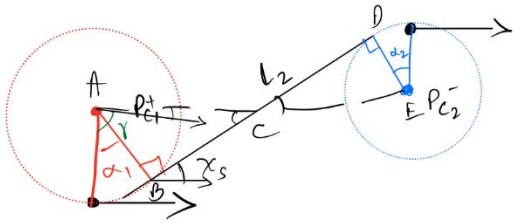
AE5335 Assignment 3

Submitted by: Purna Patel, Ritik Jain

Problem 1:

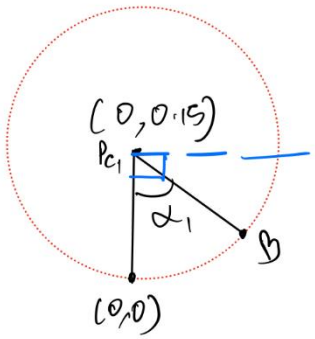
Question 1



$R_{min} = 0.15$ km $\quad x_0 = (0,0)$

Using the solution from previous assignment we find the plane that differentiates the $C^+ S C^-$ curves.

We start with finding the point which joins C^+ & S paths, ie point B



$\alpha_1 = 0.859$ rad

$$P_{c_1} = \begin{bmatrix} 0 \\ 0.15 \end{bmatrix}$$

$x = x_c + r\cos\theta \quad y = y_c + r\sin\theta$

θ for Point $P_0 = -\pi/2$

θ for Point B $= -(\pi/2 - \alpha_1) = -0.711$ rad

$x = 0.114 \quad y = 0.052$

$$B = \begin{bmatrix} 0.114 \\ 0.052 \end{bmatrix}$$

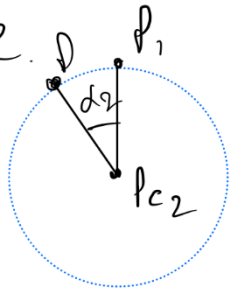
The line joining P_{c_1} & B seperates the curve and straight path.

The eq of line P_C, B is

$$\frac{y - 0.15}{x} = \frac{-0.098}{0.114}$$

$$0.86x + y - 0.15 = 0 \quad -(1)$$

Similarly we solve for the line seperating the straight path and c curve.



$$P_{C_2} = \begin{bmatrix} 1 \\ 0.85 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\alpha_2 = 0.859 \text{ rad}$$

$$x = x_c + r\cos\theta \quad y = y_c + r\sin\theta$$

θ for Point $P_0 = \pi/2$

θ for Point $D = (\pi/2 + \alpha_2) = 2.429_{rad}$

$x = 0.886 \quad y = 0.948$

$$D = \begin{bmatrix} 0.886 \\ 0.948 \end{bmatrix}$$

The line joining Pc_2 & D seperates the curve and straight path.

The eq of line Pc_2 D is

$$\frac{y - 0.85}{x - 1} = \frac{0.036}{-0.052}$$

$$y - 0.85 = -0.69x + 0.69$$

$$0.69x + y - 1.54 = 0 \quad -(2)$$

if $0.86x + y - 0.15 < 0$

the path to be followed is C^+

else if $0.86x + y - 0.15 > 0$ &

$0.69x + y - 1.54 < 0$

the path to be followed is a straight path

else if $0.69x + y - 1.54 > 0$

the path to be followed is C^-

For C^+ path we use

$e = p - p_{ref,C}$ where $p_{ref,C}$ is the center of the circle

$$x_{ref} = x_0 + \left[\frac{\pi}{2} + \tan^{-1}\left(k_2\left(\frac{\|e\| - R}{R}\right)\right)\right]$$

where x_0 is the angle made by the line $P, P_{ref,C}$ & R is the turning radius

For the straight line S, we find the point on the line Pref that forms a perpendicular between the path & line P, Pref

It is found by using the slope and solving the eq of both line simultaneously.

$$e_y^d = \begin{bmatrix} \cos x_0 & \sin x_0 \\ -\sin x_0 & \cos x_0 \end{bmatrix} (P - P_{ref})$$

where x_0 is slope of path

$$x_{ref} = x_0 - x_\infty \frac{2}{\pi} \tan^{-1}(k_1 e_y^d)$$

Eq of line BD

$$B = \begin{bmatrix} 0.114 \\ 0.052 \end{bmatrix} \qquad D = \begin{bmatrix} 0.886 \\ 0.948 \end{bmatrix}$$

$$\text{slope} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0.886 - 0.114}{0.948 - 0.052}$$

$$= 0.8616$$

$$y - 0.114 = 0.8616(x - 0.052)$$

$$x - y + 0.0692 = 0$$

slope of line $\perp$ to BD

$$m_1 \times m_2 = -1$$

$$m_2 = -1.1606$$

For C^+ path we use

$e = p - p_{ref,c}$ where $p_{ref,c}$ is the center of the circle

$$x_{ref} = x_0 + \left[\frac{\pi}{2} - \tan^{-1}\left(k_2\left(\frac{\|e\| - R}{R}\right)\right)\right]$$

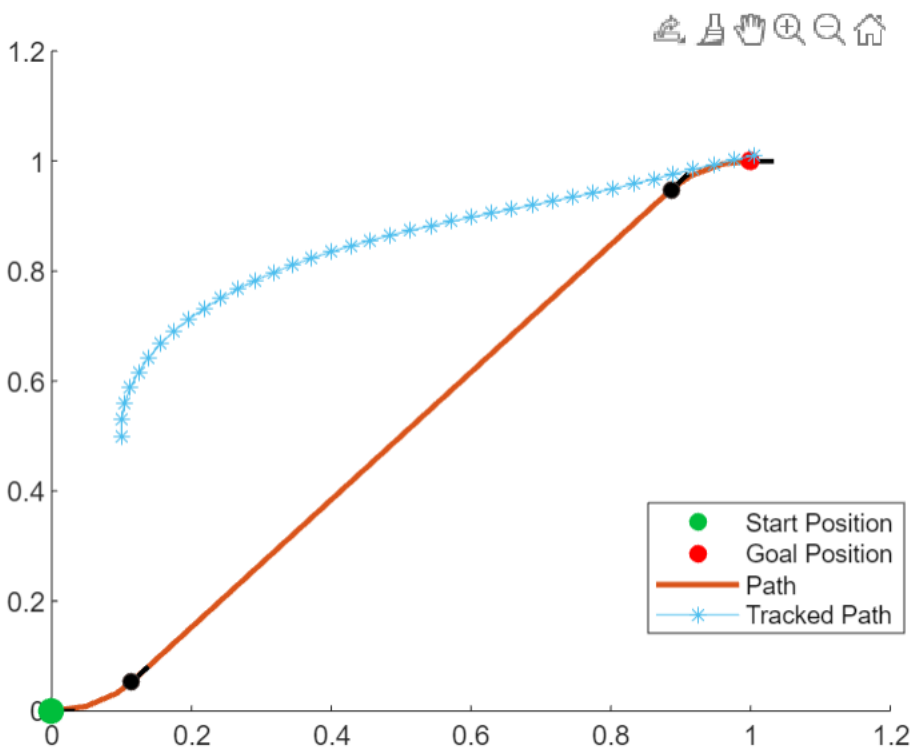
where x_0 is the angle made by the line $P, P_{ref,c}$ & R is the turning radius

For C^+ we use $k_2 = 0.01$

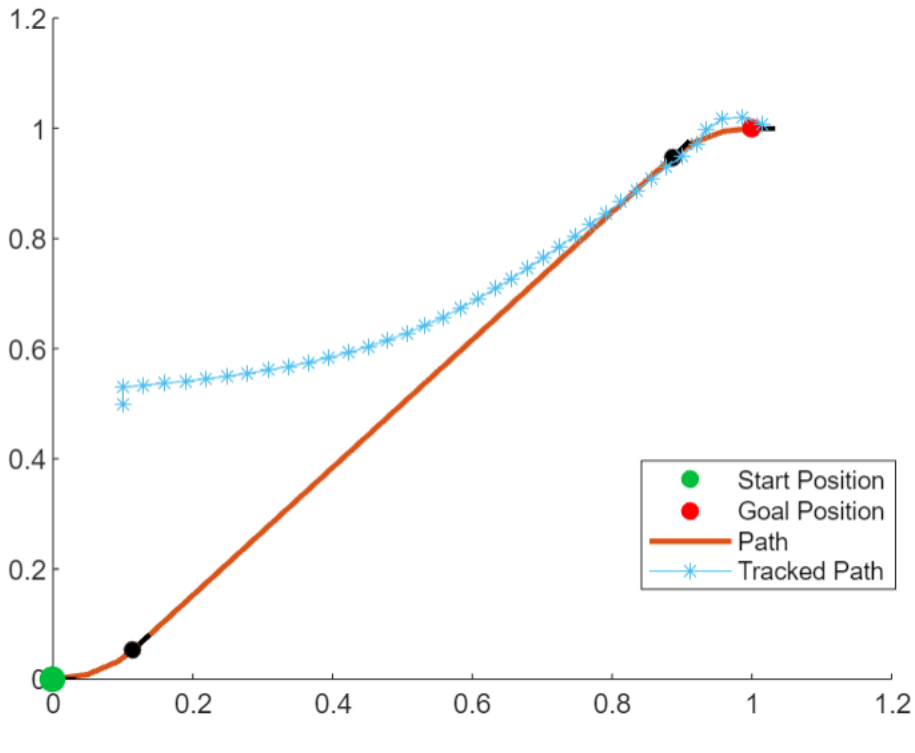
For S path we use $x_\infty = \pi/4$ $k_1 = 10$

For C^- we use $k_2 = 5$

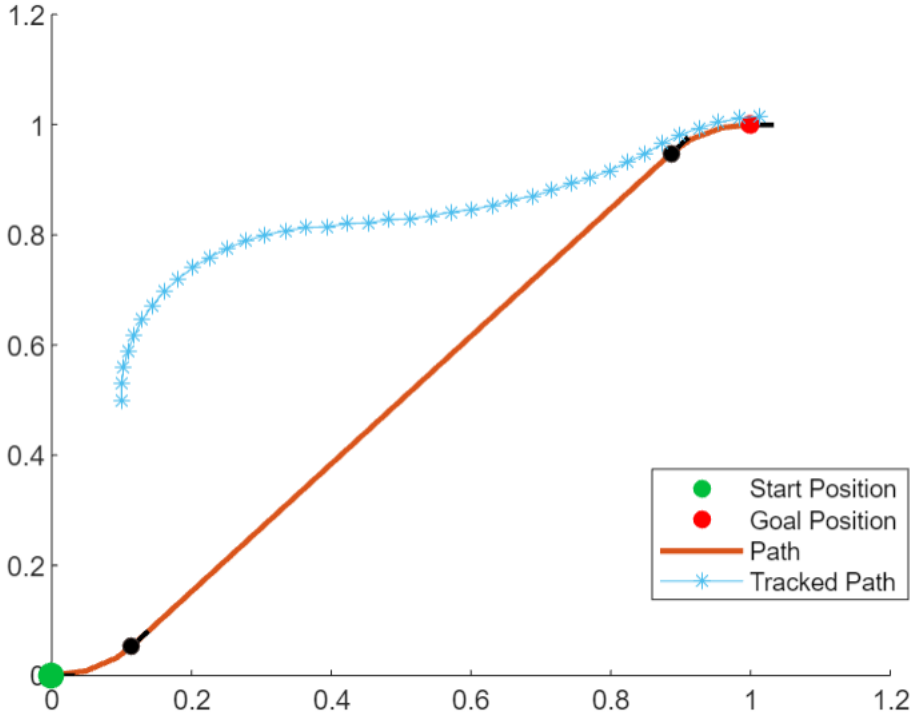
For $\alpha = 5$, the turns were too overshooting so we had to limit the x_dot in the range of -0.1 to 0.1



Path for alpha 0.1



Path for alpha 1



Path for alpha 5

Code is attached in the appendix

Problem 2:

Calculating desired states from the given trajectory:

- Given trajectory

$$p_t = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4$$

- Differentiating the above equation to get desired velocity, acceleration, jerk, and snap.

$$\dot{p}_t = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3$$
$$\ddot{p}_t = 2a_2 + 6a_3 t + 12a_4 t^2$$
$$\dddot{p}_t = 6a_3 + 24a_4 t$$
$$\frac{d\dddot{p}_t}{dt} = 24a_4$$

- Yaw angle is equal to the heading direction of the trajectory.

$$\psi = \tan^{-1}\frac{\dot{p}_y}{\dot{p}_x}$$

$$\dot{\psi} = \frac{\dot{p}_x\ddot{p}_y - \dot{p}_y\ddot{p}_x}{\dot{p}_x^2 + \dot{p}_y^2}$$

$$\ddot{\psi} = \frac{\left(\dot{p}_x^2 + \dot{p}_y^2\right)\left(\dot{p}_x\dddot{p}_y - \dot{p}_y\dddot{p}_x\right) - \left(\dot{p}_x\ddot{p}_y - \dot{p}_y\ddot{p}_x\right)\left(2\dot{p}_x\ddot{p}_x - 2\dot{p}_y\ddot{p}_y\right)}{\left(\dot{p}_x^2 + \dot{p}_y^2\right)^2}$$

- Step 1:

$$F_t = m\ddot{p} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

$$F = \|F_t\| = m\sqrt{\ddot{p}_x^2 + \ddot{p}_y^2 + (\ddot{p}_z - g)^2}$$

$$\hat{k}_b^t = -\frac{F_t}{F}$$

- Step 2:

$$\hat{j}_{a1}^t = \begin{bmatrix} -\sin\psi \\ \cos\psi \\ 0 \end{bmatrix}$$

$$\hat{\imath}_b^t = \frac{\hat{j}_{a1}^t \times \hat{k}_b^t}{\left\|\hat{j}_{a1}^t \times \hat{k}_b^t\right\|}$$

$$\hat{j}_b^t = \hat{k}_b^t \times \hat{\imath}_b^t$$

- Step 3:

$$R_b^t = [\hat{\imath}_b^t \quad \hat{j}_b^t \quad \hat{k}_b^t]$$

$$R_{temp} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} R_b^t = \begin{bmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

$$\theta = \tan^{-1}\left(\frac{-R_{temp}(3,1)}{R_{temp}(1,1)}\right)$$

$$\phi = \tan^{-1}\left(\frac{-R_{temp}(2,3)}{R_{temp}(2,2)}\right)$$

- Step 4

$$p = \frac{1}{F}(mR_t^b \ddot{p}_t) \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$q = -\frac{1}{F}(mR_t^b \ddot{p}_t) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

- Step 5:

$$\begin{bmatrix} r \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = [-\hat{k}_b^t \quad \hat{j}_{a1}^t \quad \hat{\imath}_b^t]^{-1} \left(R_b^t \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} - \dot{\psi} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)$$

$$\omega_{tb}^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

- Step 6: we know,

$$F = \|F_t\| = m\sqrt{\ddot{p}_x^2 + \ddot{p}_y^2 + (\ddot{p}_z - g)^2}$$

$$\dot{F} = \frac{m^2\left(\ddot{p}_x\dddot{p}_x + \ddot{p}_y\dddot{p}_y + (\ddot{p}_z - g)\dddot{p}_z\right)}{F}$$

$$\ddot{F} = \frac{m^2\left(\ddot{p}_x^2 + \ddot{p}_x\frac{d\dddot{p}_x}{dt} + \ddot{p}_y^2 + \ddot{p}_y\frac{d\dddot{p}_y}{dt} + \ddot{p}_z^2 + (\ddot{p}_z - g)\frac{d\dddot{p}_z}{dt}\right) - \dot{F}^2}{F}$$

$$F_b = \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \qquad \dot{F}_b = \begin{bmatrix} 0 \\ 0 \\ -\dot{F} \end{bmatrix} \qquad \ddot{F}_b = \begin{bmatrix} 0 \\ 0 \\ -\ddot{F} \end{bmatrix}$$

$$\begin{bmatrix} -\dot{q} \\ \dot{p} \\ 0 \end{bmatrix} = \frac{1}{F}\left(mR_t^b \frac{d\ddot{p}}{dt} - \ddot{F}_b - 2\omega_{tb}^b \times \dot{F}_b - \omega_{tb}^b \times \omega_{tb}^b \times F_b \right)$$

- Step 7:

We have

$$\omega_{tb}^t = \dot{\psi}\hat{k}_t^t + \dot{\theta}\hat{\jmath}_{a1}^t + \dot{\phi}\hat{\imath}_b^t$$

$$\dot{\omega}_{tb}^t = \ddot{\psi}\hat{k}_t^t + \ddot{\theta}\hat{\jmath}_{a1}^t + \dot{\theta}\dot{\hat{\jmath}}_{a1}^t + \ddot{\phi}\hat{\imath}_b^t + \dot{\phi}\dot{\hat{\imath}}_b^t$$

We know that,

$$\hat{\jmath}_{a1}^t = \begin{bmatrix} -\sin\psi \\ \cos\psi \\ 0 \end{bmatrix}$$

$$\dot{\hat{\jmath}}_{a1}^t = \begin{bmatrix} -\cos\psi \\ -\sin\psi \\ 0 \end{bmatrix}$$

From the general form of R_b^t, we can say that

$$\hat{\imath}_b^t = \begin{bmatrix} \cos\psi\cos\theta \\ \cos\theta\sin\psi \\ -\sin\theta \end{bmatrix}$$

$$\dot{\hat{\imath}}_b^t = \begin{bmatrix} -\dot{\psi}\sin\psi\cos\theta - \dot{\theta}\cos\psi\sin\theta \\ -\dot{\theta}\sin\theta\sin\psi + \dot{\psi}\cos\theta\cos\psi \\ -\dot{\theta}\cos\theta \end{bmatrix}$$

Now, we know

$$\omega_{tb}^t = R_b^t \omega_{tb}^b$$

$$\dot{\omega}_{tb}^t = R_b^t\left(\dot{\omega}_{tb}^b + \omega_{tb}^b \times \omega_{tb}^b\right) = R_b^t \dot{\omega}_{tb}^b$$

$$\ddot{\psi}\hat{k}_t^t + \ddot{\theta}\hat{\jmath}_{a1}^t + \dot{\theta}\dot{\hat{\jmath}}_{a1}^t + \ddot{\phi}\hat{\imath}_b^t + \dot{\phi}\dot{\hat{\imath}}_b^t = R_b^t \begin{bmatrix} \dot{p} \\ \dot{q} \\ 0 \end{bmatrix} + \dot{r} R_b^t \hat{k}_b^b$$

$$\begin{bmatrix} \dot{r} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} = [-\hat{k}_b^t \quad \hat{\jmath}_{a1}^t \quad \hat{\imath}_b^t]^{-1} \left(R_b^t \begin{bmatrix} \dot{p} \\ \dot{q} \\ 0 \end{bmatrix} - \ddot{\psi} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \dot{\theta}\dot{\hat{\jmath}}_{a1}^t - \dot{\phi}\dot{\hat{\imath}}_b^t \right)$$

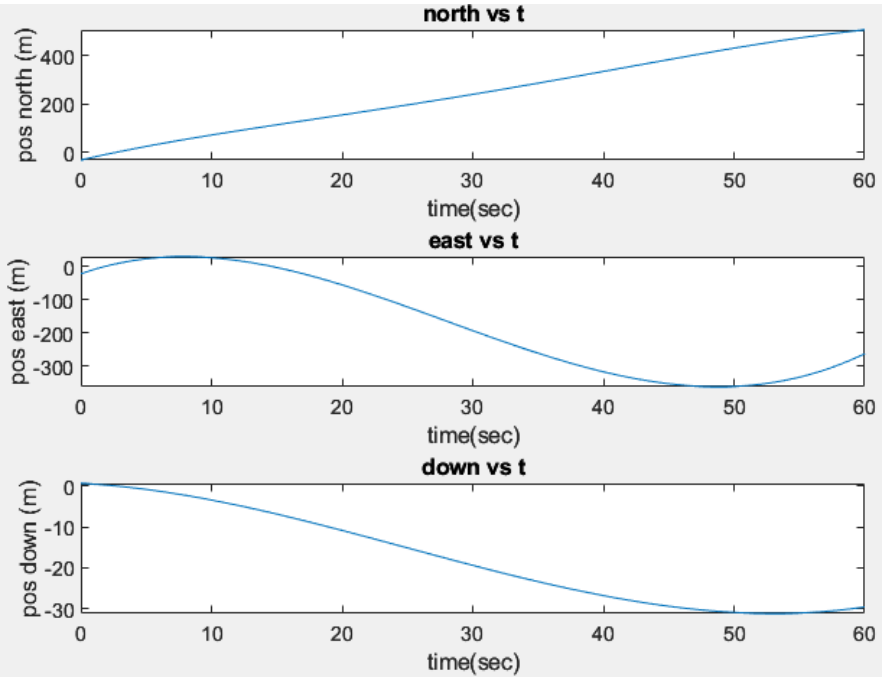
$$\dot{\omega}_{tb}^{b} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}$$

$$M^{b} = J\dot{\omega}_{tb}^{b} + \omega_{tb}^{b} \times J\omega_{tb}^{b}$$

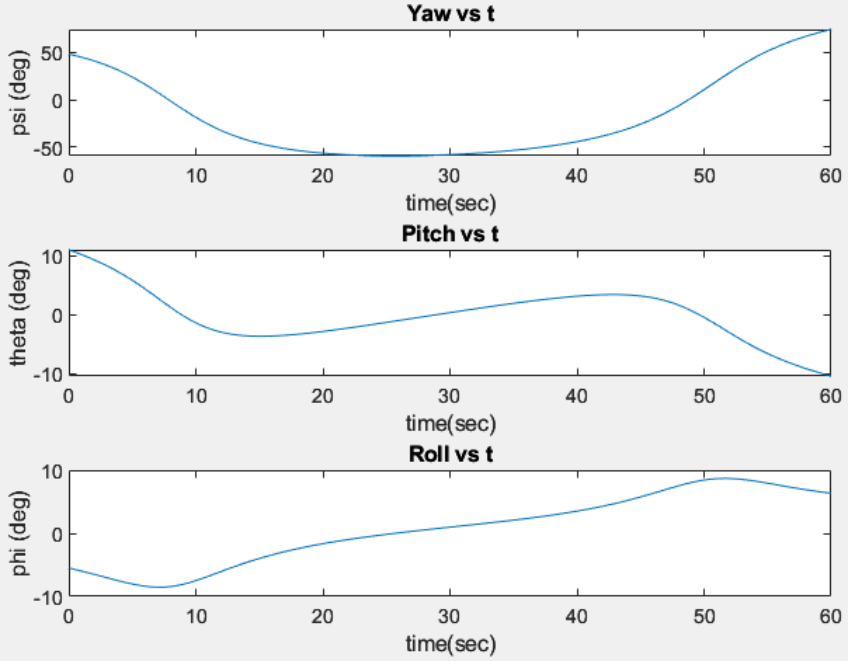
$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} -k_f & -k_f & -k_f & -k_f \\ 0 & -k_f L & 0 & k_f L \\ k_f L & 0 & -k_f L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix}^{-1} \begin{bmatrix} -F \\ M^{b} \end{bmatrix}$$

Result plots:

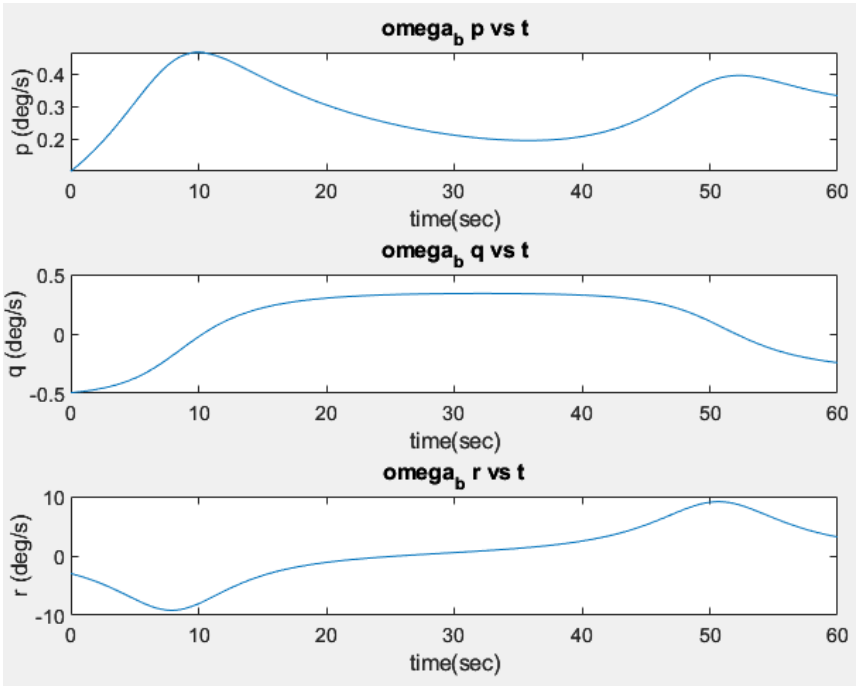
- Given trajectory:
  Following graphs shows the position vs time plots of the given trajectory for reference.



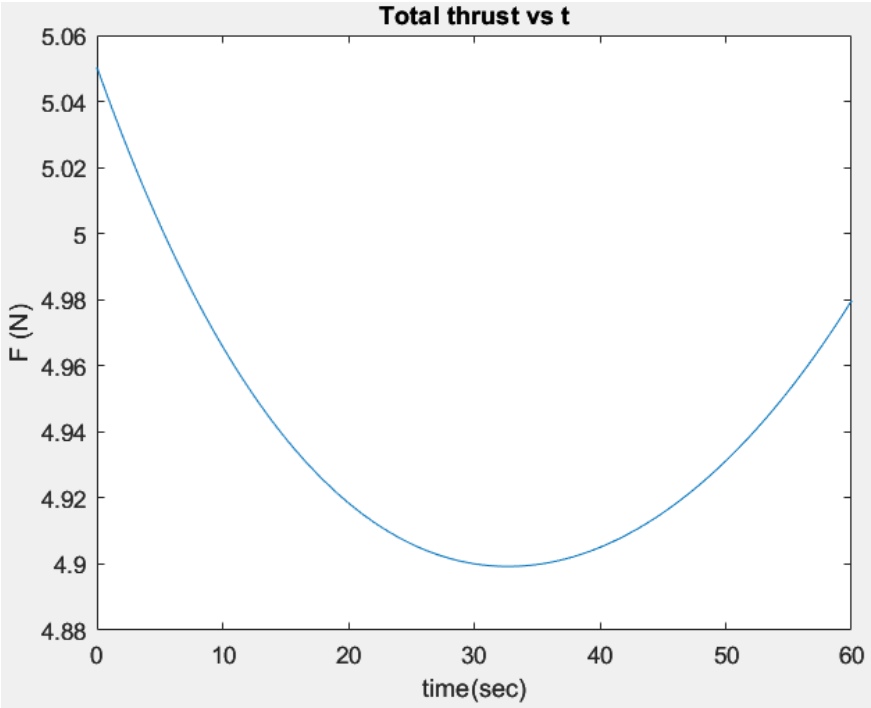
- Euler angles vs time plots:
  Following graphs shows Yaw vs time, Pitch vs time and Roll vs time plots.



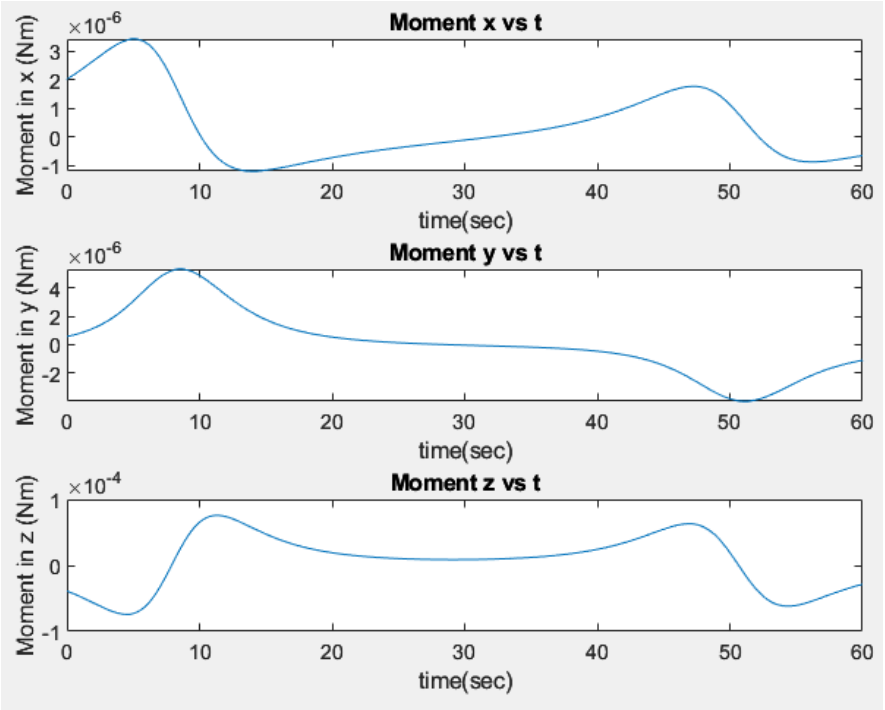
- Angular velocity ω_{tb}^{b} plot:
  Following graphs shows p vs time, q vs time, and r vs time plots.



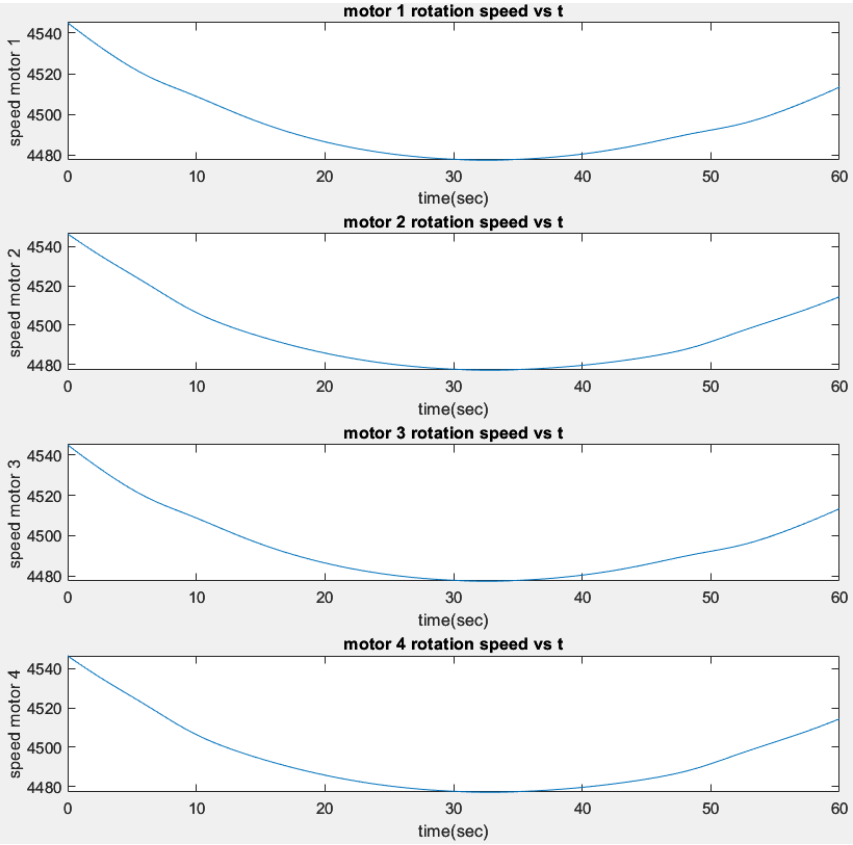
- Magnitude F plot:
  Following graph shows F vs time plot.



- Moment Plots:
  Following plots shows moment generated by the rotors.



- Motor spin rates:
  Following plots shows spinning rates of four motors.

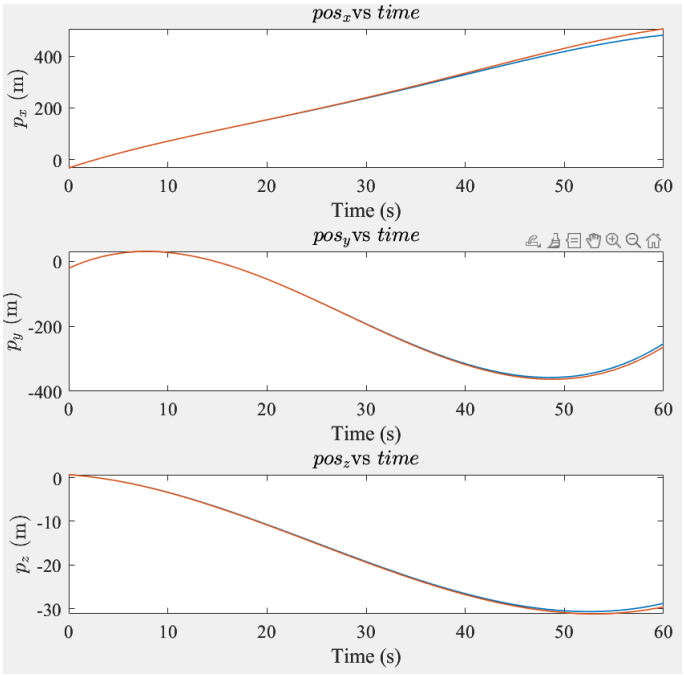


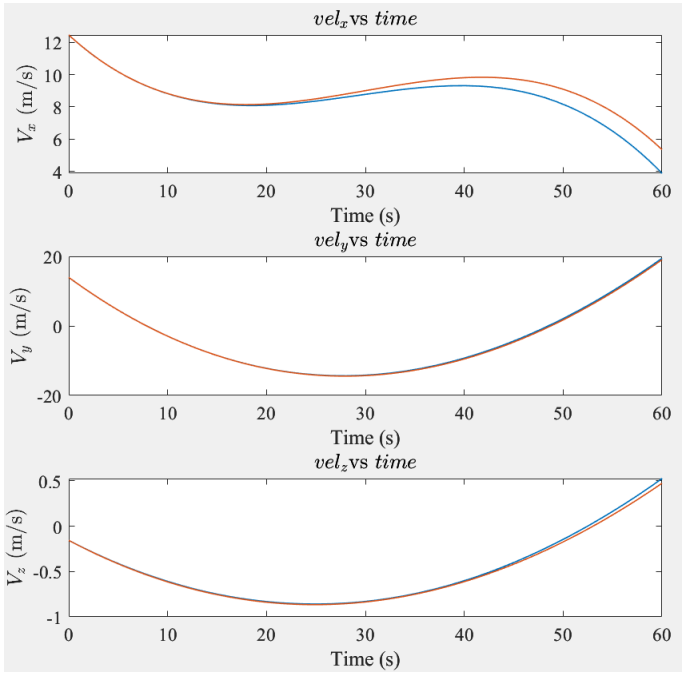
Matlab Script attached in the appendix at the end of the report.

Problem 3:

Running the output from the inverse dynamics derived from the previous problem into the simulation, following results were achieved.

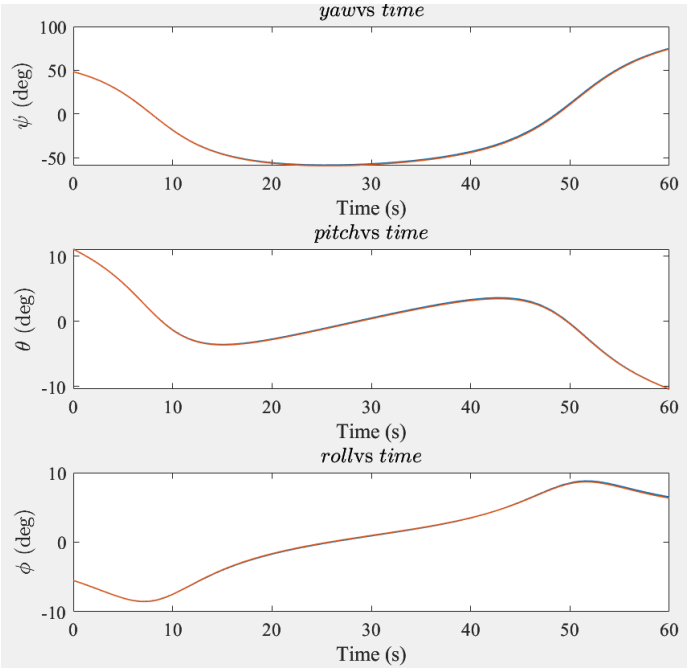
In the following plots, red curves represents the inverse dynamics (problem 2) output and blue curves represents the simulation output.

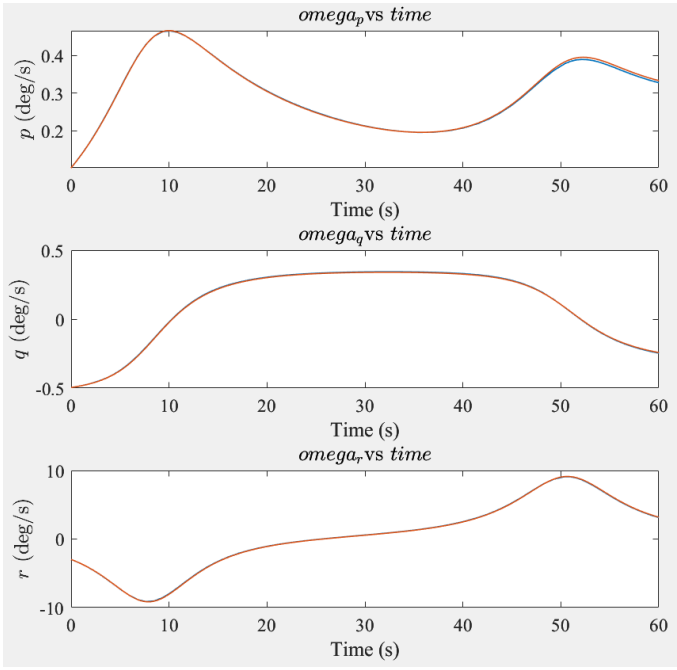




In the above plots, the figure in the left represents position vs time plots and the figure in the right represents velocity vs time plots.

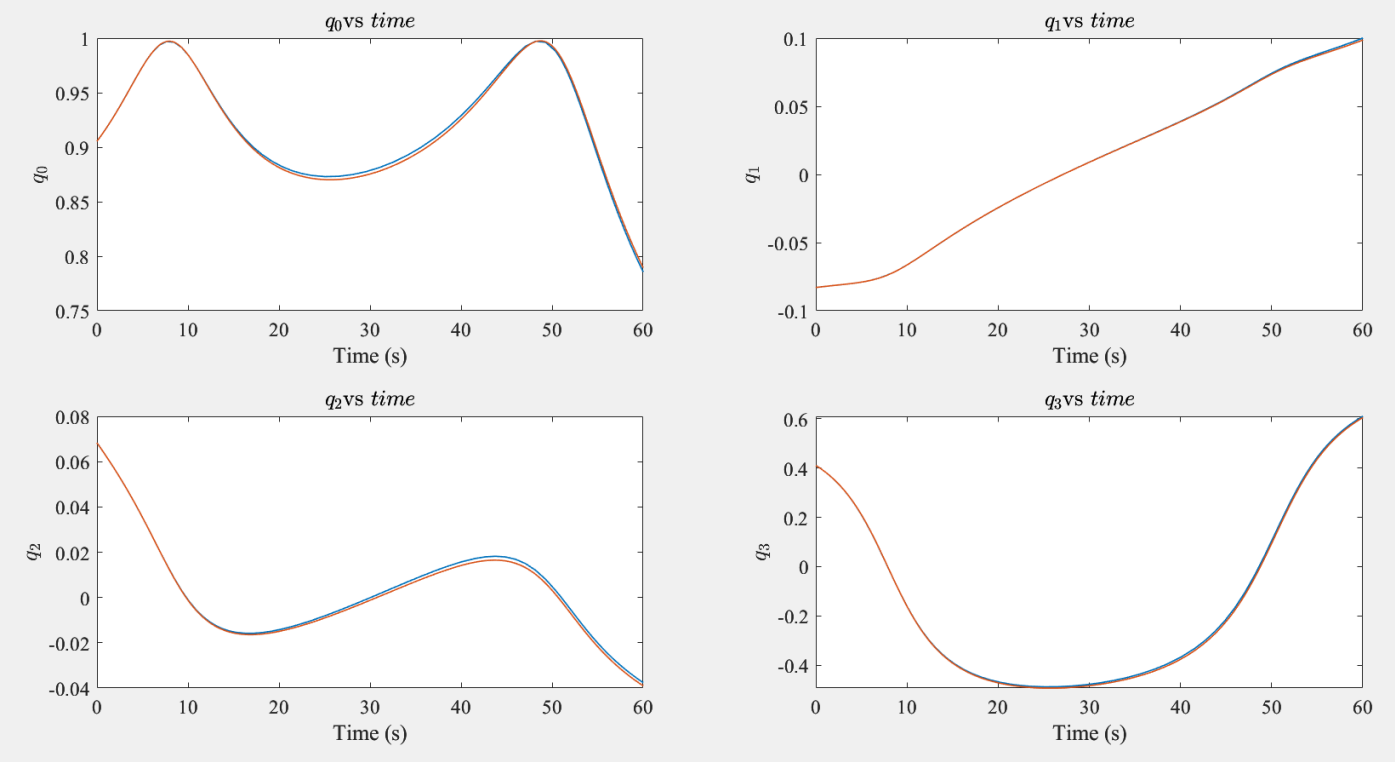
It can be observed that outputs the of simulation (blue curves) are similar to the given trajectory (red curves) and the simulated quadcopter seems to follow the given trajectory.





In the above plots, the figure in the left represents Euler angles vs time plots and the figure in the right represents angular velocity vs time plots.

It can be observed that outputs of the simulation (blue curves) are similar to the inverse dynamics outputs (red curves).



The above figure quaternions vs time plots.

It can be observed that outputs of the simulation (blue curves) are similar to the inverse dynamics outputs (red curves).

Problem 4:

- Given trajectory

$$p_t = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4$$

- Differentiating the above equation to get desired velocity, acceleration, jerk, and snap.

$$\dot{p}_t = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3$$
$$\ddot{p}_t = 2a_2 + 6a_3 t + 12a_4 t^2$$

- Step1: States

$$V = \sqrt{\dot{p}_x^2 + \dot{p}_y^2 + \dot{p}_z^2}$$

$$\chi = \tan^{-1}\left(\frac{\dot{p}_y}{\dot{p}_x}\right)$$

$$\gamma = \sin^{-1}\left(\frac{-\dot{p}_z}{V}\right)$$

- Step 2: Bank Angle

$$\dot{V} = \frac{\dot{p}_x\ddot{p}_x + \dot{p}_y\ddot{p}_y + \dot{p}_z\ddot{p}_z}{V}$$

$$\dot{\gamma} = \frac{\dot{V}\dot{p}_z - \ddot{p}_z V}{V^2\cos\gamma}$$

$$\dot{\chi} = \frac{\left(\dot{p}_x\ddot{p}_y - \dot{p}_y\ddot{p}_x\right)\cos^2\chi}{\dot{p}_x^2}$$

$$\Phi = \tan^{-1}\left(\frac{mV\dot{\chi}\cos\gamma}{mV\dot{\gamma} + mg\cos\gamma}\right)$$

- Step 3: Lift and Angle of attack

$$L = \frac{mV\dot{\chi}\cos\gamma}{\sin\Phi}$$

$$C_L = \frac{2L}{\rho V^2 S}$$

$$\alpha = \frac{C_L - C_{L_0}}{C_{L_\alpha}}$$

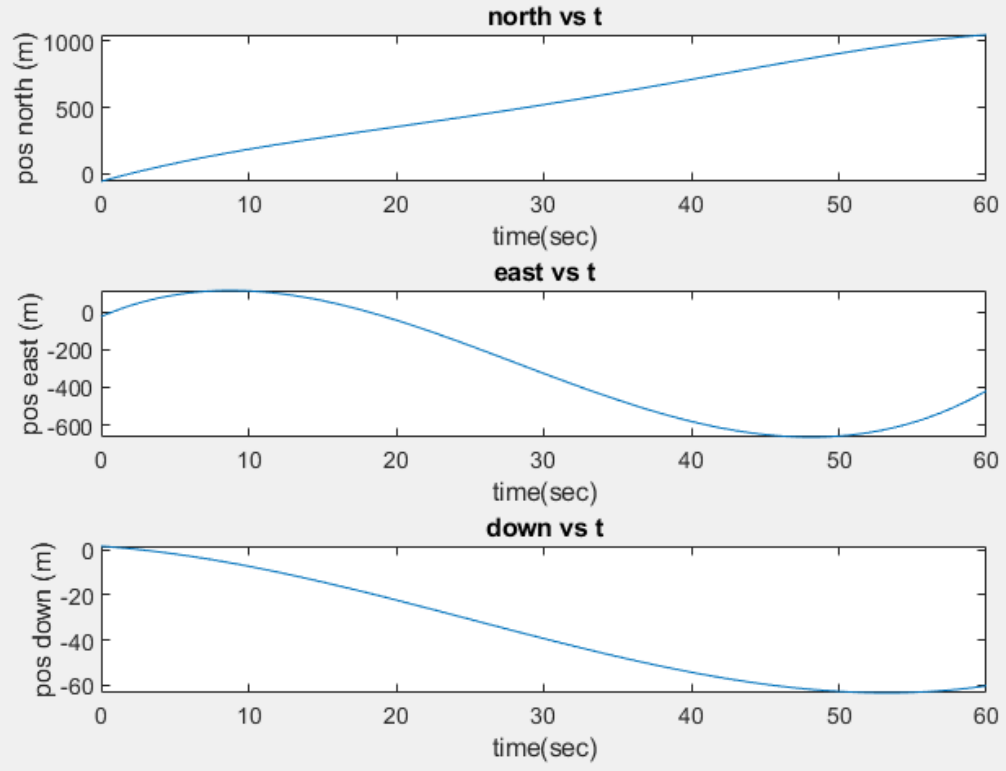
- Step 4: Drag and Thrust

$$D = \frac{1}{2}\rho V^2 S\left(C_{D_0} + \alpha C_{D_\alpha}\right)$$

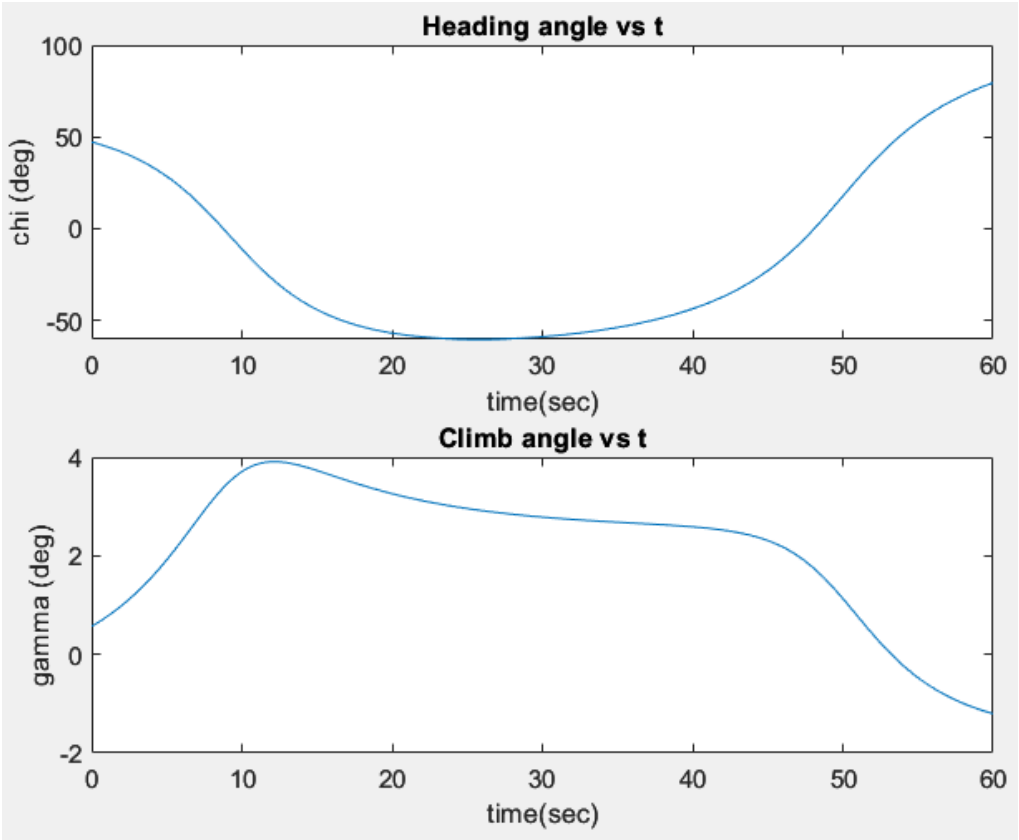
$$T = m\dot{V} + D + mg\sin\gamma$$

Result Plots:

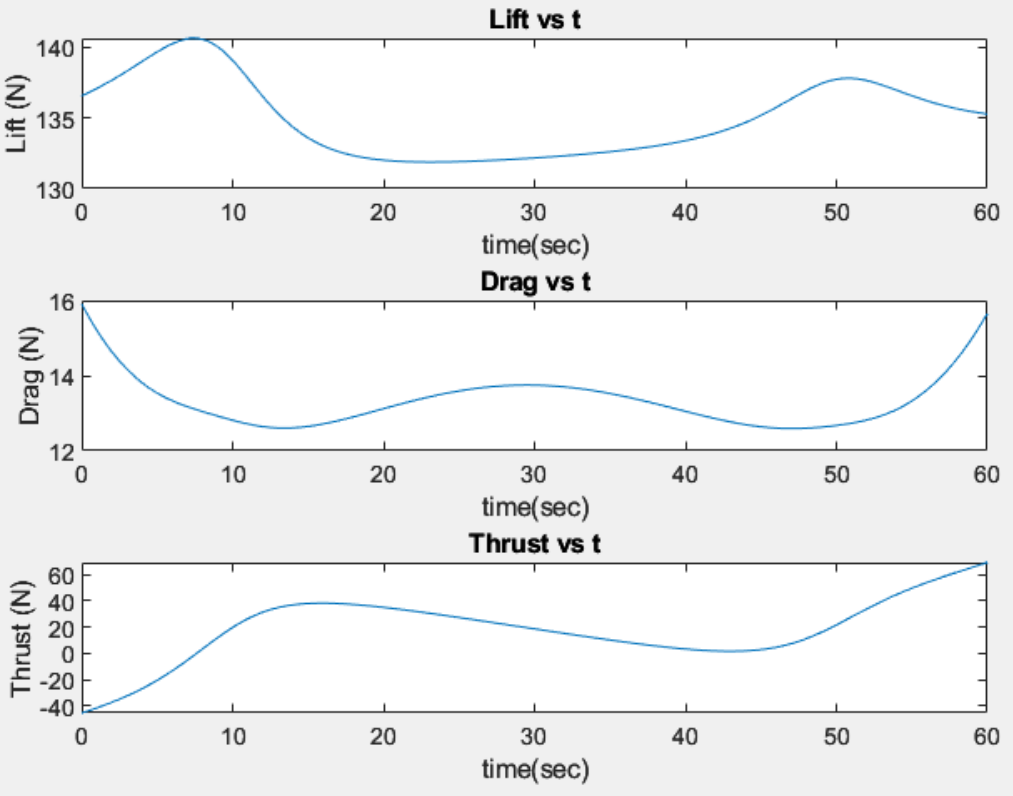
- Given trajectory:
  Following graphs shows the position vs time plots of the given trajectory for reference.



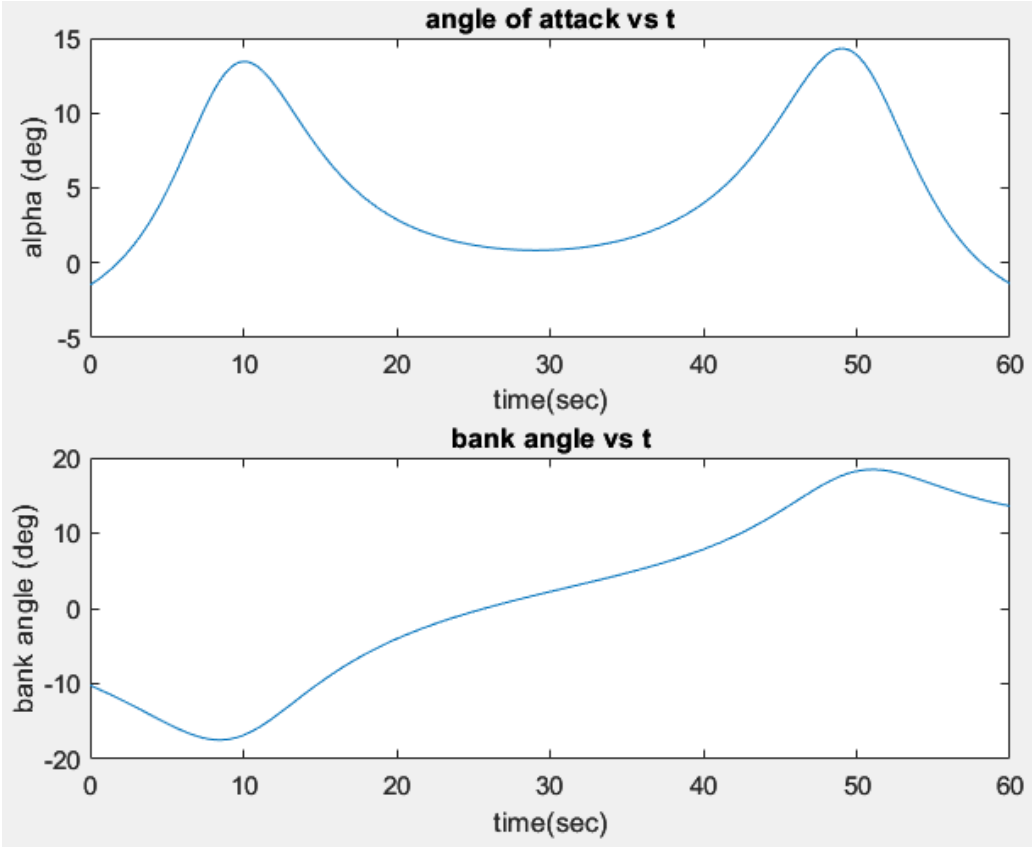
- Heading and climb angle plots:
  Following figure represents Heading angle vs time and climb angle vs time plots.



- Lift, Drag and thrust magnitude plots:
  Following figure shows Lift vs time, Drag vs time and Thrust vs time plots



- Angle of attack and bank angle plots:
  Following figure shows angle of attack vs time and bank angle vs time plots.



Matlab Script attached in appendix at the end of the report.

Problem 5:

The paper proposes a mechanically design variable pitch quadrotor, it discusses its advantages over fixed pitch quadrotor. It also discusses its closed loop control and trajectory generation algorithm, followed by the analytical and experimental results for its performance. The variable-pitch quadrotor used in this project was designed and built at MIT's ACL using mostly off-the-shelf components

The mechanical simplicity of fixed pitch multirotor have fundamental limitation on flight performances, as thrust can only be generated in one direction and the control bandwidth is limited to inertia of motor and propellers. These features restrict fixed pitch multirotor from performing aggressive and aerobatic maneuvers. These problems are amplified when the size of the quadrotor increases as they have larger inertias and then the quadrotor cannot be controlled by rpm alone. In the proposed variable pitch quadrotor reverse thrust is achievable and control bandwidth is limited only by the speed of the variable-pitch actuation, not by the inertia of the motor–propeller combination, which allows it to do aggressive and agile flight maneuver. Fast deceleration is achieved due to this which helps in flipping behaviour. With fixed-pitch propellers, the thrust produced by the propeller is constant. The only way to change the thrust produced by the propeller is by changing the voltage to the motor, thereby inducing a change in the rotational rate of the propeller. But in variable-pitch propellers to the quadrotor platform results in an additional degree-of-freedom for varying the thrust produced by each motor–propeller combination, thrust can be changed by either changing the blade pitch or by changing the rotational rate of the motors. With variable-pitch propellers, a quadrotor can hover by spinning the propeller quickly and with a low blade pitch, or by slowing the rotational rate of the motor and increasing the blade pitch. In fixed pitch multirotor there is a delay as it is limited to motor dynamics, whereas in varying pitch quadrotor it bypasses the motor dynamics as any change in the pitch of the blade is directly affecting the lift. A drawback is that increases the pitch of the blade reduces the speed of the motor which reduces the lift but that can be calculated and taken care of.

Analytically, one of the key benefits of the variable pitch propeller actuator is the ability to move anywhere on this thrust plot. With fixed-pitch propellers, the vehicle is restricted to moving only vertically on this plot with a set positive propeller pitch. The response of changing the pitch is much faster than the motor dynamics. The variable pitch setting has a higher kinetic energy which can be controlled to thrust quickly. But the drawback is that the power consumption should be taken care of for varying propeller loads. During the bench test, while the decreases in thrust and motor speed after the initial command given are greater than predicted by the simulations the overall behaviour of the system agrees with the simulated data.

Algorithms generating attitude-specific trajectories that account for actuator saturation levels are also presented in the paper. The desired attitude calculation and attitude control are computed using quaternions instead of rotation matrices. For trajectory generation This algorithm finds time-optimal polynomial paths subject to vehicle actuator saturation, allowing for paths tailored to the actuators of a specific vehicle by tuning the f max and f min to keep

the thrust in control. This decreases the potential saturation of the actuator. This solution gives a feasible and optimal solution without converging to the local minima.

The experimental results shows that it has the ability to track a trajectory in both normal and inverted orientations. While testing the negative thrust deceleration the reference acceleration needed was much faster than the response of the fixed pitch motor. It shows that the variable pitch controller is better both analytically and experimentally than fixed pitch multirotor and also has capabilities for aggressive and aerobatic maneuvers.

Problem 6:

This paper extends the existing work on polynomial trajectory generation by optimizing the path segments in an unconstrained quadratic programming that is stable for higher order polynomials, formulated for efficient computation. The paper also focuses on a technique to automatically select the amount of time required for each segment by tuning one parameter. The approach generates high-quality trajectories but sacrifices the guarantee of asymptotic convergence to the global optimum that other methods provide.

This technique is important as small unmanned aircraft have enabled high aerobatic flight maneuvers. However, motion planning algorithm fail to fulfil these capabilities at high speed. This paper addresses this problem and give a solution for unmanned aircraft to autonomously achieve these aggressive maneuvers at high speed, in indoor complex environment. There are other algorithms that provide solutions in finite time like RRT star. These algorithms have proven results for simple Dubin’s vehicles where analytical techniques can be used to move from one point to other. They fail to perform for nonlinear programming techniques, these methods are computationally expensive and may require accurate representation of environment which make this solution practically impossible. The paper shows that minimum snap trajectories can be solved numerically for long trajectories. This technique is coupled with appropriate kinematic planner to generate fast and feasible path in cluttered environment.

The solution uses RRT star algorithm to find a collision free path and considers the vehicle kinematics and ignores the dynamics initially. The path is then pruned to a minimal set of waypoints and a sequence of polynomial segments which are optimized to get a smooth path. It utilizes the differentially flat model of quadrotor and then associates control techniques. Since the desired trajectory and its derivatives are sufficient to compute the state and control inputs at every point along the path in closed form. This is the powerful capability enabled by differential flatness that eliminates the need for iterated numerical integration of equations of motion, or a search over the space of inputs during each iteration of the planning. Polynomial trajectories allow for an analytical solution via elimination as a constrained QP. While this method is acceptable for joint optimization of a few segments, it involves the inversion of matrices that may be very close to singularity. They select a random value for

time which is then later optimized based on desired average speed. The initial time selection is made large as it can be optimized later, and the motor saturation can also be taken care for. The solution is only valid for small trajectories to make it more robust they get rid of the constraints and directly solve the endpoint derivatives as decision variables. Once the optimal way point derivatives are found, the minimum-order polynomial connecting each pair of way points can be obtained by inverting the appropriate constraint matrix.

To Ensure collision free trajectory if a particular trajectory segment is found to intersect an obstacle after optimization, an additional way point is simply added halfway between its two ends, splitting this segment into two. This midpoint is known to be collision-free because it lies on the optimal piecewise-linear path returned by the search algorithm. The polynomial is re-optimized with the additional waypoint, and the process is repeated, if necessary, until the polynomial trajectory is collision free. In a dense environment it would need some optimizing as an iterative process will make the algorithm very complex. The second major factor contributing to feasibility is to ensure that the input constraints of the quadrotor are satisfied such that no portion of the commanded trajectory requires a thrust outside the range that the motors can provide, this is handled by the time optimizing parameter.

The Results were compared with RRT* and they use the Euclidean distances between the vertices divided by the desired average velocity. It is seen that RRT star runs longer and fails to find a smooth path and the algorithm proposed gives a better result in terms of optimality and time required to run. The computation feasibility enables it to run long range trajectories. They demonstrate the performance in an indoor lab space and use octomap representation and the algorithm performs better results.

Appendix

Problem 1:

```
clear variables; close all; clc;
syms x1 y1
figure()
hold on
dubConnObj = dubinsConnection("MinTurningRadius",0.15); %setting
the minimum turning radius
startPose = [0,0,0];
goalPose = [1,1,0];
[pathSegObj, pathCosts] = connect(dubConnObj,startPose,goalPose); %
get the shaortest dubins curve
show(pathSegObj{1})
% exportgraphics(gcf, 'obstacle_environment1.png')
pathSegObj{1}.MotionTypes
% disp(pathCosts);

% Using the calculation from Assingment 2

R = 0.15;

v = 0.03;
alpha = 1;%1;%5;

x = 0.1;
y = 0.5;
psi = 1.57;

x_arr = [x];
y_arr = [y];

while x<1 || y<1
    if (0.86*x + y -0.15 < 0)
        %C+ curve
        Pc1 = [0;0.15];
        psi0 = atan2(y-Pc1(2),x-Pc1(1));
        e = [x;y] - Pc1;
        k1 = 0.01;
        psi_ref = psi0 + (pi/2 + atan(k1*((norm(e)-R)/R)));

    elseif (0.69*x + y - 1.54 < 0)
```

```
            %Straight path
            eq1 = x1 - y1 == -0.069;
            eq2 = y1 - y == -1.1606*(x1-x);
            sol = solve([eq1,eq2],[x1,y1]);
            pref_x = double(sol.x1);
            pref_y = double(sol.y1);
            psi0 = atan2(0.772,0.896);
            r = [cos(psi0) sin(psi0);
                -sin(psi0) cos(psi0)];
            e = [x;y] - [pref_x;pref_y];
            ed = r*e;
            psi_inf = pi/4;%pi/4;
            k1 = 10;%10;%0.01;
            psi_ref = mod(psi0 -
psi_inf*(2/pi)*atan(k1*ed(2)),2*pi);
    else
        %C- curve
        Pc2 = [1;0.85];
        psi0 = atan2(y-Pc2(2),x-Pc2(1));
        e = [x;y] - Pc2;
        k1 = 5;%5;
        psi_ref = psi0 - (pi/2 +atan(k1*((norm(e)-R)/R)));
    end
    x_dot = v*cos(psi);
    y_dot = v*sin(psi);
    psi_dot = alpha*(psi_ref - psi);
    if alpha==5
        if psi_dot > 0.1
            psi_dot = 0.1;
        end
        if psi_dot < -0.1
            psi_dot = -0.1;
        end
    end
    x = x + x_dot;
    y = y + y_dot;
    psi = mod((psi + psi_dot),2*pi);
%     disp("x");
%     disp(x);
%     disp("y");
%     disp(y);
%     disp("Psi");
%     disp(psi);
    x_arr = [x_arr,x];
    y_arr = [y_arr,y];
end
```

```
plot(x_arr(:),y_arr(:),'-*');
display(psi);
```

Problem 2:

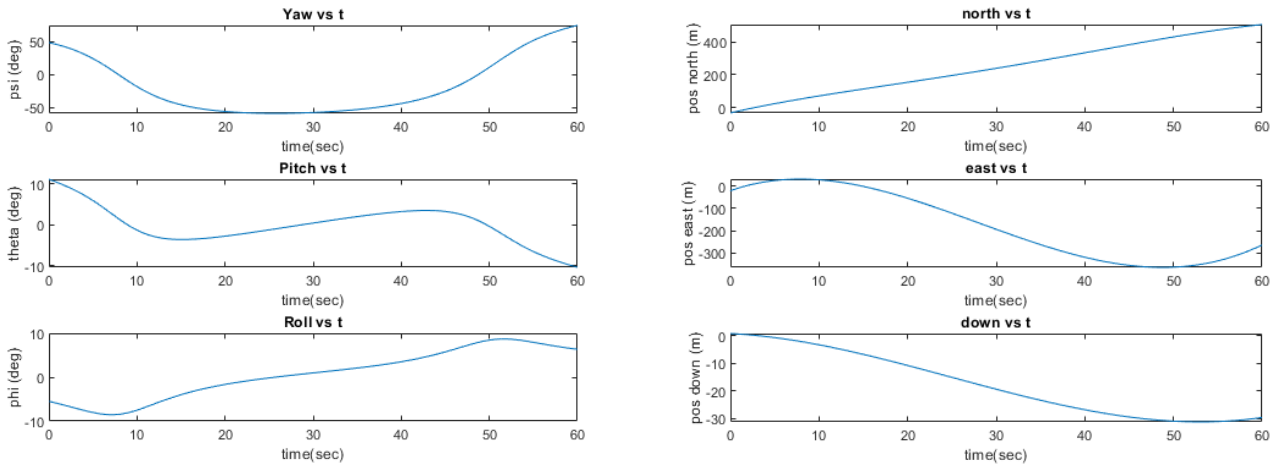
```
clear
clc

load asg3_prob2_coefficients.mat

m = 0.5;
L = 0.175;
J = diag([2.32 2.32 4]*1E-3);
Km = 1.5e-9;
Kf = 6.11e-8;


g = 9.81;

t_plot = [];
p_plot = [];
p_dot_plot = [];
attitude_plot = [];
w_plot = [];
F_plot = [];
M_b_plot = [];
motor_speed_plot = [];
n = 1;
q_plot = [];
for t = 0 : 0.1 : 60

    p = coeff_p(:,5) + coeff_p(:,4)*t  + coeff_p(:,3)*t^2 + coeff_p(:,2)*t^3 + coeff_p(:,1)*t^4;
    p_dot = coeff_p(:,4)  + 2*coeff_p(:,3)*t + 3*coeff_p(:,2)*t^2 + 4*coeff_p(:,1)*t^3;
    p_2dot = 2*coeff_p(:,3) + 6*coeff_p(:,2)*t + 12*coeff_p(:,1)*t^2;
    p_3dot = 6*coeff_p(:,2) + 24*coeff_p(:,1)*t;
    p_4dot = 24*coeff_p(:,1);


    psi = atan2(p_dot(2), p_dot(1));
    psi_dot = (p_dot(1)*p_2dot(2) - p_dot(2)*p_2dot(1))/(p_dot(1)^2 + p_dot(2)^2);
    psi_2dot = ((p_dot(1)^2 + p_dot(2)^2)*(p_dot(1)*p_3dot(2) - p_dot(2)*p_3dot(1))...
        - (p_dot(1)*p_2dot(2) - p_dot(2)*p_2dot(1))*(2*p_dot(1)*p_2dot(1) + 2*p_dot(2)*p_2dot(2)))...
        /(p_dot(1)^2 + p_dot(2)^2)^2;


    F_t = m*p_2dot - [0;0;m*g];

    F = norm(F_t);

    k_tb = -F_t/F;

    j_ta1 = [-sin(psi); cos(psi); 0];

    cross_jk = cross(j_ta1, k_tb);
    i_tb = cross_jk/norm(cross_jk);

    j_tb = cross(k_tb, i_tb);

    R_tb = [i_tb j_tb k_tb];

    temp_R = [cos(psi) sin(psi) 0; -sin(psi) cos(psi) 0; 0 0 1] * R_tb;
```

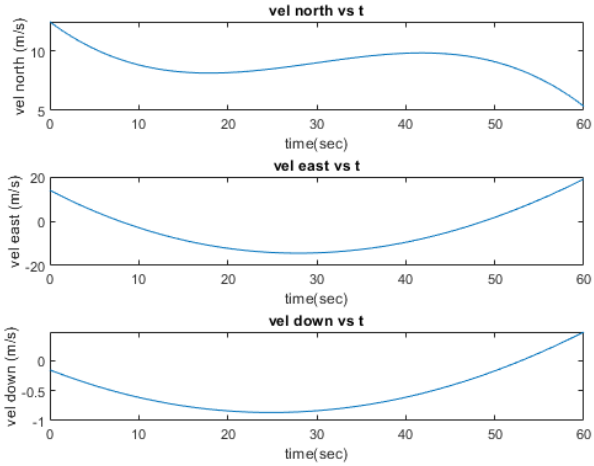
```
    th = atan2(-temp_R(3,1), temp_R(1,1));
    phi = atan2(-temp_R(2,3),temp_R(2,2));

    temp1 = (m*R_tb'*p_3dot);

    p_w = (1/F)*dot(temp1, [0;1;0]);
    q_w = (1/F)*dot(-temp1, [1;0;0]);

    temp2 = pinv([k_tb j_ta1 i_tb])*(R_tb*[p_w;q_w;0] - psi_dot*[0;0;1]);

    r_w = -temp2(1);
    th_dot = temp2(2);
    phi_dot = temp2(3);

    w_btb = [p_w;q_w;r_w];

    w_ttb = psi_dot*[0;0;1] + th_dot*j_ta1 + phi_dot*i_tb;

    F_dot = m^2*(p_2dot(1)*p_3dot(1) + p_2dot(2)*p_3dot(2) + (p_2dot(3)-g)*p_3dot(3)) / F;

    F_2dot = (m^2*(p_3dot(1)^2 + p_2dot(1)*p_4dot(1) + p_3dot(2)^2 +...
        p_2dot(2)*p_4dot(2) + p_3dot(3)^2 + (p_2dot(3)-g)*p_4dot(3)) - F_dot^2) / F;

    F_b = [0;0;-F];
    F_b_dot = [0;0;-F_dot];
    F_b_2dot = [0;0;-F_2dot];


    temp3 = (1/F)*(m*R_tb'*p_4dot - F_b_2dot - 2*cross(w_btb, F_b_dot) - cross(w_btb, cross(w_btb, F_b)));

    p_w_dot = temp3(2);
    q_w_dot = -temp3(1);

    i_tb_dot = [-psi_dot*sin(psi)*cos(th)-th_dot*cos(psi)*sin(th);
                -th_dot*sin(th)*sin(psi) + psi_dot*cos(th)*cos(psi);
                -th_dot*cos(th)];

    j_ta1_dot = psi_dot*[-cos(psi); -sin(psi); 0];

    temp4 = pinv([-k_tb j_ta1 i_tb])*(R_tb*[p_w_dot;q_w_dot;0] - psi_2dot*[0;0;1] - th_dot*j_ta1_dot -
phi_dot*i_tb_dot);

    r_w_dot = temp4(1);

    w_btb_dot = [p_w_dot; q_w_dot; r_w_dot];

    M_b = J*w_btb_dot + cross(w_btb, J*w_btb);

    const_mat = [-Kf -Kf -Kf -Kf;
                 0 -Kf*L 0 Kf*L;
                 Kf*L 0 -Kf*L 0;
                 Km -Km Km -Km];
    motor_speed = sqrt(pinv(const_mat)*[-F;M_b]);

    t_plot(:,n) = t;
    attitude_plot(:,n) = [psi; th; phi];
    w_plot(:,n) = w_btb;
    p_plot(:,n) = p;
    p_dot_plot(:,n) = p_dot;
    F_plot(:,n) = F;
    M_b_plot(:,n) = M_b;
    motor_speed_plot(:,n) = motor_speed;
```

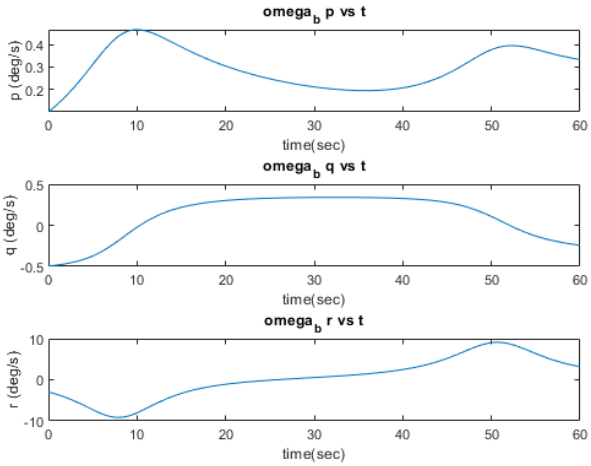
```
    qe = euler2quat([psi; th; phi]);
    q_plot(:,n) = qe;



    n = n + 1;

end

x_dynamics = [p_plot' p_dot_plot' attitude_plot' w_plot' q_plot'];
t_dynamics  = t_plot';
figure; %attitude angles plots
% plot 1
subplot(3,1,1)
plot(t_plot,attitude_plot(1,:)*180/pi)
title('Yaw vs t')
xlabel 'time(sec)';
ylabel 'psi (deg)';

% plot 2
subplot(3,1,2)
plot(t_plot,attitude_plot(2,:)*180/pi)
title('Pitch vs t')
xlabel 'time(sec)';
ylabel 'theta (deg)';

% plot 3
subplot(3,1,3)
plot(t_plot,attitude_plot(3,:)*180/pi)
title('Roll vs t')
xlabel 'time(sec)';
ylabel 'phi (deg)';


% position plots
figure;
% plot 1
subplot(3,1,1)
plot(t_plot,p_plot(1,:))
title('north vs t')
xlabel 'time(sec)';
ylabel 'pos north (m)';

% plot 2
subplot(3,1,2)
plot(t_plot,p_plot(2,:))
title('east vs t')
xlabel 'time(sec)';
ylabel 'pos east (m)';

% plot 3
subplot(3,1,3)
plot(t_plot,p_plot(3,:))
title('down vs t')
xlabel 'time(sec)';
ylabel 'pos down (m)';

%velocity plot
figure;
% plot 1
subplot(3,1,1)
plot(t_plot,p_dot_plot(1,:))
title('vel north vs t')
xlabel 'time(sec)';
```

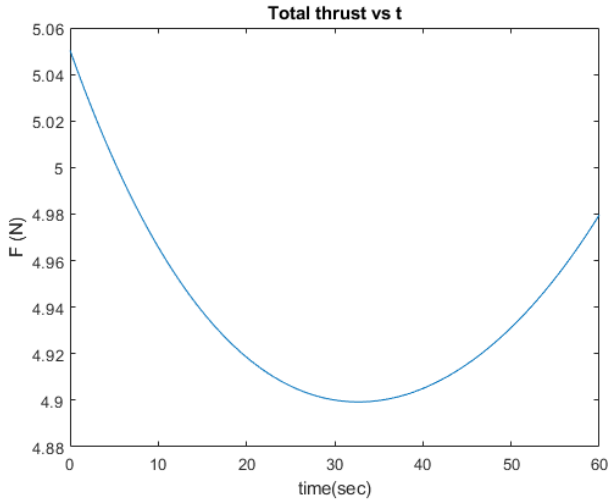
```
ylabel 'vel north (m/s)';

% plot 2
subplot(3,1,2)
plot(t_plot,p_dot_plot(2,:))
title('vel east vs t')
xlabel 'time(sec)';
ylabel 'vel east (m/s)';

% plot 3
subplot(3,1,3)
plot(t_plot,p_dot_plot(3,:))
title('vel down vs t')
xlabel 'time(sec)';
ylabel 'vel down (m/s)';


% %3D trajectory plot for visualition
% figure;
% plot3(p_plot(2,:),p_plot(1,:), -p_plot(3,:))
% title('phi vs t')
% xlabel 'east';
% ylabel 'north';
% zlabel 'neg_down';

% Omega plots
figure;
% plot 1
subplot(3,1,1)
plot(t_plot,w_plot(1,:)*180/pi)
title('omega_b p vs t')
xlabel 'time(sec)';
ylabel 'p (deg/s)';

% plot 2
subplot(3,1,2)
plot(t_plot,w_plot(2,:)*180/pi)
title('omega_b q vs t')
xlabel 'time(sec)';
ylabel 'q (deg/s)';

% plot 3
subplot(3,1,3)
plot(t_plot,w_plot(3,:)*180/pi)
title('omega_b r vs t')
xlabel 'time(sec)';
ylabel 'r (deg/s)';


% force plot
figure;
plot(t_plot,F_plot(1,:))
title('Total thrust vs t')
xlabel 'time(sec)';
ylabel 'F (N)';

figure
% plot 1
subplot(3,1,1)
plot(t_plot,M_b_plot(1,:))
title('Moment x vs t')
xlabel 'time(sec)';
ylabel 'Moment in x (Nm)';
```

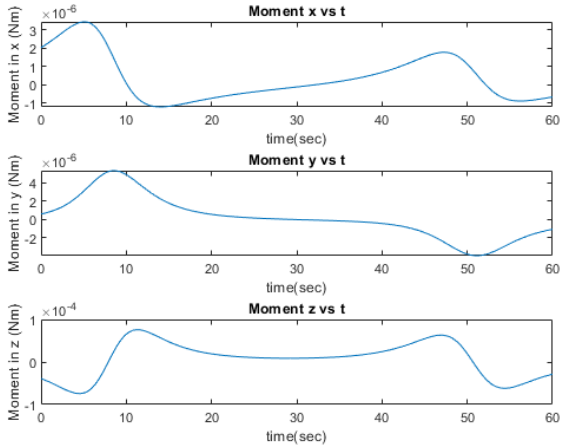
```
% plot 2
subplot(3,1,2)
plot(t_plot,M_b_plot(2,:))
title('Moment y vs t')
xlabel 'time(sec)';
ylabel 'Moment in y (Nm)';

% plot 3
subplot(3,1,3)
plot(t_plot,M_b_plot(3,:))
title('Moment z vs t')
xlabel 'time(sec)';
ylabel 'Moment in z (Nm)';



% Motor speed plots
figure;
% plot 1
subplot(4,1,1)
plot(t_plot,motor_speed_plot(1,:))
title('motor 1 rotation speed vs t')
xlabel 'time(sec)';
ylabel 'speed motor 1';

% plot 2
subplot(4,1,2)
plot(t_plot,motor_speed_plot(2,:))
title('motor 2 rotation speed vs t')
xlabel 'time(sec)';
ylabel 'speed motor 2';

% plot 3
subplot(4,1,3)
plot(t_plot,motor_speed_plot(3,:))
title('motor 3 rotation speed vs t')
xlabel 'time(sec)';
ylabel 'speed motor 3';

% plot 4
subplot(4,1,4)
plot(t_plot,motor_speed_plot(4,:))
title('motor 4 rotation speed vs t')
xlabel 'time(sec)';
ylabel 'speed motor 4';
```

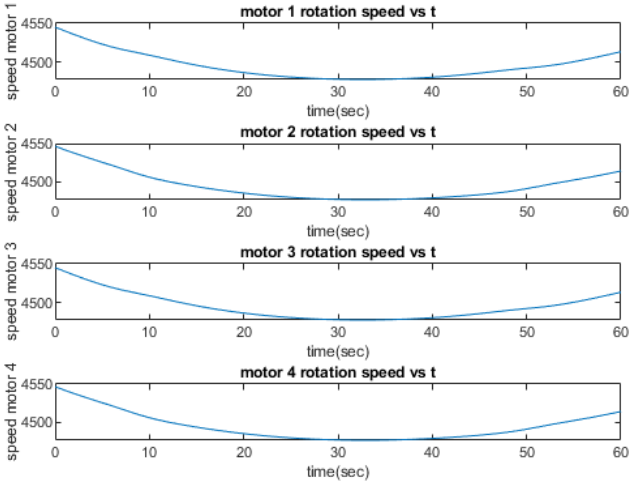












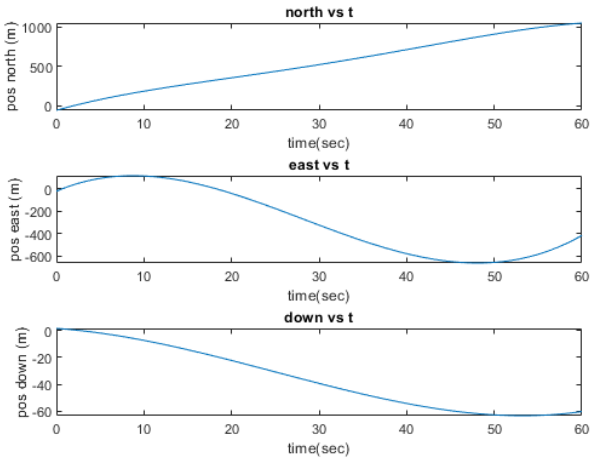
Published with MATLAB® R2022b

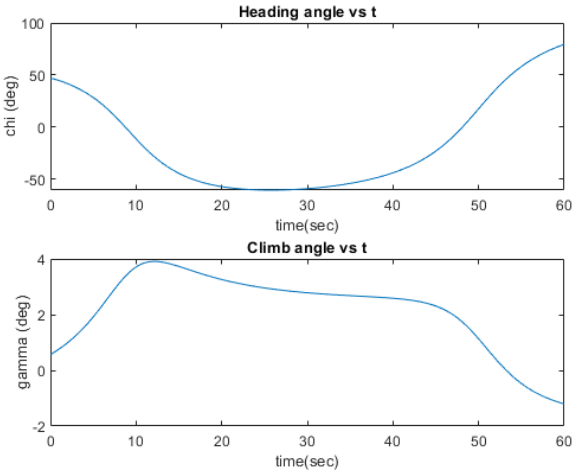
Problem 4:

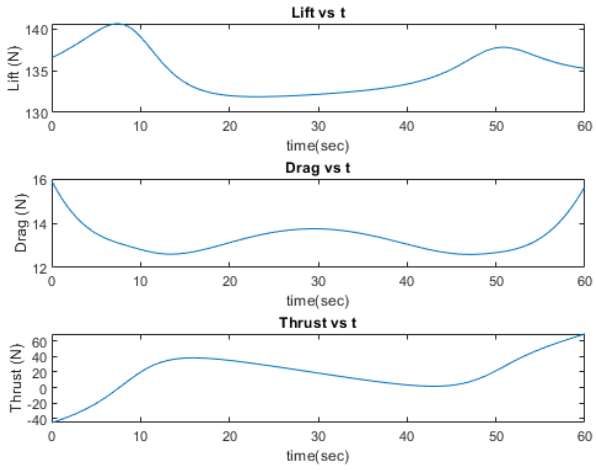
```
clear
clc

load asg3_prob4_coefficients.mat

m = 13.5; %kg
g = 9.81; %m/s^2
rho = 1.225; % kg/m^3
S = 0.55; %m^2
CL_0 = 0.28;
CD_0 = 0.03;
CL_alp = 3.45;
CD_alp = 0.30;

t_plot = [];
p_plot = [];
V_pot = [];
chi_plot = [];
gamma_plot = [];
L_plot = [];
D_plot = [];
T_plot = [];
alpha_plot = [];
bank_angle_plot = [];
n = 1;
for t = 0 : 0.1 : 60


    p = coeff_p(:,5) + coeff_p(:,4)*t  + coeff_p(:,3)*t^2 + coeff_p(:,2)*t^3 + coeff_p(:,1)*t^4;
    p_dot = coeff_p(:,4)  + 2*coeff_p(:,3)*t + 3*coeff_p(:,2)*t^2 + 4*coeff_p(:,1)*t^3;
    p_2dot = 2*coeff_p(:,3) + 6*coeff_p(:,2)*t + 12*coeff_p(:,1)*t^2;

    V = norm(p_dot);

    chi = atan2(p_dot(2), p_dot(1));

    gamma = asin(-p_dot(3)/V);

    V_dot = (p_dot(1)*p_2dot(1) + p_dot(2)*p_2dot(2) + p_dot(3)*p_2dot(3))/V;

    gamma_dot = (V_dot*p_dot(3) - p_2dot(3)*V)/(V^2*cos(gamma));

    chi_dot = (cos(chi))^2*(p_dot(1)*p_2dot(2) - p_2dot(1)*p_dot(2))/p_dot(1)^2;

    bank_angle = atan2(m*V*chi_dot*cos(gamma), m*V*gamma_dot + m*g*cos(gamma));

    L = m*V*chi_dot*cos(gamma)/sin(bank_angle);

    CL = 2*L/(rho*V^2*S);

    alpha = (CL - CL_0)/CL_alp;

    D = 0.5*rho*V^2*S*(CD_0 + alpha*CD_alp);

    T = m*V_dot + D + m*g*sin(gamma);

    t_plot(:,n) = t;
    p_plot(:,n) = p;
    V_pot(:,n) = V;
    chi_plot(:,n) = chi;
    gamma_plot(:,n) = gamma;
```

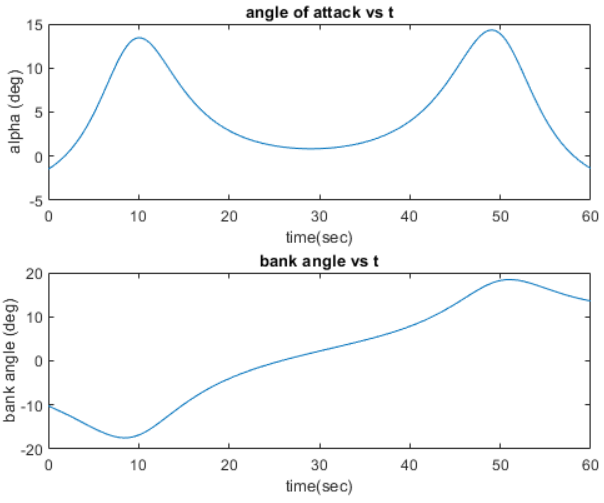
```
    L_plot(:,n) = L;
    D_plot(:,n) = D;
    T_plot(:,n) = T;
    alpha_plot(:,n) = alpha;
    bank_angle_plot(:,n) = bank_angle;

    n = n + 1;
end

% position plots
figure;
% plot 1
subplot(3,1,1)
plot(t_plot,p_plot(1,:))
title('north vs t')
xlabel 'time(sec)';
ylabel 'pos north (m)';

% plot 2
subplot(3,1,2)
plot(t_plot,p_plot(2,:))
title('east vs t')
xlabel 'time(sec)';
ylabel 'pos east (m)';

% plot 3
subplot(3,1,3)
plot(t_plot,p_plot(3,:))
title('down vs t')
xlabel 'time(sec)';
ylabel 'pos down (m)';



figure
subplot(2,1,1)
plot(t_plot, chi_plot*180/pi)
title('Heading angle vs t')
xlabel 'time(sec)';
ylabel 'chi (deg)';

subplot(2,1,2)
plot(t_plot, gamma_plot*180/pi)
title('Climb angle vs t')
xlabel 'time(sec)';
ylabel 'gamma (deg)';

figure
subplot(3,1,1)
plot(t_plot, L_plot)
title('Lift vs t')
xlabel 'time(sec)';
ylabel 'Lift (N)';

subplot(3,1,2)
plot(t_plot, D_plot)
title('Drag vs t')
xlabel 'time(sec)';
ylabel 'Drag (N)';

subplot(3,1,3)
plot(t_plot, T_plot)
title('Thrust vs t')
xlabel 'time(sec)';
```

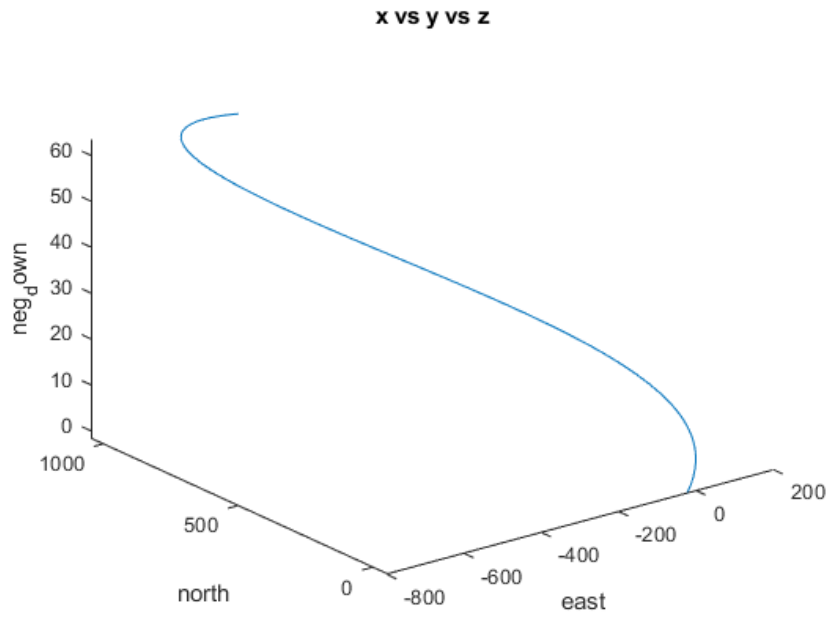
```
ylabel 'Thrust (N)';


figure
subplot(2,1,1)
plot(t_plot, alpha_plot*180/pi)
title('angle of attack vs t')
xlabel 'time(sec)';
ylabel 'alpha (deg)';

subplot(2,1,2)
plot(t_plot, bank_angle_plot*180/pi)
title('bank angle vs t')
xlabel 'time(sec)';
ylabel 'bank angle (deg)';

figure;
plot3(p_plot(2,:),p_plot(1,:), -p_plot(3,:))
title('x vs y vs z')
xlabel 'east';
ylabel 'north';
zlabel 'neg_down';
```











Published with MATLAB® R2022b