Drona Aviation: Pluto Drone Swarm Challenge

Inter IIT Tech Meet 11.0

1 PS Description

The problem statement comprises of 3 tasks:

1. Develop a python wrapper for the nano drone - the Pluto.
2. Hover the drone at a particular height and moving the drone in a 1x2 m rectangle.
3. Pluto Drone Swarm.

Our solution is built to meet all the expectations given in the tasks listed above. **Keyboard control** included in our python wrapper enables the user to move the drone in any direction, back-flip the drone and **get the values from drone sensors** like height, vario, gyroscope data, magnetometer data, battery, etc. In addition, the PID values are fine-tuned with the help of techniques like the **Ziegler–Nichols** method and extensive testing, thus enabling it to hover stably over a point. We have successfully employed techniques like **multiprocessing** and **multithreading** to run the codes efficiently and ensure our codes' best performance and **reproducibility**. Good coding practices are followed and comments/documentation is included everywhere to make codes easy to understand and maintain/update.

2 Experimental Setup

1. All the experiments and testing on the drone are done in a **closed environment**. The drone is quite lightweight and, thus, might not stabilize in open conditions. It should also be ensured that air movement is restricted in the room.
2. A 6cm square ArUco marker of type 4x4 and id '0' , is used to detect the pose of the drone. A Lenovo 300 FHD 90-degree FOV camera is used for detecting the ArUco tags. Similar marker is used for task 3.
3. The floor is covered with white chart paper to ease the detection of the ArUco markers and ensure continuous detection during the flight.

3 Proposed Solution

3.1 Task 1

Figure 1 shows the overall flow structure of our solution for Task 1. The python wrapper we built can handle all the tasks that the drone can perform when used with the Pluto Mobile App. The functionalities are coded in a class named pluto which is wrapped using `enforce.py`. As seen in the flowchart 1, we first connect the drone Wi-Fi, initiate a Pluto class object, and call `client.connect()` to establish the connection between the drone and the laptop. `Socket` is used to establish the connection to send and receive data packets. We can call `client.trim()` to stabilise the drone and set the trim values. Other functionalities can be called similarly, for example - `client.arm()`, `client.flip()`, `client.pitch_speed()`. Exhaustive list of all the commands defined can be found here.

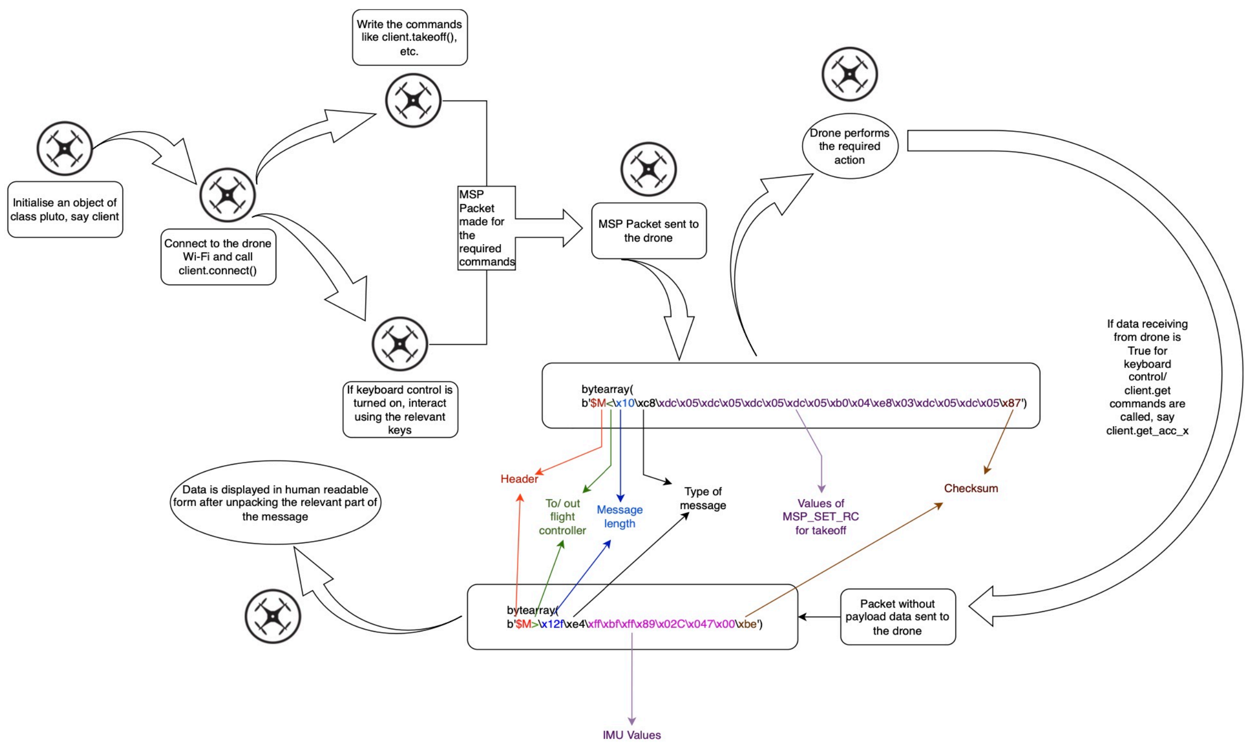
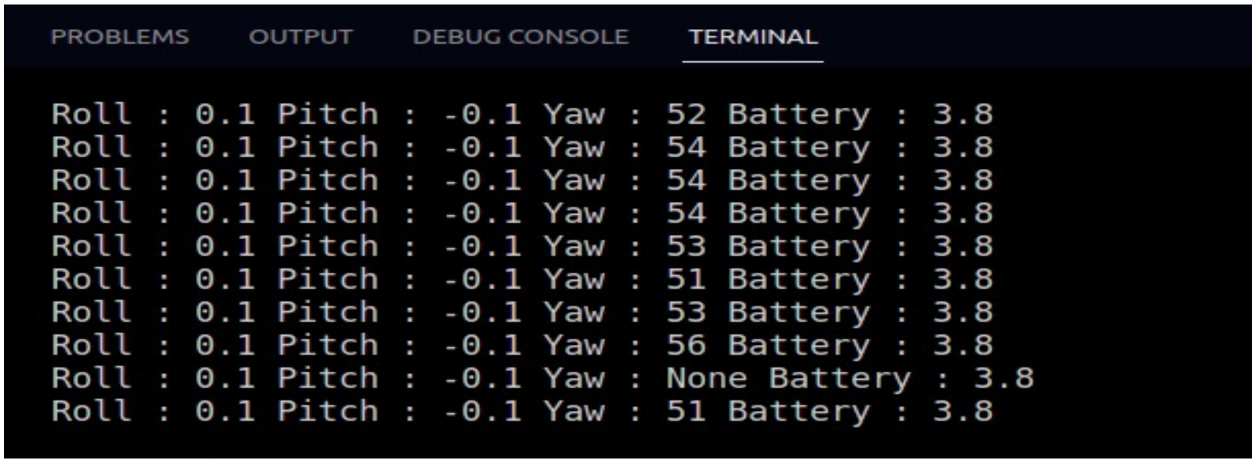

Initialise an object of class pluto, say client
Connect to the drone Wi-Fi and call client.connect()
Write the commands like client.takeoff(), etc.
If keyboard control is turned on, interact using the relevant keys
MSP Packet made for the required commands
MSP Packet sent to the drone
Drone performs the required action
bytearray(b'$M<\x10\xc8\xdc\x05\xdc\x05\xdc\x05\xdc\x05\xb0\x04\xe8\x03\xdc\x05\xdc\x05\x87')
Header
To/ out flight controller
Message length
Type of message
Values of MSP_SET_RC for takeoff
Checksum
If data receiving from drone is True for keyboard control/ client.get commands are called, say client.get_acc_x
Packet without payload data sent to the drone
bytearray(b'$M>\x12f\xe4\xff\xbf\xff\x89\x02C\x047\x00\xbe')
IMU Values
Data is displayed in human readable form after unpacking the relevant part of the message

Figure 1: Flow diagram of Task 1.

We have also implemented a keyboard control in our python wrapper which can be called as `client.keyboard_control()`. It enables us to control the drone with our keyboard, with different keys mapped for different functions, for example D to yaw right, W to increase height, P for flip, spacebar to arm the drone, etc. Exhaustive list of the key mappings of all the functionalities can be found here. Additionally, we can receive **data from the drone** while it is flying by passing additional argument to the function as `client.keyboard_control(True)`. This prints the data continually received from the drone as seen in Figure 2. Other data can be obtained from the drone by simply using the `client.get` commands. Thus, our python wrapper not only covers **all the functionalities given in the ROS package but also adds additional functionalities to it.**

3.2 Task 2

Figure 3 shows the overall flow structure of our solution for Task 2. We have used multiprocessing in task 2 to improve the performance and take advantage of parallelism. We have defined 2 processes; wherein the first process, we detect the marker and return the pose of the drone. This is then passed to `PIDmain.py` to calculate the values of roll, pitch, yaw and throttle, enabling the drone to reach the destination.
Roll, pitch, yaw and throttle values are composed into one MSP Packet and passed onto the drone. This helps us to optimize the process and speed it up. The MSP packets are sent in the same way as in Task 1. This process keeps looping.
The values for PID are decided with the help of the Ziegler Nichols Method [1], which was



```
Roll : 0.1 Pitch : -0.1 Yaw : 52 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 54 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 54 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 54 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 53 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 51 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 53 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 56 Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : None Battery : 3.8
Roll : 0.1 Pitch : -0.1 Yaw : 51 Battery : 3.8
```

Figure 2: Data Receiving with keyboard control.

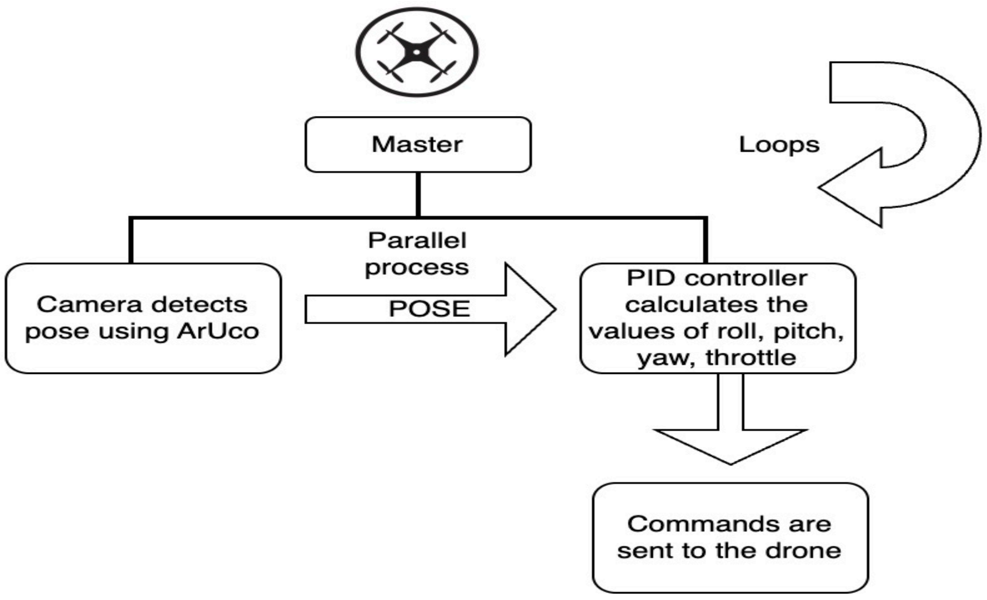

Master
Loops
Parallel process
Camera detects pose using ArUco
POSE
PID controller calculates the values of roll, pitch, yaw, throttle
Commands are sent to the drone

Figure 3: Flow chart for Task 2.

fine-tuned and extensively tested. The drone initially takes 2-3 seconds to stabilise itself after take-off.

3.2.1 PID Control

We use the **absolute error** between the detected pose and the intended target as the error function. The drone starts fast, and as the error goes down, it slows as expected. Later, as it gets close to the point, the magnitude of commands sent becomes quite low, and the drone moves slowly, especially in roll due to the high derivative, until the integral error takes over and helps the drone to reach the exact point.
The values for derivative were kept slightly higher during the testing, especially for the roll, as the movement for the roll was quite aggressive, and the risk of propeller and body damage was high due to the confined space. Thus, we were conservative while testing it.
In the case of hovering, the drone tries to maintain its position and returns to the point if disturbed. Since we have applied PID for the z-axis too(height of the drone), the drone maintains its height and hovers stably.
A few points apart from the corners of the rectangle are defined to move the drone in a rectangle. The drone starts at one point and then reaches each point one by one. The drone must reach each point several times, ensuring stable movement and that it does not cover any point due to overshoot or luck. Once it hovers over a point for sufficient time, the target of the drone is updated, and the drone moves accordingly.
The position PID control of the drone is implemented based on the following:

$$\mathcal{X} = k_{Px}(X_d^G - X^G) + k_{Ix}\int_0^t (X_d^G - X^G)dt + k_{Dx}(\dot{X}_d^G - \dot{X}^G)$$

$$\mathcal{Y} = k_{Py}(Y_d^G - Y^G) + k_{Iy}\int_0^t (Y_d^G - Y^G)dt + k_{Dy}(\dot{Y}_d^G - \dot{Y}^G)$$

$$\mathcal{Z} = k_{Pz}(Z_d^G - Z^G) + k_{Iz}\int_0^t (Z_d^G - Z^G)dt + k_{Dz}(\dot{Z}_d^G - \dot{Z}^G)$$

$$\Theta = k_{P\theta}(\Theta_d^G - \Theta^G) + k_{I\theta}\int_0^t (\Theta_d^G - \Theta^G)dt + k_{D\theta}(\dot{\Theta}_d^G - \dot{\Theta}^G)$$

The pitch and roll commands are given by the following:

$$\begin{bmatrix} \mathcal{P} \\ \mathcal{R} \end{bmatrix} = \begin{bmatrix} \cos(\Theta^G) & \sin(\Theta^G) \\ -\sin(\Theta^G) & \cos(\Theta^G) \end{bmatrix} \begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \end{bmatrix}$$

where,
$X_d^G -$ Desired global X position (pixels)
$Y_d^G -$ Desired global Y position (pixels)
$Z_d^G -$ Desired global Z position (m)
$\Theta_d^G -$ Desired global Y position (pixels)
$\mathcal{P} -$ Pitch command
$\mathcal{R} -$ Roll command

$\mathcal{Z}-$ Throttle command
$\Theta-$ Yaw command
Thus, we complete Task 2 satisfactorily as the drone moves stably in a controlled path, with minimal variation and overshoot from the designated path and capable of hovering over a point. The PID implemented by us is also **responsive enough** and covers a 1x2 rectangle in under a minute. The values of PID are fine-tuned for the Pluto drone and ensure reproducible behaviour.

3.3 Task 3

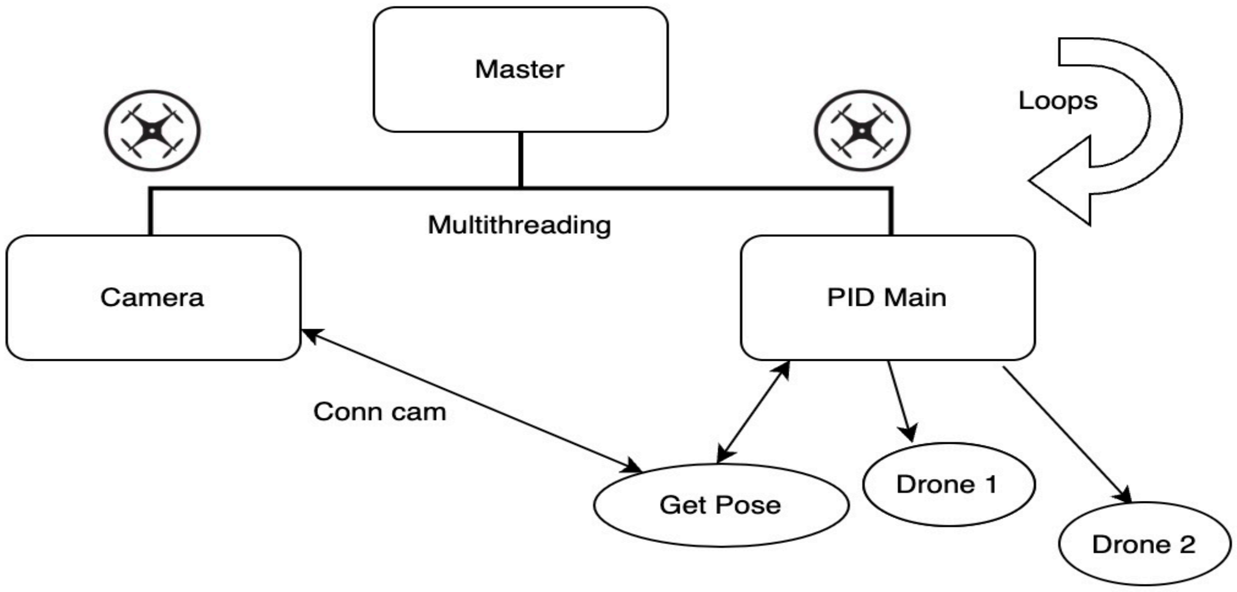
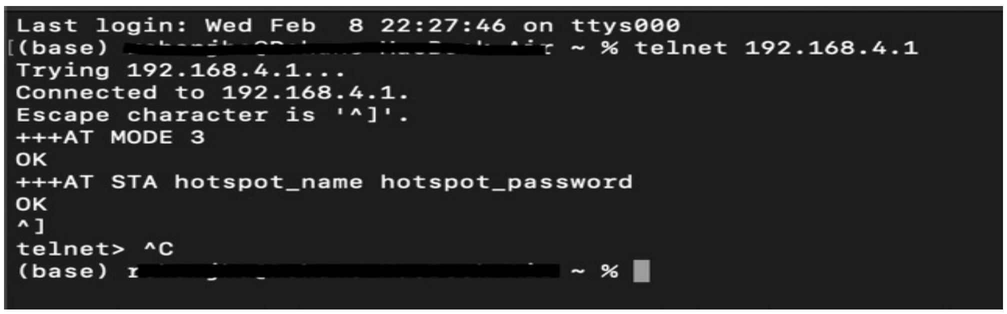


Figure 4: Flow chart for Task 3.

Task 3 is an extension of task 2. We have employed **multi-threading** to get the best performance. The overall approach can be seen in the Flow chart 4 above.
Firstly to connect the two drones, we first start the hotspot of the laptop. Ensure that telnet library of python is installed in the device and run the commands as shown in the figure 5. We have also made a **shell script** for the same which can be run on Linux machines directly



```
Last login: Wed Feb  8 22:27:46 on ttys000
[(base) [redacted] ~ % telnet 192.168.4.1
Trying 192.168.4.1...
Connected to 192.168.4.1.
Escape character is '^]'.
+++AT MODE 3
OK
+++AT STA hotspot_name hotspot_password
OK
^]
telnet> ^C
(base) [redacted] ~ %
```

Figure 5: Instructions for connecting 2 drones.

to connect the two drones.
We first run `marker.py`, where we obtain the pose of the two drones, which is composed in the form of a dictionary. This dictionary is sent to `PIDmain.py` where we have drone 1 and drone 2 in different threads. The error is calculated on basis of the pose received and accordingly the PID commands are sent to the two drones.

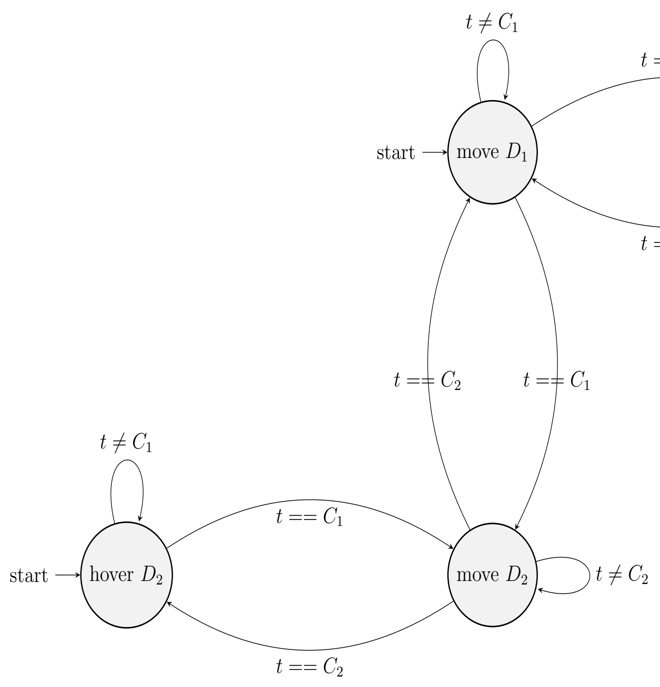


Figure 6: FSM for Task 3.

The algorithm for moving the two drones together in the rectangle can be seen in the **functional state machine(FSM)**(Figure 6). Initially, drone 1 starts moving and drone 2 hovers. C1 symbolises the set of corners for drone 1(D1) and C2 is the set of corners for drone 2(D2). They will be congruent sets except that drone 2 lags by 1 point. As drone 1 reaches a point in set C1, the FSM transitions to hover for drone 1 and move is set True for drone 2. Again, as drone 2 reaches one of the corner points, move is set True for drone 1 and drone 2 hovers. To identify the corner points efficiently, we have divided the rectangle into 4 subarrays, where we pass each subarray and the first and last points are corner points. This keeps repeating as seen in the FSM.

Thus, we solve the above tasks with the most feasible methods available and optimised for the best performance.

4 Challenges faced

1. The Pluto drone weighs just 50 gms. Thus, it is quite sensitive to air moving around, which we might not consider appreciable initially. It results in false instability and **drifting** of the drone in one direction, which results in wrong trim values for the drone.
2. The **trim values** of the drone were observed to change between charge cycles and if the battery is replaced with another battery. This resulted in the loss of a lot of time since trim values had to be set repeatedly. Also, battery charge had to be used to set the trim instead of testing the drone. This made reproducibility a challenge.
3. The **drone's battery life** is just 9 minutes, making it difficult to test the drone continuously. Charging took around 1-1.5 hours, which weighed down the overall development and experimentation work.
4. **Lack of a good camera** and lighting conditions made continuous ArUco tag detection difficult. Apart from the motion blur due to the drone's movement, the flickering detection of the tag caused problems in PID calculation. Plain white sheets were placed on the ground to improve the detection of the ArUco tags.
5. Since the propellers are made of thin plastic, **propeller damage** was quite high. This limited our experiments. Also, our team faced unexpected hardware failures during our development(replacements for the same were shipped later) which slowed our testing down.
6. **Accumulating delays** and code failure due to compiler intricacies and slow hardware processing was also a great challenge during the development phase.

5 Future Works

1. Drones are conventionally controlled using joysticks, remote controllers, mobile applications, and embedded computers. A hand gestures-based ML model can be developed to **control the drone with hand gestures**. The main challenge is that computer vision techniques are currently quite slow for the drone to continuously receive signals after processing live video. However, some code for the same has been developed by us.
2. The code developed by us currently runs at **10fps**. Further optimization can be done to improve the fps to around 30 fps.
3. Currently, the code is not adaptable to the swarming of more than two drones. The code needs to be changed to do the same. We can make efforts towards **generalising the code to swarm n-drones** without changing it.
4. Drona aviation also provides a **camera module** for the Pluto drone. We can work towards integrating it in the python wrapper and providing keyboard control to receive the video feed onto the laptop.
5. The code developed by us accounts for rectangles. The coordinates for the rectangle

are hard coded. However, we can work towards **automatic coordinate generation** for complex geometries after we provide the pixel range and fine-tune the PID for the geometry. This may be achieved by the use of reinforcement learning and use of simulations to fine-tune PID values.

6. VICON motion capture systems and other similar techniques can be explored for swarming of n-drones since the field of view(FOV) is a major restriction in the case of normal RGB cameras. However, these systems are quite expensive.

6 References

1. https://github.com/DronaAviation/Magis
2. https://github.com/DronaAviation/pluto-ros-package
3. https://www.dronaaviation.com/forums/
4. https://github.com/damiafuentes/DJITelloPy
5. https://github.com/nittvikas/PlutoX
6. https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
7. https://eng.libretexts.org/Bookshelves/IndustrialandSystemsEngineering/Book3AChemicalProcessD
Integral-Derivative(PID)Control/9.033APIDTuningviaClassicalMethods
8. https://github.com/CuriosityGym/Drone-API
9. https://move.rpi.edu/sites/default/files/publication-documents/2018-14.pdf
10. https://stackoverflow.com/questions/51584562/calculating-the-distance-and-yaw-between-aruco-marker-and-camera
11. https://github.com/CuriosityGym/Drone-API