MEGN545A Class Project

I. Introduction

Software: Matlab/Simulink. Your mines email comes with one academic license. Register an account with your mines email in the following link so you can download one on your own computer (https://www.mathworks.com/login), or you can use the computers in the school lab.

Parameters, codes, file names will be highlighted in green; important notes will be highlighted in red.

Check the files in the project package and remember their names:

project.slx: contains the model, and reference control signal.

MPCmodel.slx: the model you need in the Task 2, MPC.

Drone.pdf: the drone dynamics.

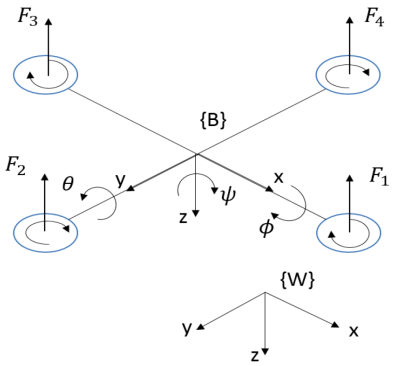


Fig. 1. Schematic of Quadrotor Dynamics

iteration.m: an example of iteratively updating the Q matrix.

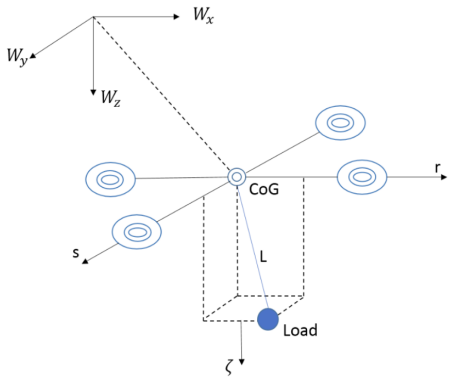


Fig. 2. Schematic of quadrotor dynamics with a slung load

IterationExample.slx: the Simulink model for the iteration example.

squaretrajectory.mat: a reference trajectory that the drone needs to follow.

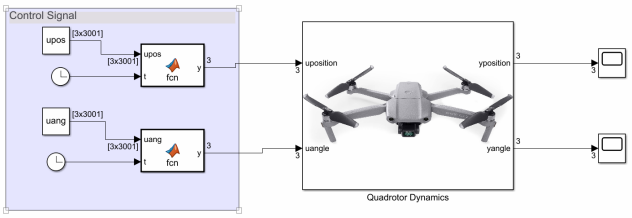


Fig. 3. Simulink Model (*classproject.slx*)

In this project, you will complete three tasks based on a quadrotor model with an uncertain suspended load (Fig. 1-2) in Matlab/Simulink (version 2020b or later). The load is connected

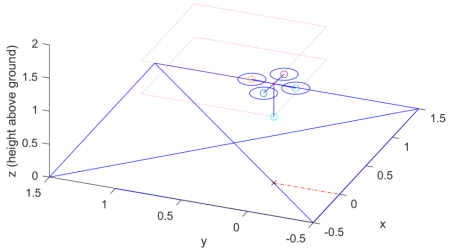


Fig. 4. Visualization of the quadrotor

to the center of the quadrotor with a cable. The provided dynamic model is built in Matlab/Simulink (Fig. 3), where the 3x1 input *uposition* is the signal to control the quadrotor's position (x,y,z), which are the translational displacements in x, y, and z directions). The 3x1 input *uangle* is the signal to control the quadrotor's rotational angle ((ϕ,θ,ψ), the angles of roll, pitch, and yaw). *yposition* and *yangle* are the corresponding output. The initial position of the quadrotor is (0,0,1.5) and all other parameters are zero. More information on the quadrotor dynamics is provided in the *drone.pdf*. It's not necessary to fully understand the dynamics to finish the task, but it will be helpful when you design the action and state your RL controller. Running the Simulink model will plot visualization of the quadrotor, as shown in Fig. 4. The blue lines are the ground, the upper square with the red dot is the initial trajectory of the center of the quadrotor, the lower square with the red dot is the initial trajectory of the load. The example control signals are provided in the model, named *squaretrajectory.mat*. You can run it to see

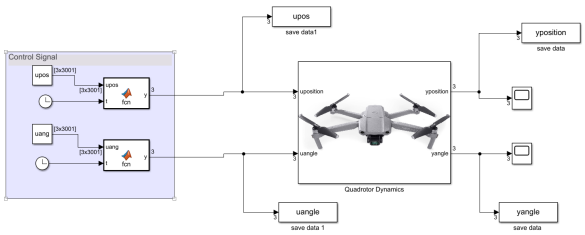
how the quadrotor tracks the trajectory. Section II introduces the details of the tasks you need to finish in this project. Hints to complete each task are provided. Section III specifies the project deliverables.

II. Project Tasks

A. To-Do 1-Train a Neural Network Model

Generate training dataset to train a neural network plant model (Fig. 5) of the quadrotor dynamics. Use *nnstart* tool or write code. Show the NN parameters' tuning process and provide training results and validation results. Discuss the results and improve the model performance. A video example of collecting data and train NN (use *nnstart* tool) is shown in https://youtu.be/C5S_IHbP3xA. Alternatively, you can follow the below hints (using code):

a. Collect the training data with the provided control signal (Fig. 5). Save the data as a 2-d array to the workspace.



Workspace

Name	Value
tout	3001x1 double
uangle	3001x3 double
upos	3001x3 double
yangle	3001x3 double
yposition	3001x3 double

Fig. 5. Example Simulink model and saved data structure.

b. Use num2cell (Fig. 6) function to transfer all the data to cell format. Example:

uposcell = num2cell(upos',1)

upos	3001x3 double
uposcell	1x3001 cell

Fig. 6. Example of converted data in cell format.

c. Run the following code in command window:

openExample('nnet/model_maglev_demo')

Copy the code to a new *.m* file. Go to the copied code. Start with the *narxnet* part, create NN first.

1. Then prepare the data. Remember to change the *[x,t]* to *[u,y]*.
2. Add the following line before the *train* function; this changes the training function. Go to this website (https://www.mathworks.com/help/deeplearning/ref/fitnet.html) to check what training function is available, and choose one that may have better performance:

   net.trainFcn = 'trainbr'
3. Train the NN (Fig. 7) by running:

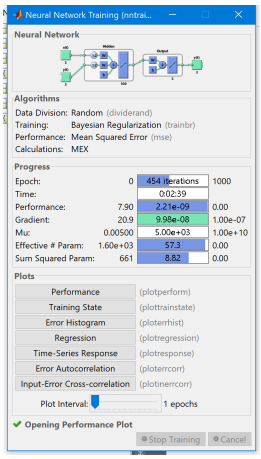


Fig. 7. Example of training NN.

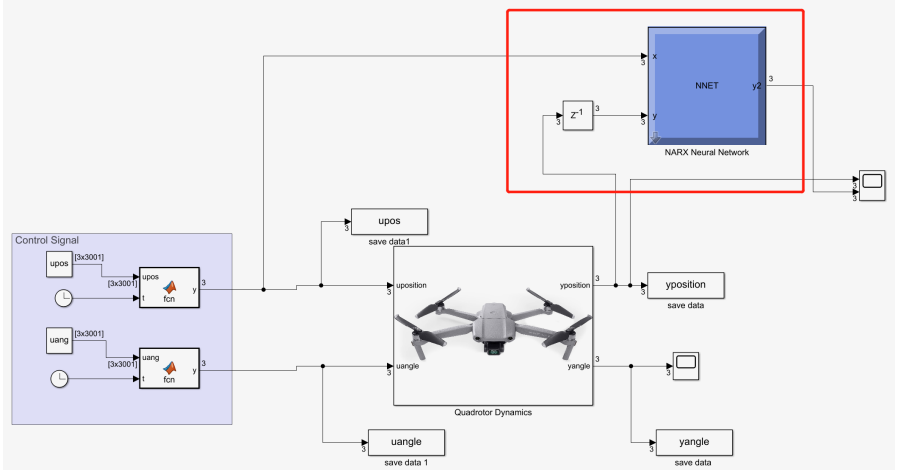


Fig. 8. Example Simulink model of NN validation.

[net,tr] = train(net,Xs,Ts,Xi,Ai);

4. Save all the performance plots, include them in the report.

d. Now start the validation process (Fig. 8).

1. First, remove the delay layer of the trained NN with the following code:

netnodelay = removedelay(net);

2. Then convert the net to Simulink block with the following code:

gensim(net netnodelay)

3. Copy the NN block to the model where you collect data and connect the block as shown below. For delayed signal, add a delay block between the output of the quadrotor model (see Fig. 8) and the delay input (y) of the NN model.

4. Run the model; you can check the output in the scope; an example scope is shown in Fig. 9:

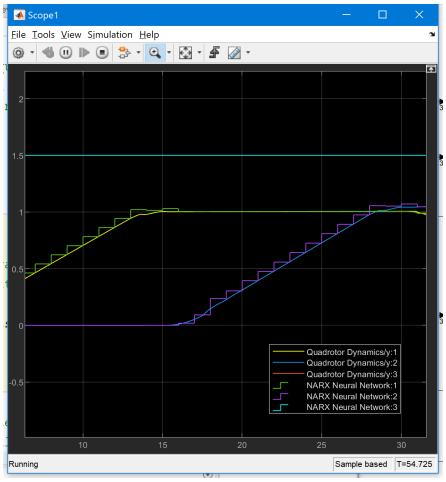


Fig. 9. Example scope plot.

5. Save the output of NN, statistically compare it to the output of the drone model (calculate the error, validate performance, etc.). Analysis the results.

6. Repeat the above steps for the *yangle*.

7. If you want to train the model with your own data, you can use your own input signal. Carefully design the input so that the collected data is suitable for training.

B. To-Do 2-Develop a MIMO MPC Controler

First, read the example:
https://www.mathworks.com/help/mpc/gs/control-of-a-multi-input-multi-output-nonlinear-plant.html
A video example is shown in:
https://youtu.be/EQ8zb13m_bM
Develop a MIMO-MPC to control the quadrotor to finish task 2 (square trajectory). Use the *MPCmodel.slx* as the plant model to be linearized in the example code. Create the system model

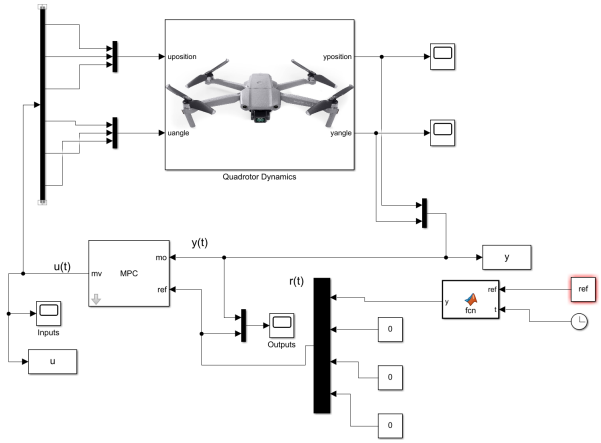


Fig. 10. Example Simulink diagram of the MPC model.

according to the video example (Fig. 10, use the Quadrotor Dynamics Block in *project.slx*).
Hints:

a. to avoid the error caused by the pole near z=0, add a line of code after the linearization:
plant = minreal(plant);

b. Build the MPC Simulink model and MATLAB code based on the video example. The reference signal input to the MPC controller is shown in Fig. 11.

```
function y = fcn(ref,t)
T = floor(t/0.025)+1;
y = ref(:,T);
```

Fig. 11. The reference signal input to MPC controller.

c. Double click the Quadrotor Dynamics block (Fig. 12), You can see the set pace block; it set the simulation block to make the animations smooth. You can delete this block for both the plant and MPC controller models to achieve better control performance.

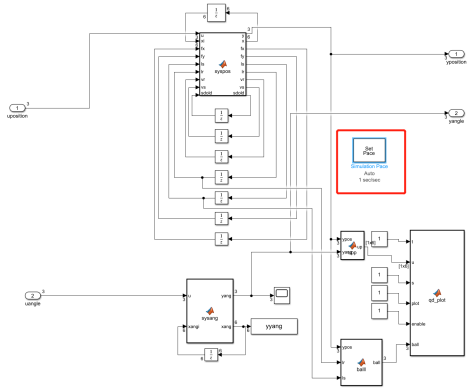


Fig. 12. Set pace block.

d. Press ctrl+E, open the model configuration parameters, set the solver as shown in Fig. 13:

e. Go to MATLAB, set the MPC parameters (Fig. 14):

f. Run the simulation. Check if the drone follows the reference trajectory, it may look like Fig. 15:

g. Because the MPC parameters are not well-tuned, the drone can follow the x,y position well but experienced oscillation in height (z position). You should tune those parameters to achieve better performance.

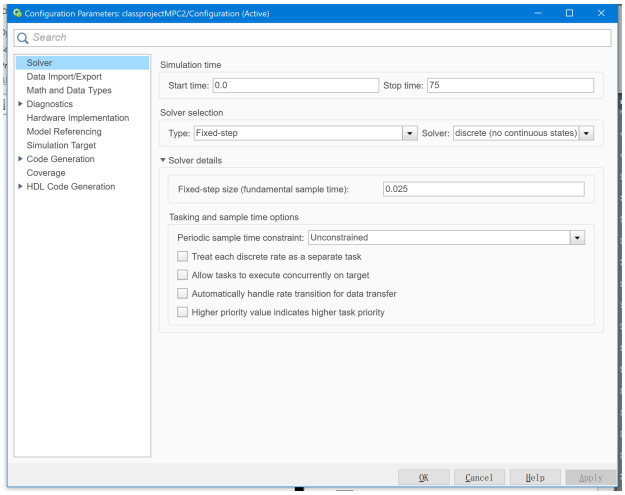


Fig. 13. Model configuration parameters.

```
%% Design MPC Controller
% Create the controller object with sampling period, prediction and control
% horizons:
Ts = 0.025;
p = 5;
m = 2;
mpcobj = mpc(plant,Ts,p,m);

%%
% Specify MV constraints.
mpcobj.MV = struct('Min',{-0.0627;-0.0627;-0.0627;-0.0777;-0.0777;-0.0777},...
    'Max',{0.0627;0.0627;0.0627;0.0777;0.0777;0.0777},...
    'RateMin',{-1000;-1000;-1000;-1000;-1000;-1000});

%%
% Define weights on manipulated and controlled variables.
mpcobj.Weights = struct('MV',[0 0 0 0 0 0],'MVRate',[.1 .1 .1 .1 .1 .1],'OV',[1 1 1 1 1

%% Simulate Using Simulink
% Run simulation.
mdl1 = 'classprojectMPC';
open_system(mdl1)
sim(mdl1)
```

Fig. 14. Set MPC parameters.

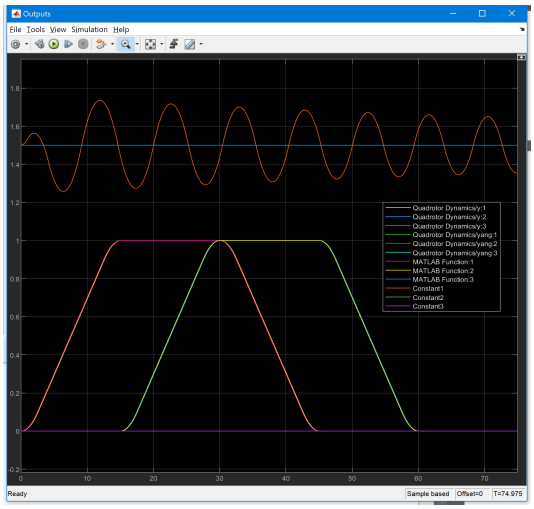


Fig. 15. Example MPC scope.

C. *To-Do 3-Develop a Reinforcement Learning Algorithm*

Develop a Q-learning RL agent from scratch to control the quadrotor. **The action range for each dimension of *uposition* is [-0.0627, 0.0627]. The action range for each dimension of *uangle* is [-0.0777, 0.0777]. The task of the RL agent is to track the initial square trajectory provided in the file *squaretrajectory.mat.* Appropriate rewards for each state should be designed to learn the correct policy.**

Compare your RL agent's performance with the provided control signal in *project.slx* and the MPC in task 2.

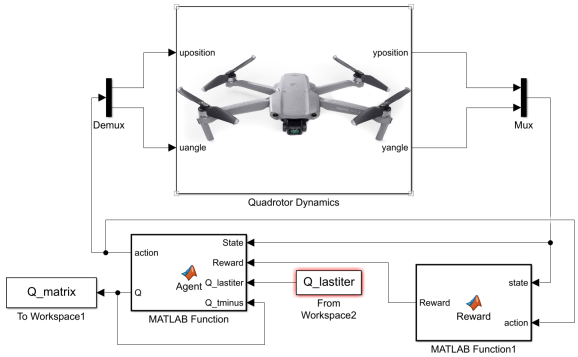


Fig. 16. Example RL model.

For the Q-learning algorithm, you have two options, you will need to implement one, not both:

1. Use the basic Q matrix to store the Q-value of all states. In this way, you will discretize the 3d state space and create the high-dimensional Q-matrix. Implement the methods shown in the in-class room navigation example to update the Q-value. More details are provided in the hints. This option is simple to understand, but require more computational resource and computer memory, which might be slow and inefficient to run.

2. Use the approximate Q-learning method, define several features to describe the state. Use a linear summed function to predict the Q value:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Your work is to design the correct algorithm to update the weights of the linear-summed Q function, whose complexity only depends on the number of features (for 10 features, you only need to update 10 weights).

Hints:

a. An example model configuration is shown in Fig. 16 (It is not provided; you should configure it by yourself. The loop might be different depending on your designed q-learning algorithm).

b. The reward block takes state and action as input, output the reward (action is not necessary). The error of position can be a distance. Other options are the weighted sum of the error in x,y, and z dimensions. You can assign different weights to each dimension so the drone can focus on x,y positions. Like

.

R_position=10(x(t)-x_refer(t))^2+10(y(t)-y_refer(t))^2+2(z(t)-z_refer(t))^2.

The reward can be designed as the negative value of the error using the above equation. Note that the above equation is just an example of a possible reward. Your final reward can be different from it.

For the angle, there is no reference signal. However, a reward is necessary to keep the drone stable. Assuming the rotation angle (0,0,0) is the most stable position, the drone wants to keep the rotation angle within 10 degrees. You can define the reward function as

R_angle = -{(10-x)^2+ (10-y)^2+ (10-z)^2}

Note that the above equation is just an example of a possible reward. Your final reward can be different from it.

c. The reference signal provided in *squaretrajectory.mat* has 3000 points. It was generated based on the following setup: the trajectory is a square with four points (0,0,1.5), (0,1,1.5), (1,1,1.5), (1,0,1.5), the total task length is 75s, the step size is 0.025s. IF you train the agent with the provided reference signal, the agent should also finish the task in 75s. If you want more flexibility, you can generate the reference signal yourself. You can make the drone fly a circle or triangle. The task length can be any number, but the step size is fixed to 0.025s. So, the total number of reference points is *tasklength/0.025*.

Depends on your option, you need to choose to follow d or e:

d. If you choose the Q-matrix option, you first need to discretize the state space and the actions. For example; the minimum state space is $(x, y, z, \alpha, \beta, \gamma)$, for each dimension, you can define 50 states, the state space will be a matrix with size (50x50x50x50x50x50). You also need to discretize the given range for the possible actions, like 6 elements (-0.0627,-0.0427,-0.0227,0,0.0227,0.0427,0.0627); the more possible actions you have, the better control performance. To define an N-dimensional matrix, simply use the zero function in MATLAB. For example, a 50x50x3x6x6x6 matrix can be defined as:

Q = zeros(50,50,3,6,6,6)

e. If you choose the Approximate Q option, define the features that can appropriately describe the state situation. Too many features may be difficult for the weights to converge. Too few features may cause poor performance.

f. Design the exploration/exploitation function to control the agent behavior during the training. Make sure it apply some random actions at the start of the training so it can collect enough experience to update the Q matrix or the weights in the approximated Q-function.

g. To implement the iteration training in matlab, , you can simply run a for loop like the following scripts in MATLAB:

```
for t= 1: 100 % total simulation steps
Sim('yourmodel.slx')
end
```

Remember: load the Q matrix or the weights for the approximated Q-function from the workspace at the start of each iteration. Save the Q matrix or the weights to the workspace at the end of each iteration. You can find the save and load block from the Simulink library. Example files (*iteration.m, iterationexample.slx*) are included in the project package to show how to implement the iteration. You can run the *iteration.m* to see how the value is updated during the simulation. In each iteration, the value is doubled and saved as Qnew to the workspace for the next iteration.

h. Now move to the Q-learning algorithm, which iteratively updates the Q matrix or the weights in the approximated Q-function as the drone is flying. Use the hint f to run the model iteratively, each iteration will be a training episode. At the start of each episode, load the Q matrix or weights of the last episode from the workspace. Save the new Q matrix weights to the workspace at the end of each episode. The drone starts from the initial position (0,0,0) every time and selects action based on the last Q matrix/Q function or random action.

i. You need to define a function to find all the possible Q values based on the available actions and form them in a vector. Then you can use MATLAB function *max()* to find the maximum Q value. Next, you can choose the corresponding action by checking the maximum Q value's direction. Alternatively, you can use MATLAB function *sort()* to return the maximum value and index for easier action selection.

j. Final and biggest hint: Carefully read the error message and search the problem online. MATLAB is supposed to have the best documentation, and you can easily find the answer. Try your best to debug the code by yourself. Double-check the dimension of the data in the MATLAB workspace. If you want to check the dimension of the input and output of the block in Simulink, turn on the information overlay (Signal dimension) as shown in the Fig. 17.

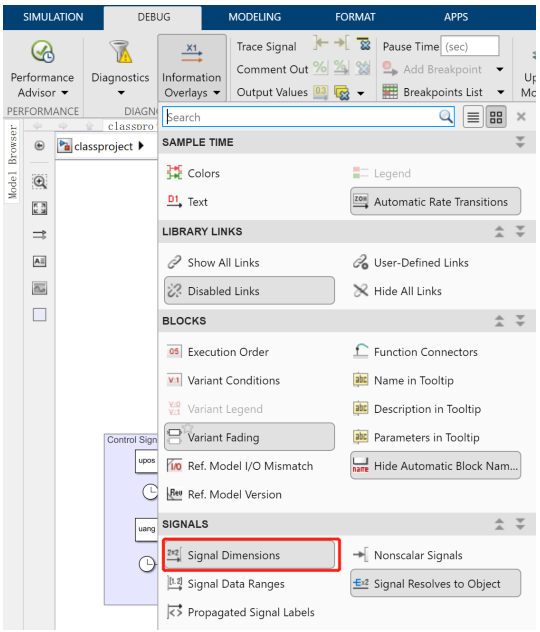


Fig. 17. Check signal dimensions.

III. Project Deliverables

Complete all tasks and write a report up to 10 pages in IEEE format. You can prepare the report with Microsoft Word or LaTex. LaTex is highly recommended, it will save you a lot of time on format. You can find the LaTex IEEE template here:

https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn

Submit any Matlab code, Simulink model, and Videos up to 1 min. The easiest way is to use screen recorder software to capture the whole screen and crop it. You can check the following links to download the free software and use the online video cutter.
https://obsproject.com/
https://online-video-cutter.com/

The grading criteria:

To do task 1: 30 pts
To do task 2: 30 pts
To do task 3: 40 pts

for each to do task:

1. Problem formulation for each To Do. 20 %
2. The design process of each controller. 20 %
3. Functioning codes and models. 30 %
    - Can update Q (functioning code); reward defined appropriately 90%
    - Everything above + Optimized Q for efficient learning 100%
4. Results and discussion. 20 %
    - Good results and discussion 100%
    - Good results and poor discussion 90%
    - Poor results and good discussion 80%
    - Poor results and poor discussion 70%
5. Writing and video visualization. 10 %

Here is the checklist for the report:

a. NN
    - Introduce task
    - Include all the plots in the NN training
    - Plot the output of the NN model and plant model together
    - Calculate the MSE between the NN output and plant output.
    - Discuss the training process and how the parameters affect the performance

b. MPC
    - Introduce task
    - Plot the output of the plant and reference signal together
    - Calculate the MSE
    - Compare performance with the provided control signal (MSE, variance)
    - Discuss the tuning process and how the parameters affect the performance

c. RL
Introduce task
In training
    - Plot the episode reward during the training process (i.e., the cumulative reward, the x axis is episode number)
    - Plot the running average episode reward during the training process (i.e., $\overline{\mathbb{R}_n} = \sum_{i=1}^{i=n-1} R_i /(n-1)$, $R_i = \sum_{t=1}^{t=T} r_t$)
    - Plot the reward for episodes in early training stage, mid training stage, late training stage. (i.e., three plots, each shows the reward within that stage, the x axis is time step)

After training
    - Run multiple times, calculate the MSE between the drone trajectory and the reference signal
    - Show a video of your trained policy
    - Discuss the training process and how the parameters affect the performance

d. Conclusion
Compare the performance of all controllers. Conclude the pros and cons.