Colorado School of Mines Mechanical Engineering

Advanced Robot Controls - MEGN545

Project Part 3

Authors:
Aarushi Doctor - 10913401
Nick Taylor - 10920730

Class Personnel:
Dr. X. Zhang

May, 2024

Contents

1 Simulation Video Link **1**

2 Introduction & Problem Formulation **1**

3 Design Process **1**
3.1 Baseline Controller Design 1
3.1.1 Q-Learning Process 2
3.1.2 Reward Function Design 2

4 Results & Discussion **3**
4.1 Training 3
4.2 After Training 6

5 Controller Comparisons **7**

A Appendix **8**

1 Simulation Video Link

Please view the following URL link to observe the simulation video:
https://youtu.be/dPVgsGoF8eE?feature=shared

2 Introduction & Problem Formulation

The objective of this project is to design and implement a Q-learning RL agent capable of controlling a quadrotor to track a square trajectory. To guide the agent's learning process, we designed and tuned reward functions that penalize deviations from the desired trajectory and orientation referencing our defined quadrotor state variable. We also implemented a functionality to balance exploration (random actions) and exploitation (using learned policies) during training to gather sufficient experience for updating the Q-function weights. By addressing these aspects, we developed a robust Q-learning RL agent capable of autonomously controlling a quadrotor to track a predefined trajectory in 3D space. This report delves into the methodology, design considerations, and implementation details of the Q-learning RL agent for efficient quadrotor control in completing the prescribed trajectory.

3 Design Process

3.1 Baseline Controller Design

We started designing our controller taking motivation from the class project assignment document. The image below shows the final controller structure of our Simulink model.

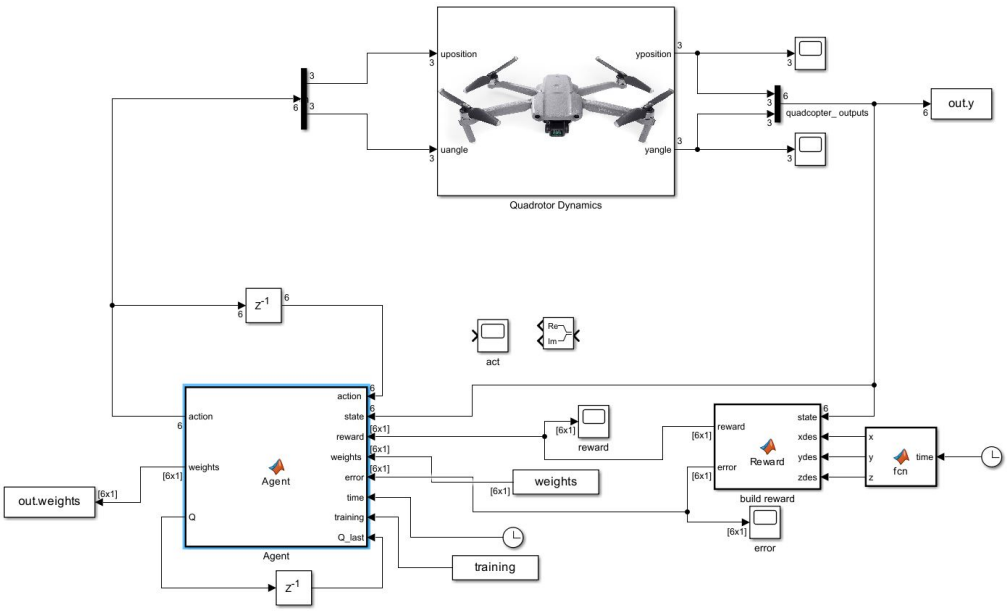


Figure 1: Controller Simulink Model

3.1.1 Q-Learning Process

For this project, we opted to follow option 2 (the approximate Q-learning method). The following equation represents the linear summed function we used to predict future Q values. The state vector used for this project and the action space discretization for each state variable are both defined as well.

$$Q(s,a) = w_x f_x(s,a) + w_y f_y(s,a) + w_z f_z(s,a) + w_\phi f_\phi(s,a) + w_\theta f_\theta(s,a) + w_\psi f_\psi(s,a)$$

$$s = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \qquad a = \begin{bmatrix} -0.0627 \\ -0.03 \\ 0 \\ 0.3 \\ 0.0627 \end{bmatrix} \tag{1}$$

We began the Q-learning process by introducing perturbations to the quadrotor trajectory during training iterations by generating a noise vector of size 6x1, filled with random numbers between -0.06 and 0.06. This noise vector represents the exploration versus exploitation feature within the model, which was summed with the state vector to introduce perturbations to the quadrotor performance. The next thing we did was define the activation function structures, $f_i(s,a)$.

$$f_i(s,a) = \rho s_i + e_i + a_i$$

It's important to note that the error term used in this activation function is an altered version of the traditional or canonical error. This will be explained in the following subsection. Our final tuned ρ value was 4. From this approximate Q-learning method definition, we then ran the Q values through the following algorithm to optimize the Q values, as discussed in class. From this algorithm, the action value for each state variable is determined by choosing whichever action variable within the discretized action space maximizes $Q(s,a)$.

$$\text{diff} = (R + \gamma \max(Q(s',a'))) - Q(s,a), \quad Q(s,a) + \alpha * \text{diff}, \quad w_i = w_i + \alpha * \text{diff} * f_i(s,a)$$

3.1.2 Reward Function Design

We started building our reward function exactly how the assignment document suggests. After lots of trial and error, we came up with this reward definition after much time and effort spent tuning our controller.

$$R_{position} = \begin{bmatrix} -500(x - x_{refer})^2 \\ -500(y - y_{refer})^2 \\ 1000(z - z_{refer})^2 \end{bmatrix} \qquad R_{angle} = \begin{bmatrix} -300(\phi)^2 \\ -300(\theta)^2 \\ -300(\psi)^2 \end{bmatrix}$$

The following vector states the expression we used to define our error vector within the Q-learning process, which is later used to build the activation function, $f_i(s, a)$.

$$e = \begin{bmatrix} 3000(x - x_{refer}) \\ -12000(y - y_{refer}) \\ -1000(z - z_{refer}) \\ -300(\phi) \\ -300(\theta) \\ -300(\psi) \end{bmatrix}$$

All state variables are all functions of time, so these errors are dynamic in time. The coefficients used within this expression arose from a tedious process of trial and error tuning. This trial and error process was motivated by an observation concerning the math of the Q-learning process described in the previous subsection. The error term defined above informs the activation function, $f_i(s, a)$. This information helps update $Q(s, a)$, but it's essentially lost after it informs Q for one given iteration. In that way, it's current disposition is only used once and then no longer preserved in the Q-learning update process. However, the reward term informs the weight vector, which also helps update Q. The difference between the weight vector w and the activation function $f_i(s, a)$ is that weights from previous iterations are used to update Q-values of future iterations. Whereas, activation functions of previous iterations are not used to update future activation functions in future iterations. Weight coefficient contributions are preserved in the Q-learning process, but activation functions are not. We used this inherent feature of the Q-learning process to motivate our reward and error tuning process. The initial weight vector the training process starts with is also shown below.

$$\vec{w} = [0.001, 0.001, 0.001, 0.001, 0.001, 0.001]$$

4 Results & Discussion

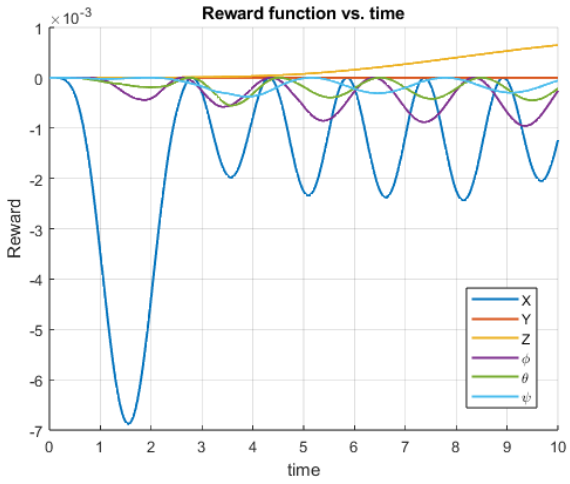
4.1 Training

The following 3 plots within the subfigure shows the reward functions of all 6 state variables as a function of time. Subfigure (a) corresponds to the first training iteration, subfigure (b) corresponds to the fifth training iteration, and subfigure (c) corresponds to the tenth training iteration. These results show there is a very large response to the x state variable early on in the training process. After that, the rewards of each state variable all shrink each iteration, but at less significant amounts. For example, the maximum ϕ reward magnitude shrinks by only 0.0001 from the fifth training iteration to the tenth iteration.

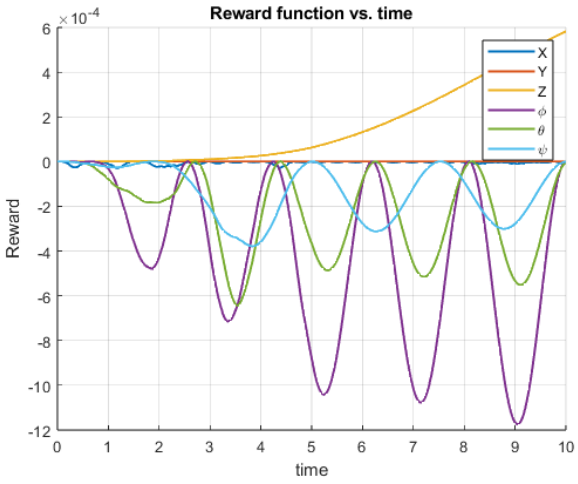
This response indicates how well the controller reward and error functions are tuned with respect to the initial weight values. Subfigure (a) shows the reward for each state variable throughout time for the very first training iteration, where the y scale is plotted at 10^{-3}. Given the aggressive reward functions used in our project (stated in section 3 above), this is a very small reward response. The reward is

calculated like a modified expression of error, except the reward in our project is larger than the actual error. So a response on the order of 10^{-3} for the first training iteration expresses the accuracy of the controller tuning. Keep in mind that the superimposed noise added to the state variable within the training process has a magnitude of 0.06, which is fairly large for a time step 0.025 seconds. We attribute the small response to training iterations 5 through 10 to the high accuracy of the reward and error function tuning. The optimized reward and error functions take lots of weight off of the weight updating process, in turn producing an accurate control system early on in training not allowing for much improvement in later training iterations.

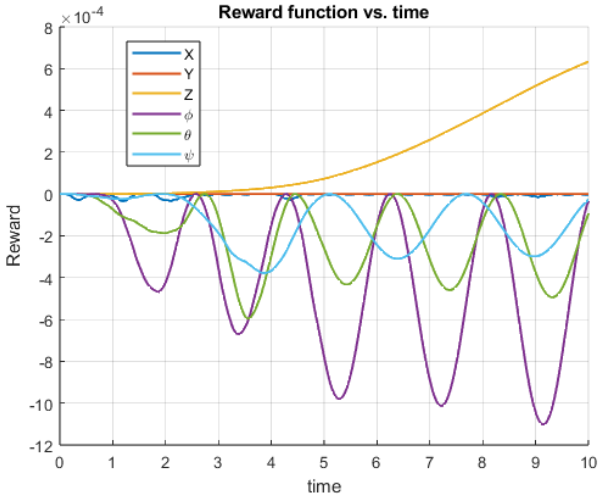
For reference, the controller performance with respect to the z direction was not as important as the x and y directions during our tuning process. The results within this report show that the controller response in the z direction is stable, but not optimized. We admit this is because we placed higher importance figuring out the x and y directions instead. The z direction reward looks large and unstable in these figures, but it's untuned reward and error function combination simply resulted in a natural frequency response that's larger than the training iteration time-length which these reward plots are plotted against. Results from section 4.2 below indicate the controller stability in this z direction.



(a) Reward from early training stage



(b) Reward from middle training stage



(c) Reward from late training stage

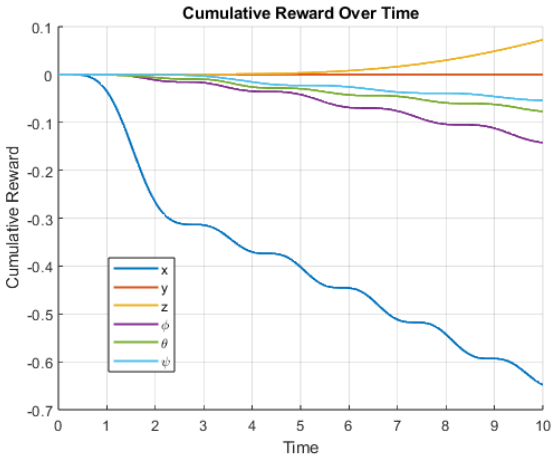
The running reward averages for training iterations 1 and 10 are plotted below. For a given state variable, it's running reward average in time was computed using the following expression.

$$R_{AVG,i}(t') = \Sigma_{t_i=0}^{t_i=t'} R_i(t')$$

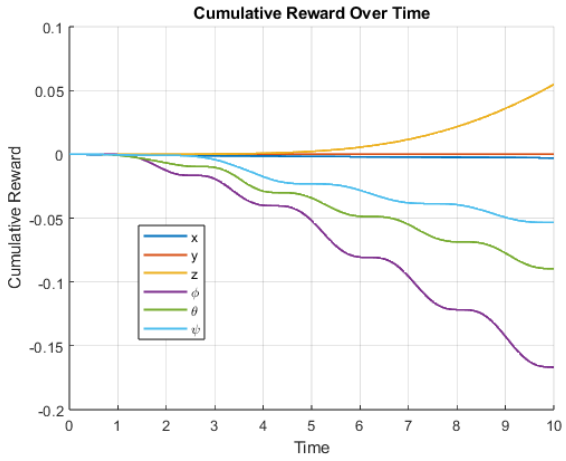
This data helps provide an intuition for the fluidity of the controller response and performance. The individual rewards for each variable form a stepped pattern in time. This is an indication that the quadrotor response is oscillating in those directions.

This data also indicates the effectiveness of the training process, arguably better than the reward functions versus time in the subfigure above. Inspection of the reward running average for the x direction shows a very large response in the first training iteration, but by the tenth training iteration, the reward running average is virtually zero, even at the end of the iteration simulation. This is an indication of ideal controller performance because reward is an expression of error. Smaller reward magnitudes indicate smaller error magnitudes, so when the reward running average

starts out at a larger number after the first training iteration, but vanishes by the end of the training process, this implies the training process is learning to optimize the quadrotor dynamic performance. This characteristic is most visibly evident in the x variable because of it's large reward running average response during the first training iteration, but all 6 state variables exhibit this performance characteristic. We know this because of the noticeable shrink in the y axis between the two plots, representing the first and tenth training iterations respectively.



(a) Reward running average from training iteration 1



(b) Reward running average from training iteration 10

4.2 After Training

Table 1 below shows how the controller performance varies as Q-learning parameters are varied. Even though the quadrotor performs extremely well for us, the implications of this data are disappointing. Table 1 shows that as α and γ are increased, the RMSE of the quadrotor performance also increases. Higher α and γ mean that the Q-learning process has more of influence. Higher errors in response to higher α and γ imply our Q-learning process doesn't always optimize weight coefficients. This data shows more Q-learning influence on our controller actually harms the controller performance. We believe this is a result of the Q-learning update process. It's also important to note that this data indicates our Q-learning process has a higher sensitivity to α than to γ. Based on the setup of our controller, that means that the error function input is more computationally important than the reward function input to the Q-learning input.

Table 2 below quantifies quadrotor performance for different weight coefficients. Rows 2 and 3 shows weight coefficients after 6 and 10 training iterations respectively. Row 4 is a random weight coefficient combination. This data is important because it indicates our controller is optimized when weight coefficients are initialized at 0.001 respectively. This data confirms that Q-learning influence on our controller harms the controller performance. We know this because the quadrotor performance gets worse after multiple training iterations, when it should get better instead. We

α	γ	E_x	E_y	E_z
0.001	0.001	0.0050	0.0038	0.0005
0.1	0.001	0.0081	0.0070	0.0005
0.001	0.1	0.0053	0.0042	0.0005
0.1	0.1	0.0226	0.0091	0.0007

Table 1: Q-learning Parameter Sensitivity

attribute this phenomena to a combination of two factors. First, the reward and error functions within the controller design are aggressively for the case when all weights are initialized to 0.001, instead of a dynamic value that arises from the Q-learning process. The other contributor to this issue is most likely the method in which our Q-learning process has been implemented.

w_x	w_y	w_z	w_ϕ	w_θ	w_ψ	E_x	E_y	E_z
0.001	0.001	0.001	0.001	0.001	0.001	0.0050	0.0038	0.0005
0.0181	0.0008	0.0008	0.0011	0.0011	0.0011	0.0206	0.0070	0.0005
0.0173	0.0008	0.0009	0.0011	0.0011	0.0011	0.0092	0.0072	0.0005
0.15	0.4	0.001	0.001	0.001	0.001	4.8142e6	2.6135e6	6.6784e6

Table 2: Q-learning Weight Coefficient Sensitivity

5 Controller Comparisons

The table below presents the best RMSE errors produced by each of the 3 controllers for the results we were able to obtain this semester. We used the row 3 data because it represents the RL agent controller performance AFTER Q-learning interactions, instead of before.

Controller	E_x	E_y	E_z
NN	0.1571	0.1703	4.8269e-13
MPC	0.2099	0.3012	0.0450
Q-L RL	0.0053	0.0042	0.0005

Table 3: Controller Comparison Results (via position RMSE)

Considering the root-mean-square error in the z direction for all 3 controllers, all controllers performed well in this dimension. However, the desired trajectory was a simple square in the x-y plane, keeping the desired z-reference value fixed at 1.5. For this reason, less significance is placed on the error results corresponding to the z direction, because the trajectory in the z direction was more trivial than the other two dimensions. Therefore, we can conclude that our Q-Learning RL controller still significantly out-performs the other 2 controllers, even though it exhibits performance flaws.

A Appendix

The purpose of this appendix section is to include figures which we didn't have space for in the body of the report.

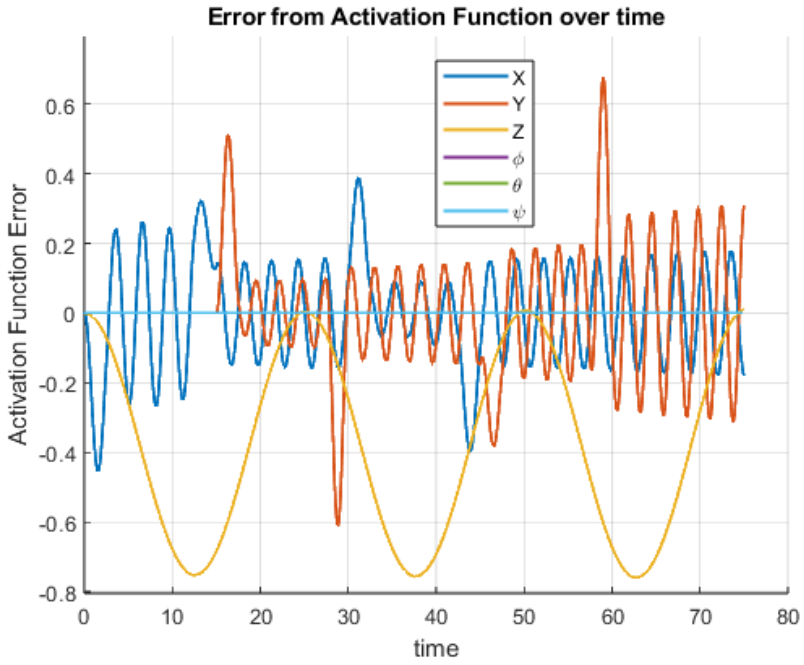


Figure 4: BEFORE TRAINING

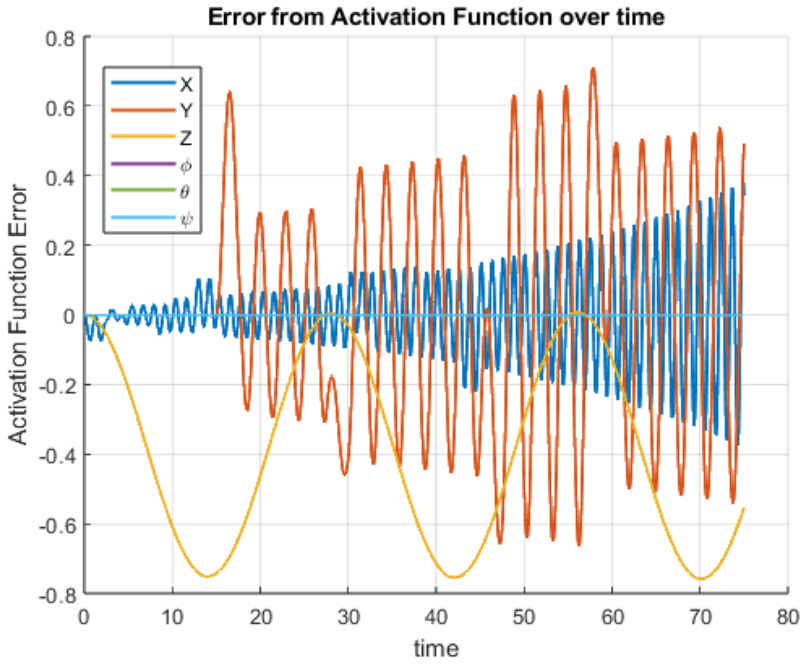


Figure 5: AFTER TRAINING

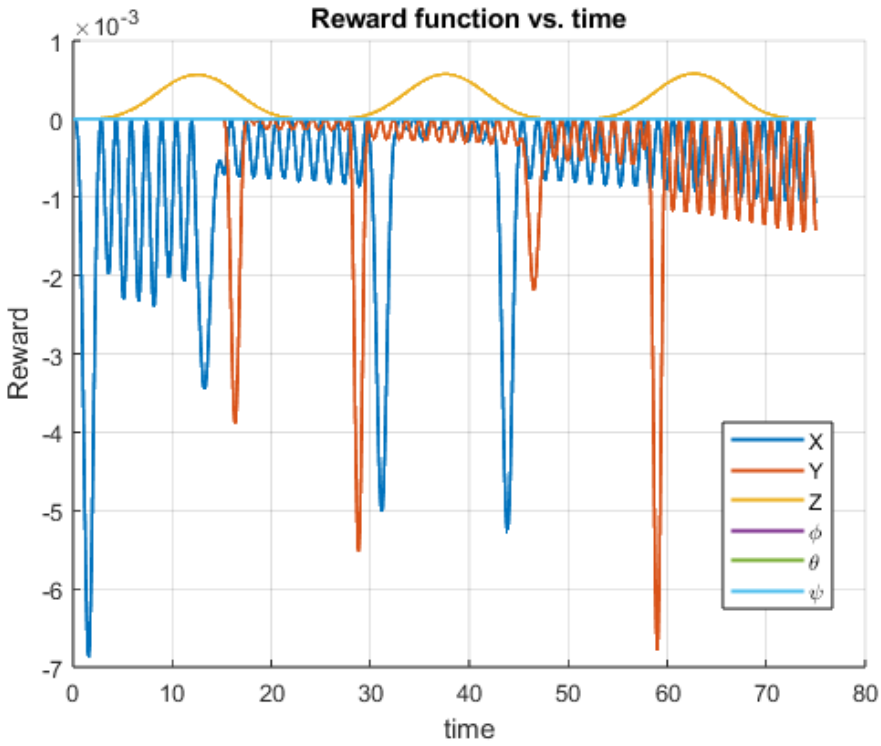


Figure 6: BEFORE TRAINING

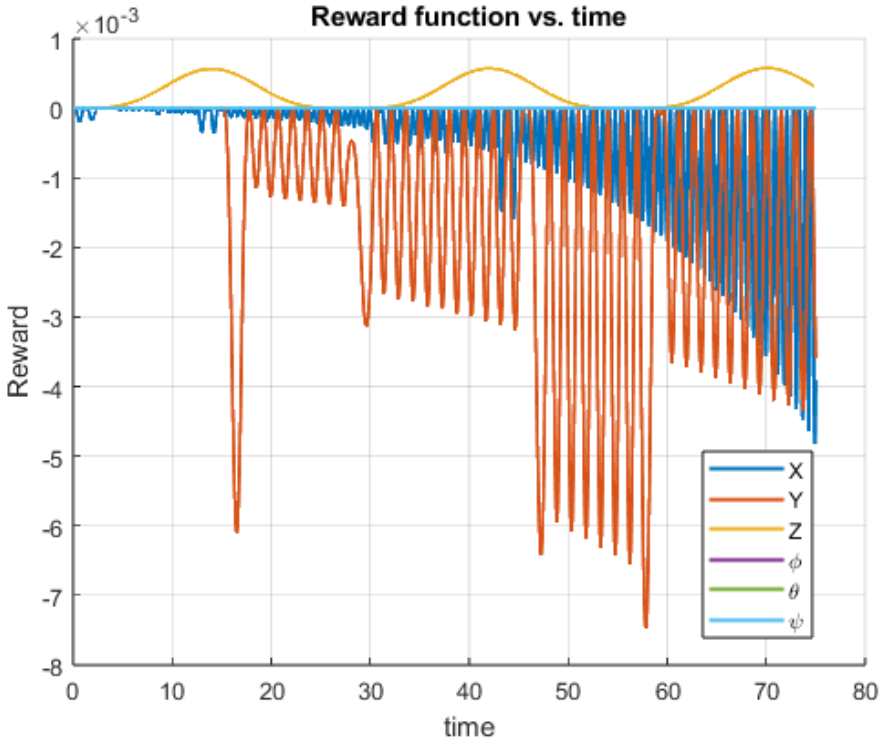


Figure 7: AFTER TRAINING

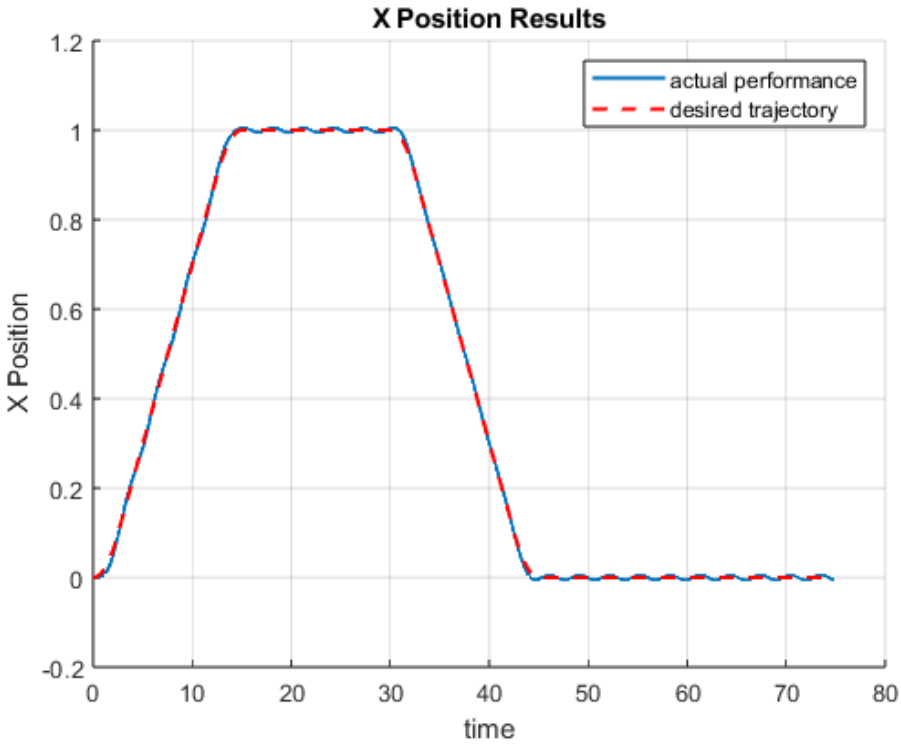


Figure 8: BEFORE TRAINING

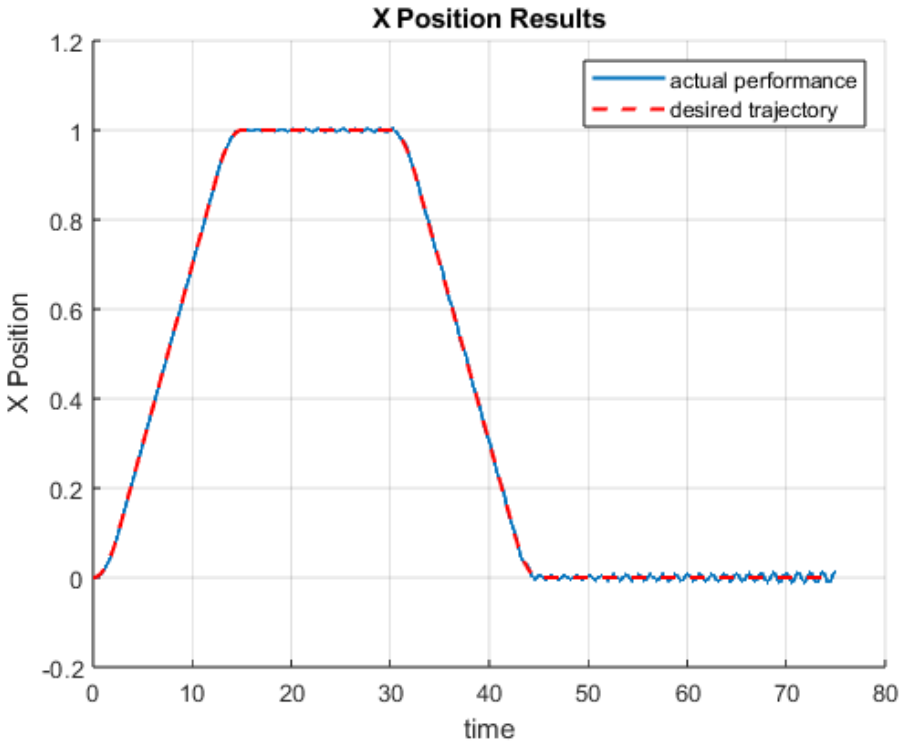


Figure 9: AFTER TRAINING

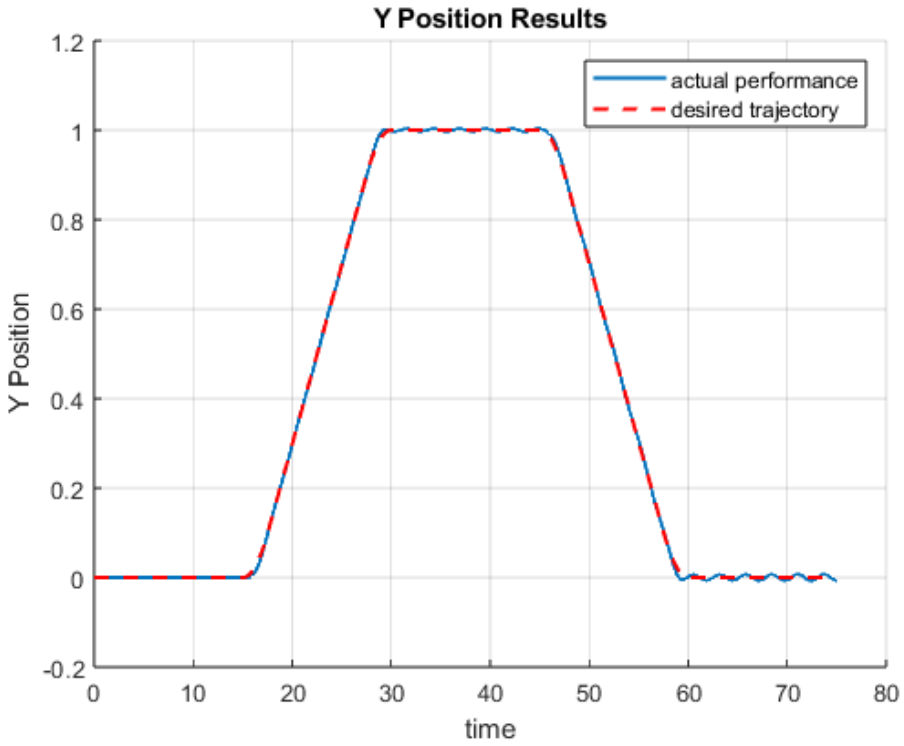


Figure 10: BEFORE TRAINING

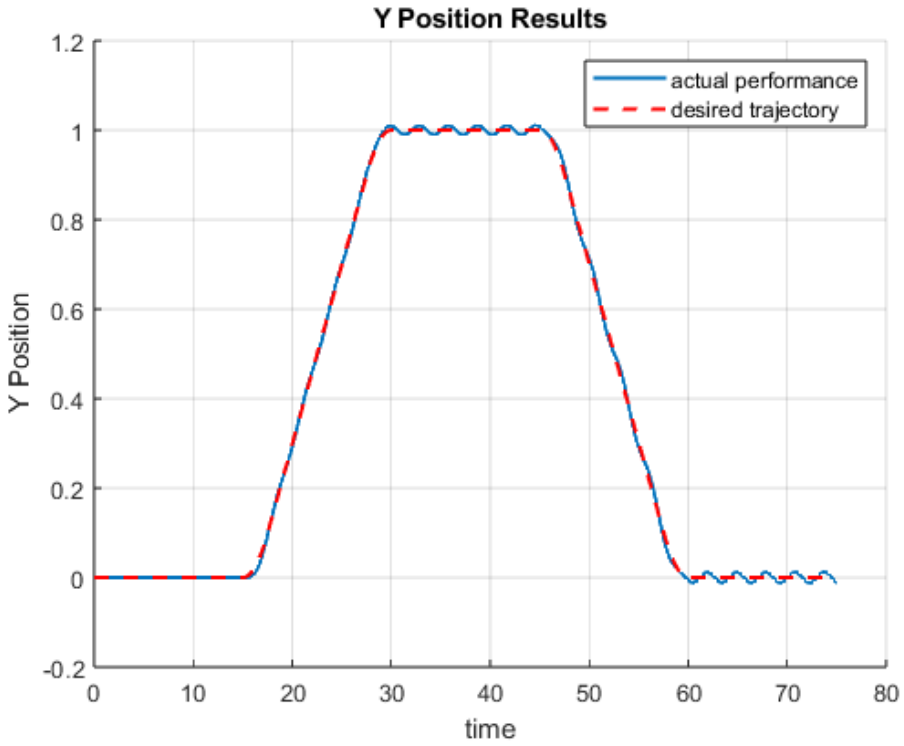


Figure 11: AFTER TRAINING

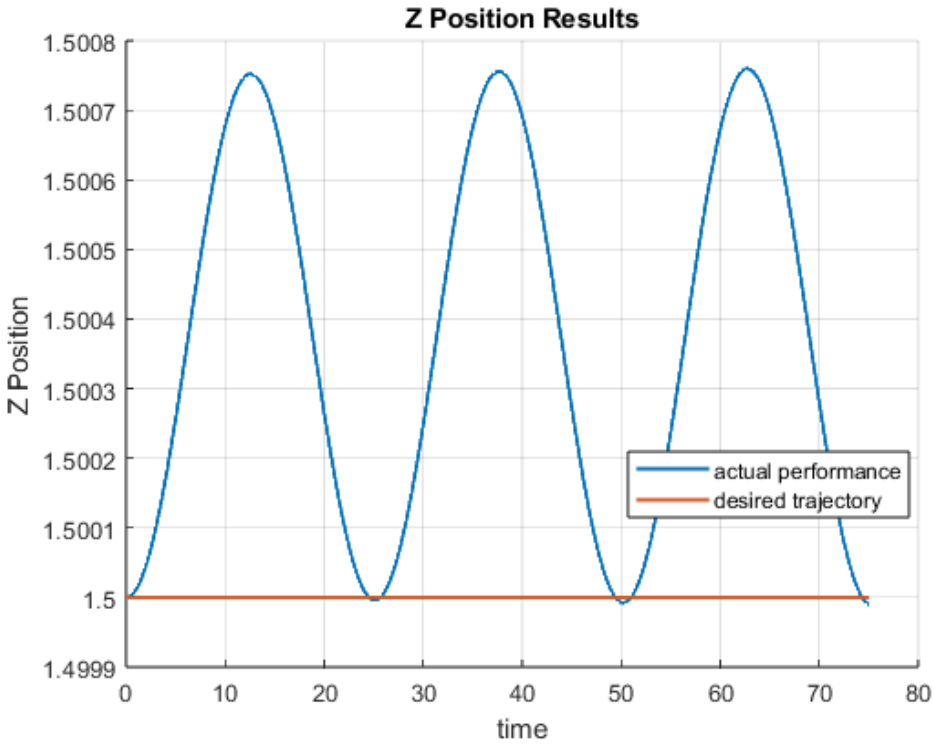


Figure 12: BEFORE TRAINING

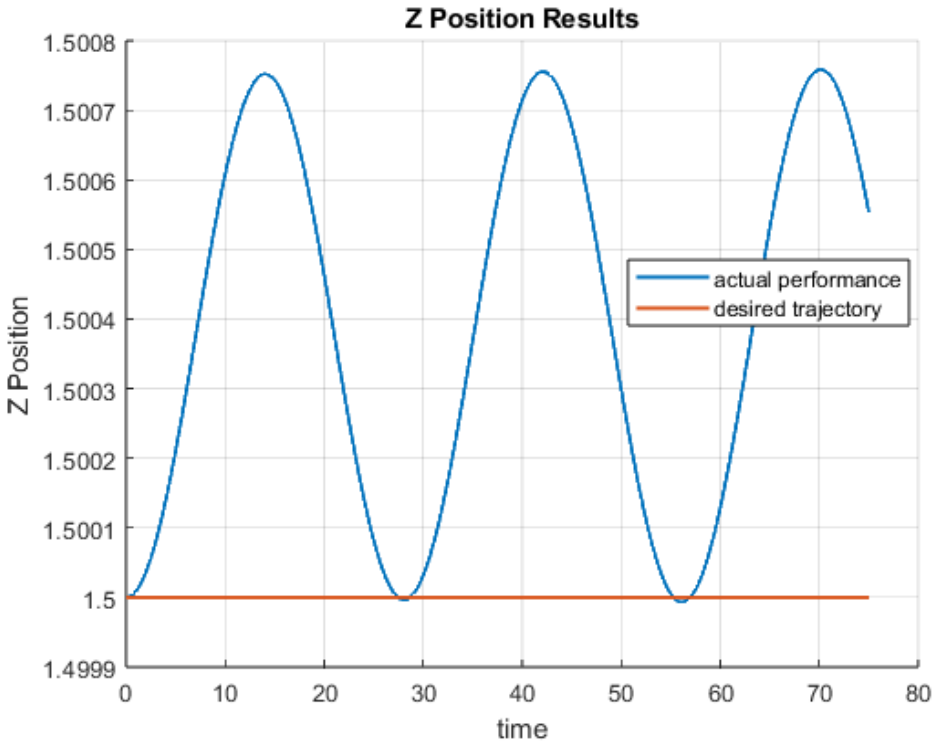


Figure 13: AFTER TRAINING

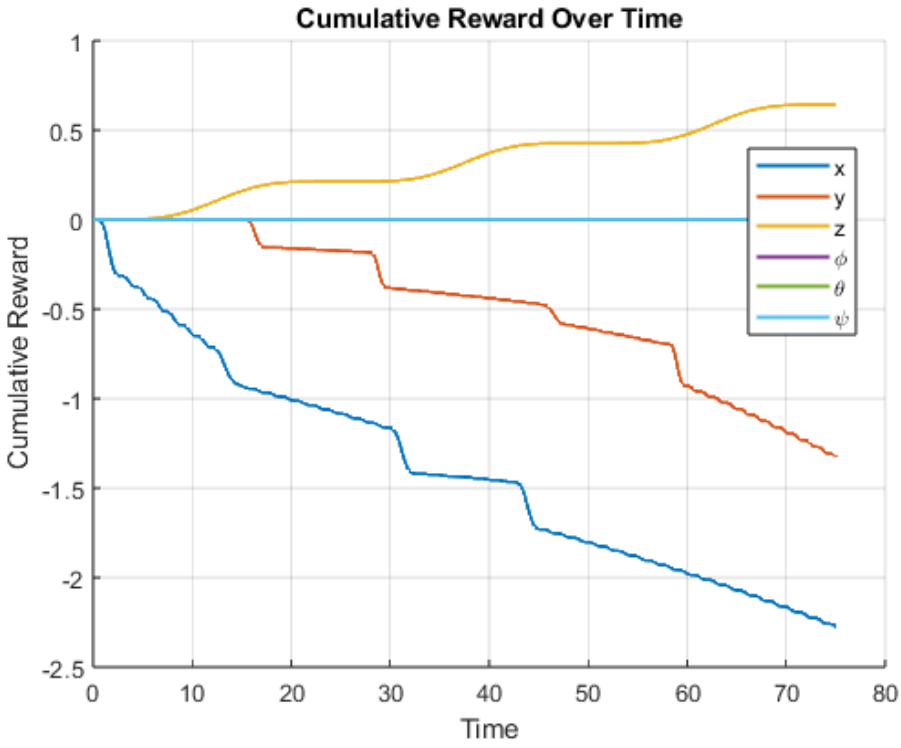


Figure 14: BEFORE TRAINING

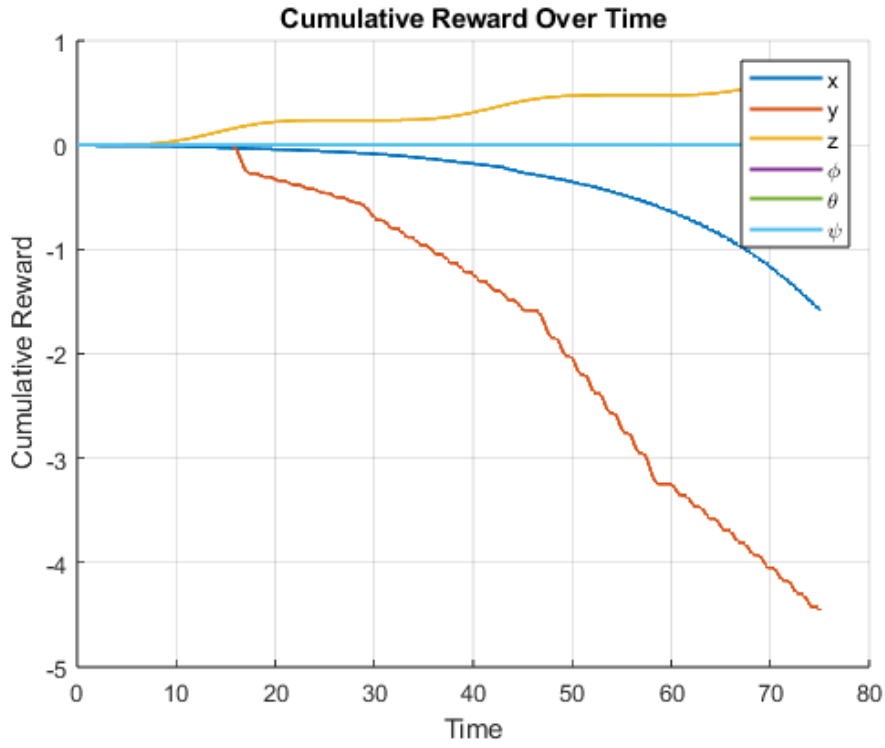


Figure 15: AFTER TRAINING