Using Q-learning to develop a Pitch Angle controller for fixed wing aircraft

Yash Manian
ymanian@umd.edu
Robot Learning
University of Maryland, College Park

December 12, 2016

Abstract

In this paper, Q-learning is applied to the control of the pitch angle on a fixed wing aircraft. The change in elevator angle is learned after a number of iterations and is able to effectively track the reference signal for the pitch angle. Later, comparisons to an LQG controller are made

1 Introduction

Modern aircraft are complicated systems with many non-linear aspects to their dynamics. Standard controllers like the PID controller come up short when trying to control the various modes of operation. This is without considering the obvious measurement and process noise present in the aircraft. Secondly, the performance of any control system is improved through adding more sensors and actuators. Hence, the control system should be able to handle multiple sensors and actuators at once. The Linear Quadratic Gaussian controller (LQG) is one the more modern approaches to the problem. This controller is made to be optimal with respect to a quadratic cost function. However, the design effort taken to get a decent convergence is drastically reduced when a learning algorithm is implemented. This allows the aircraft to learn the dynamics of the system through datasets of inputs and outputs.

One example of such a learning is the Stanford autonomous helicopter developed by Andrew Yan-Tak Ng from Stanford. It uses machine learning to learn the helicopter dynamics and generate control signals to have the helicopter follow complex trajectories. The data sets are created by recording the control inputs from expert helicopter fliers to learn complex maneuvers. This is an example of imitation learning. This project goes for Q-learning and uses an simulation environment

Figure 1: The Stanford Autonomous Helicopter.

which is a set of dynamic equations describing the longitudinal motion of an aircraft. The learning algorithm learns to use various control signals to converge on to the reference. The control signals are sent to the non-linear simulation environment to generate the states of the system. These are fed back to the learning algorithm which rewards the state-action pair.

Reinforcement learning is very flexible in that it can expand easily to accommodate any number of states and actions. Q learning is used as it is a model free learning method. This reduces the complexity of learning and also allows control of the learning rate.

2 Simulation environment Equations

Aircraft dynamics can typically be described by six differential equations. There are three kinds of control surfaces on most fixed wing aircraft i.e. *Ailerons*, *Elevators* and *Rudders*. Deflection in the *Ailerons* gives rise to change in the *Roll* angle. Deflections in the elevator controls the *Pitch* of the aircraft, and the Rudders control the *Yaw*. The simulation environment makes use of these equations to generate error states with respect to the reference.

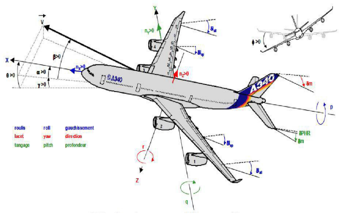


Figure 2: Aircraft dynamics nomenclature reference.

2.1 Velocity

The velocity can be described by the following equation:

$$\vec{v} = u\vec{i} + v\vec{j} + w\vec{k} \tag{1}$$

Where u,v and w are components of the velocity vector $\vec{v}$.

2.2 Forces

The forces acting on the aircraft can be described by the following equation:

$$\vec{F} = x\vec{i} + y\vec{j} + z\vec{k} \tag{2}$$

2.3 Moments

The moments acting on the aircraft can be described by the following equation:

$$\vec{T} = l\vec{i} + m\vec{j} + n\vec{k} \tag{3}$$

Where,
l is the Axial moment
m is the transverse moment
n is the Normal moment

2.4 Angular Velocities

The angular velocities can be described by the following equation [1]:

$$\vec{\omega} = p\vec{i} + q\vec{j} + r\vec{k} \tag{4}$$

Where,
p is the Roll rate

q is the Pitch rate
r is the Yaw rate

2.5 Longitudinal Dynamics

Aircraft longitudinal dynamics assume,

$$\beta = p = r = \phi = 0 \tag{5}$$

Hence, Longitudinal equations can be written as follows:

$$\dot{u} = \frac{X_u}{m}u + \frac{X_w}{m}w - \frac{g\cos\theta_0}{m}\theta + \Delta X^c \tag{6}$$

$$\dot{w} = \frac{Z_u}{m - Z_{\dot{w}}}u + \frac{Z_w}{m - Z_{\dot{w}}}w - \frac{Z_q + mU_0}{m - Z_{\dot{w}}}q - \frac{mg\sin\theta_0}{m - Z_{\dot{w}}} + \Delta Z^c \tag{7}$$

$$\dot{q} = \frac{M_u + Z_u\Gamma}{Iyy}u + \frac{M_u + Z_u\Gamma}{Iyy}w - \frac{M_q + (Z_q + mU_0)\Gamma}{Iyy} - \frac{mg\sin\theta_0\Gamma}{Iyy}\theta + \Delta M^c \tag{8}$$

$$\dot{\theta} = q \tag{9}$$

Where,

$$\Delta X^c = \frac{X_{\delta e}}{m}\delta e + \frac{X_p}{m}\delta e \tag{10}$$

$$\Delta Z^c = \frac{Z_{\delta e}}{m - Z_{\dot{w}}}\delta e + \frac{X_{\delta p}}{m - Z_{\dot{w}}}\delta e \tag{11}$$

$$\Delta M^c = \frac{M_{\delta e} + Z_{\delta e}\Gamma}{Iyy}\delta e + \frac{M_{\delta p} + Z_{\delta p}\Gamma}{Iyy}\delta p \tag{12}$$

.Now, u is assumed to be approximately zero. Hence, $\dot{u} = 0$. We can now rewrite the equations in state space form [2]:

$$\dot{x} = Ax + Bu$$

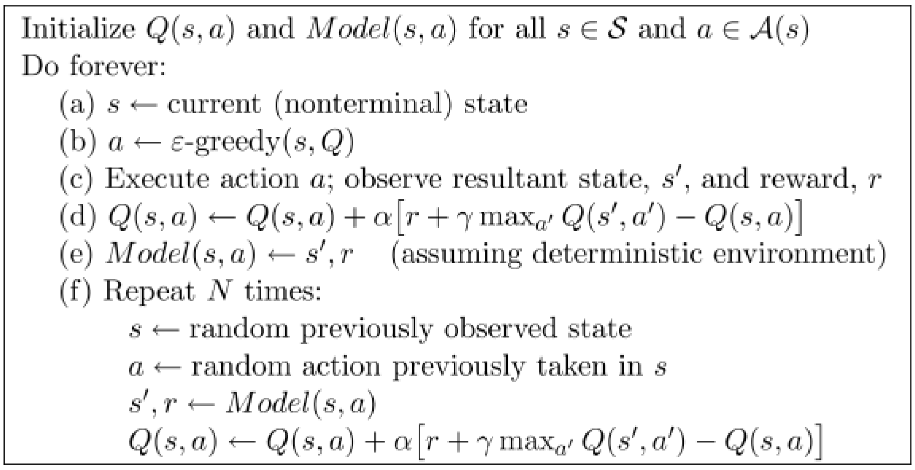
$$y = Cx + Du$$

$$\begin{bmatrix} \dot{w} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{Z_w}{m - Z_{\dot{w}}} & \frac{Z_q + mU_0}{m - Z_{\dot{w}}} & \frac{-mg\sin\theta_0}{m - Z_{\dot{w}}} \\ \frac{M_w + Z_w\Gamma}{Iyy} & \frac{M_q + (mU_0)\Gamma}{Iyy} & \frac{-mg\sin\theta_0}{Iyy} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} w \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta Z^c \\ \Delta M^c \\ 0 \end{bmatrix} \delta e \tag{13}$$

This is the state space which serves as the simulation environment for the learning system.

3 Q-learning

Q learning is a type of reinforcement learning where the agent learns an optimal policy through repeated interactions with its environment. In simple words Q learning algorithm has numerous states and a set of actions that can be performed in those states. A reward is provided for taking a specific action in a state based on the outcome of the action. This reward can be a token of appreciation or a penalty for punishment and is known as positive reinforcement learning and negative reinforcement learning respectively. Not all actions in all states lead to a final state which offer a reward and hence Q learning algorithms have a feature of accounting for a discounted future reward i.e. if performing action 3 in state 1 leads to state 3 which can then reach the reward state 4 through the means of an action 4 then the action 3 in state 1 receives a discounted reward as it has the possibility to reach the final reward state. These interactions are collected and presented as historical data into a Q table or a neural network. This data gets updated with every trial and as time progresses gives the agent a clear view of what action to take in what state to maximize its reward.



```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
    (a) $s \leftarrow$ current (nonterminal) state
    (b) $a \leftarrow \varepsilon$-greedy$(s,Q)$
    (c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
    (d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    (e) $Model(s,a) \leftarrow s', r$    (assuming deterministic environment)
    (f) Repeat $N$ times:
            $s \leftarrow$ random previously observed state
            $a \leftarrow$ random action previously taken in $s$
            $s', r \leftarrow Model(s,a)$
            $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
```

Figure 3: The Q-learning Algorithm.

The Q value is updated using the following expression:

$$Q(s,a) \Leftarrow Q(s,a) + \alpha(r + \gamma * Q_{max}(s',a') - Q(s,a)) \tag{14}$$

Given below is the algorithm used in Q-learning:

The gamma value allows consideration of future values and updates the actions that lead to the state. Q-learning makes this easy for humans, as it performs the selection of actions on its own. Actions that lead to favorable states are positively rewarded, whereas actions leading to an undesired state are negatively rewarded. Over a few thousand trials it can be noticed that algorithm not only reaches the goal but also successfully avoids all high negative reward states. The process of selecting the actions ensures that the best action to perform isn't selected every single time to prevent the agent from learning greedily and to promote exploration. For this project, the ϵ-greedy learning policy has been considered.

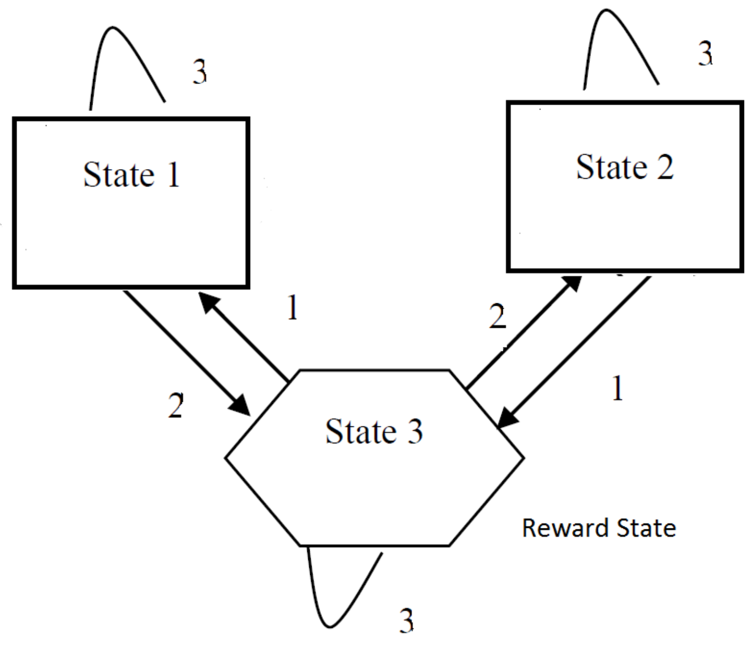


Figure 4: The State Diagram.

4 ϵ-greedy learning policy

ϵ-greedy learning is used to make sure that the chosen action is not always the one with the highest Q-value. This allows the algorithm to explore and update the effects of randomized actions with some probability of occurrence every few iterations. There is no defined method for selecting the value of ϵ, but it is constrained to be valued between 0 and 1. This value of probability is chosen based on experimentation and can yield varying results for varying values of ϵ.

5 Applying Q-learning to the system

For the sake of simplicity, only the longitudinal motion of the aircraft was considered. This means that only the pitch angle is being controlled in this project. This forms a system which has three states and one input, i.e. the elevator angle. Thus, the simulation environment system is a system which takes in the action in terms of an elevator angle deflection. It then calculates the resulting pitch angle and calculates the error with the reference signal. This error is used to generate the state of the system.
The Q-learning algorithm is then used to anticipate future states and actions. This algorithm is then run for 20,000 iterations to update the Q-table. Here, there are three states chosen according to the current value of error. The states are chosen as follows:

$$error < -0.1 \Leftarrow State = 1$$

$$error > 0.1 \Leftarrow State = 2$$

$$0.1 < error < 0.1 \Leftarrow State = 3$$

There are three actions available in each state. These actions are the unit value of deflection in the elevator angle. The Q-table can be given as follows:

States	Action 1	Action 2	Action 3
State 1	0	0	0
State 2	0	0	0
State 3	0	0	0

Where,
Action 1 = -0.1
Action 2 = +0.1
Action 3 = 0

The corresponding reward matrix can be described as:

States	Action 1	Action 2	Action 3
State 1	-10	1	-10
State 2	1	-10	-10
State 3	-10	-10	10

For the Q-learning algorithm, the values of parameters were chosen after experimentation. These are as follows:
Future discount $(\gamma) = 0.8$
Learning rate $(\alpha) = 0.2$
Random learning value $\epsilon = 0.1$

The time-axis is assumed 20 seconds in length, with a minimum division of 0.2 seconds. The reference signal is sampled along this time axis.
The algorithm is run for 100 iterations to update the Q-table. Every iteration, the time value is updated by 0.2 and used to calculate the output value of the differential equations describing the longitudinal motion of the aircraft.
These iterations are run for 20,000 episodes which update values in the Q-tables.

6 Results of the Simulation

There were two versions of implementations of the Q-learning algorithm. The first one takes into account a fixed value of reference pitch angle. This was done to measure the time taken by the algorithm to converge on to the reference. The second implementation tries to get the algorithm to track a signal with respect to time.

6.1 Convergence of Pitch angle to constant reference

The algorithm is run for 10 episodes, each episode consists of a Q-learning loop which runs until the error between the current value of Pitch angle and the reference value of the pitch angle.
The initial state of the system is randomized for the first iteration of every episode. The action taken in this state is also random for the first iteration of each episode. The values of ϵ, γ and α are the same for both the results.
The results for the time taken to converge onto the reference value is given in the figure below:

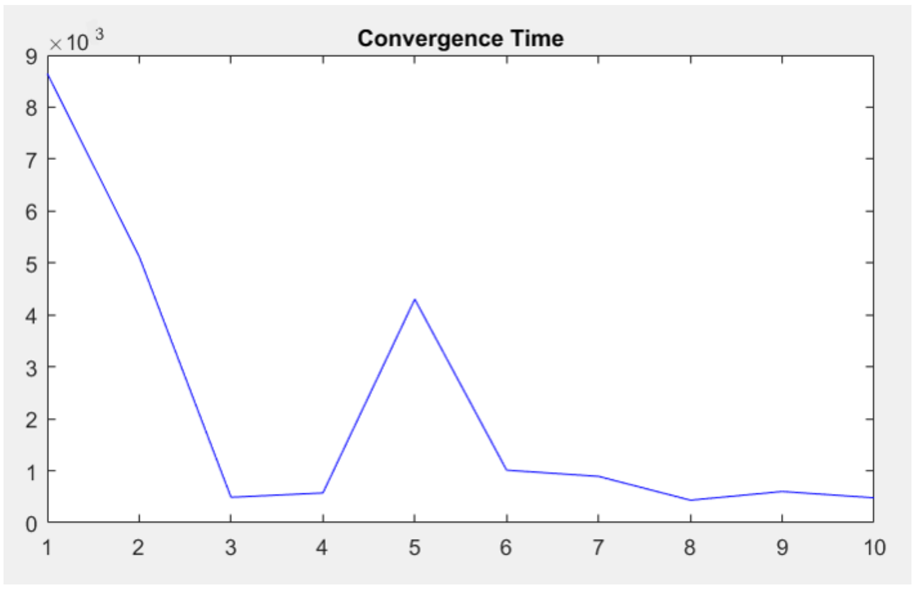


Figure 5: Time taken to converge across 10 episodes.

Here, the spike in the convergence time drops sharply by the first three episodes. The spike in time taken to converge across episodes 4 to 6 can be explained by the random actions taken by the ϵ-greedy policy.

6.2 Convergence of Pitch Angle to a time dependent signal

This algorithm is run for 20,000 episodes, each episode consisting of 100 iterations. The time is updated by $\tau = 0.2$. The values of ϵ, γ and α are the same for both the results. The pitch angle and the reference signal are plotted with respect to the time. The results of the convergence are shown below:

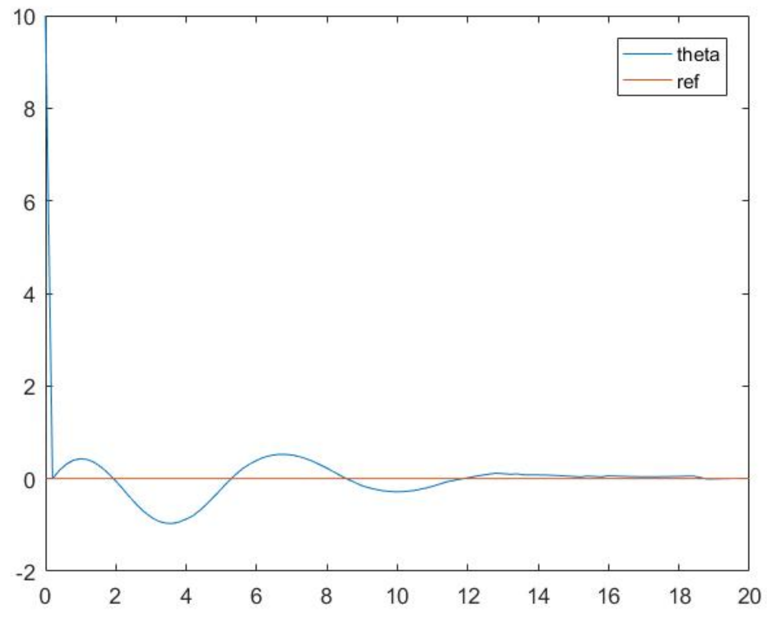


Figure 6: Convergence of pitch angle to reference signal across 20 seconds.

The value of the Pitch angle begins with an initial condition of 10 degrees. Over the episodes, the algorithm updates the Q-table enough times to sharply reduce the convergence time.

The convergence of the pitch angle to a time varying signal was also explored. The reference signal used is a sine wave of frequency 0.2Hz. The reference tracking of the sine wave is describe in the plot given below:

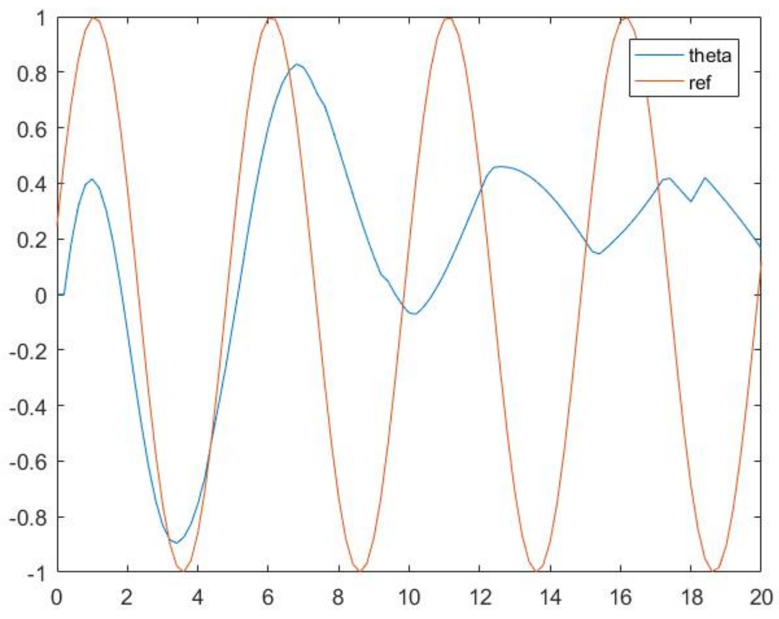


Figure 7: Convergence of pitch angle to a sine wave across 20 seconds.

The tracking for this signal suffers. However, this appears to be a consequence of the dynamics of the aircraft, and the frequency of the sine wave chosen.

Reducing the frequency of the sine wave results in a better convergence output, which results in better tracking:

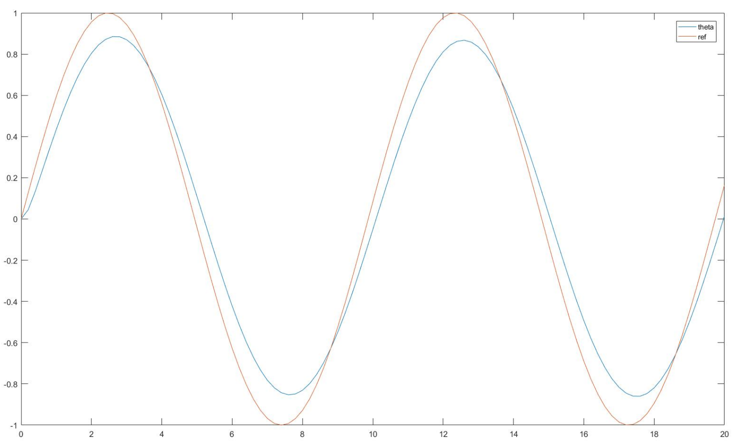


Figure 8: Convergence of pitch angle to a sine wave across 20 seconds.

7 Analysis of results

The control of fixed wing aircraft is traditionally achieved using LQR and LQG controllers. Thus, the performance of the Q-learning algorithm can be compared to the LQR response for the same fixed wing aircraft. Given below is the response for an LQR controller for the same system, converging to zero from an initial condition of 10

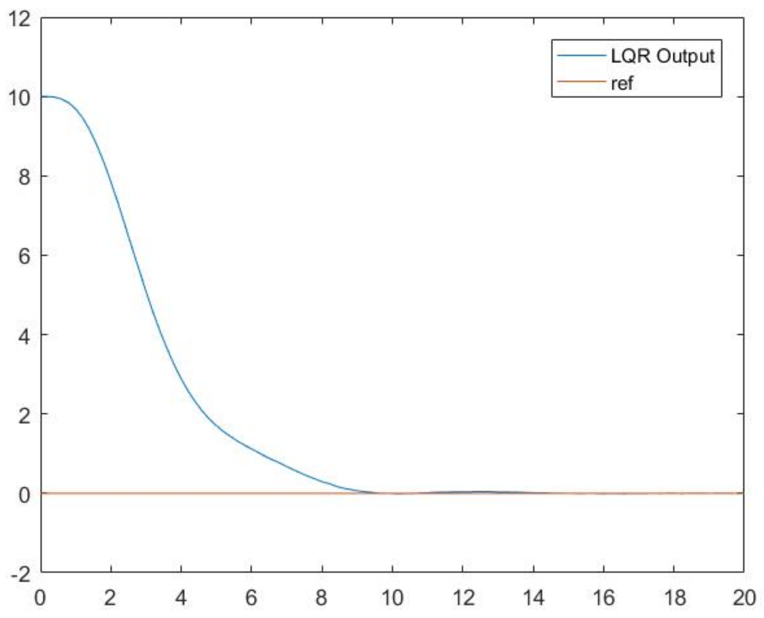


Figure 9: Convergence of pitch angle to reference signal across 20 seconds for an LQR controller.

As the results reveal, the LQR controller converges slower than the Q-learning controller, but has less steady-state error. The final Q-table for the time dependent signal tracking controller is given below:

States	Action 1	Action 2	Action 3
State 1	-792.88	10.79	-104.98
State 2	15.24	-143.34	-304.89
State 3	-252.77	-422.53	36.44

It can be observed, that the Q-values which are positively rewarded in the Reward matrix, increase in value, and the Q-values which are negatively rewarded in the Reward matrix, decrease in value. The absolute values of the Q-values continue to increase with every iteration, unless a mechanism to normalize the Q-values is implemented.

8 Conclusions

This project has demonstrated that Q-learning algorithms can achieve comparable performance to commercially popular controllers based on optimal value calculation. The control scheme developed by the end of the paper should serve as a robust controller for the regulation of the Pitch angle for a typical fixed wing aircraft. Compared to a more mainstream controller such as the PID, the LQR or the LQG the Q-learning controller seems to converge faster. However, the steady state characteristics need to be explored further
Instead of tuning gains to match performance parameters or finding optimal gains calculated by solving the Riccati equations for the system, this approach requires zero knowledge of the dynamic system, while still providing better convergence than a typical PID controller.

9 Future Work

As the number of state-action pairs grows, the size of the Q-table blows up exponentially. It becomes difficult to create a Q-table to keep track of the state-action pairs. This is where Neural Networks come in. Neural networks are used to replace the Q-table, where the individual nodes represent the various state-action pairs, and their weights represent the Q-values of the state-action pair. Thus, since Q is always changing, the algorithm is basically an infinite horizon learning function, and increasing the number of iterations improves the Q values [3].

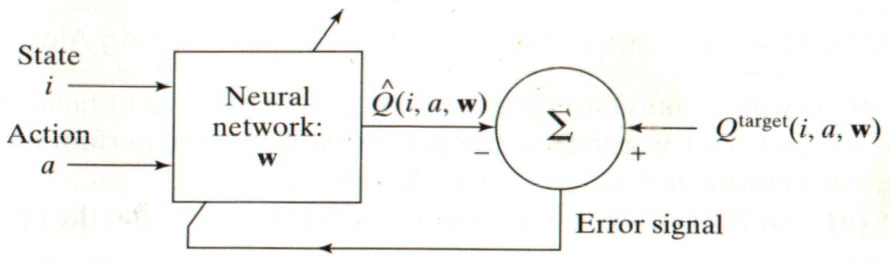

State
i
Action
a
Neural network: $\mathbf{w}$
$\hat{Q}(i, a, \mathbf{w})$
Σ
$-$
$+$
$Q^{\text{target}}(i, a, \mathbf{w})$
Error signal

Figure 10: Neural network with Q-learning.

This is something that could be used to add more states to the Q-table, and improve the performance achieved.
Another addition to the project could be the inclusion of the lateral motion states and the various velocities of the aircraft. This allows control of more states and by extension, all the states controlling the motion of the aircraft. The simulation environment for the non linear lateral motion to be controlled is as follows:

$$\begin{bmatrix} \dot{\beta} \\ \dot{p} \\ \dot{r} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{Y_\beta}{U_0} & \frac{Y_p}{U_0} & \frac{Y_r}{U_0} & \frac{g\cos\theta}{U_0} \\ L_\beta & L_p & L_r & 0 \\ N_\beta & N_p & N_r & 0 \\ 0 & 1 & \tan\theta_0 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} \frac{Y_{\delta_r}}{U_0} & \frac{Y_{\delta_a}}{U_0} \\ L_{\delta_r} & L_{\delta_a} \\ N_{\delta_r} & N_{\delta_a} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{15}$$

This system involves two control inputs, i.e. the aileron and rudder deflection angles, which are used to control the lateral motion of the aircraft i.e. the Roll and Sideslip angle. Future work in this area would include adding these to the simulation environment, so as to have a Q-learning controller capable of controlling the entirety of the aircraft's motion.

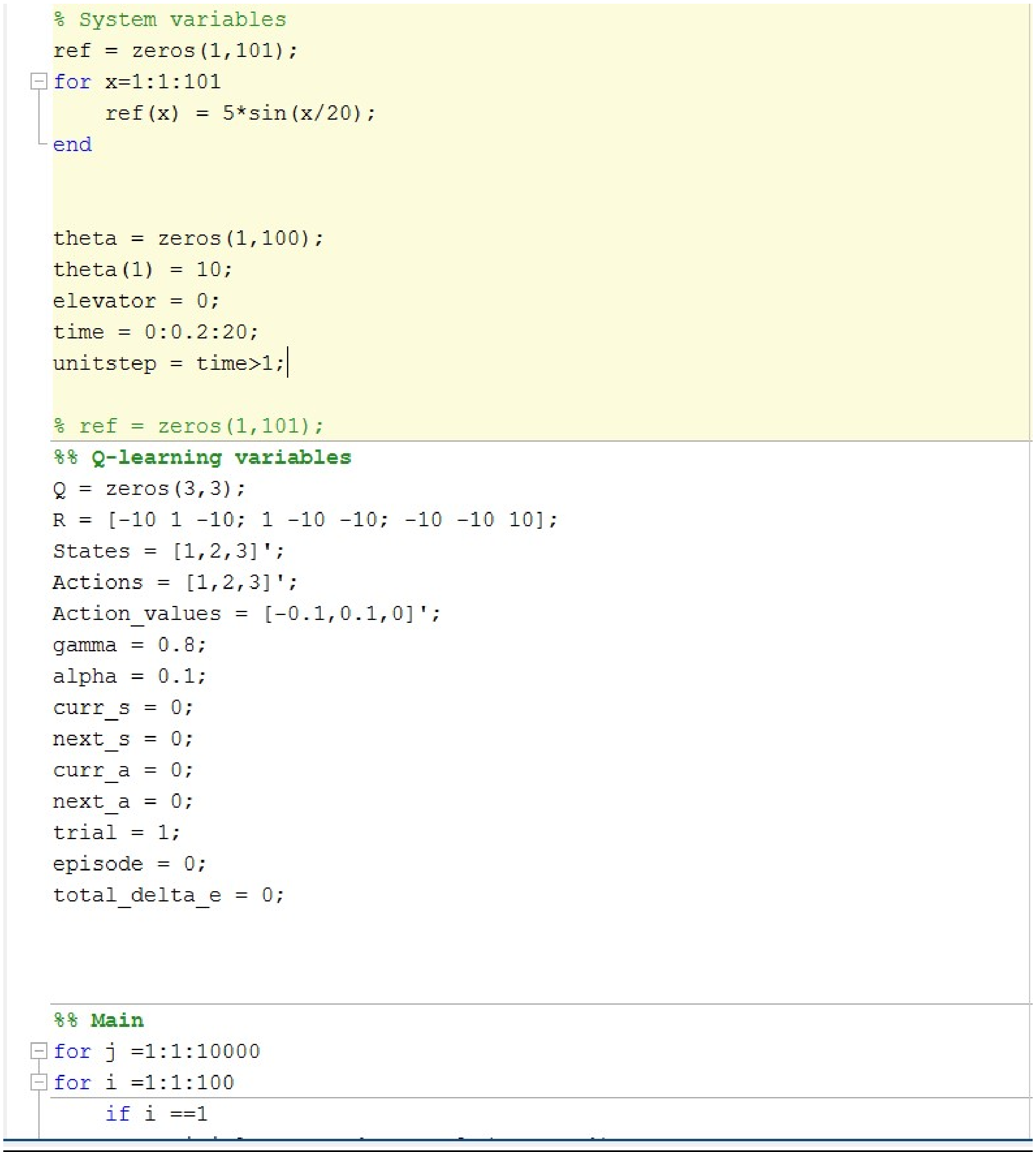
References

[1] Labane Chrif, Zemalache Meguenni Kadda,
`Aircraft Control System Using LQG and LQR Controller with Optimal Estimation-Kalman Filter Design`. 3rd International Symposium on Aircraft Airworthiness, ISAA 2013

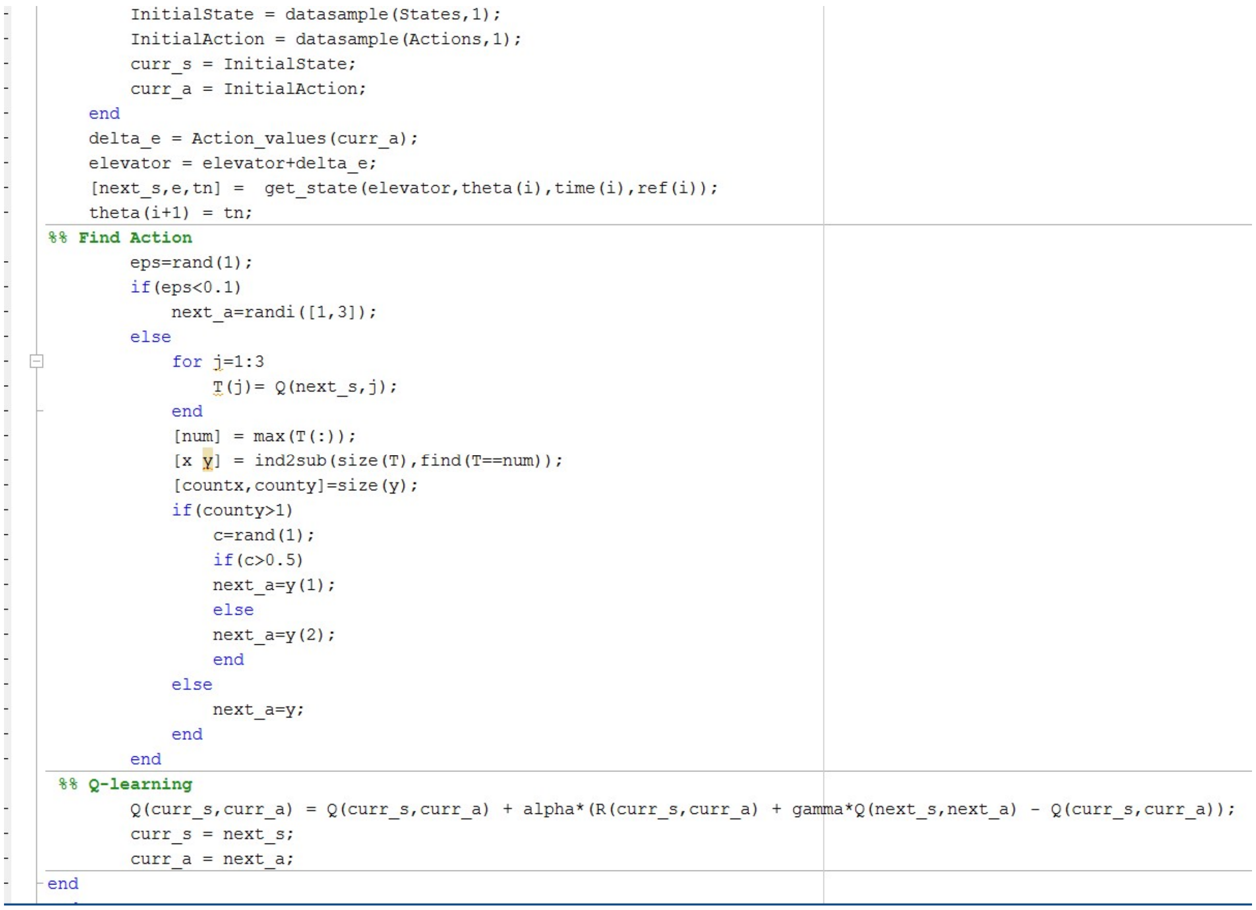
[2] Reed Siefert Christiansen (2004).
`Design of an Autopilot for Small Unmanned Aerial Vehicle` Brigham Young University.

[3] Kangbeom Cheon, Jaehoon Kim, Moussa Hamadache, and Dongik Lee
`On Replacing PID Controller with Deep Learning Controller for DC Motor System`. School of Electronics Engineering, Kyungpook National University, Daegu, Korea

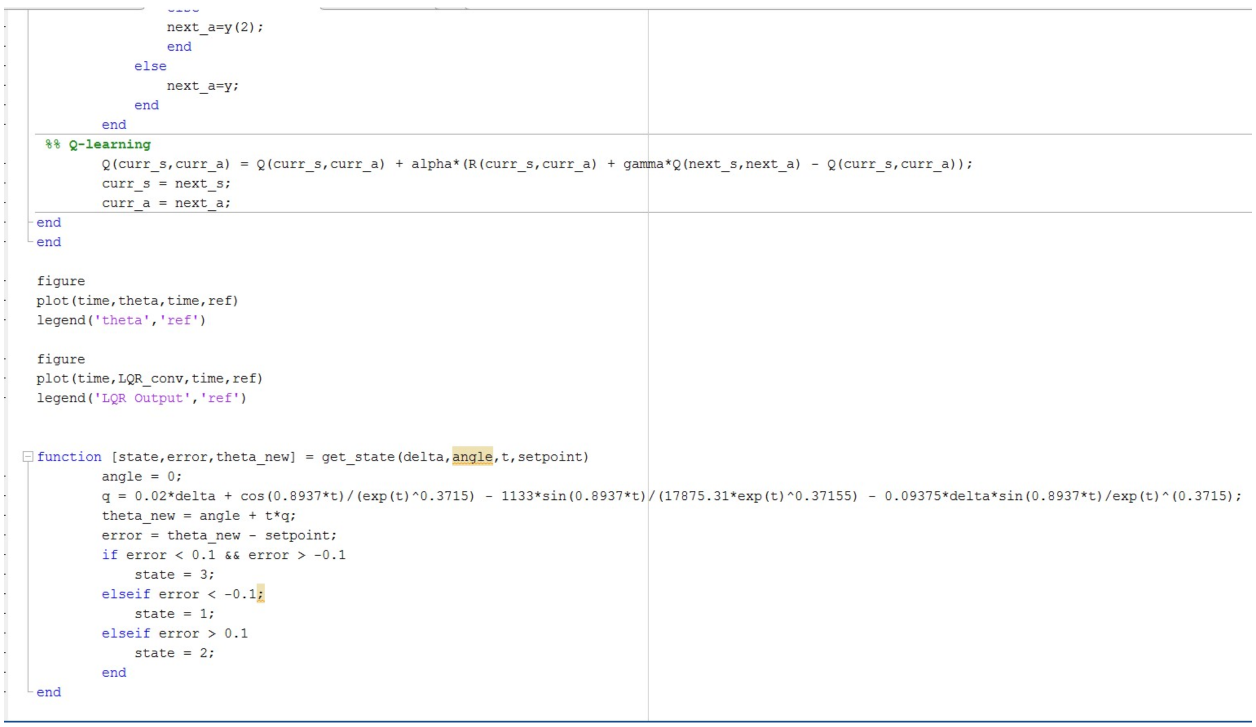
APPENDIX:



```
% System variables
ref = zeros(1,101);
for x=1:1:101
    ref(x) = 5*sin(x/20);
end


theta = zeros(1,100);
theta(1) = 10;
elevator = 0;
time = 0:0.2:20;
unitstep = time>1;

% ref = zeros(1,101);
%% Q-learning variables
Q = zeros(3,3);
R = [-10 1 -10; 1 -10 -10; -10 -10 10];
States = [1,2,3]';
Actions = [1,2,3]';
Action_values = [-0.1,0.1,0]';
gamma = 0.8;
alpha = 0.1;
curr_s = 0;
next_s = 0;
curr_a = 0;
next_a = 0;
trial = 1;
episode = 0;
total_delta_e = 0;



%% Main
for j =1:1:10000
for i =1:1:100
    if i ==1
```



```
        InitialState = datasample(States,1);
        InitialAction = datasample(Actions,1);
        curr_s = InitialState;
        curr_a = InitialAction;
    end
    delta_e = Action_values(curr_a);
    elevator = elevator+delta_e;
    [next_s,e,tn] =  get_state(elevator,theta(i),time(i),ref(i));
    theta(i+1) = tn;
%% Find Action
        eps=rand(1);
        if(eps<0.1)
            next_a=randi([1,3]);
        else
            for j=1:3
                T(j)= Q(next_s,j);
            end
            [num] = max(T(:));
            [x y] = ind2sub(size(T),find(T==num));
            [countx,county]=size(y);
            if(county>1)
                c=rand(1);
                if(c>0.5)
                next_a=y(1);
                else
                next_a=y(2);
                end
            else
                next_a=y;
            end
        end
 %% Q-learning
        Q(curr_s,curr_a) = Q(curr_s,curr_a) + alpha*(R(curr_s,curr_a) + gamma*Q(next_s,next_a) - Q(curr_s,curr_a));
        curr_s = next_s;
        curr_a = next_a;
end
```



```matlab
                next_a=y(2);
                end
            else
                next_a=y;
            end
        end
 %% Q-learning
        Q(curr_s,curr_a) = Q(curr_s,curr_a) + alpha*(R(curr_s,curr_a) + gamma*Q(next_s,next_a) - Q(curr_s,curr_a));
        curr_s = next_s;
        curr_a = next_a;
end
end

figure
plot(time,theta,time,ref)
legend('theta','ref')

figure
plot(time,LQR_conv,time,ref)
legend('LQR Output','ref')


function [state,error,theta_new] = get_state(delta,angle,t,setpoint)
        angle = 0;
        q = 0.02*delta + cos(0.8937*t)/(exp(t)^0.3715) - 1133*sin(0.8937*t)/(17875.31*exp(t)^0.37155) - 0.09375*delta*sin(0.8937*t)/exp(t)^(0.3715);
        theta_new = angle + t*q;
        error = theta_new - setpoint;
        if error < 0.1 && error > -0.1
            state = 3;
        elseif error < -0.1;
            state = 1;
        elseif error > 0.1
            state = 2;
        end
end
```