16-665 Robot Mobility (Fall 2018)
Air Mobility Project Report

Heethesh Vhavle
*(Collaborated with Ganesh Iyer)**

Carnegie Mellon University
November 4, 2018

Problem 1 - Solution

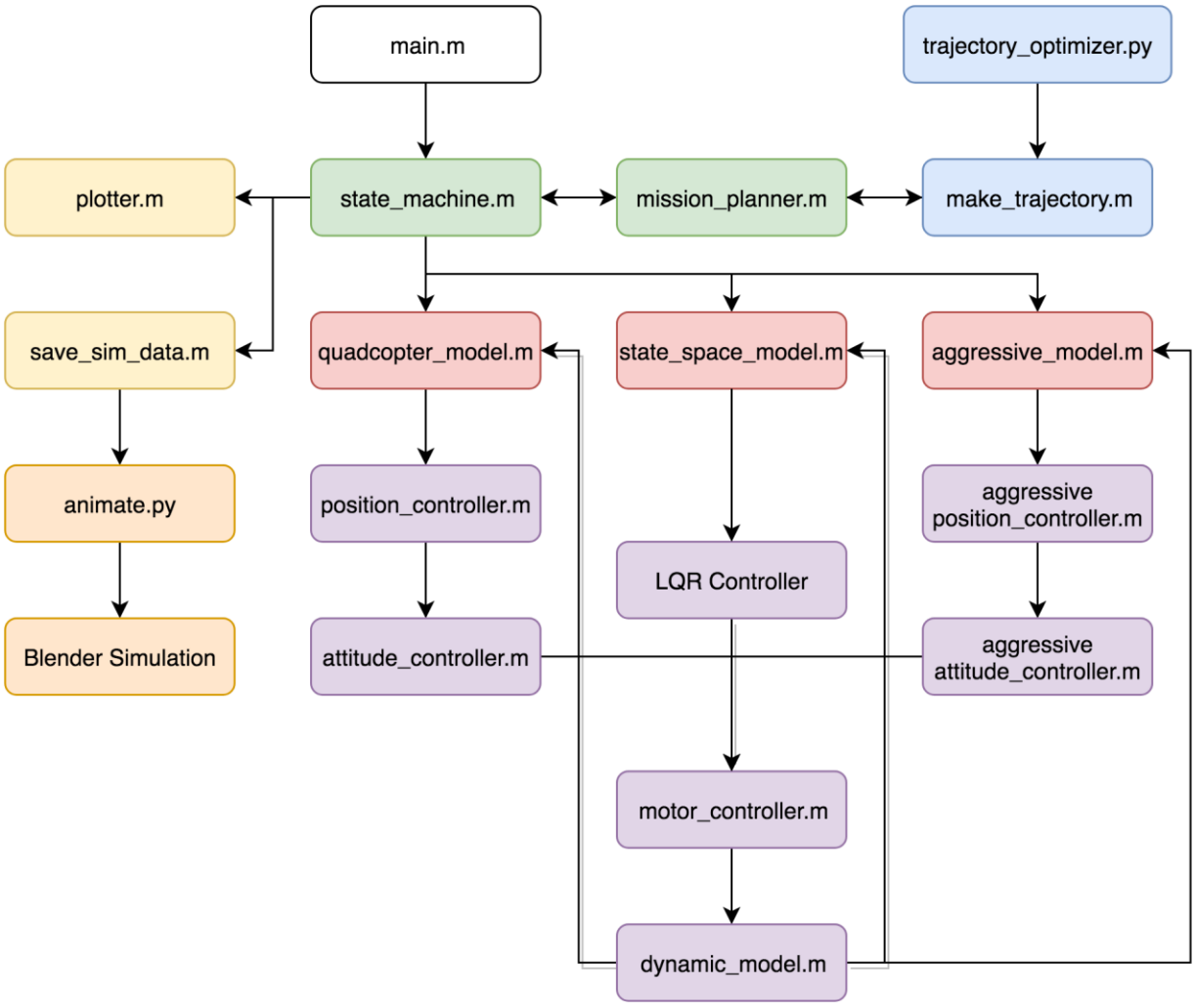


Figure 1: Software pipeline showing the various MATLAB modules
(Note: The information flow is not shown here for simplicity and is explained in detail below)

** Collaborated partly on attitude controller development and trajectory optimization in problem 8.*

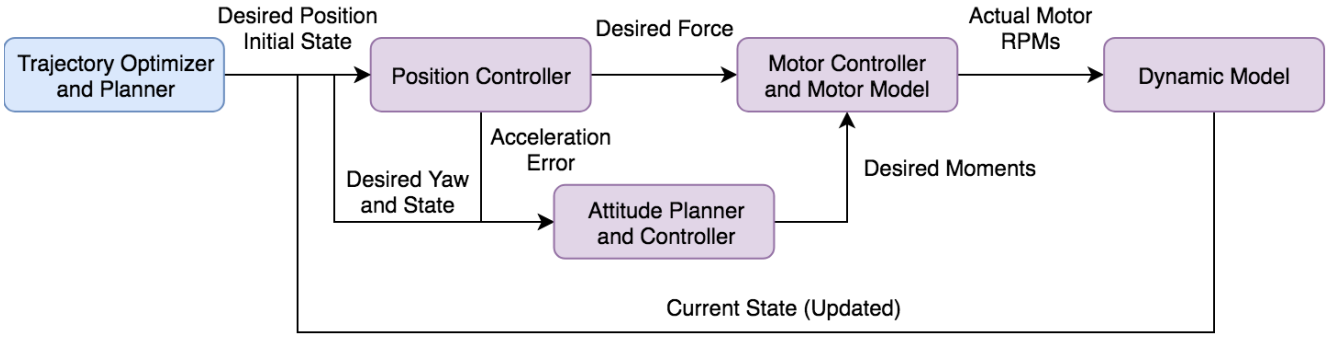


Figure 2: Quadcopter cascaded control architecture

The program flow starts with *main.m* which calls the *state_machine.m* script with a given mission. The mission planner plans the entire machine with a set of trajectories for takeoff, hover, tracking and landing, along with the type of model and gains for executing each trajectory. The actual trajectories are generated by *make_trajectory.m* which may occasionally load precomputed trajectory data generated by *trajectory_optimizer.py* for minimum snap trajectories in *.mat* format.

The state machine (which is explained in detail later) executes the planned trajectories one after the other with the help of three models designed in this project, namely, Linearized Quadcopter PD Model, State Space LQR Model and Non-Linear Aggressive Quadcopter PD Model. The cascaded control system architecture is shown above in *Figure 2*. The PD models first invokes the *position_controller.m*, with the current (state) and desired position to compute the desired force and error in acceleration. The inner loop or the *attitude_controller.m* is invoked with the current (state) and desired yaw to compute the desired moments about each of the axes. Finally, the *motor_controller.m* is called which computes the required motor RPMs with the help of the motor mixing matrix. The motor controller also applies the motor constant, clips the RPMs and returns the actual motor RPMs. This is then sent to *dynamic_model.m* which computes the actual forces and moments and updates the current state which is required for closed loop control in the next iteration. The aggressive model is exactly same, with the only difference being that the linearized assumptions around hover mode in the position and attitude controllers are relaxed. The LQR model pipeline is similar, but here, all the control and state updating are done within *state_space_model.m*.

Once the state machine executes all the trajectories and the quadcopter returns to the idle state, the loop is complete, and the simulation data is saved in *.csv* format. The various graphs are plotted showing the desired and the actual states for the tracking mode. A 3D simulation is done with the help of Blender, along with the helper script, *animate.py* which load the saved simulated data and creates keyframes for animation. Apart from these scripts, there are various other utility scripts not shown in *Figure 1*, which are used for symbolic expression generation and data handling.

Problem 2 - Solution

The linearized feedback control policy was implemented, and a simulation environment was developed. Please note that, all the plots below are part of the tracking mode of the state machine developed. The quadcopter initially dips in Z-axis for about 0.2m which is not visible in the plots below as it is part of the idle state and it was not feasible to restructure the code to plot the same. A waypoint of (0, 0, 0.5, 0) is given at *t=3s* to make the robot hover. Increments of 10 cm waypoints in the x direction are given at *t=4,7,10,13s*.

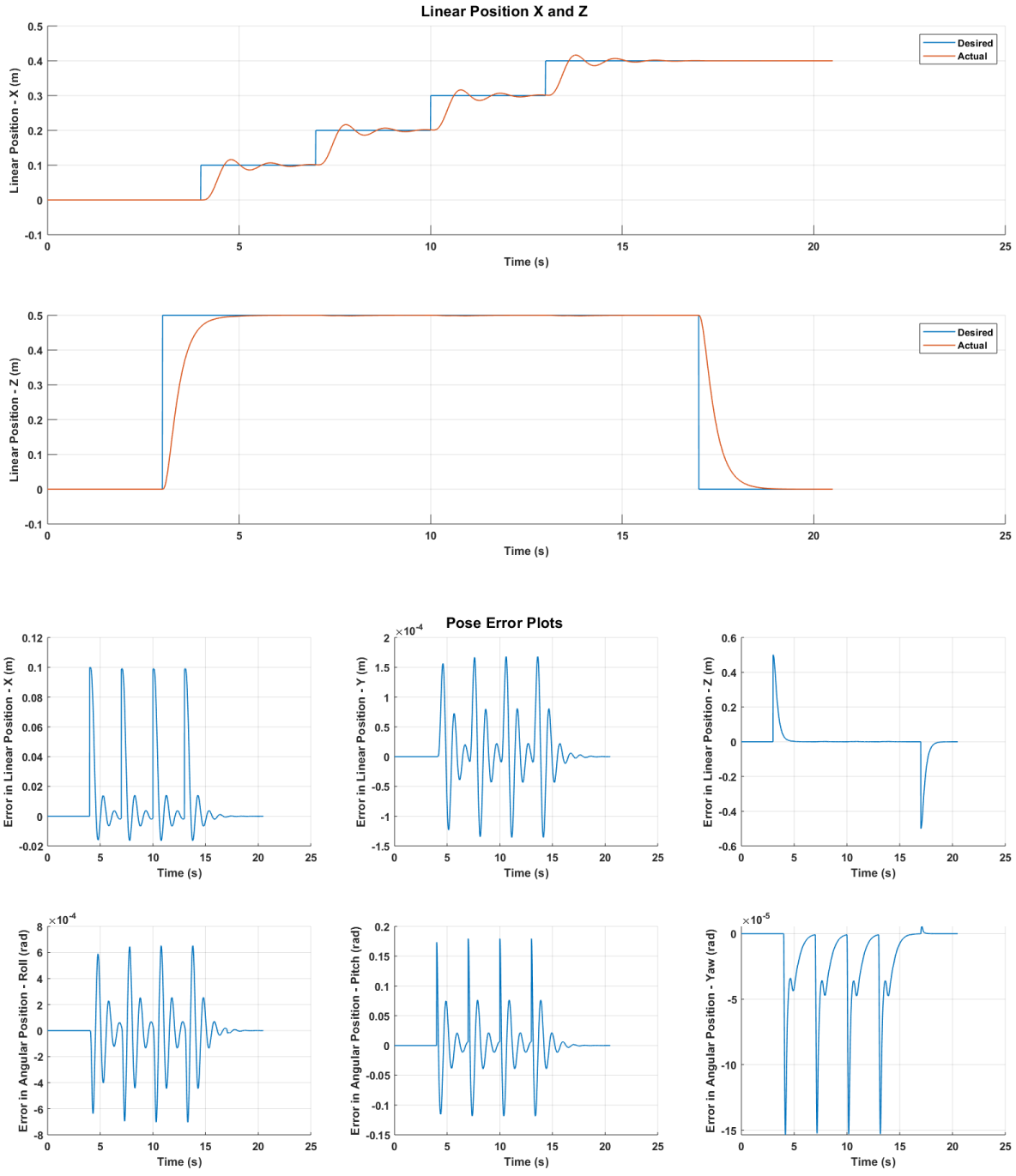


Figure 3: Position and pose error plots for PD gains set 1

Gain Set 1	***X/Roll***	***Y/Pitch***	***Z/Yaw***
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

The robot tracks the trajectory in Z axis with almost no oscillations around the waypoint and it is over-damped, however along the X axis waypoints, some oscillations and a minor overshoot can be seen.

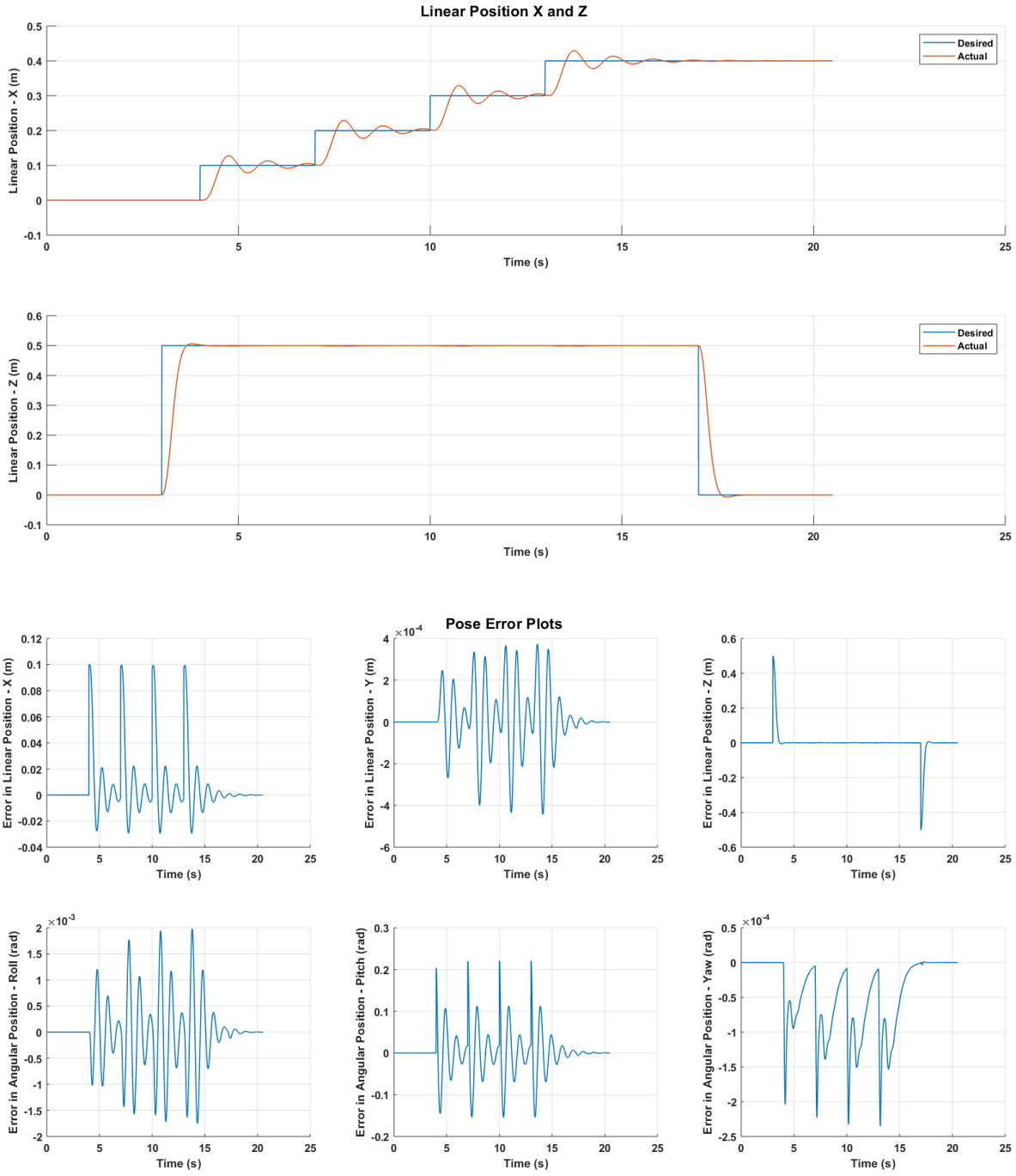


Figure 4: Position and pose error plots for PD gains set 2

Gain Set 2	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	**20.0**	17.0	**42.0**
Kd	6.6	6.6	**10.0**
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

By increasing the Kp in Z axis and minor increase in Kd, the rise time decreased, and a minor overshoot can be seen now. Increasing the proportional gain for X axis, increased the amplitude of oscillations.

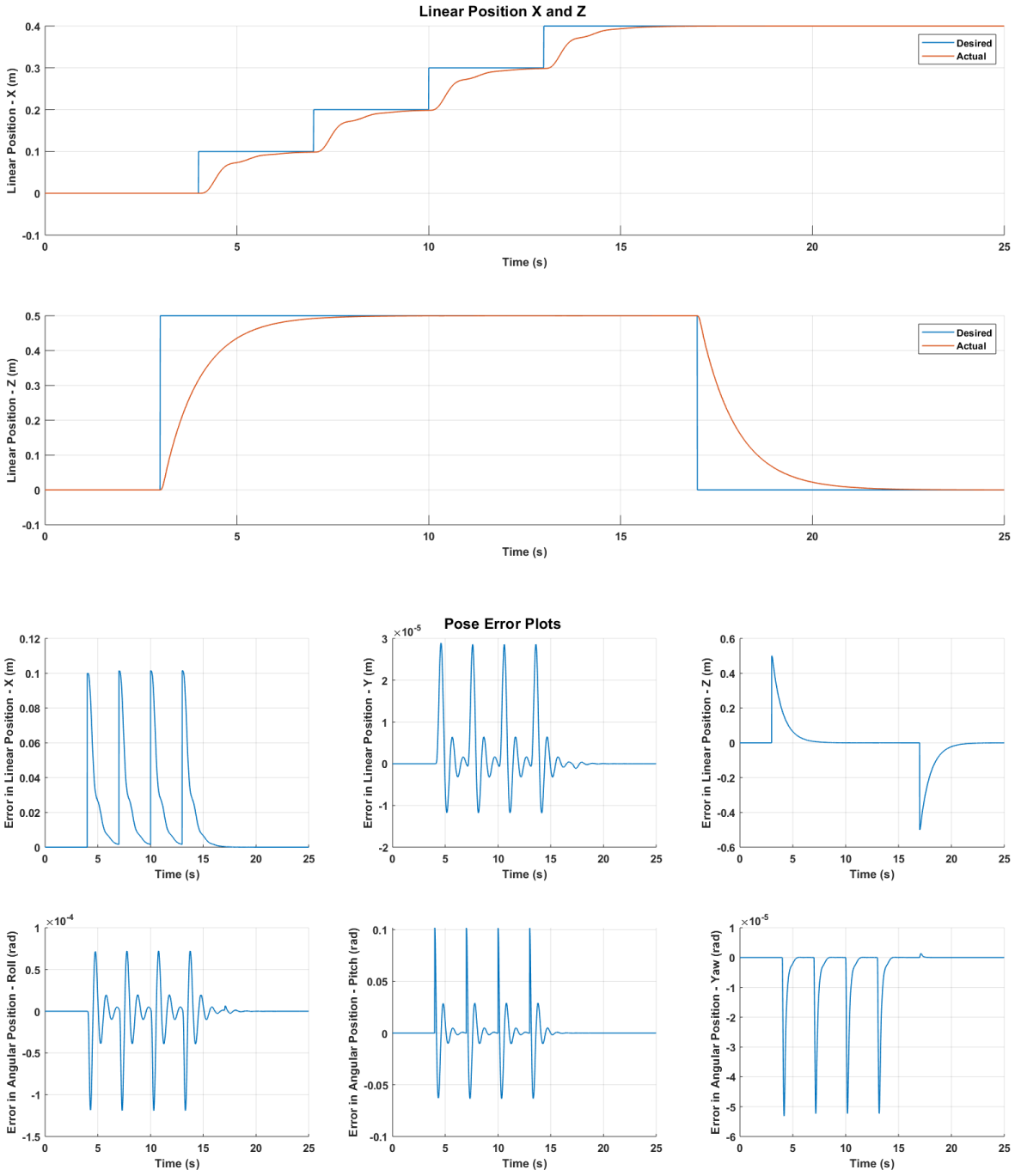


Figure 5: Position and pose error plots for PD gains set 3

Gain Set 3	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	**10.0**	17.0	20.0
Kd	**8.0**	6.6	**20.0**
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

By increasing the Kd in Z axis, the rise time significantly increased, and the system is over-damped with a slow response. Similarly, for the X-axis, on reducing Kp and increasing Kd slightly, although the amplitude of oscillations reduced, the response is very slow, and the robot takes longer to track the trajectory.

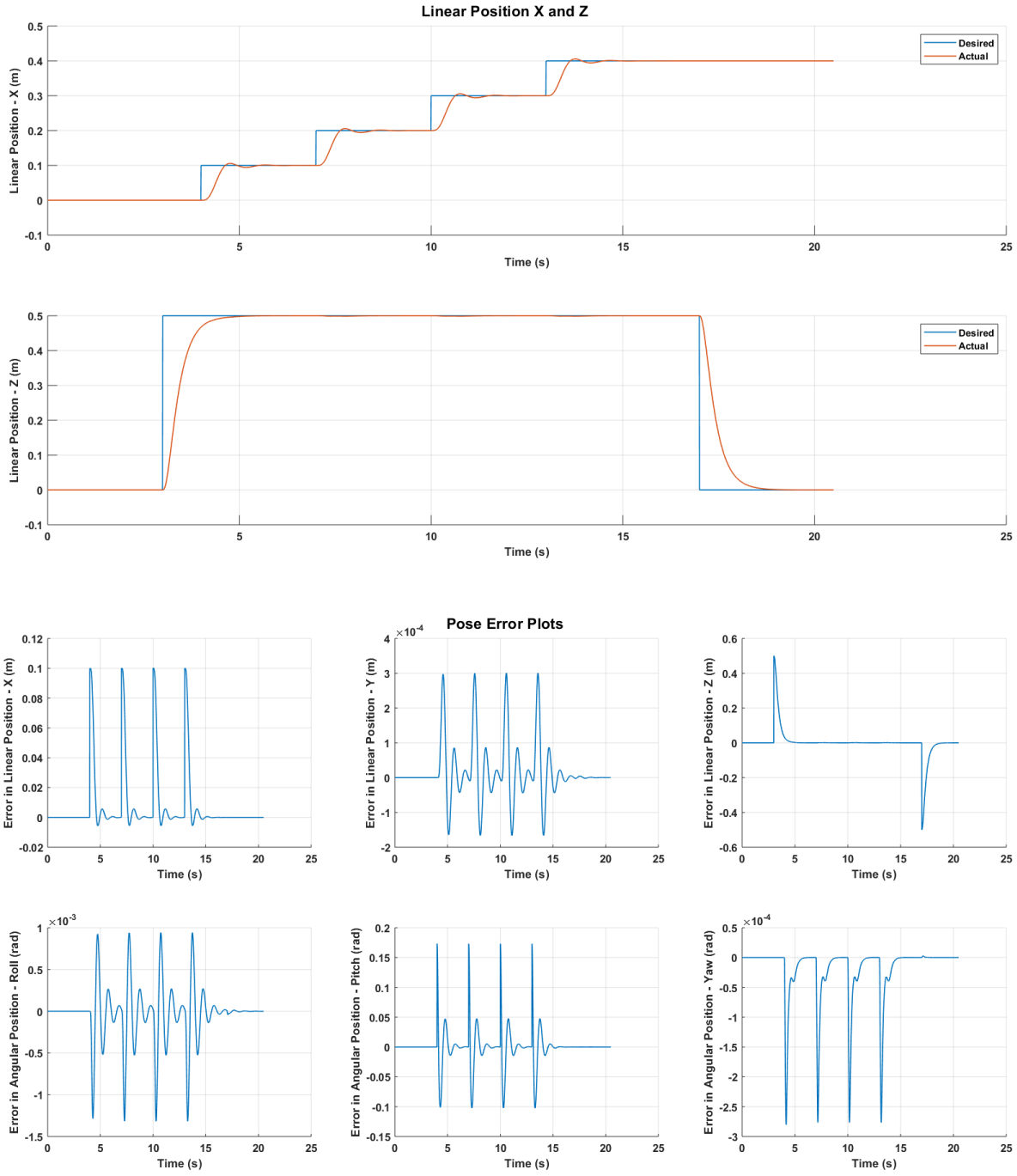


Figure 6: Position and pose error plots for PD gains set 4

Gain Set 4	***X/Roll***	***Y/Pitch***	***Z/Yaw***
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	**240.0**	80.0
Kw	30.0	30.0	17.88

By increasing the Kr for pitch in the inner loop, the oscillations about the waypoints in X axis has reduced and the response is faster compared to the previous cases. No effect on Z axis linear position.

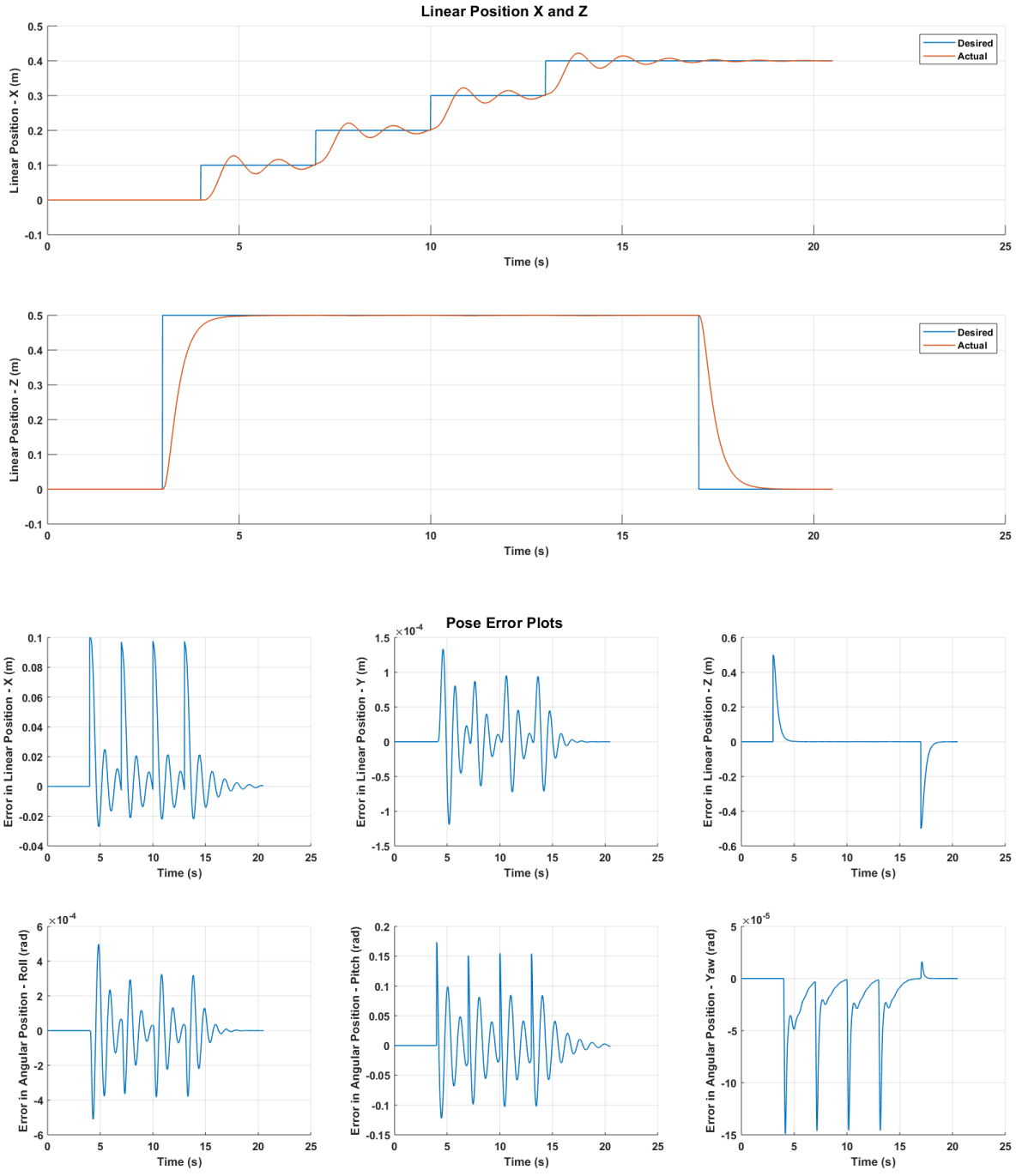


Figure 7: Position and pose error plots for PD gains set 5

Gain Set 5	***X/Roll***	***Y/Pitch***	***Z/Yaw***
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	**40.0**	17.88

By only increasing Kw for pitch, the oscillations about the waypoints in X axis significantly increased and so did the settling time and convergence about the waypoints. No effect on Z axis linear position.

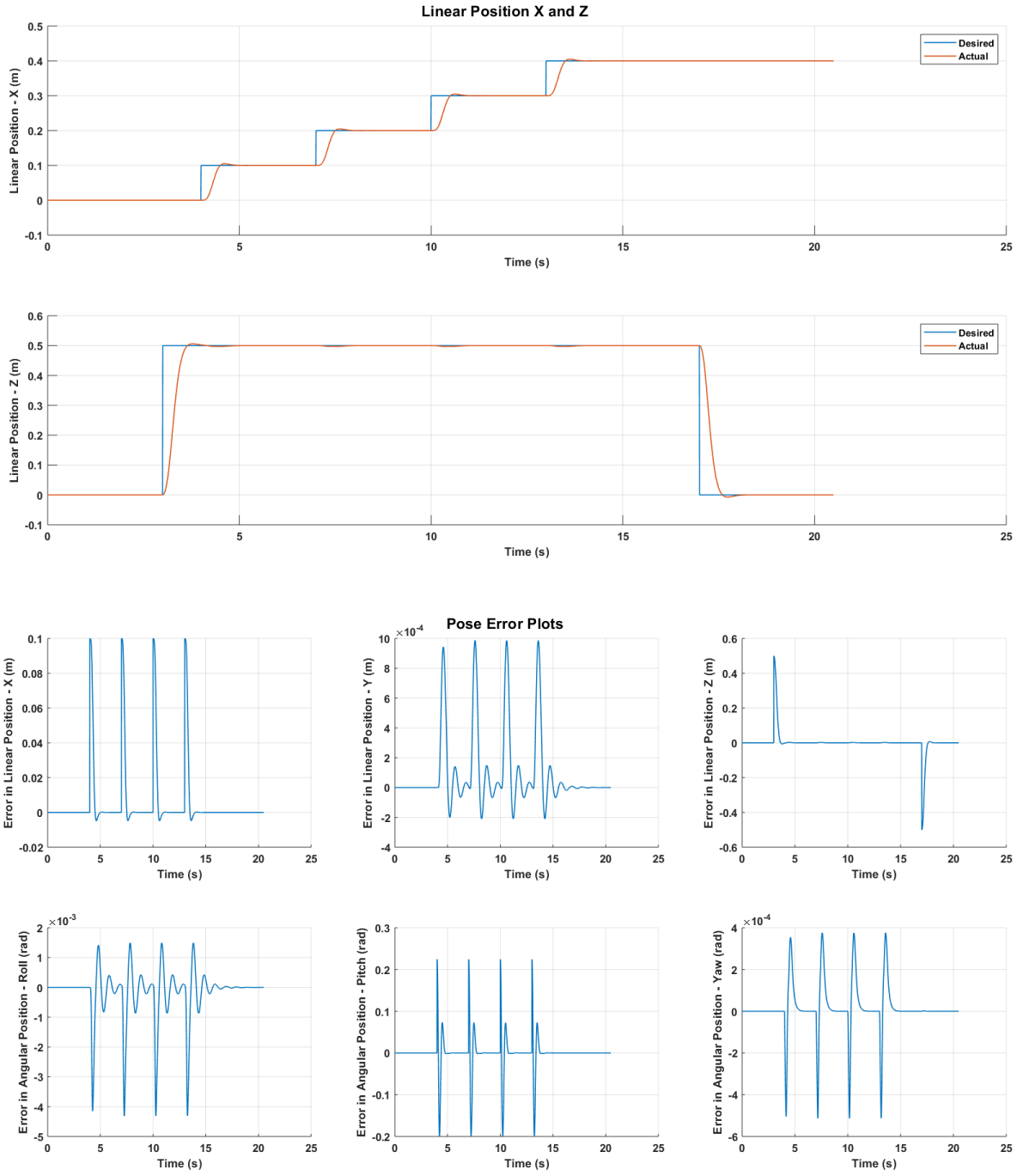
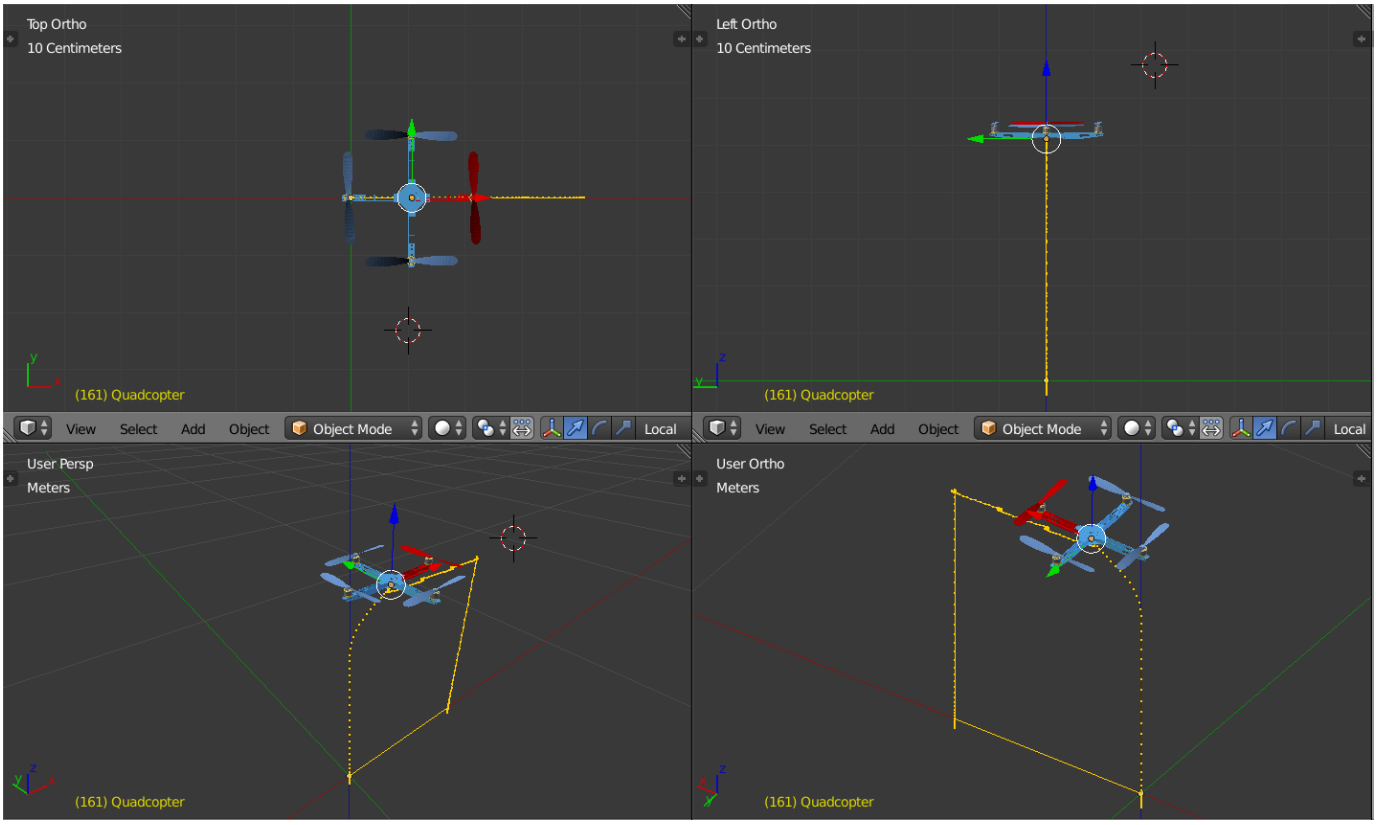


Figure 8: Position and pose error plots for PD gains set 6

Gain Set 6	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	**22.0**	17.0	**42.0**
Kd	6.6	6.6	**10.0**
Kr	190.0	**250.0**	80.0
Kw	30.0	**20.0**	17.88

Now that we know the individual effects of the inner and outer loop gains, we can use a combination of both to select a set of gains that has a quick response, with a minor overshoot and reduced oscillations for both X and Z axis waypoints. The system is now almost critically damped, and the gains selected are shown above.



Video 1: Blender simulation for X and Z waypoint tracking
(Direct link: https://drive.google.com/drive/folders/1bP0ahk_oedvzrteOZJsae2vlInPBfaSC?usp=sharing)

Problem 3 - Solution

Incremental change in waypoint is given for Z axis (1 cm increments). A ramp velocity profile is given corresponding to constant acceleration (0, ±0.25 m/s^2). The graphs are shown for linear position, velocity and acceleration in Z axis for various sets of outer loop gains.

Gain Set 1	***X/Roll***	***Y/Pitch***	***Z/Yaw***
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

Gain Set 2	***X/Roll***	***Y/Pitch***	***Z/Yaw***
Kp	17.0	17.0	20.0
Kd	6.6	6.6	**20.0**
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

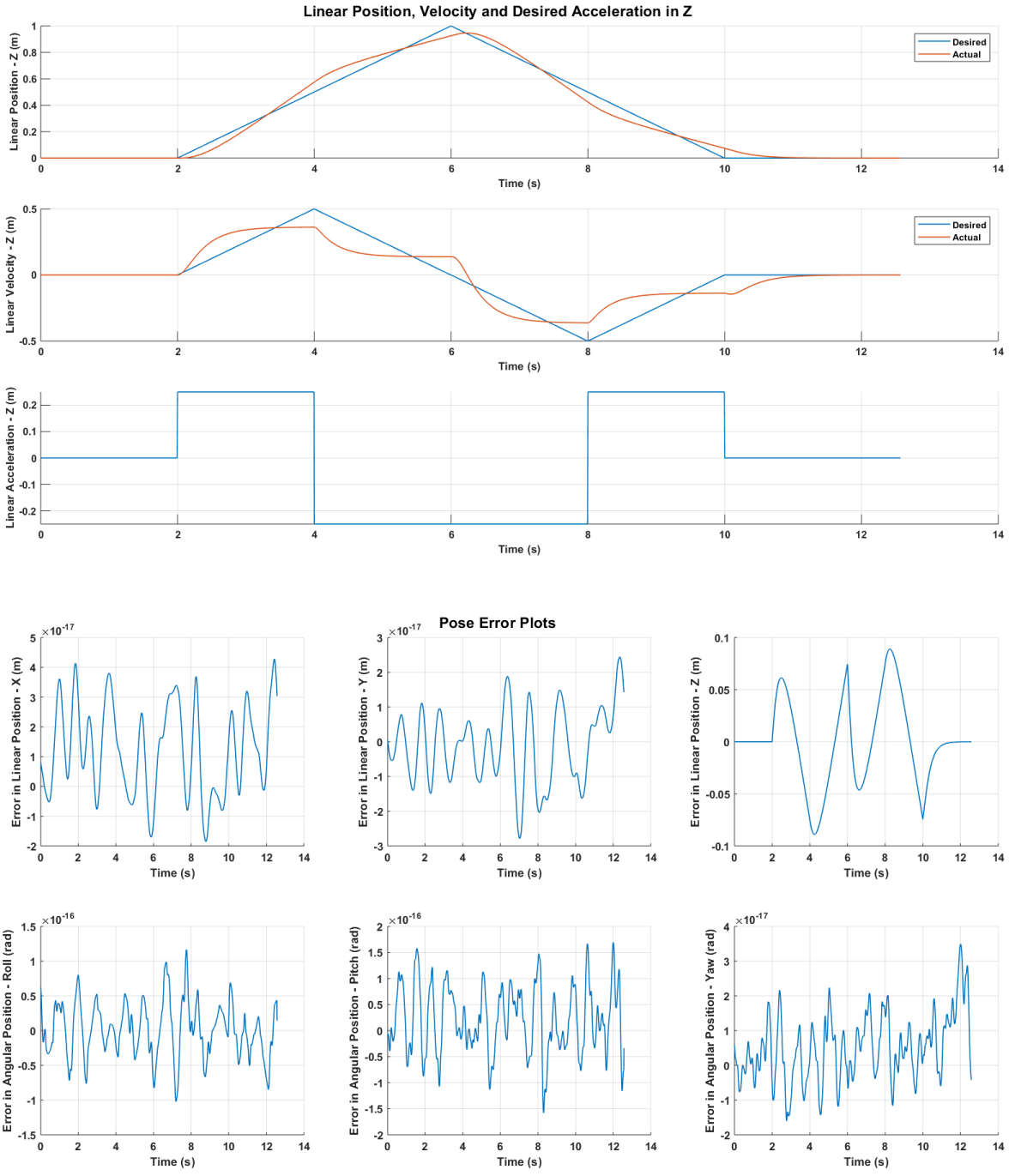


Figure 9: Z axis position, velocity, desired acceleration and pose error plots for PD gains set 1

Since the waypoints are incremental, it is not easy to analyze if there are oscillations. In general, the robot follows the trajectory in Z axis position with errors up to 10 cm. The shape of the position curve is as expected because of the constant acceleration profiles. However, the robot does not track the velocity profile well.

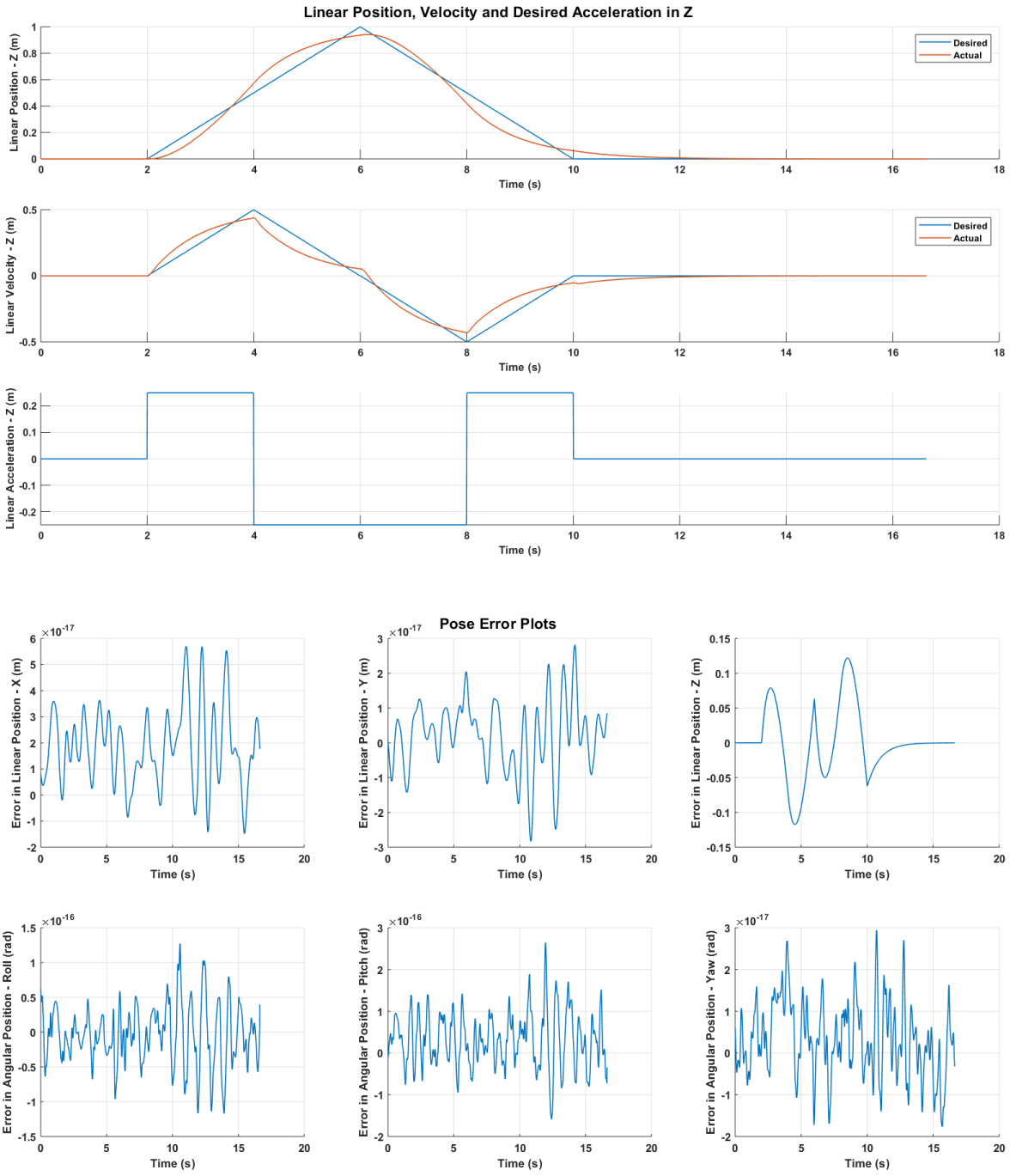


Figure 10: Z axis position, velocity, desired acceleration and pose error plots for PD gains set 2

To improve the tracking in velocity profile, we can increase the derivative gain and the results reflect the same as expected. However, the position tracking has slightly degraded now with slightly more magnitude of error and increased settling time.

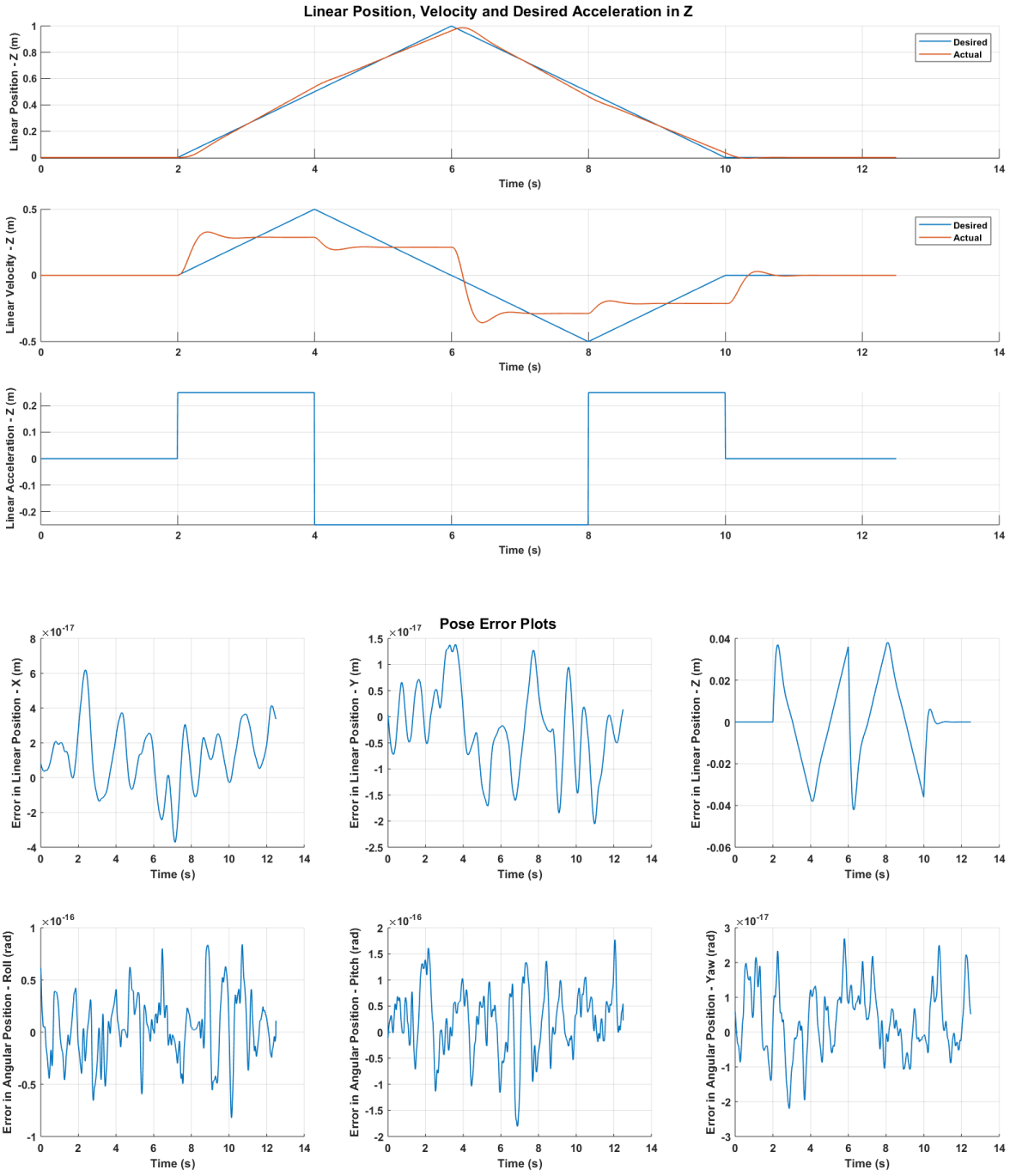


Figure 11: Z axis position, velocity, desired acceleration and pose error plots for PD gains set 3

Gain Set 3	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	**60.0**
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

On increasing the proportional gain significantly, we can see that the position tracking has greatly improved with errors less than 4 cm. We can also observe that the robot tries to fit a constant velocity profile, which is expected, given the ramp position profile.

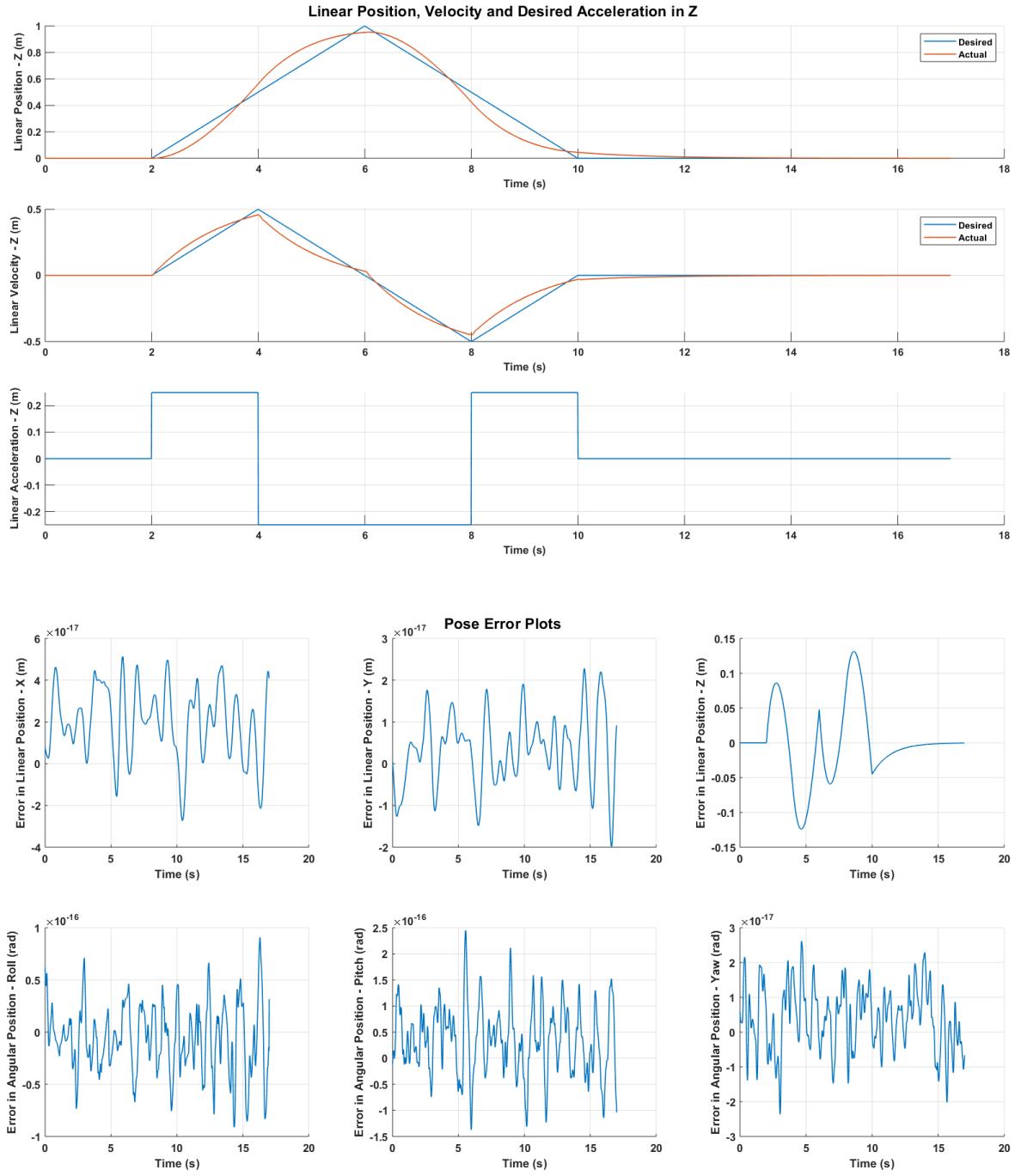


Figure 12: Z axis position, velocity, desired acceleration and pose error plots for PD gains set 4

Gain Set 4	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	**50.0**
Kd	6.6	6.6	**70.0**
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

In the final case, we have a set of gains that makes the robot follow both the position and velocity profile reasonably well. It is evident from this exercise that we need to generate some higher order optimized trajectories in order to follow both position and velocity profiles perfectly, as discussed in problem 7.

Problem 4 - Solution

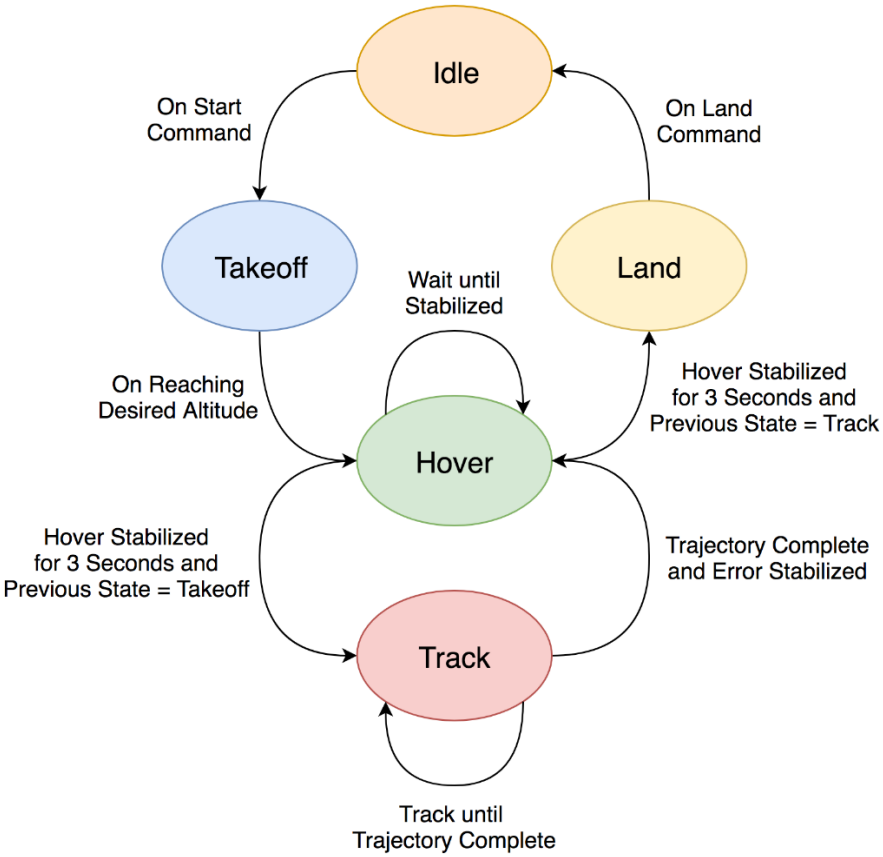


Figure 13: State machine diagram showing the various modes and transitions

A state machine with 5 states (IDLE, TAKEOFF, HOVER, TRACK and LAND) is implemented in *state_machine.m*. Initially, the robot will be in the IDLE state. All the required trajectories for all states are precomputed in *mission_planner.m* and are stored together in a MATLAB *cell* data structure.

On starting the simulation, the robot first transitions into the TAKEOFF mode and tracks the takeoff trajectory until the desired hover altitude is attained. The last desired trajectory point in this state (and the other states as well) are extended for a short duration (~0.5-5s) to ensure that the error has converged within a given tolerance (< 10e-4 – 10e-6). Once the error has converged, the state of the robot will transition automatically into a HOVER phase for a given duration (~3-5s) and will loop until the robot is stabilized at the given hover altitude.

Now after the HOVER phase, if the previous state was a TAKEOFF mode, the robot will transition into the TRACK state and will ensure that robot tracks a given trajectory until its complete. As mentioned earlier, the last desired point is extended slightly to ensure the error settles within a tolerance. Once the error is stabilized, the robot will transition back into HOVER mode which works in the same manner as described previously. Finally, when the HOVER mode is stable for a given duration and is the previous state was TRACK, the robot will transition into LAND mode which will return the quadcopter to the ground.

The total simulation time will be dynamic and depend on error convergence at every state. Please note that the implementation of *state_machine.m* and the one discussed above might vary as it was improved later for Problem 10. The changes are discussed later in the same Problem along with a more detailed description.

Problem 5 - Solution

A step waypoint of (0, 0, 0.1, 0) at zero velocity was given and the corresponding responses and error plots were obtained. Time domain performance characteristics for different sets of gains were obtained as well.

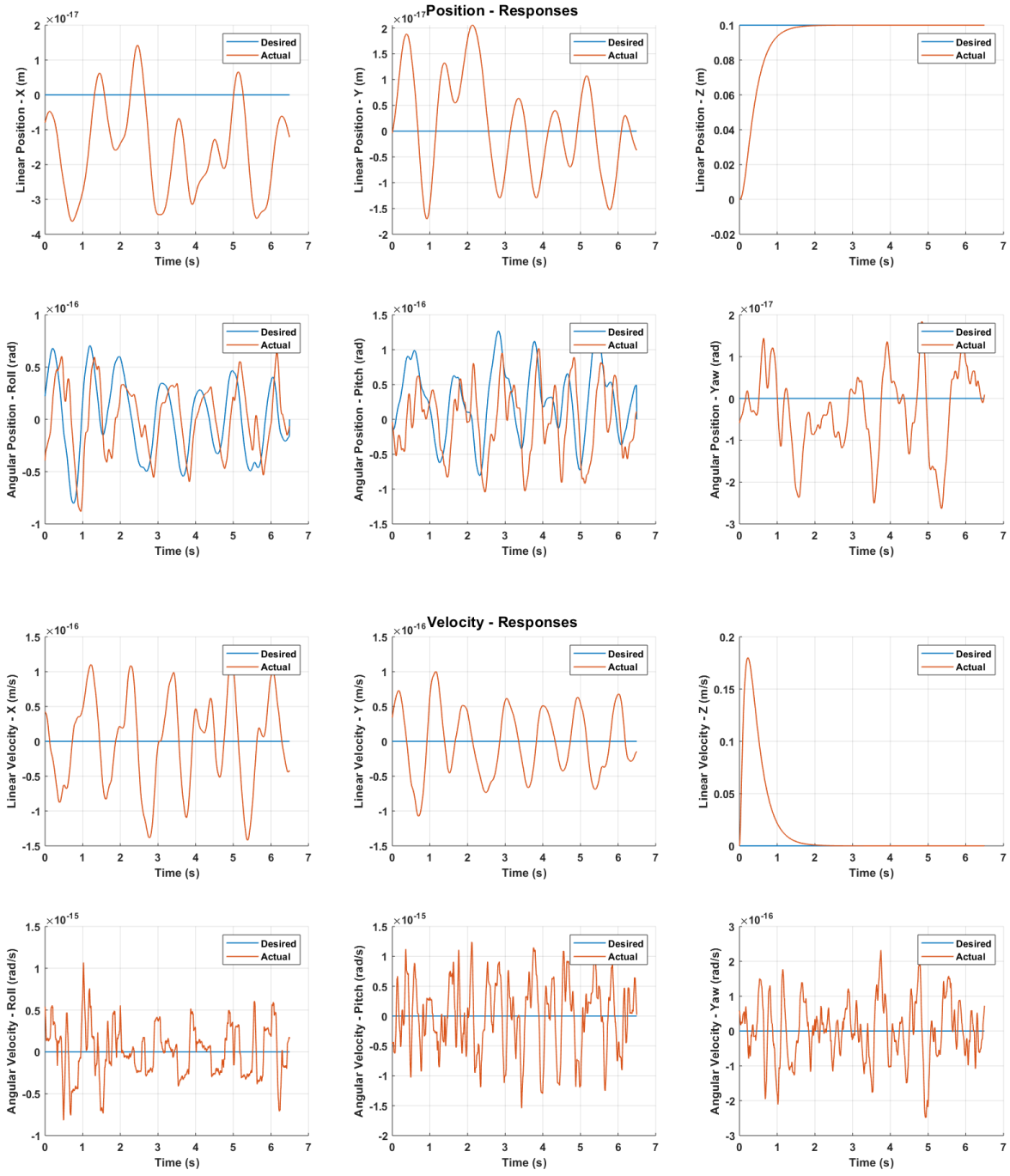


Figure 14: Linear and angular, position and velocity step responses for gains set 1

Gain Set 1	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

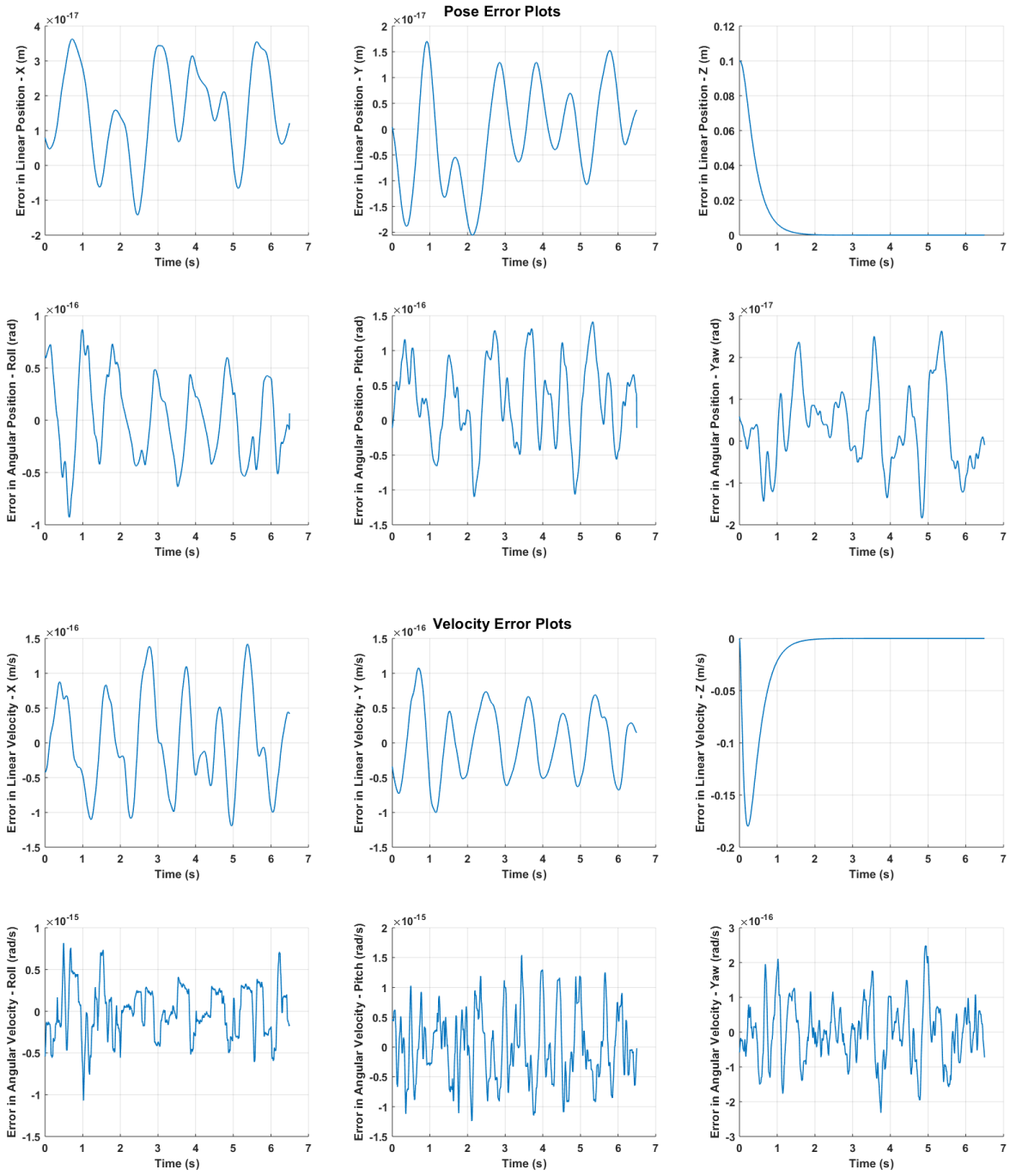


Figure 15: Error in responses for gains set 1

Position (Z) Characteristics	***Velocity (Z) Characteristics***
Rise Time: 0.7396	Rise Time: 0.6635
Settling Time: 0.8700	Settling Time: 1.0568
Settling Min: 0.0900	Settling Min: 2.7232e-10
Settling Max: 0.1000	Settling Max: 1.2048e-06
Overshoot: 0	Overshoot: Inf
Undershoot: 0.0036	Undershoot: 0
Peak: 0.1000	Peak: 0.1799
Peak Time: 6.4950	Peak Time: 0.2250
Steady-State Value: 0.1000	Steady-State Value: 8.155e-11

NOTE: Time domain performance characteristics are reported only for Z axis position and velocity as it is not relevant to report these for other axes which have very low magnitudes of errors. The data is directly reported with the help of MATLAB's *stepinfo* function.

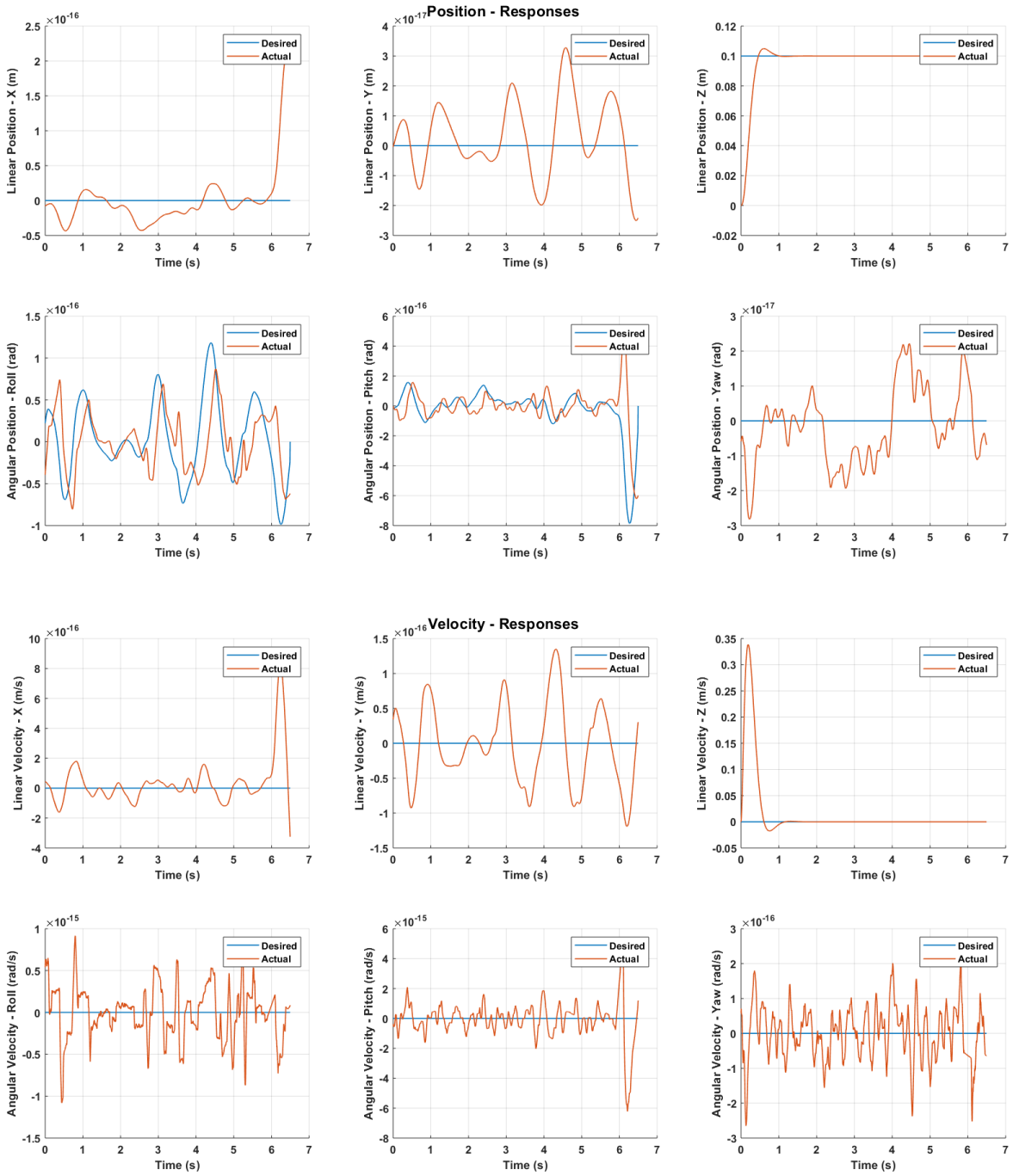


Figure 16: Linear and angular, position and velocity step responses for gains set 2

Gain Set 2	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	**42.0**
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

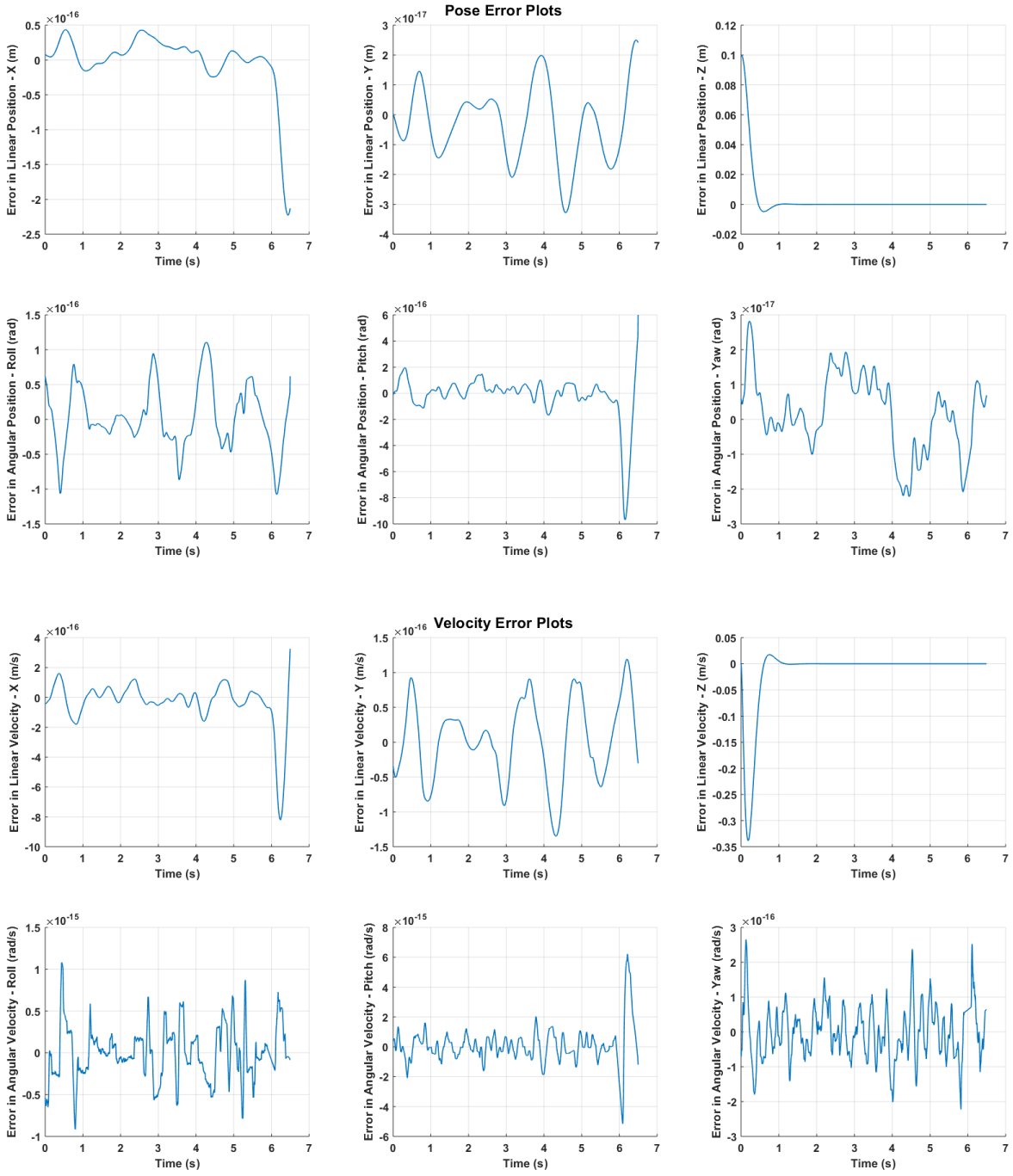


Figure 17: Error in responses for gains set 2

Position (Z) Characteristics	***Velocity (Z) Characteristics***
Rise Time: 0.2851	Rise Time: 3.3555e-05
Settling Time: 0.3769	Settling Time: 0.5283
Settling Min: 0.0905	Settling Min: -0.0175
Settling Max: 0.1049	Settling Max: 9.1342e-04
Overshoot: 4.8972	Overshoot: Inf
Undershoot: 0.0036	Undershoot: Inf
Peak: 0.1049	Peak: 0.3385
Peak Time: 0.6050	Peak Time: 0.1900
Steady-State Value: 0.1000	Steady-State Value: 4.093e-10

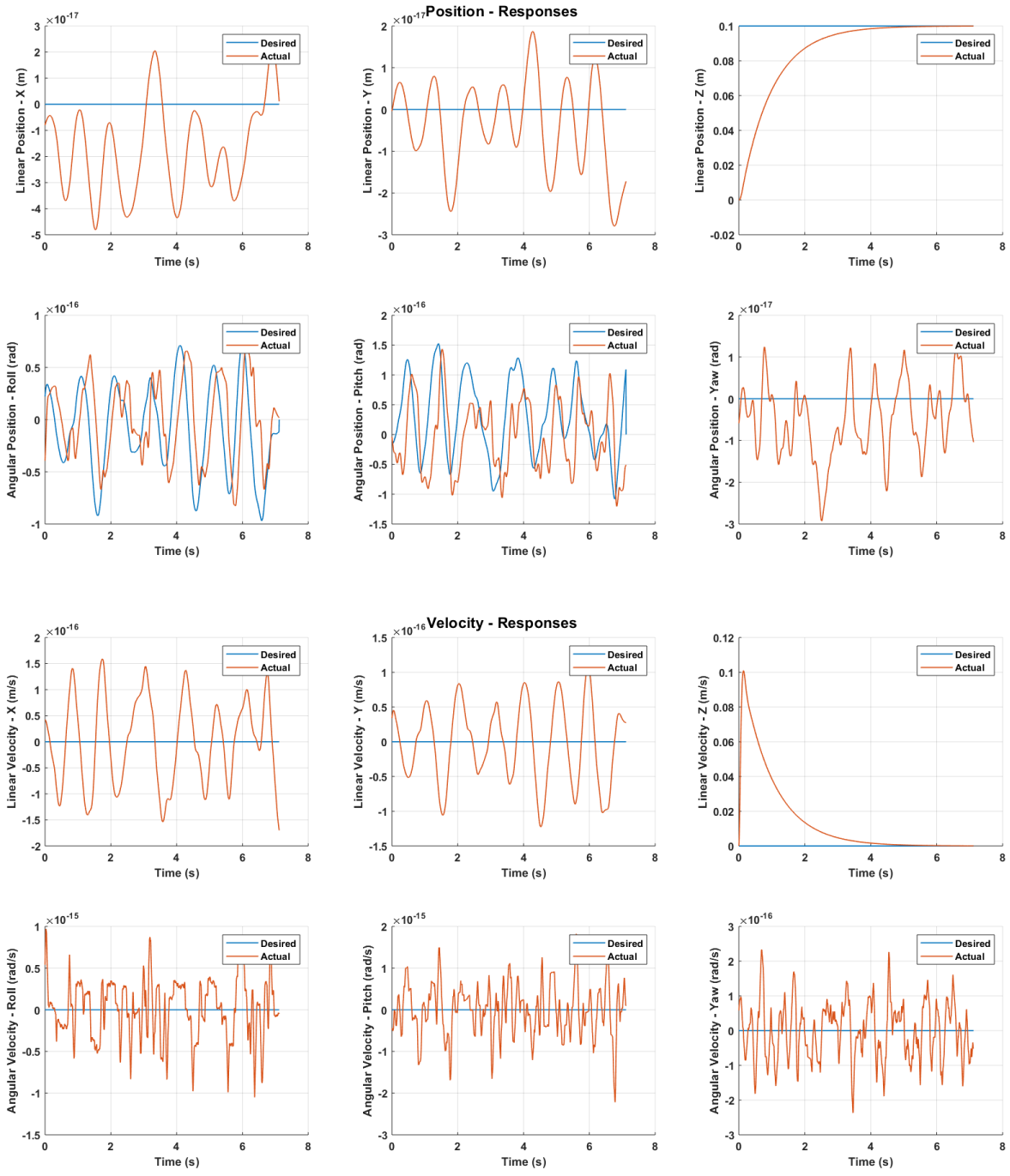


Figure 18: Linear and angular, position and velocity step responses for gains set 3

Gain Set 3	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	20.0
Kd	6.6	6.6	**20.0**
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

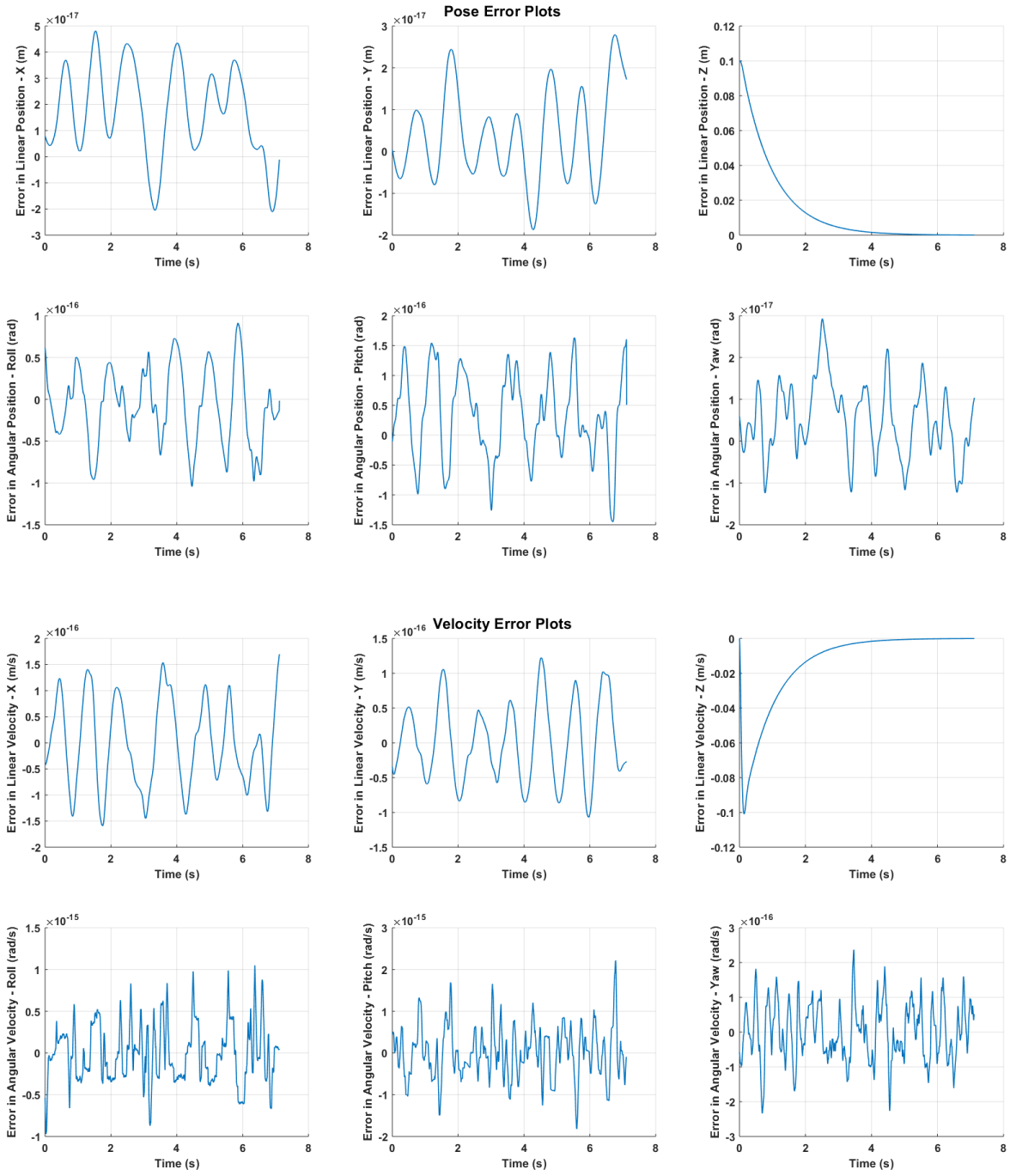


Figure 19: Error in responses for gains set 3

Position (Z) Characteristics	***Velocity (Z) Characteristics***
Rise Time: 2.0869	Rise Time: NaN
Settling Time: 2.2374	Settling Time: 2.2785
Settling Min: 0.0900	Settling Min: NaN
Settling Max: 0.0999	Settling Max: NaN
Overshoot: 0	Overshoot: Inf
Undershoot: 0.0036	Undershoot: 0
Peak: 0.0999	Peak: 0.1009
Peak Time: 7.1150	Peak Time: 0.1400
Steady-State Value: 0.099994	Steady-State Value: 6.257e-05

For Waypoint (0, 0, 0.1, rad(15°))

All angular data is reported in radians.

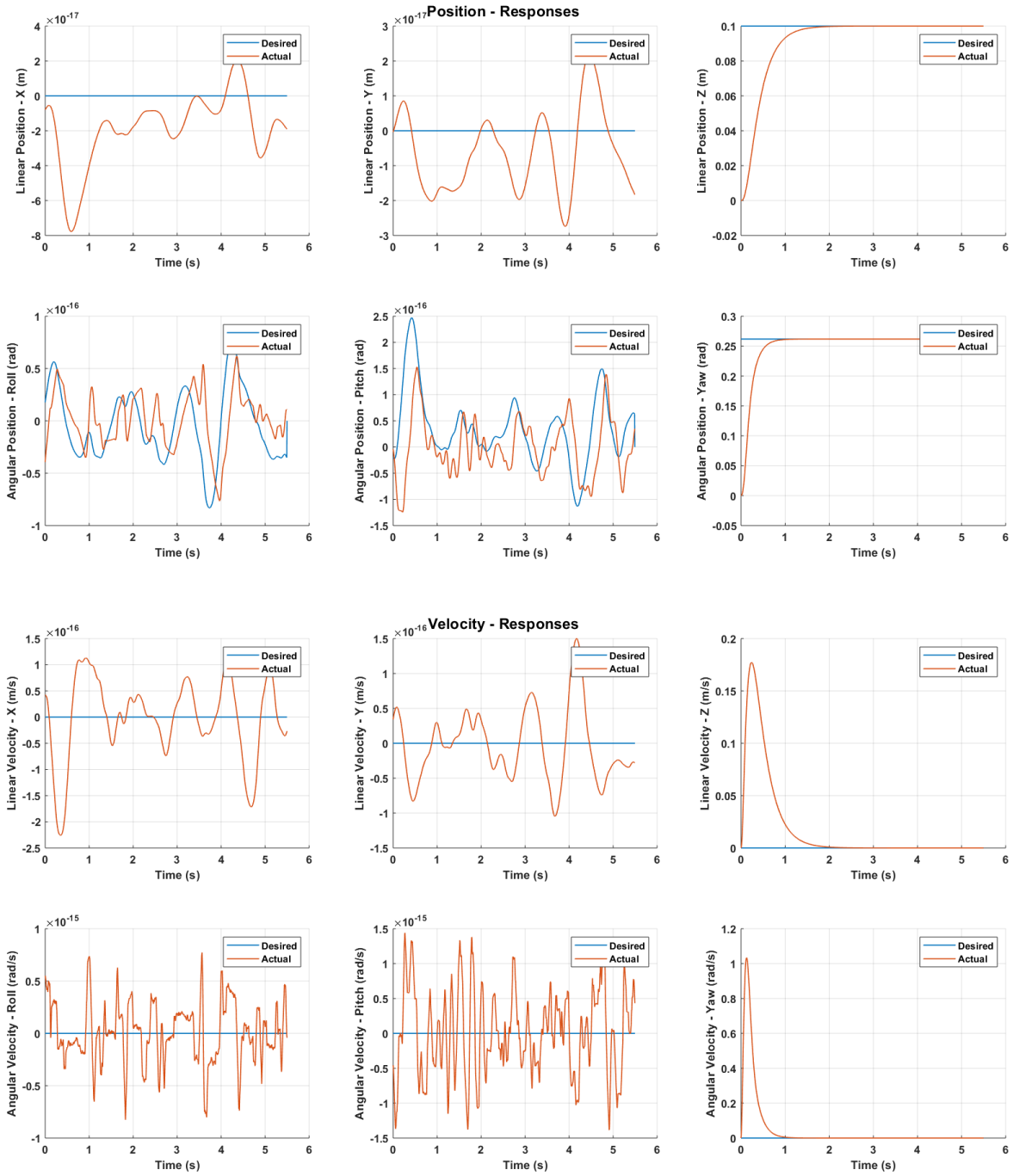


Figure 20: Linear and angular, position and velocity step responses for gains set 4

Gain Set 4	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	17.88

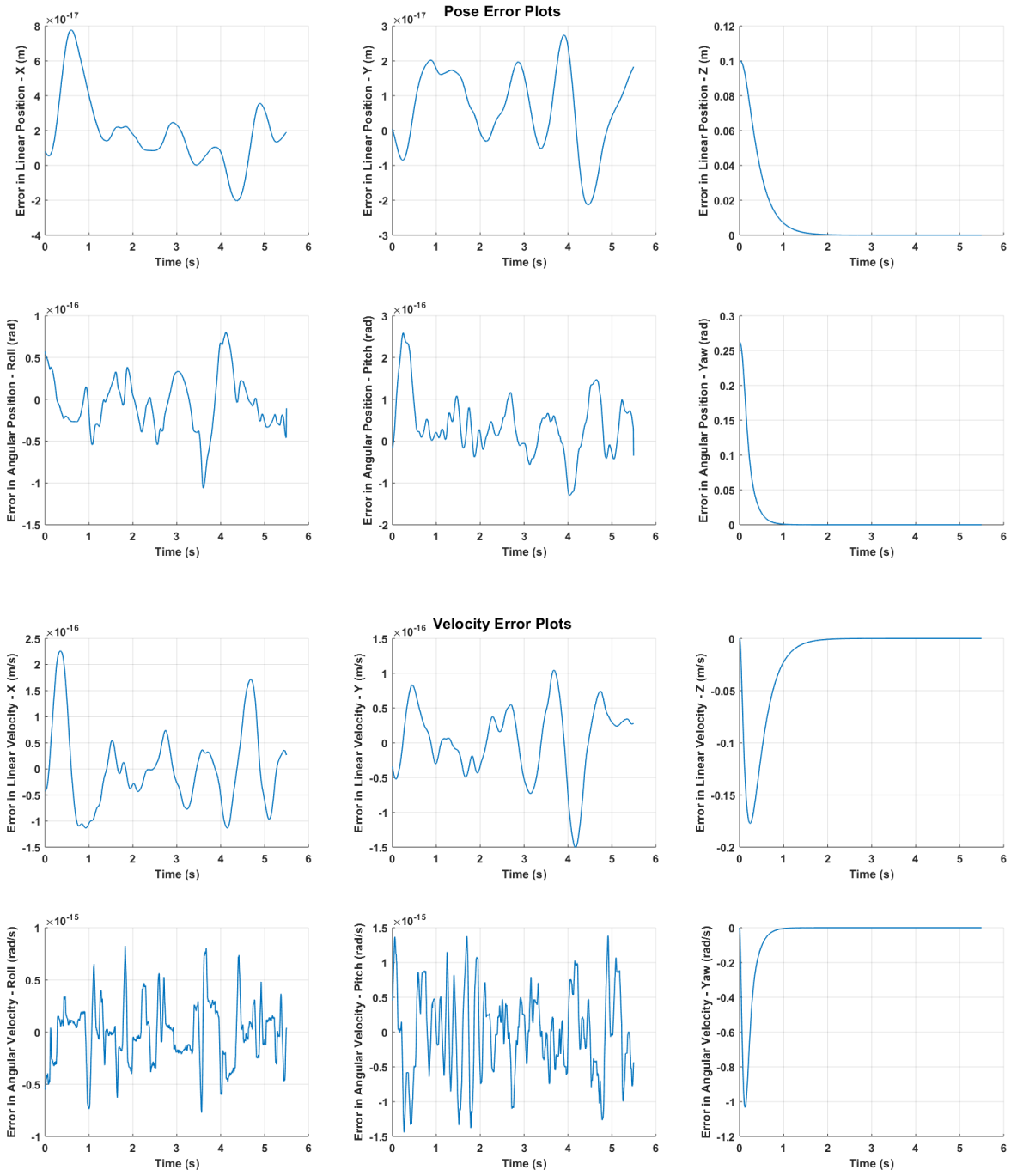


Figure 21: Error in responses for gains set 4

Position (Yaw) Characteristics	*Velocity (Yaw) Characteristics*
Rise Time: 0.3585	Rise Time: NaN
Settling Time: 0.4288	Settling Time: 0.5037
Settling Min: 0.2358	Settling Min: NaN
Settling Max: 0.2618	Settling Max: NaN
Overshoot: 0	Overshoot: Inf
Undershoot: 2.2467e-15	Undershoot: 0
Peak: 0.2618	Peak: 1.0335
Peak Time: 5.0700	Peak Time: 0.1250
Steady-State Value: 0.2618	Steady-State Value: 6.883e-15

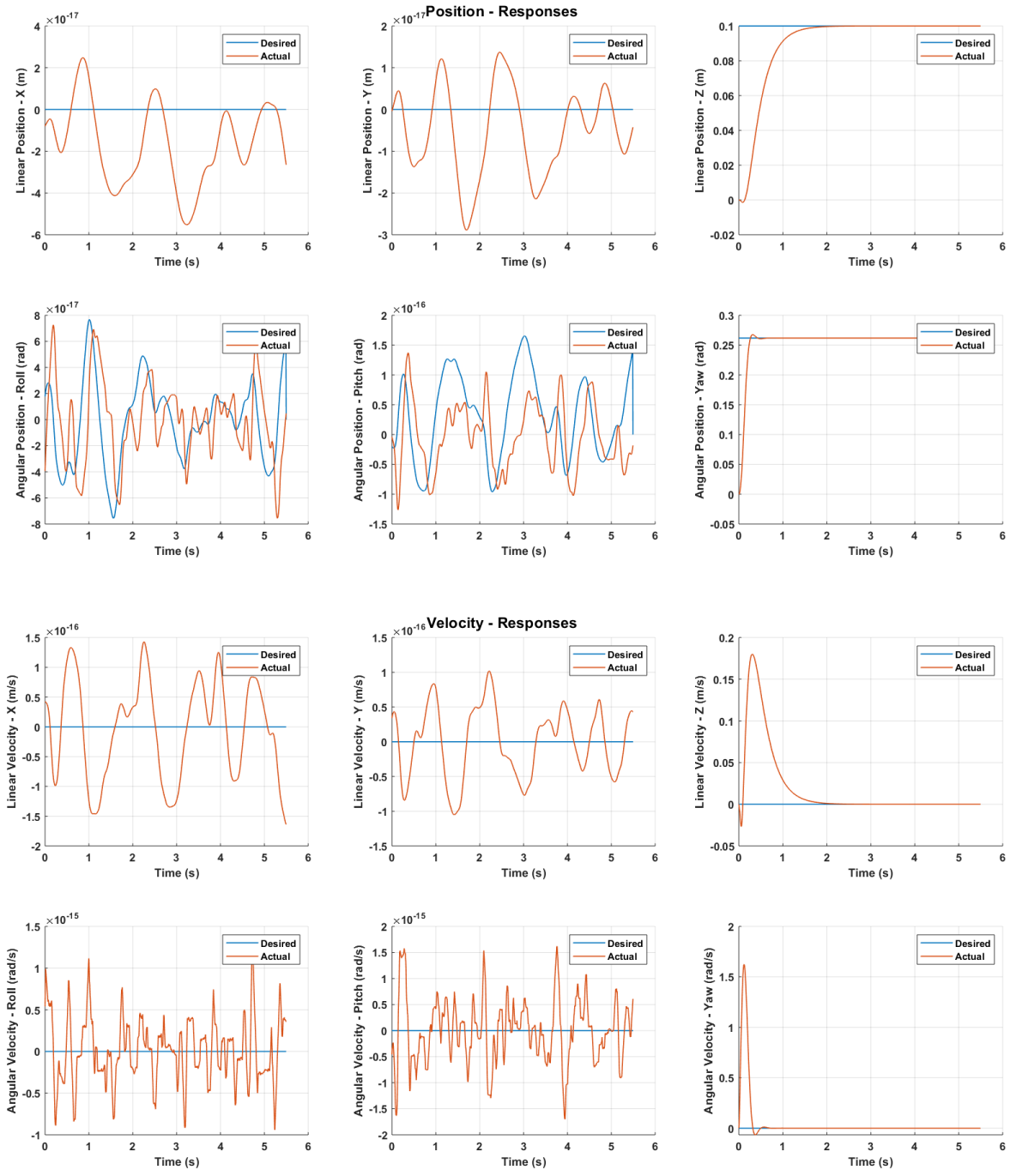


Figure 22: Linear and angular, position and velocity step responses for gains set 5

Gain Set 5	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	**140.0**
Kw	30.0	30.0	17.88

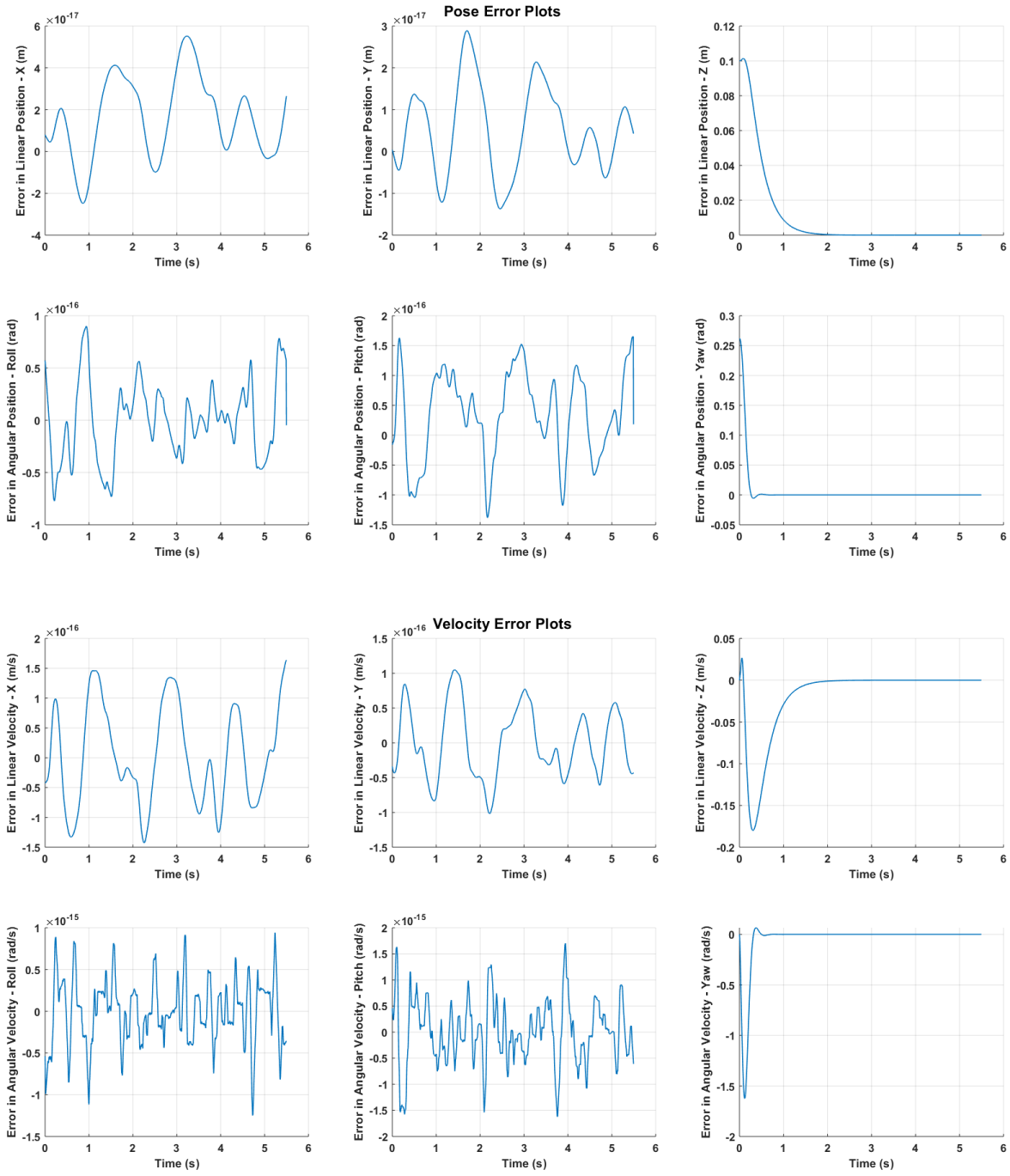


Figure 23: Error in responses for gains set 5

Position (Yaw) Characteristics	*Velocity (Yaw) Characteristics*
Rise Time: 0.1566	Rise Time: 0
Settling Time: 0.2117	Settling Time: 0.2812
Settling Min: 0.2382	Settling Min: -0.0659
Settling Max: 0.2675	Settling Max: 0.0106
Overshoot: 2.1726	Overshoot: Inf
Undershoot: 2.2467e-15	Undershoot: Inf
Peak: 0.2675	Peak: 1.6244
Peak Time: 0.3150	Peak Time: 0.1200
Steady-State Value: 0.2618	Steady-State Value: 4.871e-09

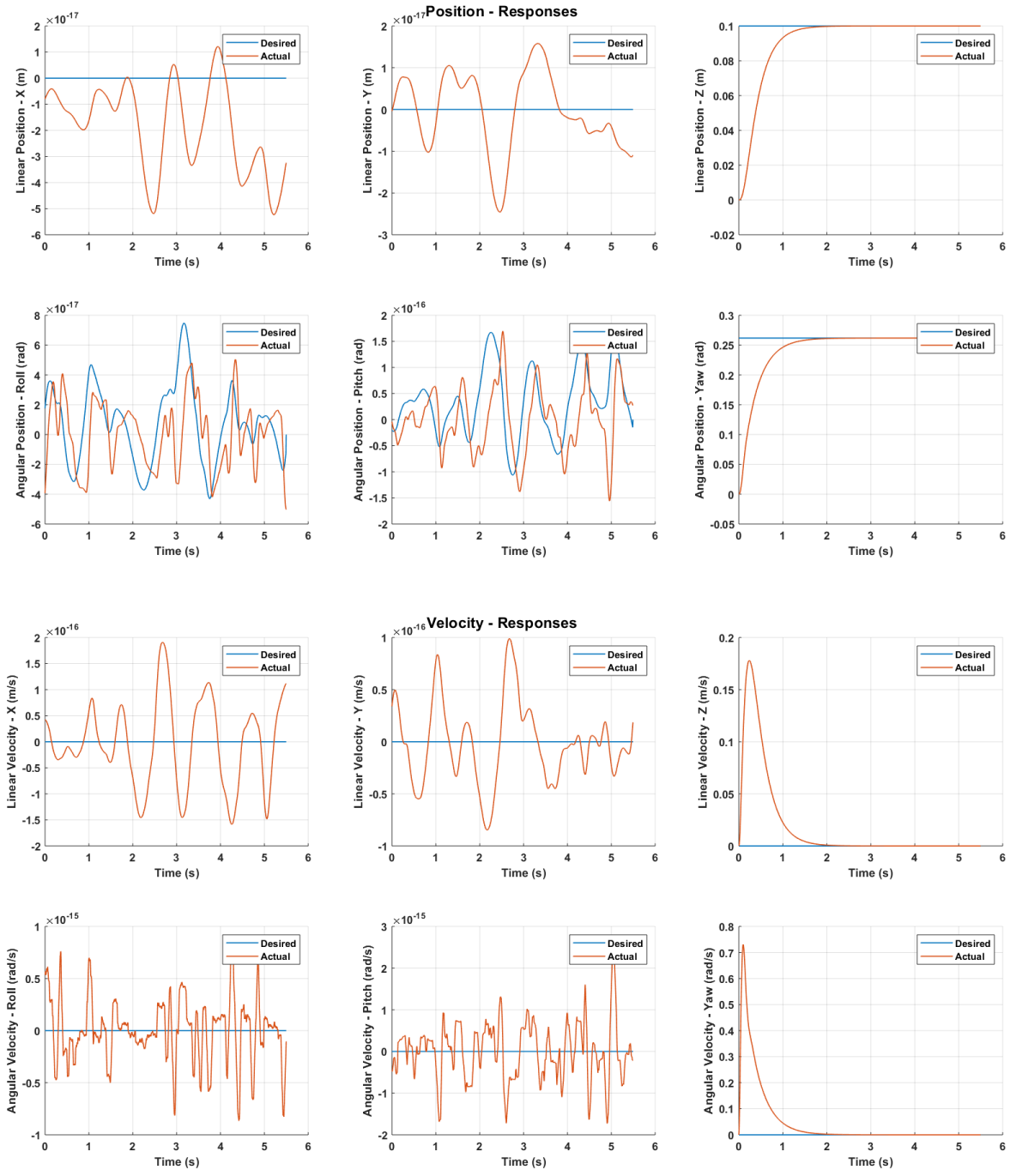


Figure 24: Linear and angular, position and velocity step responses for gains set 6

Gain Set 6	*X/Roll*	*Y/Pitch*	*Z/Yaw*
Kp	17.0	17.0	20.0
Kd	6.6	6.6	9.0
Kr	190.0	182.0	80.0
Kw	30.0	30.0	**30.0**

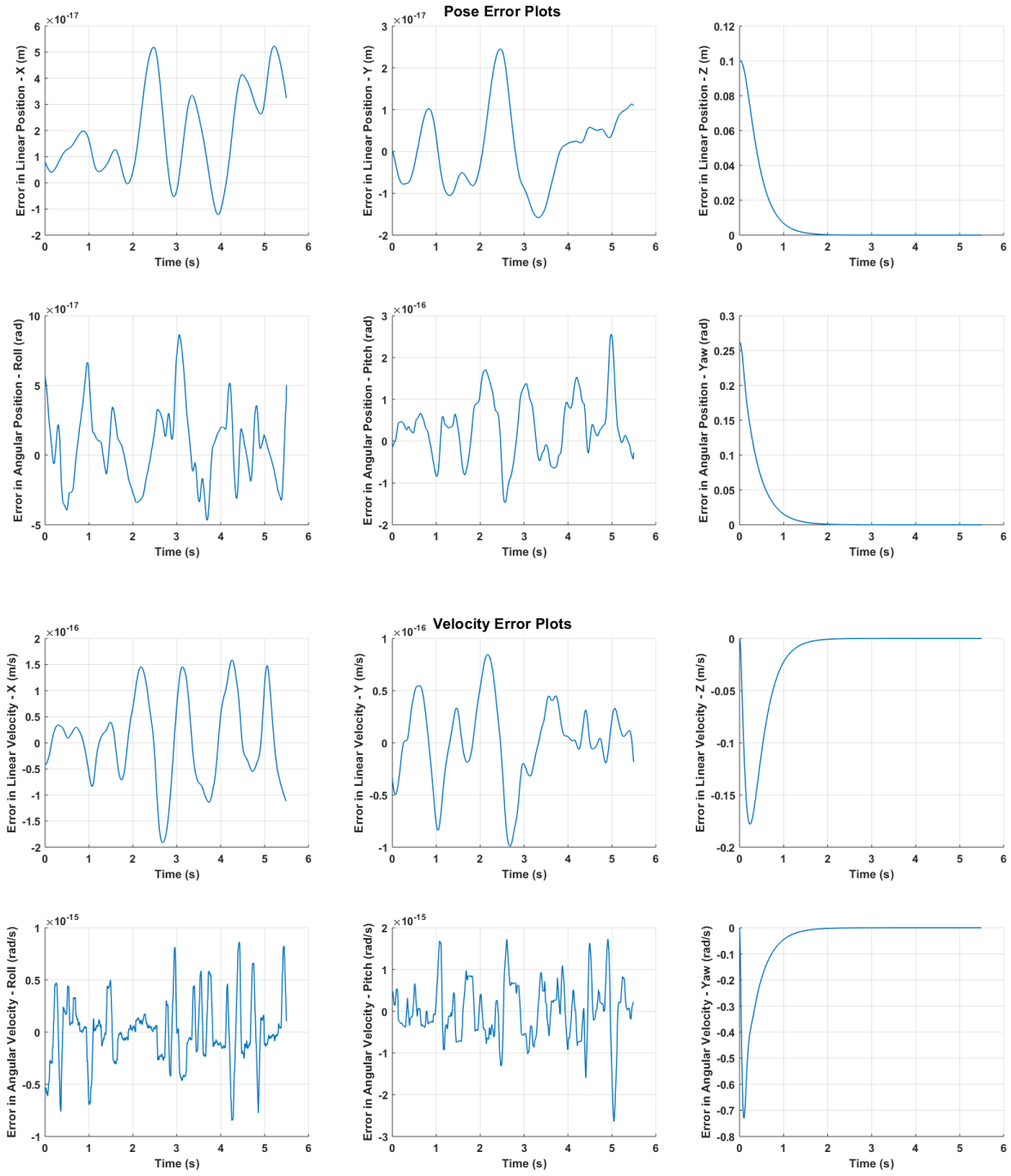


Figure 25: Error in responses for gains set 6

Position (Yaw) Characteristics	***Velocity (Yaw) Characteristics***
RiseTime: 0.7456	RiseTime: NaN
SettlingTime: 0.8195	SettlingTime: 0.8361
SettlingMin: 0.2357	SettlingMin: NaN
SettlingMax: 0.2618	SettlingMax: NaN
Overshoot: 0	Overshoot: Inf
Undershoot: 2.2467e-15	Undershoot: 0
Peak: 0.2618	Peak: 0.7305
PeakTime: 5.4950	PeakTime: 0.1000
Steady-State Value: 0.2618	Steady-State Value: 2.659e-06

Remarks

From the above experiment, it was clear that on increasing the proportional gain (for both linear and angular, position and velocity), the rise time and settling time reduced with a quick response for the system, however, the overshoot percentage increased as well. On increasing the derivative gain, the overshoot reduced, however, the response of the system was very slow with increased rise and settling times.

The overshoot percentage for velocity in several cases is *Inf*, this is because the desired velocity is zero for all cases and any overshoot computation would cause division by zero. Please refer to the corresponding Peak Time for these cases. The Rise Time for velocity is certain cases is *NaN*, this is because, the initial velocity itself is zero in all cases and Rise Time can be zero or a very low value for all such cases.

Problem 6 - Solution

The state space model was implemented and LQR control policy was implemented. Please note that, all the plots below are part of the tracking mode of the state machine developed. The quadcopter initially dips in Z-axis for about 0.2m which is not visible in the plots below as it is part of the idle state and it was not feasible to restructure the code to plot the same. A waypoint of (0, 0, 0.5, 0) is given at *t=3s* to make the robot hover. Increments of 10 cm waypoints in the x direction are given at *t=4,7,10,13s*.

Hover Performance

For LQR, we need the Q and R matrices which are the weights on the states and the control input respectively. Following are the set of gains that we'll be using to analyse the LQR responses. The values are the diagonal elements corresponding to the following variables.

Q = [X, Y, Z, R, P, Yaw, dX, dY, dZ, dR, dP, dYaw]
R = [X, Y, Z, Yaw]

Gains Set 1
Q = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
R = [2, 2, 2, 2]

Gains Set 2
*Q = [**20**, 0.1, **20**, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]*
R = [2, 2, 2, 2]

Gains Set 3
*Q = [**20**, 0.1, **20**, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]*
*R = [**0.5**, 2, **0.5**, 2]*

In general the LQR controller performed better for larger step responses in X, Y and Z copmared to the PD controller. The response was quick with very less oscillations. Howevere, it did not perform very well with tracking velocity profiles. When a waypoint with Yaw and Z position was given, the LQR controller for Yaw performed better than the PD controller, however, the settling time for Z position increased.

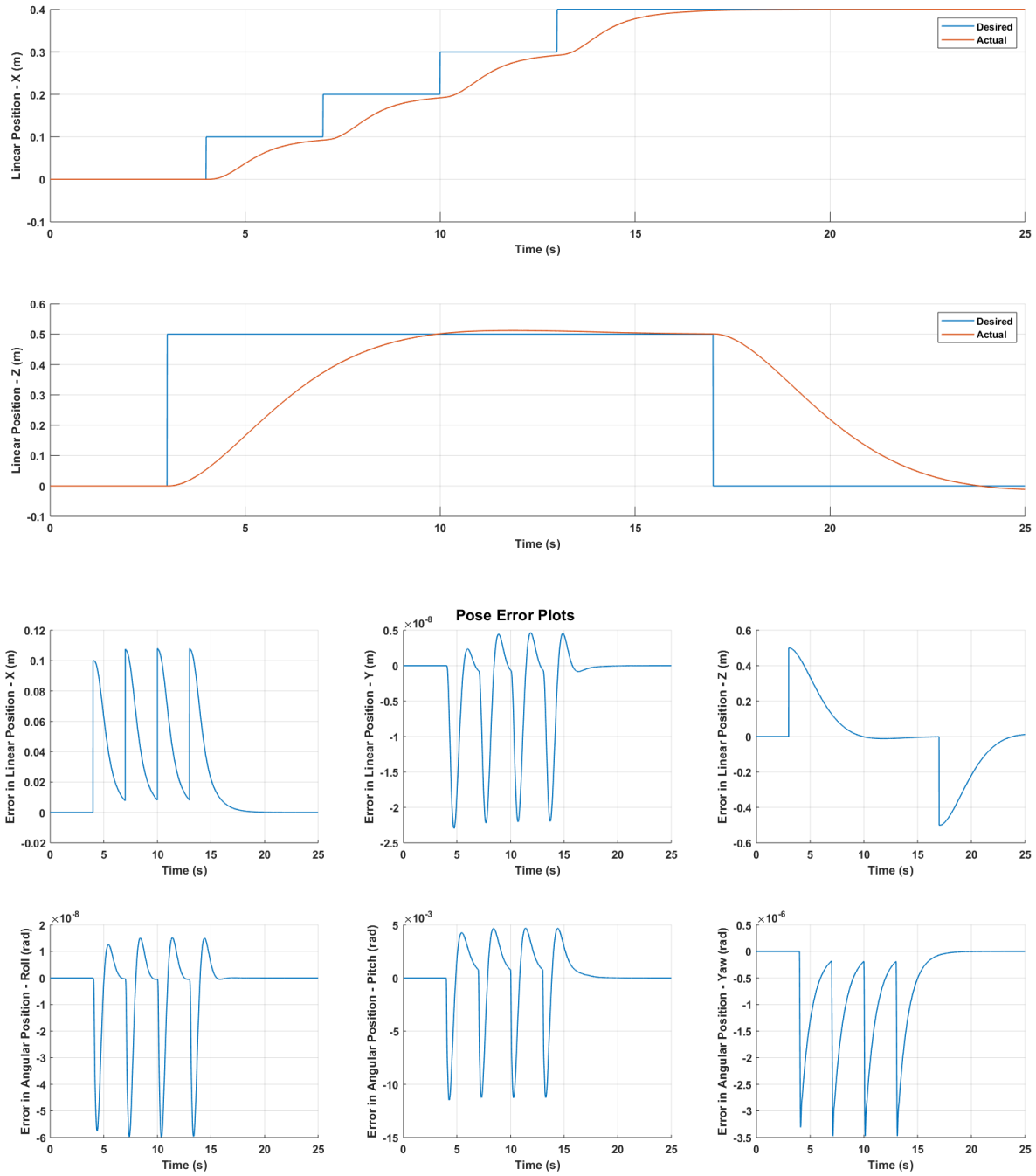


Figure 26: Position and pose error plots for LQR gains set 1

The robot tracks the trajectory in Z axis very slowly and it is over-damped, and along the X axis waypoints, it barely tracks the trajectory with the default LQR gains.

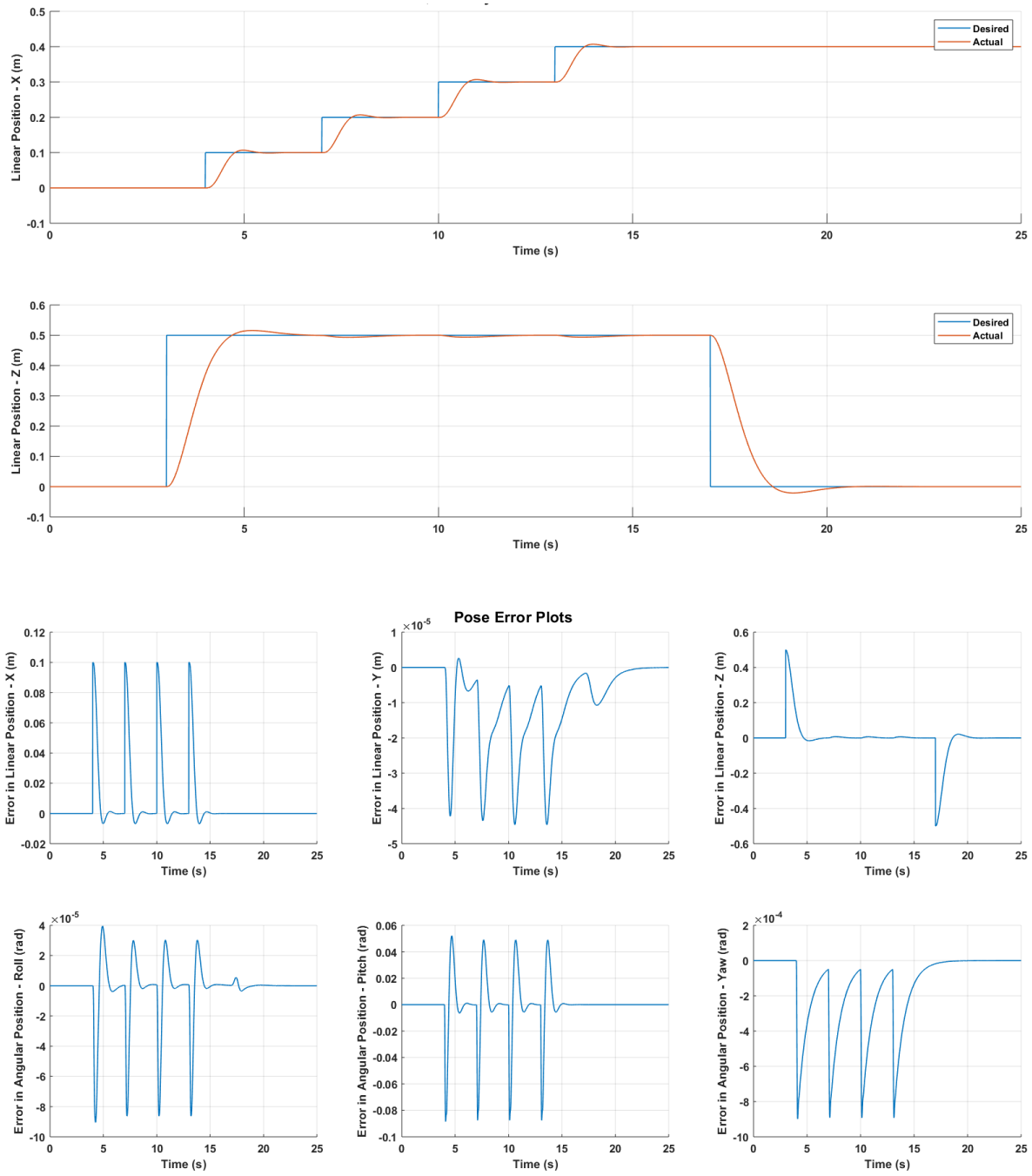


Figure 27: Position and pose error plots for LQR gains set 2

On increasing the Q weights for X and Z position, the robot now tracks the trajectory quickly with a slight overshoot and very less oscillations.

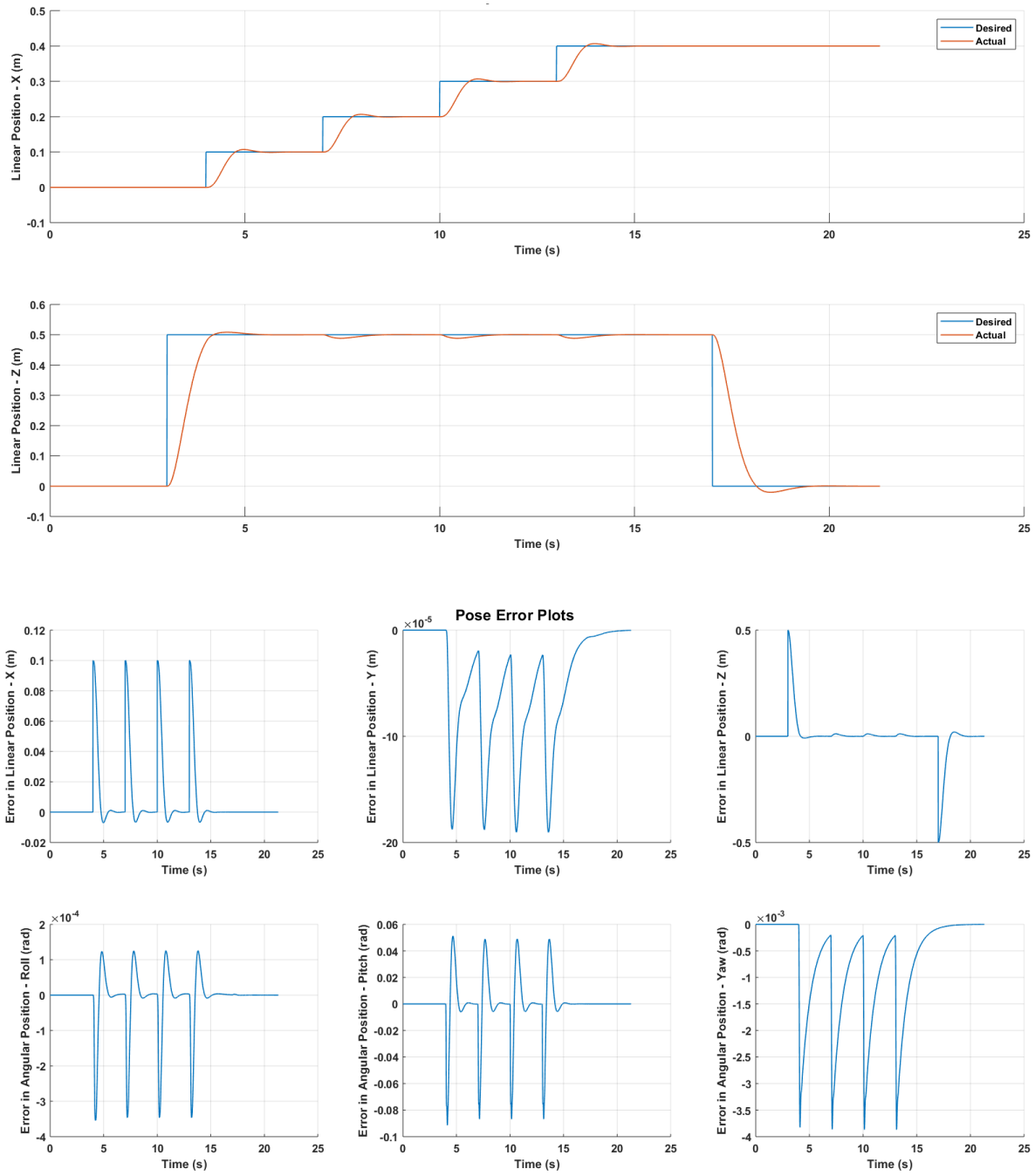


Figure 28: Position and pose error plots for LQR gains set 3

On reducing the R weights for X and Z position, there is not a lot of difference in the response and the error plots. Some minor perturbances can be seen during the hover. There is no improvement while changing the weights for pitch, X and Z velocities as their desired values are zero and it only degrades the performance.

Line-Tracking Performance

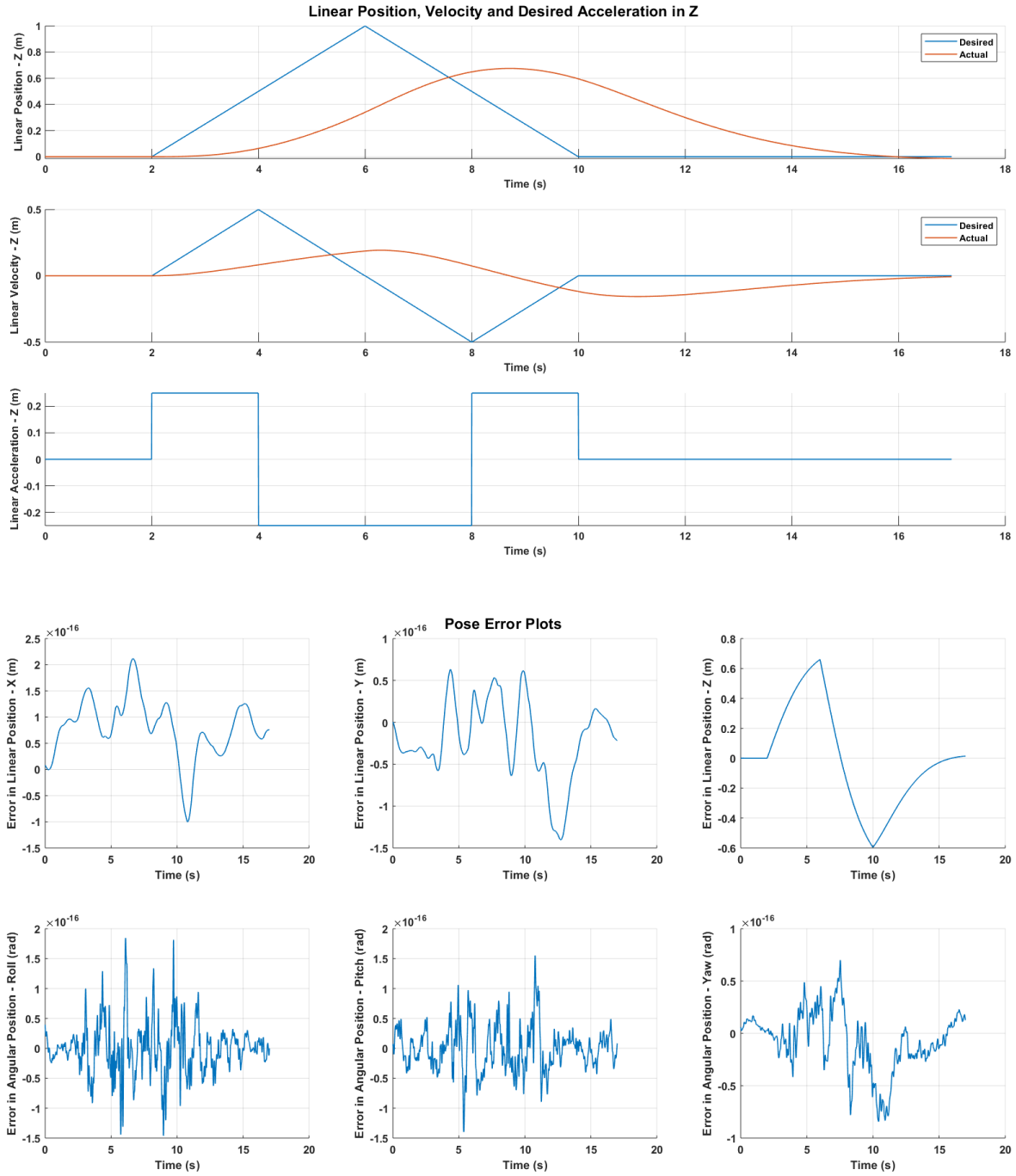


Figure 29: Z axis position, velocity, desired acceleration and pose error plots for LQR gains set 1

To tracking performance for both position and velocity is very poor with the default set of gains.

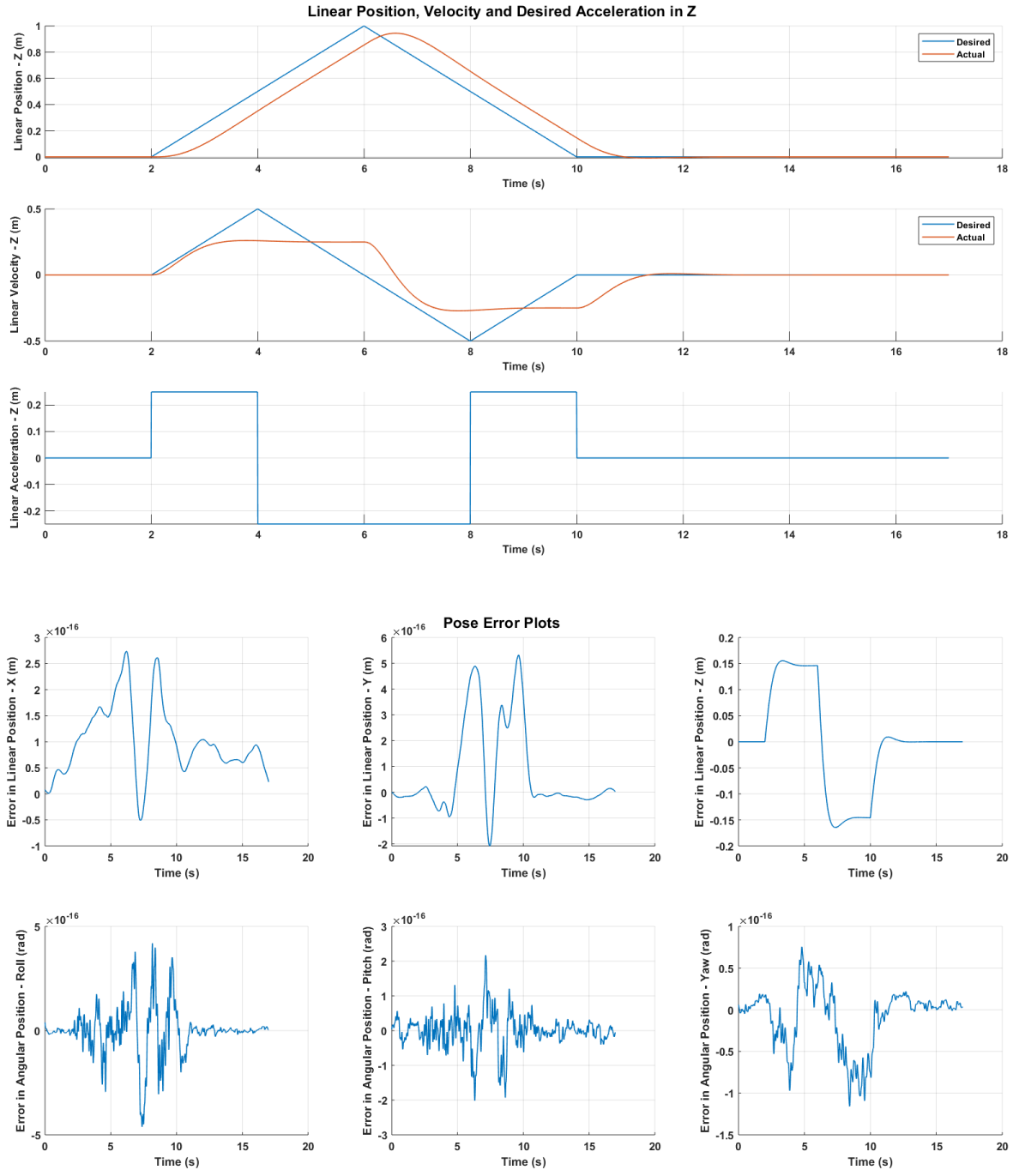


Figure 30: Z axis position, velocity, desired acceleration and pose error plots for LQR gains *Q = [0.1, 0.1, 40, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]* and *R = [2, 2, 2, 2]*

To tracking performance for position has improved slightly but the velocity tracking is very poor.

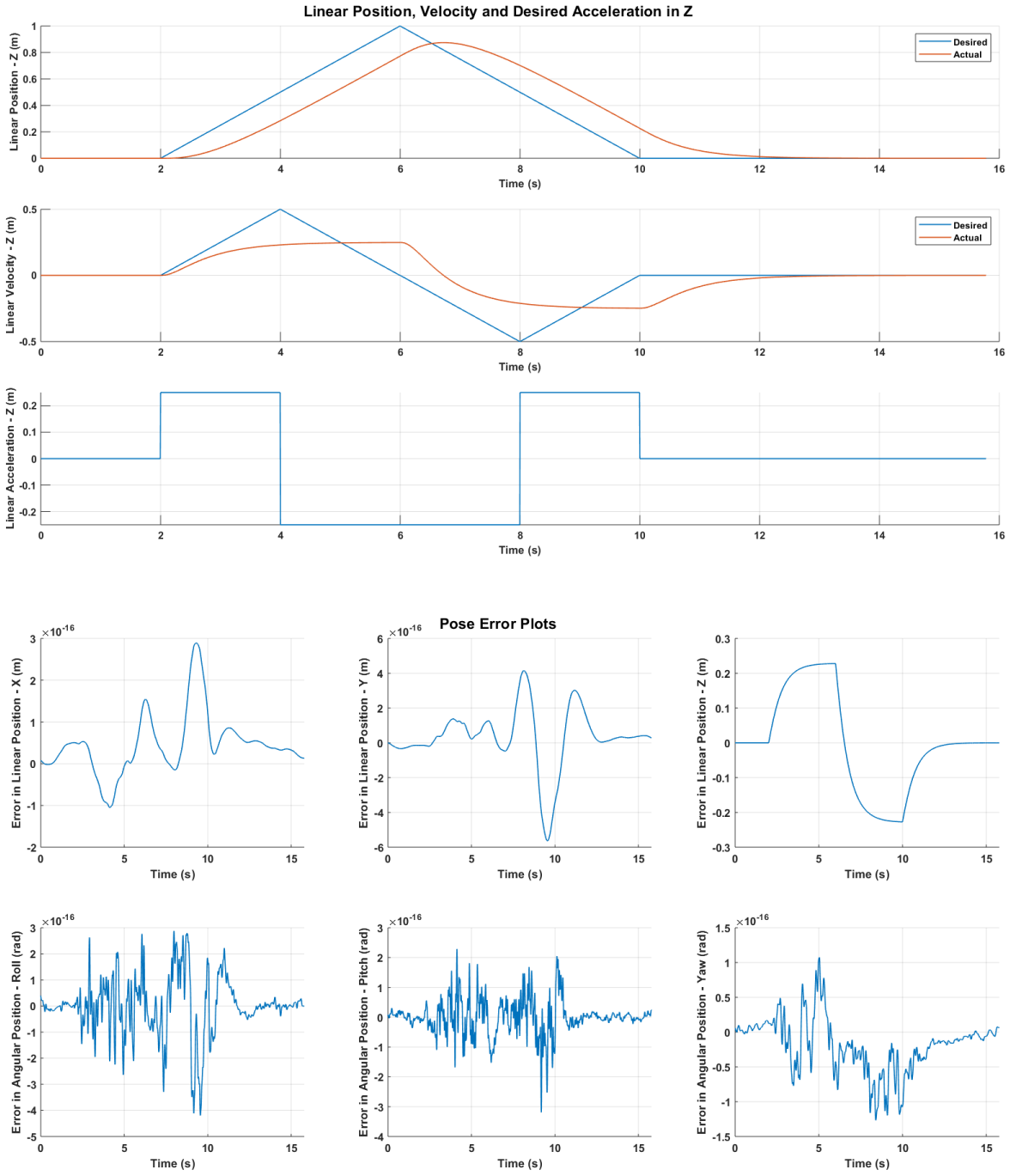


Figure 31: Z axis position, velocity, desired acceleration and pose error plots for LQR gains *Q = [0.1, 0.1, 40, 0.1, 0.1, 0.1, 0.1, 0.1, 20, 0.1, 0.1, 0.1]* and *R = [2, 2, 0.5, 2]*

On increasing the cost on Z velocity, and reducing control input cost for Z, the tracking performance has degraded further.

Gain Selection and Tuning

Now let us analyze the time domain characteristics for the LQR controller.

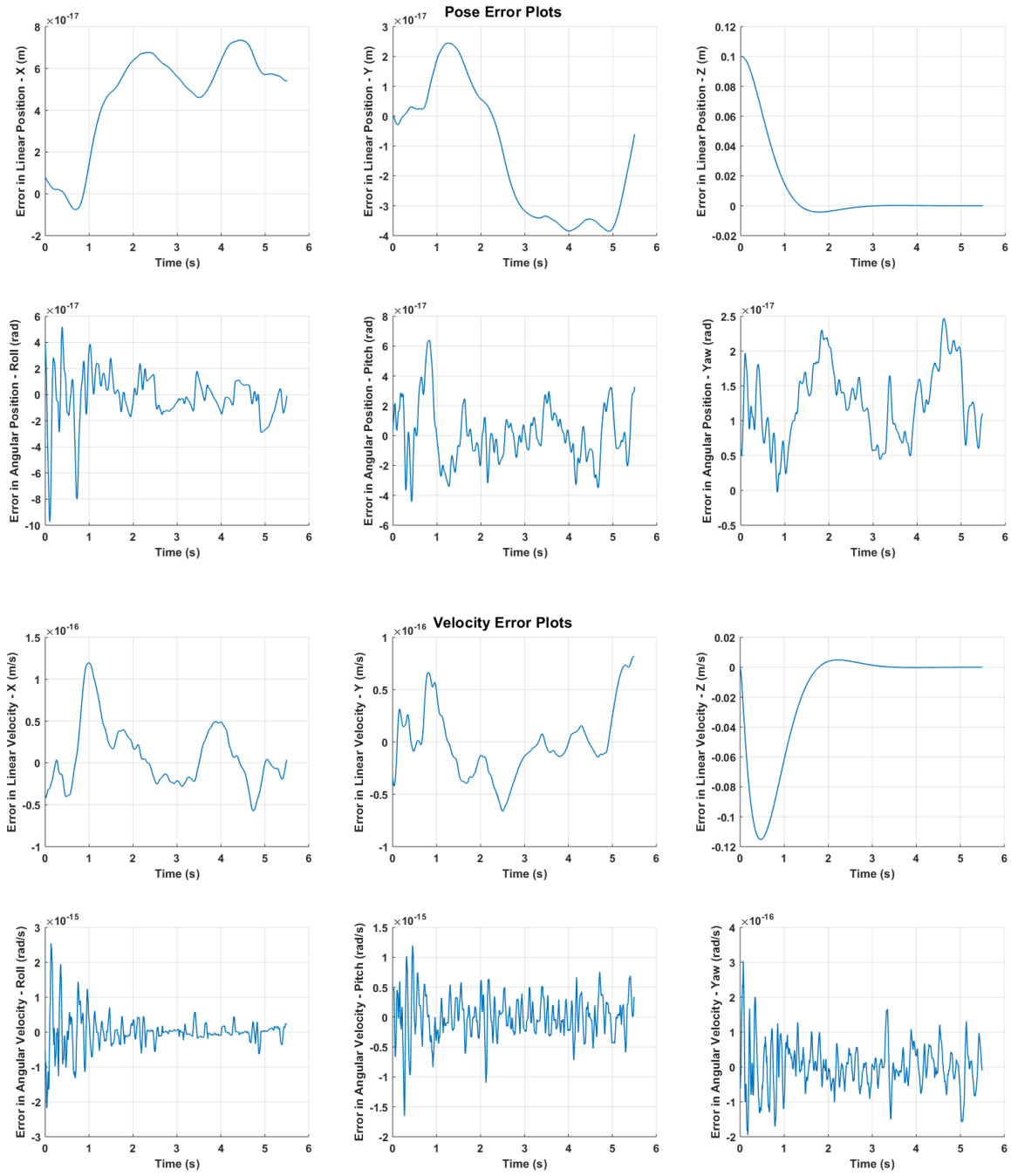


Figure 32: Error in responses for LQR gains
Q = [0.1, 0.1, 40, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]* and *R = [2, 2, 2, 2]

Position (Z) Characteristics	***Velocity (Z) Characteristics***
Rise Time: 0.8510	Rise Time: 3.5423e-04
Settling Time: 1.0768	Settling Time: 1.5146
Settling Min: 0.0902	Settling Min: -0.0049
Settling Max: 0.1042	Settling Max: 2.0837e-04
Overshoot: 4.2390	Overshoot: Inf
Undershoot: 0.0036	Undershoot: Inf
Peak: 0.1042	Peak: 0.1152
Peak Time: 1.7850	Peak Time: 0.4650
Steady-State Value: 0.09999	Steady-State Value: 4.544e-07

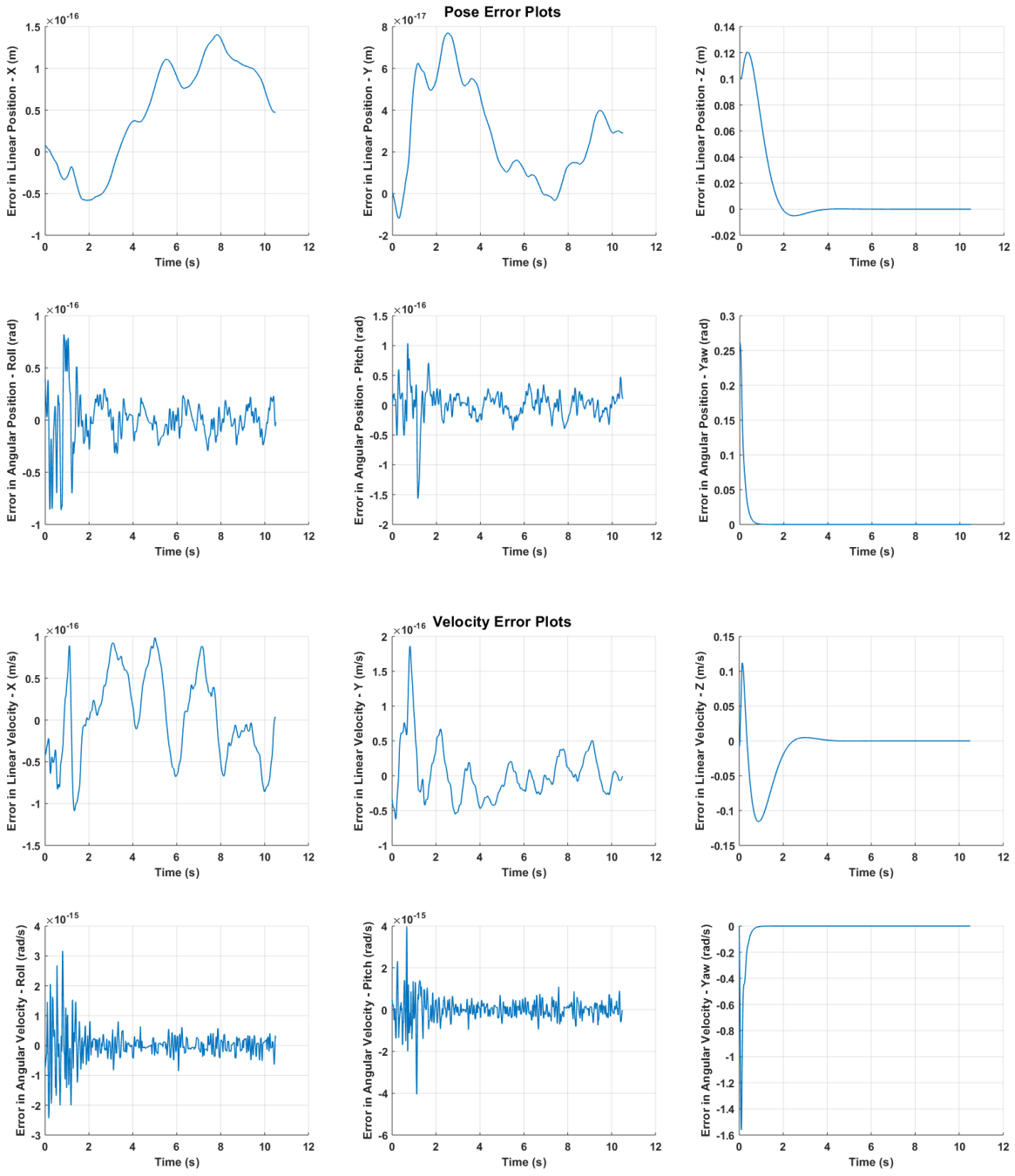


Figure 33: Error in responses for LQR gains
Q = [0.1, 0.1, 40, 0.1, 0.1, 5, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]* and *R = [2, 2, 2, 2]

Position (Yaw) Characteristics	*Velocity (Yaw) Characteristics*
Rise Time: 0.3106	Rise Time: NaN
Settling Time: 0.3657	Settling Time: 0.3828
Settling Min: 0.2364	Settling Min: NaN
Settling Max: 0.2618	Settling Max: NaN
Overshoot: 0	Overshoot: Inf
Undershoot: 2.2467e-15	Undershoot: 0
Peak: 0.2618	Peak: 1.5618
Peak Time: 4.5200	Peak Time: 0.0950
Steady-State Value: 0.2618	Steady-State Value: 5.498e-06

Problem 7 - Solution

A straight-line time parameterized polynomial trajectory with initial and final endpoint constraints at waypoints (0, 0, 1) and (0, 0, 10) with zero velocity, acceleration and jerk and the endpoints was developed with the help of a quadratic program. The optimization was done in a nested loop architecture, where the inner loop solved for coefficients for a minimum snap trajectory polynomial using QP and the outer loop ensured that the acceleration at all points was less than the given value, by iteratively increasing the final time in the cost function by a small timestep. The formulation of cost function and quadratic program is explained in Problem 8. The code is implemented in *bounded_acceleration.m* script.

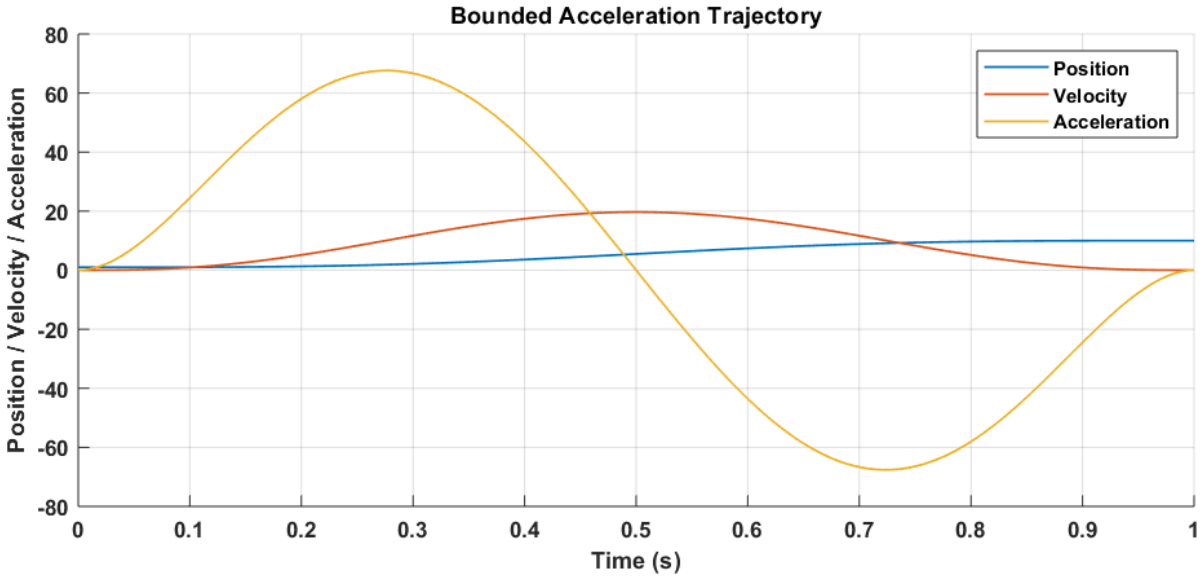


Figure 34: Unbounded acceleration (max = 67.6187 m/s^2) trajectory for T_f = 1s

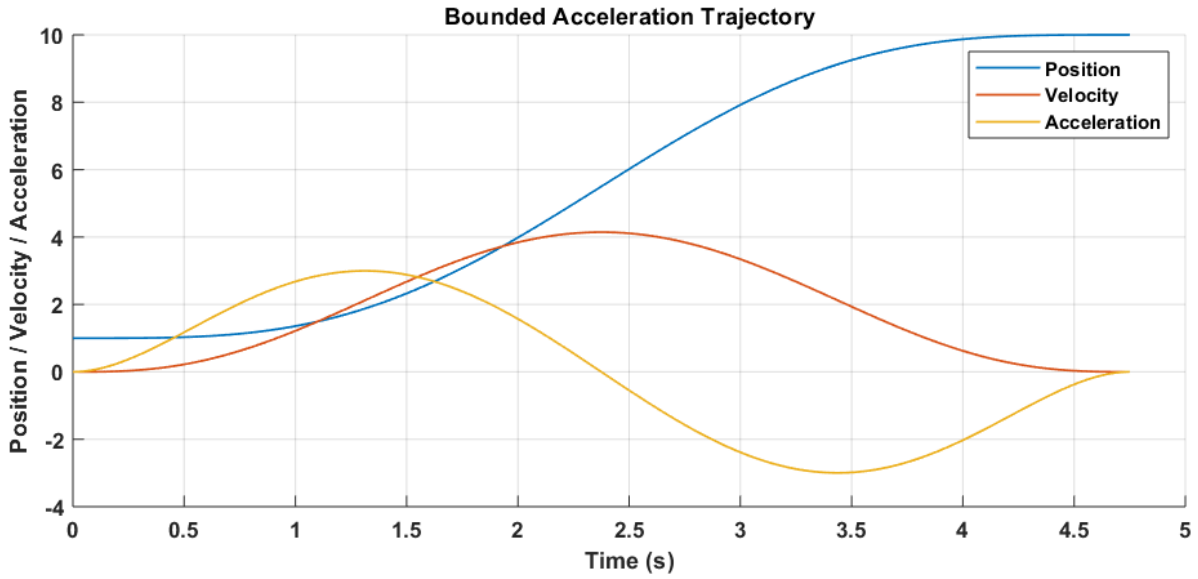


Figure 35: Bounded acceleration (max = 2.997 m/s^2) trajectory for T_f = 4.75s

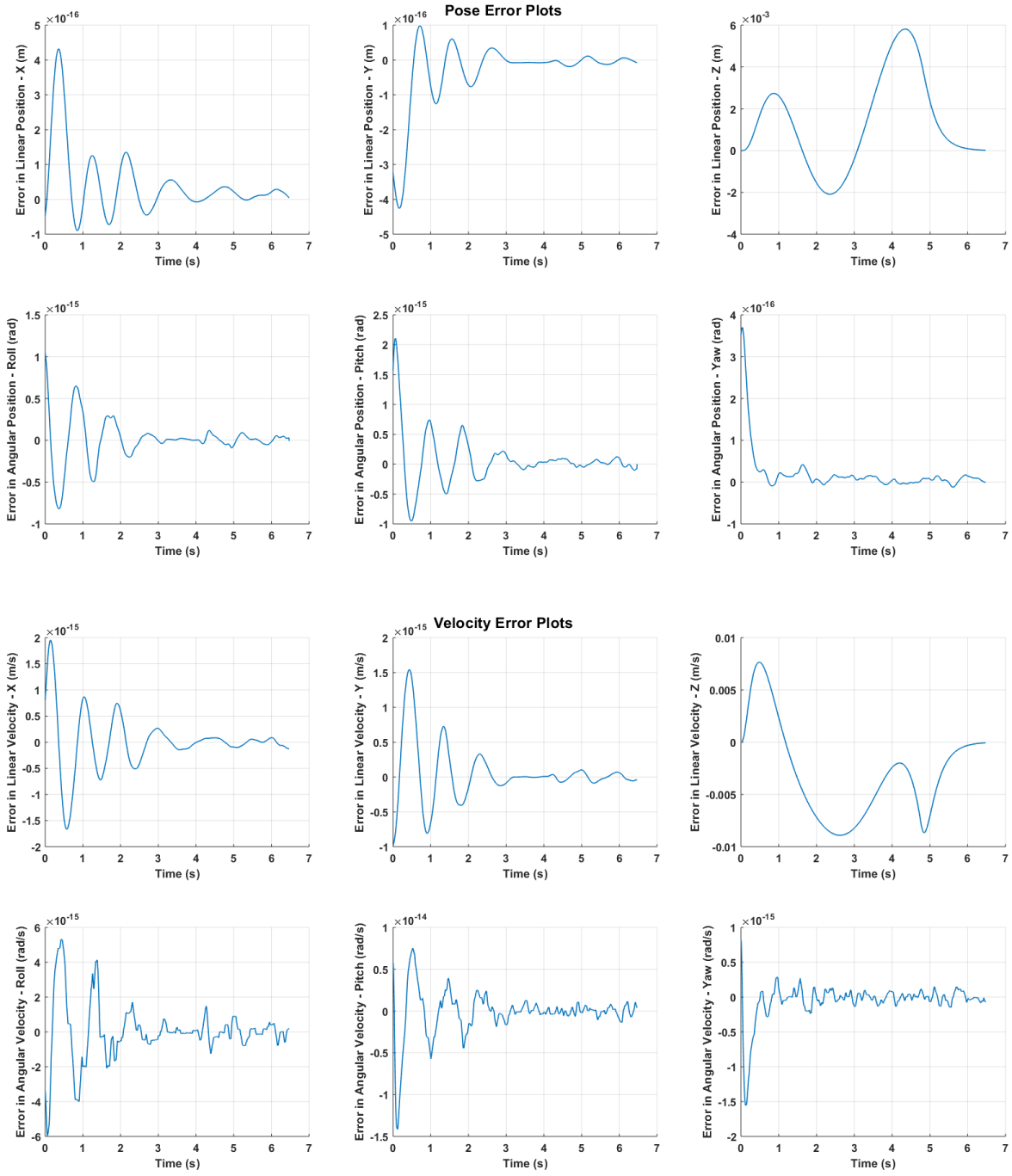


Figure 36: Error plots bounded acceleration (3 m/s^2) trajectory for T_f = 4.75s

It can be observed that the robot tracks both the position and velocity trajectories almost perfectly with very low magnitudes of errors.

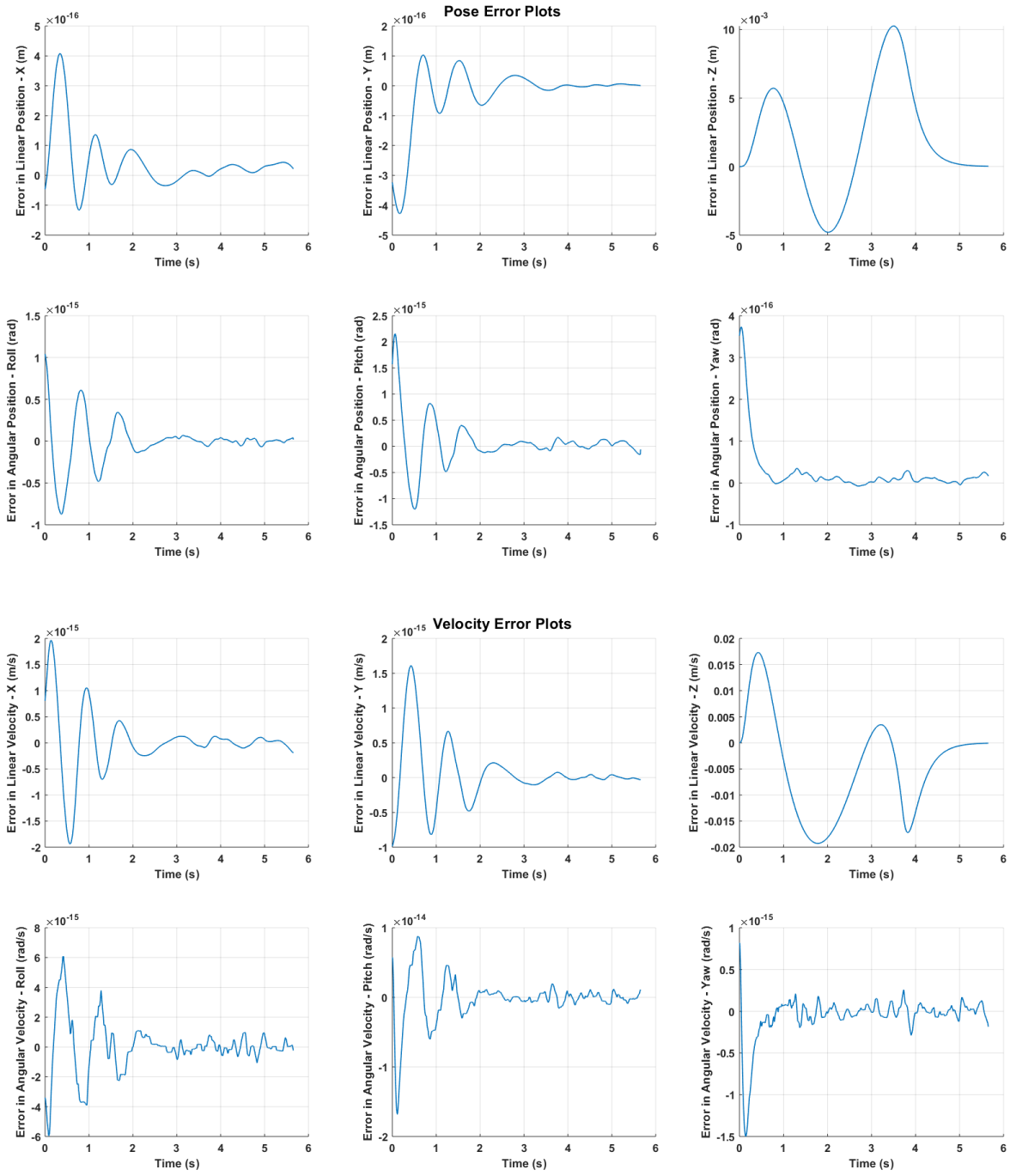


Figure 37: Error plots bounded acceleration (5 m/s^2) trajectory for T_f = 3.7s

It can be observed that the robot still tracks both the position and velocity trajectories almost perfectly with very low magnitudes of errors, with just a slight increase from the previous case.

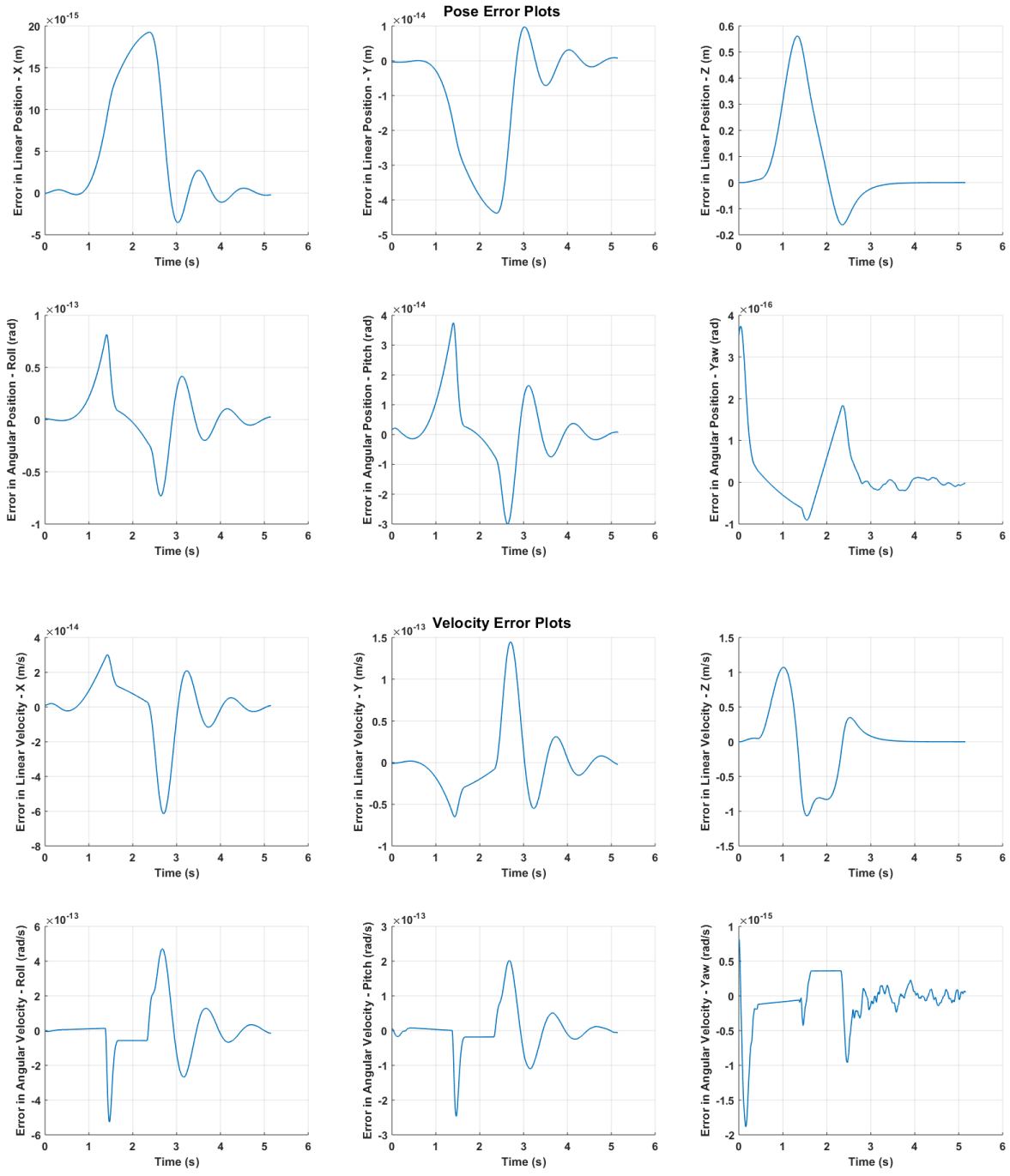


Figure 38: Error plots bounded acceleration (10 m/s^2) trajectory for T_f = 2.65s

Upon further increasing the acceleration, it can be observed that the robot tracking starts to degrade slightly in both the position and velocity trajectories. The error in position is now up to 55 cm. The source of the error is that the motor response is not quick enough to follow the desired motor RPMs, in one loop of the control system, to track the trajectory. In order to accommodate higher values of accelerations in the trajectory, the time for the robot to reach the end point will reduce, which is opposed by the slow motor response.

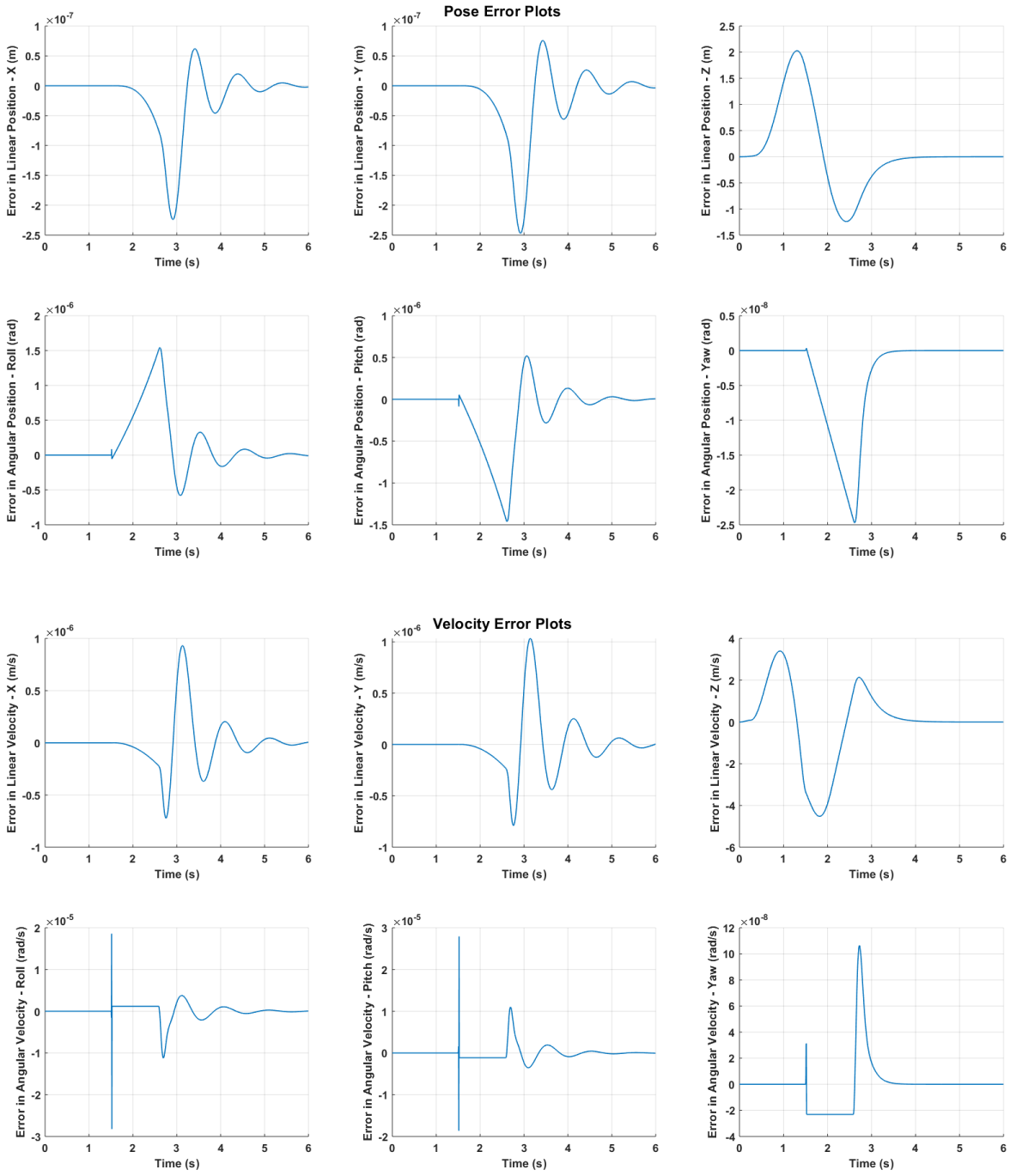


Figure 39: Error plots bounded acceleration (15 m/s^2) trajectory for T_f = 2.15s

Further increasing the acceleration again, it can be observed that the error in position is now up to 2 m. We can now increase the motor gain for this case and observe the results.

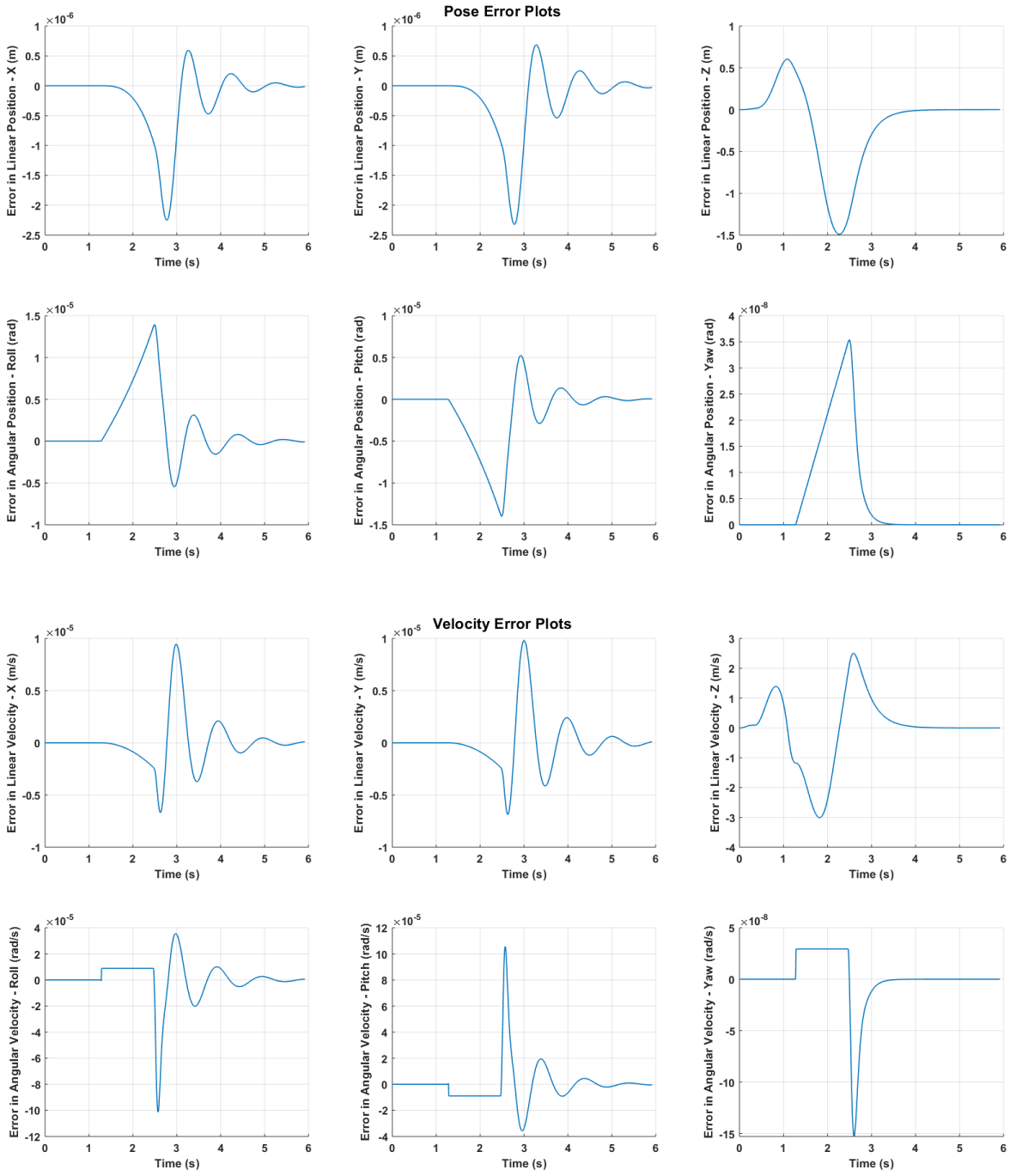


Figure 40: Error plots bounded acceleration (15 m/s^2) trajectory for T_f = 2.15s (Motor gain = 50.0, Max RPM = 22,000)

Varying the motor gain did not really improve the results much. However, increasing the maximum threshold on motor RPM to 22,000, reduced the error magnitudes and improved the tracking performance.

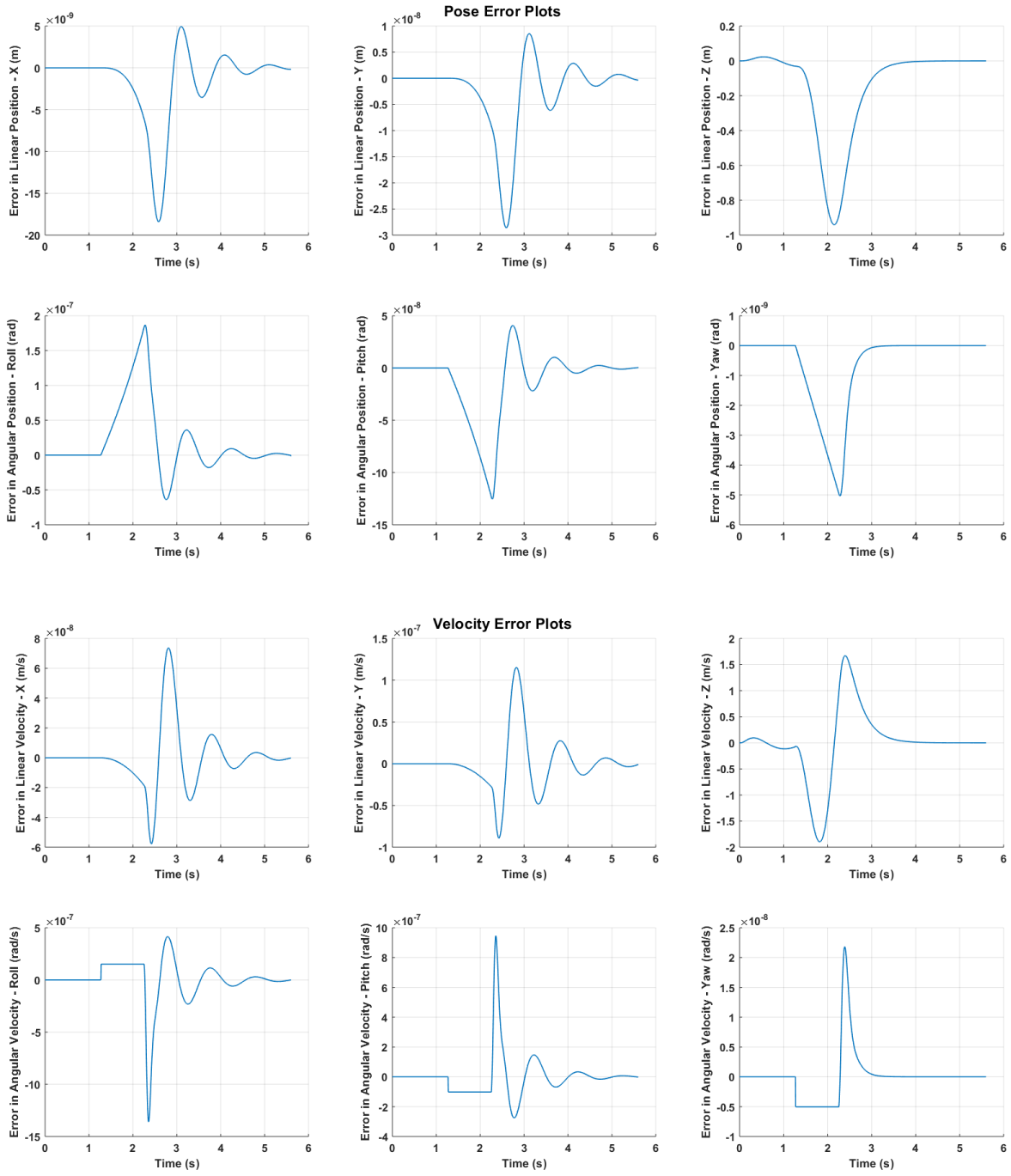


Figure 41: Error plots bounded acceleration (15 m/s^2) trajectory for T_f = 2.15s (Motor gain = 70.0, Max RPM = 25,000)

Further increasing the motor gain and the maximum threshold on motor RPM to 25,000, it significantly reduced the error magnitudes and improved the tracking performance. The undershoot in Z axis position error visible is due to selection of PD gains and corresponds to the overshoot in tracking which can be corrected by choosing a better set of PD gains. For this experiment, default PD gains are used and are not varied.

Thus, it can be concluded that by upgrading to better motors, we can increase the bound on acceleration in trajectories and improve the tracking performance of a quadrotor.

Problem 8 - Solution

In order to generate minimum energy elliptical trajectories, we can optimally generate minimum snap trajectories with certain constraints. For a minimum snap trajectory, we have a 7th degree polynomial with 8 coefficients to solve for the position expression *x*. We can differentiate this expression to obtain equations for velocity, acceleration, jerk and snap.

$$v = \frac{dx}{dt}$$

$$acc = \frac{dv}{dt}$$

$$jerk = \frac{d(acc)}{dt}$$

$$snap = \frac{d(jerk)}{dt}$$

The cost function for the quadratic program is,

$$CF = \int_0^{tf} snap^2 \, dt$$

The hessian of the above cost function can be found by differentiating it twice or by using the *hessian* function in SymPy library in Python. We now need to formulate the constraints for the endpoints. We can divide the entire elliptical trajectory into 4 small segments of each quadrant and solve for the polynomial coefficients individually. The optimizer is run separately for all the axes (X, Y, Z, and Yaw).

For the first trajectory phase, at the initial waypoint, all the conditions are zero. At the second waypoint we have the constraints for only position (2, 1) with zero acceleration and jerk constraints. Velocity constraints are not given for the initial case, however the initial velocity values for the next segment of the trajectory are updated with the current segment final velocity. The velocity is computed after the coefficients have been solved for and are substituted in the velocity expression for a given time *tf*.

The constraints for the second trajectory are formulated in a similar manner, with the difference being, both the position and tangential velocity constraints are specified at every waypoint. For example, for the first waypoint we have position (2, 1) with velocity (1, 0). The acceleration and jerk constraints were relaxed here in order to obtain the elliptical shape. However, they were updated at every segment, in a similar manner as the velocities are updated in the previous case, in order to maintain the continuity at the endpoints to ensure a smooth trajectory. A similar approach was used for the last trajectory phase with final velocities (0, 0).

The *qp solver* from *cvxopt* library in Python was used to solve the quadratic program to solve for the coefficients for the polynomial. The optimization was done iteratively in the same manner as Problem 7, to ensure that the robot could track the trajectory well and the shape of the trajectory for second phase was elliptical.

The optimal trajectory generation is implemented in *trajectory_optimizer.py* script.

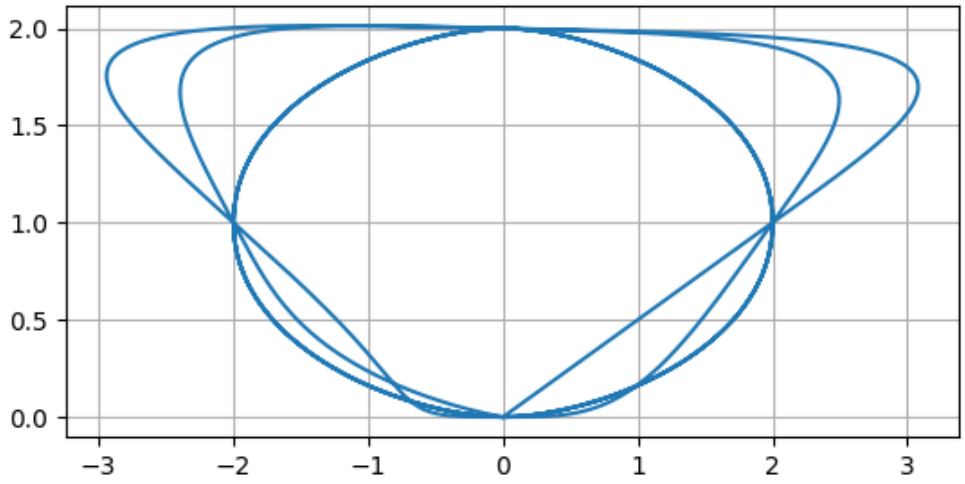


Figure 42: 2D plot showing the elliptical trajectory for all phases for *tf* = 2s

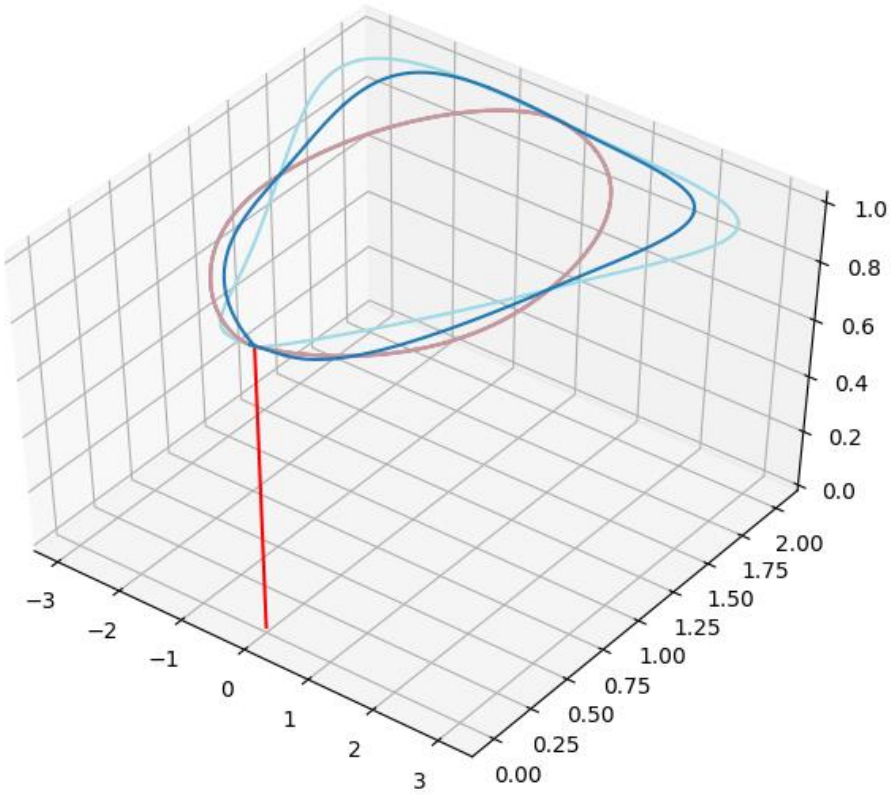


Figure 43: 2D plot showing the elliptical trajectory for all phases for *tf* = 2s

The optimal trajectory for all three phases is shown above. Trajectory phase one and there (blue colors) are not exactly elliptical as we did not have any velocity constraints for these phases. The purple trajectory for phase two loops is almost a perfect ellipse.

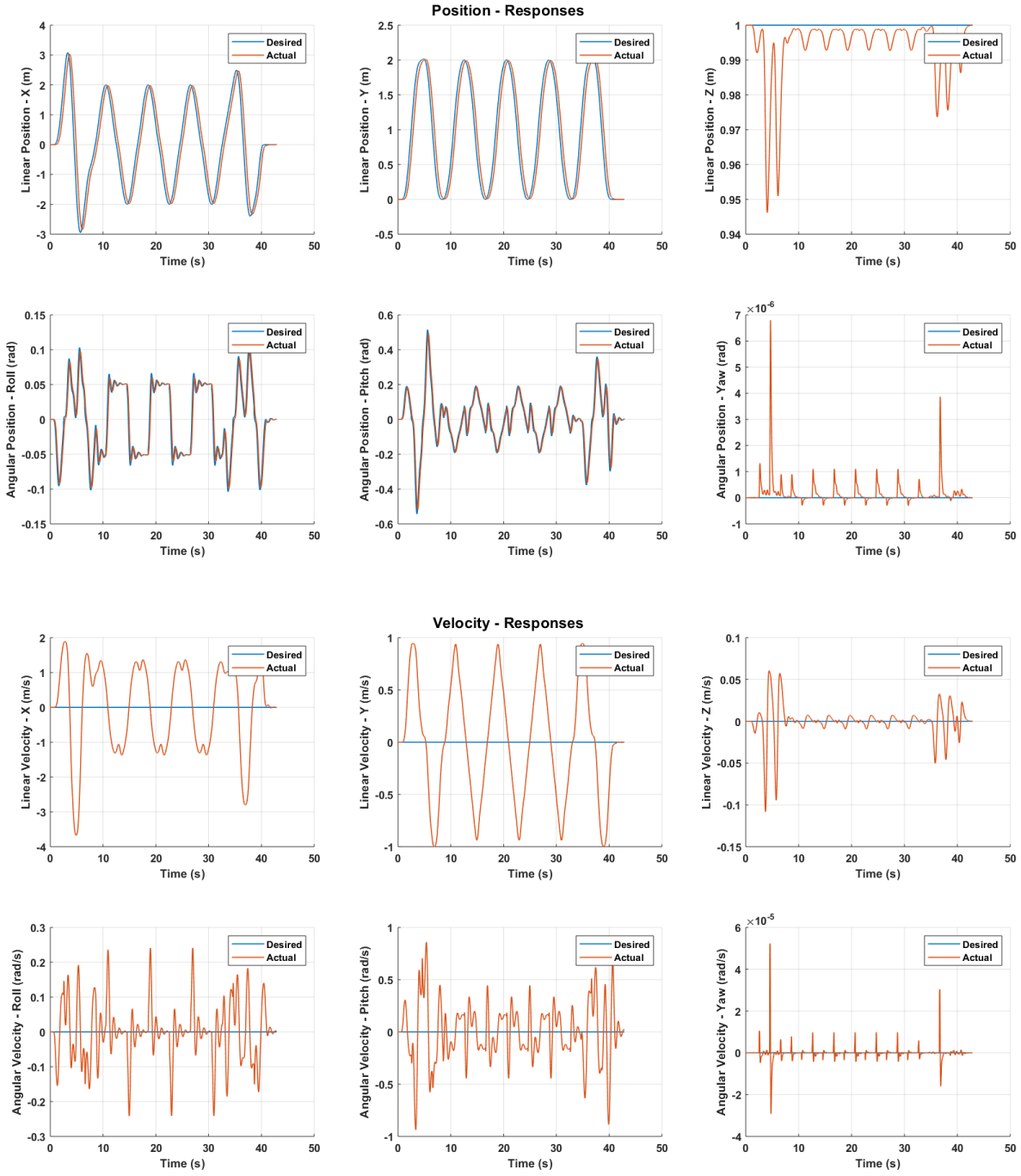


Figure 44: Position and velocity response plots for all phases for *tf* = 2s

The second trajectory is tracked **three times** in a loop for the analysis. The robot tracks the trajectories in X and Y well and also follows the velocity constraints that we specified at the waypoints.

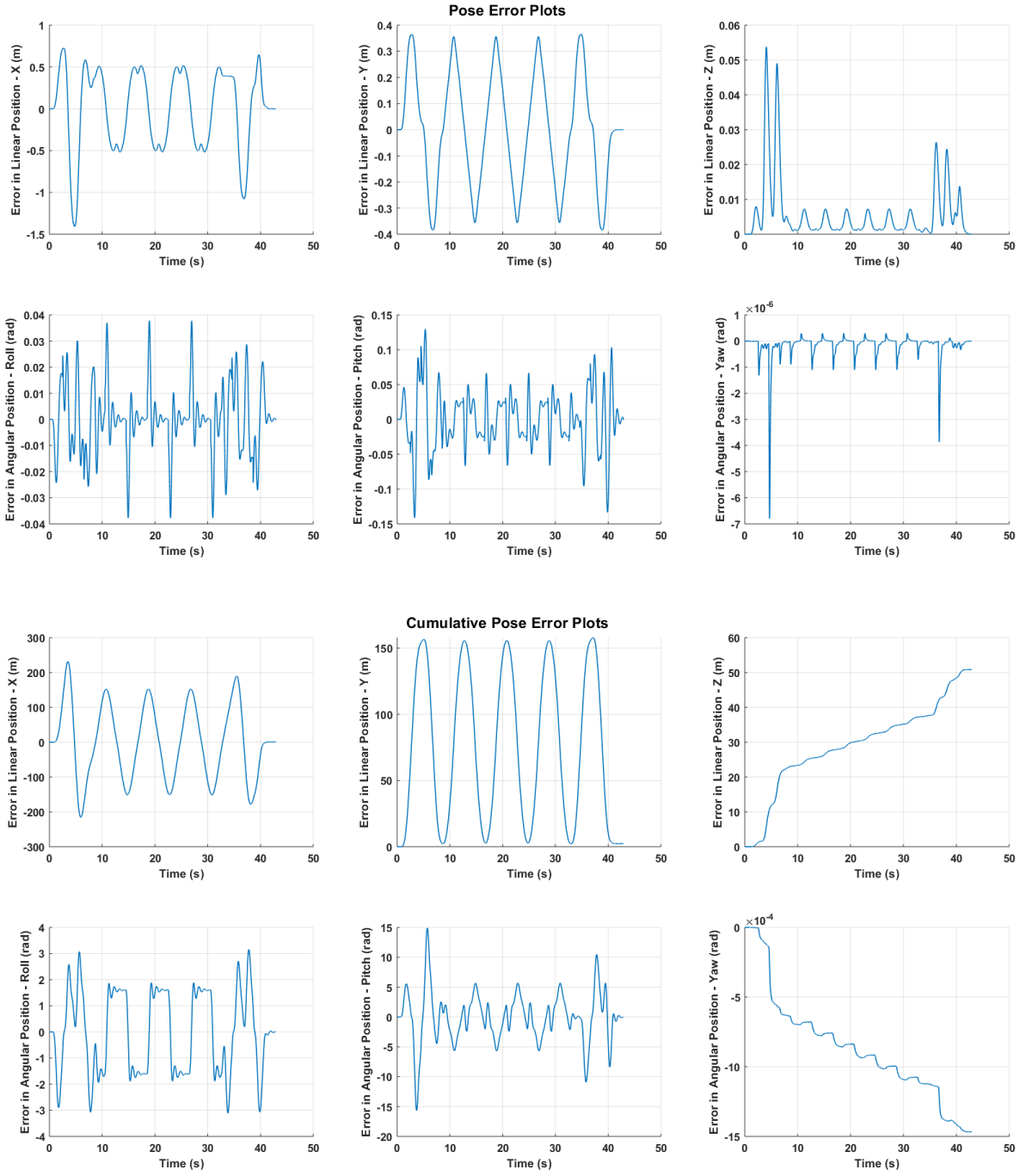


Figure 45: Error and cumulative error position plots for all phases for *tf* = 2s

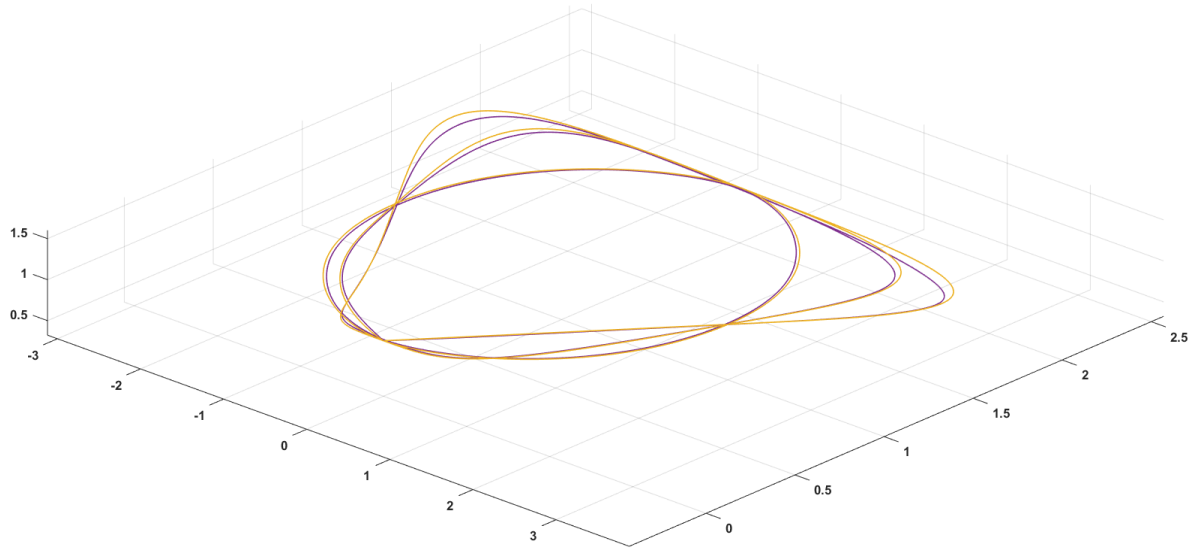
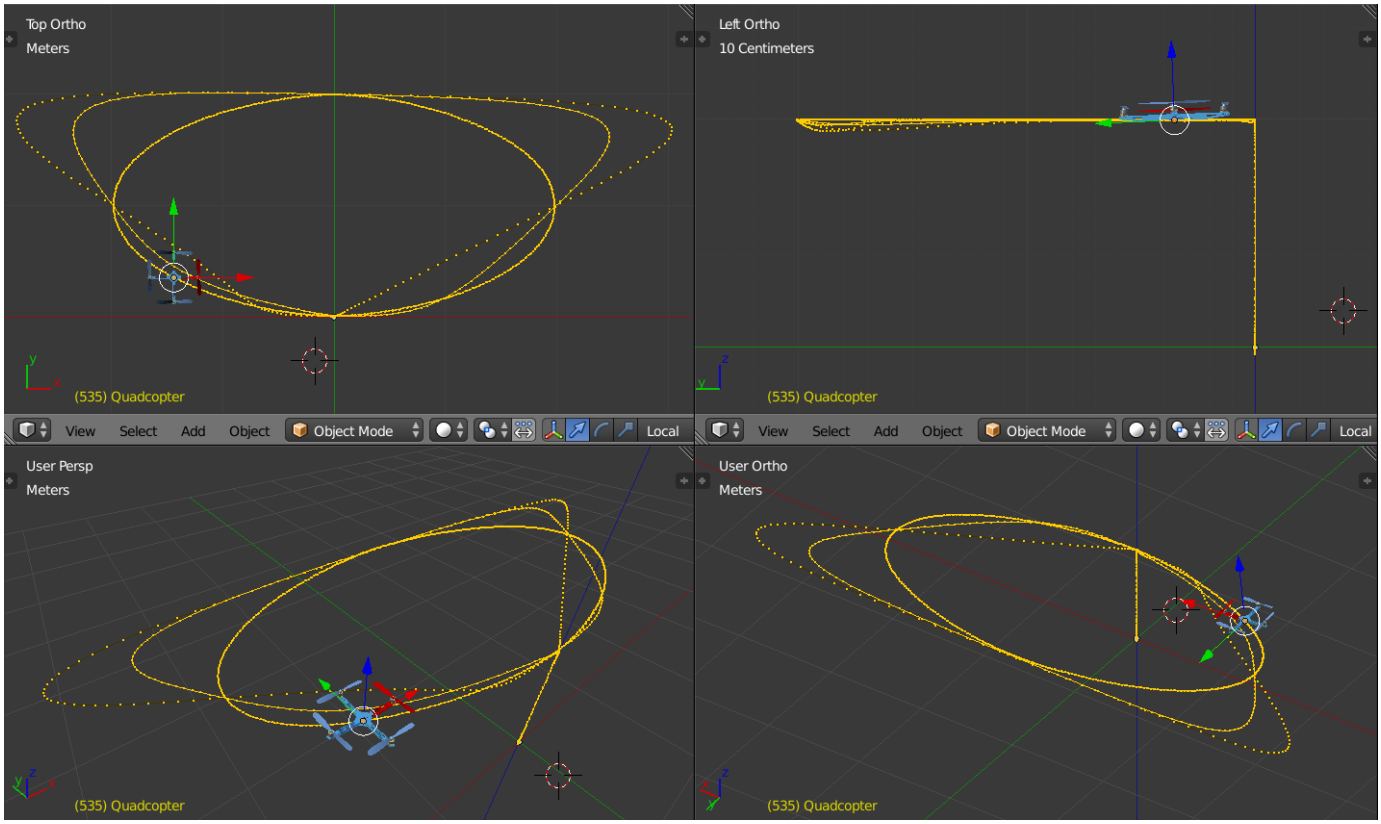


Figure 46: Desired (yellow) and actual (purple) trajectory for all phases for *tf* = 2s



Video 2: Blender simulation for elliptical trajectory tracking
(Direct link: https://drive.google.com/drive/folders/1bP0ahk_oedvzrteOZJsae2vlInPBfaSC?usp=sharing)

Tangent Velocity = 2m/s

We can now perform the same for higher velocities. In order to achieve elliptical shape trajectories, the time for each segment had to be reduced to 1.5s.

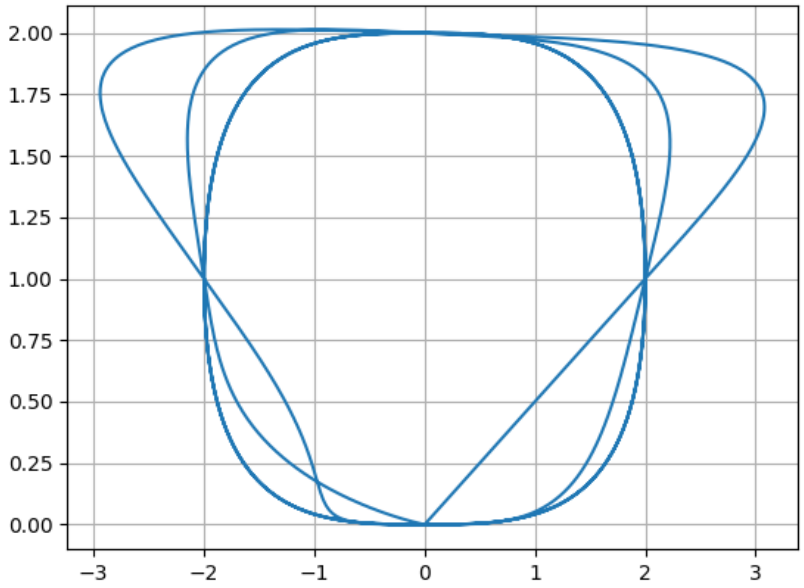


Figure 47: 2D plot showing the elliptical trajectory for all phases for *tf* = 1.5s, *v* = 2m/s

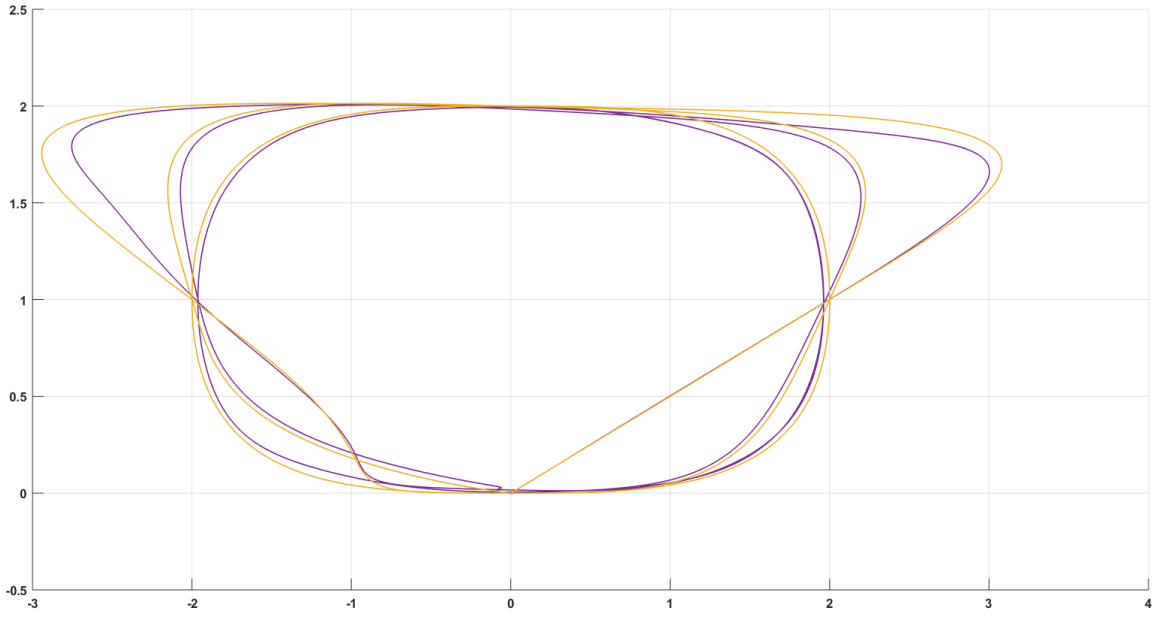


Figure 48: Desired (yellow) and actual (purple) trajectory for all phases for *tf* = 1.5s, *v* = 2m/s

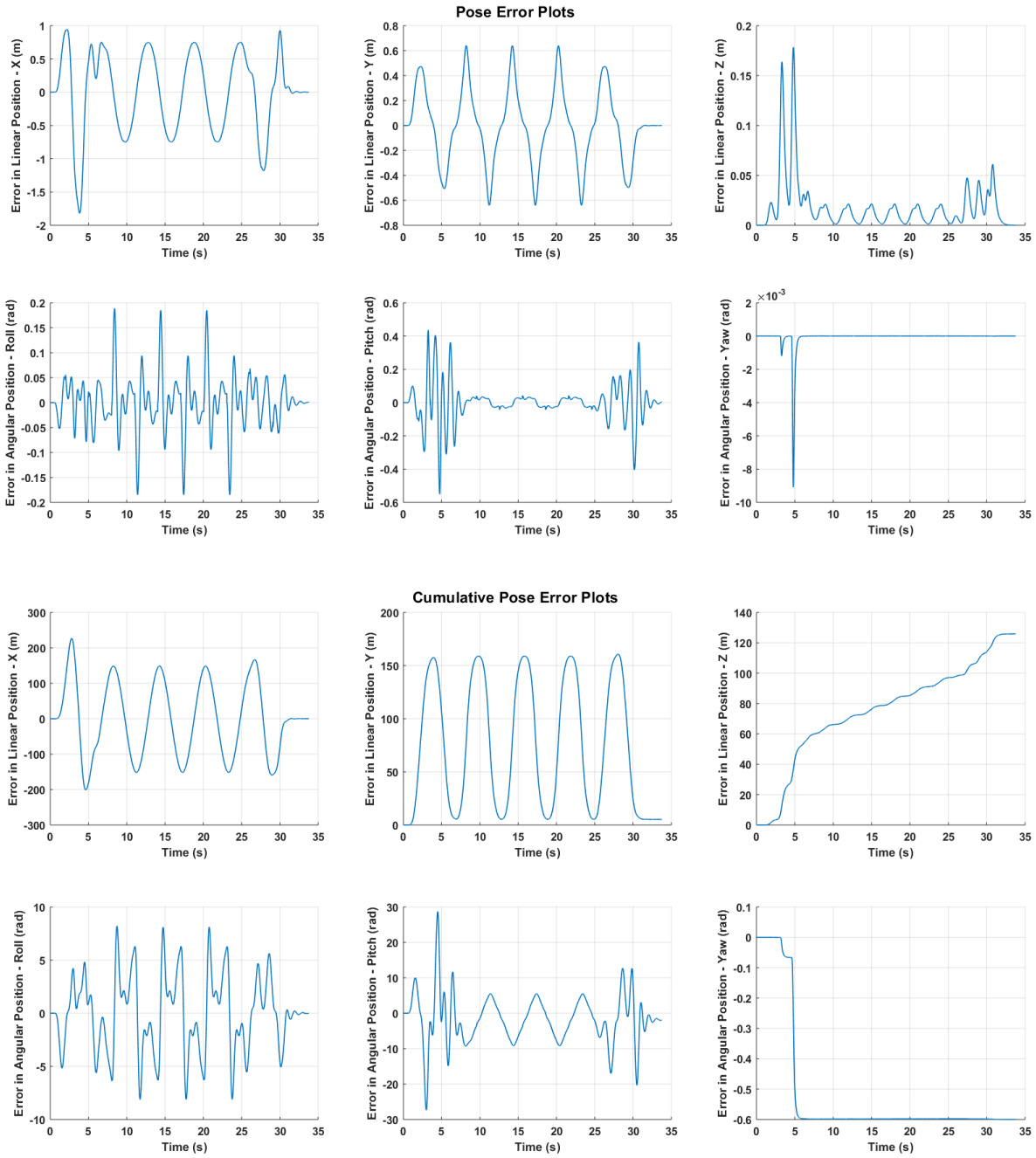


Figure 49: Error and cumulative error position plots for all phases for *tf* = 1.5s, *v* = 2m/s

Tangent Velocity = 3m/s

In order to achieve *near* elliptical shape trajectories, the time for each segment had to be reduced to 1.5s. But the trajectories were not elliptical exactly, because the time for each segment cannot be reduced further to ensure dynamic feasibility.

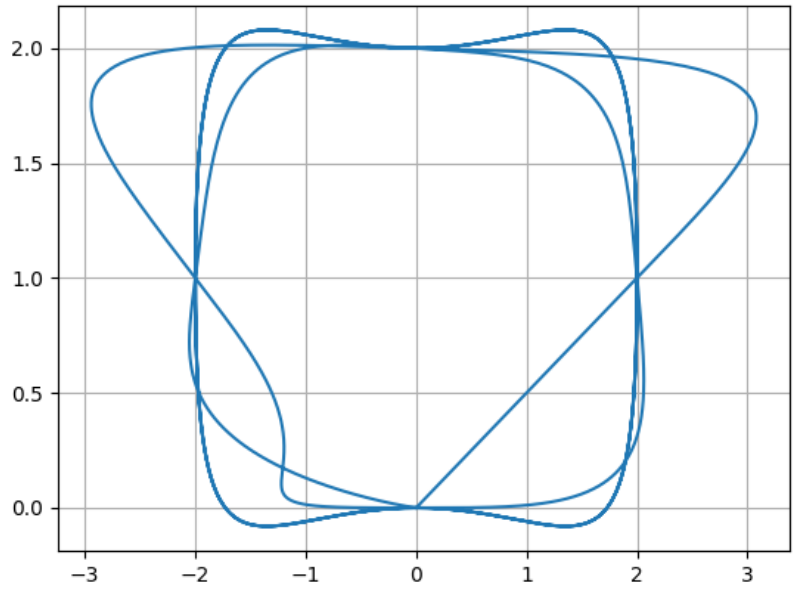


Figure 50: 2D plot showing the elliptical trajectory for all phases for *tf* = 1.5s, *v* = 3m/s

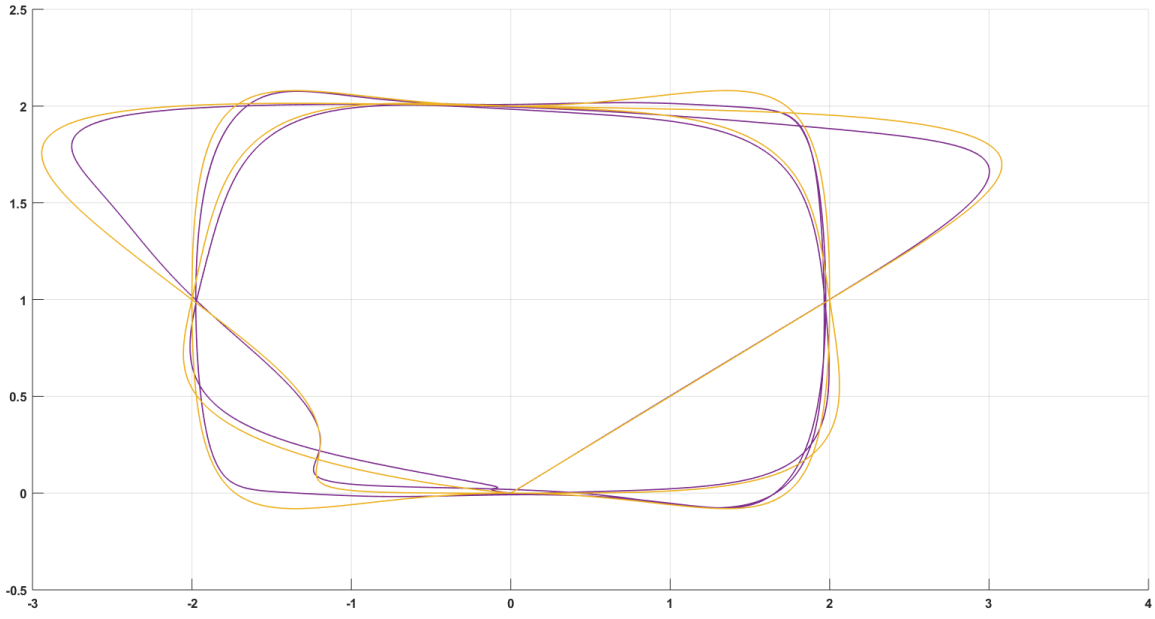


Figure 51: Desired (yellow) and actual (purple) trajectory for all phases for *tf* = 1.5s, *v* = 3m/s

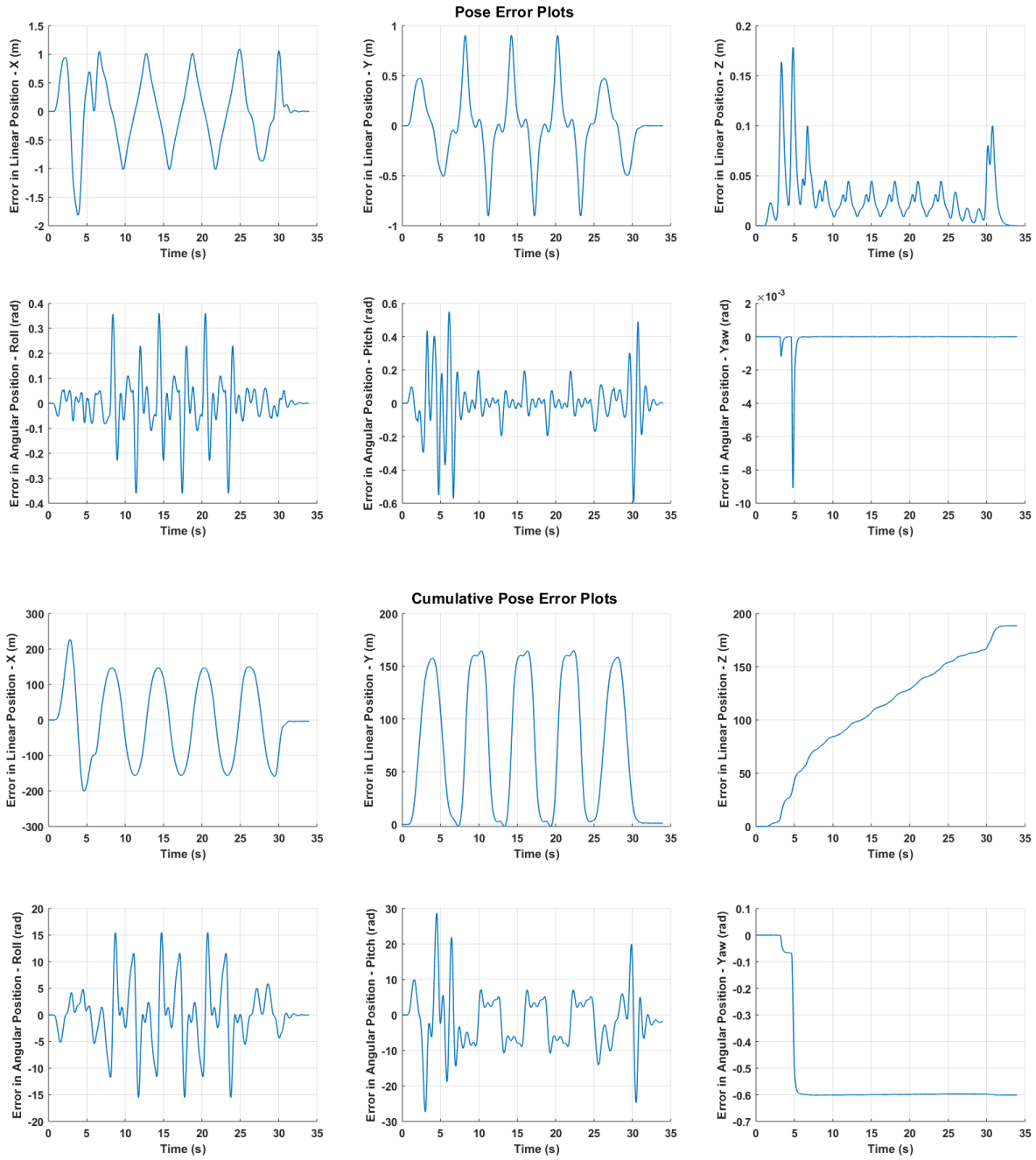


Figure 52: Error and cumulative error position plots for all phases for *tf* = 1.5s, *v* = 3m/s

Remarks

For higher velocities, the tracking performance is degraded with higher magnitudes of error, which can be seen from the cumulative error plots as well. But it still maintains the given velocity constraints, but with errors up to 0.5 m/s. If the velocity was increased further, the robot was unable to track the trajectory in the given time and lost stability.

Problem 9 - Solution

The problem is same as the previous one but this time a heading command is given for all points, pointing to the center of the ellipse at c = (0, 1). To maintain continuity in heading, an offset of 2π is added after every complete rotation in yaw. The heading or the yaw is calculated as follows.

$$heading = atan2(y - cy, x - cx) + 2n\pi + offset$$

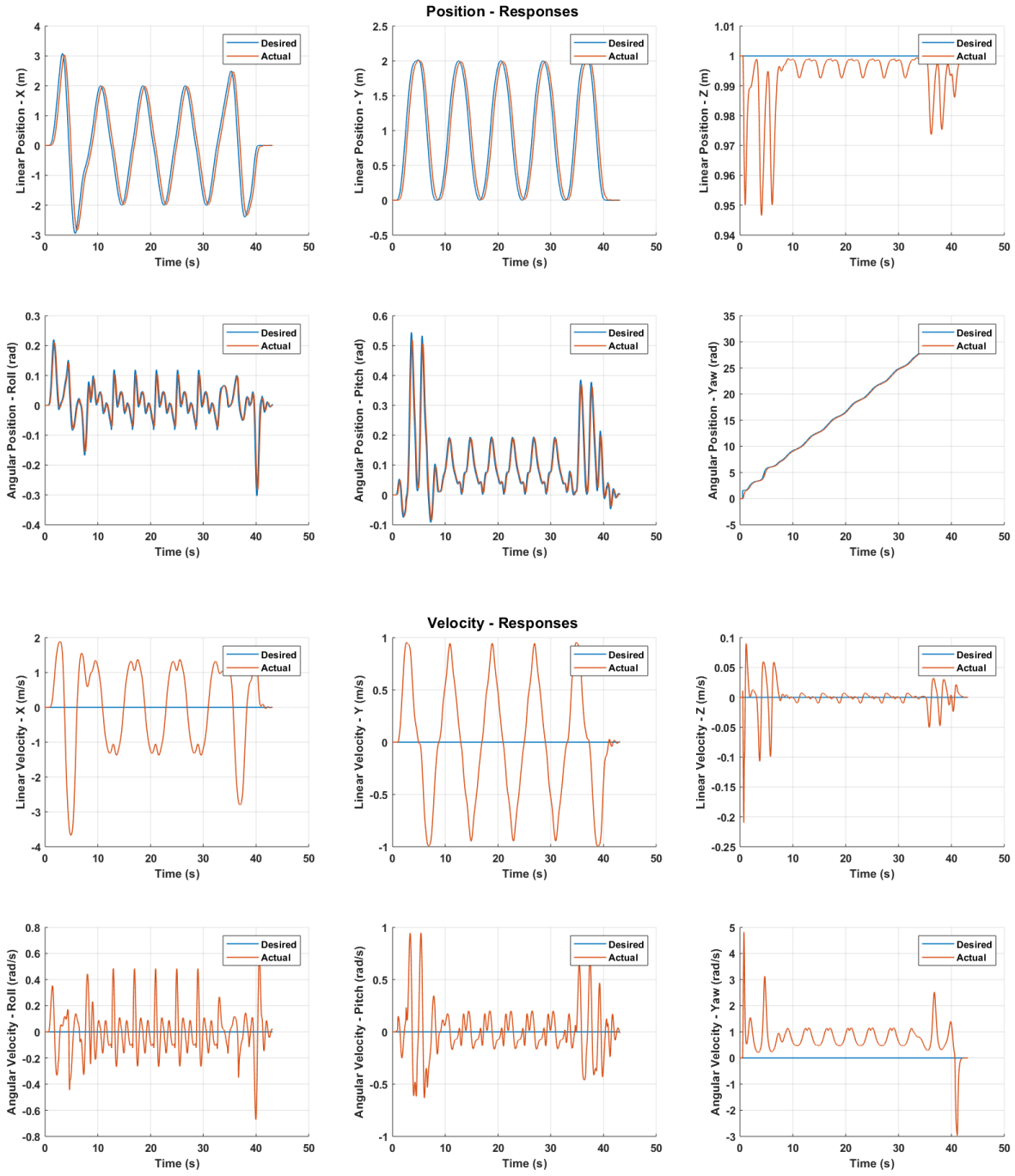


Figure 53: Position and velocity response plots for all phases for *tf* = 2s, *v* = 1m/s

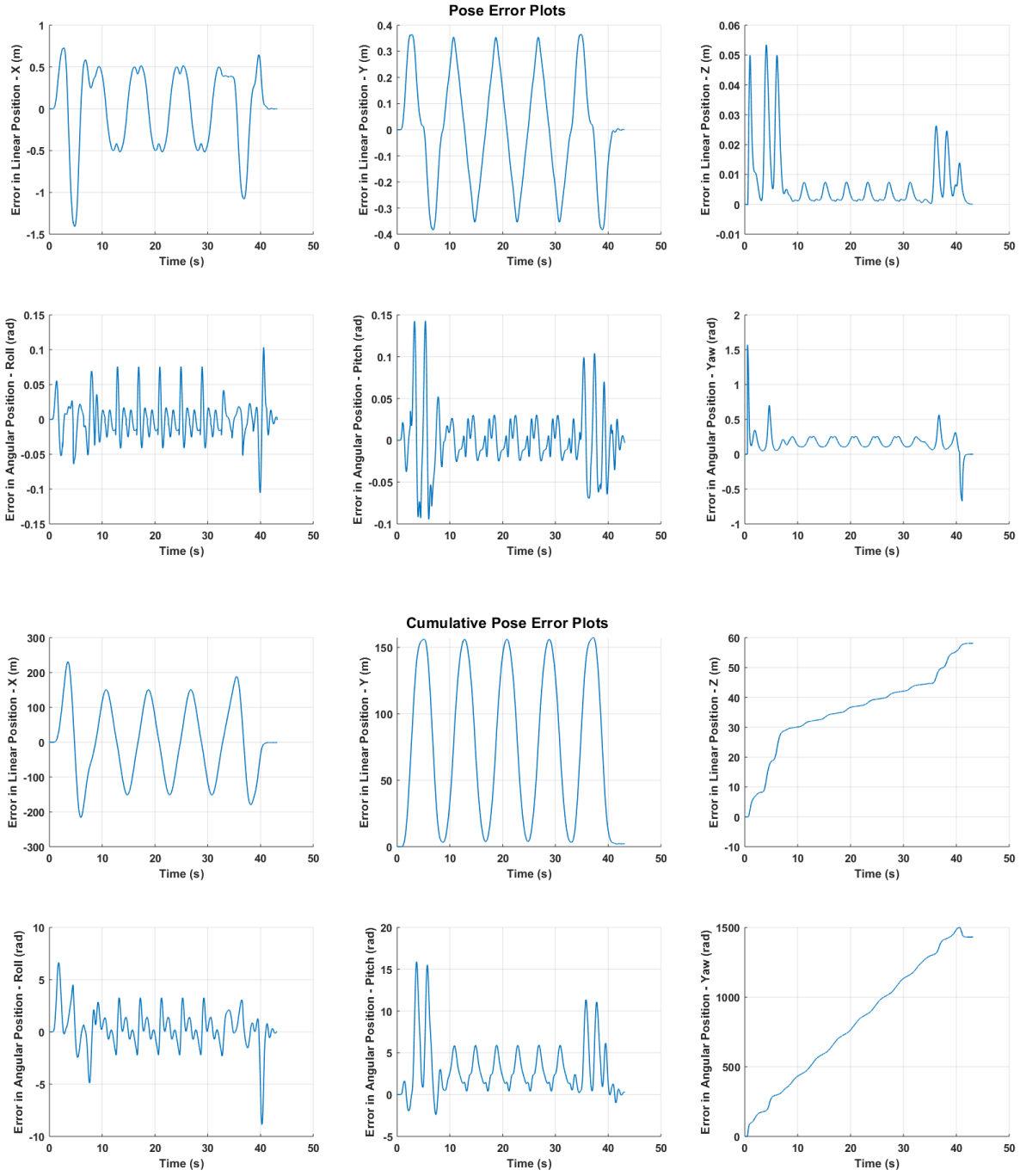


Figure 54: Error and cumulative error position plots for all phases for *tf* = 2s, *v* = 1m/s

The result of the tracking is good. The attitude controller is able to track the yaw with decent accuracy and magnitudes of error.

Tangent Velocity = 2m/s

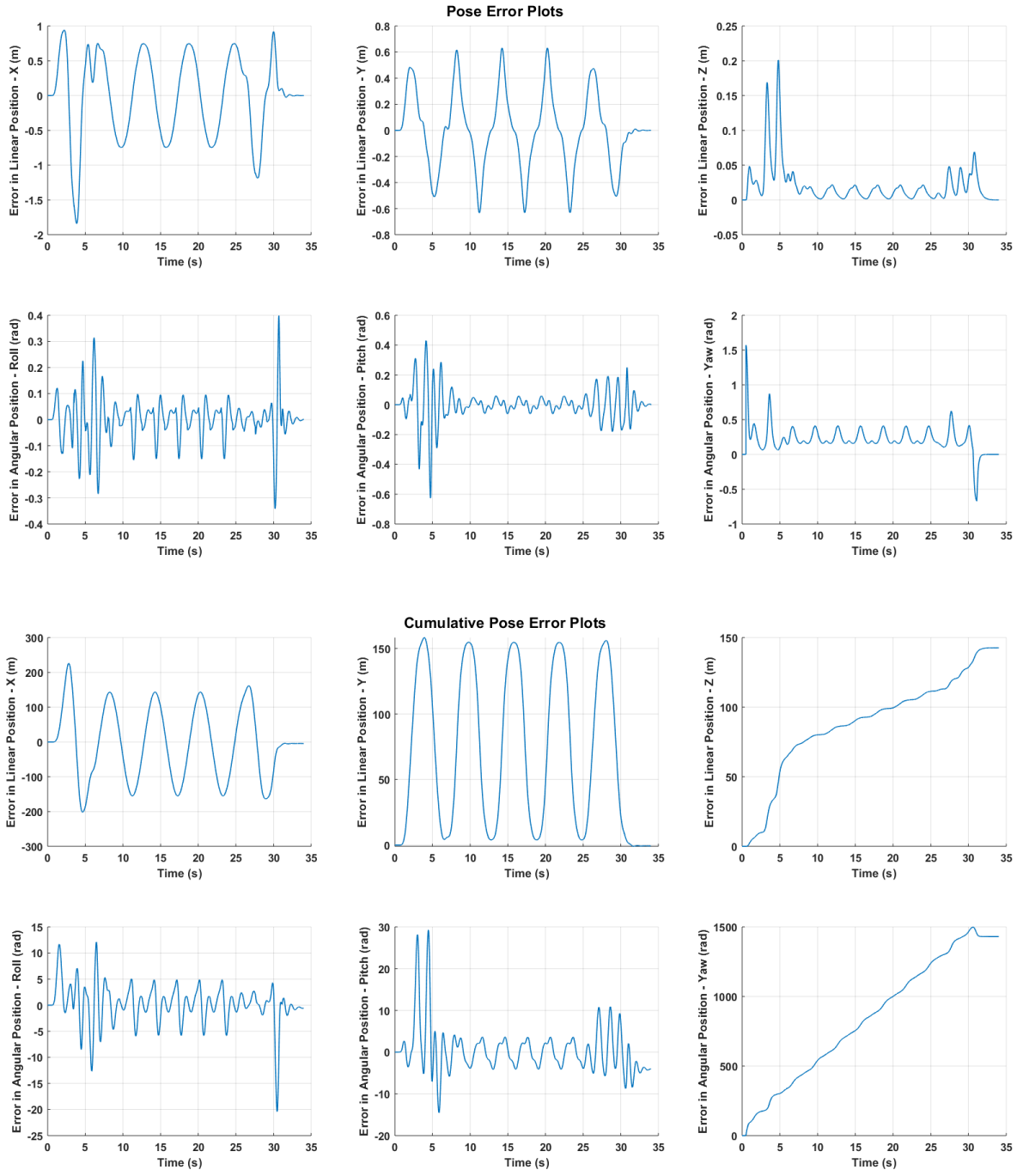


Figure 55: Error and cumulative error position plots for all phases for *tf* = 1.5s, *v* = 2m/s

Tangent Velocity = 3m/s

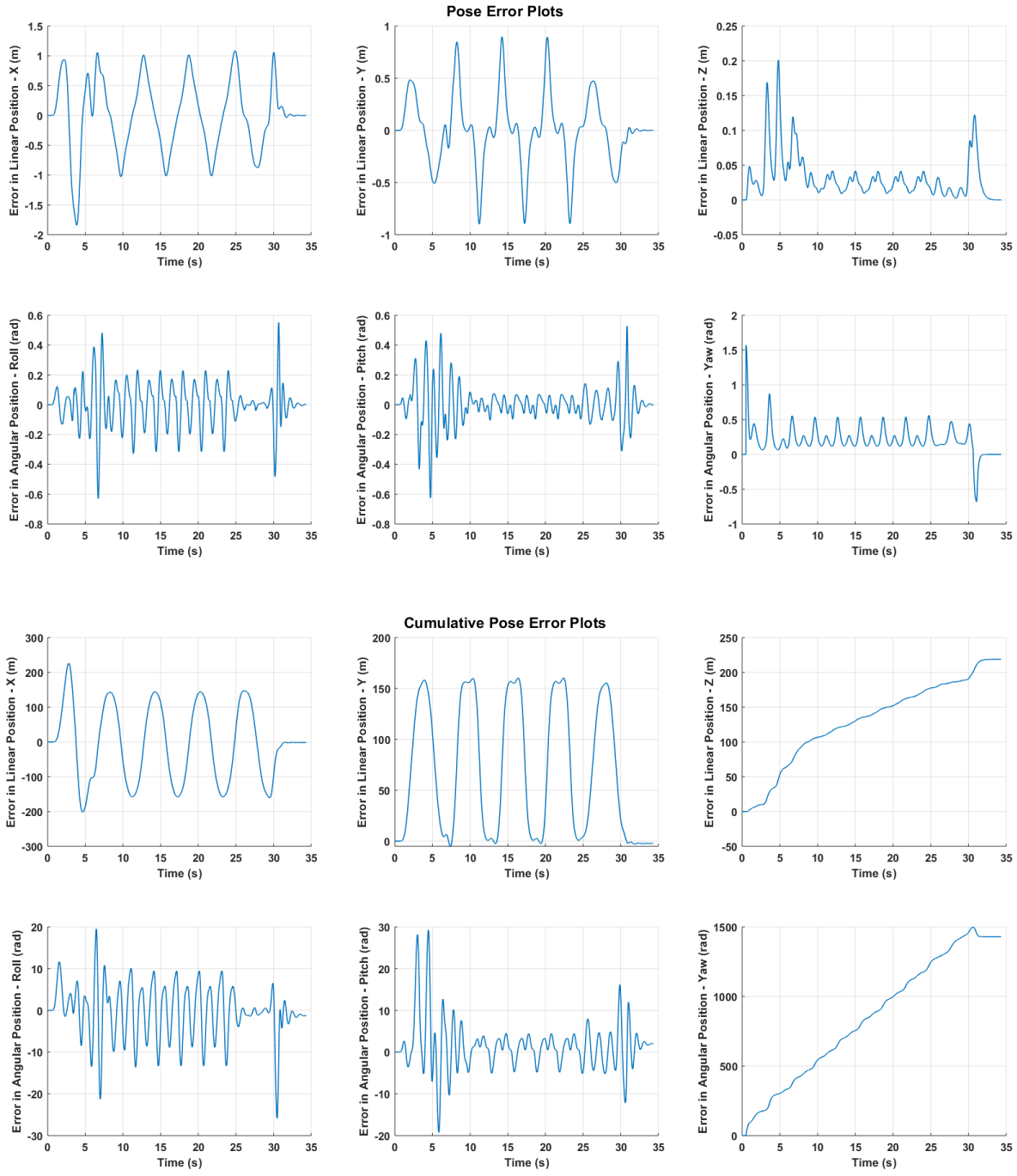
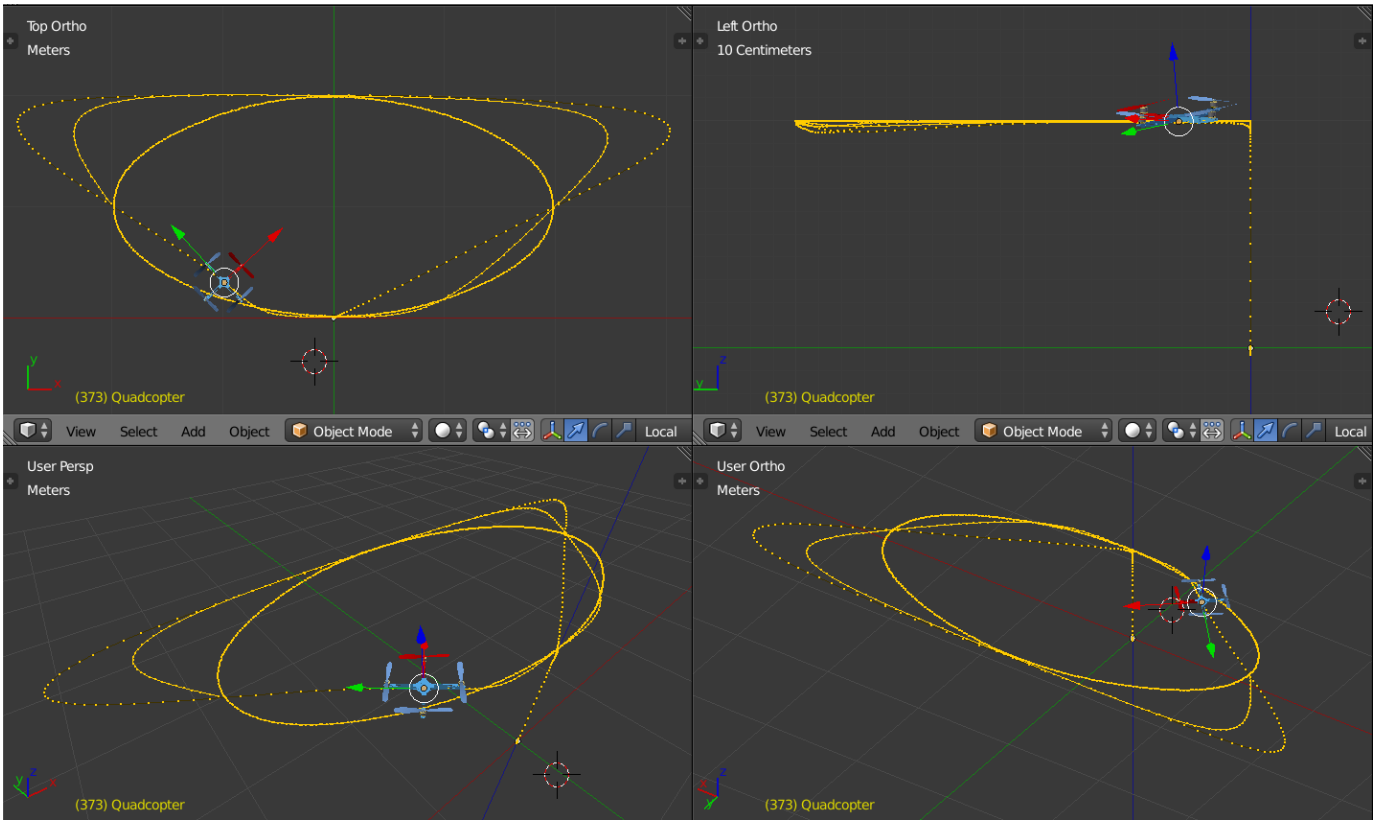


Figure 56: Error and cumulative error position plots for all phases for *tf* = 1.5s, *v* = 3m/s

The attitude controller can track the yaw with decent accuracy and magnitudes of error. For higher velocities, the tracking performance is degraded with higher magnitudes of error, which can be seen from the cumulative error plots as well. But it still maintains the given velocity constraints, but with errors up to 0.5 m/s. If the velocity was increased further, the robot was unable to track the trajectory in the given time and lost stability.



Video 3: Blender simulation for Quadrotor Pirouette
(Direct link: https://drive.google.com/drive/folders/1bP0ahk_oedvzrteOZJsae2vlInPBfaSC?usp=sharing)

Problem 10 - Solution

There were several improvements made to the state machine proposed in Problem 5. The state machine now checks error convergence in all X, Y, Z and velocities before transitioning into the next state. Furthermore, different sets of gains and even different models (PID/LQR/Aggressive) can be individually selected for each state. The implementation is updated in *state_machine.m*.

Another helper script called *mission_planner.m* was created to individually customize the trajectories for the different states to plan various missions.

There were certain other improvements for error handling such as maximum error settling bound, plotting the data for all the states, and saving the simulation data for all the states together. Another improvement, that was made was, once the state trajectory was complete, a tracker ensured that the state was stabilized for a given period, in case the state was distributed or exceeded the error tolerance, the tracker resets and dynamically allows the robot with more time to stabilize within the error tolerance.

Most of the problems solved used this state machine which worked very well. But for the sake of completeness of this problem, following are the plots for every state for a simple hover trajectory.

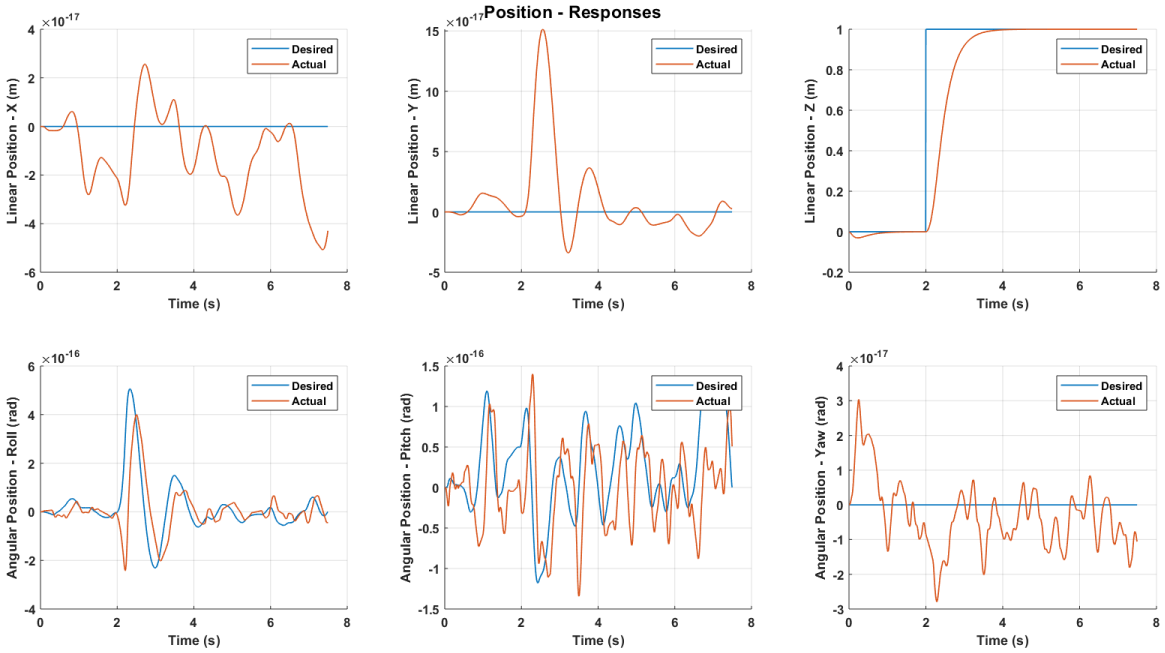


Figure 57: Pose plots for TAKEOFF state

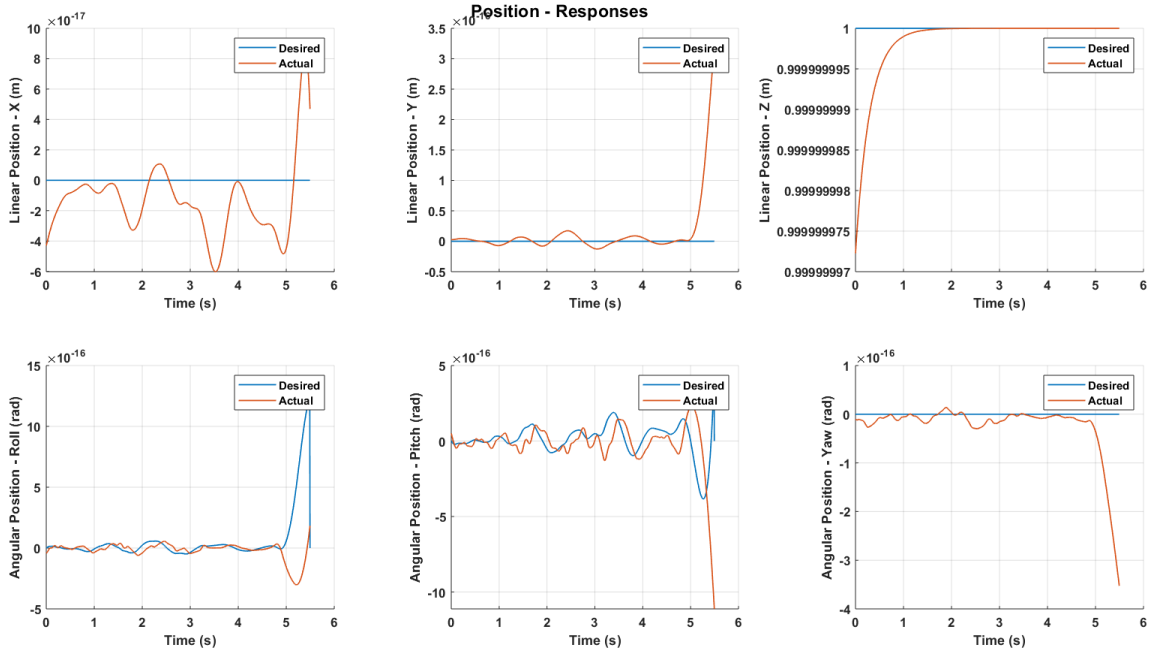


Figure 58: Pose plots for HOVER state

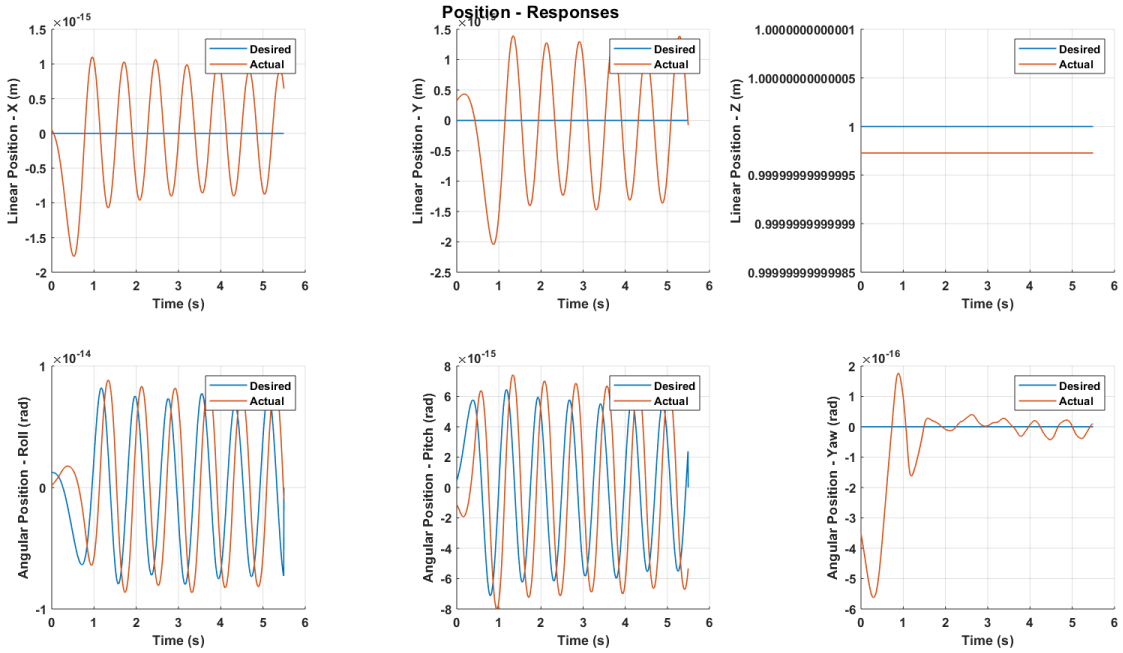


Figure 59: Pose plots for TRACK state

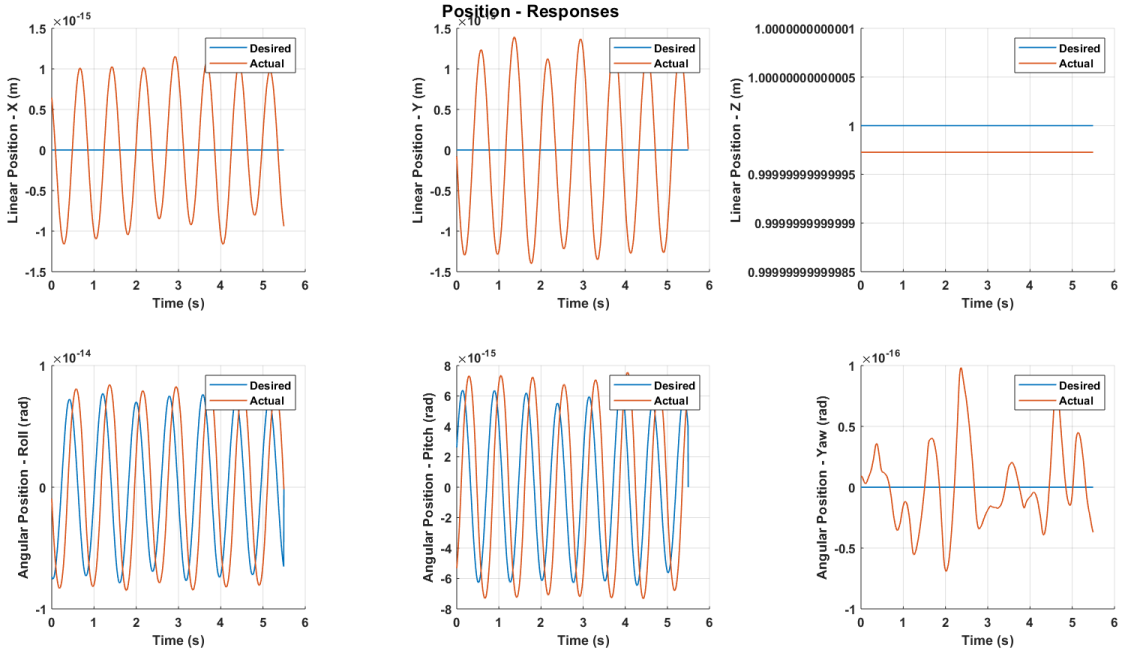


Figure 60: Pose plots for HOVER state

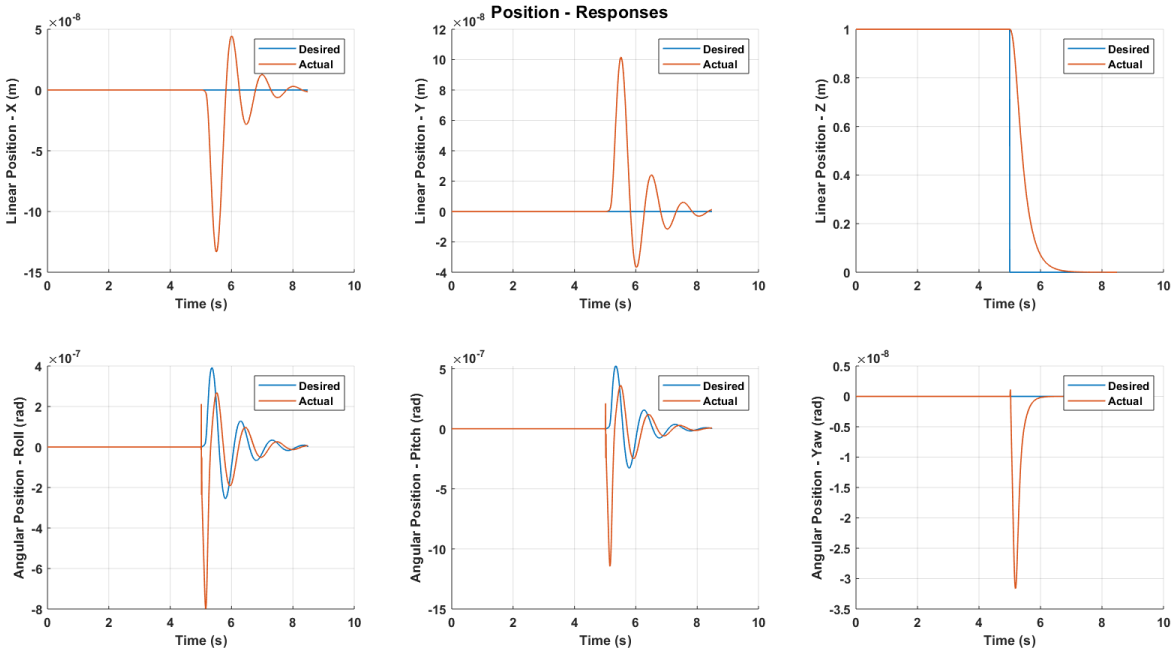


Figure 61: Pose plots for LAND state

Problem 11 – Solution

Non-linear controllers were implemented, and the aggressive model as shown in Problem 1 was used. For free skate, we need to control the quadcopter in Acrobatic mode, where we specify the 3D trajectory including the desired roll and pitch. Several free skate maneuvers were performed with roll rotation up to 328 deg/s.

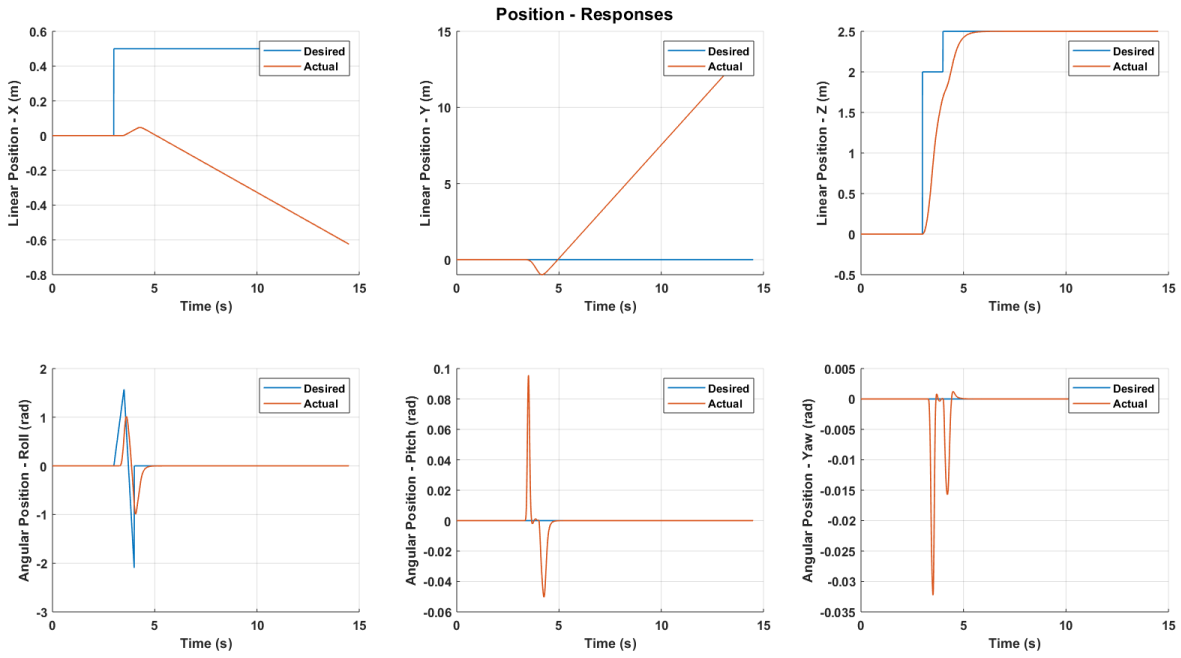


Figure 62: Pose plots for Fly-Through-Ring maneuver

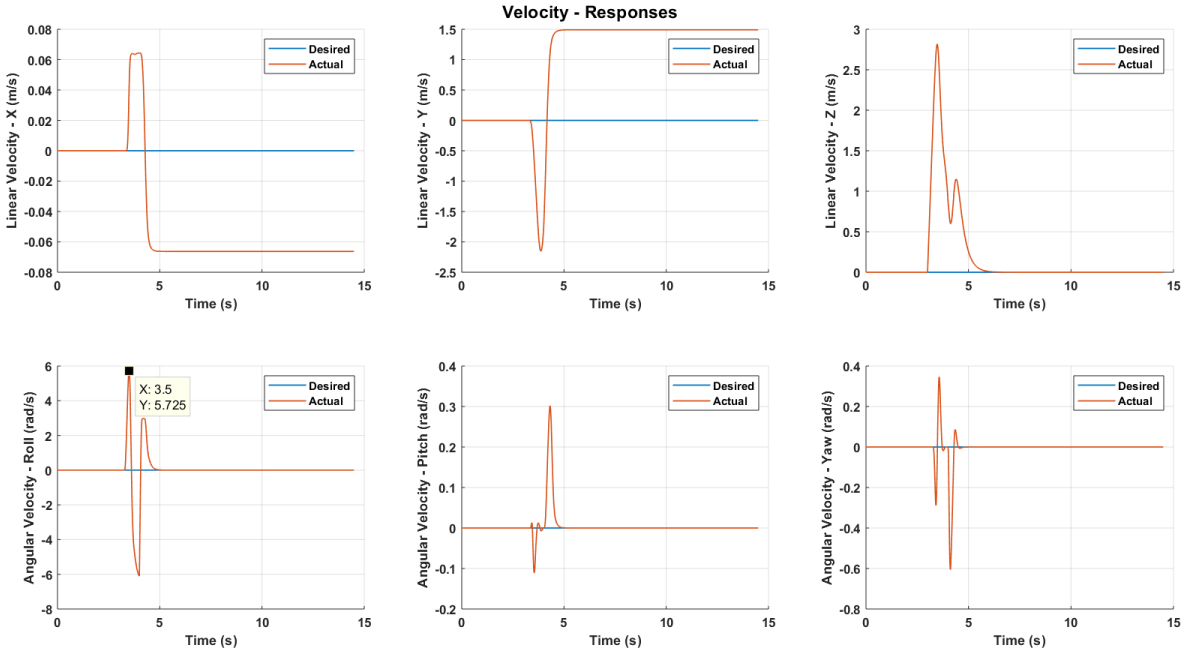
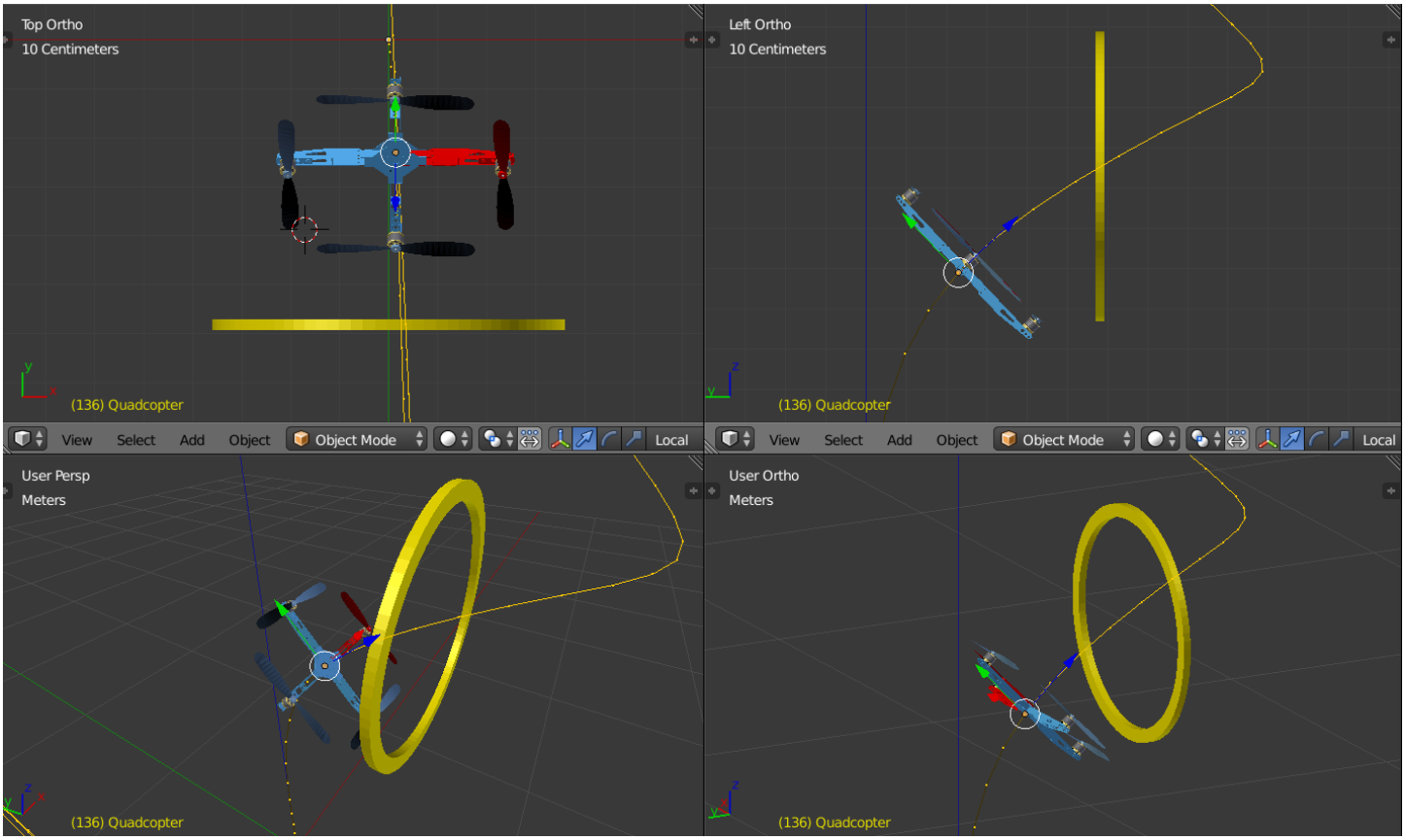


Figure 63: Velocity plots for Fly-Through-Ring maneuver



Video 4: Blender simulation for Fly-Through-Ring maneuver
(Direct link: https://drive.google.com/drive/folders/1bP0ahk_oedvzrteOZJsae2vlInPBfaSC?usp=sharing)

A 360 degree quadcopter flip was attempted, although the trajectory tracking performance is not very great, the results look pretty interesting. Again, roll velocities greater than 300 deg/s were achieved.

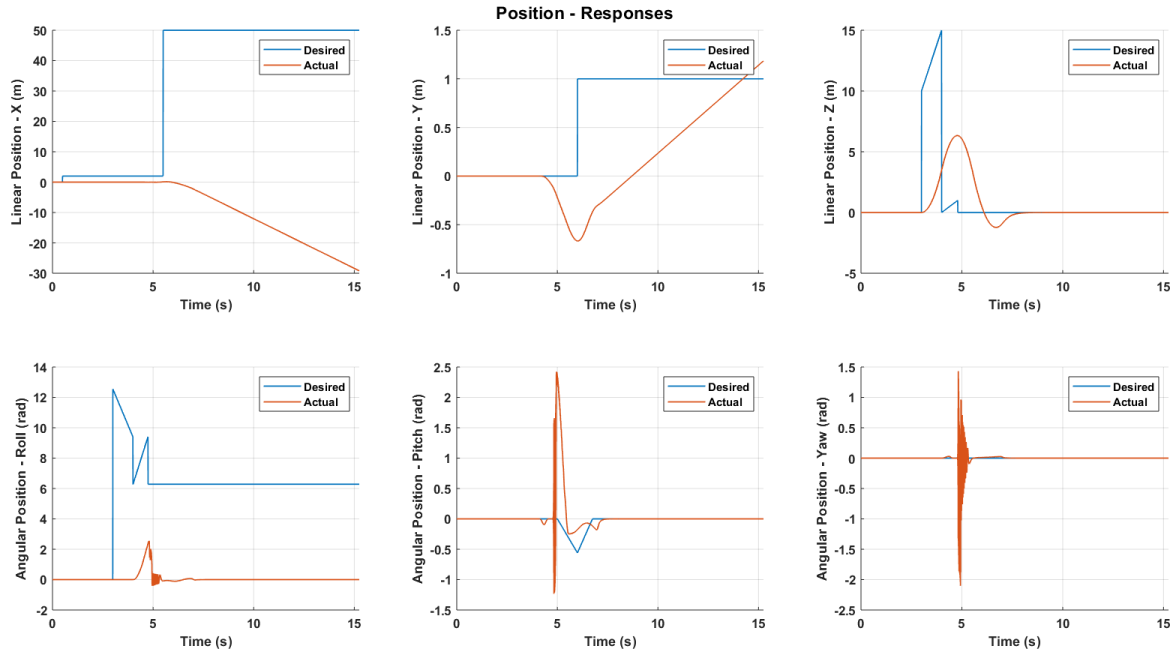


Figure 64: Pose plots for Flip maneuver

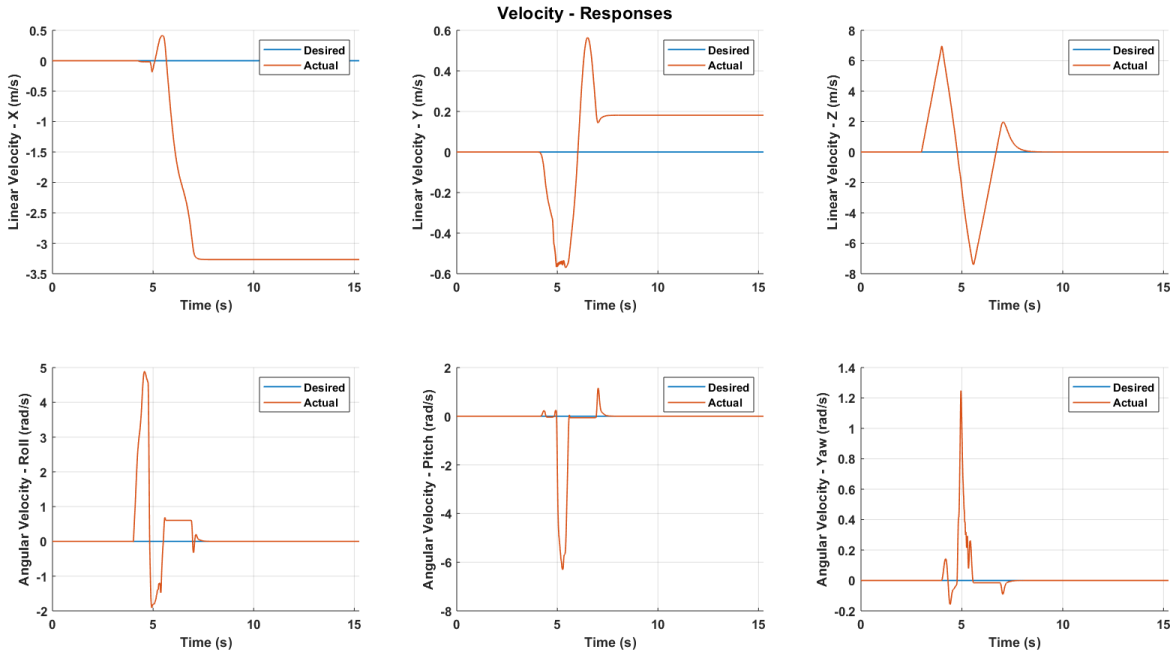
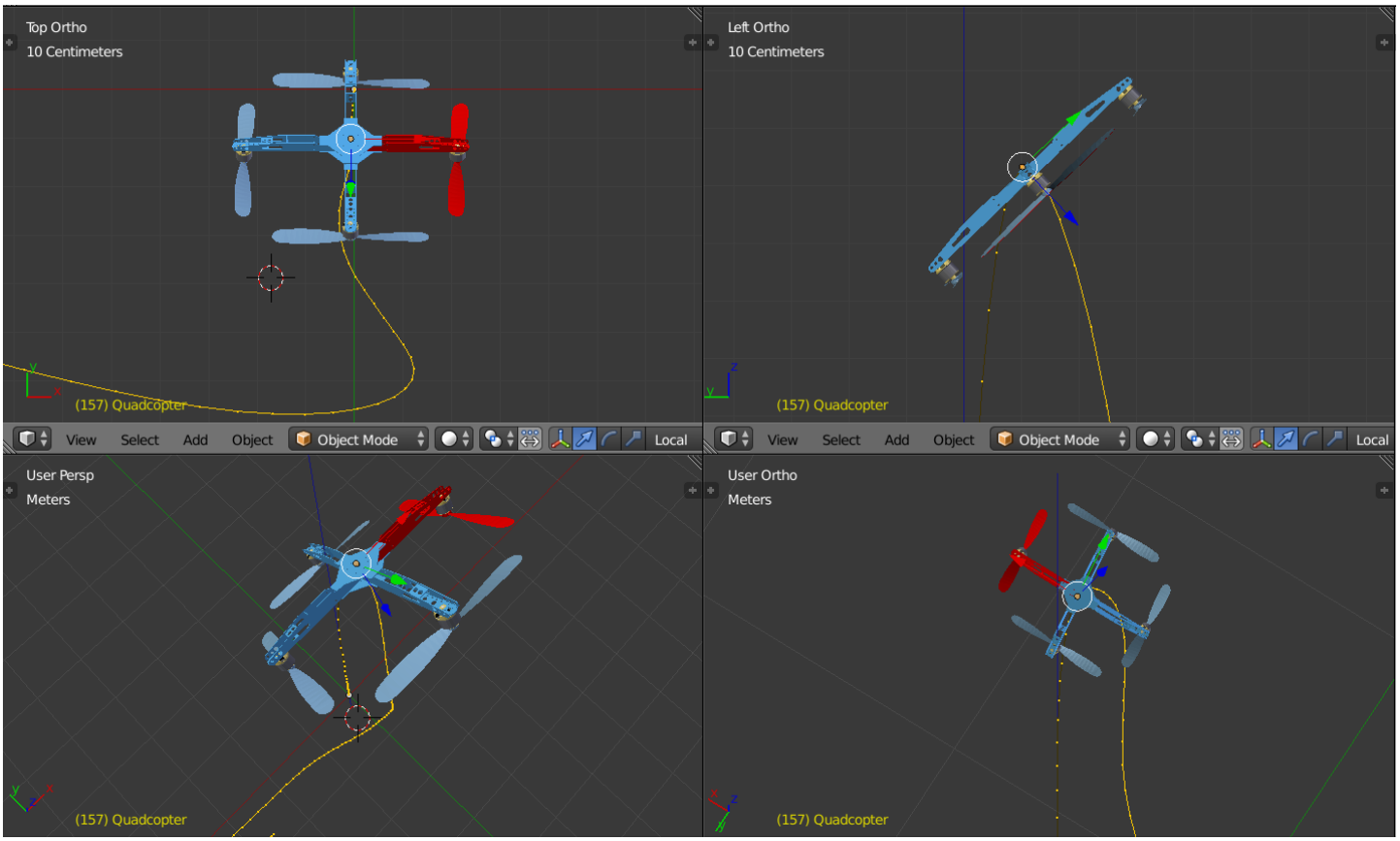


Figure 65: Velocity plots for Flip maneuver



Video 5: Blender simulation for Flip maneuver
(Direct link: https://drive.google.com/drive/folders/1bP0ahk_oedvzrteOZJsae2vlInPBfaSC?usp=sharing)

With increase in pitch speed, the thrust keeps decreasing linearly. The consequence of not modelling drag in our case that the quadcopter keeps losing altitude as aggressive maneuvers in pitch are performed. To counter this problem, I manually gave position waypoints to increase the quadrotor's thrust in order to maintain altitude up to some extent. This can be seen in Figure 64.

References and Sources

1. Nathan Michael - *16-665 Robot Mobility Lectures*, 2018
2. Taeyoung Lee, Melvin Leok, N. Harris McClamroch - *Geometric Tracking Control of a Quadrotor UAV on SE(3)*, 2010
3. Daniel Mellinger and Vijay Kumar - *Minimum Snap Trajectory Generation and Control for Quadrotors*, 2011
4. Vijay Kumar – *Aerial Mobility*, Coursera
5. The quadcopter 3D model was obtained from freeware SketchUp 3D Warehouse and customized.