16-665

Air Mobility

Boxiang Fu

boxiangf

10/31/2024

Part A

Q1.

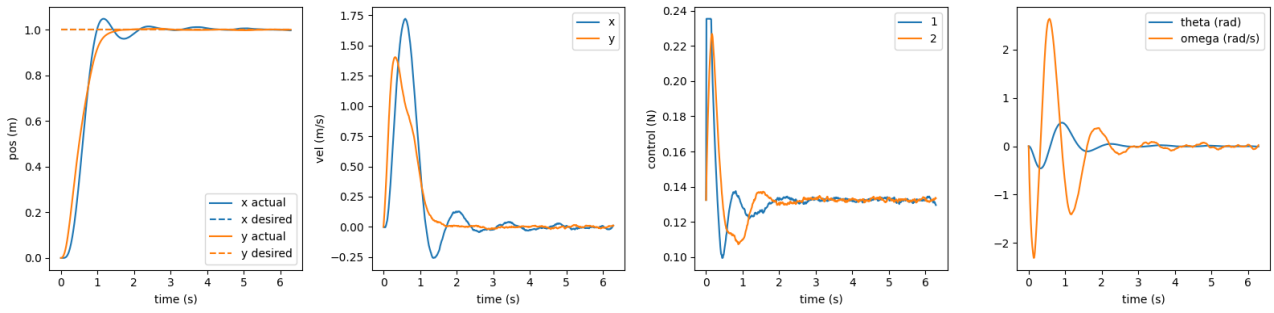
The output obtained from the dynamics function is as follows:

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 1
************************
pos: [0. 0.]
vel: [0.     0.0545]
theta: 0.0
omega: 0.1487801150528891
************************
pos: [-0.00074911  0.02996508]
vel: [-0.03744681  0.59866976]
theta: 0.08182906327908901
omega: 1.6365812655817802
************************
pos: [-0.01354985  0.11368547]
vel: [-0.3000653   1.11769881]
theta: 0.31243824161106704
omega: 3.1243824161106697
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7003/static/
```

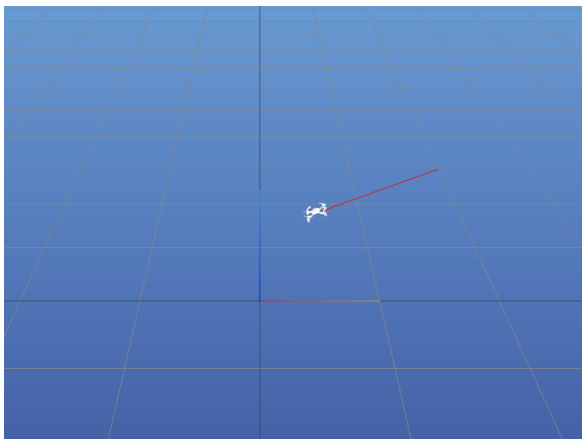
Q2.

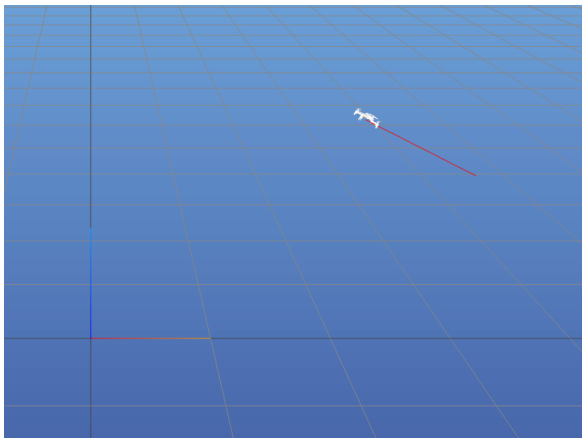
I.

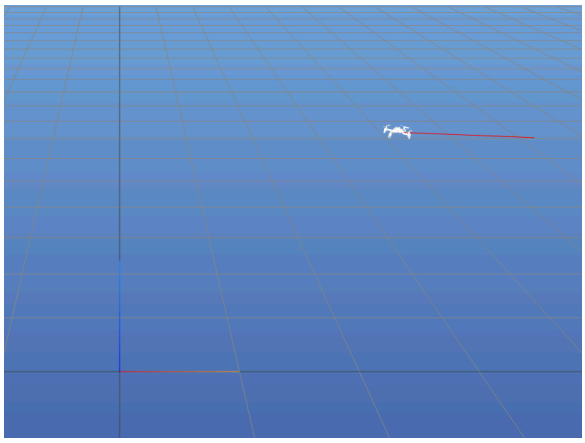
From $[0, 0]$ to $[1, 1]$:



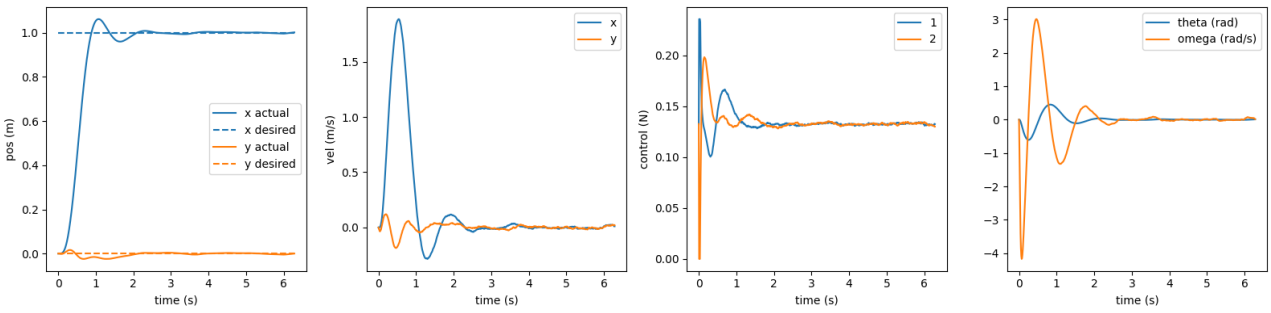
Three screenshots at approx. $t = 0.3\ sec$, $1\ sec$, and $2\ sec$ are sequentially displayed below:



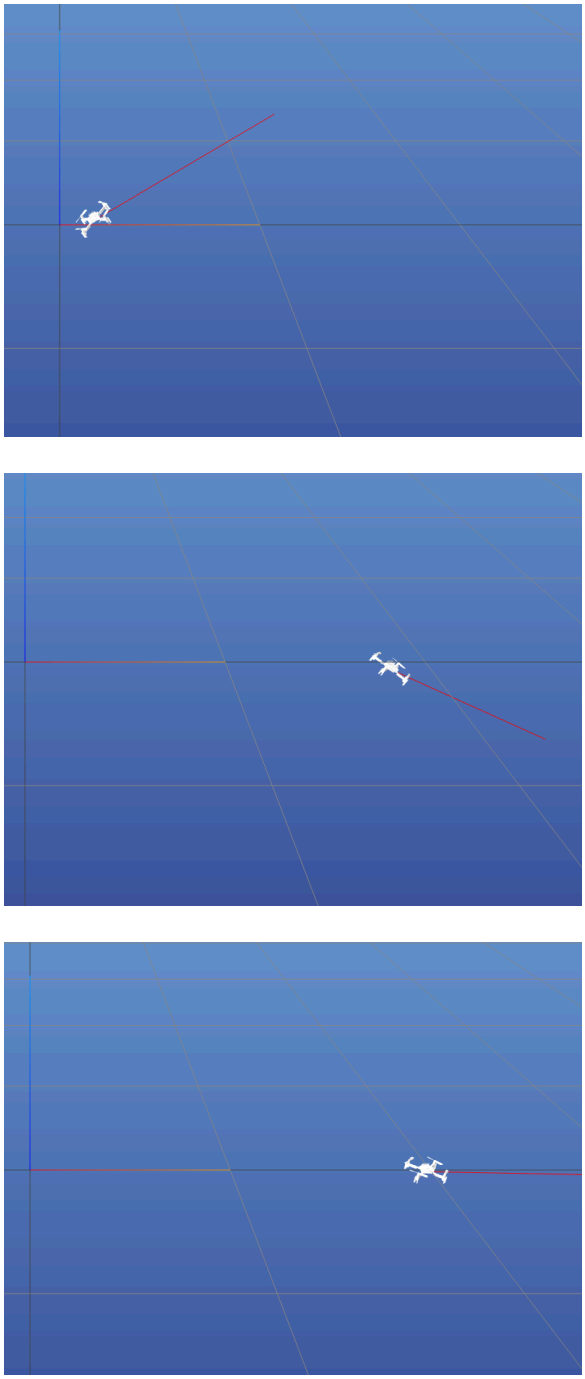




From $[0, 0]$ to $[1, 0]$:


pos (m)
time (s)
x actual
x desired
y actual
y desired
vel (m/s)
x
y
control (N)
1
2
theta (rad)
omega (rad/s)

Three screenshots at approx. $t = 0.3\ sec$, $1\ sec$, and $2\ sec$ are sequentially displayed below:



From the screenshots, both drones yaw to the left initially to move to the right. Once reaching around $x = 1$, both drones yaw to the right to stop. Finally, both drones level out once reaching their desired locations. What is different is that the yaw angle from $[0, 0]$ to $[1, 1]$ is comparatively less than from $[0, 0]$ to $[1, 0]$. This is presumably to also allow a positive upward thrust so that the drone can move from $y = 0$ to $y = 1$. A larger yaw angle would result in less upward thrust and

more sideways thrust, which is helpful in moving sideways (as is the case for moving from $[0, 0]$ to $[1, 0]$) and not so much for moving vertically.

This difference can also be seen on the telemetry figures. As expected, we see that both *x*- and *y*-positions increase for the first case, while only the *x*-position increases for the second case. To achieve this, a positive *x*- and *y*-velocity spike is seen for the first case, while only a positive *x*-velocity spike is seen in the second case.

What is more interesting is that for the first case, both rotors initially provide high thrust so that the drone moves vertically. A small thrust difference is seen so that a torque is applied to yaw the drone to the left and move the drone rightwards. This can be seen in the change in theta and omega values in the right-most figure. For the second case, only one motor is producing significant thrust initially, so that the drone can change its yaw quickly and move rightwards while not moving much vertically. This can be seen in the higher change in theta and omega values in the right-most figure. We see that for both cases, the controls are reversed after approx. 0.5 seconds to slow the drone down as it has reached its setpoint. For both cases, some diminishing oscillatory movements are observed as the drone settles down before theta and omega asymptote around 0 and the thrust from both motors' asymptote around 0.13 Newtons each.

II.

The PD gains I chose were:

$$K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$$

I obtained these PD gains as follows:

1. I initially started with $K_P = 1, K_D = 1, K_{P,\tau} = 10, K_{D,\tau} = 10$. I noticed the response is very slow, so I increased the P gain for the inner loop.

2. Next, I used $K_P = 1, K_D = 1, K_{P,\tau} = 100, K_{D,\tau} = 10$. I noticed the response is now faster, but there is significant overshoot. So, I increased the D gain for more damping.
3. Next, I used $K_P = 1, K_D = 1, K_{P,\tau} = 100, K_{D,\tau} = 30$. There is still significant overshoot, so I increased the PD gains for the outer loop.
4. Next, I used $K_P = 5, K_D = 5, K_{P,\tau} = 100, K_{D,\tau} = 30$. The overshooting is gone, but I wanted to make the response faster. So, I increased the P gain for both the inner and outer loops.
5. Finally, I chose $K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$.

I compared my PD gains against the preliminary PD gains chosen by my classmate Wu Yi of $K_P = 3, K_D = 8, K_{P,\tau} = 50, K_{D,\tau} = 40$. The 90% rise time, maximum overshoot, and average control energy of the system for my PD gains and her PD gains are shown below:

From $[0, 0]$ to $[1, 1]$:

$K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$:

$$Rise\ Time_x = 0.55028\ sec$$

$$Rise\ Time_y = 0.75038\ sec$$

$$Max\ Overshoot_x = 0.03962\ m$$

$$Max\ Overshoot_y = 0.00407\ m$$

$$Average\ Control\ Energy = 0.03646\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 2
Rise time for x is 0.55028, rise time for y is 0.75038
Maximum overshoot in x is 0.03962, maximum overshoot in y is 0.00407
The average control energy is 0.03646
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

$K_P = 3, K_D = 8, K_{P,\tau} = 50, K_{D,\tau} = 40$:

$$Rise\ Time_x = 4.82244\ sec$$

$$Rise\ Time_y = 5.5328\ sec$$

$$Max\ Overshoot_x = 0.09251\ m$$

$$Max\ Overshoot_y = 0.08541\ m$$

$$Average\ Control\ Energy = \ 0.03518\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 2
Rise time for x is 4.82244, rise time for y is 5.5328
Maximum overshoot in x is 0.09251, maximum overshoot in y is 0.08541
The average control energy is 0.03518
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

From $[0, 0]$ to $[1, 0]$:

$K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$:

$$Rise\ Time_x = 0.50025\ sec$$

$$Rise\ Time_y = 0\ sec$$

$$Max\ Overshoot_x = 0.06567\ m$$

$$Max\ Overshoot_y = 0.01804\ m$$

$$Average\ Control\ Energy = \ 0.03629\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 2
Rise time for x is 0.50025, rise time for y is 0.0
Maximum overshoot in x is 0.06567, maximum overshoot in y is 0.01804
The average control energy is 0.03629
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

$K_P = 3, K_D = 8, K_{P,\tau} = 50, K_{D,\tau} = 40$:

$$Rise\ Time_x = 4.91249\ sec$$

$$Rise\ Time_y = 0\ sec$$

$$Max\ Overshoot_x = 0.08788\ m$$

$$Max\ Overshoot_y = 0.00135\ m$$

$$Average\ Control\ Energy = \ 0.03519\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 2
Rise time for x is 5.00254, rise time for y is 0.0
Maximum overshoot in x is 0.08788, maximum overshoot in y is 0.00135
The average control energy is 0.03519
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

At the expense of a slightly larger average control energy expended, we can see that for both setpoints, my PD gains yield a much faster rise time and a mild decrease in maximum overshoot. As such, the performance of my system is comparatively better. The reason for my controller's better performance is presumably due to the more aggressive (higher) values of my PD gains. This causes the control inputs to be higher. As such, if we were to compare performance based on average control energy expended, my PD gains would lose as my drone would need to expend greater control energy to track the trajectory more closely.

Q3.

I.

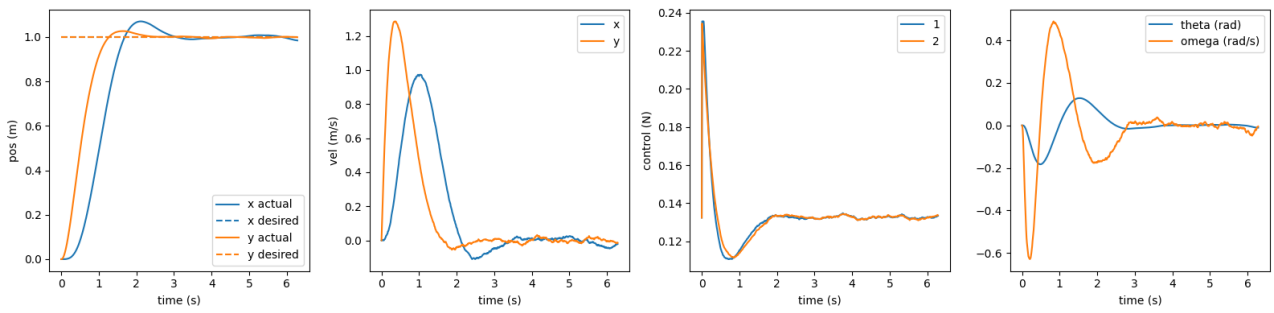
Using the following Q and R:

$$Q = diag(35, 35, 1, 1, 50, 1)$$

$$R = diag(0.5, 0.5)$$

The 90% rise time, maximum overshoot, and average control energy are:

From $[0, 0]$ to $[1, 1]$:



$$Rise\ Time_x = 0.95048\ sec$$

$$Rise\ Time_y = 0.7904\ sec$$

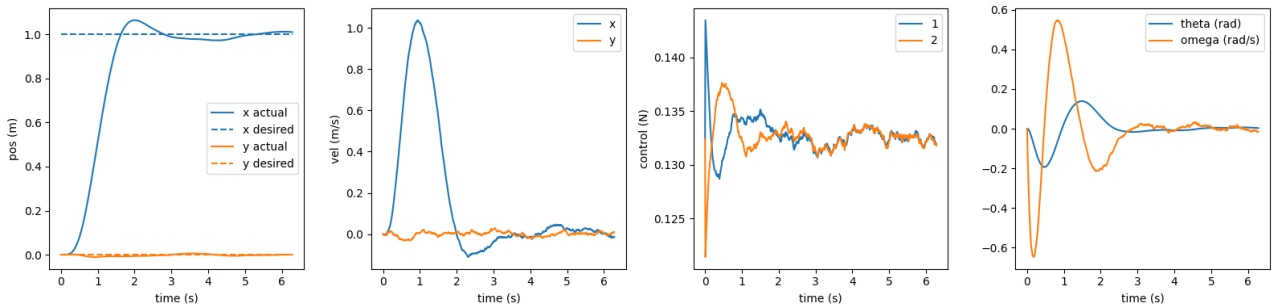
$$Max\ Overshoot_x = 0.07021\ m$$

$$Max\ Overshoot_y = 0.0264\ m$$

$$Average\ Control\ Energy = \ 0.03571\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 3
Rise time for x is 0.95048, rise time for y is 0.7904
Maximum overshoot in x is 0.07021, maximum overshoot in y is 0.0264
The average control energy is 0.03571
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

From $[0, 0]$ to $[1, 0]$:



$$Rise\ Time_x = 0.90046\ sec$$

$$Rise\ Time_y = 0\ sec$$

$$Max\ Overshoot_x = 0.06506\ m$$

$$Max\ Overshoot_y = 0.00573\ m$$

$$Average\ Control\ Energy = \ 0.03524\ N^2$$

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 3
Rise time for x is 0.90046, rise time for y is 0.0
Maximum overshoot in x is 0.06506, maximum overshoot in y is 0.00573
The average control energy is 0.03524
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

Comparing this to the values from Q2, cascaded control tends to have a lower rise time (in both the *x*- and *y*-directions) and lower maximum overshoot. However, this is at the expense of a higher average control energy compared to LQR. A possible explanation for this is that the PD controller

used for cascaded control does not have a cap for the input controls (and thus is only capped by the physical control limit of $0.024g$) whereas the R matrix limits the input controls. This results in lower input controls for the LQR (as can be seen from the lower spikes in the 3rd figure to the left in the telemetry figures) and thus a slower rise time and less average control energy.

II.

The Q and R chosen were:

$$Q = diag(35, 35, 1, 1, 50, 1)$$

$$R = diag(0.5, 0.5)$$

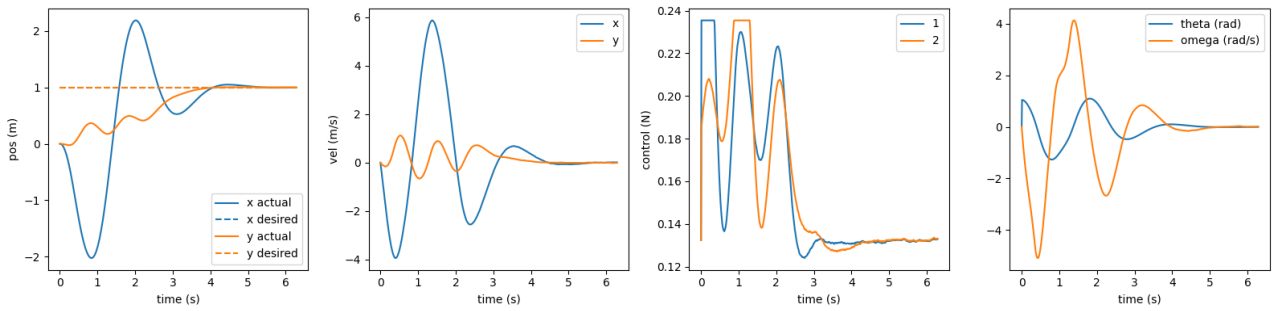
Rescaling the R matrix by 0.1, we have $R = diag(0.05, 0.05)$. We notice that for both the setpoints $[1, 1]$ and $[1, 0]$, the input controls spike higher. This results in a lower rise time for both x and y and a slightly lower maximum overshoot. However, this is at the expense of a higher average control energy. Rescaling the R matrix by 10, we have $R = diag(5, 5)$. We notice that for both the setpoints $[1, 1]$ and $[1, 0]$, the input controls spike lower. This results in a higher rise time for both x and y and a slightly higher maximum overshoot. The average control energy is also lower. This is to be expected since the R matrix determines the penalty for using input controls. A R matrix with large entries imposes a high penalty for using input controls, and *vice versa*. As such, we expect to see an inverse relationship between R matrix entries and control energy, and a positive relationship with rise time.

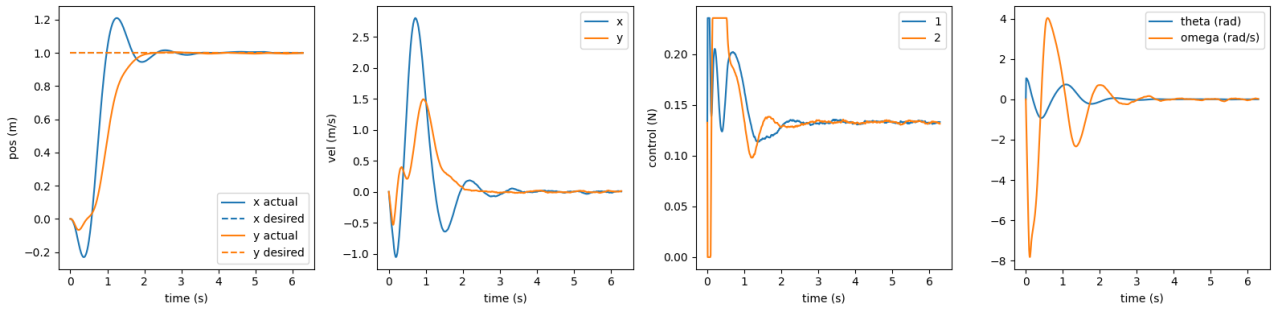
Changing the ratio between pos/vel from 35 to 350, we notice that the drone reaches the setpoint much quicker and at a much faster velocity. This is at the expense of using higher control energy. Changing to 3.5, we see that the velocity is much slower, but the drone also reaches the setpoint slowly. The control energy used is also less. We see that this is the case as the ratio between pos/vel determines how much is penalized when position or velocity differs from its setpoint value. A

higher ratio penalizes positional differences more, so the drone moves fast to obtain the setpoint position. A lower ratio penalizes the velocity difference more, so the drone moves at a slow velocity (the setpoint velocity is 0) so that the penalty is lower.

Changing the ratio between theta/omega from 50 to 500, we note that the yaw angle (theta) takes on smaller values. When changing the ratio to 5, we note that the yaw angle (theta) takes on larger values. Again, this is to be expected as a higher ratio penalizes angular differences more than angular velocity differences, and *vice versa*. This ratio does not seem to have much effect on the drone's *y*-position and velocity. However, the drone's *x*-position tends to converge slower (and at a lower velocity) for larger ratios since the smaller yaw angle means most of the thrust is projected upwards and not sidewards.

When the drone is initialized with a large angle ($\theta = \pi/3$), the LQR performs poorly when compared to cascaded control. This is because the LQR drone will first fly to the left before correcting its course and start flying to the setpoint. Not much emphasis is put on correcting the yaw angle as position control and altitude control is not distinguished. The cascaded control drone performs much better since the inner loop (altitude control) is much quicker in adjusting the yaw angle, so the drone only flies a bit to the left before the yaw angle is correctly adjusted to face the direction of the setpoint. The drone then follows a much linear trajectory to reach the setpoint. The figures for the LQR drone and cascaded control drone are respectively displayed below:

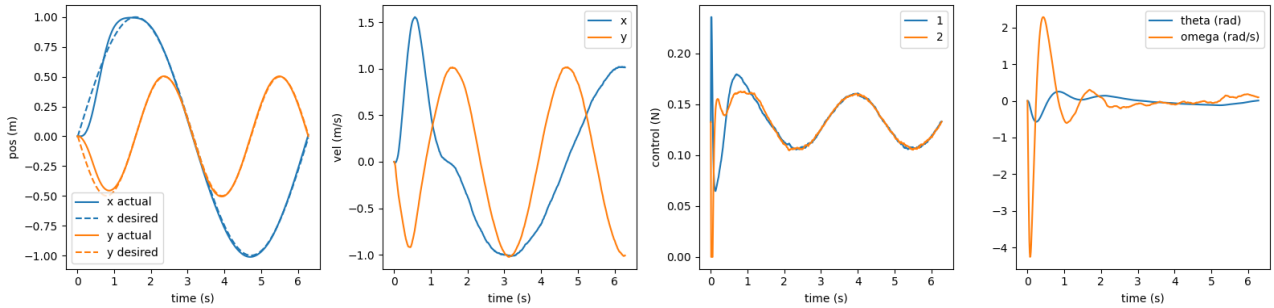




Q4.

I.

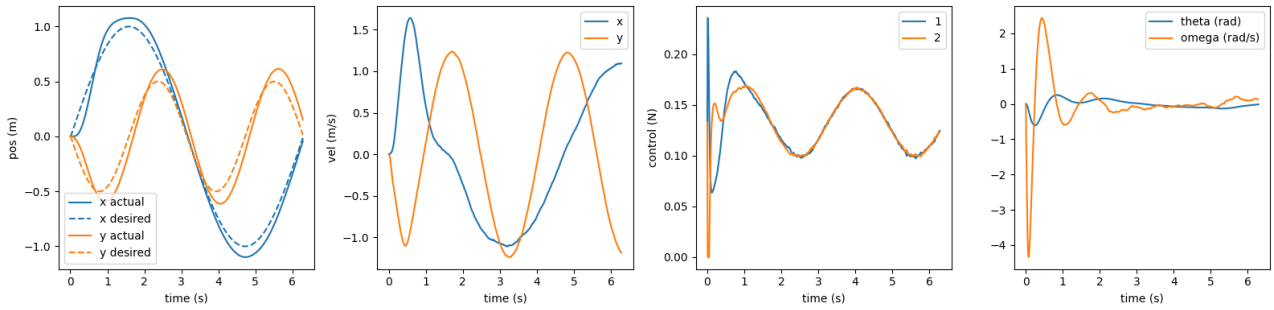
Using the same PD gains as Q2 ($K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$), the RMSE is 0.0471 meters, and the telemetry figure is as follows:



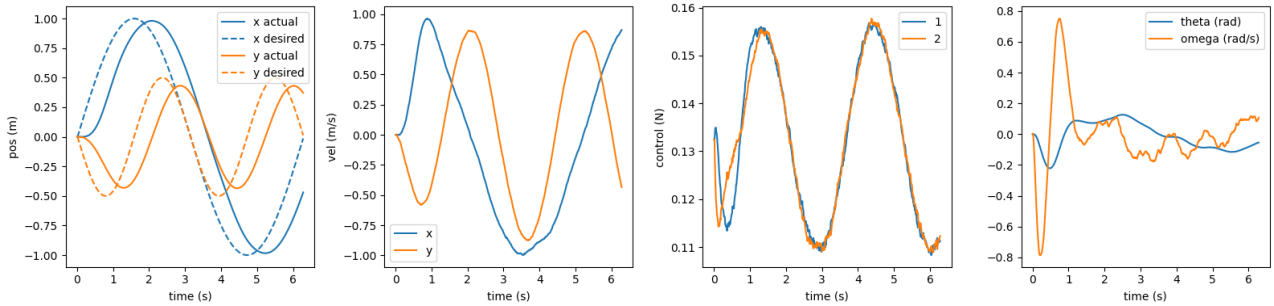

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python 2d.py 4
RMSE: 0.0471
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

II.

When we removed the acceleration feedforward term a_d, we notice that the RMSE increased from 0.0471 meters to 0.10651 meters, and an increased error appearing in the telemetry figure during turning (see figure below). The reason for the error increase during turning is because turning is a period of acceleration. The acceleration feedforward term tries to minimize the error during the acceleration phase. Removing the acceleration feedforward term means that the control input no longer tries to minimize the acceleration error. As such, an increased error arises during turning and increases RMSE.



When we further set $v_d = 0$ in addition to removing a_d, we obtain an even higher RMSE of 0.30905 meters. In addition to the error during the turning phase, the error is now also present in the constant velocity phase of the drone's kinematics (see figure below). Again, the reason for this is because the velocity feedforward term tries to minimize the error during the constant velocity phase. Setting $v_d = 0$ means that the control input no longer tries to minimize the constant velocity error. As such, an increased error arises during periods of constant velocity and increases RMSE even higher.

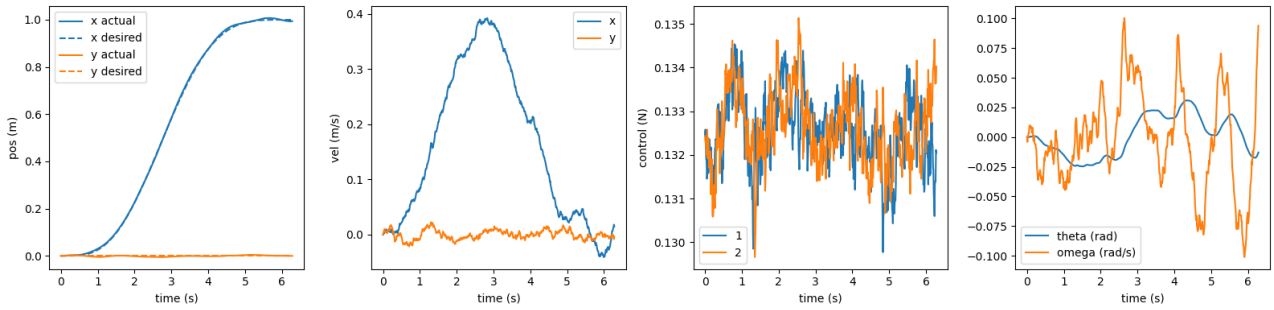


Q5.

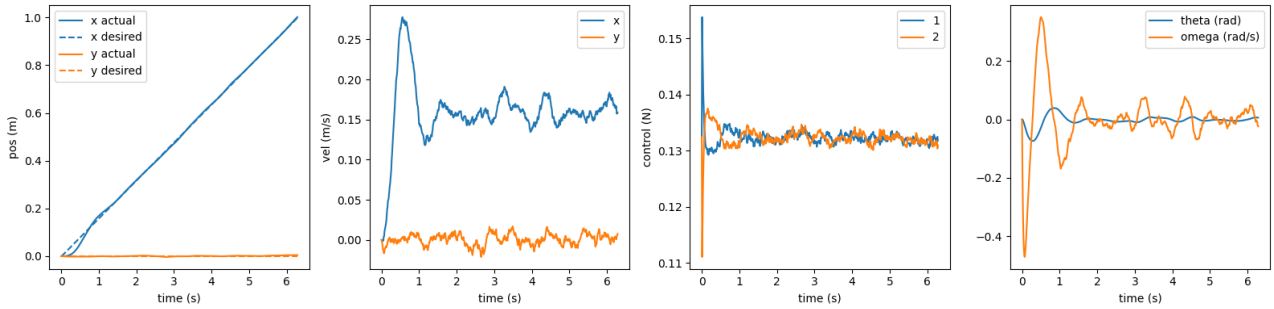
I.

For $total\_time = 2\pi$ seconds:

Using the polynomial trajectory from differential flatness, we obtain a RMSE of 0.00344 meters. Its figure is displayed below:

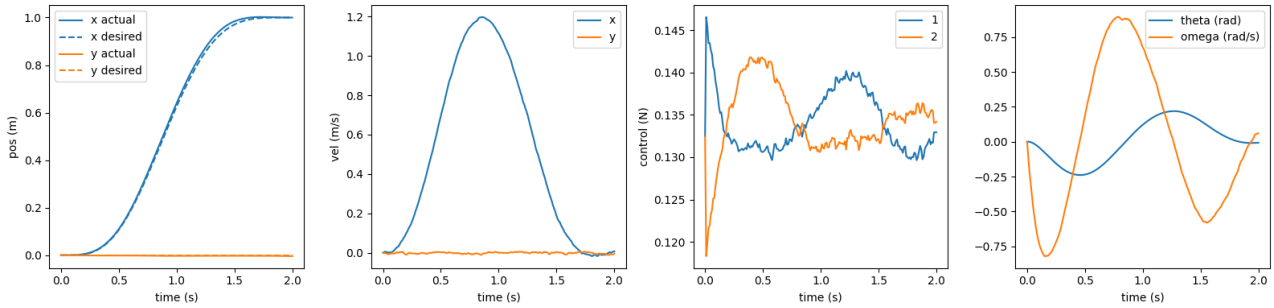


Using the naïve trajectory with constant velocity, we obtain a RMSE of 0.00634 meters. Its figure is displayed below:

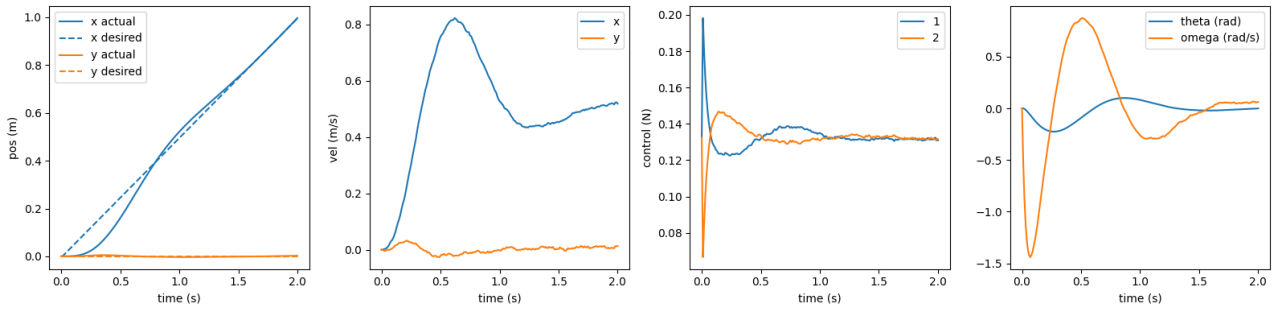


For $total\_time = 2$ seconds:

Using the trajectory from differential flatness, we obtain a RMSE of 0.00727 meters. Its figure is displayed below:



Using the naïve trajectory with constant velocity, we obtain a RMSE of 0.03305 meters. Its figure is displayed below:



From the above figures, we can see that for $total\_time = 2\pi$ seconds, the RMSE for the polynomial trajectory is about 50% less than the RMSE for the linear trajectory, while the polynomial trajectory RMSE is about 80% less than its linear counterpart for $total\_time = 2$ seconds. This is because the trajectory from differential flatness can be smoothly tracked by the drone (since it gives a 1-to-1 correspondence between $(\theta, \omega, T, \tau)$-space and (p, v, a, j, s)-space), whereas the linear trajectory does not consider of the limitations in $(\theta, \omega, T, \tau)$-space and may not be physically trackable. An example would be that the linear trajectory expects the drone to instantaneously move at a constant velocity at $t = 0$, which is physically impossible as it would require infinite acceleration.

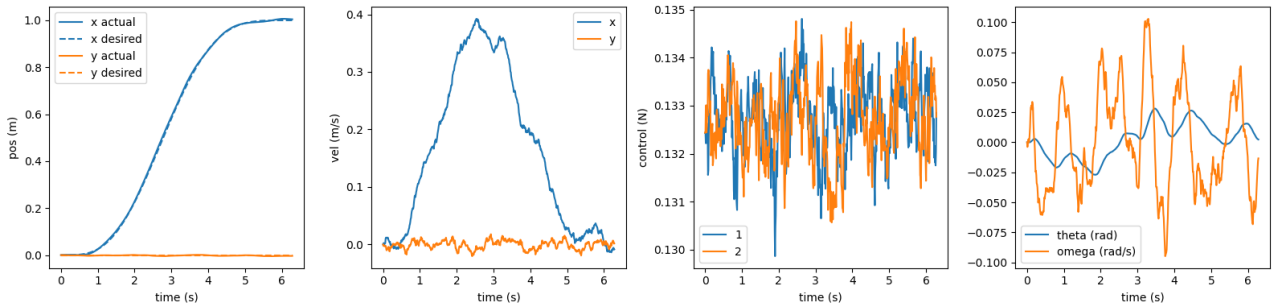
Additionally, the x-velocity of the polynomial trajectory ramps up and ramps down parabolically, while there is a huge spike at the start for the linear trajectory. The drone stops at the setpoint $[1, 0]$ for the polynomial trajectory (since both x- and y-velocities are 0) whereas the drone continues in the x-direction after reaching setpoint $[1, 0]$ for the linear trajectory. There are also less drastic changes in input controls for the polynomial trajectory compared to the linear trajectory, where a spike in input controls can be observed at $t = 0$ seconds.

II.

Setting $\omega_d = 0$ and $\tau_d = 0$.

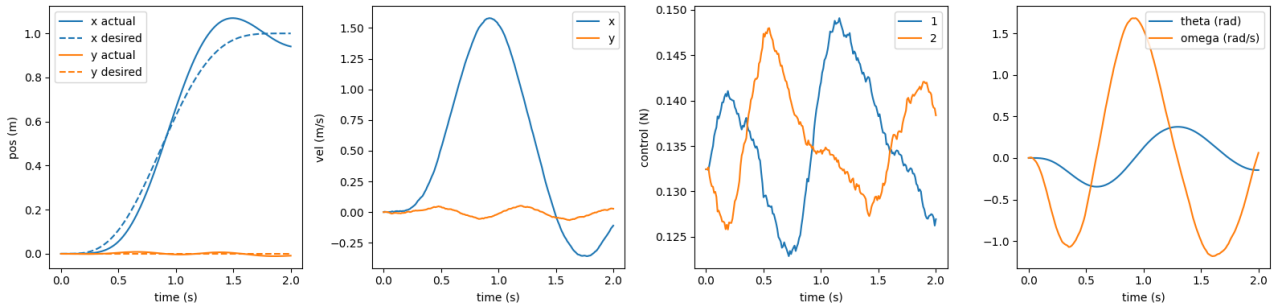
For $total\_time = 2\pi$ seconds:

Using the trajectory from differential flatness, we obtain a RMSE of 0.00408 meters (compared to 0.00344 meters previously). Its figure is displayed below:



For $total\_time = 2$ seconds:

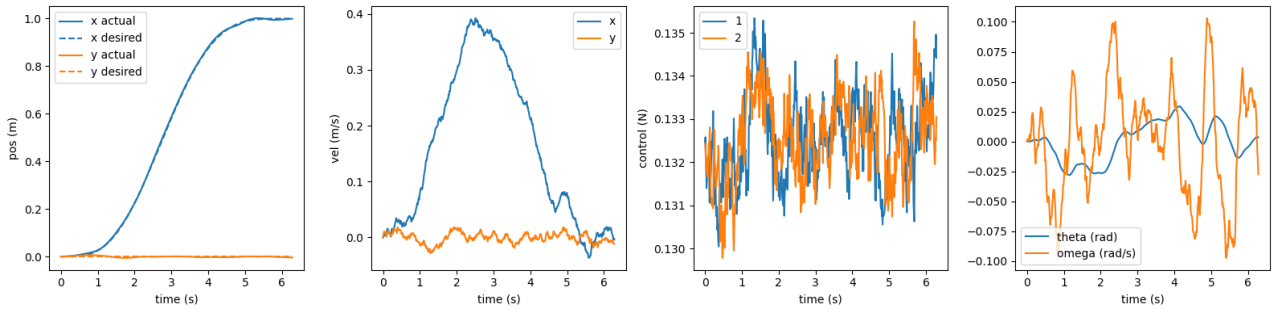
Using the trajectory from differential flatness, we obtain a RMSE of 0.04434 meters (compared to 0.00727 meters previously). Its figure is displayed below:



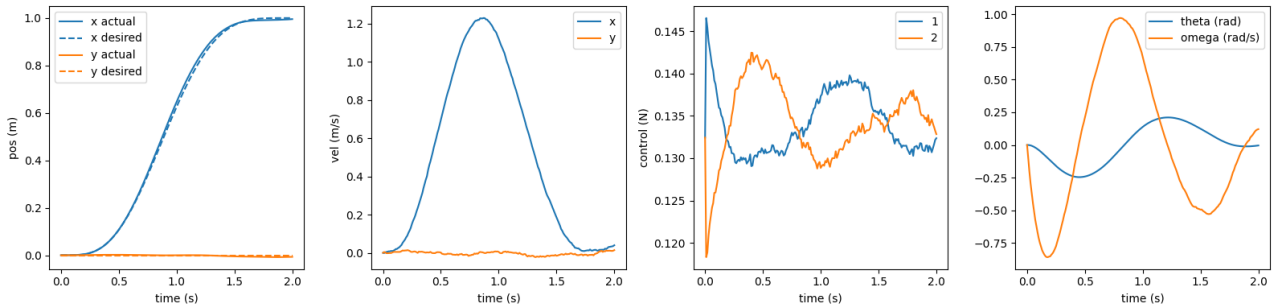
From the RMSE and figures above, we can see that for $total\_time = 2\pi$ seconds, there does not seem to be too much of a difference. This is presumably because the drone is travelling slowly over a longer time horizon, jerk and snap are insignificant enough to be safely ignored. So, setting the desired jerk and snap to zero does not affect the drone dynamics in any significant way to cause large tracking errors. This is not the case for the shorter time horizon of $total\_time = 2$ seconds. The tracking error is now much worse compared to the previous section. The drone is travelling much faster now, so the effects of jerk and snap are now considerably larger and cannot be ignored. If we naively set the desired jerk and snap to zero (as is done for this section), the precision of the inner loop (altitude control) is reduced, thus resulting in larger tracking errors and a larger RMSE.

III.

For $total\_time = 2\pi$ seconds, using the open-loop system (ω_r and τ_r), we obtain a RMSE of 0.00342 meters. Its figure is displayed below:



For $total\_time = 2$ seconds, using the open-loop system (ω_r and τ_r), we obtain a RMSE of 0.00785 meters. Its figure is displayed below:

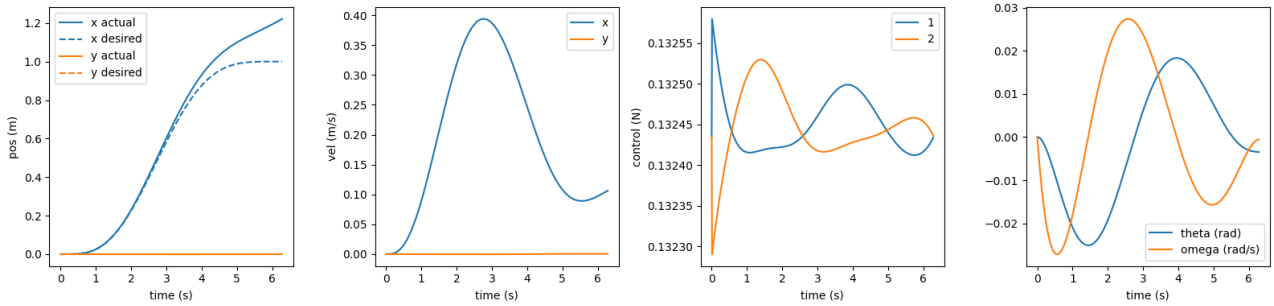


We notice that for the $total\_time = 2\pi$ seconds system, there is no noticeable change. The RMSE are roughly similar, presumably due to the drone's lower velocity not being affected too much by the disturbances. However, for the $total\_time = 2$ seconds system, the closed-loop system obtains a lower RMSE. This is expected since the closed-loop system considers the current state of the drone that is perturbed and adjusts based on this, whereas the open-loop system only adjusts based on an estimated pose and is inaccurate when there are disturbances.

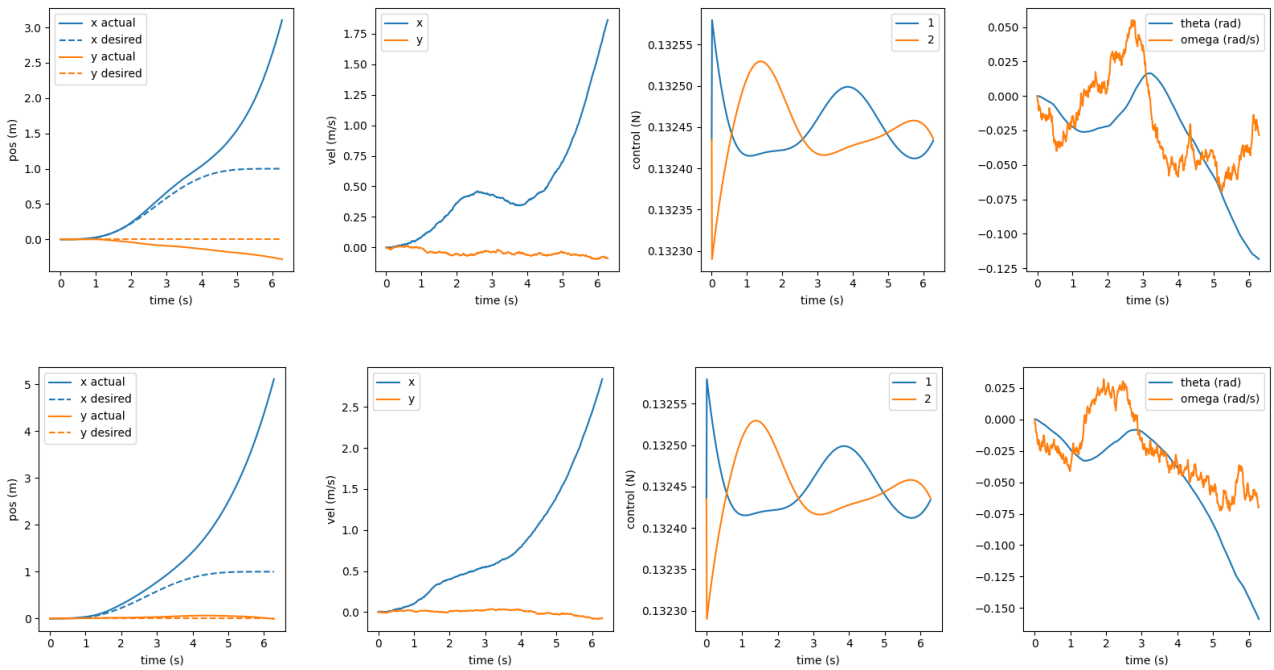
However, if the system disturbance is too large (e.g. setting $self.sigma\_t = 2$ and $self.sigma\_r = 2$), then the closed-loop system tends to have larger RMSE compared to the open-loop system. This is because the closed-loop system also needs to consider the current state of the

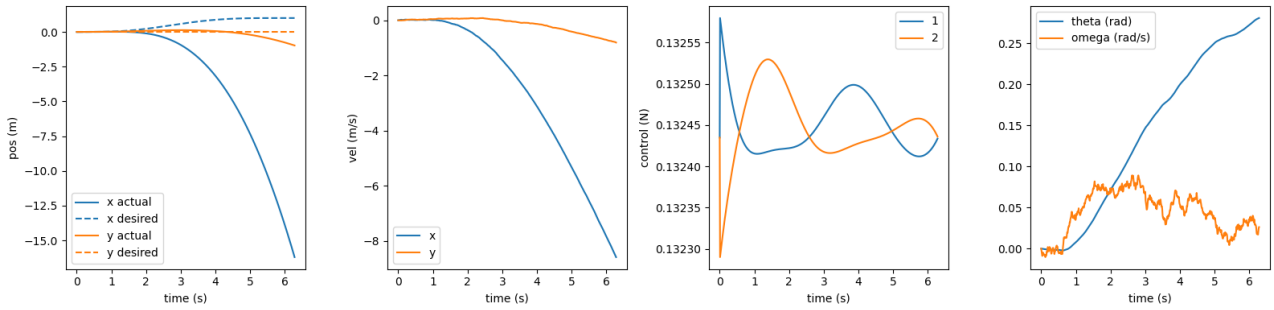
drone. If there is a large randomness in the state of the drone, the controller may over/under-compensate and cause the drone to leave its desired trajectory, thereby causing a larger RMSE. The open-loop system is only a function of time and does not consider the current drone state, so the randomness in the state does not cause the controller to over/under-compensate under large perturbations. To generalize, if the disturbance is small, the closed-loop system works better. If the disturbance is exceedingly large, setting open-loop system ω_r and τ_r might be better.

Using directly T_r and τ_r, we obtain the following without system randomness:



Using directly T_r and τ_r, we obtain the following with default system randomness (we show 3 telemetry figures since the results are quite stochastic):





First off, we see that even without system randomness, the drone does not even converge to the desired setpoint. Instead of slowing down and stopping at the setpoint, the drone continues to travel in the *x*-direction. Furthermore, the trajectory of the drone with system randomness is highly stochastic. Rerunning the algorithm will yield different trajectories, and none of them converge to the setpoint. Again, this is to be expected for an open-loop system. Looking at the control input figure, all 4 runs have the same input. Therefore, any disturbances to the drone would cause its trajectory to deviate and errors to compound. There is no way for the drone to correct these errors as all trajectory and control planning is done beforehand on the reference trajectory and not on the actual trajectory the drone took due to disturbance.

Part B.

Q1.

The output obtained from the dynamics function is as follows:

```
(base) williamfu@Williams-MacBook-Pro aerial_mobility_release % python quadrotor.py 1
**************************
pos: [0. 0. 0.]
vel: [0.     0.     0.0327]
quaternion: [1. 0. 0. 0.]
omega: [2.97560230e-01 5.01040677e-17 1.54269014e-02]
**************************
pos: [ 2.40553113e-06 -1.28359065e-03  1.79510018e-02]
vel: [ 2.05606214e-04 -6.41192722e-02  3.56855592e-01]
quaternion: [9.96644885e-01 8.17371282e-02 1.46189822e-04 4.24186640e-03]
omega: [3.27313302 0.0136595  0.16969186]
**************************
pos: [ 0.00019429 -0.02303064  0.06606604]
vel: [ 0.00683764 -0.50468237  0.595935  ]
quaternion: [0.95145924 0.30734012 0.00232097 0.01618546]
omega: [6.24783997 0.101806   0.32383767]
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7005/static/
```

Q2.

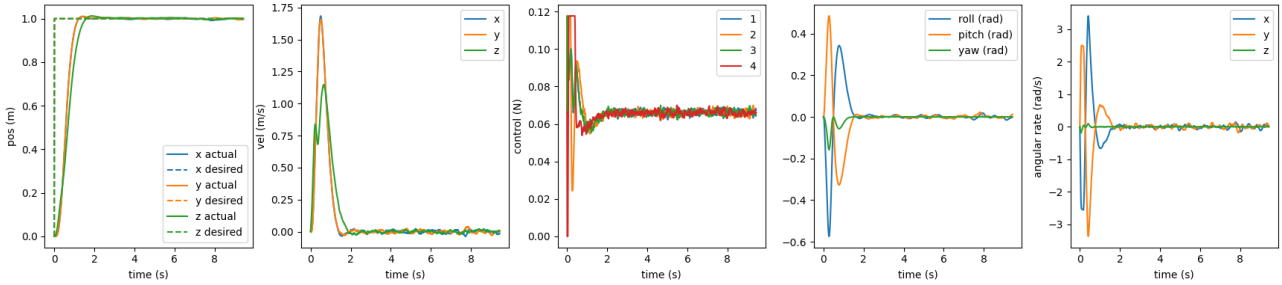
I.

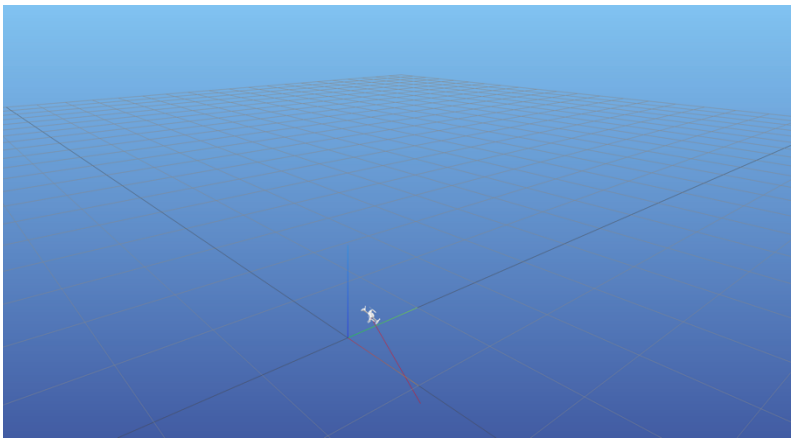
The PD gains used are $K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$ (the same gains used for Part A also works surprisingly well here). For the setpoint $[1, 1, 1]$ and $\psi_d = 0$, we obtain the following:

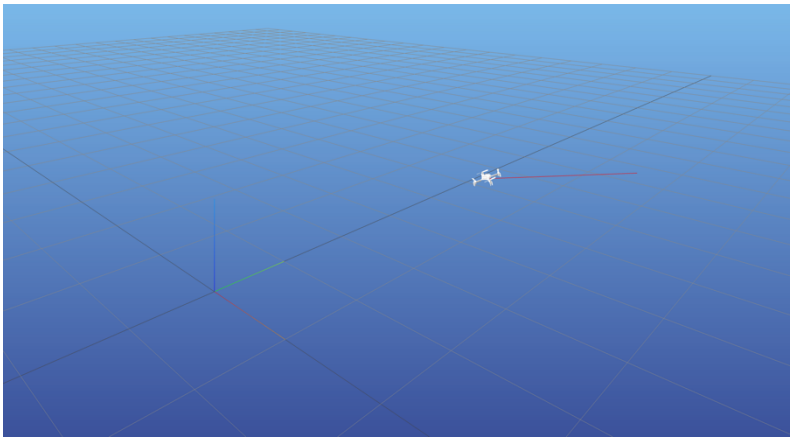
$$Average\ Control\ Energy = \ 0.01817\ N^2$$

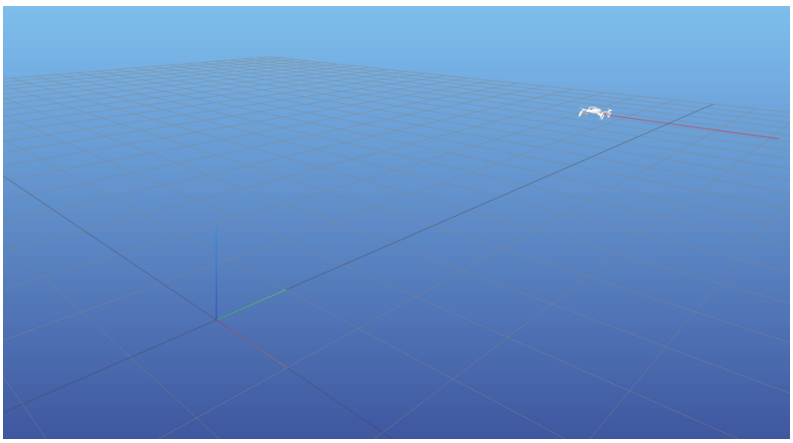
$$RMSE = 0.22326\ meters$$

The telemetry figures and meshcat animation (at approx. $t = 0.3\ sec$, $1\ sec$, and $2\ sec$) are sequentially displayed below:







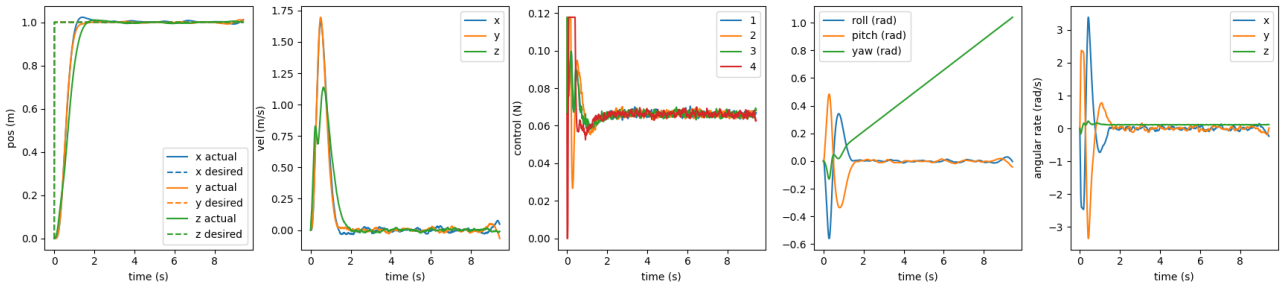


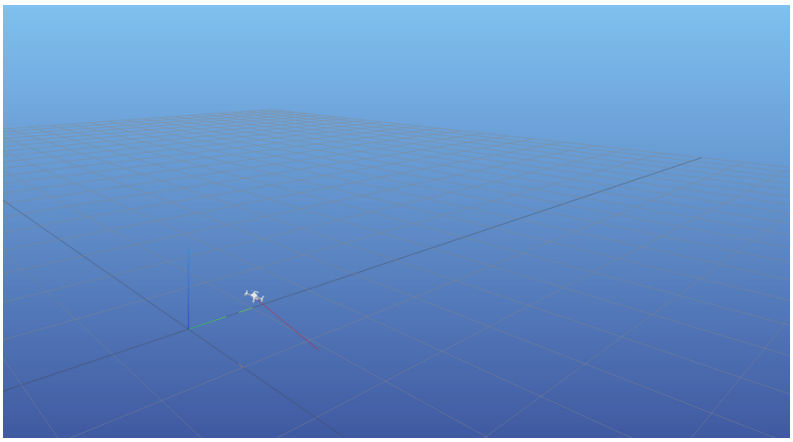
For the setpoint $[1, 1, 1]$ and $\psi_d = \frac{t}{total\ time} \times \frac{\pi}{3}$, we obtain the following:

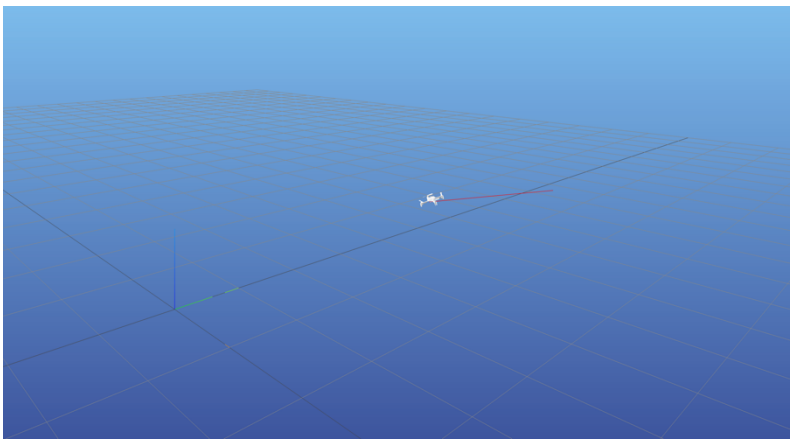
$$Average\ Control\ Energy = \ 0.01827\ N^2$$

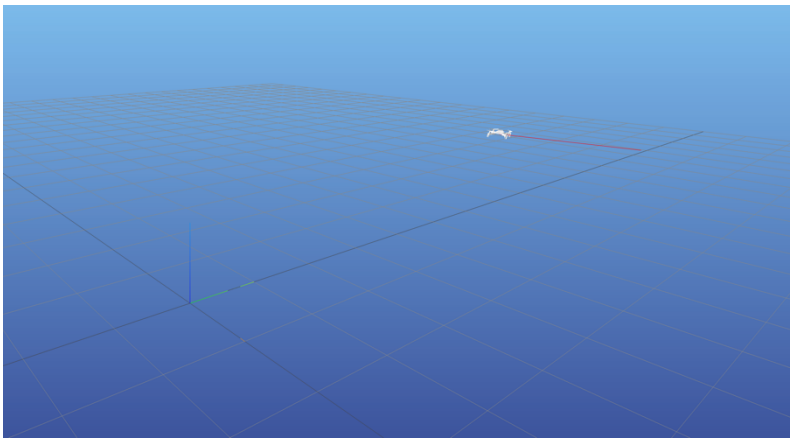
$$RMSE = 0.22453\ meters$$

The telemetry figures and meshcat animation (at approx. $t = 0.3\ sec$, $1\ sec$, and $2\ sec$) are sequentially displayed below:

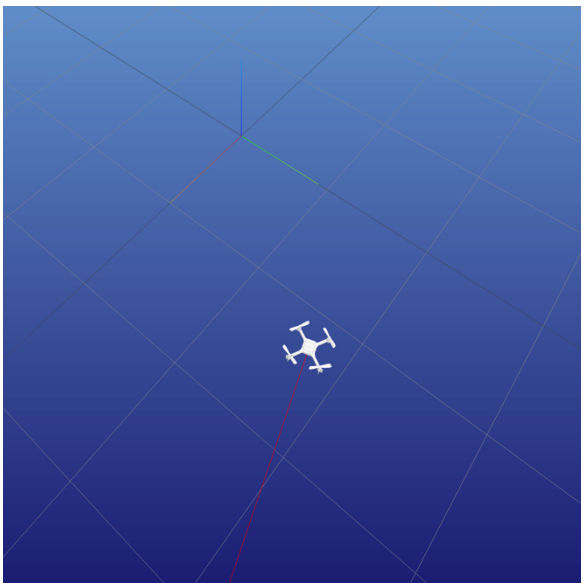


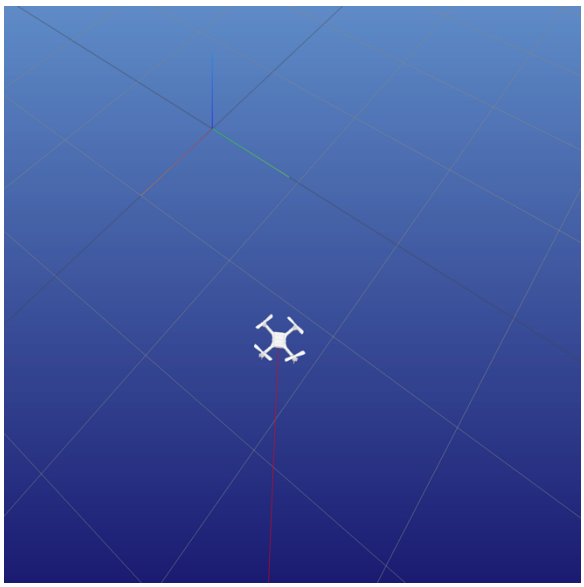






Also note the yaw angle rotation dictated by $\psi_d = \frac{t}{total\ time} \times \frac{\pi}{3}$:



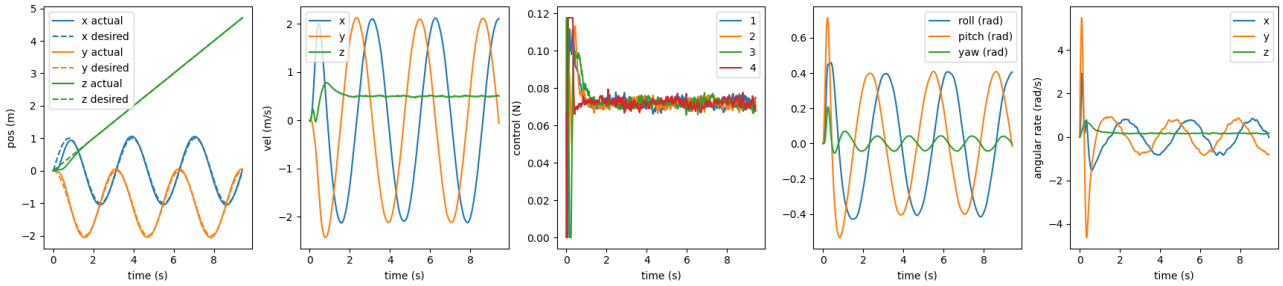


Comparing the RMSE and average control energy of the setpoint tracking control with and without a time-variant yaw angle, we see that on average, both the RMSE and average control energy is

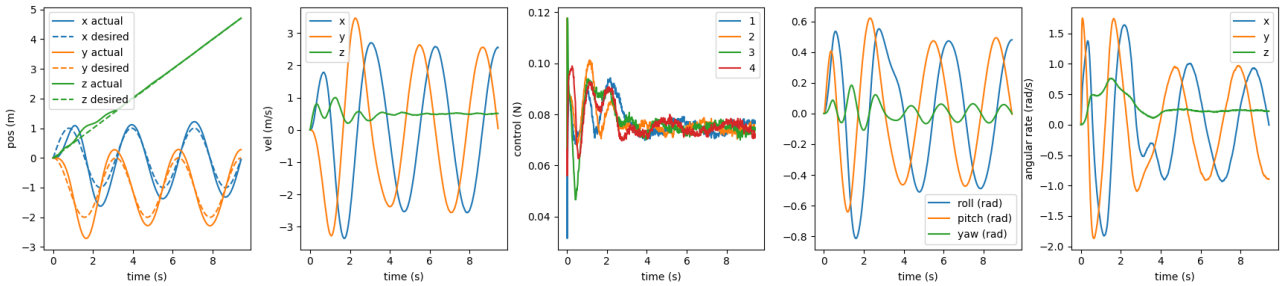
lower for a constant yaw angle of 0. This is to be expected since for a time-varying yaw angle, the controller needs to divert and/or spend additional energy (i.e. thrust) to yaw the drone instead of following the trajectory, we would expect the average control energy to increase and the RMSE of the tracking task to increase.

II.

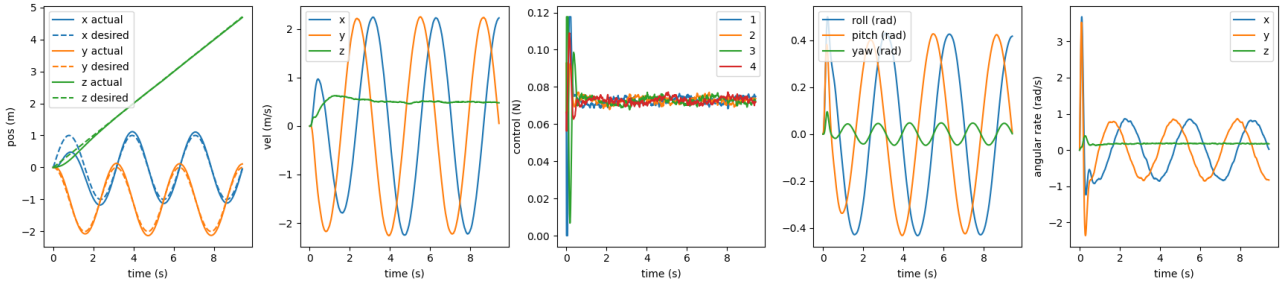
The PD gain values I chose were $K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$ (the same PD gains used for the previous section also works surprisingly well here). The RMSE achieved is 0.0702 meters. Its telemetry figure is displayed below:



I compared my PD gains to two of my classmates Wu Yi and Tom Gao. The preliminary PD gain values used by Wu Yi is $K_P = 3, K_D = 8, K_{P,\tau} = 50, K_{D,\tau} = 40$. The RMSE achieved is 0.22811 meters. Its telemetry figure is displayed below:



The preliminary PD gain values used by Tom Gao is $K_P = 2.5, K_D = 2.5, K_{P,\tau} = 170, K_{D,\tau} = 20$. The RMSE achieved is 0.13197 meters. Its telemetry figure is displayed below:



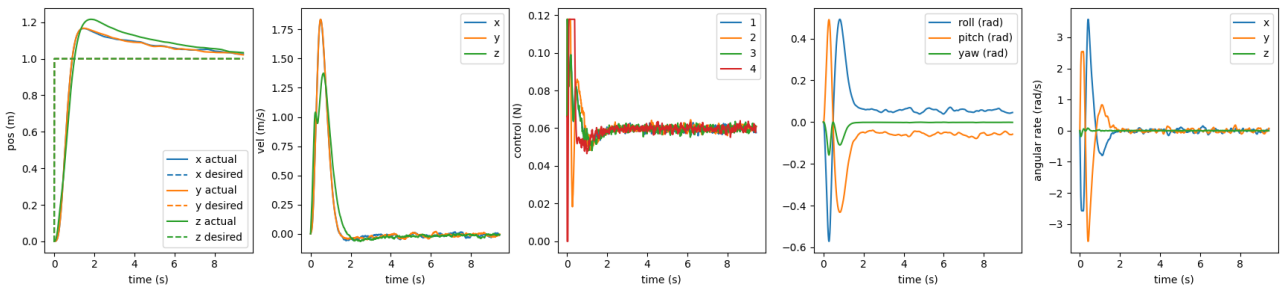
In terms of the tracking performance based on position RMSE, my PD gain values achieve the lowest RMSE and is comparatively better than my classmates' performance. The reason for my controller's better performance is presumably due to the more aggressive (higher) values of my PD gains. This causes the control inputs to be higher (as seen from the middle figure in the telemetries above). As such, if we were to compare performance based on average control energy expended, my PD gains would come last as my drone would need to expend greater control energy to track the trajectory more closely.

Q3.

I.

For the setpoint $[1, 1, 1]$ and constant disturbance value $[0.5, 0.5, 1.0]$, the following performance are obtained:

Using PD gains $K_P = 10,\ K_I = 2, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, we obtain the following:



$$Rise\ Time_x = 0.51026\ sec$$

$$Rise\ Time_y = 0.51026\ sec$$

$$Rise\ Time_z = 0.68034\ sec$$

$$Rise\ Time_{avg} = 0.56695\ sec$$

$$Max\ Overshoot_x = 0.16556\ m$$

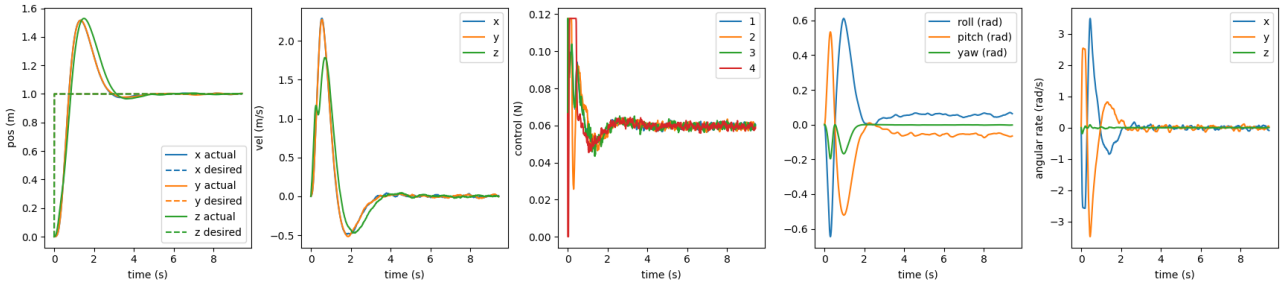
$$Max\ Overshoot_y = 0.16712\ m$$

$$Max\ Overshoot_z = 0.21597\ m$$

$$RMSE = 0.23159\ m$$

```
(base) williamfu@williams-mbp aerial_mobility_release % python quadrotor.py 3
Rise time for x is 0.51026, rise time for y is 0.51026, rise time for z is 0.68034
Average rise time is 0.56695
Maximum overshoot in x is 0.16556, maximum overshoot in y is 0.16712, maximum overshoot in z is 0.21597
RMSE: 0.23159
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7001/static/
```

Using PD gains $K_P = 10,\ K_I = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, we obtain the following:



$$Rise\ Time_x = 0.4002\ sec$$

$$Rise\ Time_y = 0.42021\ sec$$

$$Rise\ Time_z = 0.57029\ sec$$

$$Rise\ Time_{avg} = 0.46357\ sec$$

$$Max\ Overshoot_x = 0.51527\ m$$

$$Max\ Overshoot_y = 0.51651\ m$$

$$Max\ Overshoot_z = 0.52891\ m$$

$$RMSE = 0.25935\ m$$

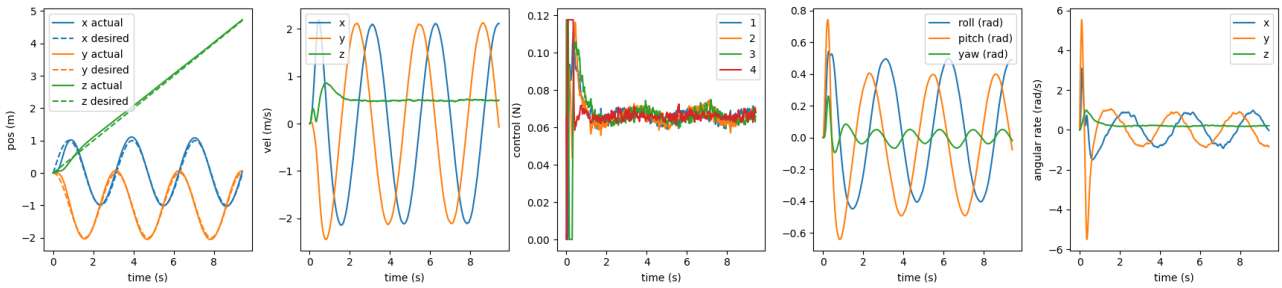
```
(base) williamfu@williams-mbp aerial_mobility_release % python quadrotor.py 3
Rise time for x is 0.4002, rise time for y is 0.42021, rise time for z is 0.57029
Average rise time is 0.46357
Maximum overshoot in x is 0.51527, maximum overshoot in y is 0.51651, maximum overshoot in z is 0.52891
RMSE: 0.25935
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7001/static/
```

Comparing the performance when choosing two different integral gains, we see that when the gain is increased from $K_I = 2$ to $K_I = 10$, we have a faster rise time and quicker convergence to the steady state value. However, this is at the expense of a greater maximum overshoot and higher RMSE tracking error. This is expected since the integral gain determines the aggressiveness of the convergence to the steady state error. A higher K_I would make the drone converge to the steady state faster. However, the tradeoff is that the transient performance is worse as we have a greater overshoot and the RMSE is higher. Whether a higher or lower integral gain offers better performance is up to the use case. If the user wants to reach the setpoint faster and does not care about the transient response, the setting a higher integral gain is arguably better. However, if the transient performance matters, then setting a lower integral gain is arguably better.

For the spiral trajectory and constant disturbance value $[0.5, 0.5, 1.0]$, the following performance are obtained:

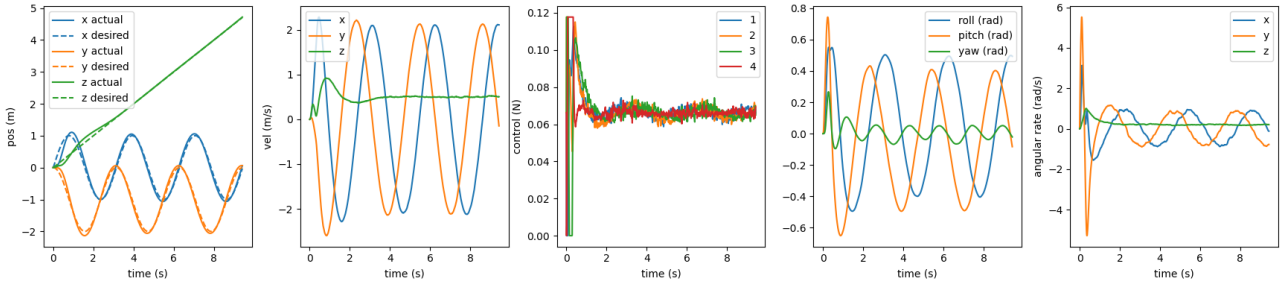
Using PD gains $K_P = 10,\ K_I = 2, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, we obtain the following:

$$RMSE = 0.08237\ m$$



Using PD gains $K_P = 10,\ K_I = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, we obtain the following:

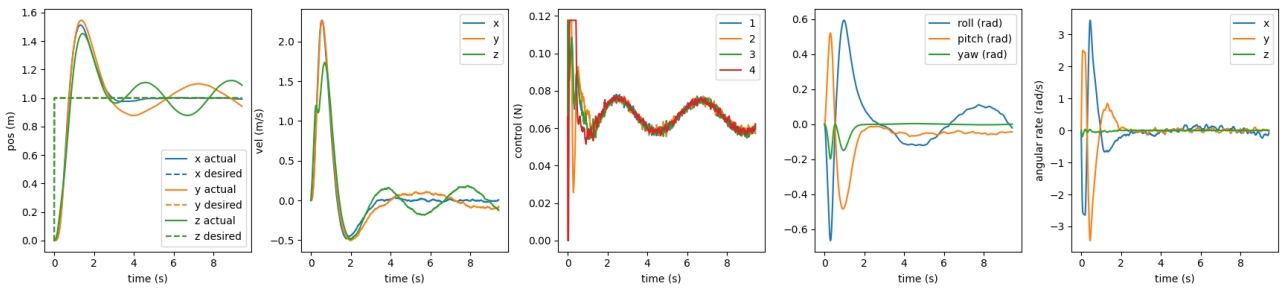
$$RMSE = 0.08722\, m$$



Again, the performance of the two integrals can be viewed as a tradeoff between faster steady state convergence against better transient performance. As with the setpoint tracking case, a higher K_I would make the drone converge to the steady state faster. However, the tradeoff is that the transient performance is worse as we have a greater overshoot and the RMSE is higher. For the trajectory tracking case, it may be optimal to set the integral gain to be less aggressive. This is because the steady state of a trajectory constantly changes over time. A high integral gain would cause the drone to follow the trajectory very aggressively, resulting in consistent overshooting and undershooting. This would increase RMSE and cause the drone to track the trajectory very poorly.

II.

Using PD gains $K_P = 10,\ K_I = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, we obtain the following for setpoint tracking:



$$Rise\ Time_x = 0.4002\ sec$$

$$Rise\ Time_y = 0.41021\ sec$$

$$Rise\ Time_z = 0.57029\ sec$$

$$Rise\ Time_{avg} = 0.46023\ sec$$

$$Max\ Overshoot_x = 0.51146\ m$$

$$Max\ Overshoot_y = 0.54515\ m$$

$$Max\ Overshoot_z = 0.45165\ m$$

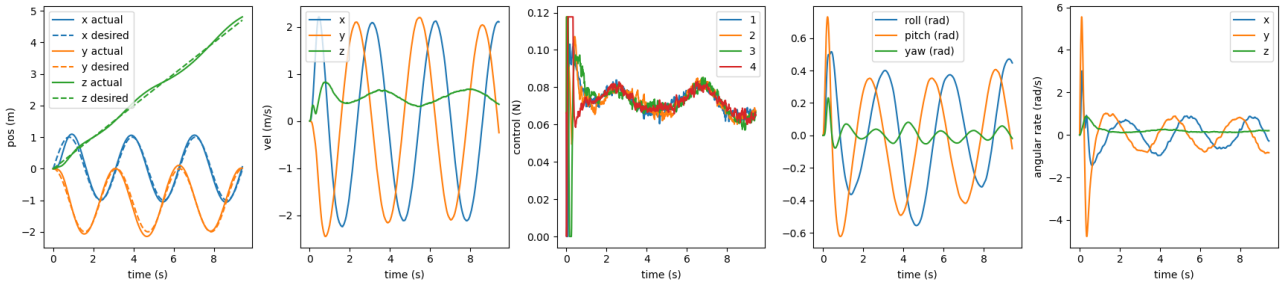
$$RMSE = 0.26142\ m$$

```
(base) williamfu@williams-mbp aerial_mobility_release % python quadrotor.py 3
Rise time for x is 0.4002, rise time for y is 0.41021, rise time for z is 0.57029
Average rise time is 0.46023
Maximum overshoot in x is 0.51146, maximum overshoot in y is 0.54515, maximum overshoot in z is 0.45165
RMSE: 0.26142
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7001/static/
```

We see that compared to the constant disturbance case, adding the integral control no longer asymptotes the steady state error to zero. Instead, for the *y*- and *z*-directions, the drone hovers in a sinusoidal pattern about the steady state, resulting in a higher RMSE (the error in the *x*-direction converges to 0 since the *x*-disturbance is a constant). This is to be expected since the *y*- and *z*-direction disturbances are sinusoidal in time, and the integral control is not well suited for time varying disturbances. The rise times and maximum overshoots for the two cases are mostly similar.

We obtain the following for trajectory tracking:

$$RMSE = 0.09601\ m$$

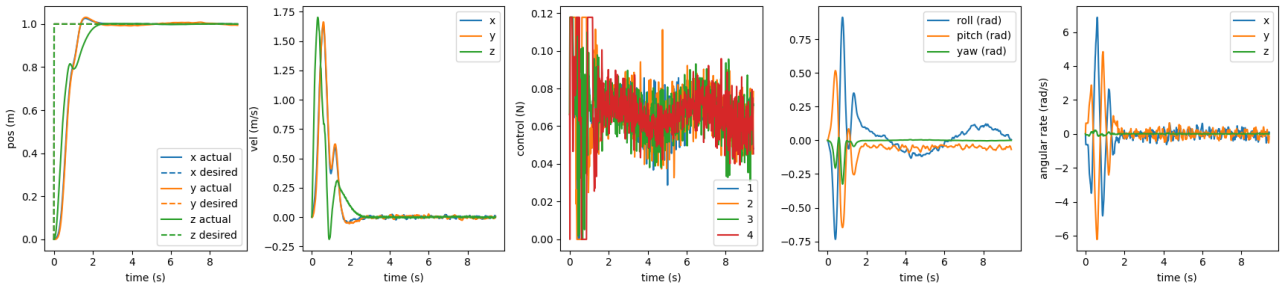


Again, compared to the constant disturbance case, we see that for the *y*- and *z*-directions, the drone varies in a sinusoidal pattern about the trajectory instead of asymptotically approaching the trajectory (the drone follows the trajectory more closely in the *x*-direction since the disturbance is

constant in the x-direction). This has the effect of increasing RMSE compared to the constant disturbance case.

III.

Using time-variant disturbance, PD gains $K_P = 10, K_D = 5, K_{P,\tau} = 200, K_{D,\tau} = 30$, low pass filter $\alpha = 0.5$, and Hurwitz matrix $A_s = -0.5 \times I_{3\times3}$, we obtain the following for setpoint tracking:



$$Rise\ Time_x = 0.82042\ sec$$

$$Rise\ Time_y = 0.80041\ sec$$

$$Rise\ Time_z = 1.34068\ sec$$

$$Rise\ Time_{avg} = 0.98717\ sec$$

$$Max\ Overshoot_x = 0.02661\ m$$

$$Max\ Overshoot_y = 0.03129\ m$$

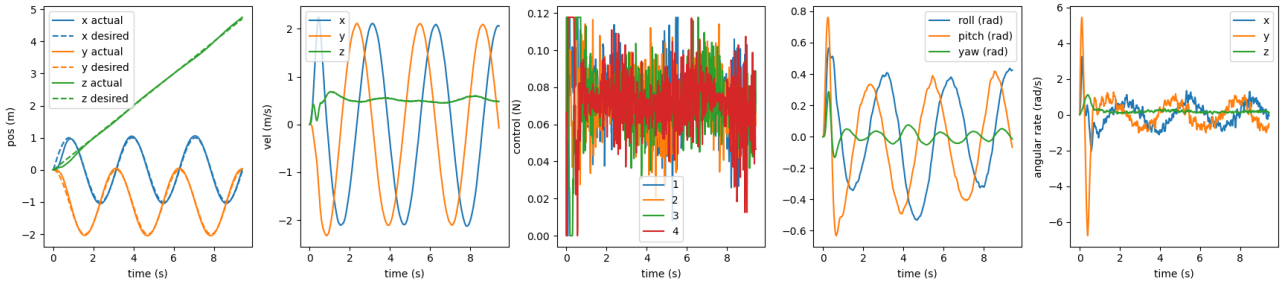
$$Max\ Overshoot_z = 0.00204\ m$$

$$RMSE = 0.22356\ m$$

```
Rise time for x is 0.82042, rise time for y is 0.80041, rise time for z is 1.34068
Average rise time is 0.98717
Maximum overshoot in x is 0.02661, maximum overshoot in y is 0.03129, maximum overshoot in z is 0.00204
RMSE: 0.22356
You can open the visualizer by visiting the following URL:
http://127.0.0.1:7001/static/
```

We obtain the following for trajectory tracking:

$$RMSE = 0.06566\ m$$



The performance of the L-1 adaptive control is considerably better than integral control based on several factors. Firstly, the overshoot from setpoint tracking is now almost non-existent. The sinusoidal hovering about the setpoint has also disappeared in both the *y*- and *z*-directions. This results in a decreased RMSE and better setpoint tracking. However, one drawback is that the rise time is now slightly longer by about 0.5 seconds. I argue that this is a reasonable tradeoff for a significantly reduced overshoot and steady state error. The same can also be said for the trajectory tracking. Using L-1 adaptive control, the sinusoidal pattern about the trajectory is now non-existent, and the drone now tracks the trajectory much better and with a lower RMSE.

The reason for the better performance is because L-1 adaptive control uses a closed-loop predictor to estimate the disturbance. It then feeds this estimate into the control loop so that the desired force (and thus produced thrust) can take account of this exogeneity. To avoid erratic drone behavior, a low-pass filter is implemented so that the estimated disturbance changes slowly over time. In summary, the L-1 adaptive control has better performance compared to integral control in both the setpoint tracking and trajectory tracking case.