Digital Improvement of Propeller Inspection Throughput

Undergraduate Student: Manasa Akella

Georgia Institute of Technology

Investigators: Dan Berrigan (AFRL), Andrew Gillman (AFRL), Surya R. Kalidindi (GT)
Student: Manasa Akella (GT Undergraduate)

April 1, 2021 - June 30, 2021

Project Objectives

The technical goals for this quarter included continued testing of the software to test it's functionality and optimize the algorithm as well as obtain data necessary for a publication as well as working on abstraction of the code to work with many different manufacturing setups. The main overall objective of this project is to create an optimized scheduling algorithm(s) to increase blade throughput for the Warner Robins AFB blade inspection process. We formulated our approach with the aim to study and optimize a real world manufacturing process - a penetrant dye quality control inspection process in the defense industry. The inspection is centered around a rigidly programmed robotic arm, which moves components through stations according to various precedence and time based rules. There is significant loss of productivity time and throughput due to frequent changes in the process requirements, leading to a cycle of constant reengineering and reprogramming of the robot arm.

Summary of Major Technical Accomplishments

- Determine hyperparameter settings for optimal learning and reward gain for random arrivals
- Track and measure behavior of RL algorithm over different parameters for random arrivals
- Analyze timing during testing to better understand the behavior of the model

Technical Discussion

Background

Manufacturing processes often require agility to adapt to ever changing demand and requirements. This presents a challenge if these processes must also adhere to strict time and spatial constraints. With the need for agile process control under manufacturing uncertainties, reinforcement learning can be a viable approach for implementing intelligent manufacturing systems. In this paper we present a method for implementing reinforcement learning frameworks on manufacturing systems with time based and precedence constraints in order to optimize throughput in a real world quality control process. We test

our approach on a simulation of a system in question as well as a small scale physical version of the manufacturing process to validate our results. The reinforcement learning problem typically consists of an agent and an environment, where at each time step, the environment is in a well-defined state, and the agent selects an action from a finite set of possible actions. The agent receives a reward for the action chosen in this state and the chosen action causes the environment to change to a new state. The objective of the problem is to maximize the long term expected reward by determining the optimal policy for choosing decisions based on the states experienced. If the desirable outcomes of the system are easily identifiable, but the system is too complex to directly determine an optimal solution, we can employ reinforcement learning to learn an optimal decision-making policy for the process. In

Methods, Assumptions and Procedures

The inspection process itself is very straightforward, it is centered around a Fanuc R-2000iC robot and follows the process flow outlined in figure 1. The current method allows for a 3 blade throughput per day which is 9 blades less than the eventual goal of a 8 blade daily throughput.

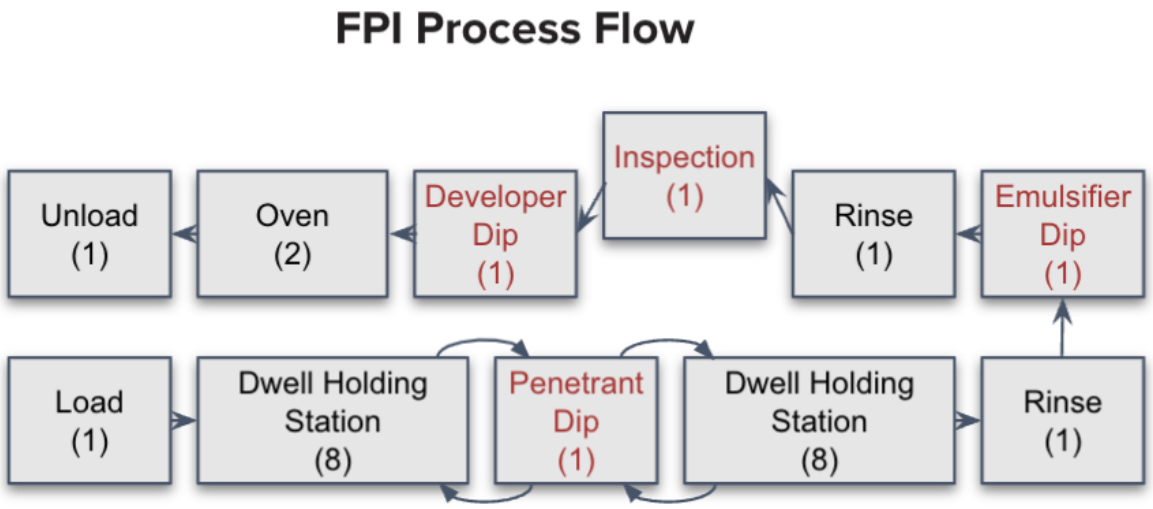

Figure 1: Inspection Process flow

The current program is rigid, as most robot programming is, but the input stream of blades is largely unpredictable. Therefore a maintainer has to choose between stopping the robot to reset for a new batch or waiting until the current batch is finished. This is what I mean when describing blades arriving in a batched fashion or a random fashion. When the blades arrive in a batched fashion the blades are all already in the queue when the process starts and in a random process the blades arrive whenever they arrive.

A type of RL known as Q-learning is being used for this project. This is used when all of the different states are not known.

In the past few weeks the focus has been on the reward signal and maximizing the reward gained by the algorithm. The reward signal defines the goal. On each time step, the environment sends a single number called the reward to the reinforcement learning agent. The agent's objective is to maximize the total reward that it receives over the long run. The reward signal is used to alter the policy. So in the case of the DIPIT algorithm the agent is given a positive 10 reward every time it dips the blades before their hour is up and 100 reward for every blade exiting the cycle. The agent is penalized with a negative reward when it takes the wrong action or an action that leads to a terminal state(invalid action) or breaks one of the timing or operational rules.

In order to increase the reward earned during a training episode, the algorithm needs to learn more efficiently. One of the first ways to approach that is by allowing the algorithm to see more states during the training process. By increasing the variability in the number of blades that are entered into the system, the overall number of unique states seen during the process is increased.

Results and Discussion

After determining the best exploration rate for the batched arrivals, it was decided to collect the same data for random arrivals.

The following data was collected to determine the best exploration rate for the RL Algorithm:

1) 5 Exponential exploration rate(s) to create graph representing x (training episodes)/y(reward) with various exploration approaches:

Constant rates: 0.1, 0.2, 0.3 (500 episodes) w/ 20 blades arriving randomly throughout the training processes

Variable Rate: a*b^x (where x is the episode number that is currently being trained, 1-500)

a = 0.7, b = 0.99
a = 0.8, b =0.99
a = 0.9, b = 0.99
a = 0.7, b = 0.95
a = 0.8, b =0.95
a = 0.9, b = 0.95
a = 0.7, b = 0.9
a = 0.8, b =0.9
a = 0.9, b = 0.9

This reward data can be seen in figures 2.1-2.3

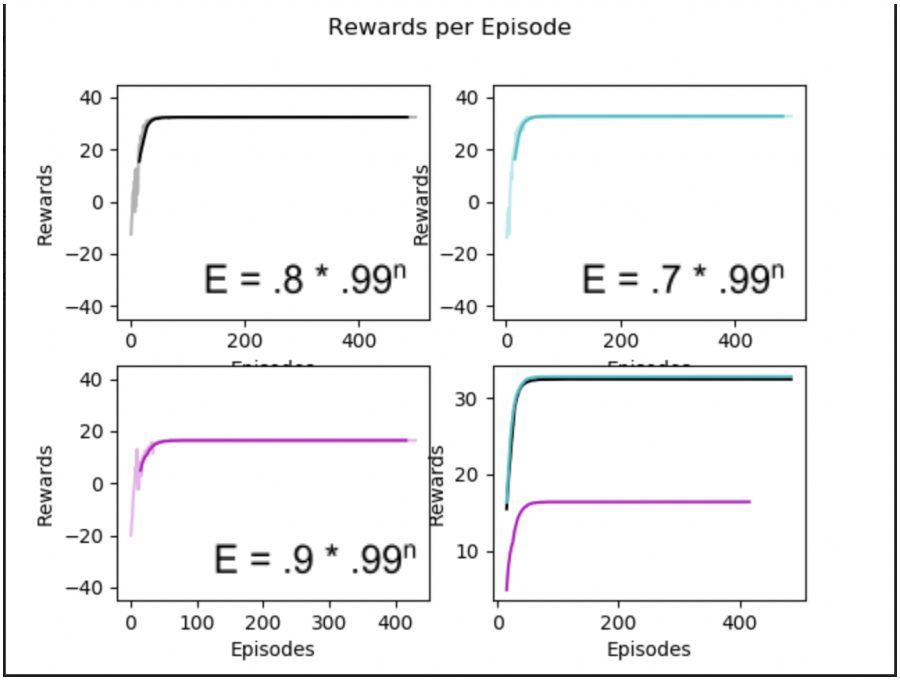

Figure 2.1: Rewards per episode for multiple variable exploration rates

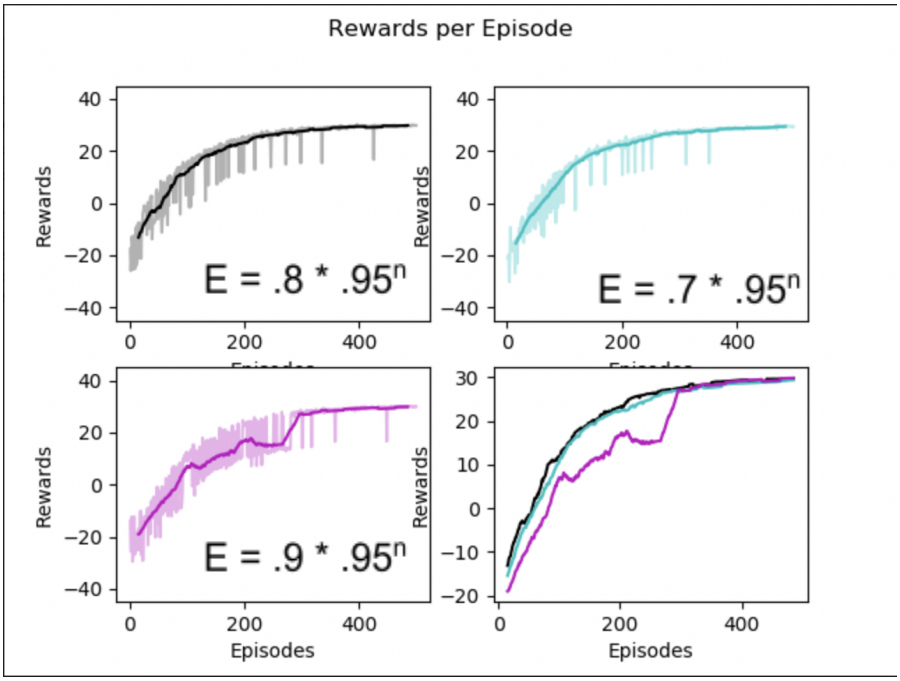

Figure 2.2: Rewards per episode for multiple variable exploration rates

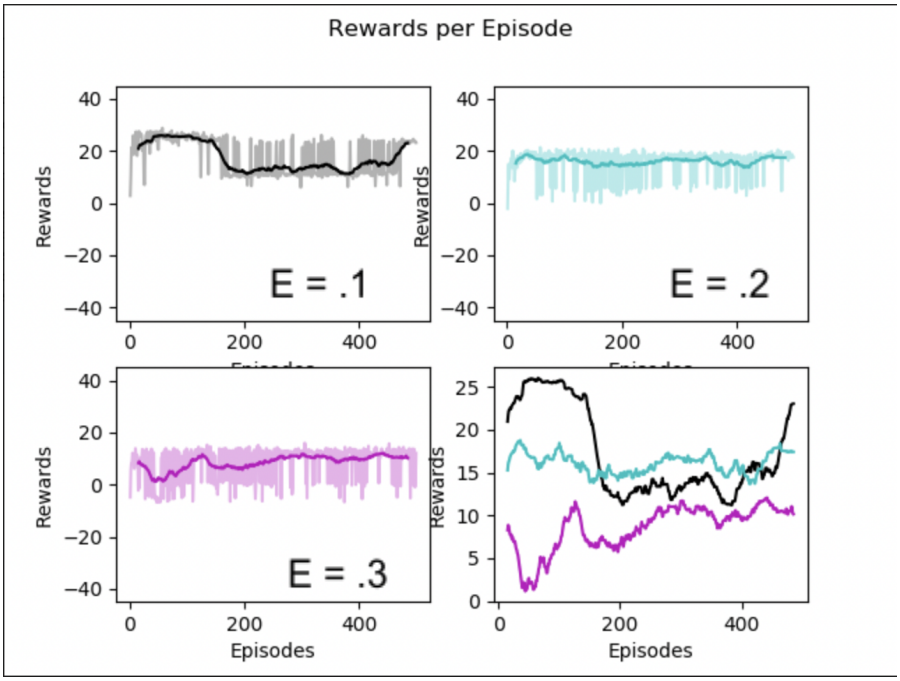

Figure 2.3: Rewards per episode for multiple variable exploration rates

As can be seen from these results the pink rewards graph seen in the figure 2.1 corresponding to the exploration rate $\varepsilon = .7 \times .99^{n}$, where n is the episode number that is being trained for, has the highest total reward and although it begins with some variable reward it levels out towards the last 100 episodes. Therefore it can be assumed that the exploration rate is the best that can be used. This is validated by figure 1.5 that plots the training method vs. % of successful policy tests out of 50 trials (1-50 blades).

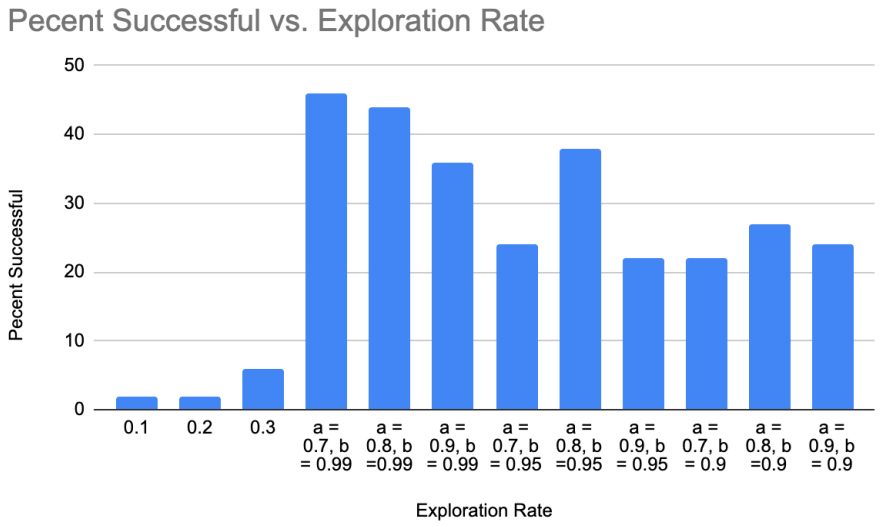

Figure 1.5: Percent Successful test for each of the policies generated with the corresponding exploration rate

Last quarter, the way that the RL model reacted to selecting a random number of blades to train on for every training episode was studied. The random number of blades would be selected between 1 and an upper bound that was specified by the researcher. The following ranges were trained for: 1 - 4, 1 - 8, 1 - 12, 1 - 16, 1 - 20. This was done to allow the model to see more state spaces and therefore generate a better policy. The results of that test can be seen in figure 2.1.

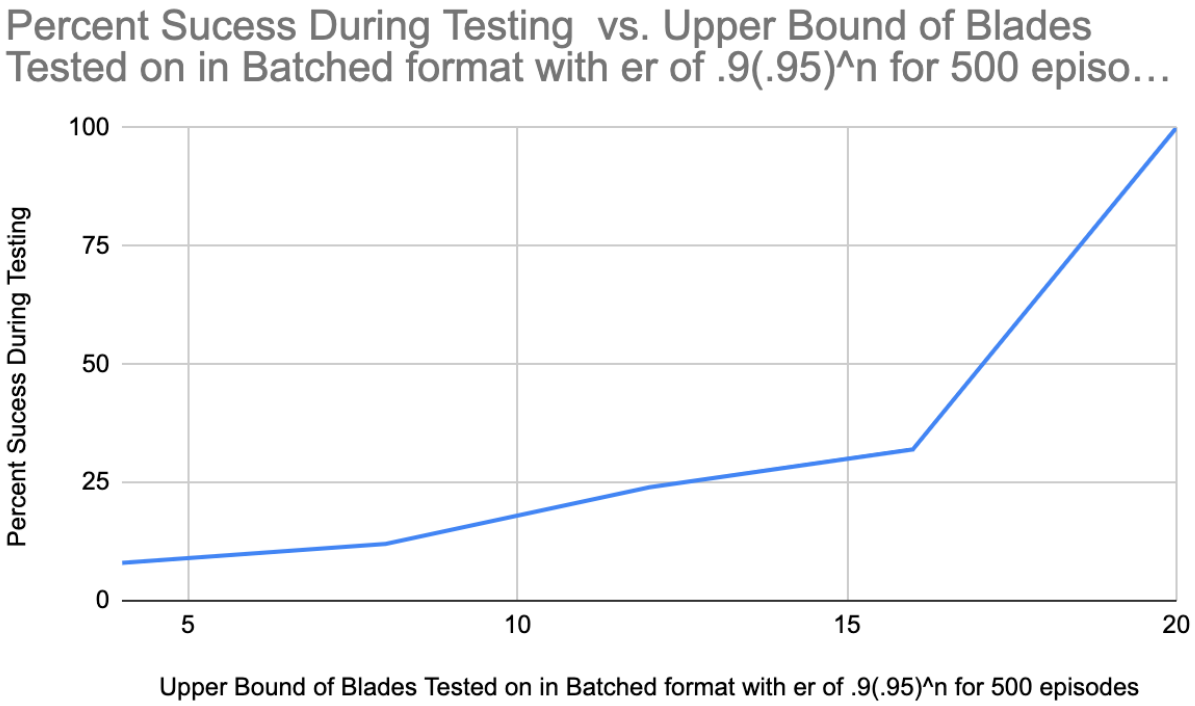

Figure 2.1 - Percent Success of Policy During Testing vs. Upper Bound of Blades Trained on in Batched format with er of .9(.95)^n for 500 episodes

It can be seen that the policy generated ranges between 8-32% success rate when trained for 4, 8, 12, and 16 blades set as the upper bound. However, seeing that the success rate jumps quite high between 16 and 20 (from 32% to 100%) it was determined that one of the next steps should be to test the ranges between 16 and 20 to see where the jump occurs in order to better understand the model's behavior. Policies where the upper bound was set to 17, 18, and 19 were also trained and tested on and added to the data. These results can be seen in figure 1.2.

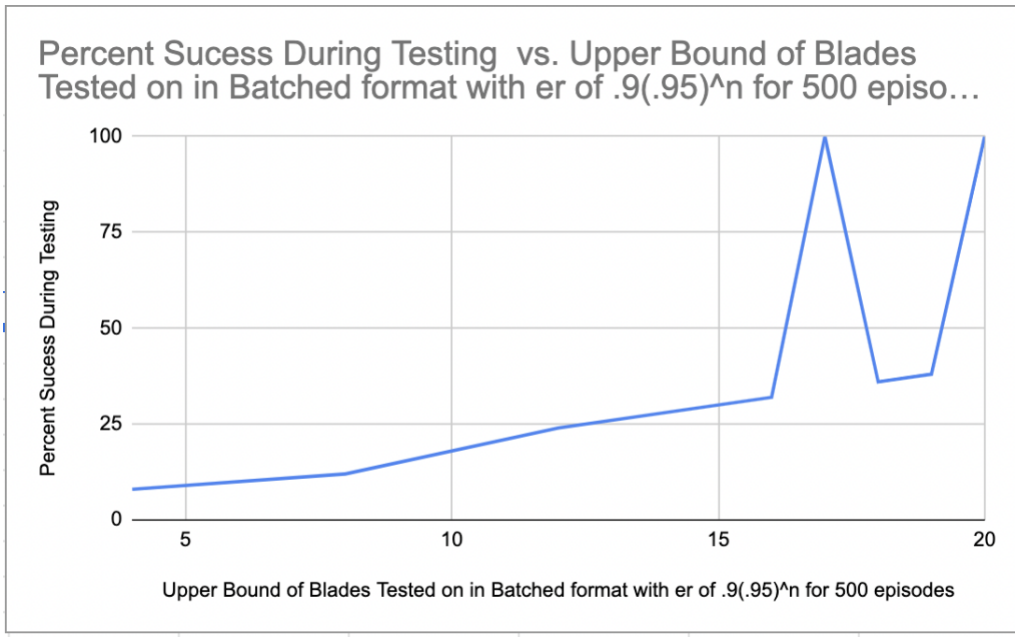

Figure 2.2: Percent Success During Testing vs. Upper Bound of Blades Tested on in Batched format with er of .9(.95)^n for 500 episodes (including blades 17-19)

These results were not what was expected. It was assumed that we would see another large jump or a gradual rise to 100% success. However, the fact that 17 blades had a 100% success rate but not 18 or 19 prompted further data collection into the timing during testing to better understand the model behavior.

For the same data that is seen in figure 2.2, the times that it took for it to process the number of blades were collected and graphed. These results are seen in figure 3.1.

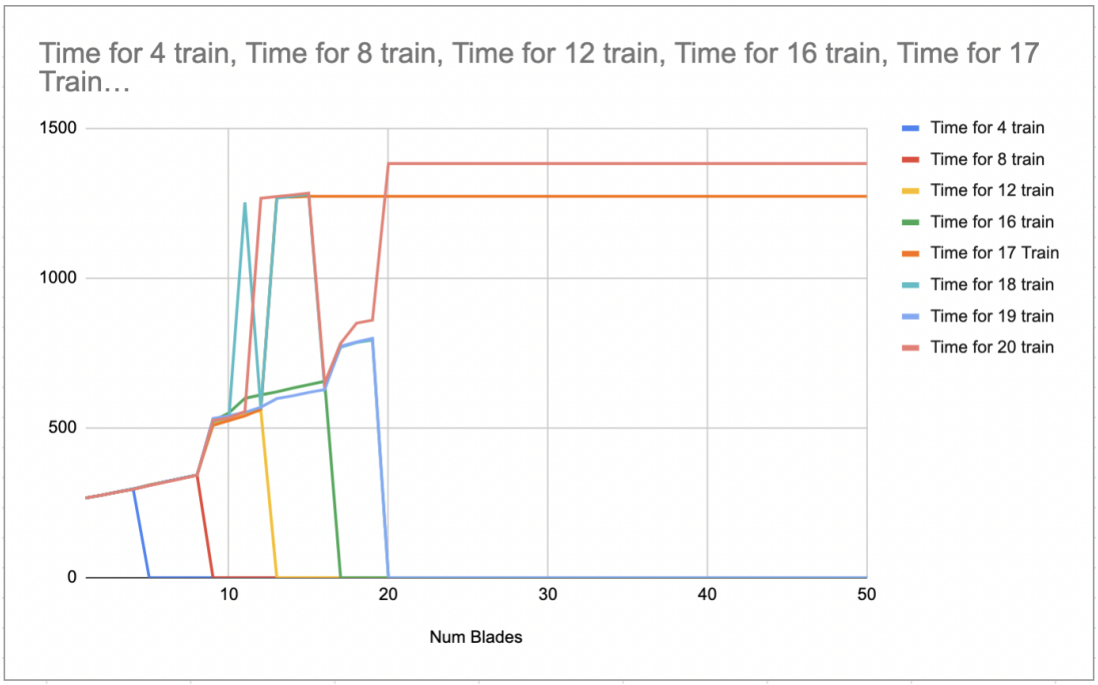

Figure 3.1: Time to test on 4,8, 12, 16, 17, 18, 19, and 20 blades

These results show some concerning trends as the time, which should be increasing in a fashion similar to the trend seen in figure 3.2, actually seems surprisingly high in the 12-15 blade range for a couple of the trainings and then eventually levels out to a constant value for the two trainings that were deemed 100% successful.

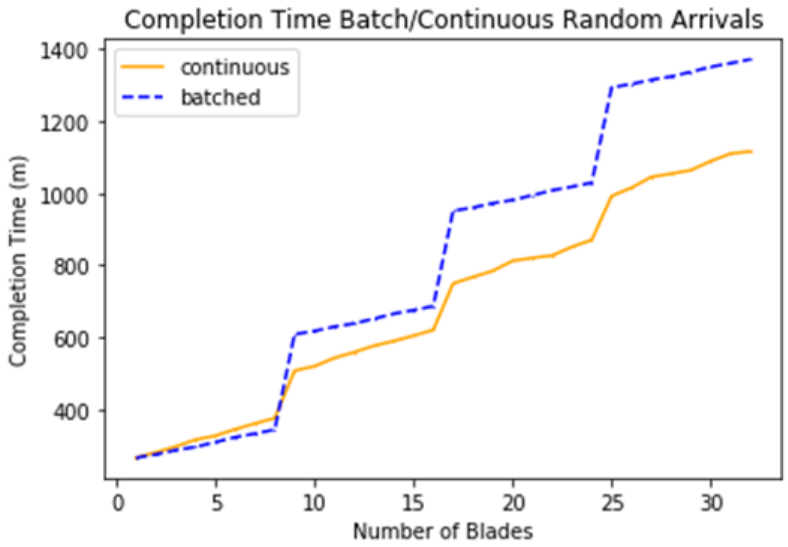

Figure 3.2: Trend that should be seen in timing charts

Seeing the times level off at a certain number led to the suspicion that these testing runs were never being completed as 50 blades should take much longer than 30 blades to process. Seeing this trend, the actions taken during these testing runs were further examined to see if there was an explanation for the trend. Prior to these findings, a successful test was deemed to be a test that did not get stuck in an endless loop of negative reward. However, all the tests for the 17 blades policy and 20 blade policy were seen as successful but led to the same elapsed time. At this time, the method of determining a successful test was changed to be determined by actions. If the robot unloaded all of the blades that were being inputted it would save an action denoted by the number -1 and the test would be labeled a successful test. After re-examining the tests for 17 and 20 blades using this new method of determining a successful test, it was deemed that all of the tests that resulted in the “max” elapsed time were really getting stuck in a loop of positive reward and were not ever completing the process. This issue was never realized in the prior method of determining a successful test because the reward that was being received was still positive because the robot was taking valid actions based on the policy that was generated; however it was stuck in a loop and would never exit the process.

This new finding has changed the standard for how data are collected for any figures made regarding this project. Many tests and figures in the past have used the faulty method of determining successful tests and have resulted in unreliable data, including data presented previously in this report and prior reports.

Conclusions and Recommendations

- Hyperparameter tuning greatly affected the policy functionality and it has been determined that $\varepsilon = .9 \times .99^{i}$ is the best exploration rate to use for random arrivals
- Using an upper bound/ range for the number of blades trained on can be highly effective in increasing the number of state spaces that the algorithm sees and therefore the effectiveness of the policy, however issues still persist in determining the true successfulness of a policy
- Any charts and figures that were previously made using the incorrect method for determining a test's success should be assessed and corrected using the new method of determining success

Anticipated Technical Tasks for Next Quarter

- Re-assessing the all prior charts and figures using the new method of determining success
- Determining and fixing the issue that is causing the model to get stuck in an infinite loop of positive reward
- Further analysis of algorithm and policy functionality given different initial circumstances