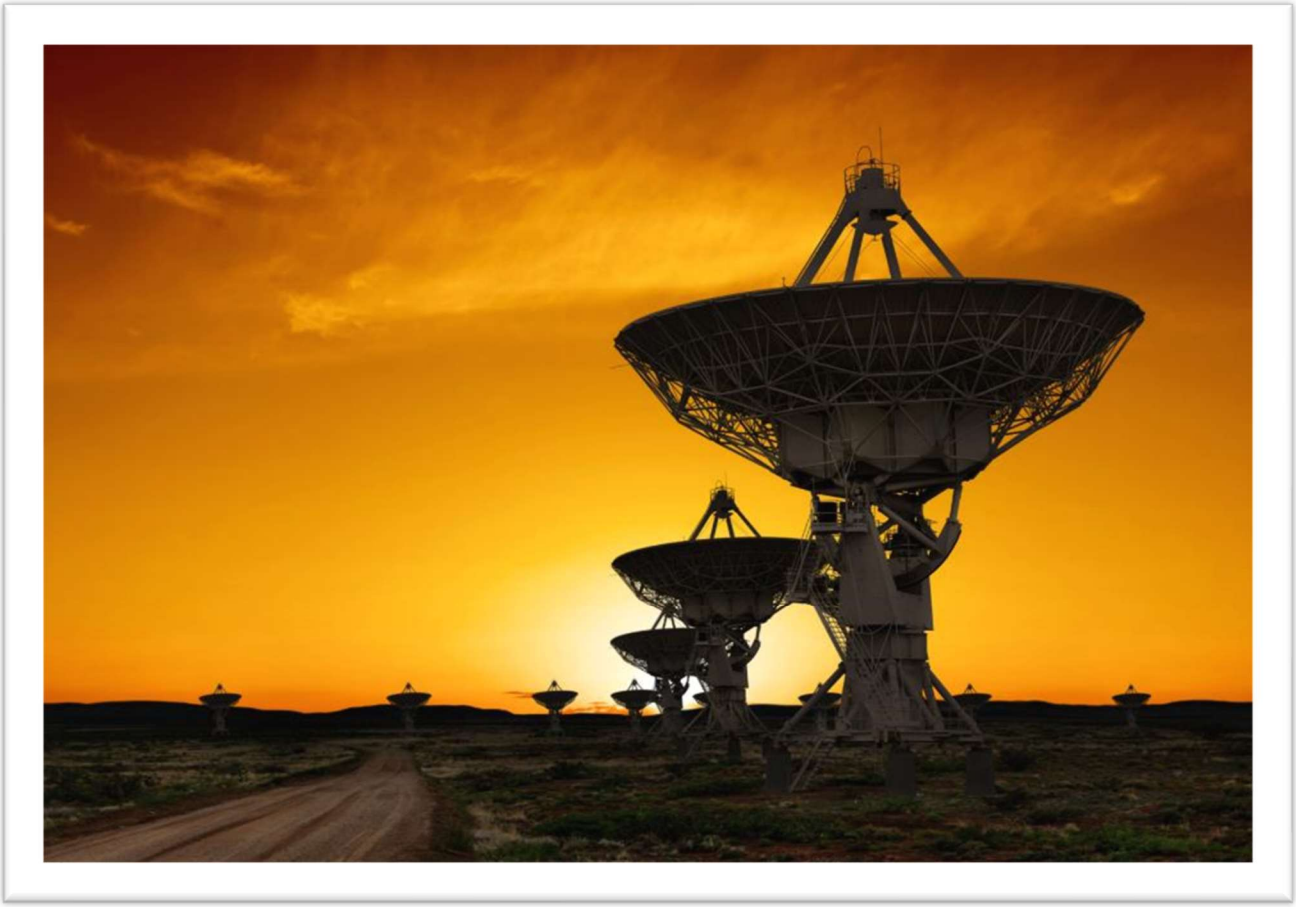


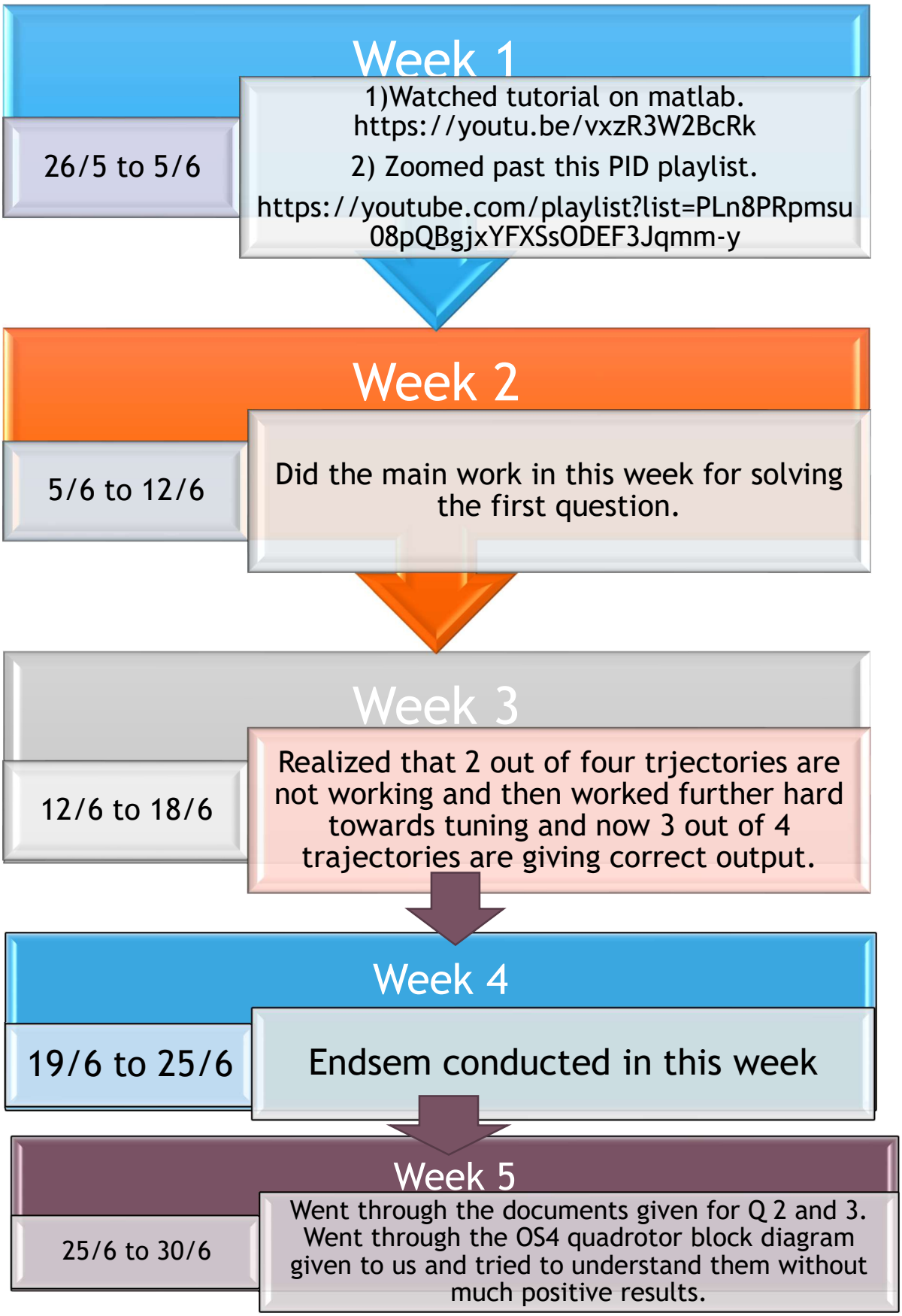
TASK ROUND DOCUMENTATION (Aerial Robotics Kharagpur)

NAME: MITADRU DATTA

ROLL: 21AE10024

Date: 30th June, 20

Timeline


Week 1
26/5 to 5/6
1)Watched tutorial on matlab. https://youtu.be/vxzR3W2BcRk
2) Zoomed past this PID playlist.
https://youtube.com/playlist?list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y
Week 2
5/6 to 12/6
Did the main work in this week for solving the first question.
Week 3
12/6 to 18/6
Realized that 2 out of four trjectories are not working and then worked further hard towards tuning and now 3 out of 4 trajectories are giving correct output.
Week 4
19/6 to 25/6
Endsem conducted in this week
Week 5
25/6 to 30/6
Went through the documents given for Q 2 and 3. Went through the OS4 quadrotor block diagram given to us and tried to understand them without much positive results.

My approach towards solving Q1:

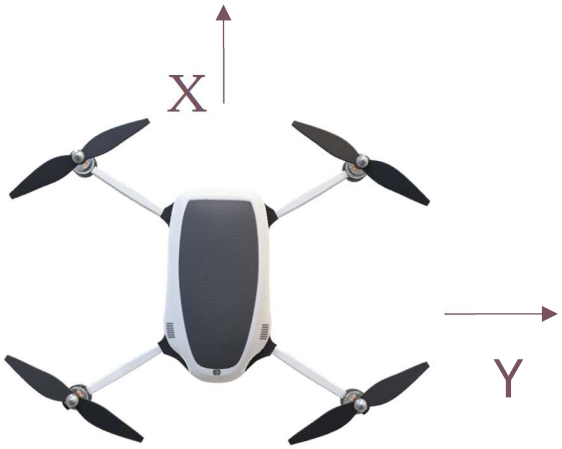
- Went through all the files given to us.
- Realized that I have to write the code on controller.m and run the simulation on runism.
- Initially I found out that for u1 = 0 and u2 = 0, the drone in free falling from the starting point.
- In between a meet in MS Teams, I was told that this u1 means thrust and u2 means moment.
- Then I immediately put u1 = mg
  [i.e. (params.m)*(params.g)], and found out that drone is still at a place constant.
- Now I could connect the dots and understood that u1 now is basically balancing the weight of quadrotor since it is the thrust.
- Also the error terms must be added to the desired acceleration for u1
- Now we get :
  U1 = m(g + des_acc + Kp*(Errorp) + Kv*(Errorv))
  Where, 1) des_acc = desired acceleration
  2) Errorp = Desired position – current position (proportional term Kp)
  3) Errorv = Desired velocity – current velocity (derivative term Kv)
  Also there there was nothing in system parameters for the integral term and hence I ignored that.

- Now, for u2 , we get that Moment = Ixx * (alpha). But for the stability of the drone in the single YZ plane, we would definitely want desired alpha to be 0.

- But again, the Errorp and Errorv terms must be added here too. So we get:
  U2 = Ixx *(0 + Kp*(Errorp) + Kv*(Errorv))
  Where, 1) Ixx = moment of Inertia along XX axis
  2) Errorp = desired angle - current angle
  [desired angle we need to find out, current angle is given to us in system parameters]
  3) Errorv = 0 – current omega [desired angular velocity should be zero because we want the final orientation of the drone same as initial orientation, hence no net change in angle wrt time ,i.e., zero Omega.]

- Now we just need to find the value of “desired angle”. Given orientations of the drone in the question, that angle must be necessarily a “roll angle” for movement along the YZ plane. I started by drawing the FBD at this stage

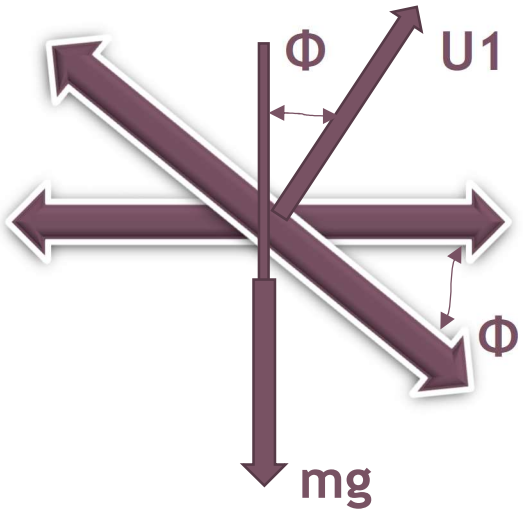
FBD of the quadrotor:



TOP VIEW

FRONT VIEW

Since, we are restricted to move in YZ plane:



F(thrust)=u1

This is the desired Orientation in order To negotiate a curve

$$u_1 \cos\phi = mg$$

$$u_1 \sin\phi = ma_y$$

$$So,\ \tan\phi = \frac{a_y}{g} = (des\_accn)/g$$

$$Finally, \phi = \arctan\left(\frac{des\_accn}{g}\right)$$

This Φ, is the value of the desired angle in the equation

U2 = Ixx *(0 + Kp*(Errorp) + Kv*(Errorv))

And Errorp = Φ - current angle

Now, for tuning the coefficients
I first set Kp=0 and Kv=0 in both equations and then increased them in steps of 10 and observed the behaviour of the drone on runism simulator. During the process I did keep in mind that the D coefficient Kv generally works for stabilizing.

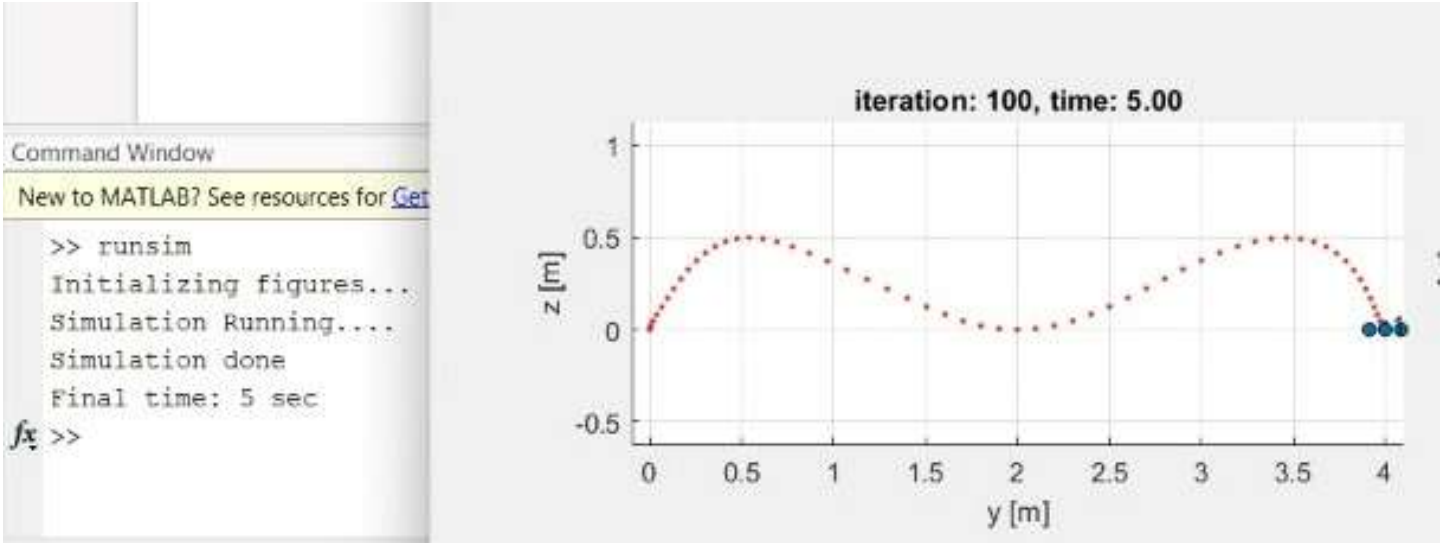
Still, it took me more than 3 hours to tune coefficients and for the sine trajectory to give correct result.

The coefficient which I set as 1050, works perfectly even if I set it 800 or 1200 or 1300. The reason for this I am still curious to know.
Anyways, I chose a value in the middle of this range.

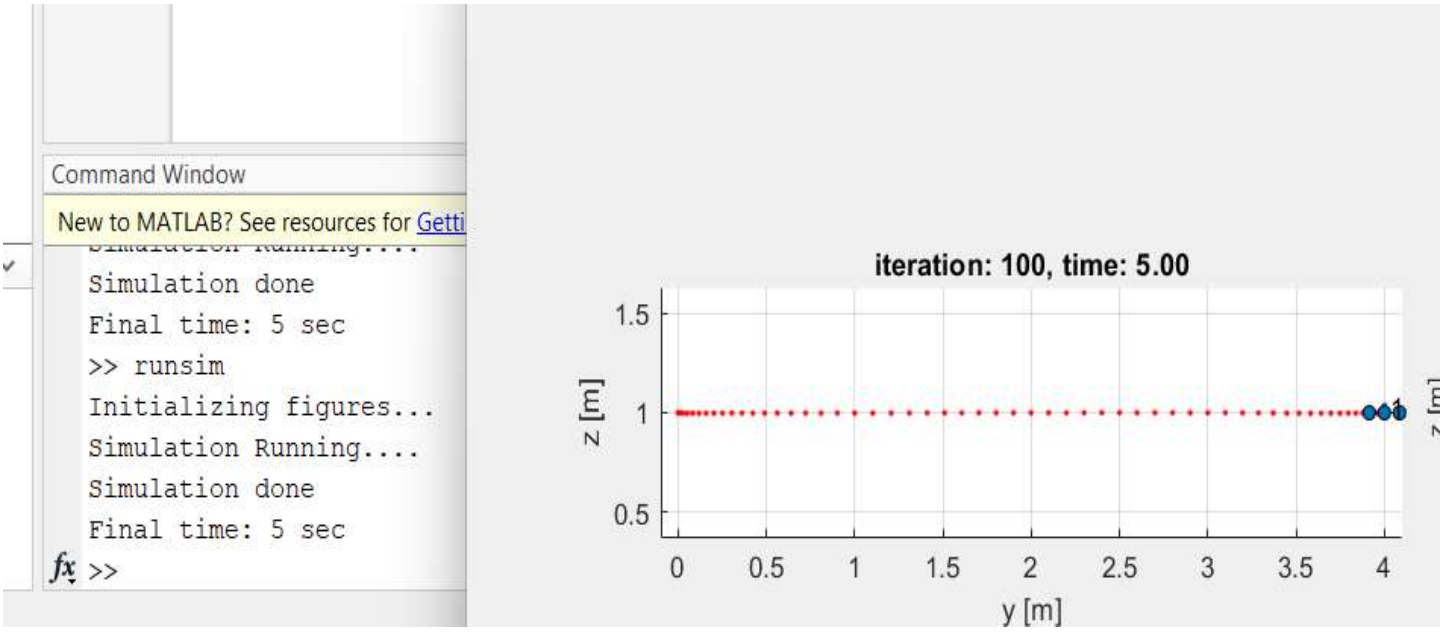
Henceforth, the code stands as follows(in the next page)

```
function [ u1, u2 ] = controller(~, state, des_state, params)
%CONTROLLER  Controller for the planar quadrotor
%
%   state: The current state of the robot with the following fields:
%   state.pos = [y; z], state.vel = [y_dot; z_dot], state.rot = [phi],
%   state.omega = [phi_dot]
%
%   des_state: The desired states are:
%   des_state.pos = [y; z], des_state.vel = [y_dot; z_dot], des_state.acc =
%   [y_ddot; z_ddot]
%
%   params: robot parameters

%   Using these current and desired states, you have to compute the desired
%   controls

error_y_vel = des_state.vel(1) - state.vel(1) ;
error_z_vel = des_state.vel(2) - state.vel(2) ;
error_y_pos = des_state.pos(1) - state.pos(1) ;
error_z_pos = des_state.pos(2) - state.pos(2) ;


u1 = (params.mass)*(params.gravity + des_state.acc(2)+ 140*error_z_pos +
20*error_z_vel) ;

%from FBD, we get u1cosphi = mg
%          and    u1sinphi = m*a , where a = plus/minus desired
acceleration
% dividing the equations; tanphi = a/g and so current angle is arctan(a/g)

angle = (-1)*atan((des_state.acc(1)+ 10*error_y_vel +
50*error_y_pos)/(params.gravity)) ;

% i multiplied minus 1 because
% i dont know whether the desired acceleration is towards left or right
% without multiplying minus 1,  the drone was moving to the opposite
% direction
error_rot = angle - state.rot;
error_omega = -state.omega; % desired omega should be zero

u2 = params.Ixx*( 60*error_omega + 1050*error_rot) ;

end
```

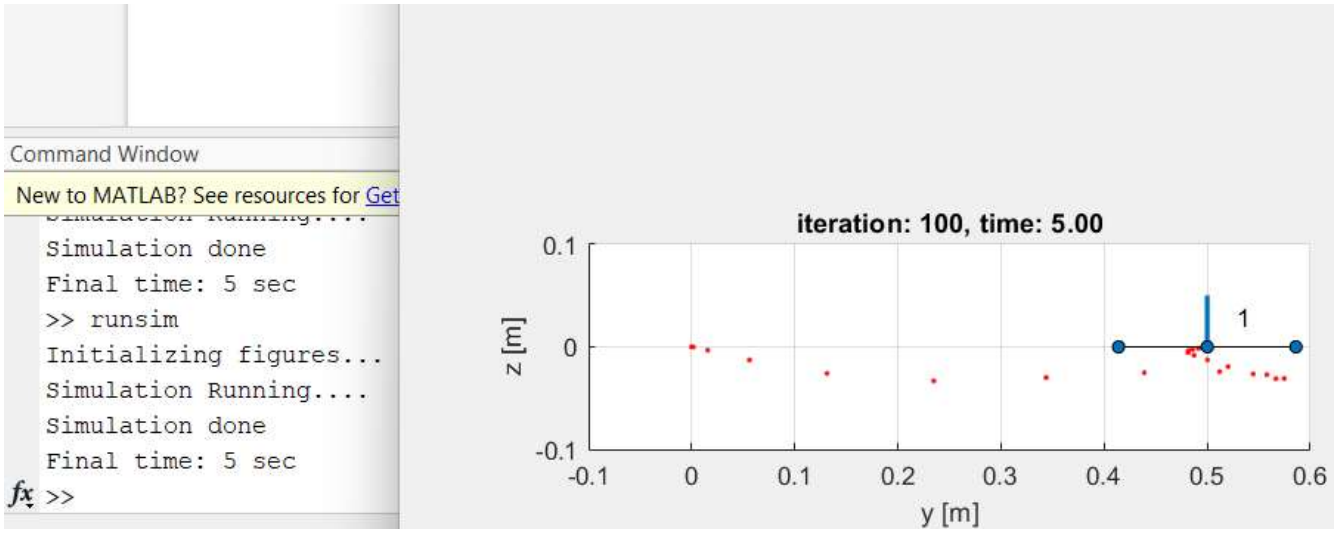
✓Working runism simulation of sine trajectory:



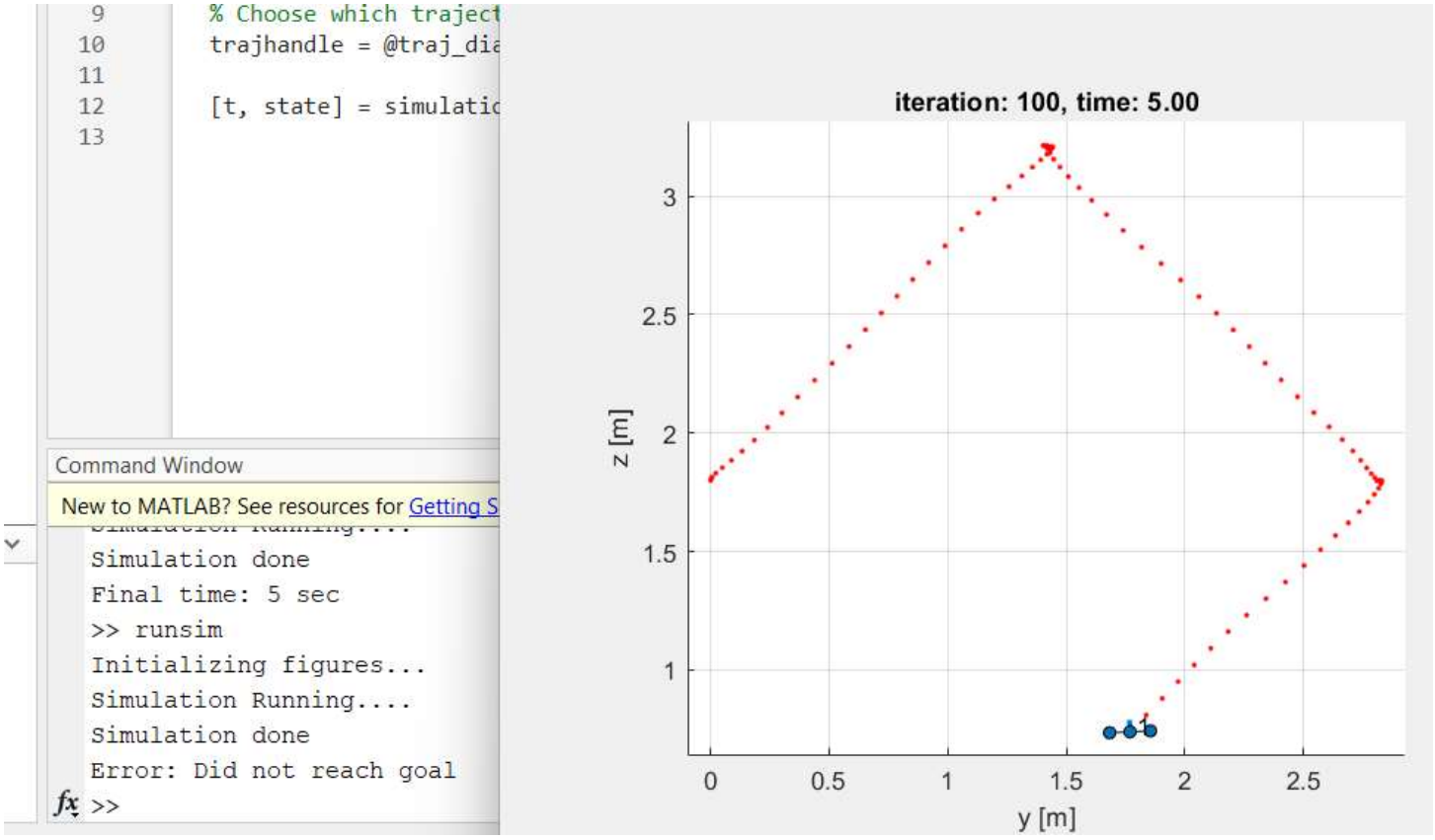
✓Working runism simulation of line trajectory:



✓Working runism simulation of step trajectory:



▪ Missing runsim simulation of diamond trajectory:



My effort in trying to solve problem 2:

I went through the documentations of problem 2 and 3 without much positive results.

All I could figure out was that in Problem 2 we were given the empty Simulink blocks of an OS4 quadrotor drone. I got to know of many equations from the documentation given, but was not sure of how to apply those in the empty Simulink blocks.

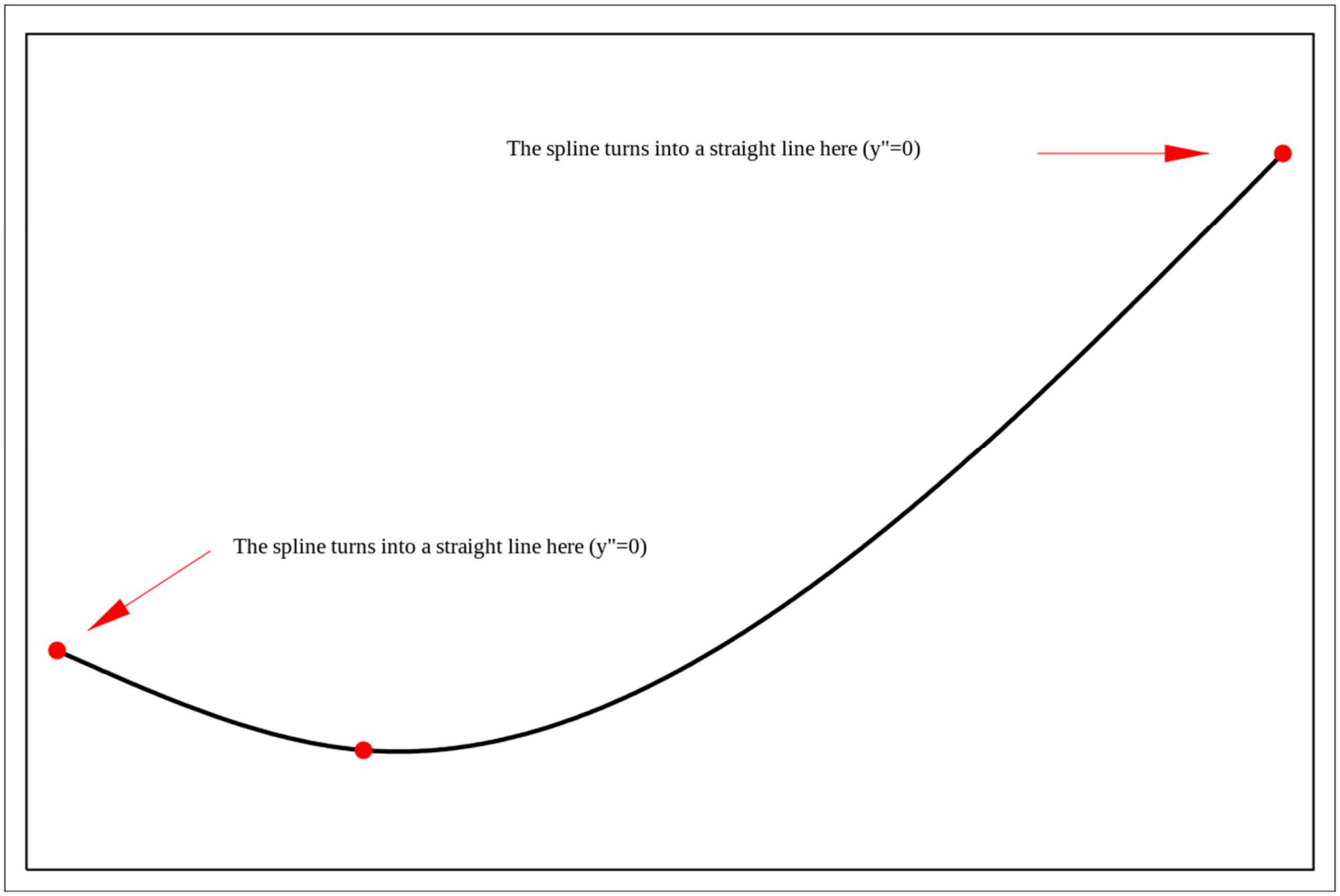
My effort in trying to solve problem 3:

Problem 3 was related to trajectory generation of problem 2 in some way, it asked to write code for Spline Interpolation, which is an interpolation method like those of the Numerical analysis.

On searching about this Spline Interpolation topic, I got to know that unlike other interpolation methods, Spline interpolation works in a bit different way.

Other interpolation methods fits a single high-degree polynomial to all of the values at once, whereas spline interpolation fits low-degree polynomials to small subsets of the values.

The following image from wikipedia gave me a graphical concept of Spline Interpolation:



The algorithm works like:
To make the spline take a shape that minimizes the bending (under the constraint of passing through all knots), we will define both and to be continuous everywhere, including at the knots. Thus the first and second derivatives of each successive polynomial must have identical values at the knots.

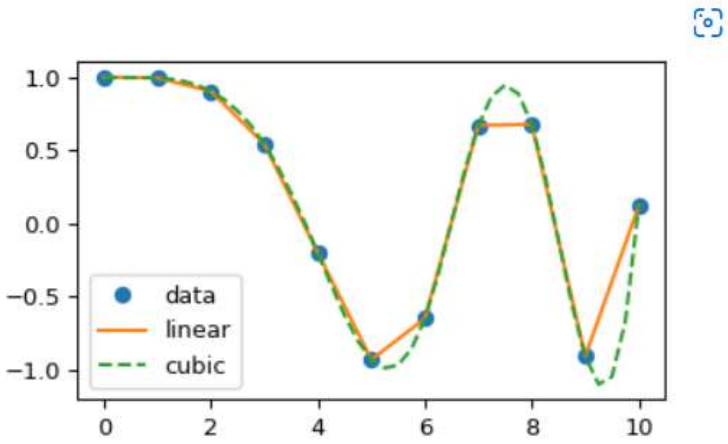
On further searching, I got to know that there is a ready made library in python to apply such interpolation formula: scipy.interpolate

Here is a snapshot of the python code in Jupyter interface.

```
>>> from scipy.interpolate import interp1d
```

```
>>> x = np.linspace(0, 10, num=11, endpoint=True)
>>> y = np.cos(-x**2/9.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
```

```
>>> xnew = np.linspace(0, 10, num=41, endpoint=True)
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
>>> plt.legend(['data', 'linear', 'cubic'], loc='best')
>>> plt.show()
```



Above all, these tasks were very interesting and enjoyable to work at =)

■ THE END