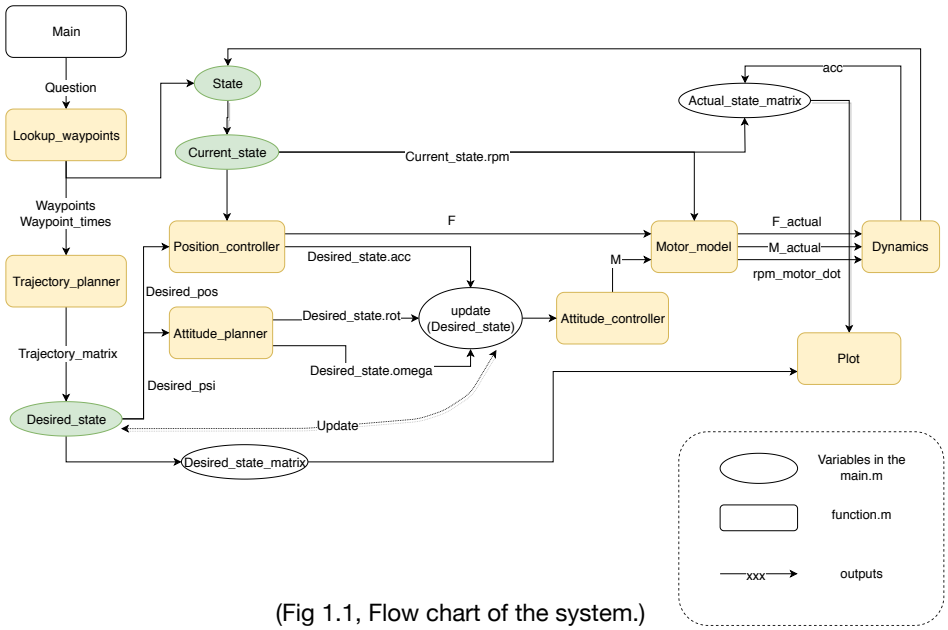
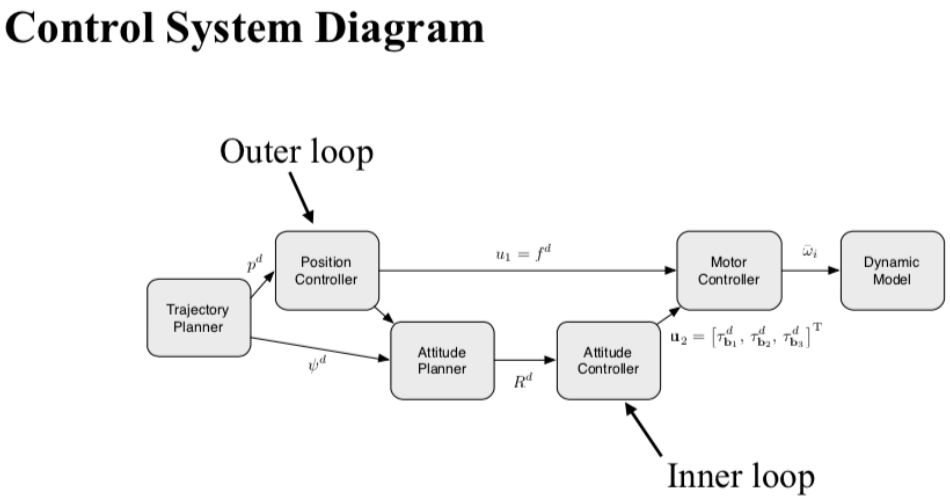
Mobility HW2 — Air Mobility
chichiaw@andrew.cmu.edu
collaboration: Mo, szuyum@andrew.cmu.edu

Q1)



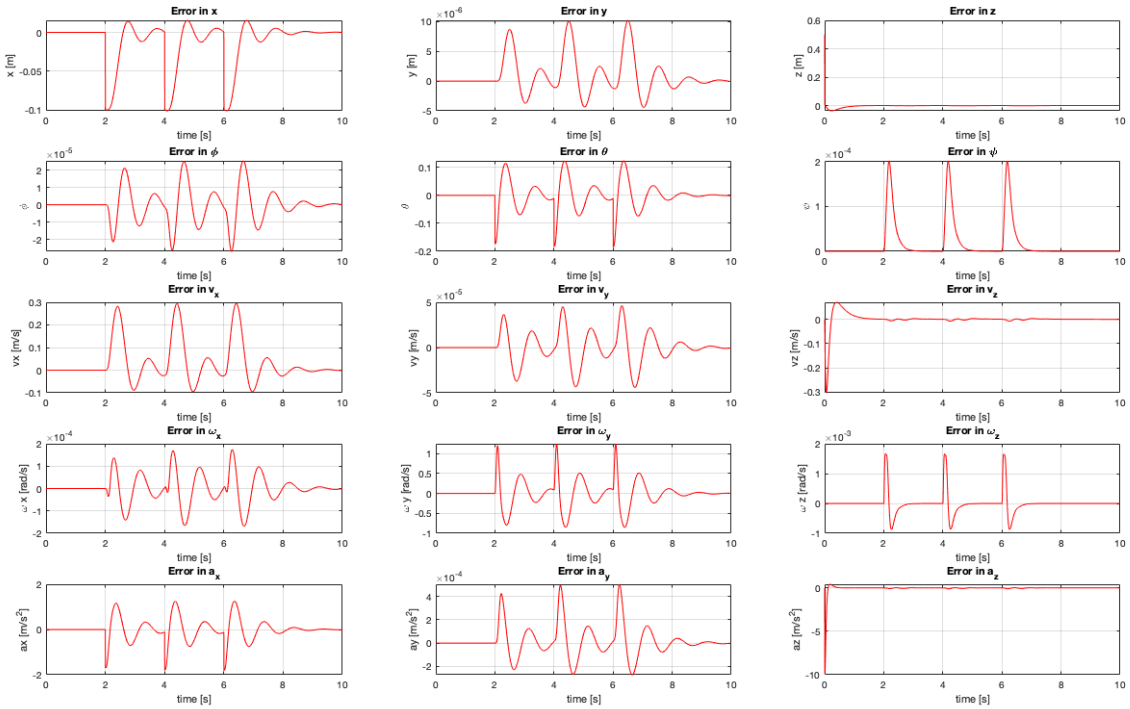
(Fig 1.1, Flow chart of the system.)



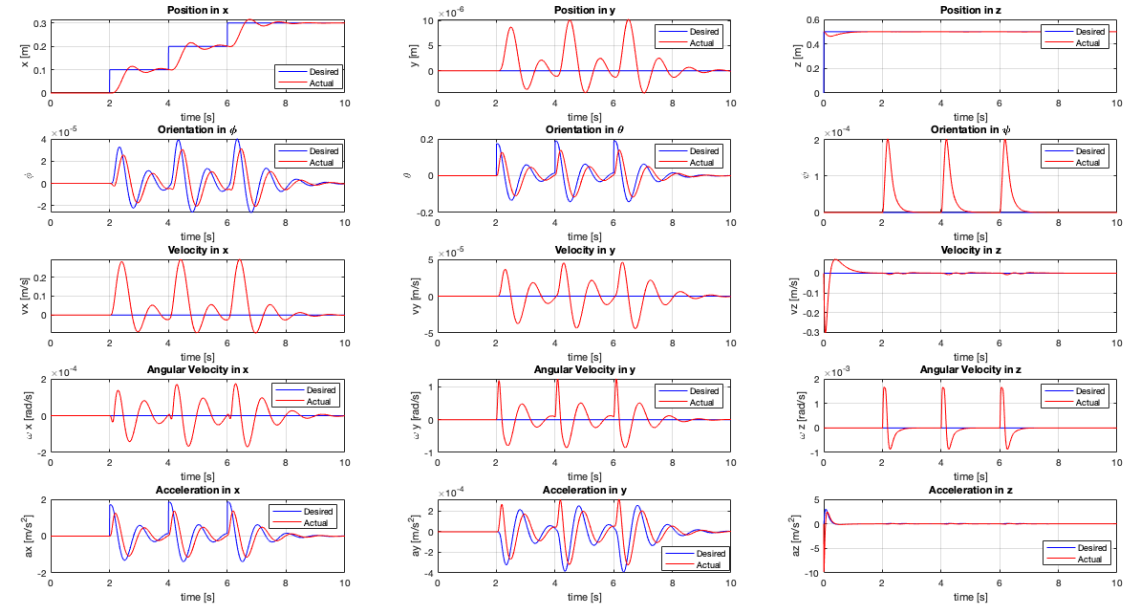
(Fig 1.2, Control System Diagram from the lecture.)

The code structure is based on the control system diagram as in figure 1.2. However, a lookup_waypoints function is added to generate the desired input trajectory for each question. One of the things that is not clearly stated in figure 1.1 is the variable "question" is also an input to "Trajectory_planner", "Position_controller", "Attitude_controller". Depending on different question numbers, the trajectory planner stores in different numbers of desired inputs, the position controller and attitude controller have different gain values to achieve the optimal time responses of the system. Three state vectors are used to store the values of current state, desired state and state. Lookup_waypoints function passes out desired state at each time step and updates the "Desired_state". The results calculated by dynamics function and ode45 at each time step will be stored in "State". "State" passes useful values to store in "Current state". "Position_controller" controls the positions of the robot with respect to the world frame, and "Attitude_controller" deals with the orientation deviations. Then desired force and moments are fed into the "Motor_model" to obtain the time deviation of the speeds of the motors. Last but not lease, all are the inputs of the dynamics model. By implementing the ode45 function in MATLAB, one can calculated the result (state) at each time step and feed into the plot function to generate plots.

Q2)



(Fig 2.1, Plot of error.)



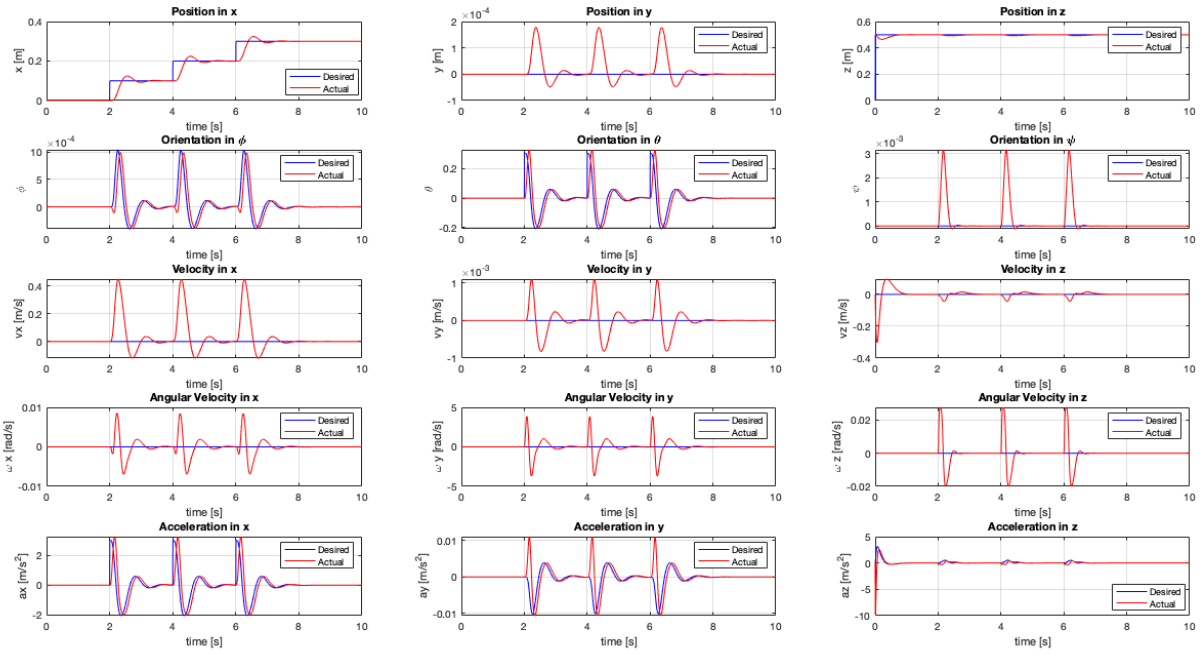
(Fig 2.2, Plot of position.)

At time equals 2,4,6 second, new waypoints are sent to the system. Each waypoint input acts as a step input to the system, and the system converges approximately 1.5 sec after the step input. In figure 2.2, one could observe a slight oscillation in both x and y positions. Significant oscillations can be observed in "position in x", "orientation in theta", "velocity in x", "angular velocity y", "acceleration in x". The oscillations are expected since at each point, the robot accelerates (moves) then stops at the waypoints. Also, the robot has to lean forward to accelerate and lean backward to stop. Therefore, there is also oscillation in "orientation in theta".

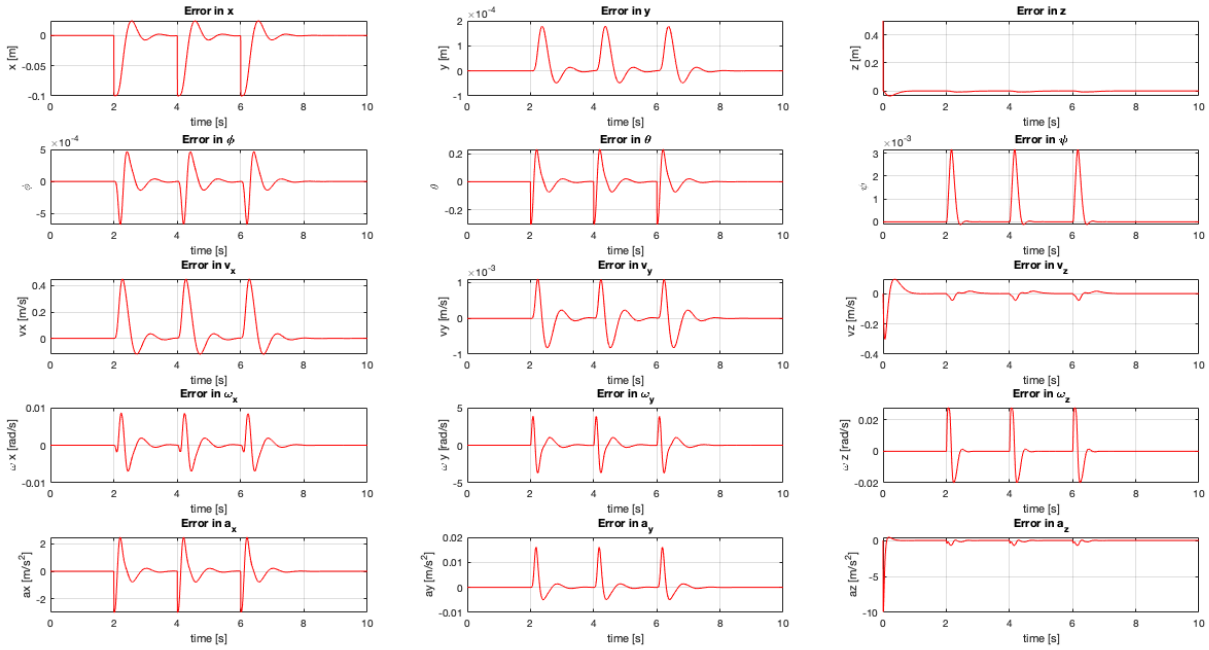
(Table 1, Effect on system response for change gains.)

Parameter	Change	Rise Time	Overshoot
Attitude Controller Kr	increase	same	decrease
Attitude Controller Kr	decrease	same	increase
Attitude Controller Kw	increase	same	increase
Attitude Controller Kw	decrease	same	decrease
Position Controller Kp	increase	decrease	increase
Position Controller Kp	decrease	increase	decrease
Position Controller Kd	increase	increase	decrease
Position Controller Kd	decrease	decrease	increase

When the gains of the attitude controller and position controller are changed, the behaviors of the response would also change accordingly, as shown in table 1. If the gains of position controller are changed, both the rise time and percent overshoot of the system will change. However, only the percent overshoot is effected when the gains in attitude controller are changed.



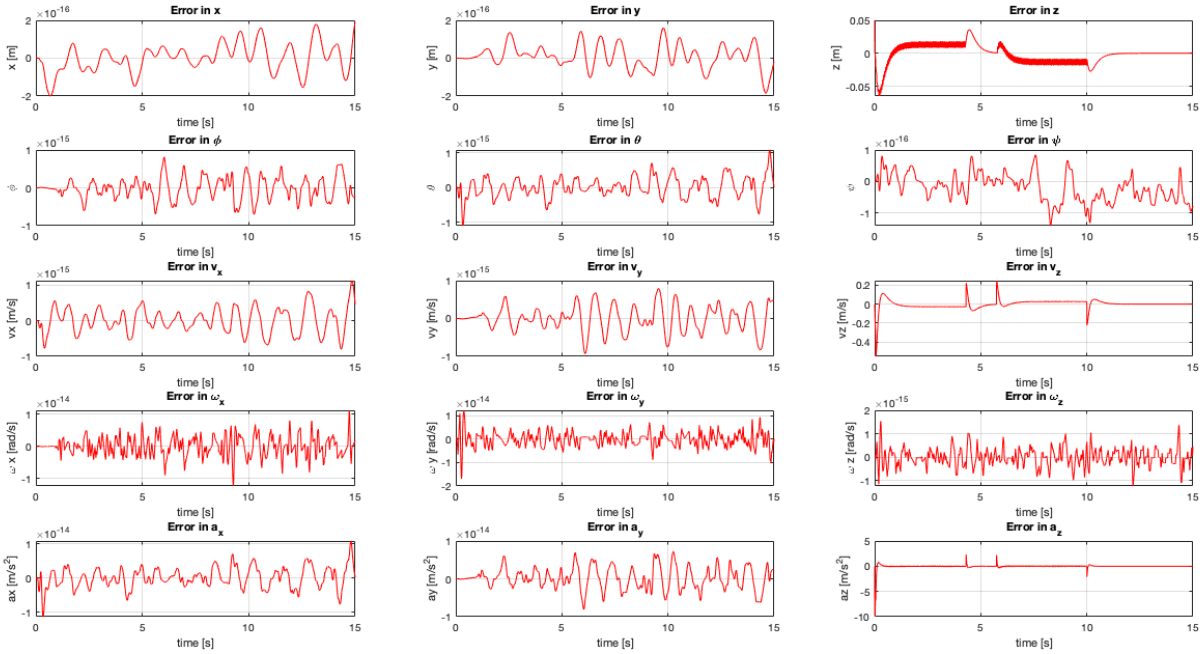
(Fig 2.4, Plot of position, Kp=[30,30,30] Kd=[6.6,6.6,9] Kr=[380,380,160] Kw=[30,30,17.88].)



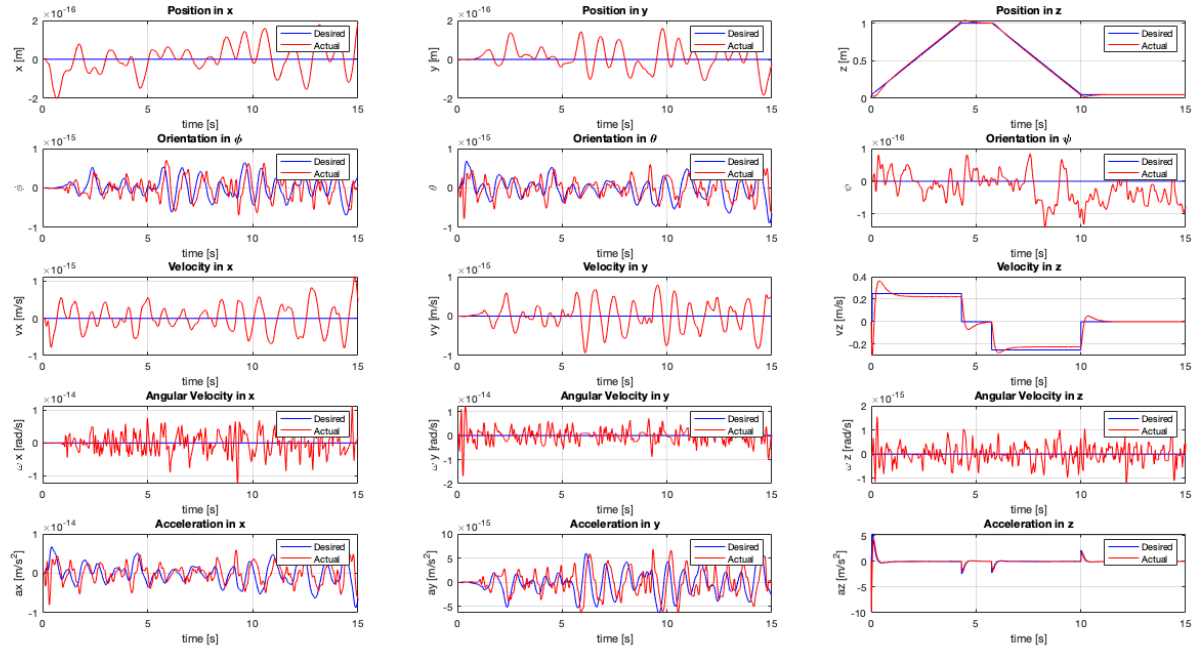
(Fig 2.3, Plot of error, Kp=[30,30,30] Kd=[6.6,6.6,9] Kr=[380,380,160] Kw=[30,30,17.88].)

In figure 2.3 and 2.4, Kp and Kr are increased, while Kd and Kw stay the same (In figure 2.1, 2.2, Kp=[17, 17, 20], Kr=[190, 198, 80]). As predicted in table 1. The rise time is decreased due to increasing in Kp. Kr decreases the percent overshoot. Overall, the rise time decreases while percent overshoot stays the same.

Q3)



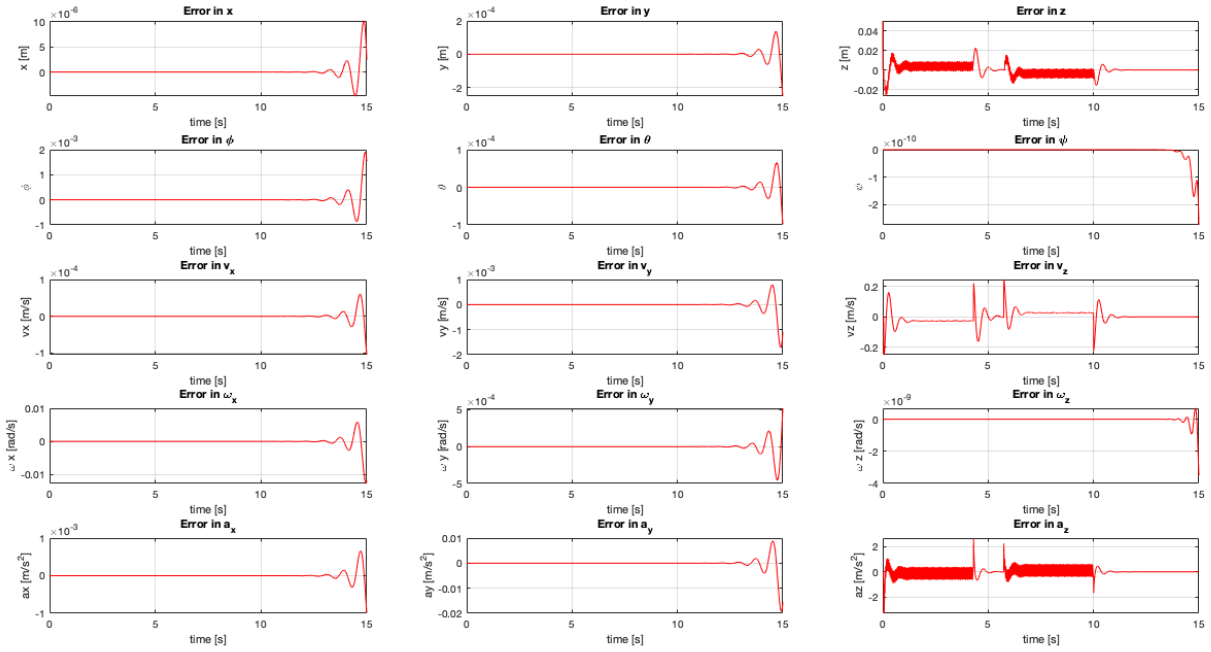
(Fig 3.1, Plot of error.)



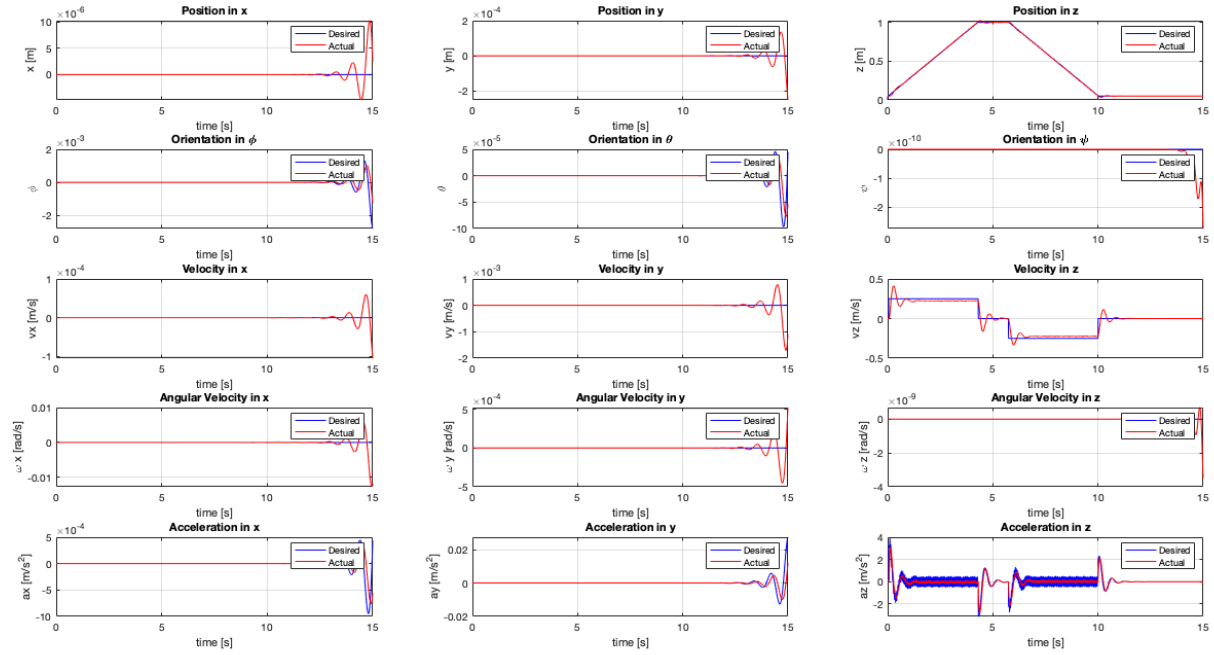
(Fig 3.2, Plot of position.)

Kp=[17,17,20], Kd=[6.6, 6.6 9], Kr=[190, 198, 80], Kw=[30, 30, 17.88]

The oscillations of the system can be seen in most of the plots in figure 3.1 and 3.2. However, they are in 1e-15 to 1e-16 scale, which can be ignored. In this problem, the system is moving in one direction only, so all the desired inputs are given in b3 direction. All the other responses in b1 and b2 directions are minimal and could be seen as 0.



(Fig 3.3, Plot of error.)



(Fig 3.4, Plot of position.)

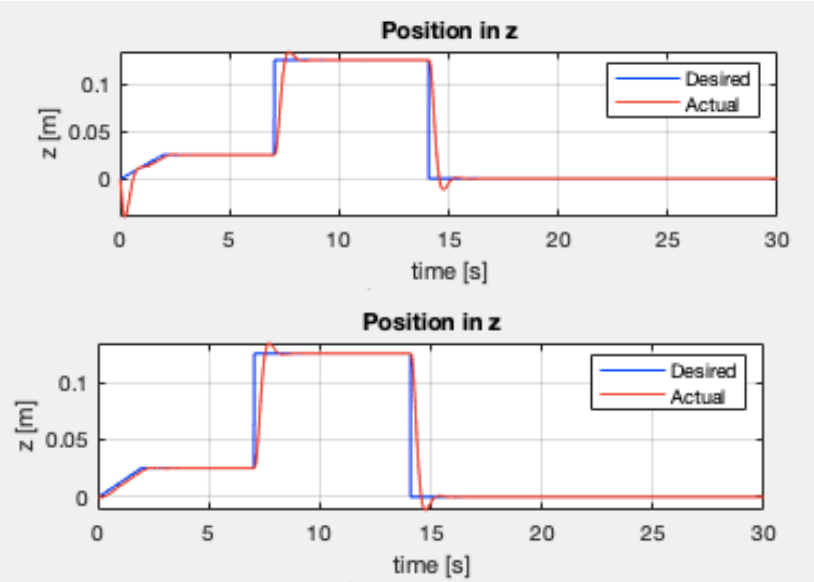
Kp=[80, 80, 95], Kd=[6.6, 6.6 9], Kr=[190, 198, 80], Kw=[30, 30, 17.88]

In figure 3.3 and figure 3.4, Kp in the position controller(outer loop) is five times the Kp in figure 3.1 and 3.2. As one can see, the responses of "position in z" and "velocity in z" have shorter rise times and larger percent overshoots. Also, there are oscillations in "velocity in z" in figure 3.3 and 3.4. As mentioned in previous question, tuning up Kp will result in shorter rise time, higher percent overshoot, and might cause oscillations in the time responses. One method to get rid of the oscillation is to tune Kd in the position controller to smooth out the maximum percent overshoot.

Q4)

Takeoff:

At this state, one has to think of the ground reaction force that holds the robot before the motor could generate enough lift (greater than mg) to leave the ground. In practice, throughout the takeoff state, the minimum external force acting on the robot should never be less than mg (i.e. the ground will support the robot to stay on the ground). Otherwise, the robot will drop bellow ground level and experience a sharp negative acceleration at the beginning of the takeoff, as shown in figure 4.1. An easy if and else condition can solve the problem, as shown in figure 4.2. The input and output of the motor model are modified slightly. A variable flag is initialized at 0 in the main.m and once after the robot leaves the ground, flag is updated to 1. The motor model function reads in the current value of variable flag and output the updated value, as shown in figure 4.2. After the robot takes off, there should be no restriction on the minimum force. When the force is less than mg, the robot descends. The robot will ascend when the force(lift) is greater than mg.



(Fig 4.1up, Plot of position without considering ground reaction force.)
(Fig 4.1down, Plot of position considering the support from ground reaction force.)

```
F_motor=result(1);


    if F_motor>params.mass*params.gravity
        flag=1;
    end

    if flag==0 && F_motor< params.mass*params.gravity
         F_motor=params.mass*params.gravity;
    end

    if F_motor> params.mass*2.6*params.gravity
          F_motor=params.mass*2.6*params.gravity;
    end
```

(Fig 4.2, If else condition to take into account ground reaction force.)

Hover:
At hover state, the robot will store the last values of the state from takeoff state and maintain at those values for a given time(here is 5 seconds), as in figure 4.3. A big advantage of starting to track a trajectory after the hover state instead of takeoff or idle state is that the robot is in a relatively stable condition in the air and the speed of the motors are sufficient to move and maneuver the robot right away.

```
%hover for 5s
waypoint_times_hover=linspace(timespan_takeoff+timespan_track+2,timespan_takeoff+timespan_track+7,10);
waypoints_x_hover=linspace(waypoints_x_stop(end),waypoints_x_stop(end),10);
waypoints_y_hover=linspace(waypoints_y_stop(end),waypoints_y_stop(end),10);
waypoints_z_hover2=linspace(waypoints_z_stop(end),waypoints_z_stop(end),10);
waypoints_psi_hover=linspace(0,0,10);
waypoints_xdot_hover=linspace(0,0,10);
waypoints_ydot_hover=linspace(0,0,10);
```

(Fig 4.3, Example of hover state code in question 8.)

Track:
At tracking state, the robot tracks whatever the desired inputs are. The desired inputs might consist of positions, velocities, accelerations and angles. All the remaining questions have the same takeoff, hover and land state. The only difference between them is the tracking state.

Hover:
The hover state before landing is similar to that after takeoff state. The robot stores the last values of the state from the track state and maintains at the same values for a give time. The benefit of having a hover state before landing is that at the end of the tracking state the velocities and accelerations of the robot might not be zero. Therefore, it is better to start landing after the robot becomes stable (hover at a point) by going through hover state.

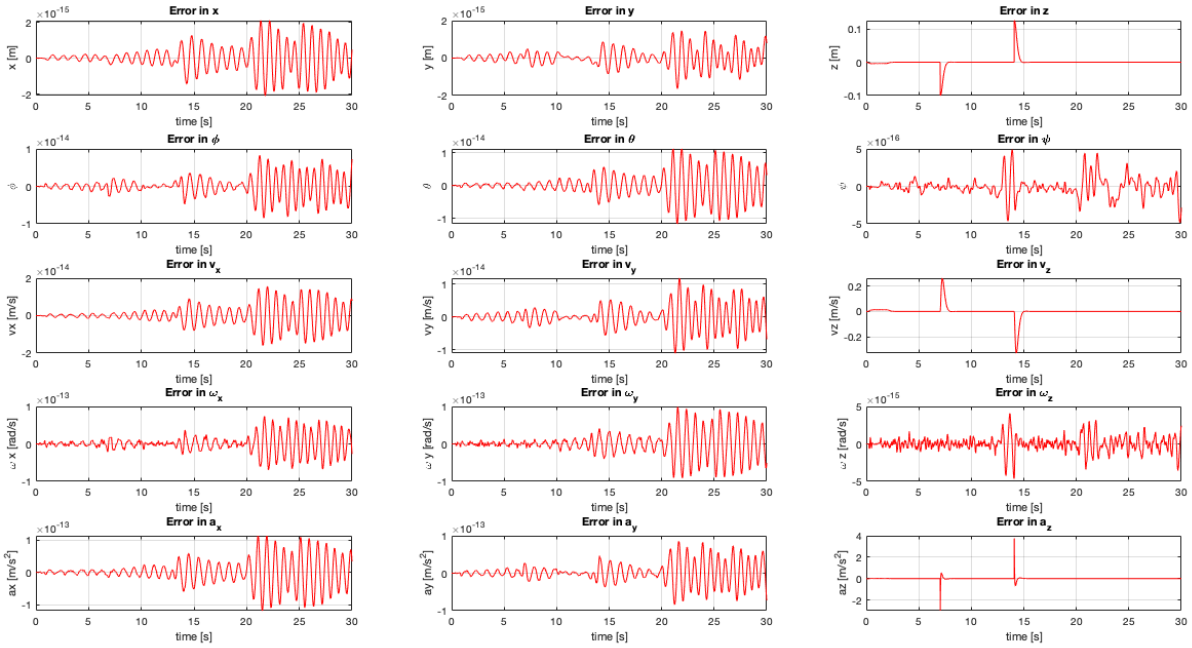
Land:
In landing state, the robot stays at the same x y positions and descends to the ground. A first order trajectory is used to land the robot. The timespan of the landing state varies as the hover position (z-direction) varies, in order to bound the error in z direction throughout the landing state.

```
if question ==5

    timespan_track=16;
    way_x=linspace(0, 0,160);           %x position
    way_y=linspace(0, 0, 160);          %y position

    way_z1=linspace(0,0.025, 20);       %takeoff state
    way_z2=linspace(0.025,0.025, 50);   %hover state
    way_z3=linspace(0.125,0.125,20);    %tracking state
    way_z4=linspace(0.125,0.125,50);    %hover state
    way_z5=linspace(0,0,20);            %landing state

    ang=15*pi()/180;

    way_psi1=linspace(0,0,70);
    way_psi2=linspace(ang,ang,90);      %psi

    waypoints=[ way_x;
                way_y;
                way_z1 way_z2 way_z3 way_z4 way_z5;
                way_psi1 way_psi2;
                way_xdot;
                way_zdot;
                ];
    waypoint_times=linspace(0,timespan_track,160);
end
```

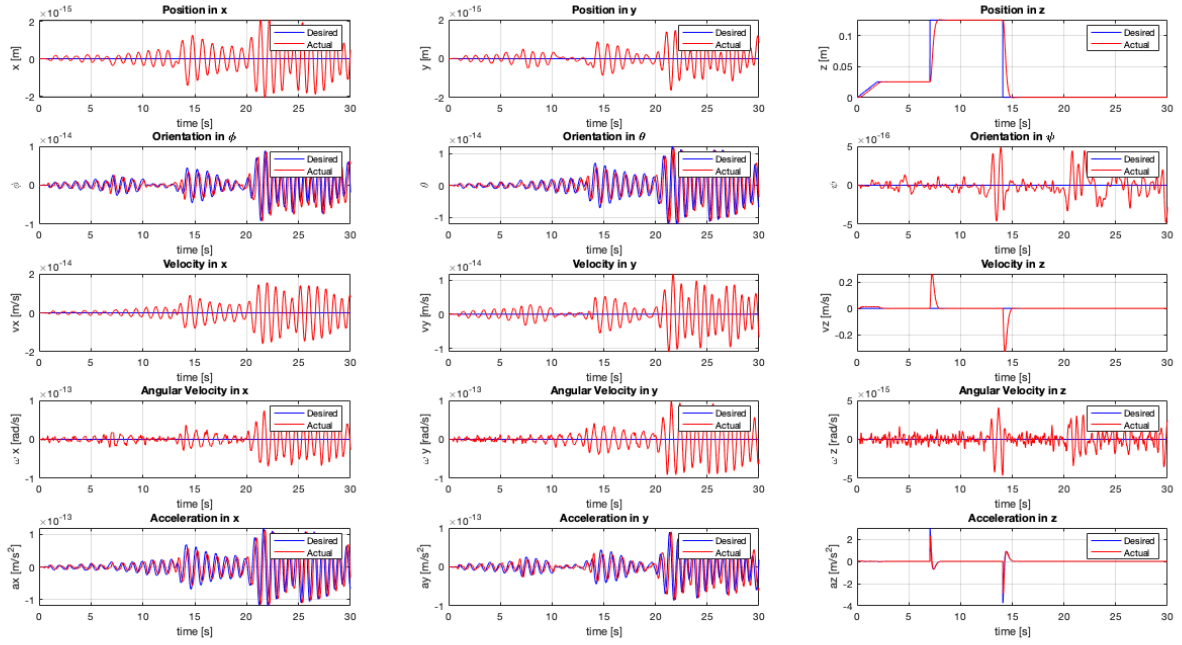
(Fig 4.4, Example of the state machine lookup waypoint for question 5.)

Q5)
part 1 (heading =0)

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]



(Fig 5.1, Plot of error.)

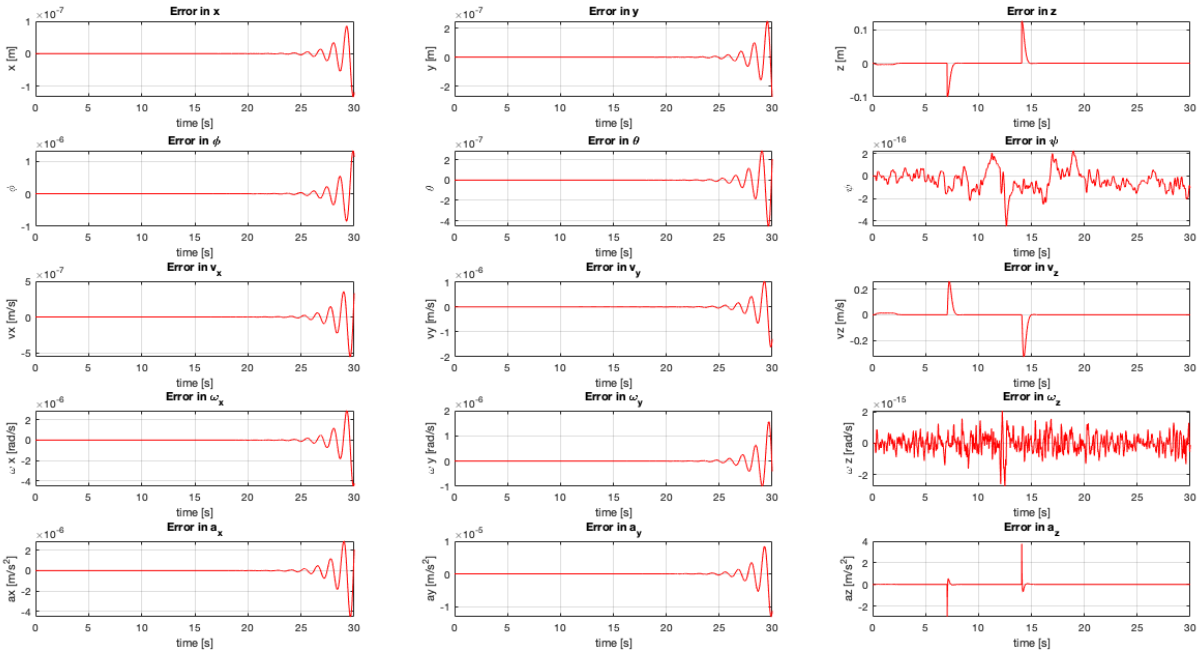


(Fig 5.2, Plot of position.)

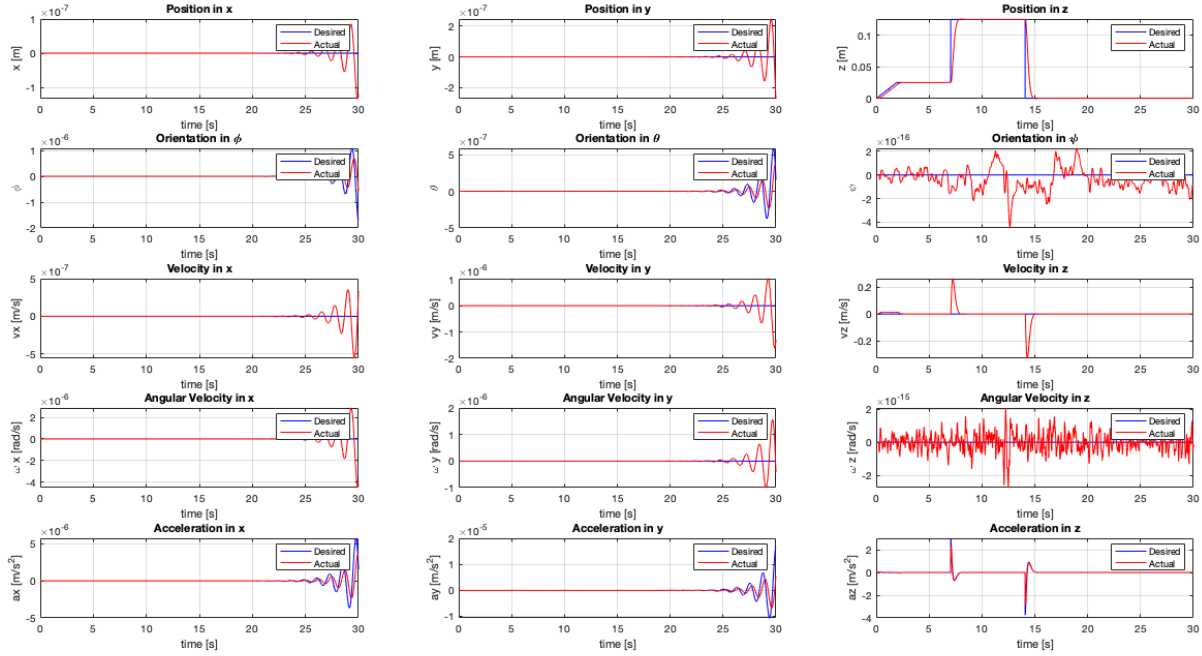
(Table 2, Time domain performance characteristics.)

Rise Time (time after hover)	0.5350s
Settling Time (time after hover)	0.5650s
Steady-state Value	0.125
Max percent overshoot	0.44

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[100, 100, 40]	Kw=[30, 30, 17.88]



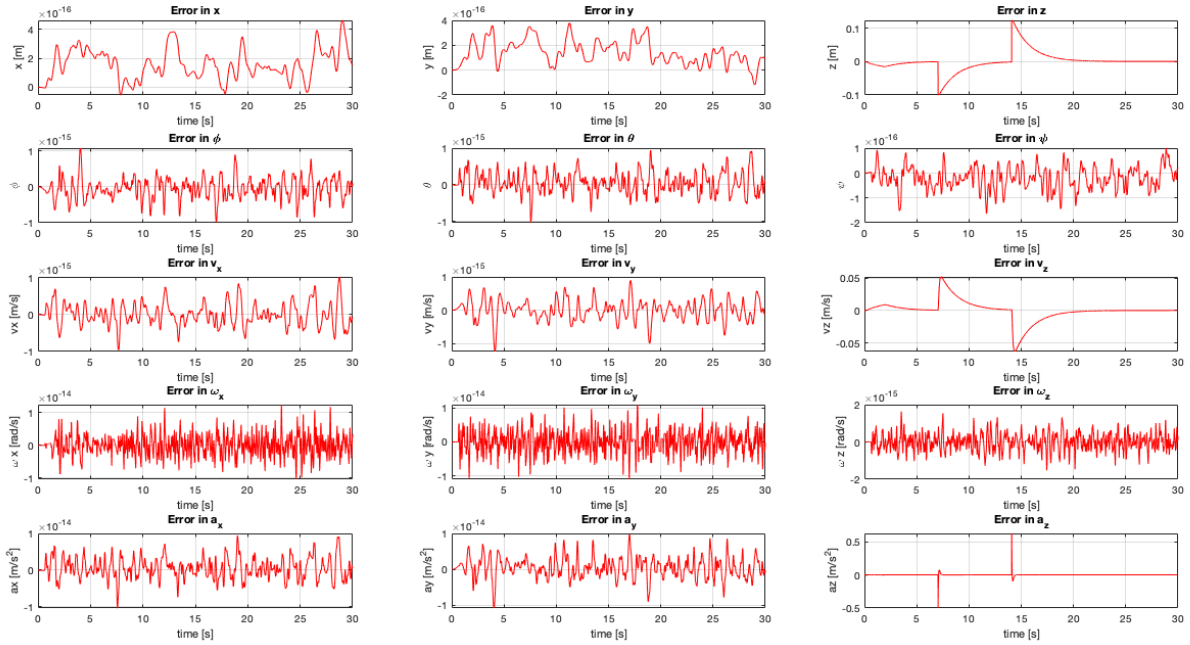
(Fig 5.3, Plot of error.)



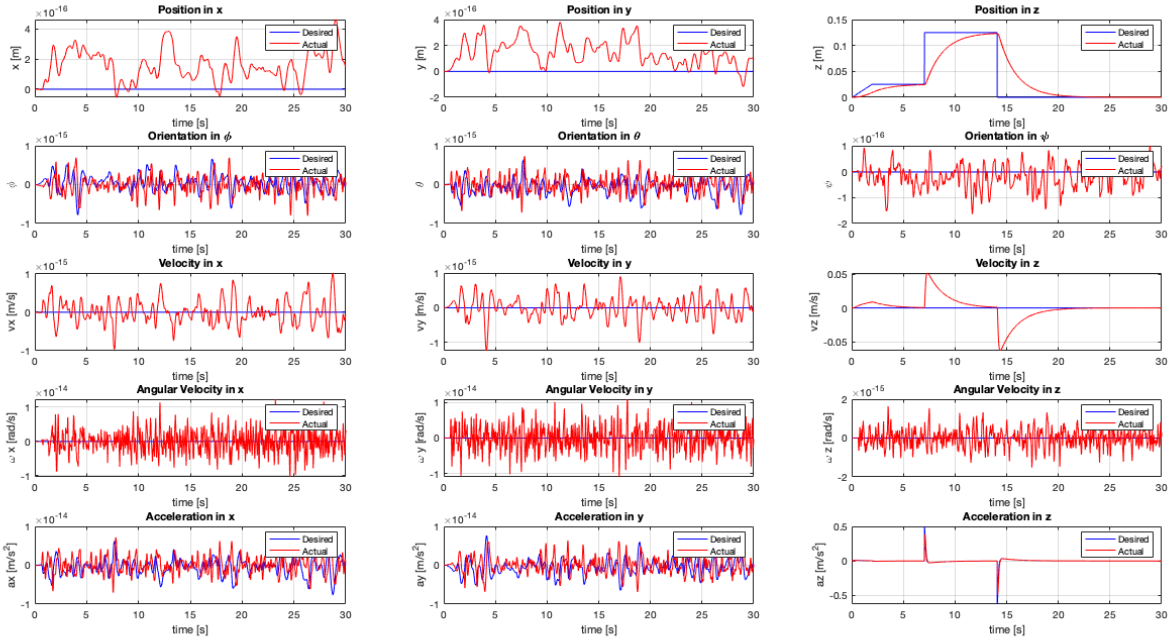
(Fig 5.4, Plot of position.)

In the first part of question 5 (heading =0), the system is moving only in the z direction. Therefore, changing the gain values (half the Kr) in attitude controller has minimal effect to the overall system responses. (i.e. the time domain performance characteristics stay the same.) However, one can still observe the effects in the oscillation magnitude in the orientations of the system. Where figure 5.1 they are in 1e -14 scale, and 1e-6 in figure 5.3.

Kp=[3, 3, 5]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]



(Fig 5.5, Plot of error.)



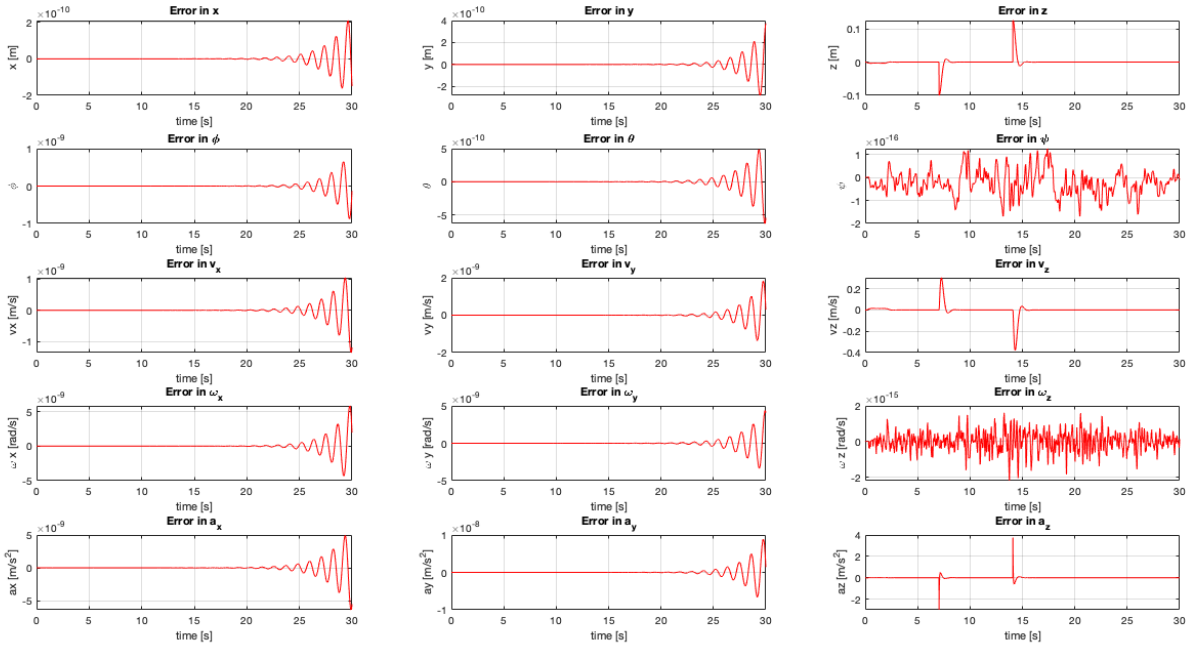
(Fig 5.6, Plot of position.)

When Kp is reduced, the responses are less keen, as shown in table 3. Rise time and settling time increase and max precent overshoot decreases compared to to table 2.

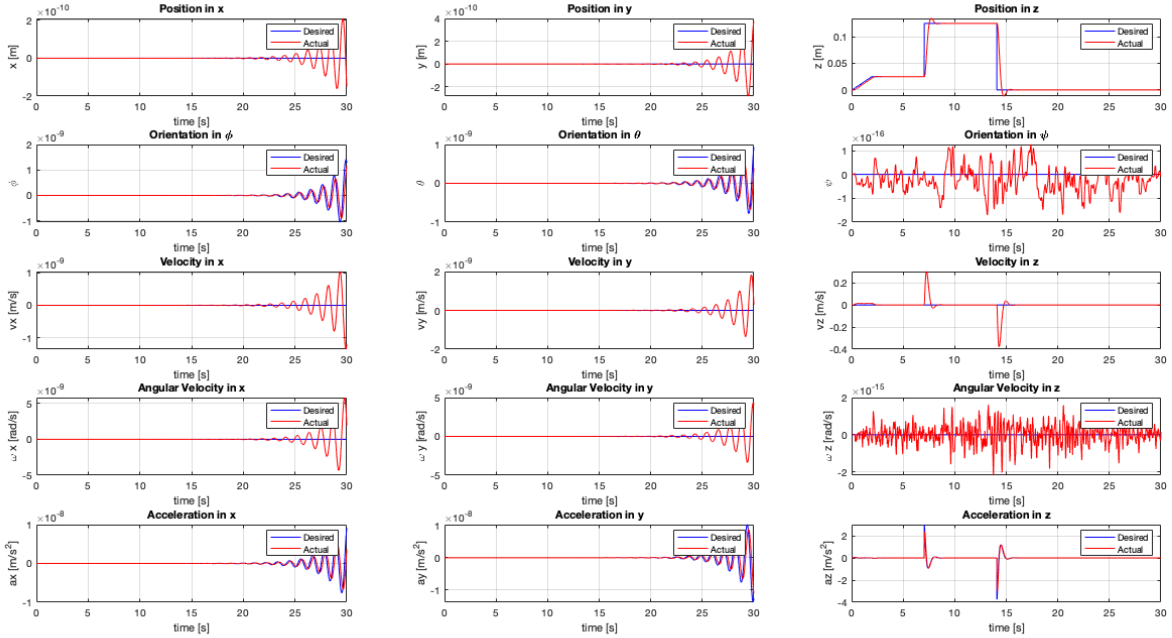
(Table 3, Time domain performance characteristics.)

Rise Time (time after hover)	3.6050s
Settling Time (time after hover)	4.0400s
Steady-state Value	0.125
Max percent overshoot	0

Kp=[30, 30, 30]	Kd=[4, 4, 7]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]



(Fig 5.7, Plot of error.)



(Fig 5.8, Plot of position.)

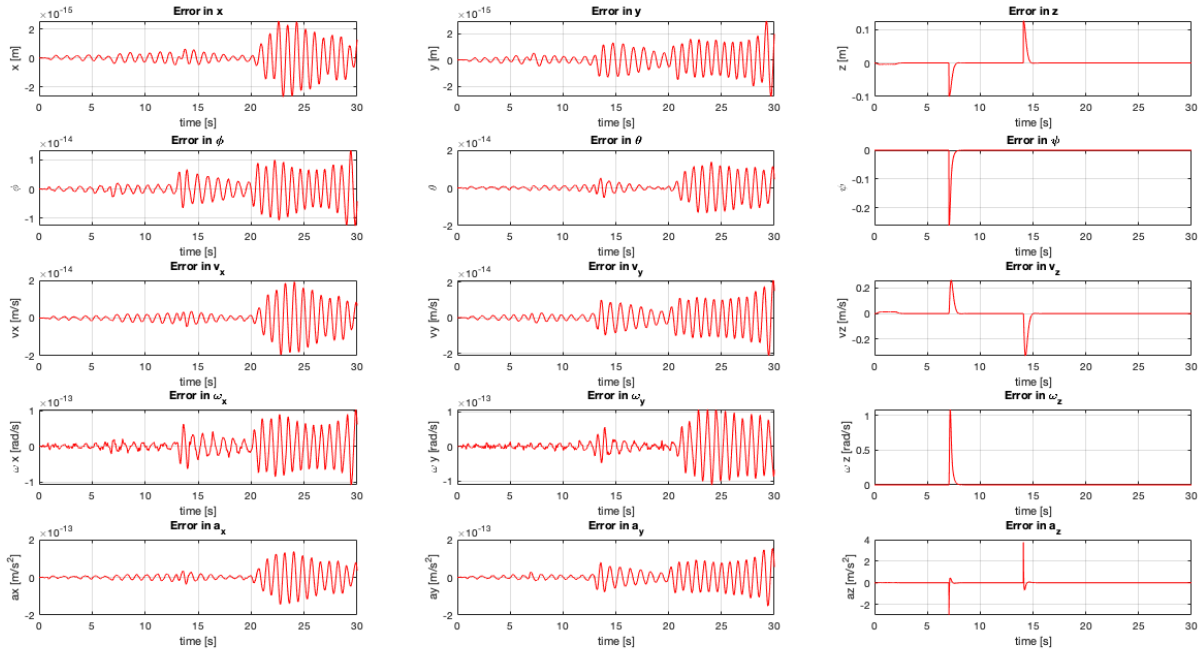
If Kd is tuned down, rise time is reduced while settling time and max percent overshoot are increased compared to table 2. There are oscillation at the end of the simulation. However, they are in 1e -10 scale, which can be ignored.

(Table 4, Time domain performance characteristics.)

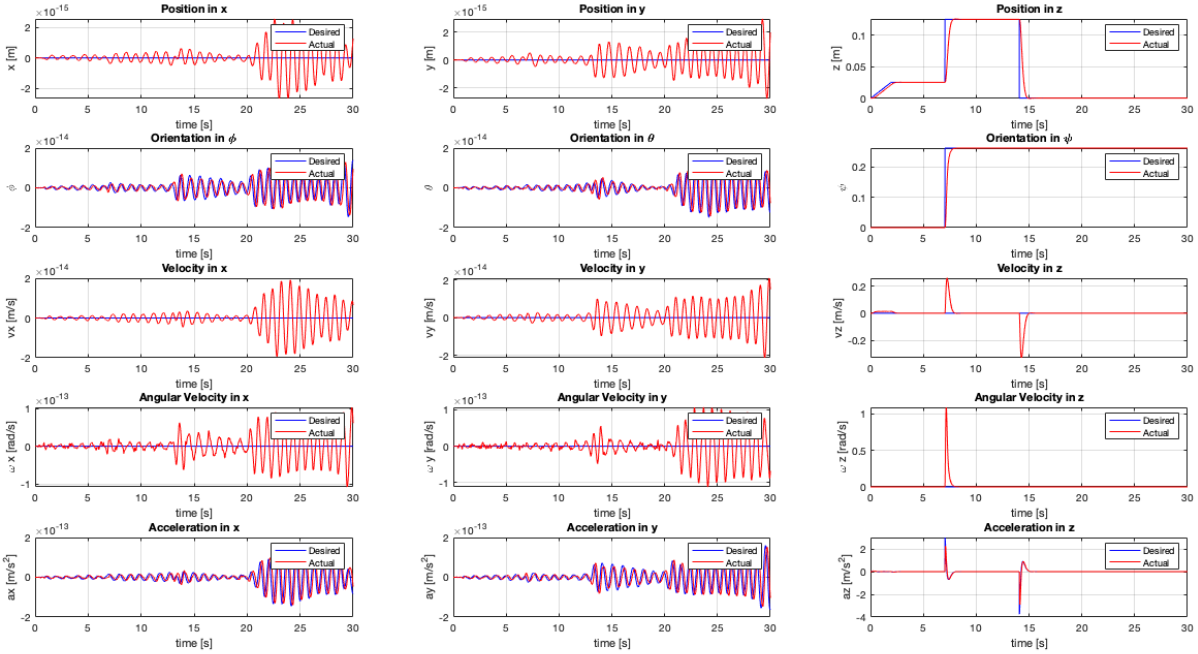
Rise Time (time after hover)	0.4450s
Settling Time (time after hover)	0.4600
Steady-state Value	0.125
Max percent overshoot	7.34

part 2 (heading =15 deg)

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]



(Fig 5.9, Plot of error.)



(Fig 5.10, Plot of position.)

(Table 5, Time domain performance characteristics.)

Rise Time (time after hover)	0.5450s
Settling Time (time after hover)	0.5700s
Steady-state Value	0.125
Max percent overshoot	0.43

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[100, 100, 40]	Kw=[30, 30, 17.88]

(Table 6, Time domain performance characteristics.)

Rise Time (time after hover)	0.5400s
Settling Time (time after hover)	0.5650s
Steady-state Value	0.125
Max percent overshoot	0.44

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[25, 25, 14]

(Table 7, Time domain performance characteristics.)

Rise Time (time after hover)	0.5450s
Settling Time (time after hover)	0.5750s
Steady-state Value	0.125
Max percent overshoot	0.43

Kp=[3, 3, 5]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]

(Table 8, Time domain performance characteristics.)

Rise Time (time after hover)	3.6600s
Settling Time (time after hover)	4.0950s
Steady-state Value	0.125
Max percent overshoot	0

Kp=[30, 30, 30]	Kd=[4, 4, 7]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]

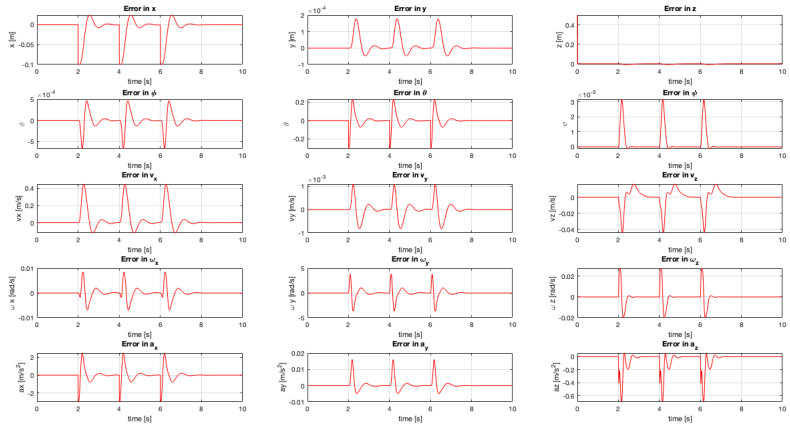
(Table 9, Time domain performance characteristics.)

Rise Time (time after hover)	0.4550s
Settling Time (time after hover)	0.4650s
Steady-state Value	0.125
Max percent overshoot	7.24

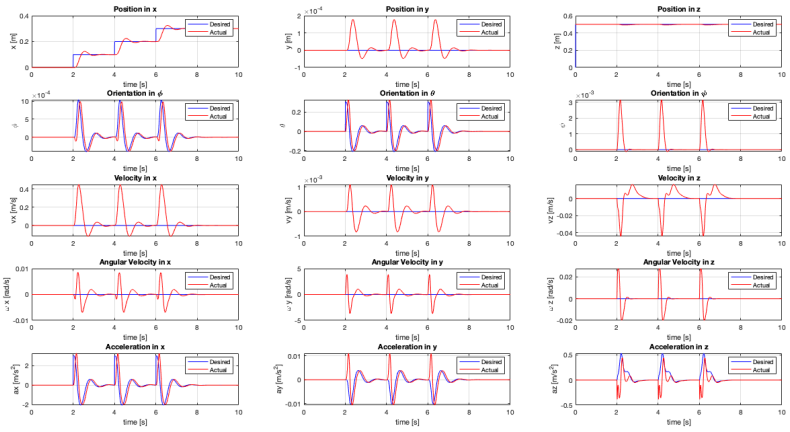
At this part of the problem, rise time and settling time are determined mainly by Kp and Kd. By looking at table 6-9, even with different Kd, Kr, and Kw, as long as Kp stays the same, the time domain performance characteristics are similar. For the time response in psi orientation, it is like Kp and Kd controlling position responses. If Kr increases, rise time for psi orientation decreases and settling time increases. Additionally, Kr and Kw have effects on the responses of the orientations of the robot but have little to do with the time domain performance characteristics in position x, y and z, which is proved in table 6-9.

Q6)
Overall, LQR method takes more time than PD control to run the simulation. The reason is that at each time step LQR function has to calculate multiple 12 by 12 matrixes, which in PD control there are only several scalar multiplications and subtractions during each time step.

question 2
PD control:

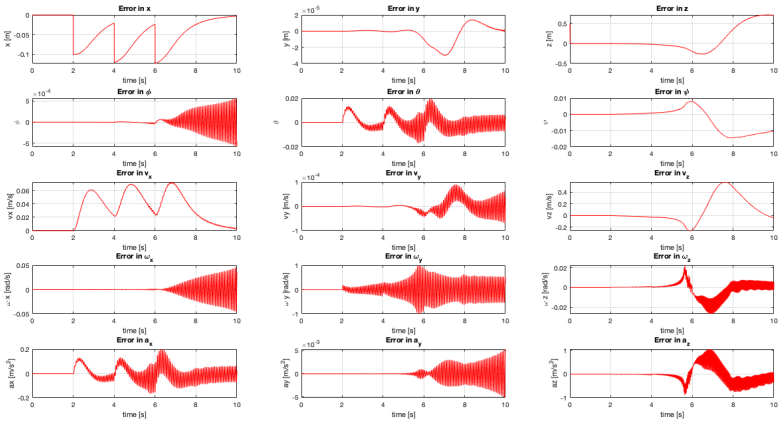


(Fig 6.1, Plot of error for PD control.)

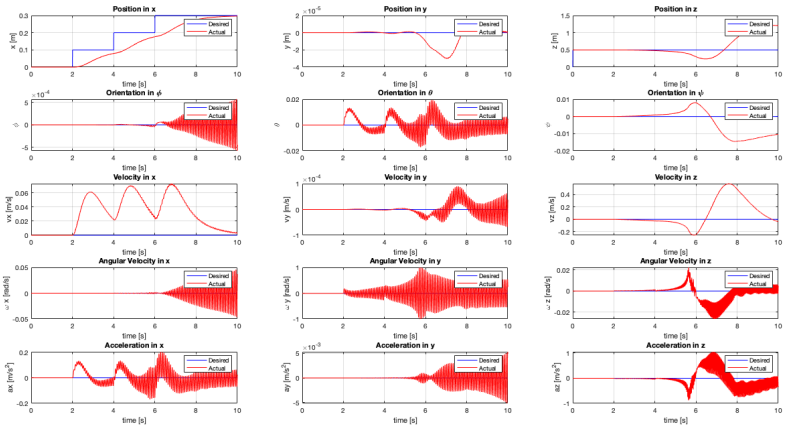


(Fig 6.2, Plot of position for LQR control.)

LQR control:



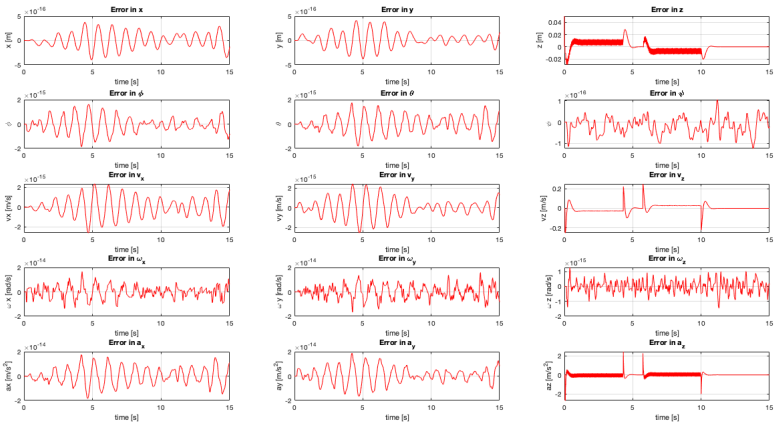
(Fig 6.3, Plot of error for PD control.)



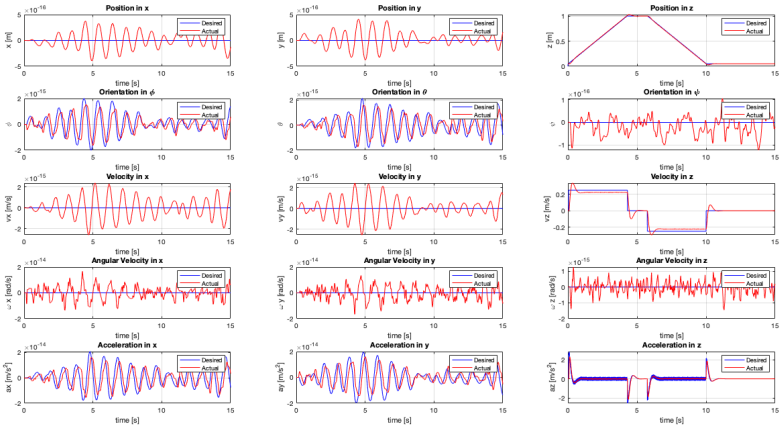
(Fig 6.4, Plot of position for LQR control.)

PD control is better than LQR control for question 2. If we try to decrease R (less cost on the input) and increase Q, the system breaks down at some point. There is a limitation when using LQR control.

question 3
PD control:

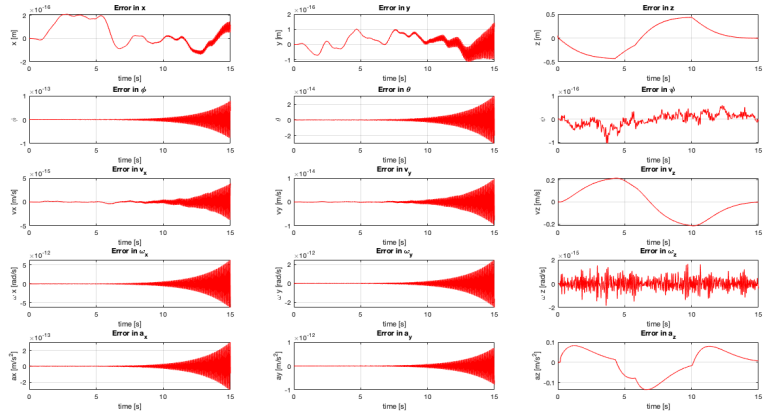


(Fig 6.5, Plot of error for PD control.)

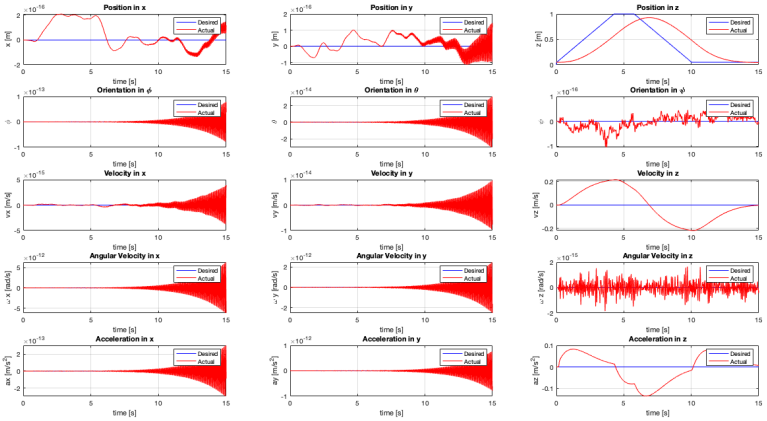


(Fig 6.6, Plot of position for PD control.)

LQR control:



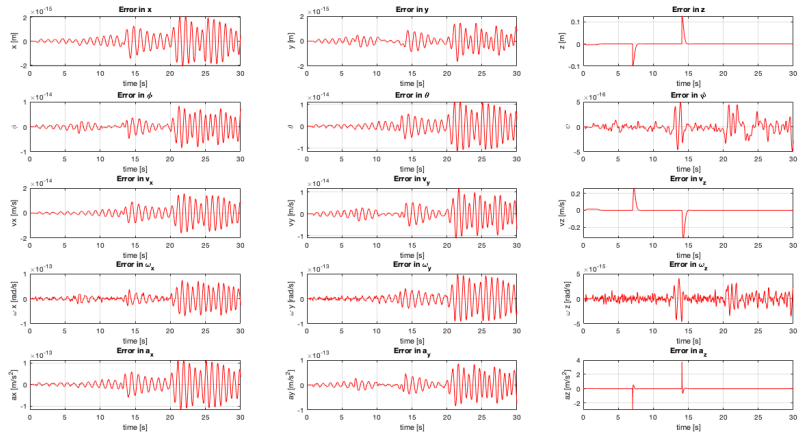
(Fig 6.7, Plot of error for LQR control.)



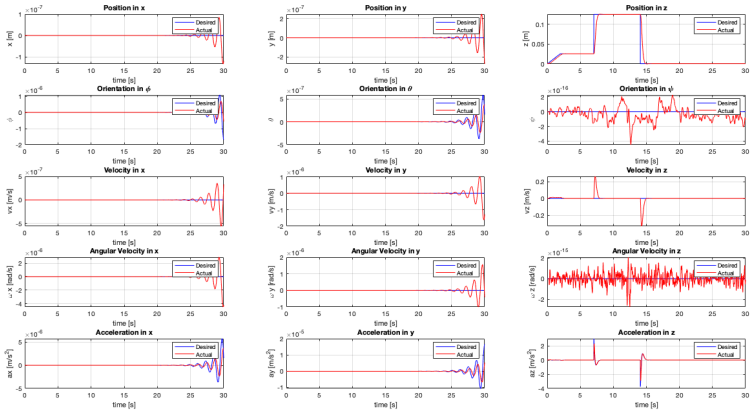
(Fig 6.8, Plot of position for LQR control.)

For question 3, PD control is still a better way for the robot a track the trajectory. In figure 6.7, 6.8 Q=0.1*eye(12), R=0.2*eye(4). It is the limit of the LQR control. If we try to decrease R (or increase Q) beyond this value, it would lead to a calculation error in LQR function.

question 5
part 1 (heading=0 deg)
PD control:

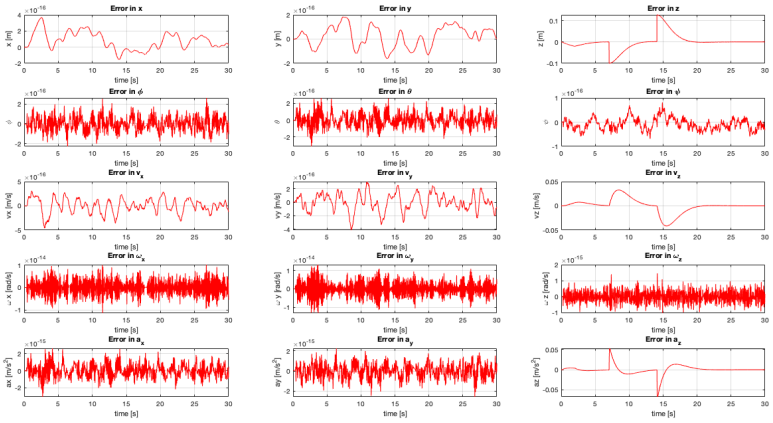


(Fig 6.9, Plot of error for PD control.)

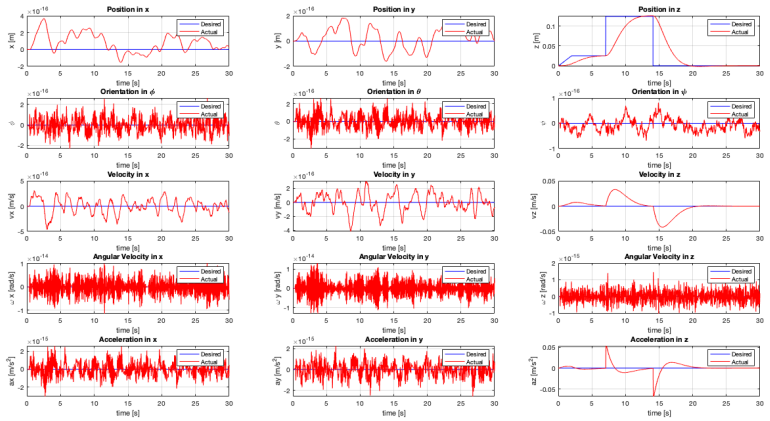


(Fig 6.10, Plot of position for PD control.)

LQR control:



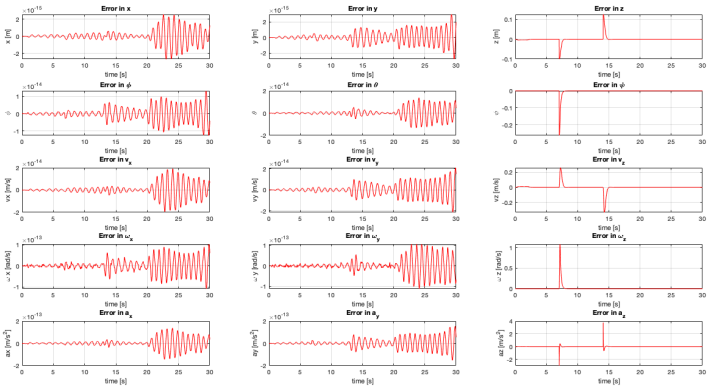
(Fig 6.11, Plot of error for LQR control.)



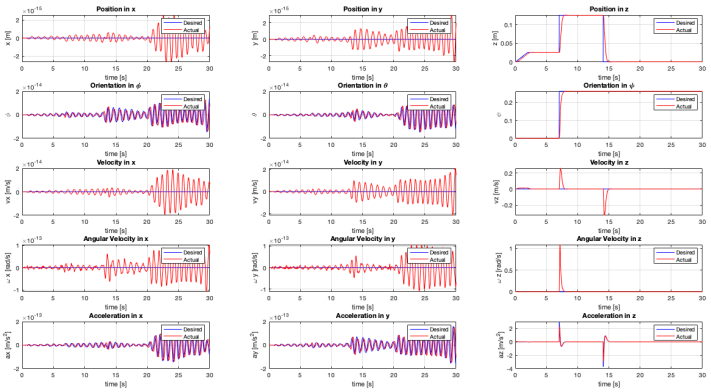
(Fig 6.12, Plot of position for LQR control.)

In question 5, PD control is a better controller if we compare the position plots in figure 6.10 and 6.12. The robot using PD control tracks the trajectory better. However, if we look at the maximum error plots in figure 6.10 and 6.12, one could notice that with LQR control, the maximum error is smaller than that in PD control.

part2 (heading =15 deg)
PD control:

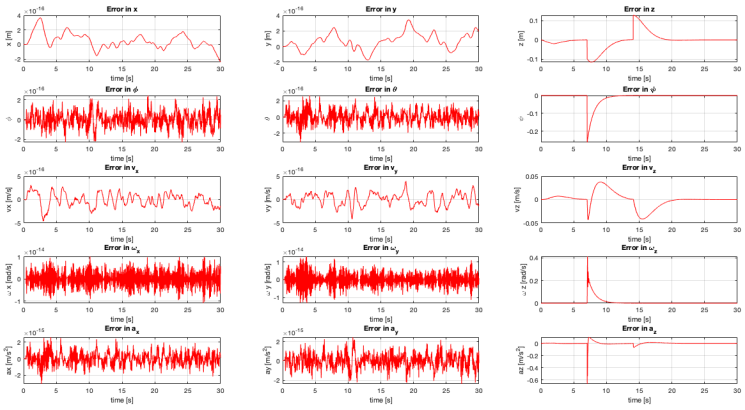


(Fig 6.13, Plot of error for PD control.)

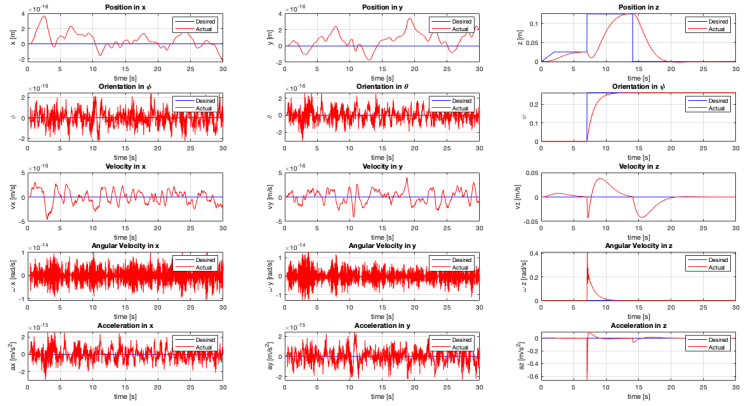


(Fig 6.14, Plot of position for PD control.)

LQR control:



(Fig 6.15, Plot of error for LQR control.)



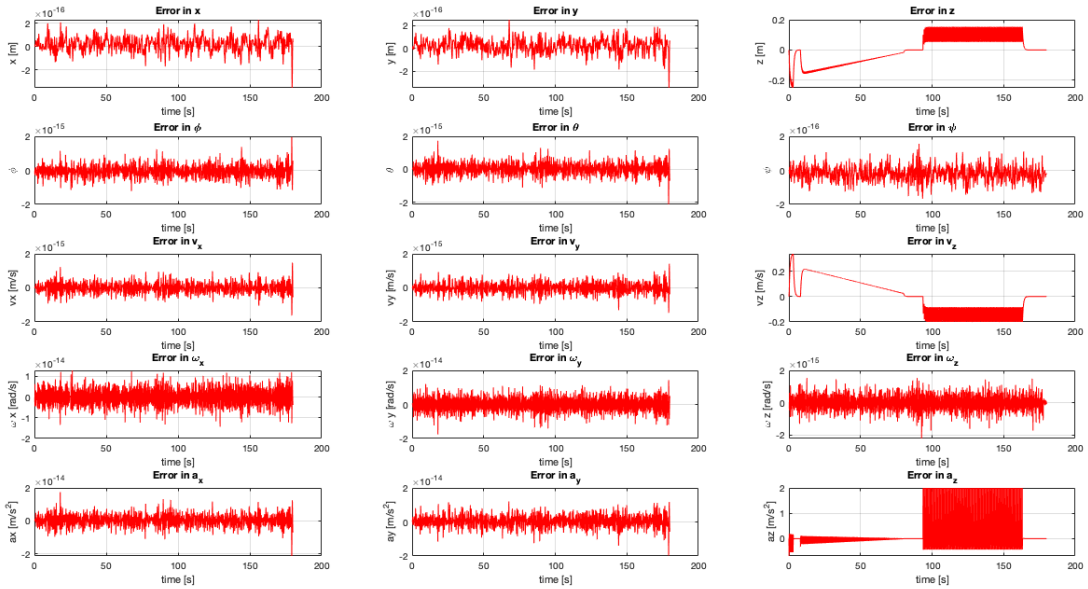
(Fig 6.16, Plot of position for LQR control.)

For part 2 in question 5, when there is a desired heading input to the robot, it effects the tracking performance of the LQR method. As shown in figure 6.16, at t=7s, there is a sudden drop in actual z position, and it coincides with the time when there is a step input to the heading.

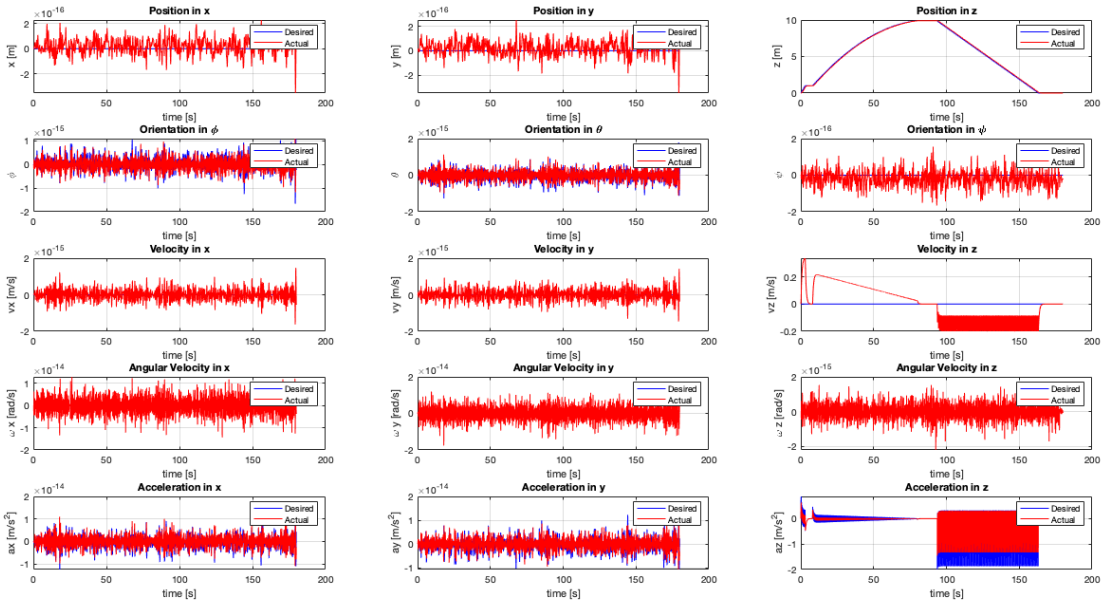
Theoretically, LQR control should be able to tune down R to minimize the cost of input and give out similar results. However, there is calculation issue that limits the LQR performance. Overall, PD control has a better performance in tracking. On the other hand, it is more straightforward to tune a LQR control. Either tune up R or Q to weight the cost of input or the state(error). In PD control, tuning Kw and Kr might influences the effects of Kp and Kd. One has to tune back and forward to get the optima result.

Q7)

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]	track_time=80s



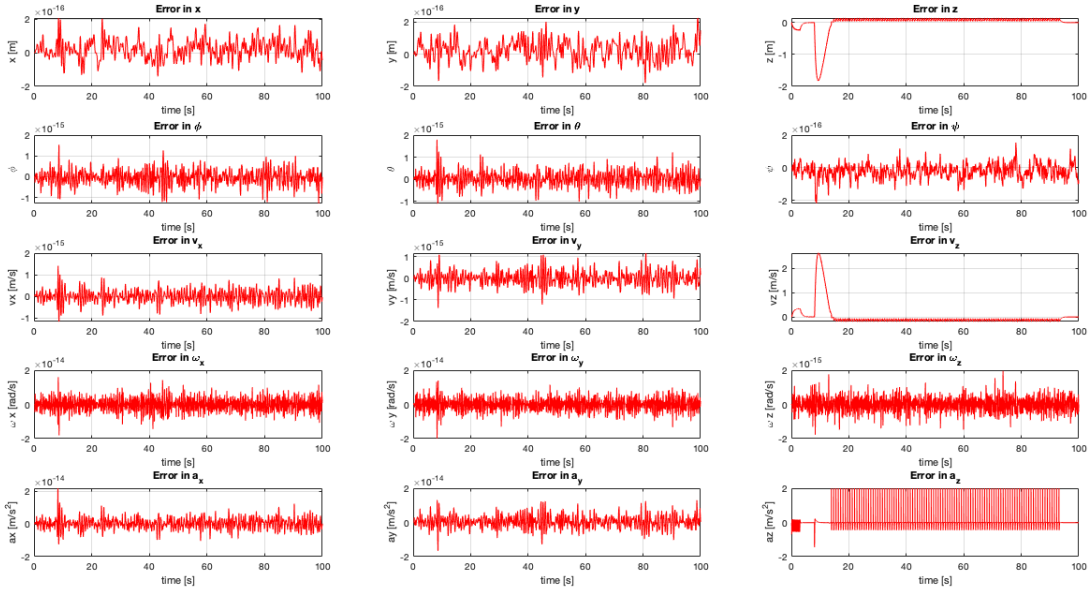
(Fig 7.1, Plot of error.)



(Fig 7.2, Plot of position.)

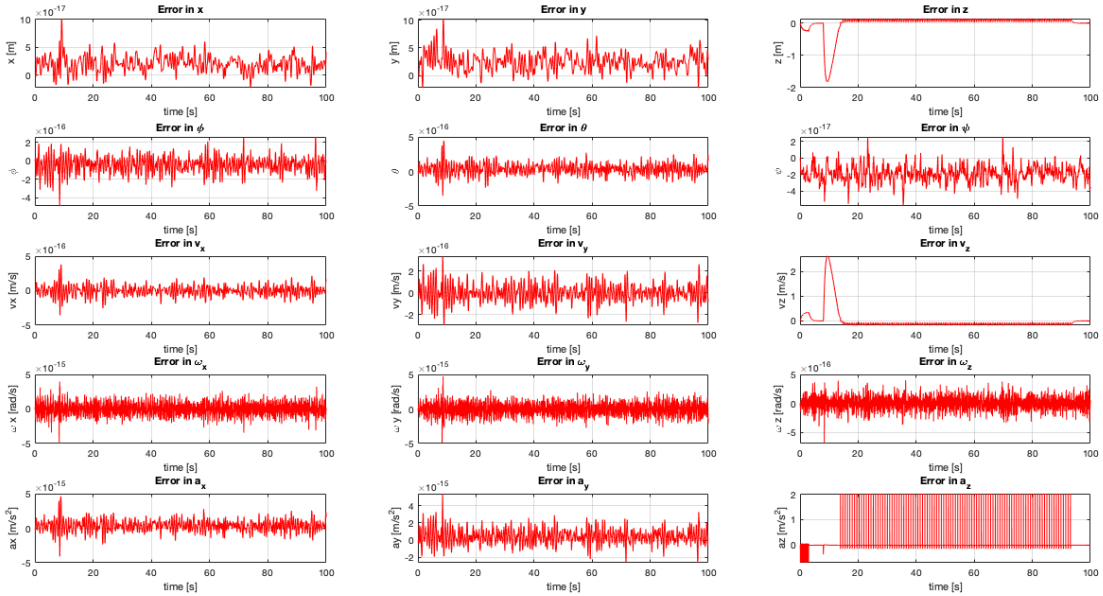
Yes, the robot tracks the system very well. The maximum and minimum accelerations are 0.6349 m/s^2 and -1.4567 m/s^2 . As in figure 7.2, the robot takes off and hovers at z=1. After 5s, the robot follows a second order polynomial function to z=10. The robots hovers at z=10 for 5s and descends to z=0. The acceleration of the robot comes from three main reasons, the trajectory, waypoints time discretion, tracking timespan. Firstly, as one can see in figure 7.2, polynomial trajectory tends to have less error. Therefore, the robot would need less velocity and acceleration to compensate for the error. Secondly, if waypoints time discretions are larger, maximum (minimum) acceleration would increases (in magnitude). Larger waypoints time discretions will lead to larger error in position, and as the same reason for trajectory, it needs higher acceleration to compensate for the errors. Last but not least, longer the tracking time span is, lower (in magnitude) the maximum(minimum) acceleration is.

Kp=[30, 30, 30]	Kd=[6.6, 6.6, 9]	Kr=[190, 198, 80]	Kw=[30, 30, 17.88]	track_time=0.1s



(Fig 7.3, Plot of error.)

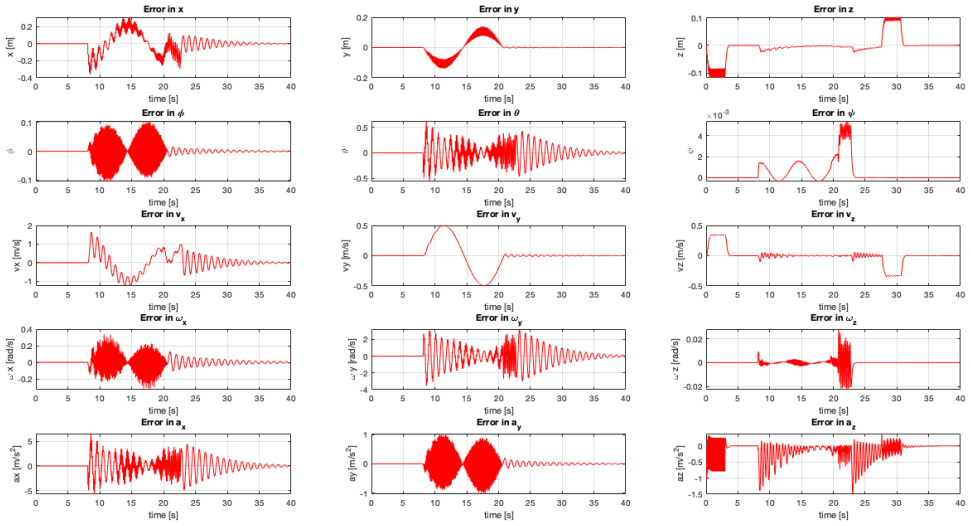
When the track time is set to 0.1s, the robot begins to have a large error in z direction as shown in figure 7.3. The maximum acceleration is 4.7927 m/s^2, which is way larger than the previous result. When the acceleration bound is set beyond this value, the system begin to exhibit degraded tracking accuracy.



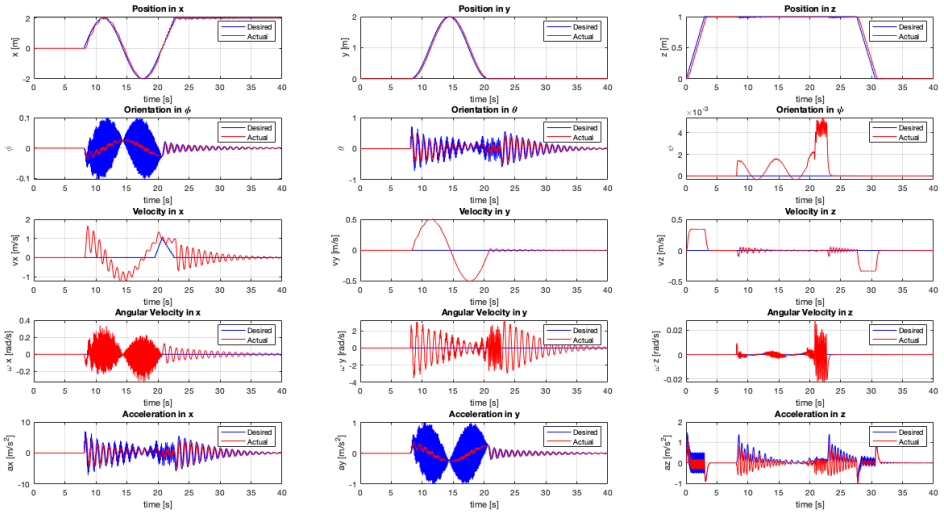
(Fig 7.4, Plot of error.)

If we increase the motor gain (upgrading the motor), that means the motor can generate more force at a given error between desired rpm and current rpm of the motors. In figure 7.4, the motor constant is increased from 36.5 to 200. The maximum acceleration is reduced to 4.28. The reason for this is that, the motors can generate higher rpm which reduces the error between actual and desired trajectory, and error in z velocity, as in figure 7.4. Therefore, the robot needs less acceleration to compensate for the error.

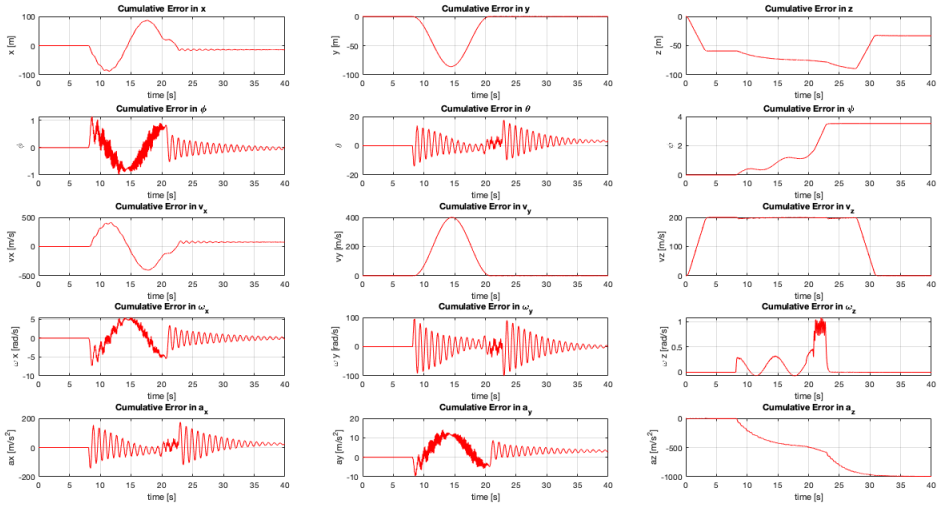
Q8)
part 1



(Fig 8.1, Plot of error.)

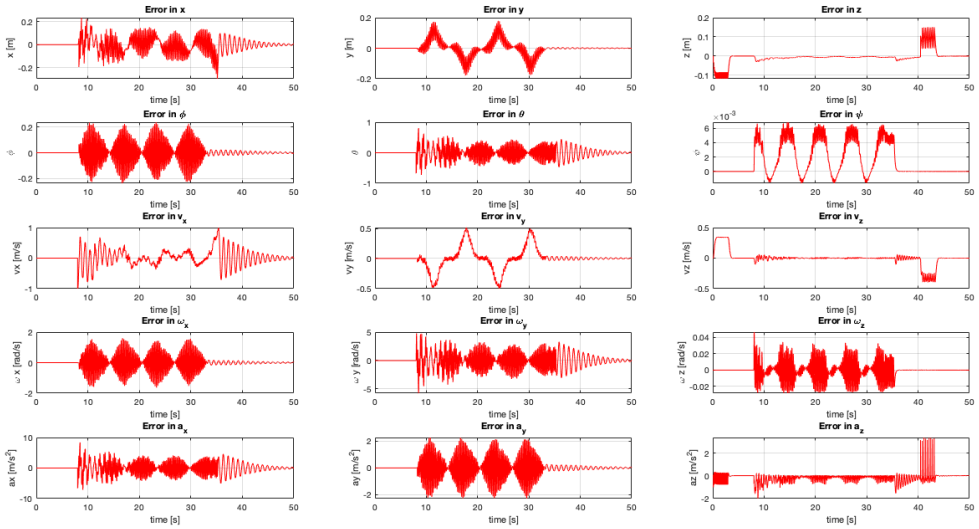


(Fig 8.2, Plot of position.)

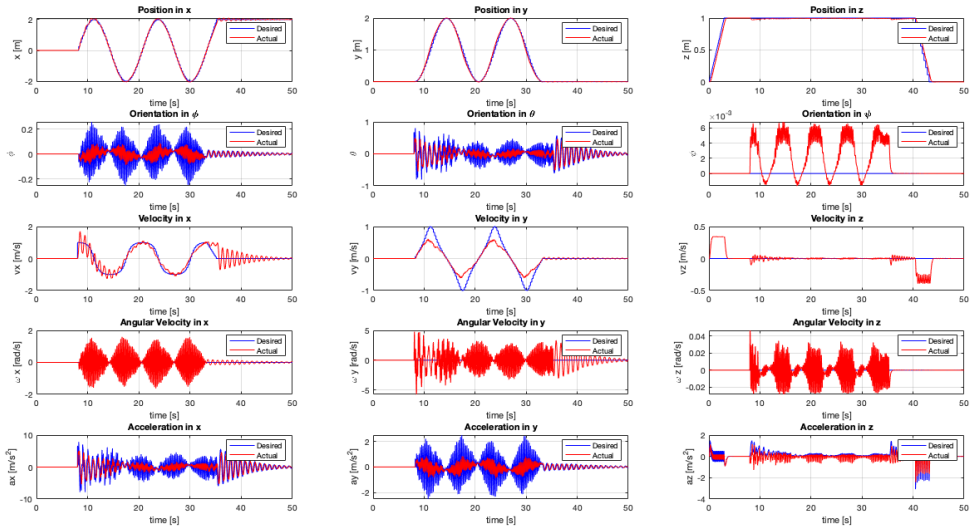


(Fig 8.3, Plot of cumulative error.)

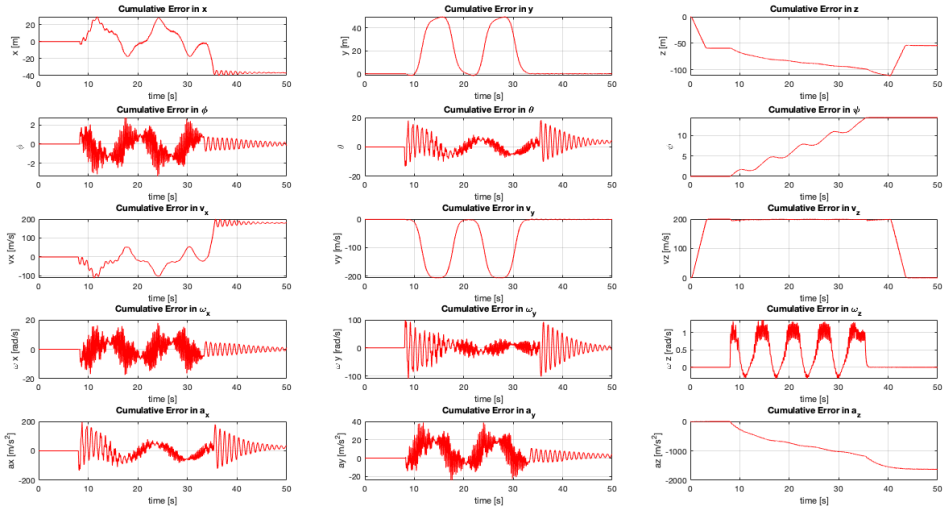
part 2



(Fig 8.4, Plot of error.)

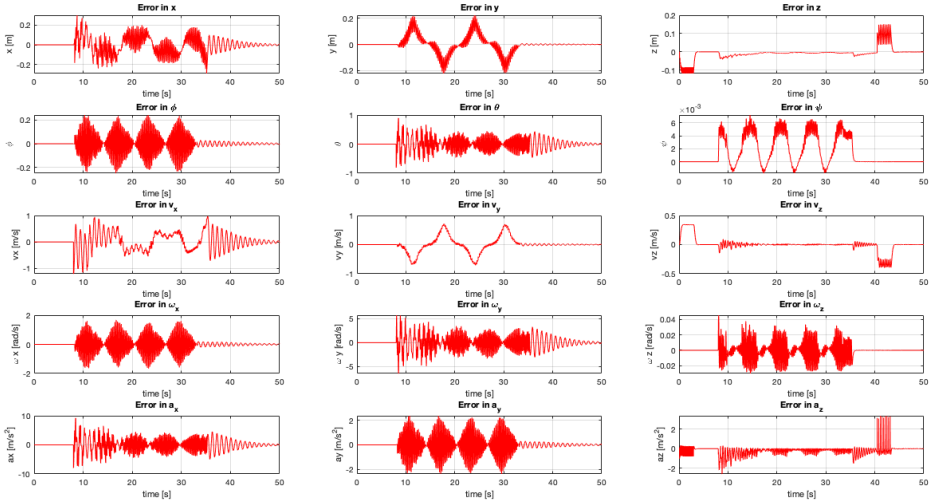


(Fig 8.5, Plot of position.)

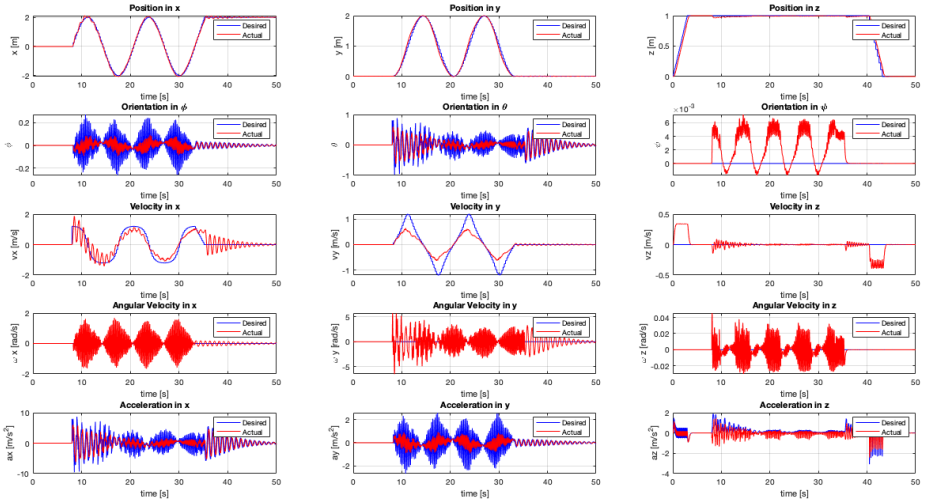


(Fig 8.6, Plot of cumulative error.)

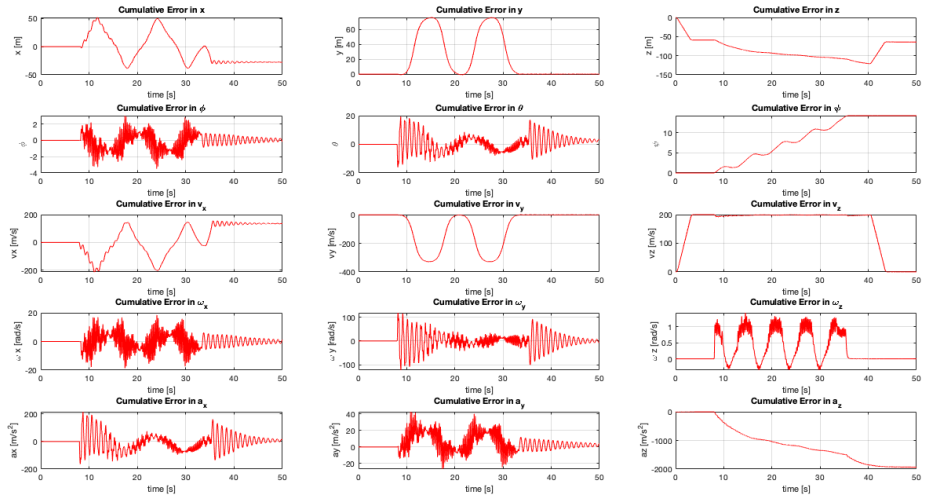
When the tracking velocity increases the errors in position and velocity increase, as shown in figure 8.7-9 (v=1.2m/s). It is more obvious to look at the cumulative error plots, "cumulative error in x", "cumulative error in y", "cumulative error in vx", "cumulative error in vy". The system becomes unstable when tracking velocity increases to 1.5 m/s. If velocity decreases, then the errors would decrease.



(Fig 8.7, Plot of error.)

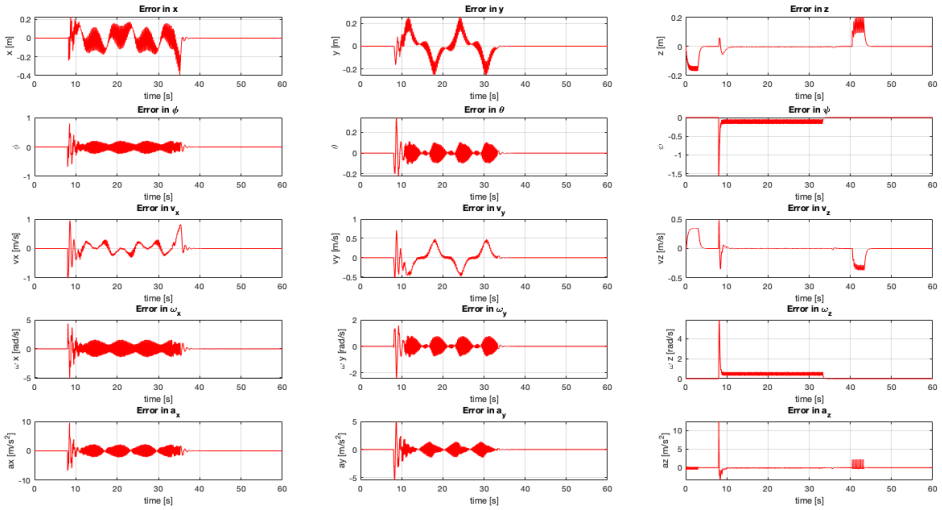


(Fig 8.8, Plot of position.)

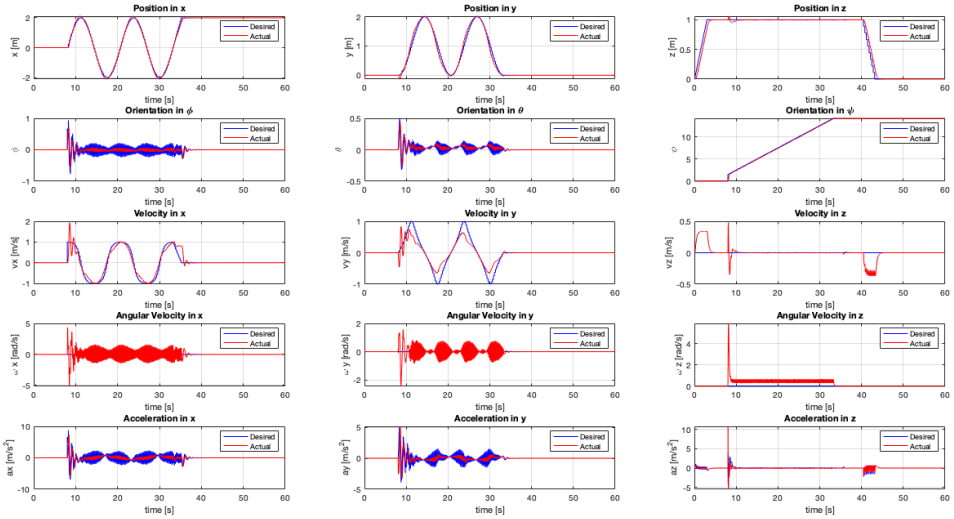


(Fig 8.9, Plot of cumulative error.)

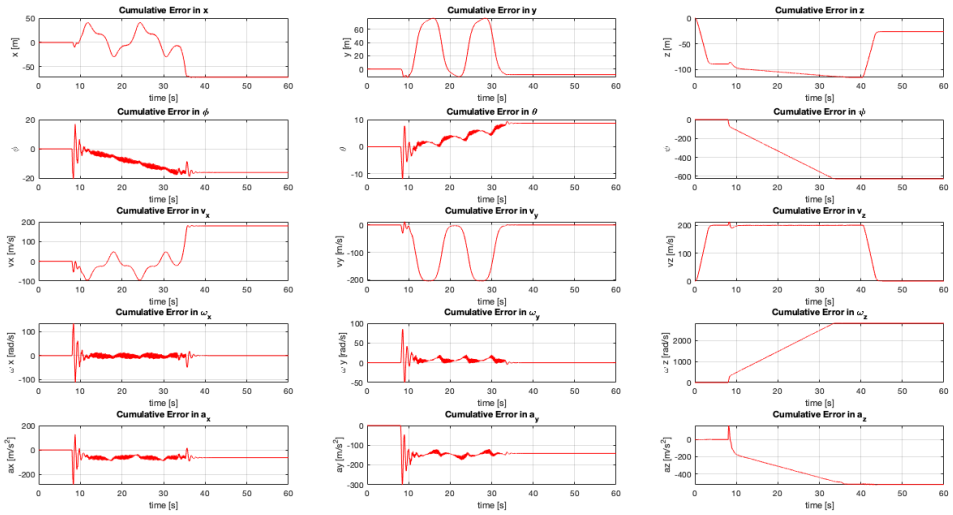
Q9)



(Fig 9.1, Plot of error.)



(Fig 9.2, Plot of position.)



(Fig 9.3, Plot of cumulative error.)

The changes are more obvious if we look at the cumulative error plots. Cumulative errors in x and y position are increased by around 50 percent. (There is no immense effect on z direction, since both question 8 and 9, the robot flies at the same height.) Most of the errors increase, but the cumulative error in theta(pitch) decreases. The reason is because while tracking an ellipse trajectory, the robot has to lean toward the origin of the ellipse to provide enough centripetal force. When the heading is fixed, the roll and pitch angle of the robot are constantly changing along the trajectory to lean toward the origin. However, when the heading of the robot is always aiming toward the origin of the ellipse, the desired roll and pitch angles only change within a smaller boundary.