Drone-tracking final report

Aerospace Engineering

Hao Liu

wdliu@umich.edu

5/1/2022

1 Introduction

This is a report about what I've done during the undergraduate research, including literature review, simulation and testing with Tello EDU. In this report, I'll discuss all the details so that readers can easily reproduce my result and continue working on this project. All the codes have already been uploaded to Google drive and Github.

2 Literature review and simulation

Here is a brief summary to the goal of literature review and simulation. Our general purpose is to have multiple quadrotors track multiple moving target platform. To achieve that, the research is divided into several steps:

- Build the dynamic system of a quadrotor so that it can track certain target speed.
- Find an appropriate tracking policy so that the quadrotor can track a moving traget
- Improve the 1 vs. 1 case to a n vs. n case, leading to the lowest tracking cost.
- Improve the full observation case to partial observation case.

The main reference paper for each part will be introduced in detail in the next part.

2.1 Dynamic system and general control

I referred to [1] for the dynamic system of a quadrotor, which system is also used by many other papers. In brief, the state space system is defined as:

$$\mathbf{x}^T = (\dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}), \tag{1}$$

where x,y, and z are the position of the quadrotor and ϕ,θandψ are the roll, pitch and yaw angle. And based on the dynamic model, we can get

$$\dot{\mathbf{x}} = \begin{bmatrix} -(\cos x_4 \sin x_5 \cos x_6 + \sin x_4 \sin x_6) u_1/m \\ -(\cos x_4 \sin x_5 \cos x_6 - \sin x_4 \sin x_6) u_1/m \\ g - (\cos x_4 \cos x_5) u_1/m \\ x_7 \\ x_8 \\ x_9 \\ x_8 x_9 I_1 - \frac{I_R}{I_x} x_8 g(u) + \frac{L}{I_x} u_2 \\ x_7 x_9 I_2 + \frac{I_R}{I_y} x_7 g(u) + \frac{L}{I_y} u_3 \\ x_7 x_8 I_3 + \frac{1}{I_z} u_4 \end{bmatrix}, \tag{2}$$

where

$$u_1 = b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \quad (3)$$
$$u_2 = b(\omega_2^2 - \omega_4^2) \quad (4)$$
$$u_3 = b(\omega_1^2 - \omega_3^2) \quad (5)$$
$$u_4 = d(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \quad (6)$$

Here ω is the rotation speed of each motor b is the thrust factor and d is the drag factor.The purpose of the control is to calculate the desired u_1-u_4, with which we can get the rotation speed of each motor and update the state of the quadrotor.

The overall control system can be separated into two parts and the logic is shown in Fig. 1

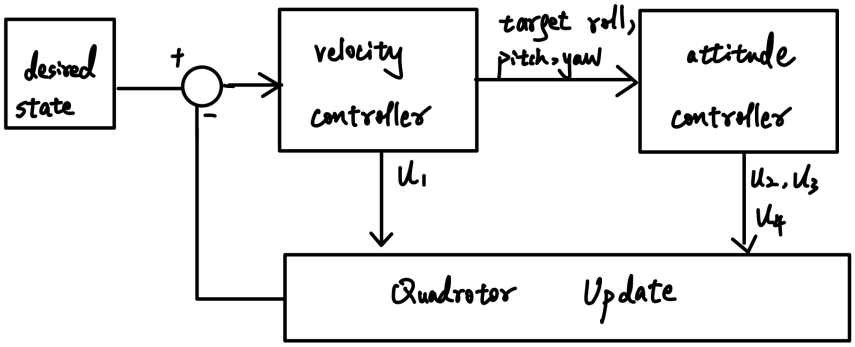


Figure 1: Angles during attitude control

For the velocity control part, I set x_{4d}, x_{5d} and x_{6d} to be the desired roll pitch yaw angle that are transferred to the attitude controller and they are also the desired angle to finish the velocity control. It's obvious that all the desired velocity and attitude can be reached without any yaw angle, thus I can set ψ=x_{6d}=0. In this way, the first three rows of the state can be updated by the following equations:

$$\dot{\mathbf{x}} = \begin{bmatrix} -(\cos x_{4d} \sin x_{5d}) u_1/m \\ -(\cos x_{4d} \sin x_{5d}) u_1/m \\ g - (\cos x_{4d} \cos x_{5d}) u_1/m \end{bmatrix}, \quad (7)$$

I apply simple proportional controller to solve the problem which is shown as follows.

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} = \begin{bmatrix} k_1(x_{1d} - x_1) \\ k_2(x_{2d} - x_2) \\ k_3(x_{3d} - x_3) \end{bmatrix} \quad (8)$$

Solving Eqn.(7) and Eqn.(8) and setting

$$\alpha = sinx_{4d} \quad (9)$$
$$\beta = sinx_{5d}, \quad (10)$$

I can get the following results if $\beta \neq 0$:

$$\begin{aligned} \beta &= \pm\left[(\tfrac{g-\hat{x}_3}{\hat{x}_1})^2+1\right]^{-\frac{1}{2}} && (11)\\ u_1 &= m\sqrt{\frac{\hat{x}_1^2}{\beta^2}+\hat{x}_2^2} && (12)\\ \alpha &= \hat{x}_2\frac{m}{u_1} && (13) \end{aligned}$$

Here β is negative if $\hat{x}_1$ is positive ad vice versa. If $\beta = 0$, I set

$$u_1 = mg.$$

With α and β calculated, I can calculate the desired x_{4d} and x_{5d} and make them an input into the attitude controller.

In the attitude controller, since the gyroscope term (g(u)) is very small compared with other terms thus I can neglect them first and the system becomes:

$$\begin{bmatrix}\dot{x}_7\\ \dot{x}_8\\ \dot{x}_9\end{bmatrix} = \begin{bmatrix}x_8x_9I_1+\frac{L}{I_x}u_2\\ x_7x_9I_2+\frac{L}{I_y}u_3\\ x_7x_8I_3+\frac{1}{I_z}u_4\end{bmatrix}. \quad (14)$$

I apply a feedback linearization in order to obtain a linear system:

$$\begin{bmatrix}u_2\\ u_3\\ u_4\end{bmatrix} = \begin{bmatrix}f_2(x_7,x_8,x_9)+u_2^*\\ f_3(x_7,x_8,x_9)+u_3^*\\ f_4(x_7,x_8,x_9)+u_4^*\end{bmatrix} \quad (15)$$

with new input variables u_2^*, u_3^* and u_4^*. To make it a linear system, the following requirements must be meet:

$$\begin{aligned} x_8x_9I_1+\frac{L}{I_x}f_2(x_7,x_8,x_9) &= K_2x_7 && (16)\\ x_7x_9I_2+\frac{L}{I_y}f_3(x_7,x_8,x_9) &= K_3x_8 && (17)\\ x_7x_8I_3+\frac{1}{I_z}f_3(x_7,x_8,x_9) &= K_4x_9 && (18) \end{aligned}$$

so that the system can be written as

$$\begin{bmatrix}\dot{x}_7\\ \dot{x}_8\\ \dot{x}_9\end{bmatrix} = \begin{bmatrix}K_2x_7+\frac{L}{I_x}u_2^*\\ K_3x_8+\frac{L}{I_y}u_3^*\\ K_4x_9+\frac{1}{I_z}u_4^*\end{bmatrix}, \quad (19)$$

where $K_{2,3,4}$ are constants and

$$\begin{bmatrix}u_2^*\\ u_3^*\\ u_4^*\end{bmatrix} = \begin{bmatrix}w_2(x_{4d}-x_4)\\ w_3(x_{5d}-x_5)\\ w_4(x_{6d}-x_6)\end{bmatrix}. \quad (20)$$

$w_{1,2,3}$ are constants and $x_{4d,5d,6d}$ are the desired roll, pitch and yaw angle. For detailed proof of this transformation, please refer to [1].

With all the calculations above, I can calculate the input **u** at each time step and have the quadrotor track a target speed as well as keep itself stable. Here I first neglect the g(u) term in Eqn.(2) since it's very small compared with other term. Some simulation result is shown below. First, the initial roll, pitch and yaw angle is ($\phi = 30°$, $\theta = -20°$, $\psi = 10°$) and the target state is steady state with all the velocities to be zero. I got the following simulation result in Fig. 2.

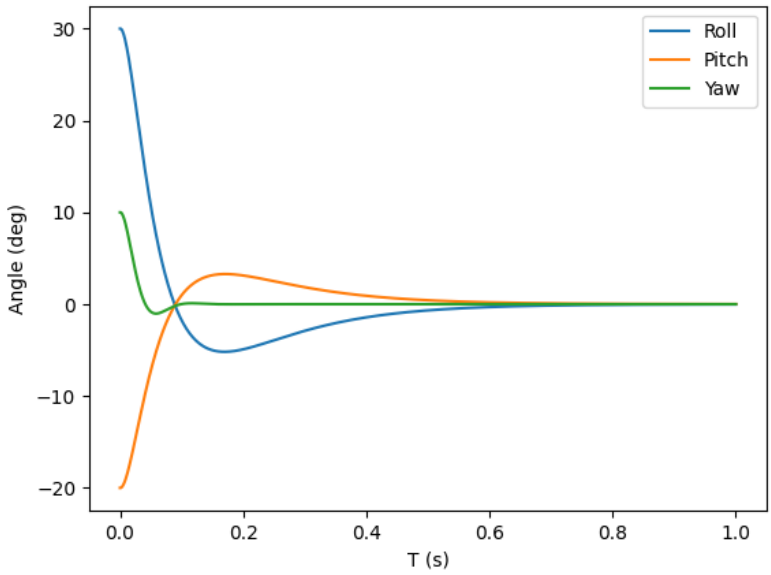


Figure 2: Angles during attitude control

It's easy to see that the system comes to stale after about 0.6 seconds.

The second test was proceeded with hovering initial condition and stable desired state with the speed to be 0.5 m/s in all the three directions. The result is shown in Fig. 3.

However, when I wanted to calculate rotation speed of each motor, it turned out that, at some time steps, ω^2 was negative, which indicated that there's no possible motor speed to realize such control policy. By testing, I found that negative ω^2 usually happened at the beginning of the control, which is usually the most intense part. So I guessed that the reason for the negative ω^2 was the too intense control input. Thus, I chose to control the magnitude of Eqn.(8) and Eqn.(20) to get a less intense control input, especially Eqn.(20). The less reasonable the initial state is, the smaller the maximum magnitude can be set and the slower the control is. For example, for the simulation result in Fig. 3, (from hovering state to another steady flight), the maximum magnitude for Eqn.(8) can be set to 100 and the maximum magnitude of Eqn.(20) can be set to 2.5 and get the result shown as Fig. 4 . However, for the simulation result in Fig. 2, (from certain angled state to hovering state),

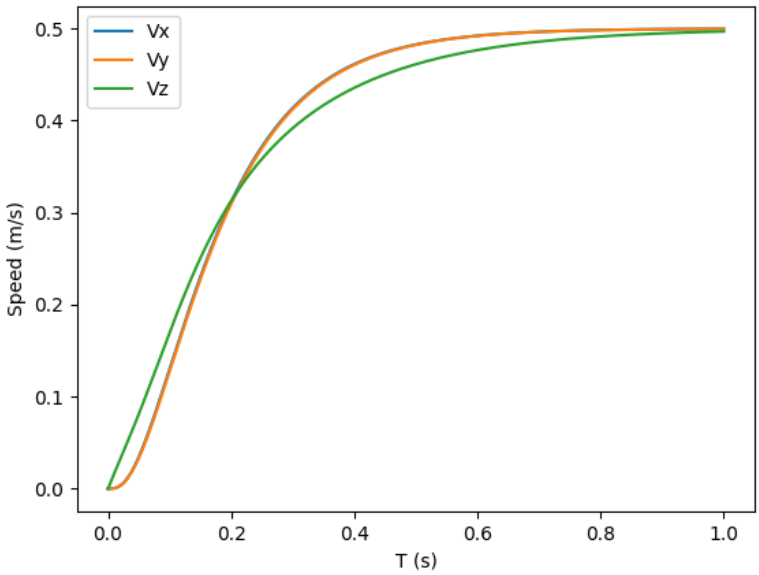


Figure 3: Speed during velocity control

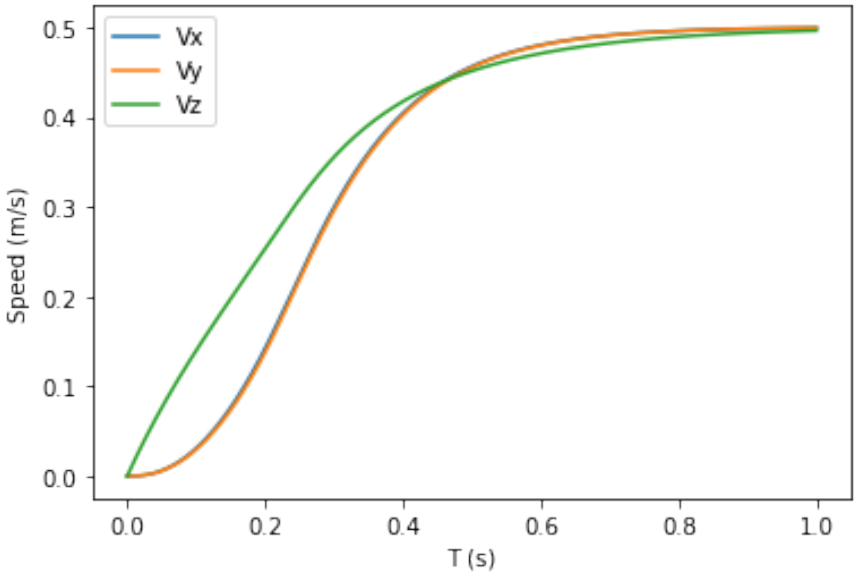


Figure 4: Speed control with control input constrain

since, unless set manually, it's impossible for a hovercraft reach an angled state by itself without any other speed or acceleration, the initial state has little physical meaning which means it's much harder to be controlled. In this case, the maximum magnitude for Eqn.(8) can only be set to 2 and the maximum magnitude of Eqn.(20) can only be set to 0.6 and get the result shown as Fig. 4 .

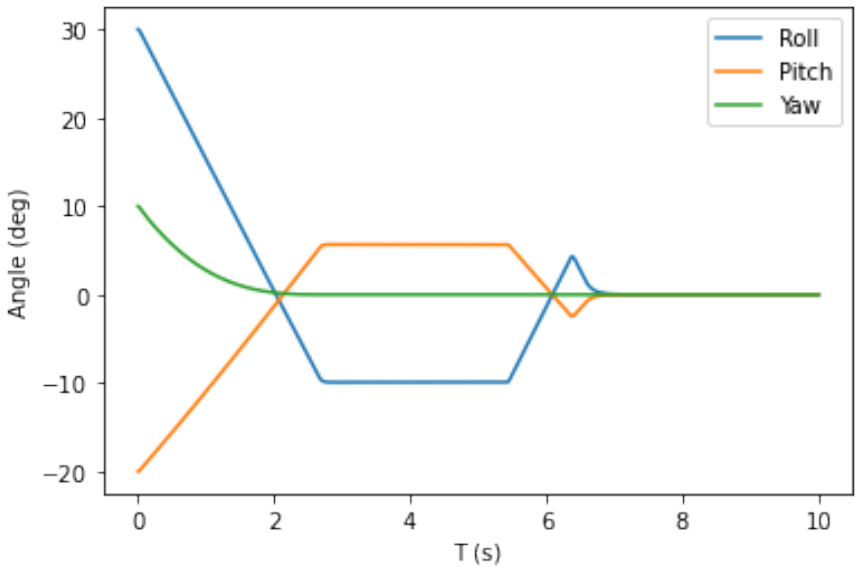


Figure 5: Angle control with control input constrain

Finally, I proved that the constrained control policy, if set proper constrain magnitude, can help track from initial states with physical meanings to different final states which also have physical meanings.

2.2 Tracking policy

Section 2.2 is the tracking simulation based one section 2.1. Instead of tracking some desired speed , the tracking target becomes to a moving platform. Generally speaking, I add a tracking controller of a classical missile guidance problem to provide the desired $\vec{v}$, with which I can calculate u_1 and the desired roll, pitch and yaw angle for the attitude controller. The improved block diagram is shown as Fig.6

I referred to [2] for the tracking and landing policy. In [2], it constructed a tracking system based on the dynamic system shown in [1]. Instead of directly combining [1] and [2], I applied our constrained control policy in this part to make sure the control policy is not too intense to reach. The landing policy is just a first order control policy and the tracking policy is derived from classical missile guidance problem. Different from the state system in the previous section, positions of quadrotors were added to the states since the target of the control at this section is to reach certain target position. For now, the quad state is defined as

$$\mathbf{x}^T = (\dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, x, y, z). \tag{21}$$

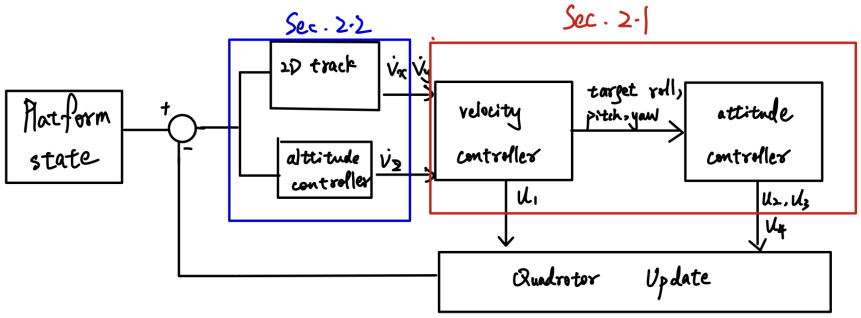


Figure 6: Block diagram of the tracking process

Two more new states, the distance state and the platform state, are also defined here:

$$\mathbf{distance}^T = (R, \sigma) \tag{22}$$

$$\mathbf{x}_p^T = (x_p, y_p, z_p, v_{P_x}, v_{P_y}, v_{P_z}) \tag{23}$$

The definition of distance is shown in Fig. 7. The platform state is composed of the position

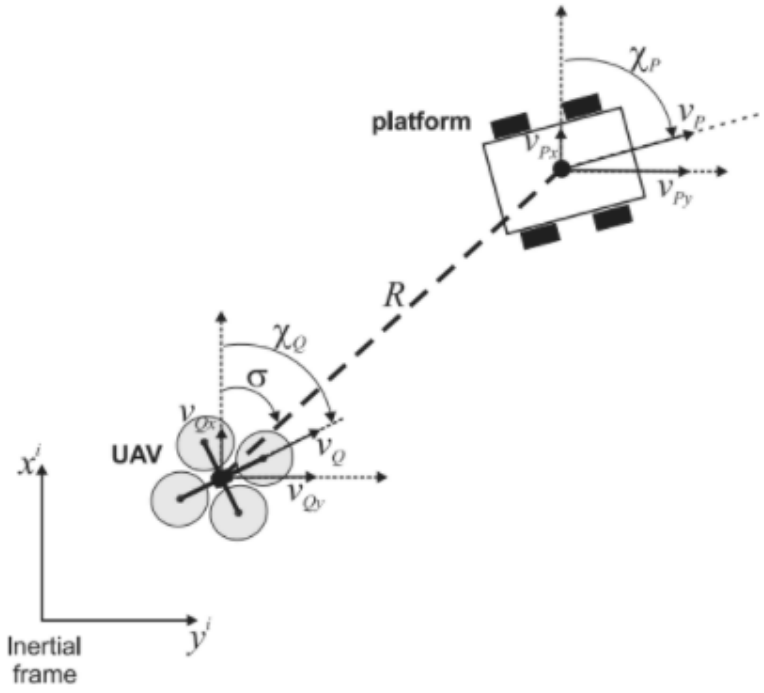


Figure 7: Definition of distance state[2]

of the platform and velocity of the platform. The distance and the quad speed is now updated

by:

$$\dot{R} = v_{P_x}\cos\sigma + v_{P_y}\sin\sigma - v_{Q_x}\cos\sigma - v_{Q_y}\sin\sigma \quad (24)$$

$$\dot{\sigma} = \frac{1}{R}(v_{P_y}\cos\sigma - v_{P_x}\sin\sigma - v_{Q_y}\cos\sigma + v_{Q_x}\sin\sigma) \quad (25)$$

$$\dot{V_{Q_x}} = -(1/T_1)V_{Q_x} + (1/T_1)u_1 \quad (26)$$

$$\dot{V_{Q_y}} = -(1/T_2)V_{Q_y} + (1/T_2)u_2 \quad (27)$$

Here, if R, the distance between the quadrotor and the target is smaller than a designed threshold, $u_1 = v_{P_x}$ and $u_2 = v_{P_y}$. Else,

$$u_1 = v_{P_x} + T_1 R\cos\sigma - T_1\frac{\sigma}{R}\sin\sigma \quad (28)$$

$$u_2 = v_{P_y} + T_2 R\sin\sigma - T_2\frac{\sigma}{R}\cos\sigma \quad (29)$$

The key issue is to apply this track controller with the quadrotor dynamic control system shown in the previous section. I use Eqn. (26) and Eqn. (27) to take the place of the first two rows in Eqn. (8). For the third row in Eqn. (8), the control logic is shown as Fig. 8, where the transfer functions can be written as:

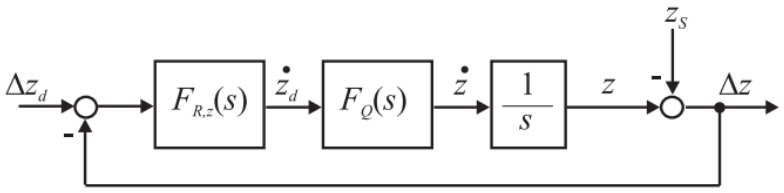


Figure 8: Altitude control loop. [2]

$$F_Q(s) = \frac{V_z(s)}{V_{zd}(s)} \approx \frac{1}{T_3 \times s + 1} \quad (30)$$

$$F_{R,z}(s) = K(1 + T_C \times s) \quad (31)$$

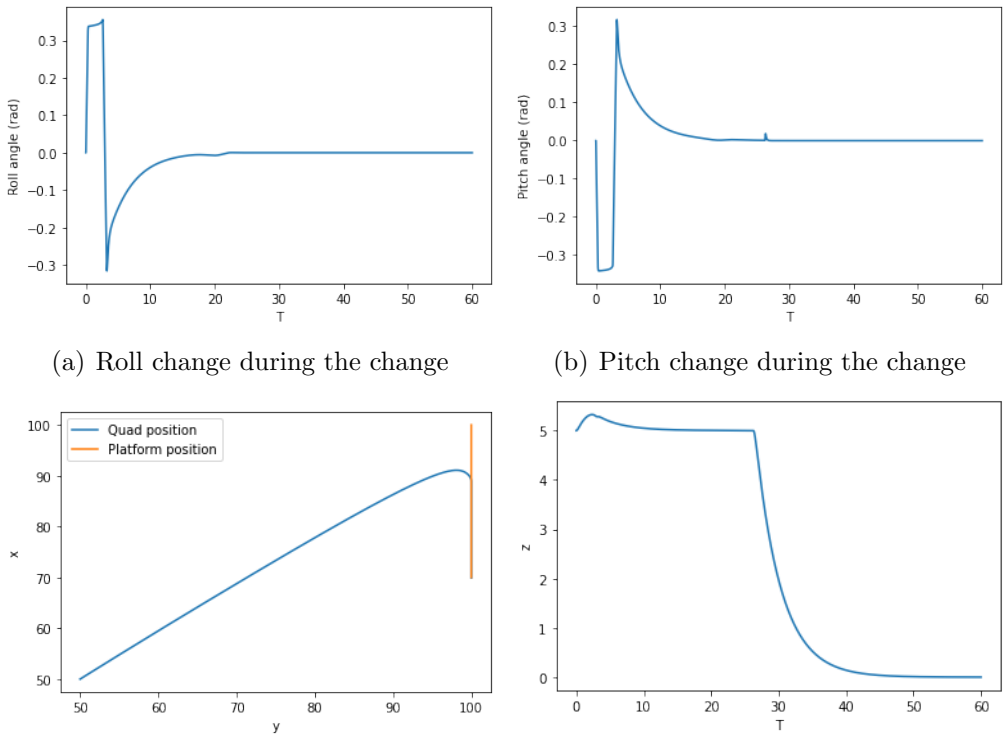
Here z is the altitude of the quadrotor, z_d is the target altitude (initial height during tracking process and the height of the platform during the landing process) and z_s is the surface altitude. $\Delta z = z - z_s$ and $\Delta z_d = z_d - z_s$.TC and k_3 are all constants. Inversely Laplace transform the transfer function, I can get the update equation shown as follows

$$v_{zd} = \frac{z_d - z - TC \times v_z}{1/K - TC} \quad (32)$$

$$a_z = k_3 \times (v_{zd} - v_z), \quad (33)$$

Eqn. (32) and Eqn. (33) took the place of the third row in Eqn. (8). Here, the magnitude constrain I chose for Eqn.(8) is 5 and for Eqn.(20) is 3.

The tracking result as well as its corresponding attitude change and altitude change is shown as Fig. 9.



(a) Roll change during the change

(b) Pitch change during the change

(c) Track

(d) Attitude

Figure 9: Tracking example with $\mathbf{x}_q$=[90,90,5] and $\mathbf{x}_p$=[10,10,0] and $V_{yp} = 0.5$

2.3 Cost assignment

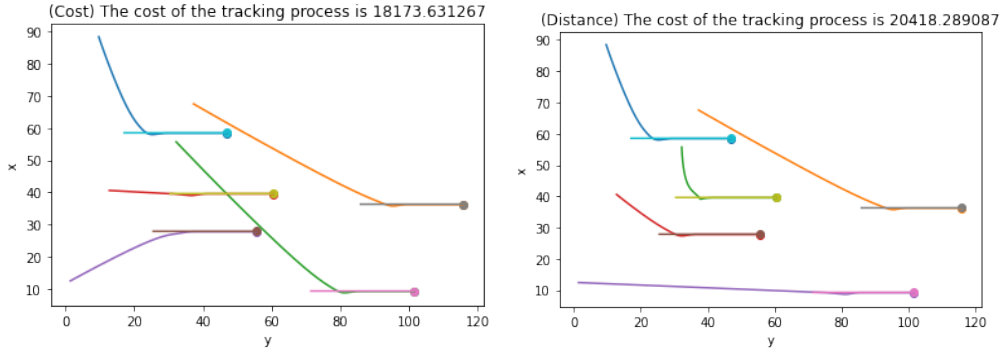
In this section, the problem is upgraded from 1 v.s. 1 problem to n v.s. n problem. The key point here is to find a best quad-platform assignment so that the tracking cost is to the lowest. First, the instant cost need to be defined to better illustrate how hard the tracking process is. Here, the Lyapunov function defined in [2] is taken as the instant cost.

$$\dot{q} = 0.5 \times (R^2 + \sigma^2 + (V_{qx} - V_{px})^2 + (V_{qy} - V_{py})^2) \tag{34}$$

Integrate $\dot{q}$ over time, I can get the tracking cost of one assignment. Then the problem changes to find the assignment with the lowest cost.

Firstly, calculate the cost of all the possible assignments and store them in a n×m matrix, where n is the number of quadrotors and m is the number of platforms (n≤m). Here I are talking about n v.s. n cases so n=m. Then apply the Python Optimal Transport package to find the best assignment based on the cost matrix and the number of quadrotors and platforms. The emd (intended for solving Earth Moving Distance problem) function is used to create the assignment matrix. The assignment matrix A is also a $n \times n$ matrix and it is sparse matrix with 1 on the specific assignment element. For example, if the p^{th} quadrotor is assigned to the q^{th} platform, then all the A[p,q]=0 and all other elements on the p^{th} row are 0. To fit the assignment matrix A, the state of quadrotors are stored in a $n \times 12$ matrix and the state of platforms are stored in a $n \times 6$ matrix, where n represents the number of quadrotors and platforms.

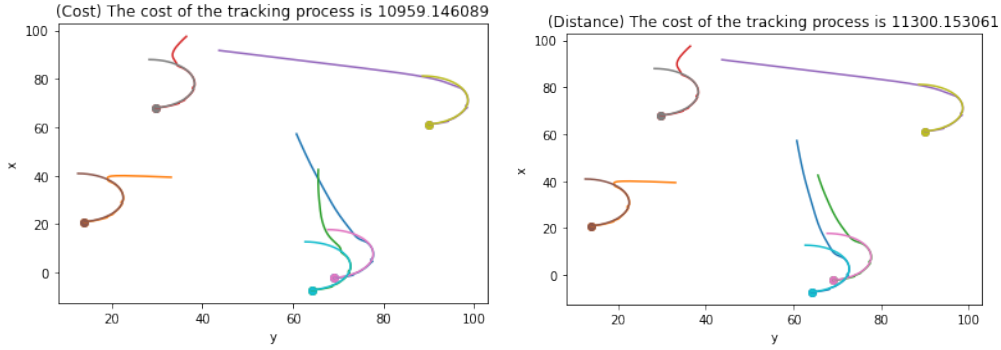
To prove the assignment can give the lowest cost, I calculated the cost of assignment based on distance as is shown in Fig. 10



(a) Based on cost (b) Based on distance

Figure 10: Tracking example with different assignment

The initial positions of all the quadrotors and platforms are generated randomly and I can easily see from the figure that the first assignment based on the cost has a lower cost than the distance-based assignment, proving the correctness of the method. This can also be applied to tracking platforms with non-constant velocity as is shown in Fig.11.



(a) Based on cost (b) Based on distance

Figure 11: Tracking example with different assignment and non-constant velocity platforms

Here, the magnitude constrain I chose for Eqn.(8) is 2 and for Eqn.(20) is 0.6.

2.4 Partial observation

In real life case, usually, it's hard for the quadrotors to monitor the speed of the platforms, leading to a partial observation case. In this part, I use the position of the platform read to estimate the speed of the platform so that the previous control policy still works. The overall control logic is shown as Fig. 12. Here I assume that I can fully observe the quadrotor with different sensors but I can only observe the relative position of the platform.

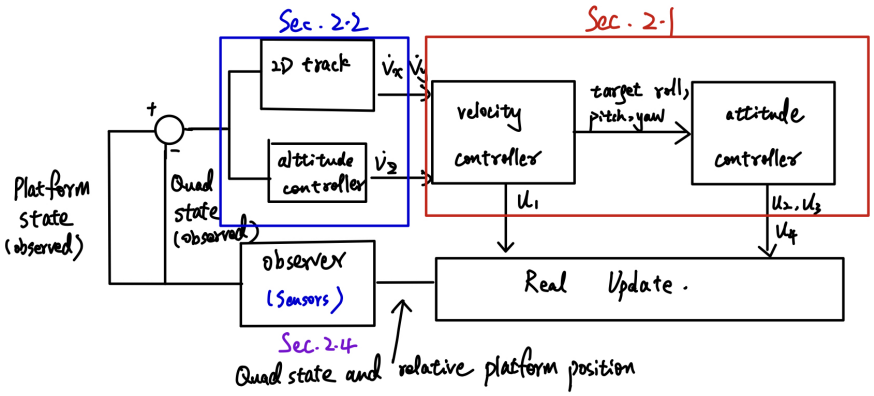


Figure 12: Block diagram of the partial observation tracking process

The main reference here is [3]. The logic of the observation system is shown in Fig. 13. I obtain relative position of the platform (by camera in [3]), based on which I can estimate the real position and speed of the platform. Since in our case, all the coordinates are in the

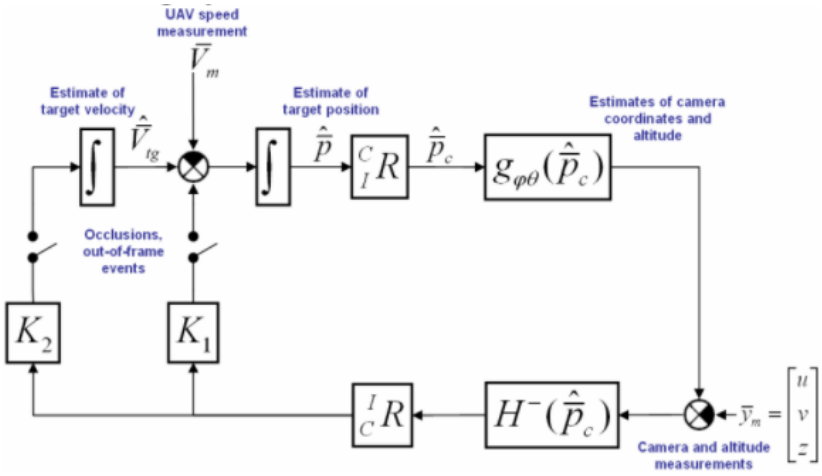


Figure 13: Observation system logic flow[3]

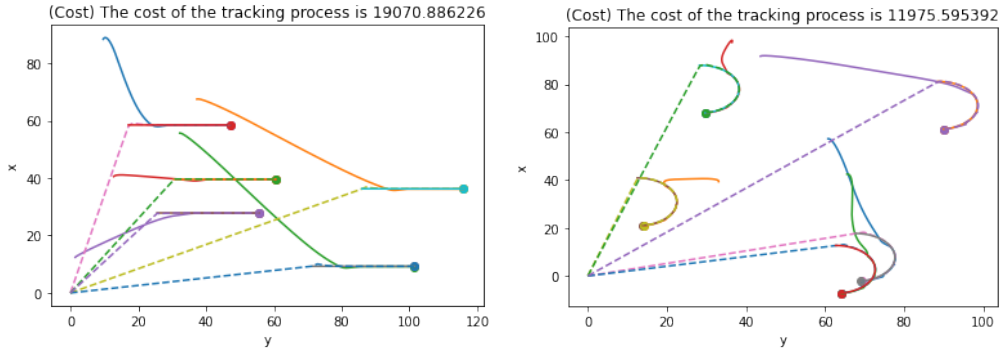
same reference frame, I can neglect the coordinate transformation terms R. The system can be written as:

$$\frac{d}{dt}(\bar{p}_{ob}) = -\bar{V}_m + \hat{\bar{V}}_{tg} + K_1 \times (\bar{p}_{ob} - \bar{y}_m) \tag{35}$$

$$\frac{d}{dt}(\hat{\bar{V}}_{tg}) = K_2 \times (\bar{p}_{ob} - \bar{y}_m) \tag{36}$$

Here $\bar{p}_{ob}$ is the observed position of the platforms, $\bar{y}_m$ is the exact position of the platform (should be the data extracted from the camera as [3], but since it's wasting of time converging the state to the camera reference frame and then transfer back to the global state, here I directly used the exact position), $\bar{V}_m$ is the velocity of the quadrotor and $\hat{\bar{V}}_{tg}$ is the estimated target speed relative to the quadrotor. The exact estimated speed of the platform in the global frame is $\frac{d}{dt}\bar{p}_{ob}$. In our simulation, I set the initial observation position of the platform

to be [0,0,0] and use the observed value in the tracking controller instead of the real position of the platform. I applied partial observation to the simulation the the previous section and achieved the result shown as Fig. 14 , where the dotted lines shows the observed position of



(a) Constant speed (b) Non-constant speed

Figure 14: Tracking example with partial observation

the platforms

Also, in [3], it says this observation method can only be used to those with constant speed but as is shown in Fig. 14, it also works well with non-constant speed target. The limitation seems to be that the target can't change its speed to rapidly or it can be observed.

3 Tello EDU

With the simulation results achieved, I used Tello EDU, a programmable quadrotor, to test whether the control model simulated can be applied to real life. Before going deep into the tests, I'd first introduce Tello EDU so that readers can have a general idea of what Tello EDU is and how to control it with codes. Most of the information are acquired from the Github page of DJITelloPy (https://github.com/damiafuentes/DJITelloPy) and Tello user guide (https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf).

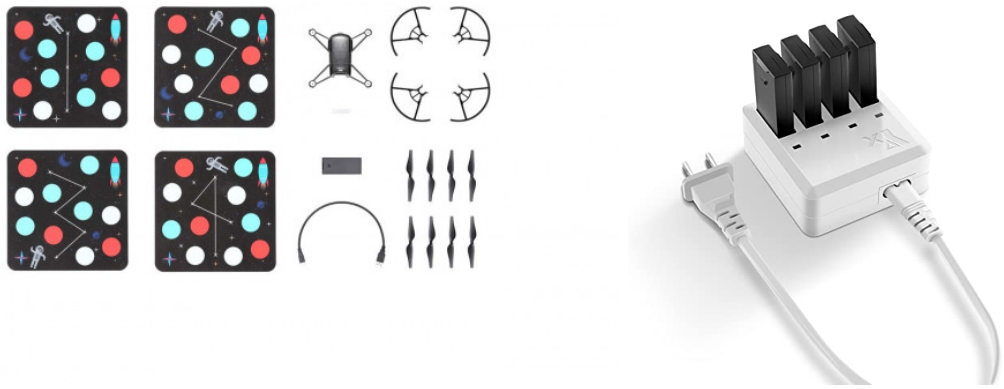
For this project, I purchased a Tello EDU combo including one drone, one battery, four mission pads and for spare propellers. I also purchased four additional batteries and one battery charger so that I can recharge batteries and do tests sychronically (Fig. 15). For detail procedures of assembling and recharging, please refer to the instruction book of Tello EDU.

I used Python and WiFi to connect and control the drone. Before operating the control code, turn on the drone and connect your laptop to the corresponding WiFi signal. Basic control commands will be discussed in the following sections.

3.1 Python package used

3.1.1 djitellopy

This is the package provided by Dji. Tello.py inside includes most of the functions we used to control the drone



(a) Tello EDU combo (b) Battery charger

Figure 15: Purchased items

3.1.2 cv2

We use cv2 package to modify on the image we obtained from the cameras of the drone and show it on computer.

3.1.3 pygame

We use pygame to write the function of keyboard reading so that we can send command to the drone with our keyboard.

3.2 Tello function introduction

3.2.1 Connection and vedio setup

1. connect() : Send the connection request to the drone
2. streamon() : Enable video stream
3. streamoff() : Disable video stream.
4. set_video_direction(int) : The Tello Edu drone has two cameras. One is in the front and one is at below. This function is used to indicate which camera is now being displayed on the screen. When the input is set 0, the front view is displayed and when the input is set 1, the down view is displayed.
5. get_frame_read() : Get the image from the camera.

3.2.2 Basic control Command

1. takeoff(): Automatic takeoff.
2. land(): Automatic landing.
3. get_state_field(str): Get the a specific component of the drone's state. Possible entries a relisted as follows:

1. Tello EDU with mission pads enabled only 'mid', 'x', 'y', 'z', 'mpry': (custom format 'x,y,z')

2. Common entries: 'pitch', 'roll', 'yaw', 'vgx', 'vgy', 'vgz', 'templ', 'temph', 'tof', 'h', 'bat', 'time','agx', 'agy', 'agz')

All other get_sth. functions are written based on this function. The description of each input is shown as Table. 1

4. send_rc_control(lr,fb,ud,yv): Four channels correspond to four different control command. lr represents the left_right velocity, fb represents the forward_backward velocity, ud represents the up_down velocity and yv represents the yaw velocity. The range of all inputs is -100 - 100.

Table 1: Input command description

mid	The ID of the Mission Pad detected. If no Mission Pad is detected, a "-1" message will be received instead
x,y,z	The "x,y,z" coordinate detected on the Mission Pad. If there is no Mission Pad, a "0" message will be received
mpry	The pitch roll and yaw angle relative to the detetcted Mission Pad
pitch,roll,yaw	The degree of the atiitude pitch roll and yaw angle
vgx,vgy,vgz	The speed of the "x,y,z" axis
templ,temph	The lowest or highest temperature in degree Celsius
tof	The flight distance in cm
h	The height in cm
time	The amount of time the motor has been used
agx,agy,agz	The acceleration of the "x,y,z" axis

3.3 Mission Pad related command

1. enable_mission_pads( ): Turn on the Mission Pad detection function.
2. disable_mission_pads( ): Turn off the Mission Pad detection function.
3. set_mission_pad_detection_direction(int): Set the detection direction. 0 enables downwards detection only. 1 enables forward detection only. 2 enables both forward and downward detection.
4. go_xyz_speed_mid(x,y,z,speed,mid): Fly to x y z relative to the mission pad with id mid. Speed defines the traveling speed in cm/s. All the inputs are integers and if no Mission Pads are found, an error will be returned.
5. go_xyz_speed_yaw_mid(x, y, z, speed, yaw, mid1, mid2): Fly to x y z relative to mid1. Then fly to 0 0 z over mid2 and rotate to yaw relative to mid2's rotation. Speed defines the traveling speed in cm/s. All the inputs are integers and if no Mission Pads are found, an error will be returned.

4 Test

This section discusses the tests I did with Tello EDU.

4.1 Example tests

There is a series of tutorial videos on YouTube and I mainly referred to the video posted by Murtaza's Workshop - Robotics and AI(https://www.youtube.com/watch?v=LmEcyQnfpDA). I also referred to the tutorial on CVzone (https://www.computervision.zone/courses/drone-programming-course/). These instruction videos are very much in detail and they also uploaded their code. The most relevant test within all the examples is the Object Tracking project. In this project, he used the cv2 package and the front camera of the drone to have the drone track a certain object in front of it, as is shown in Fig. 16.

Figure 16: Object tracking demonstration

To achieve this, he designed a color filter based on hue, saturation and value (HSV), with which we can obtain a figure with objects of only certain desired color. After that, it is easy to use functions to find the contour of the objects as well as the approximate center of the object. The image obtained from the camera is divided into 9 parts, and the drone will operate differently if the contour center is placed within different parts, as is shown in Fig. 17. For example, if the contour center is listed within the left-center area, yv (yaw velocity) in the send_rc_control(lr,fb,ud,yv) function is set to be -30 to have the drone rotate counterclockwise so that the contour center can move towards the center block.

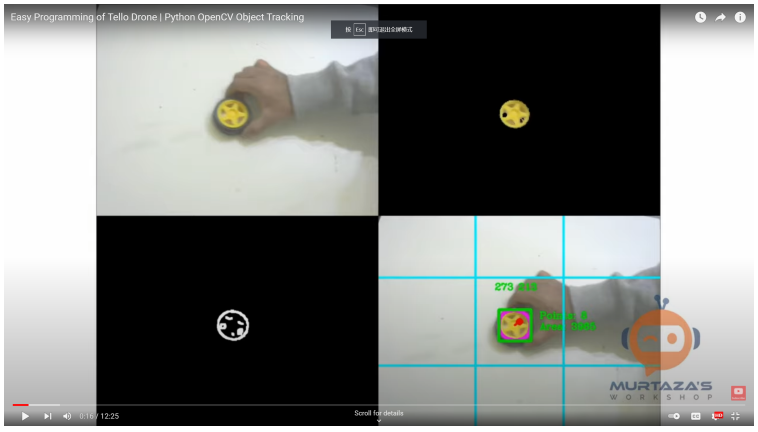


Figure 17: Object tracking demonstration

Another test is based on the mission pad in the Tello EDU combo. As is mentioned in the previous section, there are functions to have the drone tracking a specific mission pad and move towards some specific position within the reference frame of the mission pad. However, it turned out that when operating those built-in commands, the drone could not receive any other commands. In this way, I just gave up using the built-in functions and tried to apply our own tracking policy to Tello EDU.

4.2 Test with control policy

The example test built a basis for my further test with control policy. In this test, I used the bottom camera to obtain the image and it happened that the decoder provided by djitellopy had some issue with decoding the bottom camera. Sometimes, in addition to the figure it obtained, there are some digital gibberish below. To solve this problem, since I know the resolution of the bottom camera is 320×240, the matrix of the image should be in the shape of 320×240×3 and any elements out of this matrix will be deleted.

Recalling Section 2, since I cannot apply very low-level control (control the rotation speed of each of the rotor) to the drone, I just applied the tracking policy discussed in Section 2.2. To simplify the problem, I just considered it as a 2D problem without a landing process, which means

$$\mathbf{x}_p^T = (x_p, y_p, v_{P_x}, v_{P_y}). \tag{37}$$

At the same time, in order to apply the control policy, I had two general assumptions.

1. There is a linear relationship between pixels and centimeters:

   I made this assumption because the only way we had to get the distance towards the landing platform is the distance between the center of the image and the center of the contour center. Since all the states of the drone directly read by the built-in functions are in centimeters, we had to find a relationship between pixels and centimeters. Assuming the drone is flying at a constant height, there should be a linear relationship between them

2. There is a linear relationship between fb,lr in send_rc_control(lr,fb,ud,yv) with the acceleration of the drone

   I did several test to find the relationship between fb, lr and acceleration but it seems that the accelerometer on the drone is not that accurate and has a larger noise. However, during the test, I found that changing fb and lr is actually changing the roll and pitch angle. Ideally, if the air resistance is not that large (flying at a low speed), we can consider it as keeping the drone fly at a constant acceleration. Thus, there should be a linear relationship between them.

Once again, recalling Section 2.2, I need the speed of the drone and the moving platform to fulfill the control policy. However, it's obvious that there is no direct way to measure the speed of the platform. Thus, I applied the observer discussed in Section 2.4 to estimate the speed of the moving platform based on its position change. Similar to the content discussed in Section 2.4, I created a new global axis based on the beginning point of the drone and used

it to update the drone state and the platform state. Time step is obtained by recording the time spent each loop (assuming there's no delay between receiving the order and operating the order). The logic of the code is just the same as is shown in Fig. 12, which is easy to implement based on the test example code. The main difficulty of the coding part is to find the linear parameter between pixels and centimeters as well as between fb,lr with acceleration. The only solution to it is to do a large number of tests. Fortunately, I just found a good combination of parameters to keep the state from diverging and the drone succeeded in tracking a moving platform. The place to change the parameters are underlined in Fig. 18. The demonstration videos have also been uploaded to google drive.

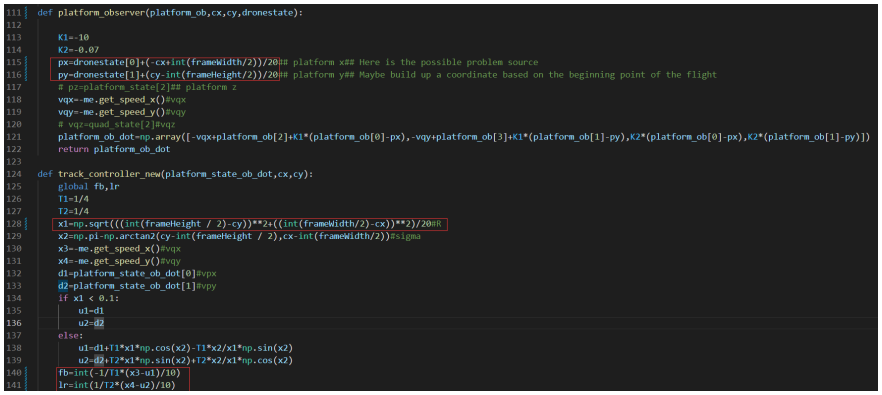


Figure 18: Place to change the parameters

5 Conclusion and Future

Based on all the discussions above, I've succeeded in applying the control policy to Tello EDU drone and have it track a moving platform. Unfortunately, due to time limit, I didn't have enough time to work on the multi-agent case study. At the same time, I just found one good combination of linear parameters but I can't promise whether it is the best combination. At the same time, for the pixel-centimeter relationship, there should be some more precise ways, such as using stereo cameras. For future study, I hope this control policy can be applied to quadrotors which can be controlled at lower level so that the entire system can be tested. At the same time, I hope, if applicable, a further multi-agent test can be accomplished with Tello EDU.

It's my great honor to participate in this project and I hope future student can continue working on it.

References

[1] H. Voos, "Nonlinear control of a quadrotor micro-uav using feedback-linearization," in *2009 IEEE International Conference on Mechatronics*, 2009, pp. 1–6.

[2] H. Voos and H. Bou-Ammar, "Nonlinear tracking and landing controller for quadrotor aerial robots," in *2010 IEEE International Conference on Control Applications*, 2010, pp. 2136–2141.

[3] V. N. Dobrokhodov, I. I. Kaminer, K. D. Jones, and R. Ghabcheloo, "Vision-based tracking and motion estimation for moving targets using small uavs," in *2006 American Control Conference*. IEEE, 2006, pp. 6–pp.