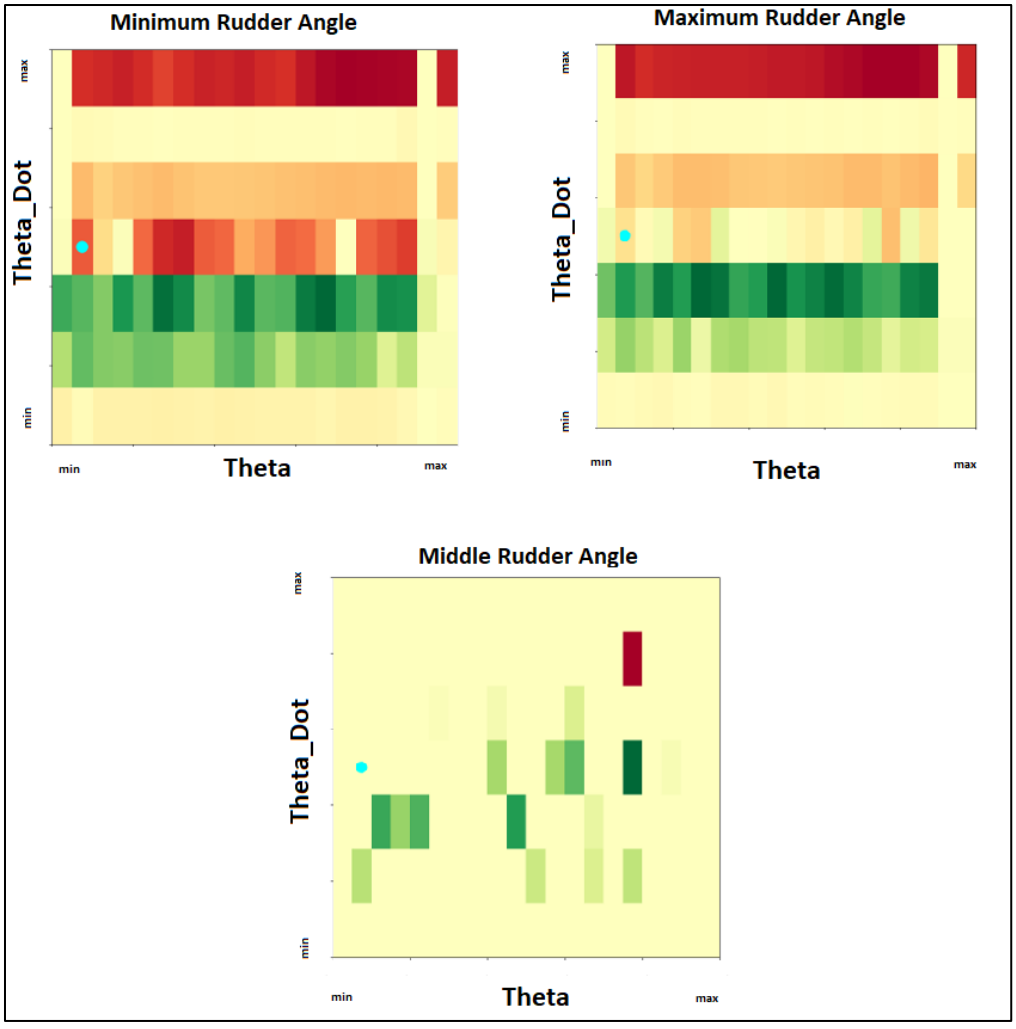

Figure 4.1 Graphical display of explored state space for policy 1 after 200 episodes

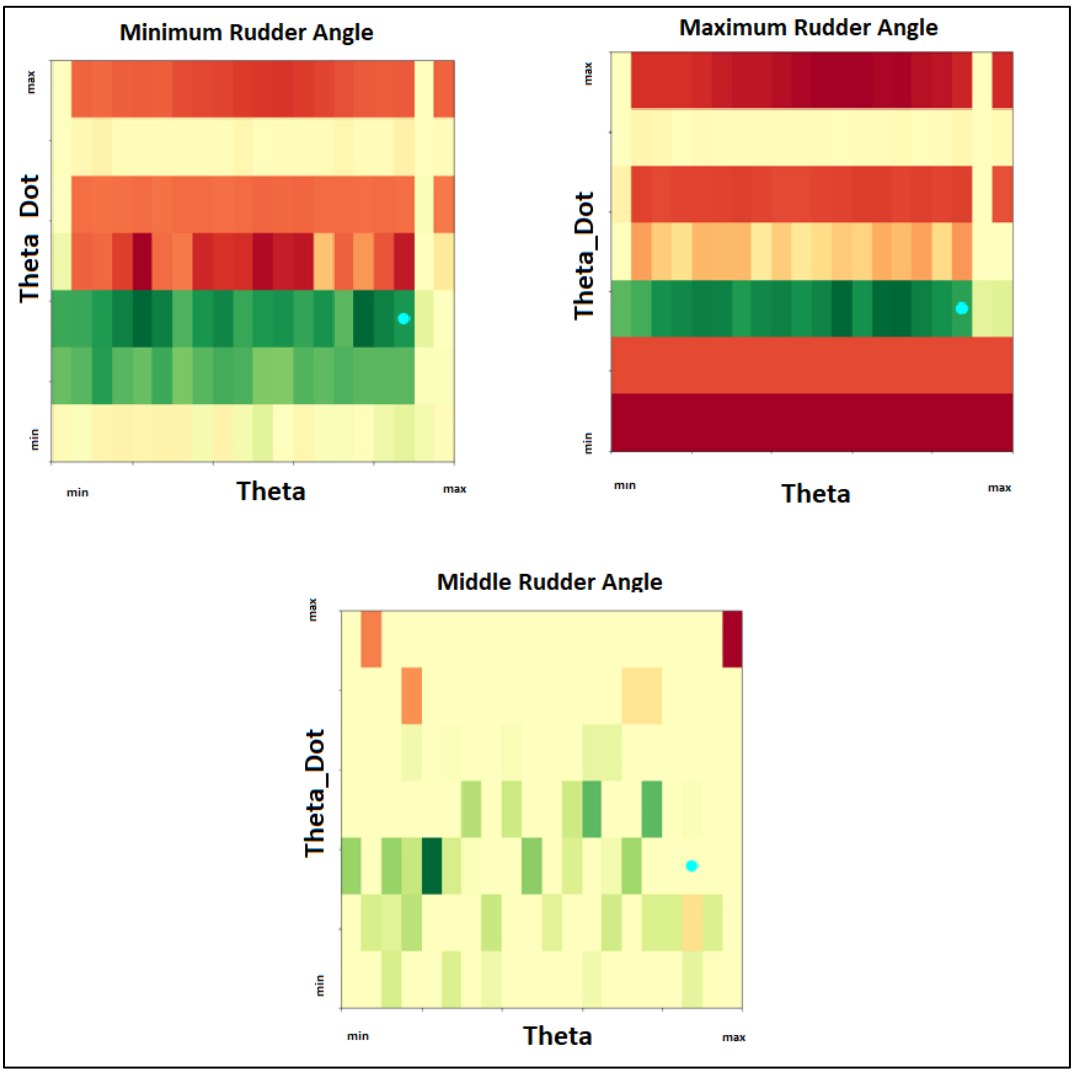

Figure 4.2 Graphical display of explored state space for policy 1 after 400 episodes

With a slight change to the algorithm's reward system a second policy was generated. Learning rates, discount rates and state space discretization were kept the same as policy 1. The only change to the reward system was that by experiencing a $\dot{\theta}$ state value ranging between a positive range of (0.3RPS) to (0.8RPS) a positive reward value of 5.0 was produced. The reinforcement learning algorithm showed a sufficiently explored state space in roughly the same

number of episodes as policy 1. Figure 4.3 shows a graphical display of the explored state space and each cell's maximum Q-Values after 400 episodes for the second policy.

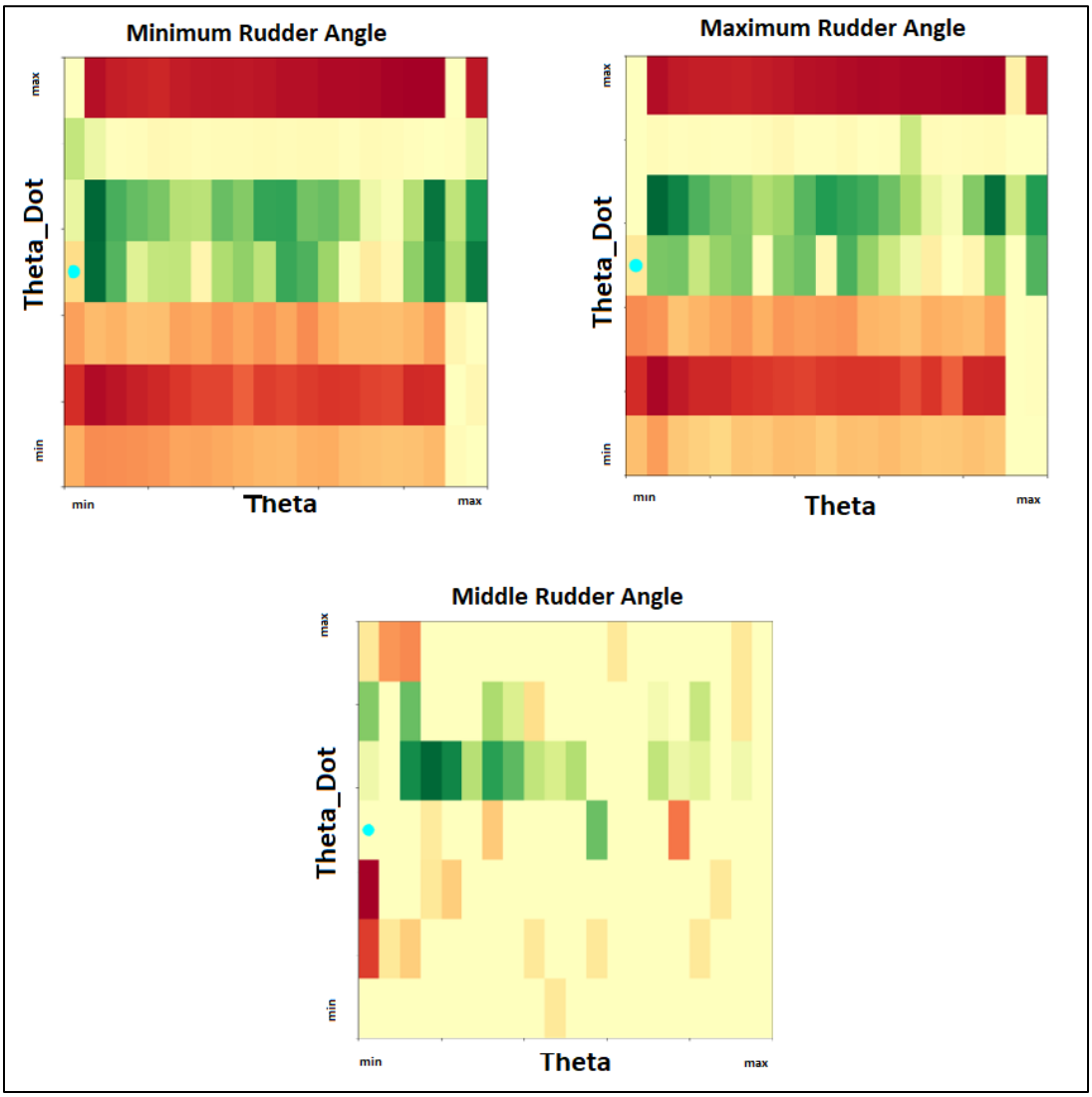

Figure 4.3 Graphical display of explored state space for policy 2 after 400 episodes

Finally, a third policy was generated through a final modification of the reward system. The learning rate and discount rate were kept constant at 0.15 and 0.98 respectively. However, the discretized size for state $\dot{\theta}$ was decreased from 7 to 6 to allow the system to sufficiently explore the state space in a fewer number of episodes. For this third policy positive rewards were associated with the current and previous states of both the angular position and angular velocity of the Blotator's arm. If the value for $\dot{\theta}$ transitions from a negative value to a positive value or vice versa then the algorithm has simulated a stopping action and a positive reward value of 10.0 is received. If a stop is simulated within roughly 40% of $\theta = 0$ a positive reward value of 75.0 is produced, and if a stop is simulated within roughly 20% of $\theta = 0$, which will be referred to as "Home Base", the maximum positive reward value of 500.0 is produced. Negative reward values of -10.0 were produced if the system experienced simulated values for $\dot{\theta}$ that were greater than 1.5RPS or less than -1.5RPS. A negative reward value of -100.0 was produced if the simulation produced relatively low values for $\dot{\theta}$ ranging between ±0.5RPS but then increases beyond that. A positive reward value of 25.0 was produced whenever the system entered the roughly 40% region surrounding Home Base, and a negative reward value of -100.0 was produced if the system leaves the roughly 40% region surrounding home base. Graphical display of the system's explored state space was recorded at 300 episodes, 700 episodes and 1200 episodes, after which a fully explored state space can be seen. These results are displayed in Figure 4.4, Figure 4.5 and Figure 4.6 below.

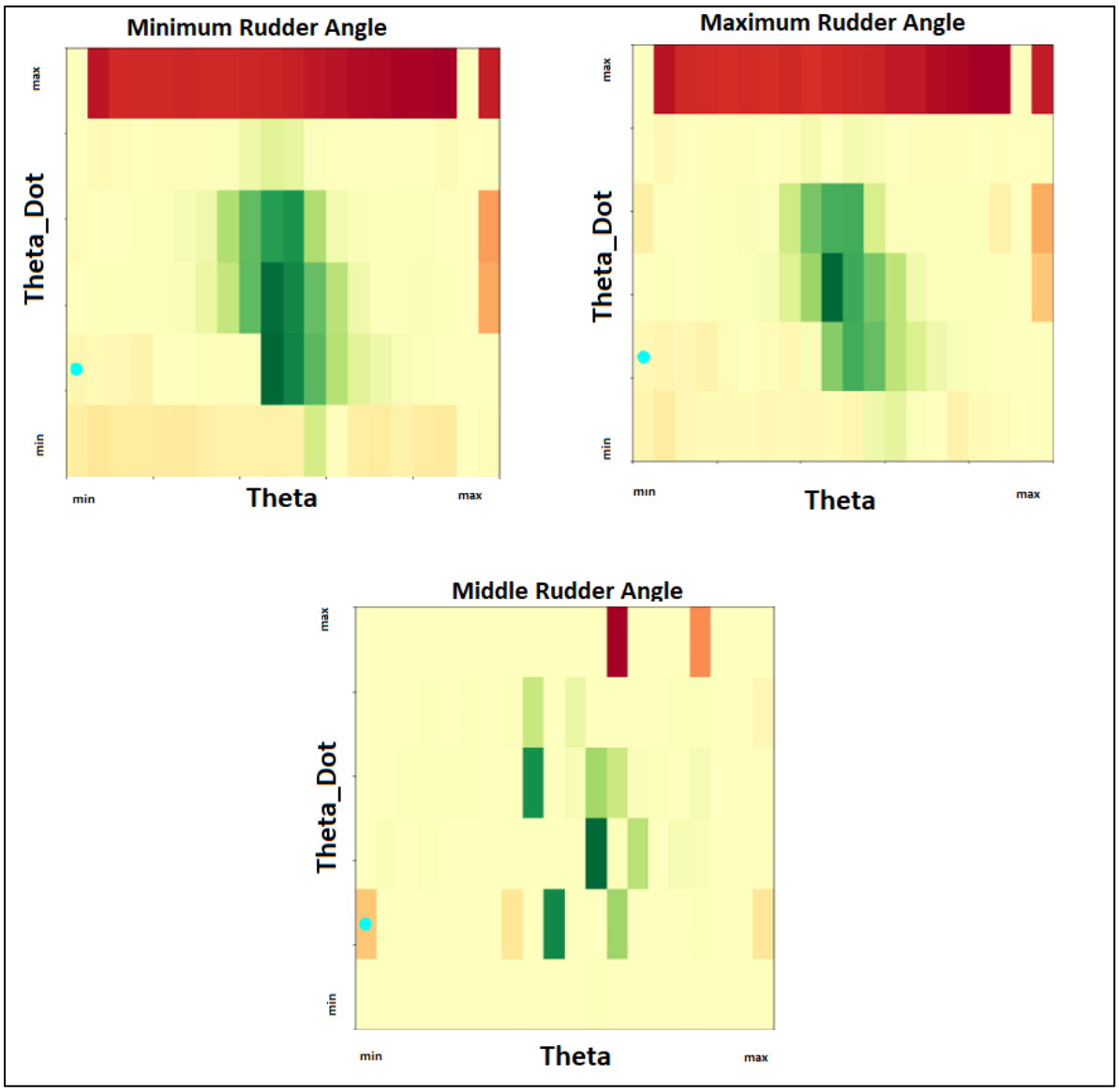

Figure 4.4 Graphical display of explored state space for policy 3 after 300 episodes

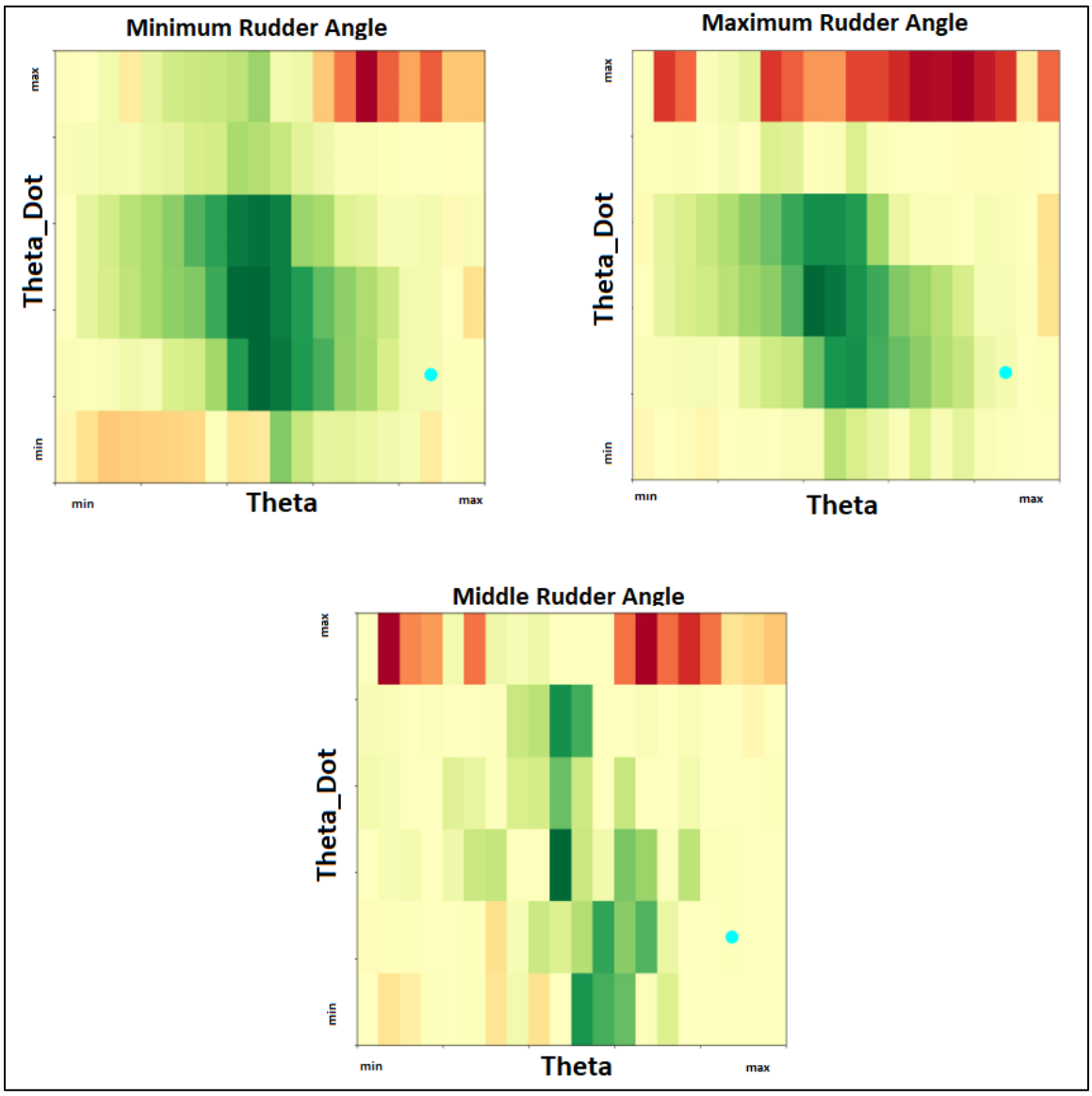

Figure 4.5 Graphical display of explored state space for policy 3 after 700 episodes

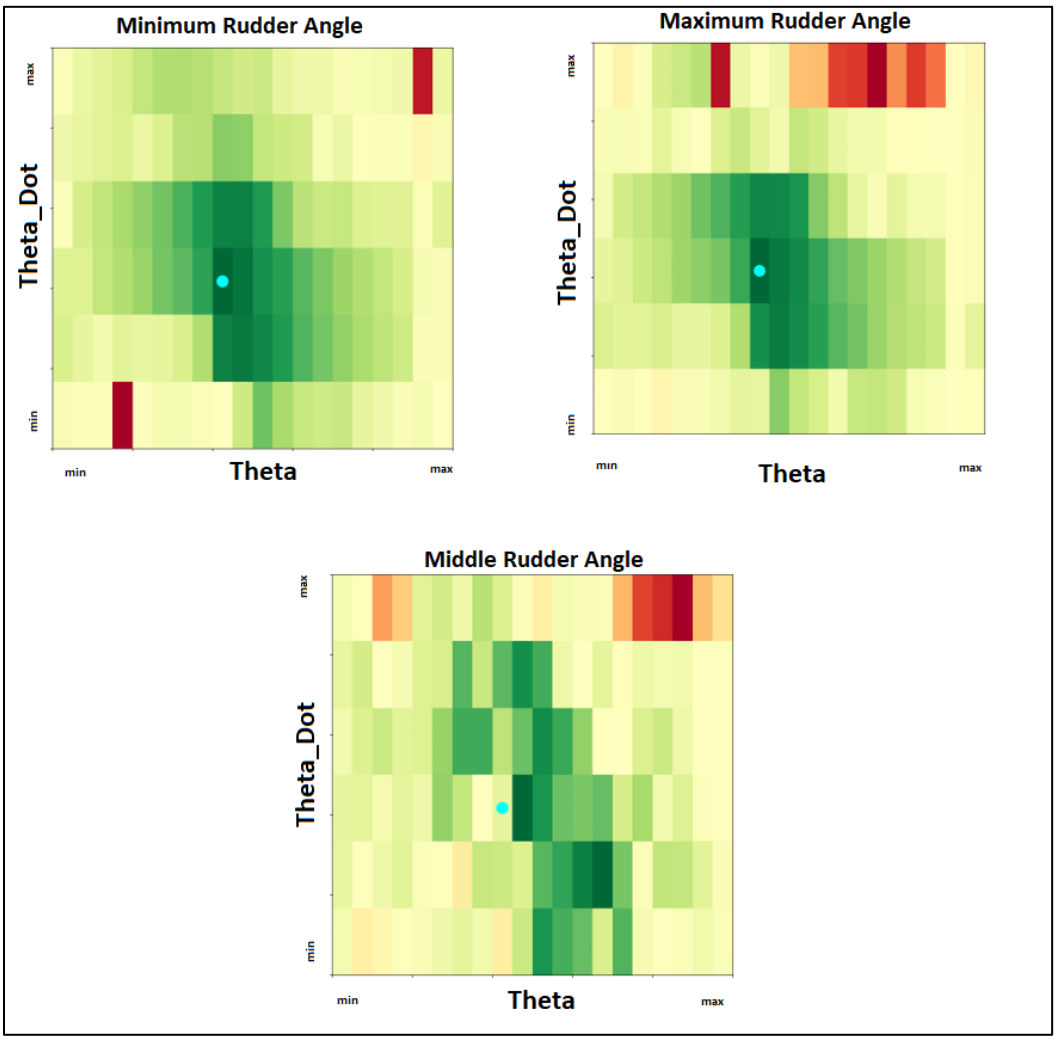

Figure 4.6 Graphical display of explored state space for policy 3 after 1200 episodes

4.3 Simulated System Behavior

Estimated values for states θ and $\dot{\theta}$ were recorded for each policy throughout the exploration process to show roughly 10 seconds of simulated dynamic motion. Simulated state results for policy 1 and policy 2 show that the expected physical control objectives were achieved

by episode 200 and can be seen in Figure 4.7 and Figure 4.8. Discontinuity seen in state θ is representative of a transition from -π to π or from π to -π along the circumference which the simulated arm travels about.

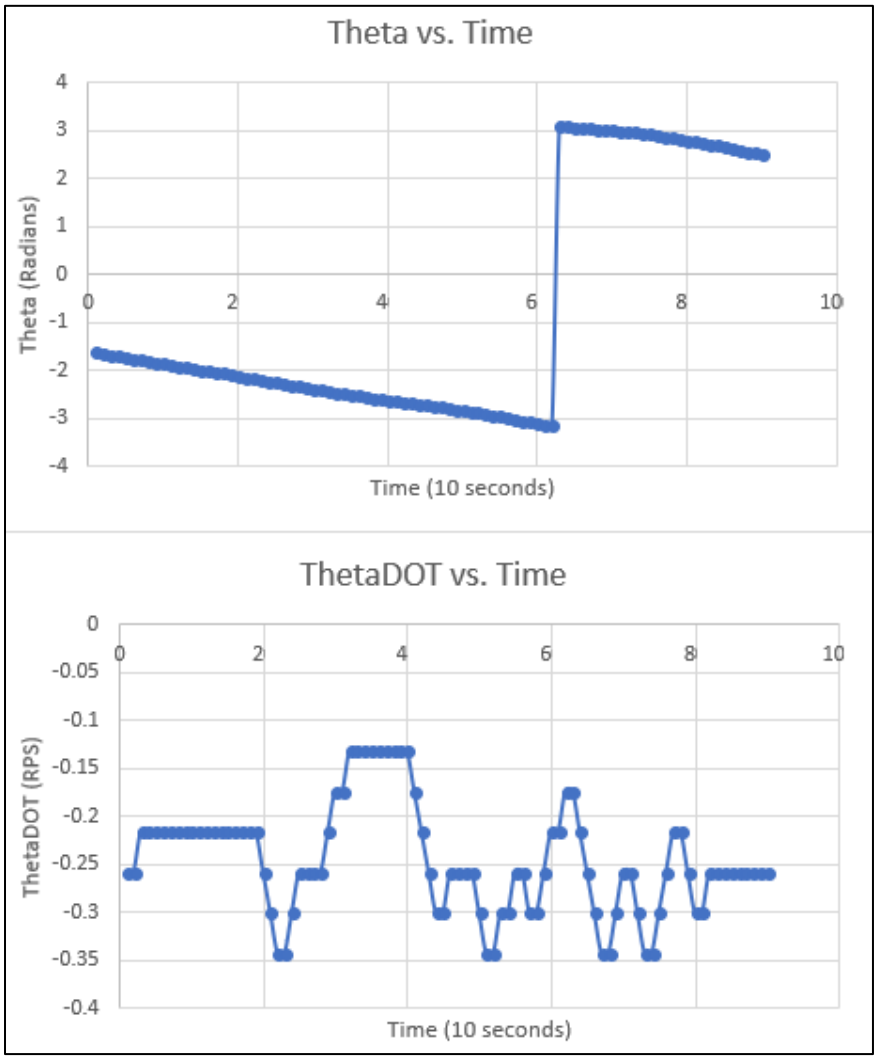

Figure 4.7 Simulated behavior experienced under simulation policy 1 after 200 episodes

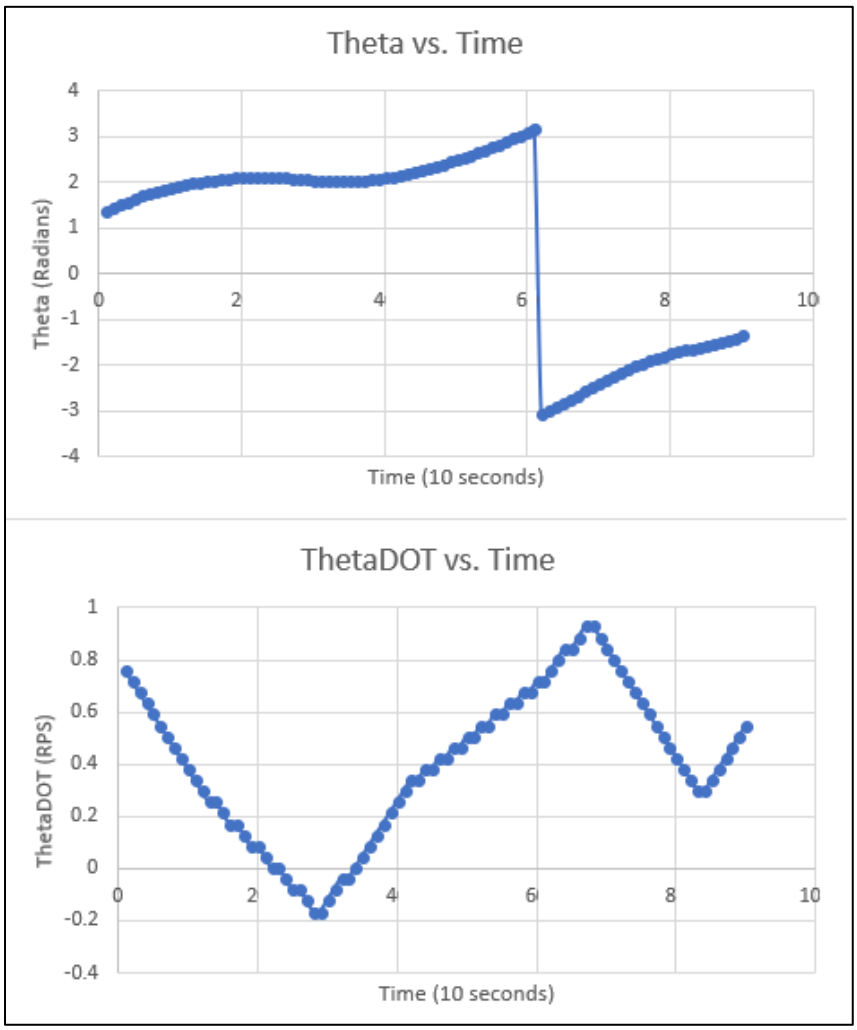

Figure 4.8 Simulated behavior experienced under simulation policy 2 after 200 episodes

Simulated state results for policy 3 were recorded at 300 episodes and 700 episodes and can be seen in Figure 4.9 and Figure 4.10 respectively. At each of these stages of the exploration process the simulated state results show that the control objective for the third policy were also achieved.

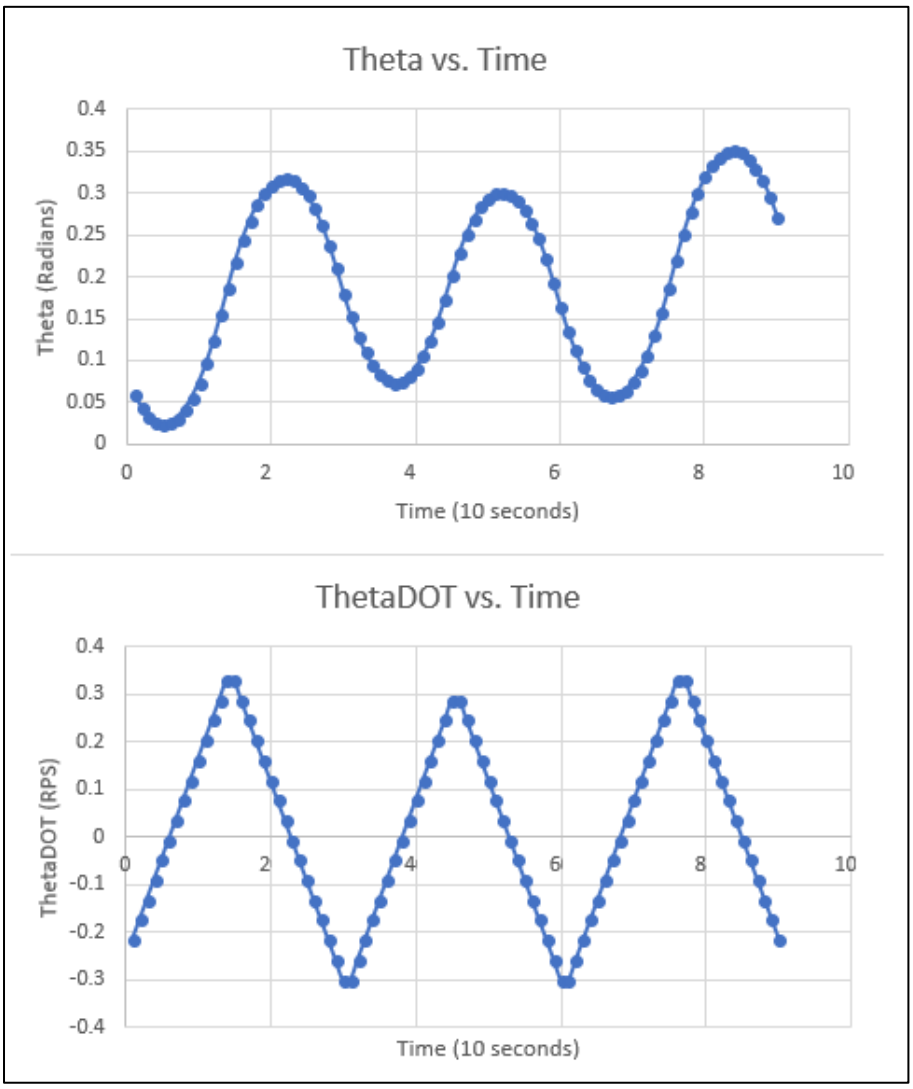

Figure 4.9 Simulated behavior experienced under simulation policy 3 after 300 episodes

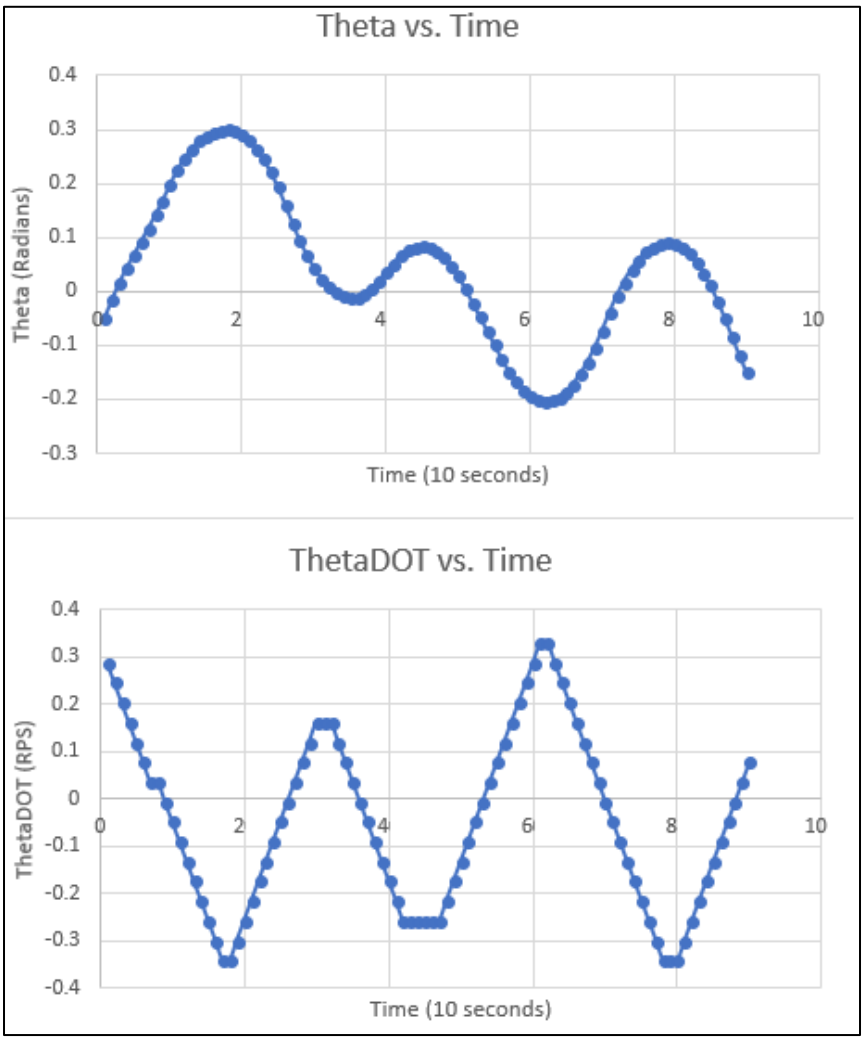

Figure 4.10 Simulated behavior experienced under simulation policy 3 after 700 episodes

It can be seen from the comparison between the Figures in Chapter 4.3 and the Figures in Chapter 4.2 that a full exploration of the state space is not necessary to achieve simulated state behavior in accordance with the expected control objective defined by the policies' reward function. Policy 1 and policy 2 both experienced simulated states in accordance with their control objectives within 200 episodes of simulated exploration per Figure 4.7 and Figure 4.8. Policy 3

experienced simulated states in accordance with its' control objective within 300 episodes, while a full exploration of the state space is not seen until 1200 episodes per Figure 4.6 and Figure 4.9.

4.4 Convergence Time

Simulated state convergence occurred for all three policies with respect to their control objectives and convergence times were estimated for each individual policy. The first two policies that were generated cause the Blotator to rotate slowly in only one direction. Each of these simulated policies require roughly 200 simulated episodes before the simulated states produced dynamics in accordance with the control objectives. To complete 200 total episodes at roughly 20 simulated seconds per episode the estimated convergence time for the first two policies is roughly 4000 seconds, or 1 hour of real time training.

The third policy required roughly 300 episodes of simulated Q-Learning before the simulated states showed signs of following the expected control objective per Figure 4.9. To complete 300 episodes at 20 seconds per episode it may take roughly 6,000 seconds of training. This leads to an estimated convergence time of roughly 1.5 hours in real time.

CHAPTER 5
EXPERIMENTAL RESULTS

Physical results from the Blotator apparatus were generated for two unique policies. Each of these individual policies were created through slight modifications of the reward function to produce different sets of Q-Values. Similarities between simulated and experimental state spaces attained throughout the exploration process are observed along with the physical behavior of the Blotator apparatus.

5.1 Experimental Reward Function

Two separate reward functions were generated to created two unique policies. The first reward function was created on the BeagleBone Blue and rewarded the Q-Learning algorithm with a positive reward of 5.0 for positive $\dot{\theta}$ values that were within a range of 0.01RPS to 1.1RPS after an action had been taken, and a negative reward values of -30.0 for any $\dot{\theta}$ values that were less than 0.0RPS or greater than 1.1RPS.

The second reward function was also created on the BeagleBone Blue. This reward function rewarded the Q-Learning algorithm with negative reward values of -10.0 whenever $\dot{\theta}$ values were greater than 1.5RPS or less than -1.5RPS to discourage the Blotator from experiencing

large angular velocities. Positive reward values of 3.0 were produced whenever θ values fell within a range of -20° and 20° which will be referred to as "Home Base", and a positive reward value of 1.0 whenever θ values were within 20% of Home Base. Positive rewards of 10.0 were also produced when $\dot{\theta}$ values changed signs, indicating a stop. A positive reward value of 25.0 was granted when the value for θ transitions into Home Base and a negative reward value of -100.0 was assigned if θ transitions out of Home Base. Finally, a Grand Prize positive reward value of 500.0 was given whenever $\dot{\theta}$ changed signs within Home Base.

5.1.1 Experimental Policies

Two policies were generated through experimentation, each of which describe a specific type of motion. The first policy generated through experimentation describes a counterclockwise rotation and was generated through implementation of the first reward function. This experimental policies' explored state space can be seen in Figure 5.1 and was generated after 34 experimental episodes each consisting of 200 actions.

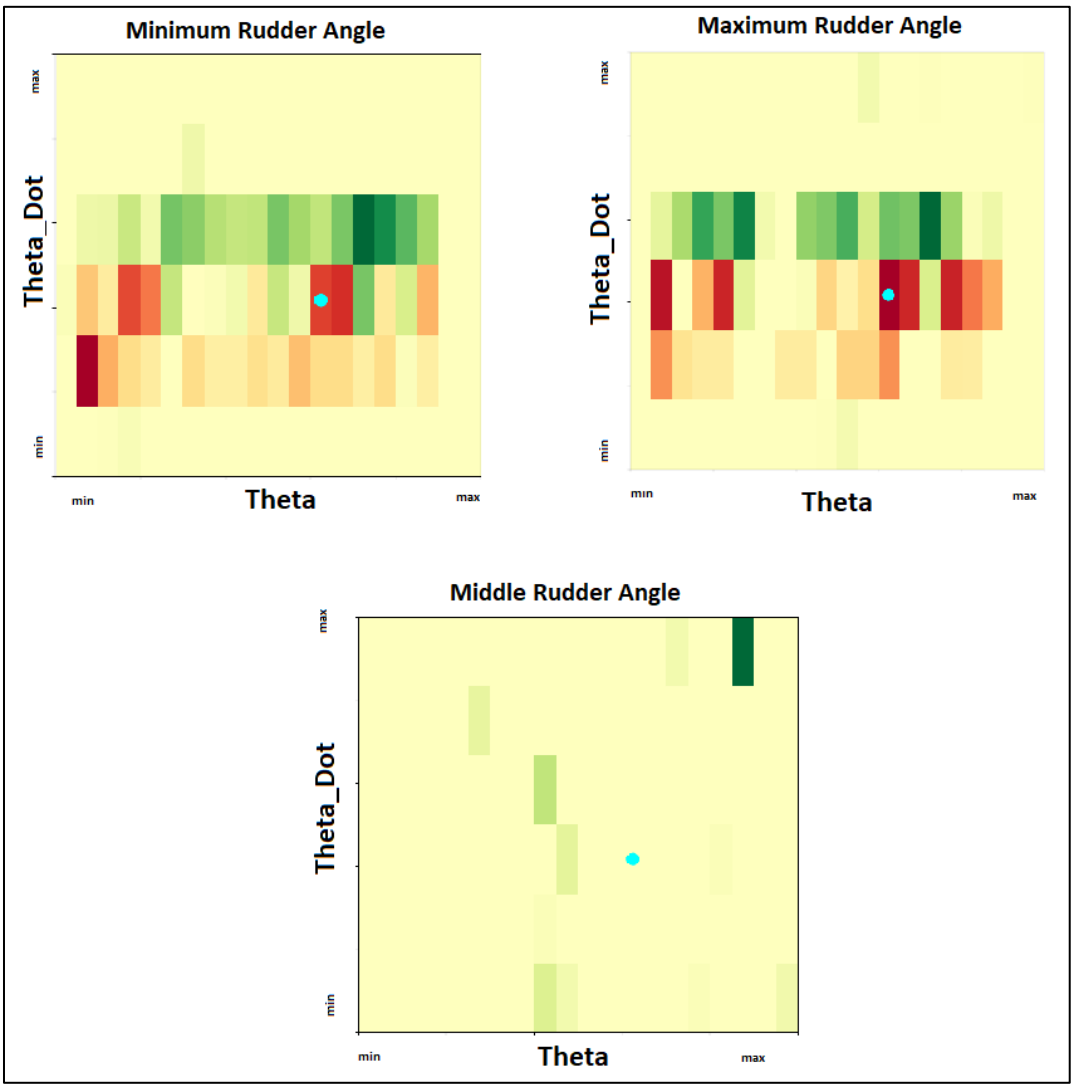

Figure 5.1 Graphical display of experimental state space for policy 1 after 34 experimental episodes

The second policy generated through experimentation describes a stopping motion that occurs at Home Base and was generated through the second reward function. This policies' explored state space can be seen in Figure 5.2 and was generated after 36 experimental episodes.

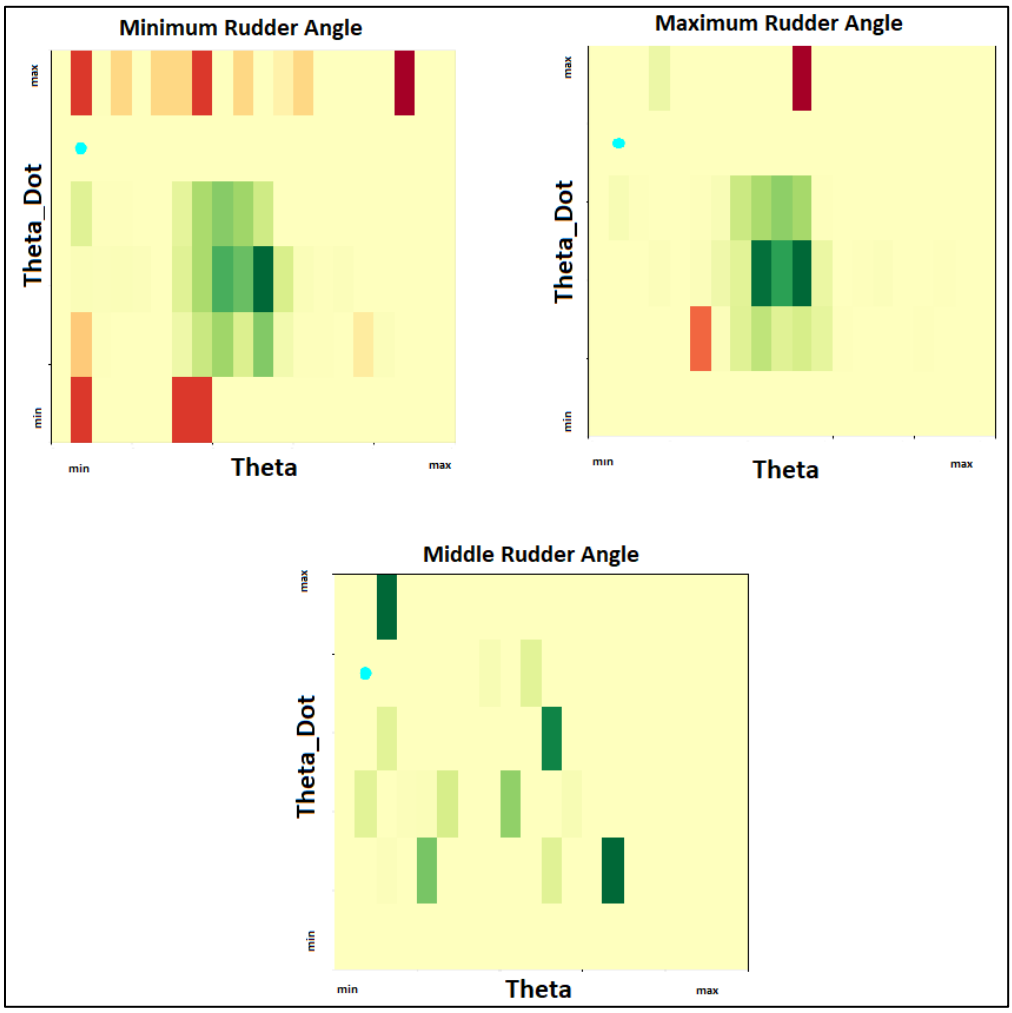

Figure 5.2 Graphical display of experimental state space for policy 2 after 36 experimental episodes

5.2 Physical Behavior

Physical behavior and state values for θ and $\dot{\theta}$ according to each experimental policy discussed in Chapter 5.1.1 were recorded for roughly 10 seconds. When performing actions in

accordance with the first experimental policy the Blotator experienced constant counterclockwise motion which can be seen in Figure 5.3. A change in direction can be seen when the value for $\dot{\theta}$ drops below zero in Figure 5.3 but the algorithm quickly adjusts for this error and continues with constant counterclockwise motion. Discontinuity seen in state θ is representative of a transition from 180° to -180° along the circumference which the Blotator's arm travels.

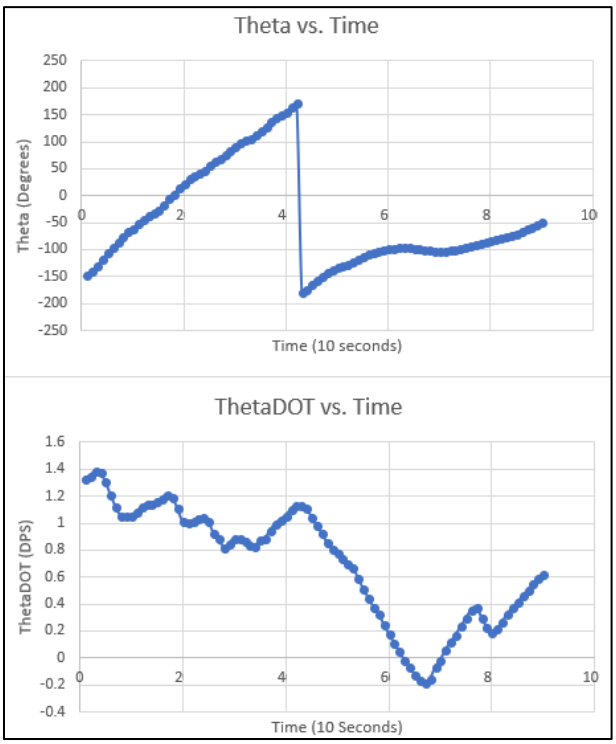

Figure 5.3 Physical behavior experienced under policy 1

When performing actions in accordance with the second experimental policy the Blotator would oscillate over Home Base and can be seen in Figure 5.4. Positive reward values were given when θ was within a range of -20° and 20°. Positive reward values were also given whenever $\dot{\theta}$ crossed zero on the y-axis.

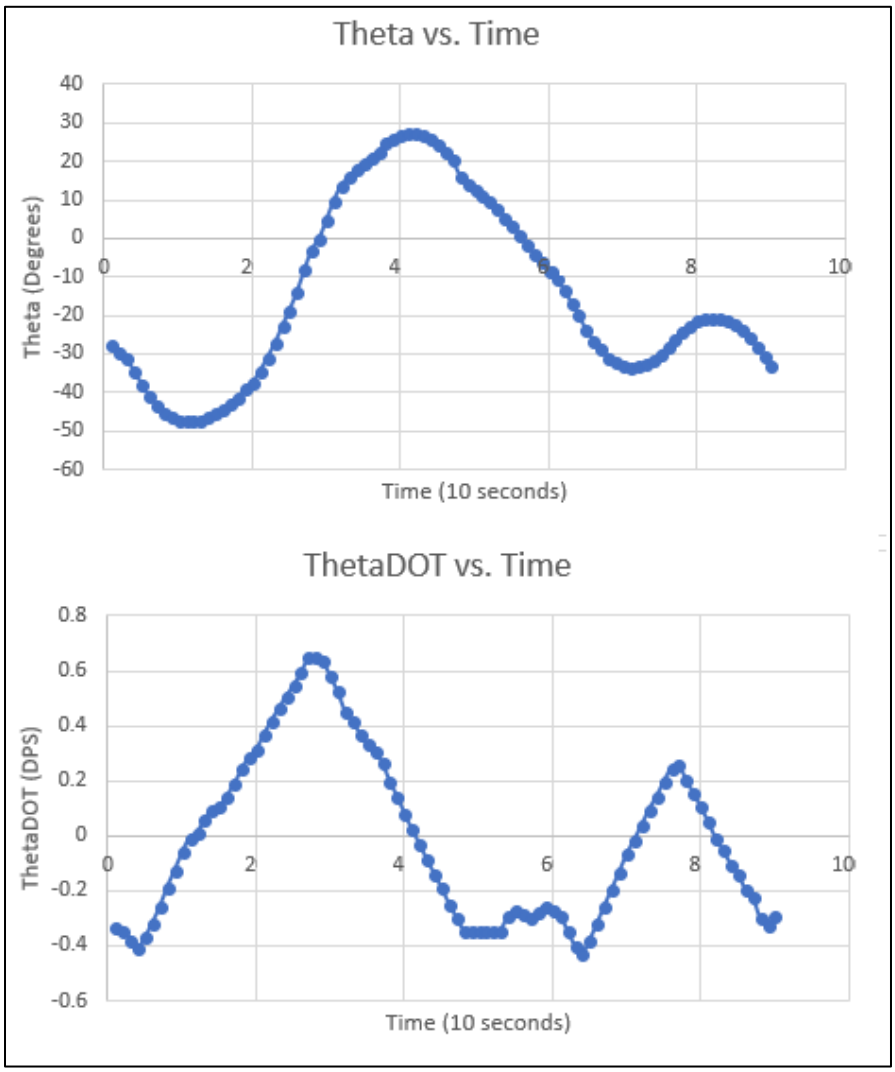

Figure 5.4 Physical behavior experienced under policy 2

5.2.1 Unique Learning Experience

While undergoing the experimental exploration process it was possible to manipulate the way that the Blotator apparatus would learn. Depending on which state cells are explored at the very beginning of the exploration process the physical system may converge toward different

unique solutions early in the exploration phase. When observing these different types of unique solutions in the experimental system unique physical phenomena began to occur. The Blotator apparatus was capable of learning "Bad Habits" throughout the exploration process.

One unique Bad Habit that was learned throughout the exploration process while attempting to generate the second experimental policy was that the Blotator developed a bias to always spin clockwise. If the Blotator accidentally overshot Home Base, it would often prefer to swing 360° to find Home Base again rather than simply change direction. These policy generation attempts were disposed of; however, they implemented the same reward system that was used to generate the policy seen in Figure 5.2. One key difference in the learning process when attempting to generate this policy was the physical conditions that the Blotator experienced during the first few episodes in the exploration process.

If the Q-Learning algorithm received very large rewards associated with stopping at Home Base early in the exploration process the solution would converge toward that set of physical states. Physically manipulating the system through external means and bringing it to a set of ideal states early in the exploration process not only eliminated certain types of Bad Habits but also played a role in experimental state convergence time.

5.3 Experimental Convergence

Experimental state convergence was achieved for both experimental policies. These two policies showed physical state behavior in accordance with their respective control objectives at

34 episodes and 36 episodes respectively. The experimental state spaces seen in Figures 5.1 and 5.2 are visually similar to those seen in Figures 4.3 and 4.4. Roughly the same amount of the total state space was explored in 30 to 40 experimental episodes as was explored in 300 to 400 simulated episodes. Experimental physical behavior for both policies showed physical state convergence towards the desired control objectives of either spinning counterclockwise or stopping at Home Base as seen in Figure 5.3 and Figure 5.4.

CHAPTER 6
FUTURE IMPROVEMENTS

Future improvements into the system are briefly discussed in this chapter. Efforts to further investigate the exploration of the system's state space could be made and hardware upgrades to the experimental Blotator apparatus could be implemented to develop these more complex policies.

6.1 Cumulative Reward

Each performed action has the opportunity to receive a reward, and the cumulative sum of these received rewards is calculated at the end of each 200-action episode per Equation 2.1. As stated in Chapter 2, the generated policy is created by attempting to maximize this cumulative reward. As the total number of explorative episodes increases with the generation of a unique policy the cumulative reward value at the end of each episode should increase as well. It would be beneficial towards the investigation of the exploration process to further investigate the behavior of this cumulative reward.

6.2 Upgraded Experimental Apparatus

To develop more complex policies hardware upgrades to the experimental Blotator apparatus could be implemented. One possible upgrade to the system could be the addition of a vision sensor. This vision sensor could add at least one additional dimension to the state space by allowing the system to identify specific colors and more complex policies could be generated with this simple hardware upgrade.

6.2.1 Vision Sensor

A vision sensor could be attached to the end of the Blotator apparatus to create a higher dimension state space which can generate more complex policies. The addition of a vison sensor would allow the Blotator apparatus to sense things like color and visual motion, each of which could produce an additional dimension to the Q-Learning algorithm's state space. The additional state space dimensions will consequentially lead to longer state convergence times but will be capable of producing more complex policies once the exploration process has been completed.

REFERENCES

[1] Sutton, R. S., & Barto, A. G. (2005, January 4). Temporal-Difference Learning. Retrieved June 13, 2018, from http://incompleteideas.net/book/ebook/node60.html

[2] Sutton, R. S., & Barto, A. G. (2005, January 4). Reinforcement Learning: An Introduction. Retrieved June 13, 2018, from http://incompleteideas.net/book/ebook/the-book.html

[3] Klein, D. (2012, August/September). UC Berkeley CS188 Intro to AI -- Course Materials. Retrieved June 13, 2018, from http://ai.berkeley.edu/lecture_videos.html

[4] Strawson Design. (2017). Retrieved June 13, 2018, from http://strawsondesign.com/#!board-default

[5] Klein, D. (2012, August/September). Reinforcement Learning I. Retrieved June 13, 2018, from http://ai.berkeley.edu/lecture_videos.html

[6] Givan, B., & Parr, R. (n.d.). An Introduction to Markov Decision Processes. Retrieved June 13, 2018, from https://www.cs.rice.edu/~vardi/dag01/givan1.pdf

[7] Pandey, P., & Kumar, S., Dr. (2010, August). Reinforcement Learning by Comparing Immediate Reward. Retrieved June 13, 2018, from https://arxiv.org/ftp/arxiv/papers/1009/1009.2566.pdf

[8] Watkins, C. J., & Dayan, P. (1992). Q-learning. Retrieved June 13, 2018, from https://doi.org/10.1007/BF00992698

[9] Sutton, R. S., & Barto, A. G. (2005, January 4). Q-Learning: Off-Policy TD Control. Retrieved June 13, 2018, from http://incompleteideas.net/book/ebook/node65.html

[10] Salakhutdinov, R. (2018). 10703 Deep Reinforcement Learning and Control. Retrieved June 13, 2018, from http://www.cs.cmu.edu/~rsalakhu/10703/Lecture_Exploration.pdf

[11] Toussaint, M. (2008, July 5). Stochastic Optimal Control – part 2 discrete time, Markov Decision Processes, Reinforcement Learning. Retrieved June 13, 2018, from https://ipvs.informatik.uni-stuttgart.de/mlr/marc/08-optimal-control/slides-part2.pdf

[12] BeagleBone Blue. (2017). Retrieved June 13, 2018, from https://beagleboard.org/blue

[13] Hextronik HXT900 - 9g Micro Servo. (2009). Retrieved June 13, 2018, from https://servodatabase.com/servo/hextronik/hxt900

[14] LDPOWER M2204-2300KV Brushless Multicopter Motor (CCW). (n.d.). Retrieved June 13, 2018, from https://hobbyking.com/en_us/ldpower-mt2204-2300kv-brushless-multicopter-motor-ccw.html