CENG 483 – BEHAVIORAL ROBOTICS

Homework Set #6 ***Due : 04.12.2023***

Name : Cem Tolga MÜNYAS

Student ID : 270206042

1. In the first question, we were given that the joint angles of the PUMA 560 6-DOF revolute manipulator's 6 joint angles.

a) In this part, we found out the final position (x,y,z) of the end effector of the robotic arm by using MATLAB Robotics Toolbox.

I have created a matrix and then converted the given degrees to radians. We need to use the model of PUMA 560 manipulator to complete the tasks.

mdl_puma560

Create model of Puma 560 manipulator

MDL_PUMA560 is a script that creates the workspace variable p560 which describes the kinematic and dynamic characteristics of a Unimation Puma 560 manipulator using standard DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY'configuration
qstretch	arm is stretched out in the X direction
qn	arm is at a nominal non-singular configuration

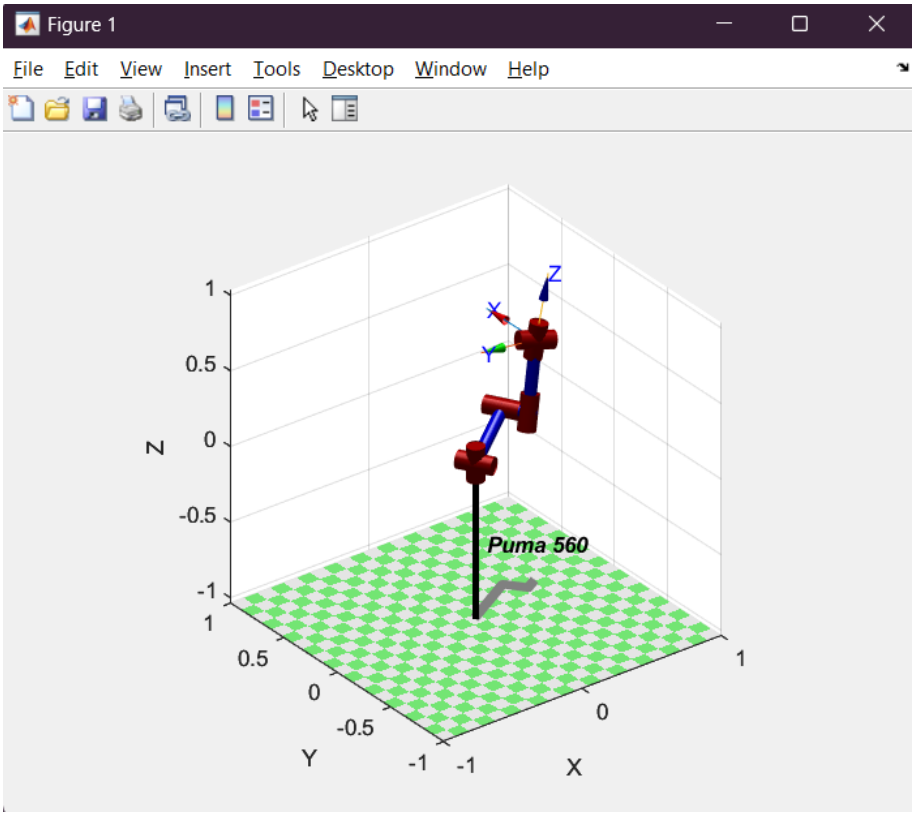
I used mdl_puma560 and p560 commands to create the DH parameters table. In this way, we could create the qr, qs and qz whose explanations are given below.

```
%% qn arm is at a nominal non-singular configuration

qn = [deg2rad(30), deg2rad(20), deg2rad(-35), deg2rad(45), deg2rad(10), deg2rad(20)];

%qz -> zero angle
%qr -> ready, the arm is straight and vertical
%qs -> stretch, the arm is straight and horizontal

p560.plot(qn);
```

Command Window

```
>> robotics_homework6
>> mdl_puma560
>> p560

p560 =

Puma 560 [Unimation]:: 6 axis, RRRRRR, stdDH, slowRNE
 - viscous friction; params of 8/95;
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|     1.5708|          0|
|  2|         q2|          0|     0.4318|          0|          0|
|  3|         q3|    0.15005|     0.0203|    -1.5708|          0|
|  4|         q4|     0.4318|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

With the help of plot function which is p560.plot(qn), I could create the 3D model according to the base as floor. Here is the pose of end effector.



In order to calculate the position of the end effector, I used p560.fkine(qn) function.

```
11  %% Forward Kinematics Calculation
12
13  TE = p560.fkine(qn)
```

Command Window

```
TE =
   -0.0664   -0.9815    0.1794    0.5402
    0.9965   -0.0744   -0.0382    0.1386
    0.0508    0.1763    0.9830    0.5595
         0         0         0         1
```

b) For a position (x,y,z) = (0.1, 0.5, -0.25) in task space, find all joint angles that can achieve such an end point in joint space of the Puma.

We know the end effector positions this time, and we were asked to find the joint angles. Firstly, we started with creating the T matrix by SE3(). Then we used the p560.ikine6s(T_matrix) function in order to calculate the inverse kinematics which are joint angles. Here is the process for finding the joint angles for (x,y,z) = (0.1, 0.5, -0.25)

```
15  %% Inverse Kinematics Calculation
16
17  T_matrix = SE3(0.1, 0.5, -0.25);
18  p560.ikine6s(T_matrix)
```

```
ans =

    4.2163   -3.5525    0.2447    3.1416    2.9754   -1.0747
```

c) Use the initial homogeneous transformation matrix of part (a) as initial position of the Puma and the final homogeneous matrix of part (b) as final position. Use the toolbox to create a joint trajectory using any of the trajectory generation commands in the toolbox.

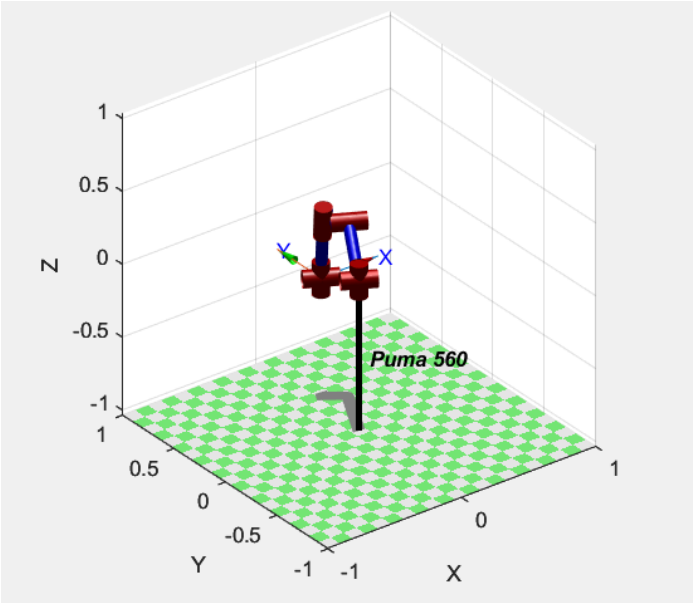
In this task, we were asked to create a joint trajectory using the initial homogeneous matrix of part(a) as initial position of the robot and the final homogeneous matrix of part (b) as final position of the robot. I have recreated both of the homogeneous matrices which we found earlier and used an interpolation function tpoly() to generate the trajectory.

```
%% Question 1.3

q1 = [deg2rad(30), deg2rad(20), deg2rad(-35), deg2rad(45), deg2rad(10), deg2rad(20)];
q2 = p560.ikine6s(T_matrix);
t = [0 : 0.2 : 2]';

trajectory = mtraj(@tpoly, q1, q2, t);
display(trajectory);
```

Each result in a 10 × 6 matrix trajectory with one row per time step and one column per joint. Here is the output.

```
trajectory =

    0.5236    0.3491   -0.6109    0.7854    0.1745    0.3491
    0.5552    0.3157   -0.6035    0.8056    0.1985    0.3369
    0.7375    0.1231   -0.5613    0.9219    0.3368    0.2666
    1.1258   -0.2872   -0.4713    1.1696    0.6313    0.1169
    1.6958   -0.8895   -0.3393    1.5333    1.0636   -0.1029
    2.3699   -1.6017   -0.1831    1.9635    1.5749   -0.3628
    3.0441   -2.3140   -0.0269    2.3936    2.0863   -0.6227
    3.6141   -2.9163    0.1052    2.7573    2.5186   -0.8425
    4.0024   -3.3265    0.1951    3.0051    2.8131   -0.9922
    4.1847   -3.5191    0.2374    3.1214    2.9514   -1.0625
    4.2163   -3.5525    0.2447    3.1416    2.9754   -1.0747
```

I also displayed the animation of the robot with the command *p560.plot(trajectory).*



This demonstration shows the final position of the robot.

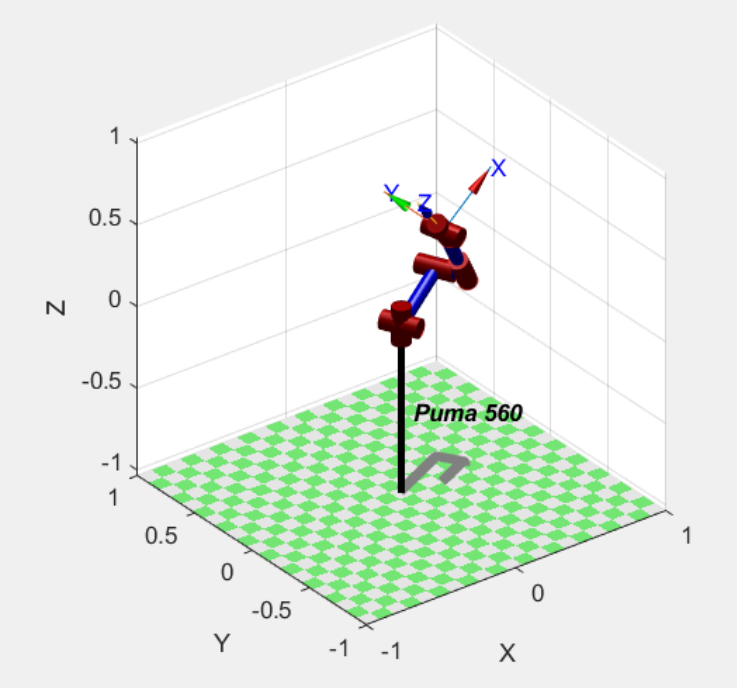
d) Using the initial joint angles of θ = [0, 90, 25, 0, 30, 20], T and a torque vector of τ = [1, 0, 0, 0, 0, 0.5] simulate the Puma 560 on the toolbox and plot the 3-D trajectory of the robot for 5 different instances of time in the range of $0 \leq t \leq 10$.

For the Puma 560 robot in its nominal pose, if showing torque is applied, which results in joint torques is shown below.

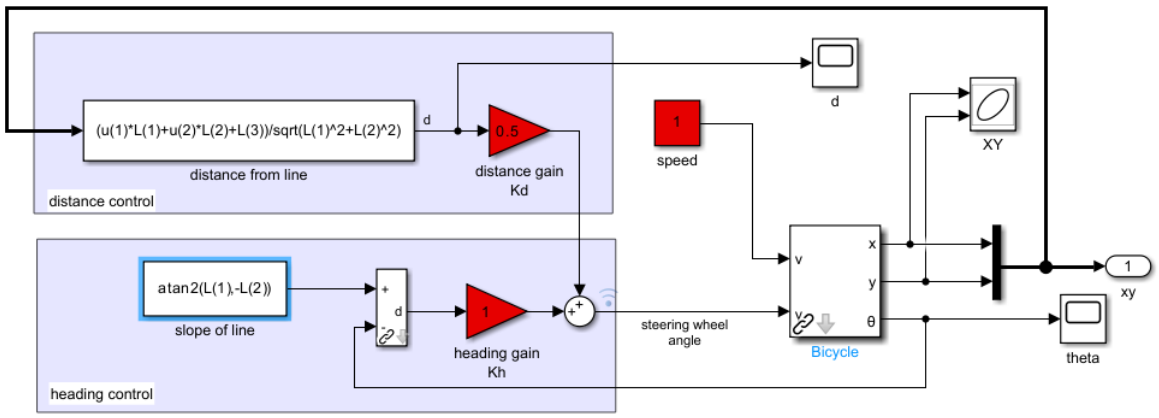
```
%% Question 1.4

q0 = [deg2rad(0), deg2rad(90), deg2rad(25), deg2rad(0), deg2rad(30), deg2rad(20)];
torque = [1, 0, 0, 0, 0, 0.5];
time = [0 : 2 : 10]';

tau = p560.jacob0(q0)' * torque';
display(tau');
traj = mtraj(@tpoly, q0, tau', time);

figure(2);
p560.plot(traj);
```

```
>> tau'

ans =

    0.6501   -0.2677    0.1641   -0.2113    0.0000   -0.4096
```

Here is the simulation of the Puma 560 and the 3-D trajectory of the robot for 5 different instances of time in the range of 0 0 ≤ t ≤ 10. The figure is cut from the action of our Puma 560 robot.



2. Consider the *sl_driveline* command in section 4.2.2. We wish to create a map of 10 x 10 like Figure 4.9 and draw 2 lines as y = -2x + 10 and y = 2x -10 on this map. These lines will divide the (x,y) plane into 3 segments. Use Simulink and bicycle command and at least one pose in each of the 3 regions. Discuss the results and justify your answers.

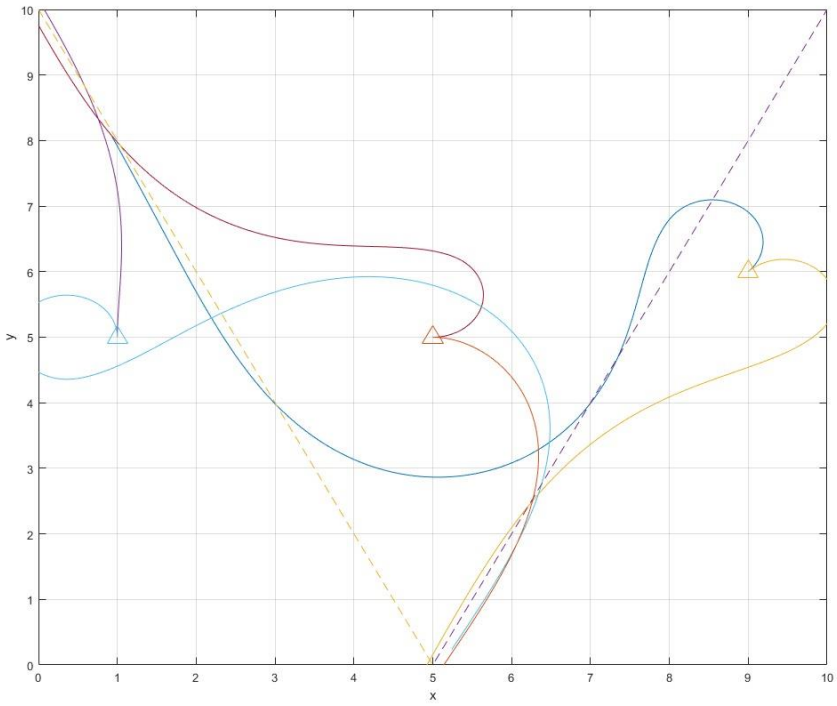


We need to change some parameters from the workspace in order to observe what is happening for our lines y = -2x + 10 and y = 2x -10.

Here is the code for corresponding processes.

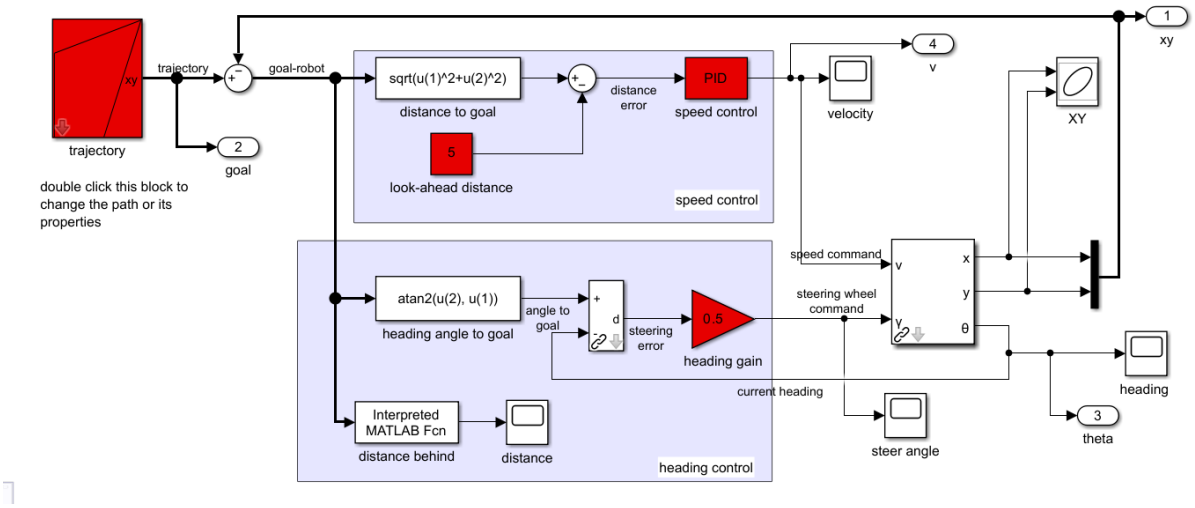
```
%% Question 2

% Run it with the L = [-2, 1, 10] also
L = [-2, 1, 10];
x = [0 : 10];
y1 = -2*x + 10;
y2 = 2*x - 10;

% Then we need to define 3 poses for each 3 of the regions
poses = [9, 6, pi/4];
poses(:, :, 2) = [1, 5, pi/2];
poses(:, :, 3) = [5, 5, 0];

for i = 1 : 3
    x0 = poses(:, :, i);

    % Finally simulate the motion
    r = sim('sl_driveline');

    % Extract the y as a function of time
    qy = r.find('y');
    qx = r.find('x');

    % Plot and hold the trajectories
    plot(qy(:,1),qy(:,2), "-^", "MarkerIndices", [1], "MarkerSize",15); grid on; xlim([0,10]); ylim([0,10]);
    hold on; plot(x, y1, "--"); hold on; plot(x, y2, "--");
    xlabel("x"); ylabel("y");

end
```

Here, we divided the 10x10 map to 3 regions, and determined one pose from each of them. The output is the path for each pose and is shown below.



3. Consider the *sl_pursuit* command in section 4.2.3. We wish to create a map of 10 x 10 like Figure 4.11a and use 4 initial poses from the 4 corners of the map and one pose inside the circular path. Plot all x-y and speed vs time trajectories. Discuss the results and justify your answers.

Here, we will be using the Simulink model "pursuit". The Simulink model *sl_pursuit* drives the vehicle along a piecewise linear trajectory. The model includes a target that moves at constant velocity along a piecewise linear path defined by a number of waypoints. I began with initializing the model in Simulink by typing *sl_pursuit* to the command window. Here is the model.

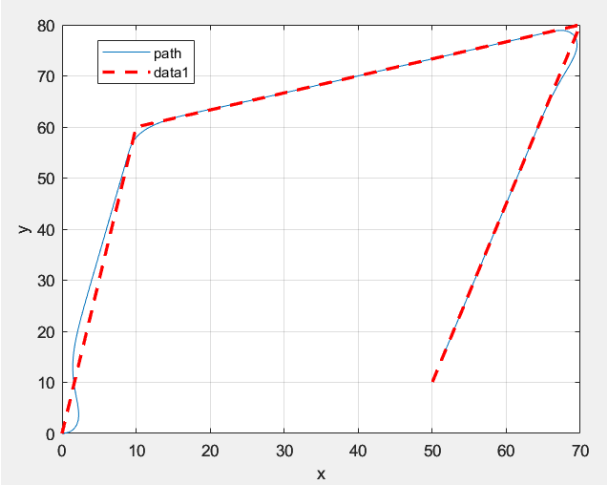


I run the simulation with the command *r = sim("sl_pursuit").*

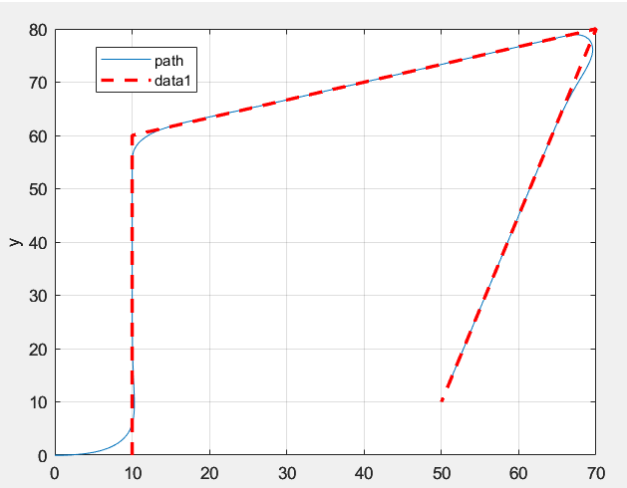
```
r = sim("sl_pursuit");
q = r.find("y");
t = r.find("t");
plot(q(:,1), q(:,2)); % the vehicle's path in the plane
display(r);
display(t);
```

We also want the x-y and speed vs time trajectories. So, I plotted the path of our vehicle as well.

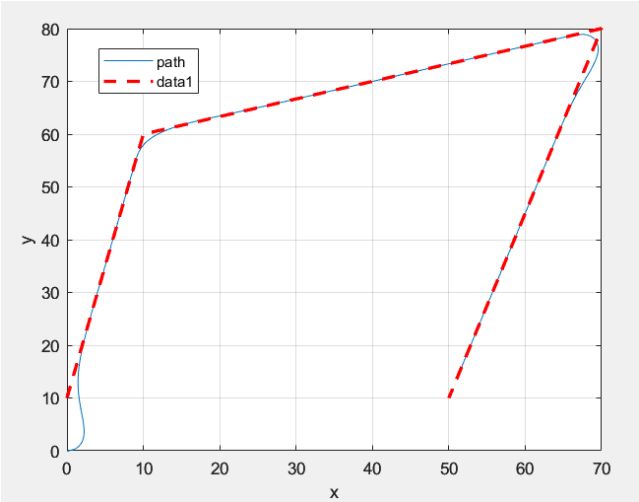
We will be changing the initial positions rapidly in order to observe the affects of the movement. The initial positions for every robot will be the origin point (0,0). Every figure represents the path of the robot in the x-y plane. The red dashed line is the path to be followed and the blue line in the path followed by the robot. Here are the outputs for the initial positions (0,0), (10,0), (0,10) and (10,10) respectively.



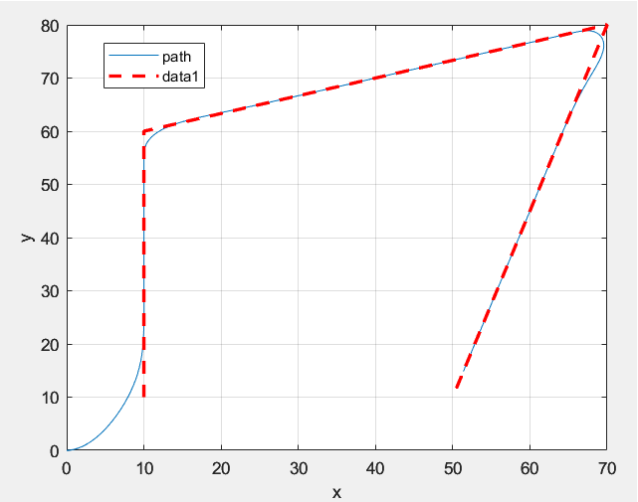
Initial position : (0,0)



Initial position : (10,0)

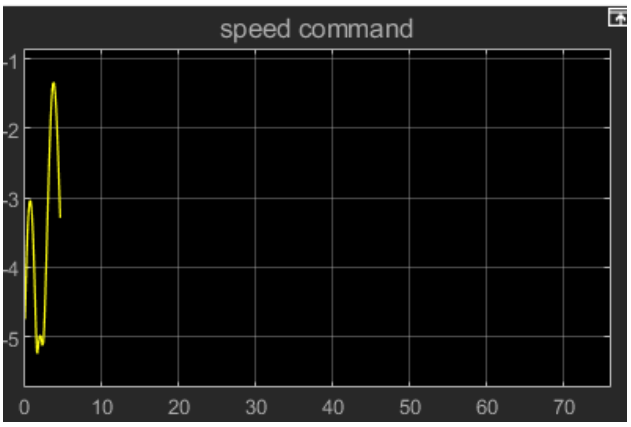


Initial position : (0,10)



Initial position : (10,10)

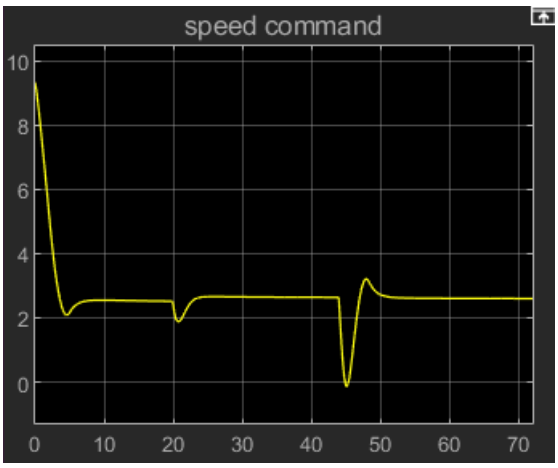
Here are the speed versus time trajectories for each starting position starts with (0,0), (10,0), (0,10) and (10,10).



Speed versus time for initial point (0,0)

Speed versus time for initial point (10,0)

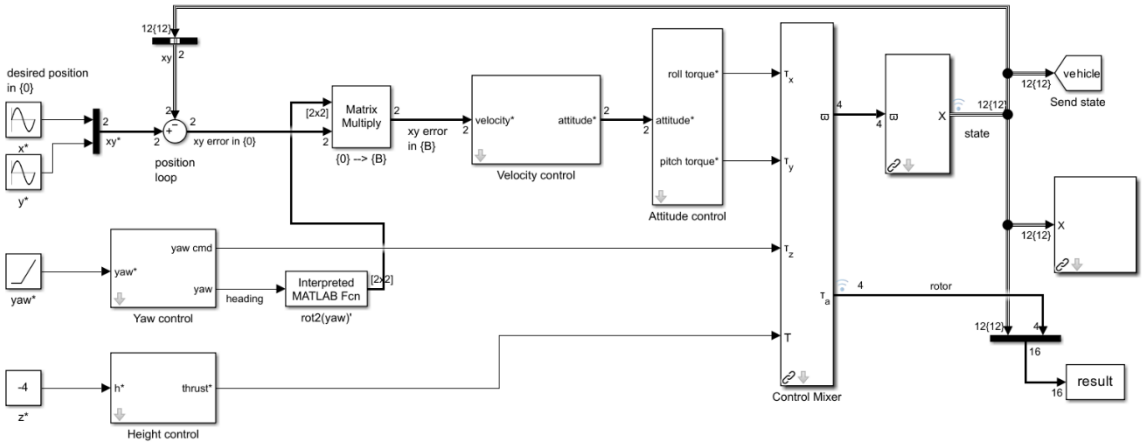
Speed versus time for initial point (0,10)



Speed versus time for initial point (10,10)

4. Consider the flying robots section 4.3. Repeat exercise starting from page 83. Repeat the exercise and use different set of initial conditions and scenarios in *sl_quadrotor* command. Discuss the results and justify your answers.

Here, we have created a model for a quadrotor flying vehicle. They are now widely used in many applications. First of all, I started with initializing the Simulink model *sl_quadrotor* from the command window. The Simulink page showed up and I analyzed it deeply to understand the functions of the blocks.



After that, with the command *mdl_quadrotor,* I loaded up the parameters of our quadrotor which created a structure called quadrotor in the workspace. The elements are the dynamic properties of the quadrotor.

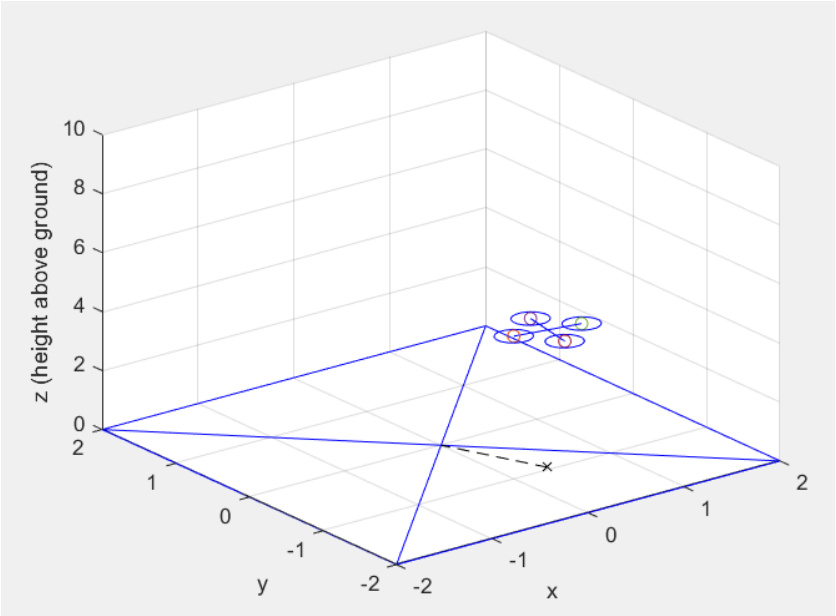
Variables - quadrotor

quadrotor | quadrotor.theta0

1x1 struct with 33 fields

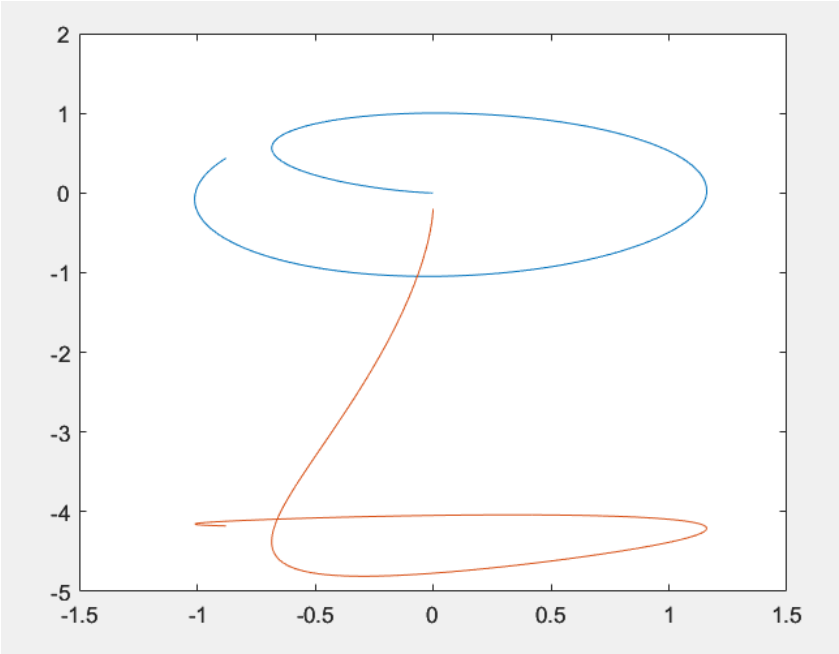
Field	Value
nrotors	4
g	9.8100
rho	1.1840
muv	1.5000e-05
M	4
J	[0.0820,0,0;0,0.08...
h	-0.0070
d	0.3150
nb	2
r	0.1650
c	0.0180
e	0
Mb	0.0050
Mc	0.0100
ec	0.0040
Ib	3.2401e-05
Ic	4.0000e-08
mb	0.0042
Ir	6.4883e-05
Ct	0.0048
Cq	2.3515e-04
sigma	0.8000
thetat	0.1187
theta0	0.2548
theta1	-0.1361
theta75	0.9000
thetai	Inf
a	5.5000
A	0.0855
gamma	2.6781
b	1.3234e-05
k	1.0697e-07
verbose	0

Here are the variables in the workspace. We can clearly observe and change some of the values of parameters in order to observe the changes in the simulation.



I run the simulation using the run button in Simulink and it displayed and animation in another window. The quadrotor vehicle lifts off and flies around a circle while spinning slowly about its own z-axis. I changed some of the parameters from the workspace such as theta0, theta1, theta75, sigma and A. In general, it looked like those parameters effects the speed of rotation of the vehicle as well as velocity of spinning about its own axis.

Finally, I wrote to the command window *plot(result(:,1), result(:,2:3));* which shows us the control of the quadrotor. A position error results in a required translational velocity.



REFERENCES

- **Ozan Gülbaş**

- **Boğaçhan Düşgül**

- **Robotics, Vision, and Control Fundamental Algorithms in MATLAB®, Peter CORKE**