
# EE202C Project Report

## IOT Smart Drone

*Peidong Chen(204432674), Yang Yang(804522285), Yitian Hu(904516321),*
*Jiayu Guo(504513188)*
*Advisor:Professor William Kaiser*

School of Electrical Engineering
University of California, Los Angeles
December 2015

CONTENTS

## 1.1 *Purpose and Scope*

Our goal is to design a robust smart drone system on Intel Edison IOT platform from the fundamental point to a reliable demo and further elaborate its functionality to photography, videography, goods delivery and multi-drone communication. It is a drone system integrated with Edison microprocessor for IOT communication, a couple of sensors used for position detection and flight activity control. Another microprocessor beagleboard serves as motor controller and sensor interface. On the software section, the online cloud server Ubidots stores and analyses our IoT data. Pictures and videos could be stored on Intel Edison at first and then sent with commands from Linux platform.

The purpose of our drone design is come from the heat robotic trend in both the research and industry area. A couple of successful companies have invested great amount of energy and resource on UAV and unmanned cars. There are big companies like Google, Amazon and ever Facebook involving in these design. In china, a company called DJI is leading the high-technology revolution in drone platform. All the information indicates that in the next few years, an intelligent world combines the machine with our human being together is going to uncover its veil. Yet until now many of these robotic design is based on how to intelligently analyze the surrounding environment data precisely. Internet of things as a collection of separate nodes are slightly different from these nodes, making all the devices online in one super-brain and coordinating them with each other. It is exciting that one day several of our single drones are able to compose of a drone network and even communicate with various IoT components.

The functionality of our current version is to navigation, tracking and image processing recognition. The realization of the fundamental system is built on Intel Edison to manually receive commands from cloud server and continuously feedback flight information to the server. A GPS module, a 9dof sensor and a camera module embedded on beagleboard microprocessor is to collect current location information and detect the surrounding environment. Image or video recognition is used to detect any specified target in a certain range by OpenCv Camshift algorithm and the location of the flight would be roughly set by the GPS signal. This application has the potential to track criminal, traffic flow evaluation and multi-drone communication.

## 1.2 *Poject Executive Summary*

### 1.2.1 *System Overview*

The system-level diagram demonstrates how those separate functional modules work with each other clearly. The beagle board receives the commands from Intel Edison and sends corresponding PWM speed signals to the motors. The motors then turn on those propellers to fly. At the

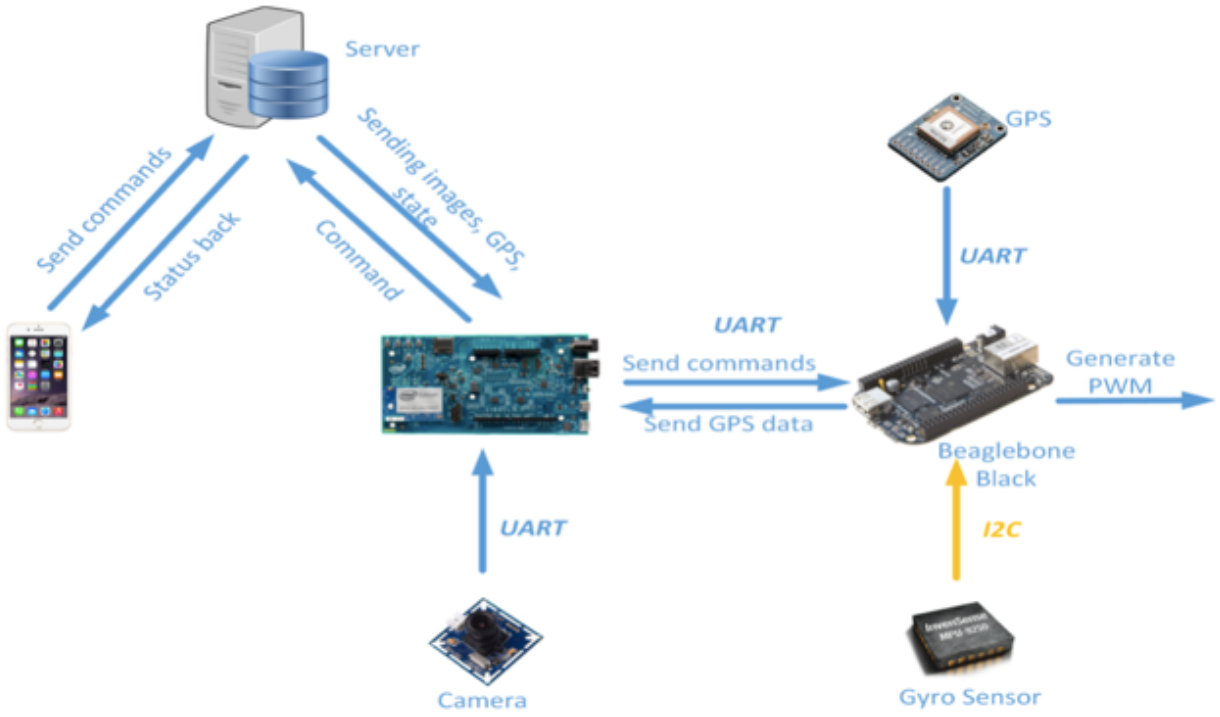



Figure 1: System-level diagram demonstrates

meantime, the real-time GPS data link and 9dof gyro, acceleration and magnitude information are transmitted to the beagle board. Within milliseconds delay, these data will arrive at Intel Edison board and quickly be saved on the cloud server as there are requests on that front-end. To better control the whole system, a user interface application is developed on the IOS as shown in the below picture.

Our drone flight controller includes two types of setting instructions. Under manual instructions, the drone would fly to certain directions according to the instruction buttons clicked by the user. These instructions are ”Left, Right, Forward and Backward” for direction determination and ”Up and down” for speed control. Under automatic flight control, by clicking ”suspend”, the flight would stay stable in the air. With further instructions still need to be attached, the automatic flight drone would move to specific direction and detect the nearby environment. The automatic flight mode is based on PID feedback loop control and collision avoidance algorithm( continued to be integrated). A second remote controller app is designed to test four PWM motors simultaneously with various speeds just by dragging the circle to different positions. So in this part, we use a slider to control the speed of each motor. We can change the value of sliders(including button maximum, minimum and medium), and send these value to server. Edison, using C code, can get these information from web. According to these values, it can then change the speed of motor. We also use AFNetworking to send information to server.

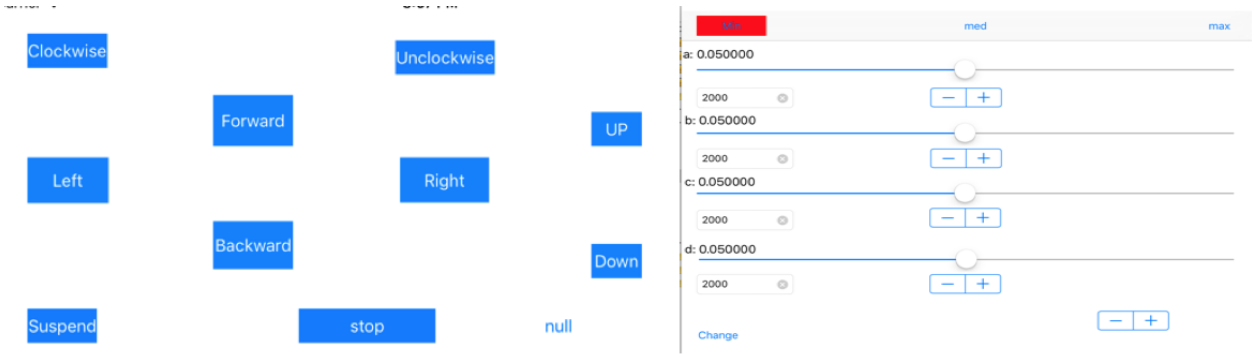
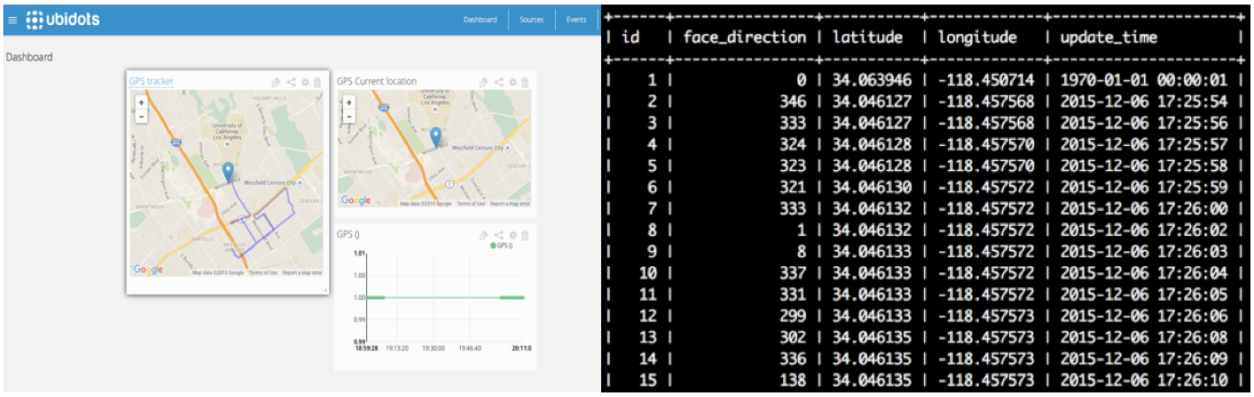



Figure 2: Drone flight controller screenshot

On the server end, the transmitted data would be listed on the website with different types of forms. On the GPS track map, all the past paths could be shown as with the real-time nodes shown in the map. A list of raw data values such as longitude and latitude would also be included on the website server and stored for further analysis.Besides of that, all the data and information is sent to another sever in the research lab to keep a safe copy of raw data by HTTP. These information would contain security information as WIfi IP address, current user, and flight raw data. Two design of server cloud, one is taken for our test and further administrative user and the former is designed for remote user interaction.




| id | face_direction | latitude | longitude | update_time |
|---|---|---|---|---|
| 1 | 0 | 34.063946 | -118.450714 | 1970-01-01 00:00:01 |
| 2 | 346 | 34.046127 | -118.457568 | 2015-12-06 17:25:54 |
| 3 | 333 | 34.046127 | -118.457568 | 2015-12-06 17:25:56 |
| 4 | 324 | 34.046128 | -118.457570 | 2015-12-06 17:25:57 |
| 5 | 323 | 34.046128 | -118.457570 | 2015-12-06 17:25:58 |
| 6 | 321 | 34.046130 | -118.457572 | 2015-12-06 17:25:59 |
| 7 | 333 | 34.046132 | -118.457572 | 2015-12-06 17:26:00 |
| 8 | 1 | 34.046132 | -118.457572 | 2015-12-06 17:26:02 |
| 9 | 8 | 34.046133 | -118.457572 | 2015-12-06 17:26:03 |
| 10 | 337 | 34.046133 | -118.457572 | 2015-12-06 17:26:04 |
| 11 | 331 | 34.046133 | -118.457572 | 2015-12-06 17:26:05 |
| 12 | 299 | 34.046133 | -118.457573 | 2015-12-06 17:26:06 |
| 13 | 302 | 34.046135 | -118.457573 | 2015-12-06 17:26:08 |
| 14 | 336 | 34.046135 | -118.457573 | 2015-12-06 17:26:09 |
| 15 | 138 | 34.046135 | -118.457573 | 2015-12-06 17:26:10 |

Figure 3: GPS teack map and bash output of GPS

### 1.2.2 *Design Constrains*

Our drone flight controller includes two types of setting instructions. Under manual instructions, the drone would fly to certain directions according to the instruction buttons clicked by the user. These instructions are ”Left, Right, Forward and Backward” for direction determination and ”Up

and down" for speed control. Under automatic flight control, by clicking "suspend", the flight would stay stable in the air. With further instructions still need to be attached, the automatic flight drone would move to specific direction and detect the nearby environment. The automatic flight mode is based on PID feedback loop control and collision avoidance algorithm( continued to be integrated). A second remote controller app is designed to test four PWM motors simultaneously with various speeds just by dragging the circle to different positions. So in this part, we use a slider to control the speed of each motor. We can change the value of sliders(including button maximum, minimum and medium), and send these value to server. Edison, using C code, can get these information from web. According to these values, it can then change the speed of motor. We also use AFNetworking to send information to server.

### 1.2.3 *Future Contingencies*

Our original objective is to design a kind of smart drone based on Intel Edison IOT platform. As a smart drone, it should be able to fly autonomously and communicate with another smart drone to complete the task like taking pictures, following and detection. The drone has the capability to navigate using GPS/Wifi and to arrive at a specific destination. Furthermore, functionality as an anti-drone drone, which control other anonymous drones into a restricted area, and transformer-drone, drone can intelligently change its shape could also be extended as the size of the drone can be reduced when embedded with smaller integrated Intel Edison IoT board.

The robust of our system also is another figure of merit needed to be argued. The communication part is still imperfect when transmitting and receiving data simultaneously. The stability of our drone for accurate PID value and safety protection strategy has to be found an evaluation approach.

The robust of our system also is another figure of merit needed to be argued. The communication part is still imperfect when transmitting and receiving data simultaneously. The stability of our drone for accurate PID value and safety protection strategy has to be found an evaluation approach.

## 2.1 *System Hardware Architecture*

### 2.1.1 *Hardware Components:*

Moving from zero to one, our entire drone system is totally built from several individual electronic and mechanical components, and later assembled by hand manually. The system design is first considered from

whether to buy a commercial drone plane or a DIY drone implemented by ourselves. Our idea is to understand the embedded system's functionality better in great details and to find a system which is acceptable to simple modification between the normal embedded system and Iot embedded system design. Thus, a flexible DIY drone is selected as our first design plan.

To those who are interested in reliable and well-designed commercial drone system, we also offer some good places to research and built their system or interface application based on those tested drone products. DJI is a world-leading unmanned aerial vehicles recently founded in China. It provides good and reasonable price for developers to design new videography and photography platform. A good news is that it opens its SDK for Android or IOS apps and with its cutting-edge technology, the developers could customize its products in very different and nice ways. Other companies like 3D robotics and Parrot AR.drone also show a good standpoint design for us to get familiar with drone control and test. And drone code is a nice and cute website to learn some basic drone software knowledge. Since IoT system, the drones are usually connected with indoor or outdoor wireless hotspot, traditional RF communication from remote to the drone is now substituted by a laptop or our personal phone. The antenna on the drone is now replaced by Wifi module or GPRS modem on the platform to further communicate with nearby devices.

Our design goal is to construct our own unmanned aerial vehicle architecture and implement this drone with our preowned Edison development board.

First thing in our consideration is the size of our drone. The frame for our drone is decided by the force needed to control our drone. In order to figure out the reasonable size of our drone. The first constraint is our Intel Edison Arduino board. It is a board with at least 10cm *20 cm area, accounting for almost the entire size of a micro-drone. Thus, even though originally we are supposed to create a drone with super lightness and agility, it seems that only a large size UAV is tolerant for us to setup our Arduino Kit. A small drone requires us to put Intel Edison chip on a self-designed PCB board with necessary interfaces. It could be a possible project carried out in the next stage. Second, considering the moment of force is the primary drive strength to keep the drone moving. The stability of drone now is maintained by the force obtained from the speed of propellers multiplying the length from the propeller to the geometric center. Obviously, a larger size of drone alleviates the demand of control force from propeller and hence the equilibration state of the whole system is easier to reach with the same adjustment from our speed motor. Below

is the structure we purchased from Amazon, providing enough room for us to hold the Intel Edison board, our sensors, battery distribution module and along with multiple periphery blocks.

We have already widely accepted the core concern lying between the performance of drone and the functionality of drone. That is the power supply or the batter life time. A normal Lithium battery can support its corresponding size UAV to work between 10 to 30 min in sky. Thus, it is incredibly critical to reduce every bit of power consumed in the drone system to build a highly power efficient project. Also, perhaps a battery level detection module is necessary for a commercial drone since the duration in the air is a short unsure period. Because without any guarantee for the drone landed on the ground automatically when the battery is low, it increases the risk of the system safety. Take our battery as an example. The battery module is a 11.1V supply Lithium rechargeable battery module, offering enough voltage to our ESC(electronic speed controller). Its 3000mAh storage makes sure that our total system can work at 200A current consumption at around 20min. (AMP=3000mAh* 30C= 90A*h) So before choosing the matched battery online, it is important to first calculate the approximate amount of current dissipated in your design and find the voltage requirement for other module. A voltage level shifter may be included when the design become complex. Also a four-connector power distribution board is necessary to produce four interface for our ESC modules.

After the power issue, the next thing come to our mind is the propeller. Unlike those motors on the cars or RF ship, the force generated by our drone is significantly larger but also with precise value. To satisfy this criterion, the high-speed motors from the drone is no longer directly connected to the battery and microprocessor. A ESC(electronic speed controller) is the bridge between the motors and the microprocessor as well as the battery. It absorbs the power provided from the power distribution and transforms it to the motors. Even amazing is that with the ESC module, the power of the Intel Edison board or other microcontroller can be turned on as the BEC(battery eliminator circuit) on the ESC reversely feed back power to our microprocessors. Second, the signals of controlling those high-speed motors are now translated on the ESC, which could also be viewed as an intermediate control module associating the instructions from processors with the motors through the ESC inner firmware. One thing to notice when selecting our ESC module is the current tag indicating how large the maximum current could be tolerant for the ESC module. This current requirement must match the current delivered from the center power module, otherwise the ESC modules would easily get burned because of the overheat in the circuits.

The high-speed motor has several features that is different from traditional low-speed motor. Its brushless motor's inner core physical theory is its magnet array inside the motor is excited by the current going through the motor. The directions of current determine the rotation of the motors. And the speed is proportional to how current you put into the motor. The figure of merit of brushless motor is Kv. Kv is the motor velocity constant, measured in RPM per volt. Basically the higher the kv merit, the motor can reach to higher maximum rotation speed. For the propellers, they can be divided into two types of propellers. The Clockwise propellers and Counter Clockwise

propellers. Those two different kinds of propellers work together perfectly as to provide correct lift force for the drone. Each type of propeller mush be installed in the opposite direction. Due to the weight of our drone and the high speed of our drone, it is dangerous to fly the drone without any protection method, a guard protection protector is surrounded our designed UAV.

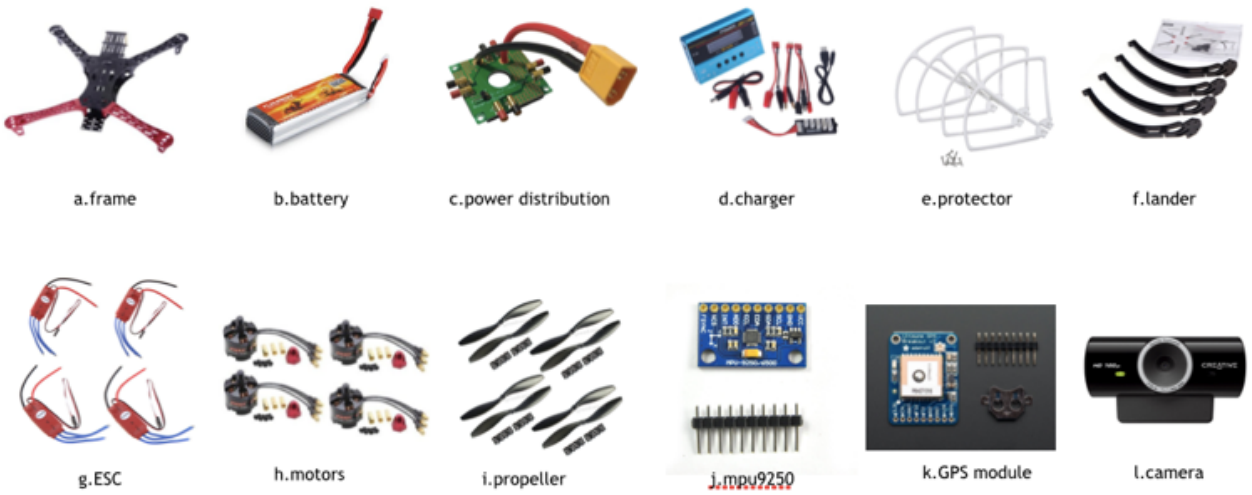



Figure 4: Component for our drone

a.Frame structure

GoolRC New HJ MWC X-Mode Alien Multicopter Quadcopter Frame Kit (Red Black) b. Battery module Floureon 11.1V 3000mAh 30C Li-Polymer/Lipo Battery Packs RC Battery with Dean-Style T Connector for RC Airplane, RC Helicopter, RC Car/Truck, RC Boat c.Power distribution Quadcopter Power Distribution Board XT60 XT-60 20a Quad Mutlicopter 3.5mm

d.Battery charger

Neewer B3 AC 100-240V 2S-3S Cells 7.4V 11.1V Lipo Battery Balancer Charger

e.Protector:

Mudder 4 PCS Syma X8C, X8W, X8G Plastic Quadcopter Propeller Guard Protection Frame (White), Necessary Parts

f.lander

Andoer Tall Landing Gear Skids Black for DJI F450 F550 SK480 Qudcopter Multirotor

g.ESC module

Andoer 4Pcs Simonk 30AMP 30A SimonK Firmware Brushless ESC w/ 3A 5V BEC For DJI F450 Align TREX 250 450 RC Multicopter Qudcopter Helicopter ESC Part

h. Brushless motor

Emax Mt2213 935kv 2212 Brushless Motor for DJI F450 X525 Quadcopter Hexcopter(pack of 4pcs)

i.Propeller

10x4.5" 1045/R CW CCW orange Propeller,Multi-Copter clockwise rotatingcounter

j.Sensors:

MPU9250:Diymall Mpu-9250 9dof Module Nine-axis Attitude Gyro Compass Acceleration Magnetic Field Sensor

k.GPS:Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version3

l.Camera:Creative Live! Cam Sync HD 720P Webcam

Intel Edison is a good board to support basic IoT functionality as Wifi and Bluetooth communication. However, to extend our system to a drone with super stability, we find Intel Edison's interface has limited advantage over other flight control boards such as Beagleboard and Raspberry Pi. Their interface communication system is more reliable and with more UART and pins. Compared to Edison, although it provides a couple of I2C communication ports, at high-speed communication in 10ms, the signals transmitted to the Intel Edison board are aggressively degraded. Thus, our system would easily get stuck in the middle of programs and the I2C communication would freeze. For safety consideration, UART/ SPI communication between sensors or other periphery device and Intel Edison is preferred. It is interested to research what is happened in the Intel X86 embedded system kernel preventing a stable ideal rectangle waveform to show at the output as the signal changes from high to low. Anyway, in our second version of drone, a Beagleboard is selected to collect all the surrounding data from sensors, and works as a bridge to communicate between Intel Edison, who sends the commands from the cloud, and the Beagleboard who reflects those instructions to hardware. A few types of sensors are utilized in

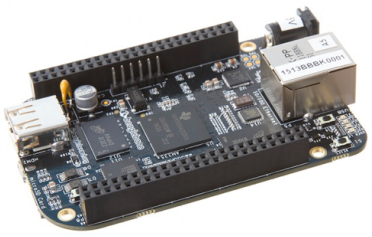


Figure 5: Beagleboard (BeagleBone Rev C)

our system. The MPU9250 is a 9dof attitude gyro compass acceleration and magnetic field sensor, providing the information of rotating speed, acceleration and magnetic field in the nearby field. A similar high-end 9dof could be found here: https://www.sparkfun.com/products/10724 To realize navigation of our automatic drone, it would be discussed in the following chapters that GPS navigation and imaging processing

detection technologies are both adapted in our system. The accurate altitude of drone with the respect to the ground will be evaluated by the ultrasonic sensor. The camera on board is for image processing and detection.

Based on the sensor fusion data we get after processing the raw data of the MPU9250, flat flying control algorithms can be designed. Since during the fly, any small tilt will accelerate the drone, the processed date should be very accurate and real-time, which means that the latency should be as small as possible. And in order to make sure that the drone fly flatly and steadily, we will basically simplify the flying into 9 modes: suspending, ascending, descending, going forward, going back, going right, going left, clockwise rotation and anticlockwise rotation. For going forward, back, right and left, we only need to accelerate two neighboring motors. Specifically, first we record the suspend rotation speed($R_{sus}$) and define the maximum tilt angle for each movement ($e.g.$10). Second, if the sensor tells that the drone reaches10, we adjust all the four motors' speed to $R_sus(1 + sin10)$. And then the drone will not falling down because of a small tilt. For clockwise and anticlockwise rotation, we just need to speed up two opposite motors and speed down other two motors, the drone will rotate.

### 2.1.2 *Assemble Process and Basic Knowledge*

**Assemble Process**

The setup Process of the drone requires to understand the basic knowledge of how the drone could fly and also how to minimize this drone's total weight. There is a amazing TED video to refer from Vijay Kumar[1] to know some basic mathematics of drone flight, although some small typos existed when he shows the propeller rotation.

First, the first question come into our mind is that why these quadrotors could fly in the middle of the air. The above picture of the quadrotors shows that the take-off and stabilization of the drone is determined by the relationship between the rotation speed of these four propellers.

From the fundamental aerodynamics functions, we know: $q = \frac{1}{2}\rho V^2; F = CL \cdot A \cdot q\ V^2$.where p is the density of air and V is the velocity of blades. The lift force on the drone is a function with q, A(area). Also the Reynold coefficient, Mach coefficient and the shape of done have an influence on the final result. Taking the moment of force into consideration, the lift force and the drag force would multiple the arm length R. Thus, velocity acceleration is proportional to $V^2$.

Now we prepare to talk about the flight control behavior of our drone. In the other words, how could the drone change its directions during the flight. In general, the drone has 10 actions: Turning up, down, right, left, forward, backward, clockwise rotating, anticlockwise rotating, suspending and stop. All other nine actions are based on the suspending action, which mans that when we get the command of suspending action, we can start to wait for a small amendment to carry on all other nine actions.

Considering the following image, the four propellers are labelled Propeller1, Propeller2, Propeller3 and Propeller4. And the PWM duty cycle are named P1, P2, P3 and P4.

When the drone is suspending, P1 = P2 = s1 and P3 = P4 = s2. (Supposing P1 and P4 are clockwise, P2 and P3 are anticlockwise. And this image is in Bird's-eye view.)

1. Flying up action :P1 = P2 = s1 + a, P3 = P4 = s2 + a
2. Flying down action: P1 = P2 = s1 - a, P3 = P4 = s2 - a
3. Flying right action: P1 = s1 + a, P2 = s1, P3 = s2 + a, P4 = s2
4. Flying left action: P1 = s1, P2 = s1+a, P3 = s2, P4 = s2+a
5. Flying forward action: P1 = s1, P2 = s1, P3 = s2 + a, P4 = s2 + a
6. Flying backward action: P1 = s1 + a, P2 = s1 + a, P3 = s2, P4 = s2
7. Clockwise rotating: P1 = s1, P2 = s1 + a, P3 = s2, P4 = s2 + a
8. Anticlockwise rotating: P1 = s1 +a, P2 = s1, P3 = s2 + a, P4 = s2

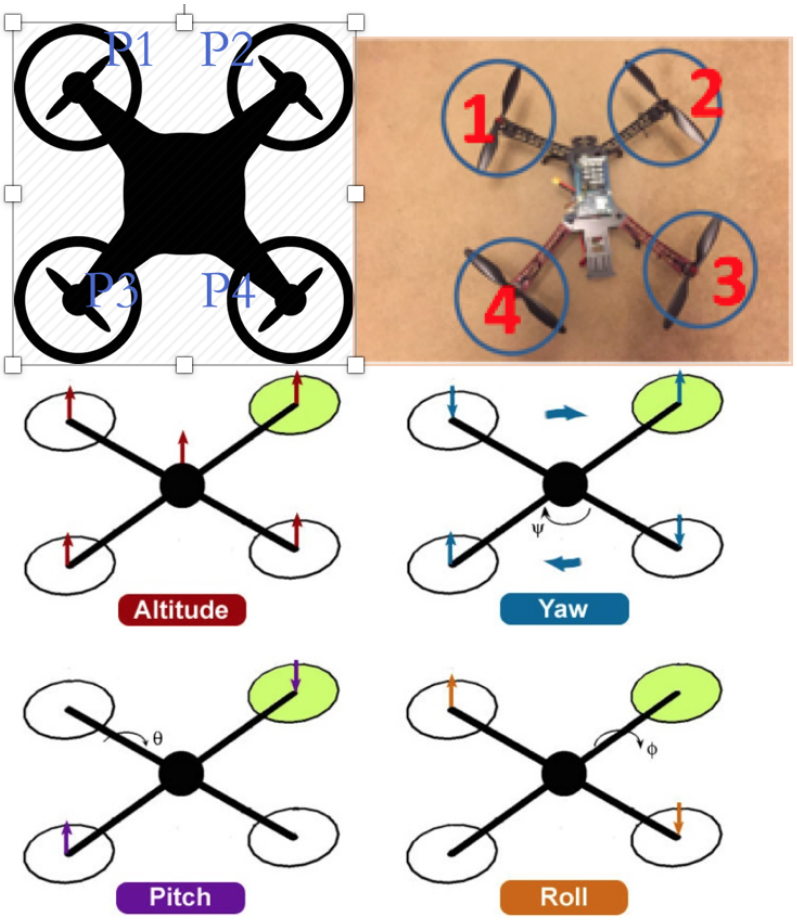



Figure 6: Demonstration of drone propellers

To better calibrate the drone flight behavior, we may add a function called "flight learning", which means that when the drone changes its weight due to modules modification on the drone, we can start the "flight learning" function. Then the drone can learn the PWM variable value, at which the drone can suspend in air.

The geometric symmetry of the drone could relieve our effort on repeating finding the PID parameters as discussed in the later chapter. In order to ensure the geometric center is the same as the gravity center, we hang on the quadrotors and gradually balance the drone by adding weight on the four arms of the drone. First we fix two ends of two opposite wings, and we check whether the other two wings are balanced. Then we fix two ends of two other opposite wings and we check the balance. After the two steps, we can make sure that the drone is correctly assembled.

**PWM Generation**

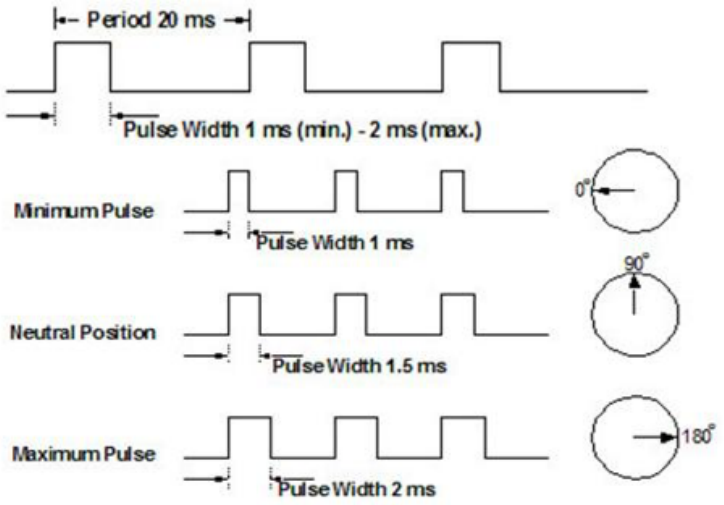



Figure 7: Duty circle of PWM waveform

Since we have known the flight control of our drone is depended on the relative speed values between four motors, the next step is how to interpret these values from microcontroller in a mechanical method. PWM(pulse width modulation) and PPM(pulse position modulation) is solutions for these values. The traditional PWM control signal for a RC ESC is a variable width of 5V pulse. As shown in the above picture, repeating every 20ms, the pulse width during that 20ms period points to a ESC output value, which is the motor's rotation speed. For a traditional PWM system, the minimum pulse width is 1ms and the maximum pulse width is 2ms.

Before writing code to divide duty cycle of our PWM waveform, the first thing is to check whether our hardware supports the resolution goal. Normally, the bits of the timer on board limits how accurate value you could obtain as a PWM waveform. A software PWM waveform generated by GPIO or a fewer bits timer would encounter severe noise in the duty cycle. Because of that, a PWM port with hardware timer is preferred or at least a PWM generated by reusing the fewer bits timer. Lucky, although the Intel Edison board only offers us a 8bits timer on board, both the UNO board and BBB board have a time more than 16bits. The resolution accuracy of our PWM is 0.5us and 4000 levels of duty cycles are divided between 1ms and 2ms.

**PID**

Before writing code to divide duty cycle of our PWM waveform, the first thing is to check whether our hardware supports the resolution goal. Normally, the bits of the timer on board limits how accurate value you could obtain as a PWM waveform. A software PWM waveform generated by GPIO or a fewer bits timer would encounter severe noise in the duty cycle. Because of that, a PWM port with hardware timer is preferred or at least a PWM generated by reusing the fewer bits timer. Lucky, although the Intel Edison board only offers us a 8bits timer on board, both the UNO board and BBB board have a time more than 16bits. The resolution accuracy of our PWM is 0.5us and 4000 levels of duty cycles are divided between 1ms and 2ms.

The real-time response of the drone contributes to the PID feedback control loop. Actually, PID control loop is a common concept in our daily life. The historical story happened in 1922, Minorsky was asked to design an automatic steering systems for the US Navy, noting the helmsman controlled the ship based not only on the current error, but also on past error as well as the current rate of change. This is similar to how we brake our car; slow down gradually before a long distance to the traffic light and stop abruptly within a short space. Even in the circuits design, a close loop feedback use the PID concept to reduce the noise influence.

Basically, PID is composed of three part of parameters: the proportional part P, the integration part I and the derivative part D. Thus, the output value is compensated by the current error between input and output, the total past error accumulation and the amount of error change. A mathematical formula could explain it much more clearly.

$$\begin{aligned} &Output = K_p e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t) \\ &Where: e = Setpoint - Input \end{aligned} \tag{2.1}$$

However, in our microcontroller all the information is digital. We cannot assume the whole system is a continuous change among the time. The transfer function from input to output is usually derived as a discrete system to resemble the control process. In practice, we monitor

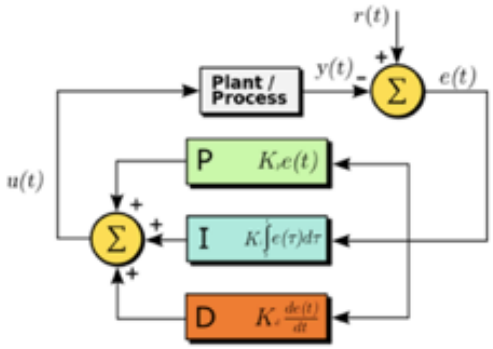



Figure 8: Algorithm of PID

the drone according to the motor response and sampling the velocity and acceleration data from the sensors within suitable period of time. Usually in our drone it would sample with a pre-determinate interval and save much cost in time period calculation. The derivative value in discrete system is obtained by subtracting the current error and the previous error. However, because the derivative of error is equal to the derivative of setting point and input. There is a spike in the PID tuning when the setting point change abruptly. By altering the derivative error to the negative derivative of input, the spike could be alleviated. For the output and integration term, the sum-up value should have a boundary to limit the maximum and minimum, otherwise this PID tuning would intrude into invalid range.

$$G_t(z^{-1}) = \frac{U(z^{-1})}{E(z^{-1})} = \frac{p_0 + p_1 z^{-1} + p_2 z^{-1}}{1 - z^{-1}} \tag{2.2}$$

Although the single PID implementation would satisfy the primary purpose for our feedback system, a robust choice is to improve it into a series PID. In reality, it is an essence for us to improve the single PID to a series PID. The reason to design a series PID is following. As mentioned previously, the theory of a drone balance system is a non-linear system. In other words, the non-linearity come from the actual relationship between the angular velocity and the moment of force, which is a second-order system. When investigating the inner system relationship, the angular acceleration in the drone and the moment of force shows a linear relationship. So notice that all the PID tuning is valid under the linear situation best and a small signal change for drone, a PID loop controls the angular acceleration change is taken into our design.

The series PID idea is connecting the first outer angular loop with the second inner angular acceleration loop. The outer loop has a setting point of angular according to the degree from pitch, yaw and roll, which generates an output of angular velocity change. The inner loop absorbs the angular velocity change and through manipulation gives an output of

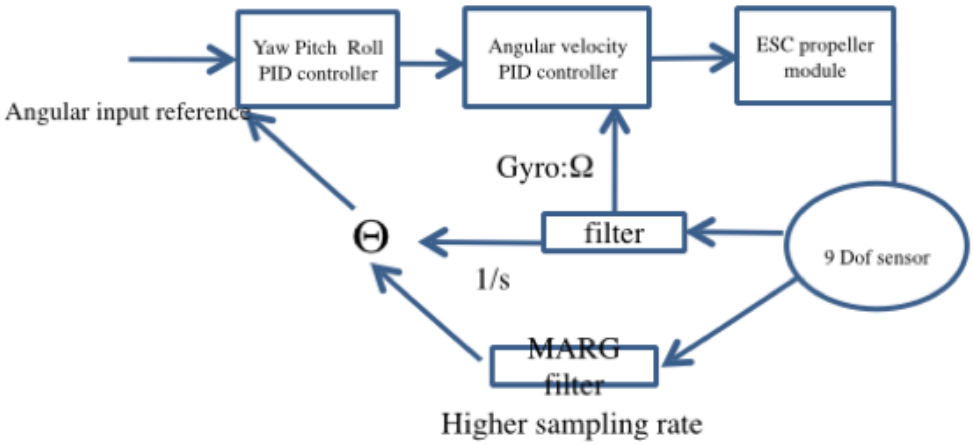



Figure 9: Algorithm of PID

angular acceleration. Instead of increasing and decreasing angular directly to the degree, the second-order drone system favors the smooth change from the angular acceleration, which swiftly and precisely reaches to the desired point.

**Noice Cancellation and Calibration**

PID module stabilize our drone system and the better performance is realized by noise cancellation and sensor calibration. The philosophy of noise cancellation and calibration is based on digital filter and its algorithm. In the MPU9250 chip, we have 10 options of digital low filter bandwidth, among which we may choose the lowest value, 5 Hz. However we will suffer from 33.48ms delay as a trade-off. Since the noise still influence the performance of the drone badly by using these inner filters, we try some filter algorithms such as Kalman filter, which uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimation of unknown variables that tend to be more precise than those based on a single measurement alone. At the end of the day, we constructs two channels for the filter dealing with various task. A simple filter designed to reduce the drift and white noise by rounding process. The int value of the gyro output, which has a range between +-32768, is first changed to the float type and divided to a decimal number. By selecting the round digits, the amount of noise could be chosen to ignore. A complex MARG matrix filter is utilized to correctly decode the yaw, pitch and roll degree. The MARG matrix filter is based on quaternion expression. All the gyro, acceleration and magnitude information is calculated in this matrix. This algorithm completes the noise calibration with fast and moderate accuracy. The process diagram is shown below. However, if the sensor raw data of accelerometer, gyroscope and magnetometer is not correct or there is a offset, the calculated gesture angles will contain noise and what's worse, the pitch, roll and yaw will interfere with each other. For example, we encountered the problem that when we tilt the

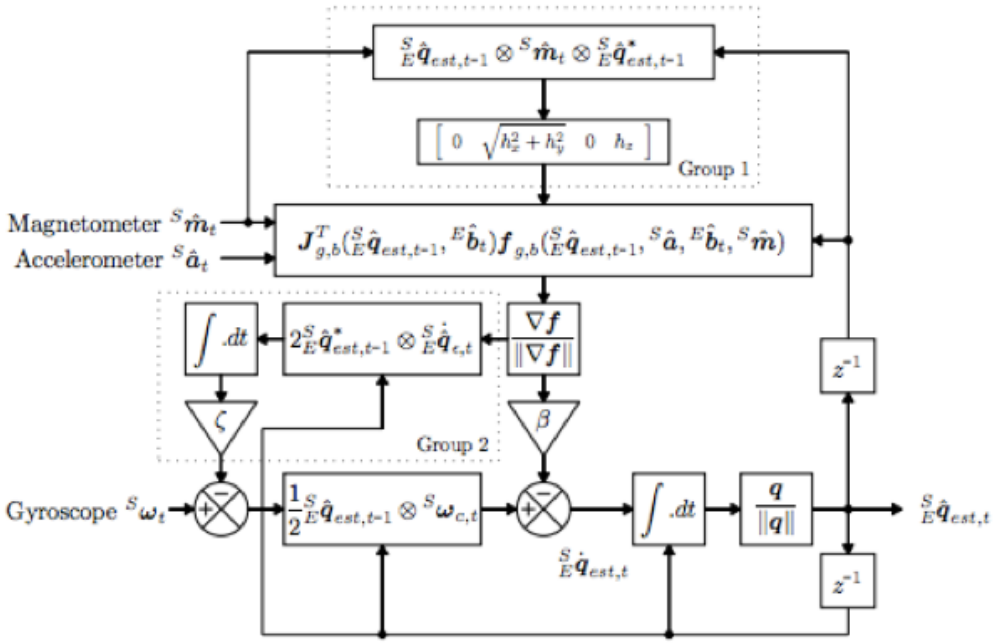



Figure 10: The test of calibration method

drone only changing pitch, the yaw will also change dramatically. This is totally because we did not calibrate the magnetometer. And another problem is when we place the drone still on the ground, the pitch, roll and yaw are very unstable and noisy. That is because the gyroscope has offset. In order to make sure that those raw data is correct, we need calibrate the MPU9250. For the accelerometer and gyroscope, we only need to make the drone horizontal and still. Then we read 500 raw gyro data and raw acc data and calculate the mean value which is the actual offset. For the magnetometer calibration, it is more complicated. Because the magnetic field is not parallel to the ground. So mainly we need to scan the intensity of mag in all directions and do spherical fitting, shown in the figure below. Then we will be able to find the offset of x, y, z axis.

**System Architecture Review**

The system architercture review is shown below

### 2.1.3 *Problems and Solution*

**Communication Latency**

When it comes to the latency, actually we worry a lot. The balance of drone need a very low latency, otherwise, it is very easy to encounter a case that before the last commend of adjustment is done, a new commend is sent to the processer. Then we need to reduce the

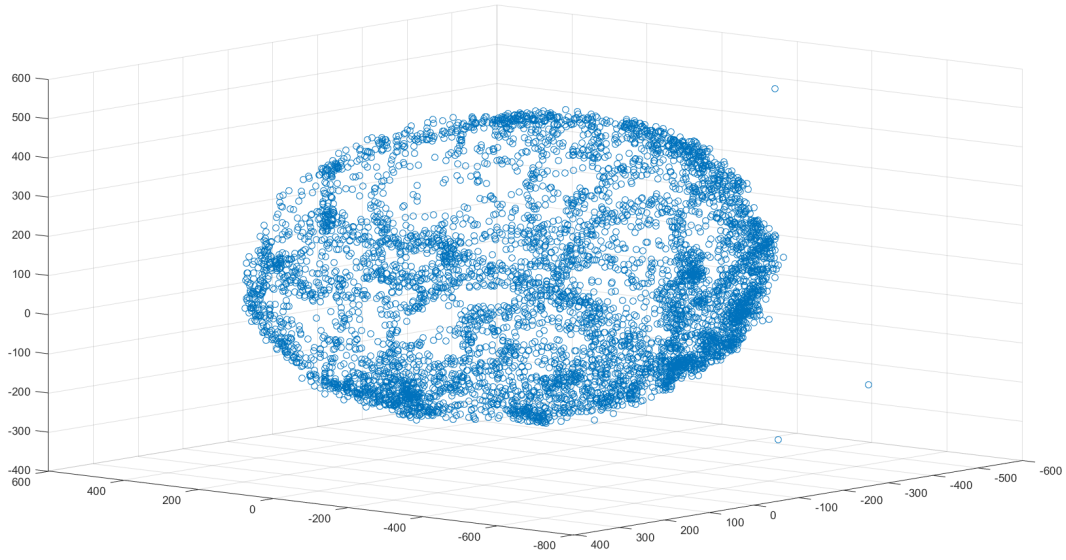


Figure 11: calibration result

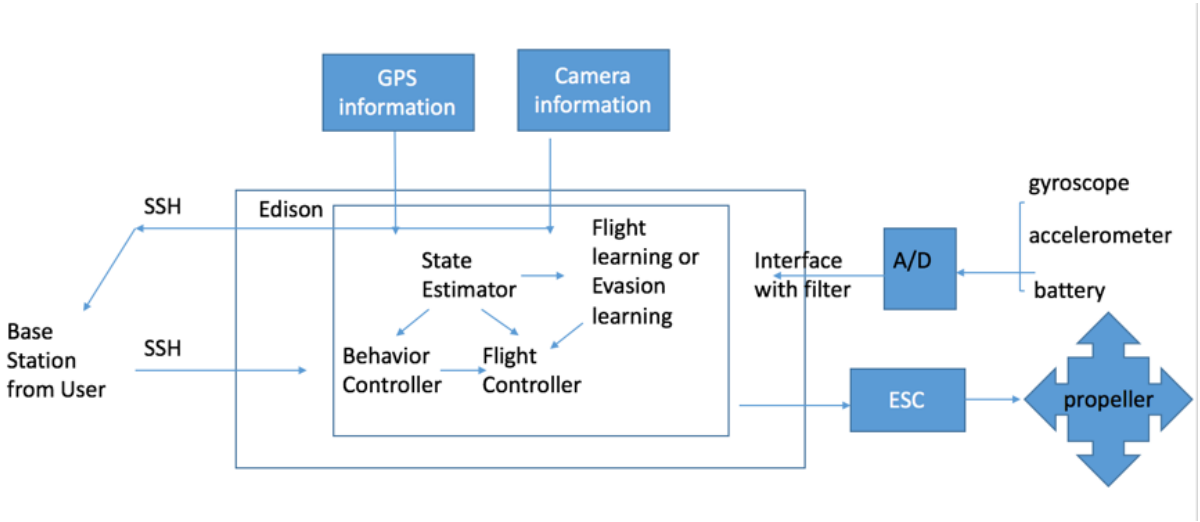



Figure 12: system architecture review

complexity of our data process algorithm and motor control algorithm. We have to run some tests on our drone to know how critical the latency is. We can use timer to check the delay in C code.

**Power Efficiency and Arrangement**

The battery we selected has around 3000mAh, with which the drone can fly about 15min

to 20min. In the very beginning, we use the same battery for powering Edison and three Arduino UNO. However, the battery has a better effect and longer duration when work together with some distributed Intel Edison small-size battery. The whole system would not need drain current from the main battery but stay in the sky for a longer period of time.

## 2.2 *System software Architecture*

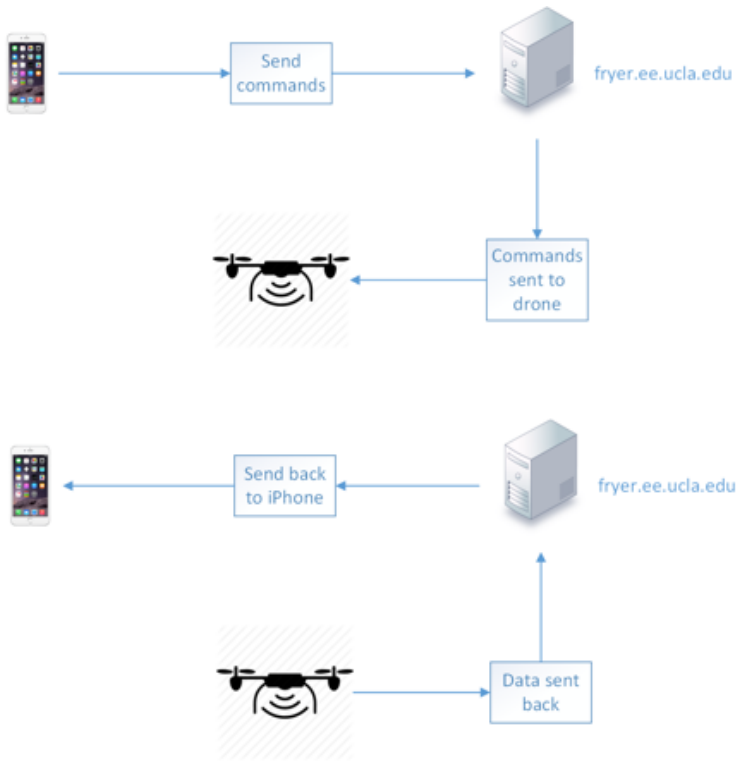



Figure 13: Communication flow chart

As shown in the figure, we need to let iPhone communicate with the drone. The drone is supposed to be controlled by smart phones remotely, so we cannot use phone to control using Bluetooth for the simple reason that Bluetooth has a range limit. Also, we cannot use "ssh method" to connect the drone, since the drone doesn't have a public IP address or public domain. Then we use a server "fryer.ee.ucla.edu" to be the solution. When we want to control the drone, the iPhone sends the command to the server, and the drone fetches the command from the server. Then the drone will react as the command does. Also, the drone can send its data to the server, so that we can get it and save it for analyzing purposes. The processing is shown below. We use HTTP Restful request as a communication way between devices and the server. Also we use MySQL as the database for managing the data that has been sent from the drones.

### 2.2.1 *Edison Board Software System*

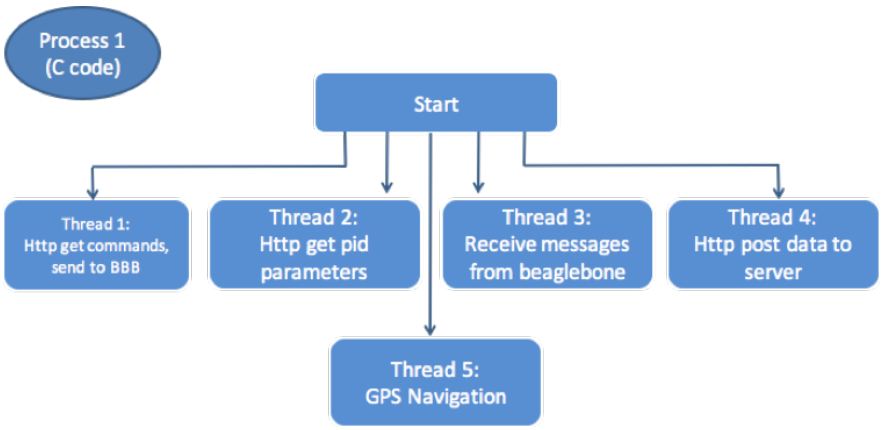



Figure 14: Software algorithm on ediosn

As shown in the figure, the first process in Edison is written in C code. There are 5 threads in this process. The first thread is to get commands from the server using HTTP request. The second thread is to get pid parameters from server via HTTP request. The third thread is to receive messages from BeagleBone Black via UART. The fourth thread is to post data to server via HTTP request. The fifth thread is to do GPS navigation using the GPS data received from BeagleBone Black.

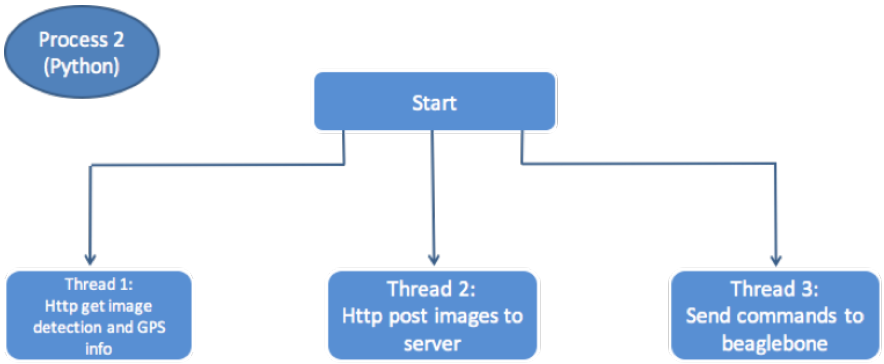



Figure 15: Software algorithm of image following

As shown in the figure, there is another process in Edison which is written in Python code. There are three threads in this process. The first two threads are about HTTP get/post images from the server, and the third thread is to send commands to BeagleBone Black via UART.

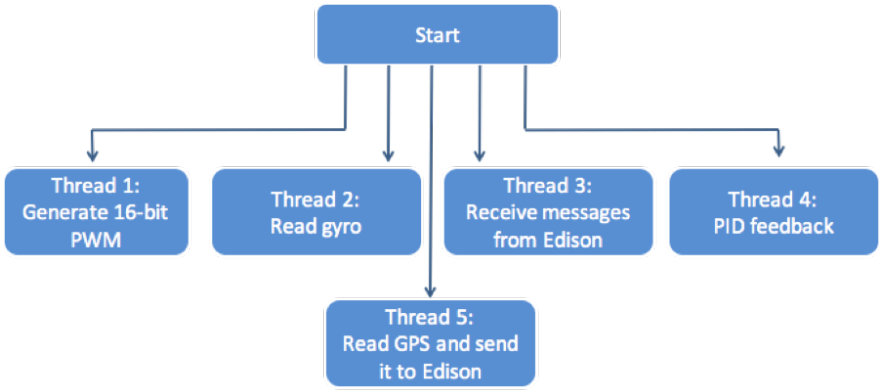



Figure 16: Software algorithm on bbb

### 2.2.2 *BeagleBone Board Software System*

As shown in the figure, there is another process in Edison which is written in Python code. There are three threads in this process. The first two threads are about HTTP get/post images from the server, and the third thread is to send commands to BeagleBone Black via UART.

### 2.2.3 *BeagleBone Autostart Script*

```
[Unit]
Description=beaglebone always
After=runlevel6.target dbus.service serial-getty@.service

[Service]
User=root
ExecStart=/root/drone/Edison/main/beaglebone_always.sh
Restart=always
RestartSec=1s

[Install]
WantedBy=multi-user.target
```

```
chmod 755 beaglebone_always.service kill_beaglebone_always.service
systemctl enable beaglebone_always.service kill_beaglebone_always.service
```

Figure 17: BeagleBone Autostart Script

As shown in the figure, we can auto start any program that we want when BeagleBone boots, since there is no Internet access for BeagleBone.

Our internal communication architecture evolves through three versions. In the version 1.0, seen in

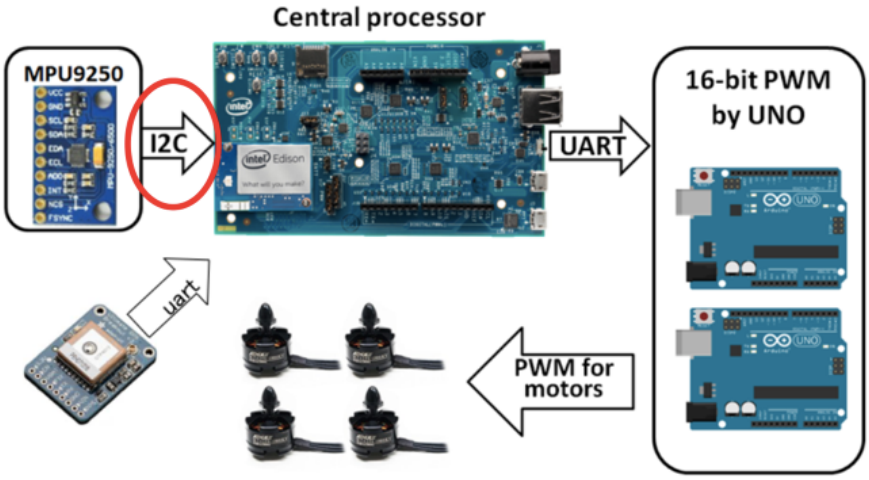



Figure 18: interface communcation version 1

the below architecture, the sensor data processing part would be divided into two categories. The first category consists of gyroscope sensor and accelerator sensor would provide us real-time data for the drone's 3D information. The second category would be the slower sampled GPS sensor, which contains the location information. All the sensors would be connected to the Edison board by I2C bus. At the same time, PWM signals are generated by UNO through UART communication. Yet, the unreliable I2C communication delays our progress. We have tested our I2C interface by using oscilloscope. The output signal is not like square wave. Especially for SCL port, the output contains sort of noise and the rise and fall time is poor. We first thought that would be our wire issue. After changing the wire the noise is reduced, but SCL is so sensitive to its loading and it would easily be locked to high during our operation. So the situation is that when we run software with I2C, the program would freeze in the middle. Sometimes maybe not freeze entirely, but it becomes super slow. One way to solve freezing is connecting SCL and SDA together for a while or plug out and in MPU's VCC. Yet the freezing issue would appear very soon after two or three times normal software running.

After noticing the annoying I2C problem, we try to define whether there is problem in our Edison board or somewhere else in our sensor. First, we developed a second version 2.0 drone system including 9dof Sparkfun sensor recommended by Chris. This sensor has a excellent performance on variables range, however the gyro noise from Sparkfun sensor increases. Worse still, the filter algorithm has to be modified to match this sensor and I2C problem cannot solve. Is this the kernel problem from Intel Edison board for I2C device commutation at high speed?

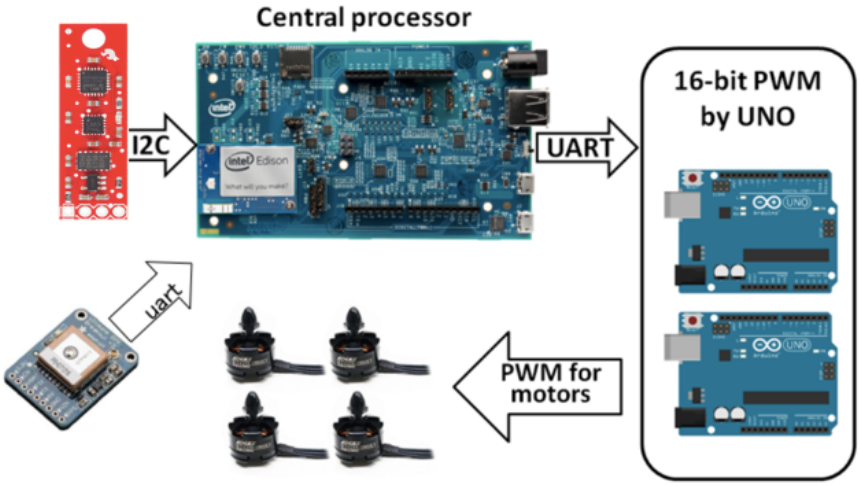



Figure 19: interface communcation version 2

We try to reboot the system, flash the firmware to the version 1.6/2.0/2.1. It seems that nothing surprising happens. The same problem displays again after our original joy.

How about abandoning the I2C communication? We re-design the system communication approach. Now on Intel Edison board, all the communication methods are series communication. The heavy-load MPU9250 sensor is I2C connected to UNO board. We just avoid I2C problem in a tricky way. However, the communication speed is decreased from 0.1ms to more than 3ms delay. The trade-off between the PID feedback and the sensor information reliability harms the whole system performance.

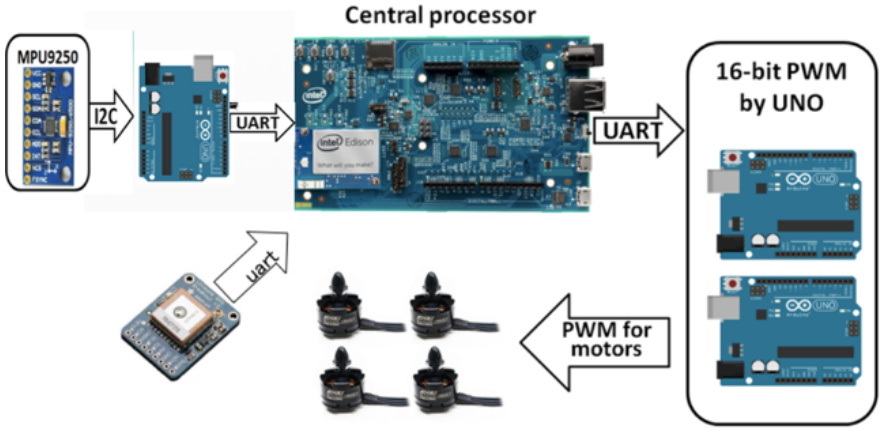



Figure 20: interface communcation version 3

The final version 4.0 interface communication is based on two microprocessor cores. The first is Intel Edison board and the second is Beaglebone black board. Now gyro sensor could be read by

on board I2C from beaglebone board with high speed. Meanwhile, the communication between Edison and beaglebone black board maintain a fast transmit speed with light task.

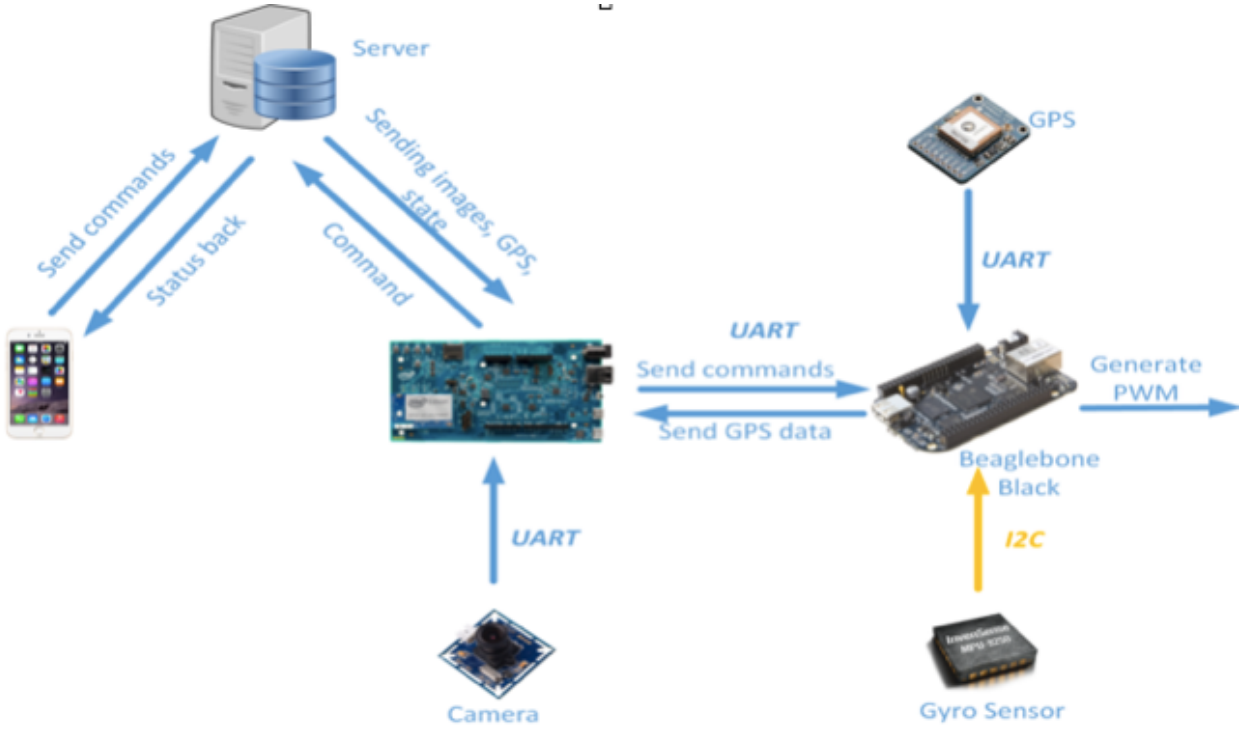



Figure 21: Currently used interface communcation

## 2.4 *Test and Evaluation*

### 2.4.1 *PID Test*

The PID test should be divided into three parts: pitch pid, roll pid and yaw pid. For pitch and roll pid, we just need to fix x or y axis so that we can test pitch or roll pid. As is mentioned, we use dual-loop pid for high performance. So what are good PID parameters? As far as we know, the only thing we need to consider is just how to make the drone stabilize in a very short time. For some PID parameters we tried, we found that the drone just swings back and forth and is very hard to stabilize. The ideal swinging times before stabilization is about 2-3 when the motors are not in full speed (This is experimental and experiential data.). When we keep increasing the speed of motors, the adjustment of PID would be much sharper mainly because when the propeller is rotating in a high speed, a small difference in speed would be amplified. After we get the PID parameters for pitch and roll, we can test the yaw PID when the drone is suspending.

### 2.4.2 *System Flight Balance Test*

After we have all the PID parameters, we can carry on the flight performance test, including suspending, forward, backward, left, right, up and down. For these days we are mainly focusing on that to complete our drone flight functions.

### 2.4.3 *GPS Tracker Test*

Our GPS tracker test installs our GPS module on the BBB and transmits the real-time data to server. During our test, the current latitude and longitude degree would be sent to our server and displayed on the map. And all of the data points collected from GPS will plot a continuous route on the map and point out the current direction you are heading to. Besides of the current location, all the other information like speed, course and altitude, is able to be fetched if necessary.

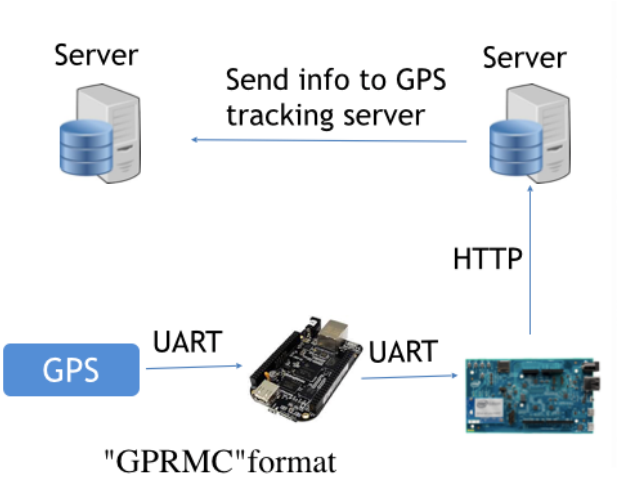



Figure 22: GPS interaction between the microprocessors and the server

## 3 SERVER AND DATABASE DESIGN

### 3.1 *Server REST API Design*

Since Edison is connected to the Internet via WiFi, we can use Edison Board to generate HTTP requests. We can get the messages from the server, and iPhone can send data to the server, so that they can communicate with each other.

# Ubidots:

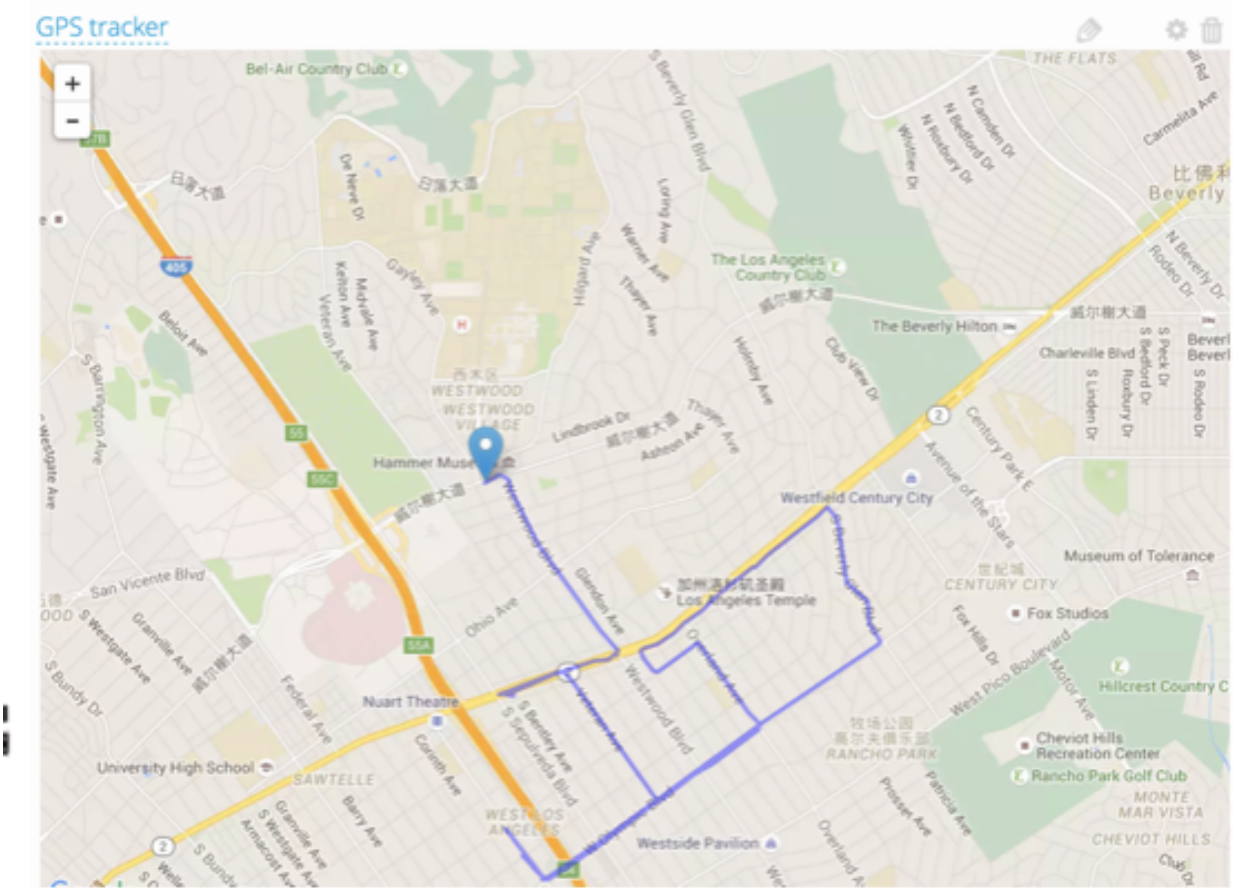



Figure 23: Ploted GPS route on Ubidots

## 3.2 *Server Database*

As shown in the figure above, we have 9 tables in the ”Edison” database. We use these database to store the drone’s data.

As shown in the figure above, we have 9 tables in the ”Edison” database. We use these database to store the drone’s data.

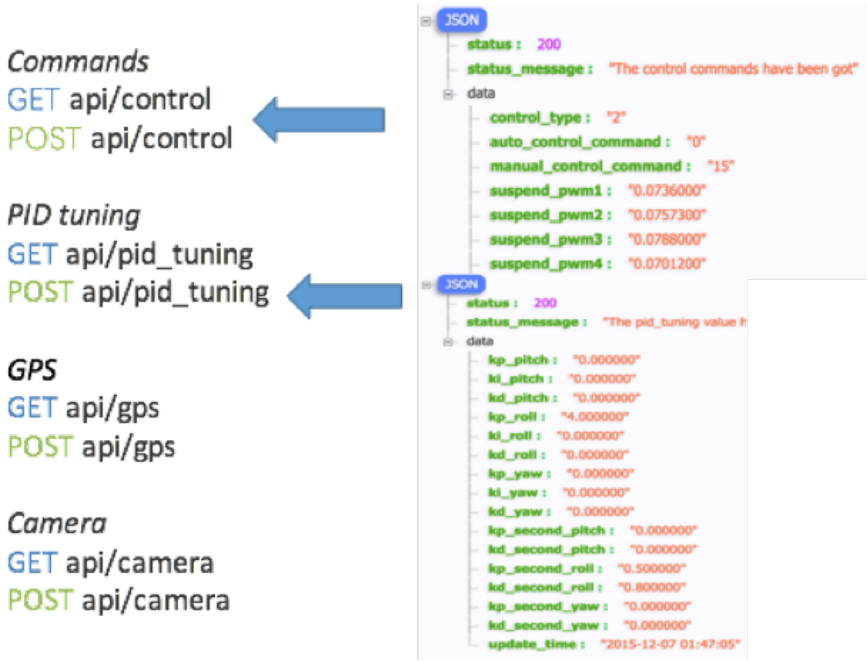



Figure 24: Server flow chat

```
+------------------+
| Tables_in_edison |
+------------------+
| camera           |
| control          |
| general          |
| gps              |
| gps_ubidots      |
| ip_address       |
| led              |
| pid_tuning       |
| pwm              |
+------------------+
```

Figure 25: Database screen shot

```
+----+-------------------+--------------+---------------------+----------------------+--------------+--------------+--------------+--------------+---------------------+
| id | mac_address       | control_type | auto_control_command | manual_control_command | suspend_pwm1 | suspend_pwm2 | suspend_pwm3 | suspend_pwm4 | update_time         |
+----+-------------------+--------------+---------------------+----------------------+--------------+--------------+--------------+--------------+---------------------+
|  1 | fc:c2:de:3d:7f:af |            2 |                   0 |                   15 |    0.0736000 |    0.0757300 |    0.0788000 |    0.0701200 | 2015-12-07 08:17:35 |
+----+-------------------+--------------+---------------------+----------------------+--------------+--------------+--------------+--------------+---------------------+
```

Figure 26: Database screen shot

## 4 TRACKING AND DETECTION ALGORITHM

The drone will patrol around a specific area. Once it detecting an invader, it will take pictures and report to us. For the detecting function, we are now planning to use a camera to take pictures automatically, then using image pattern recognition algorithm to detect the potential drone.

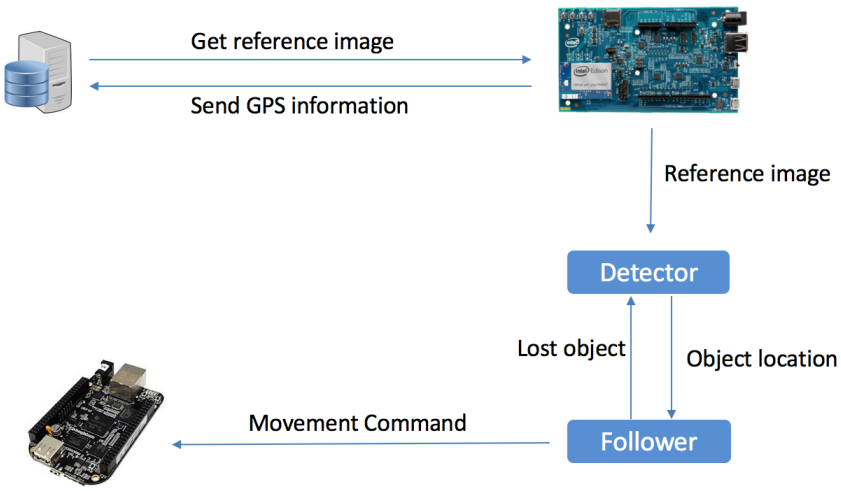



Figure 27: Process of detection and following algorithm

The process of the detection and following algorithm is shown in Fig.27. Firstly we should store the reference image to our server. Then when edison receive the command to enter the detection and following model, it will fetch the reference image from server and send the image to our detector. In detector, we will use SURF algorithm[2] to do features detection to find the object in the frames captured by the camera connected to the Edison. Because feature detection needs a lot of computation and may make mistakes in some frames. So a tracker is very important for us not only to same the computational consumption but also improve our following accuracy. So after finding the reference object in the frame, the location and new frames from camera will be sent to our follower. In the section, camshift algorithm[3] will be used to do object tracking. Then our Edison will send command to the our flight controller according to the movement of reference object in the pictures. Of course, the tracker sometimes will lose the object. In this condition, our Ediosn will go back to the detector patrolling around the area to do object detection. GPS information, in this process, will keep sending to our server to realise object location tracking.

## 4.1 *SURF Algorithm*

SURF is a algorithm for keypoints detection and description. It is a speed up version of SIFT.

For speed up purpose, SURF need a approxiamtes method. Fig.28 shows a demonstration of LoG approximates with Box filter in SURF. This approximation can be computed with the help of integral images. And it can be done in parallel for different scales.

Then to detect with orientation, SURF uses wavelet responses in both horizontal and vertical direction. This is applied for a neighbourhood size of 6s. We can plotted the process in a space as shown in Fig.29. Then we can estimate the dominant orientation by the sum of all responses within

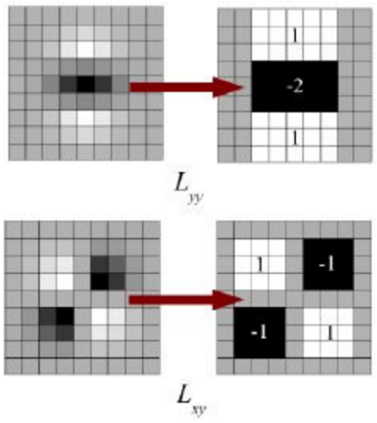



Figure 28: Graphical representation of approximates in SURF[4]

a specific window. For feature description, SURF uses the result of wavelet responses in horizontal and vertical direction. A window size of $20s \times 20s$ is taken around the keypoint where s is the size. Then it will be divided into $4 \times 4$ subregions. Horizontal and vertical wavelet responses are taken and verctor is formed as $v = (\sum d_x, \sum d_y, \sum |d_X|, \sum |d_y|)$ in each subregion. This will be used as the feature descriptor. For more details, SURF can compute the sum of $d_x$ and $|d_x|$ separately for $d_y \leq 0$ and $d_y \geq 0$. The same as $d_y$ and $|d_y|$. This can double the number of features.

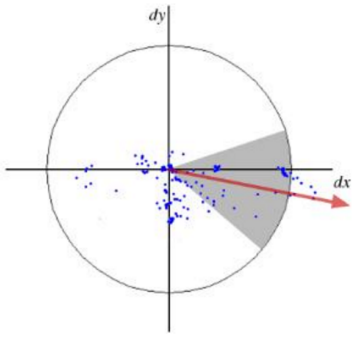



Figure 29: Wavelet responese in SURF algorithm[4]

Analysis shows that SURF is 3 times faster than SIFT. This algorithm has a acceptable accuracy and speed. The Fig.31 shows the test result when running this algorithm on Edison platform.

## 4.2 *Camshift Algorithm*

Histogram backprojection is a important part of camshift algorithm. It help fetch the needed pixel for computation. Back Projection is a way of recording how well the pixels of a given image fit the distribution of pixels in a histogram model. For example you get a specific histogram, for

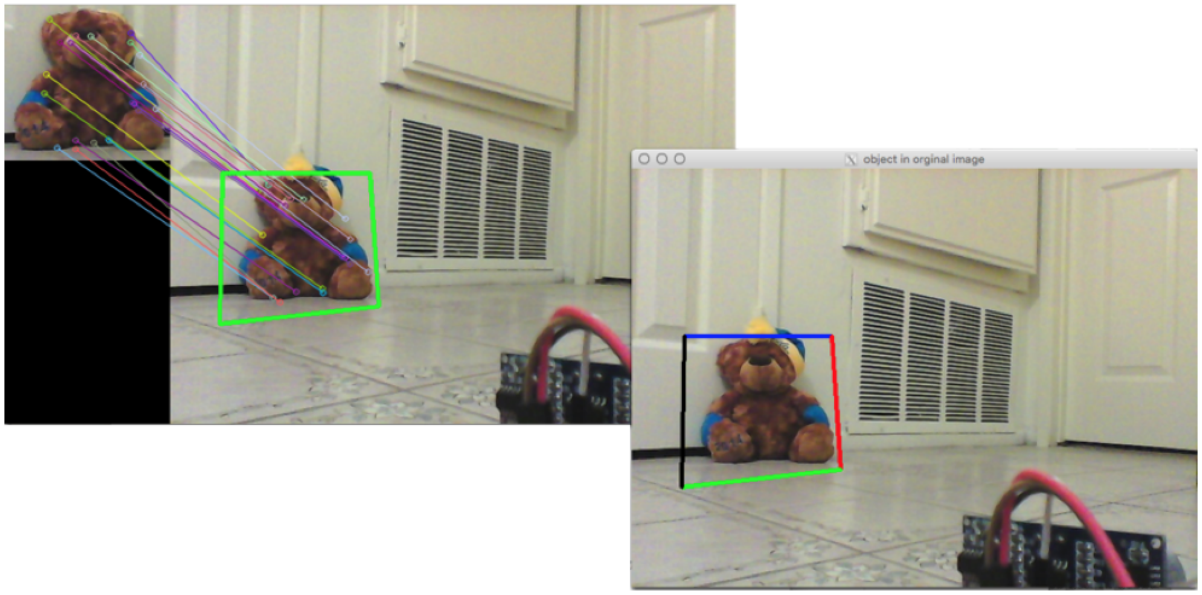



Figure 30: Result of surf running on Edison

a new image as test image, you should firstly collect the data and find the correspondent bin location in the histogram for that pixel($i.e.(h_{i,j}, s_{i,j})$). Then lookup the specific histogram in the correspondent bin getting the bin value. This bin value should be stored in a new image which is the bcakprojection image. This image represent the probability that a pixel in test image belongs to the specific histogram, which represent a specific area. One result of Histogram backprojection is shown below, the red flower is the specific object.

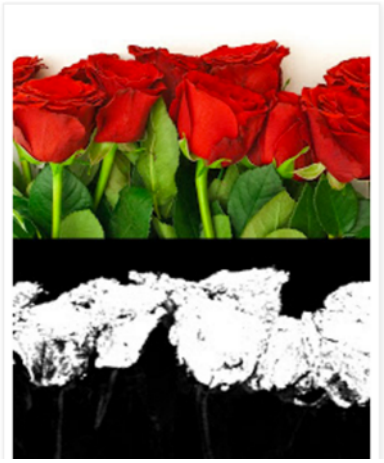


Figure 31: Sample result of histogram bcakprojection on red flower

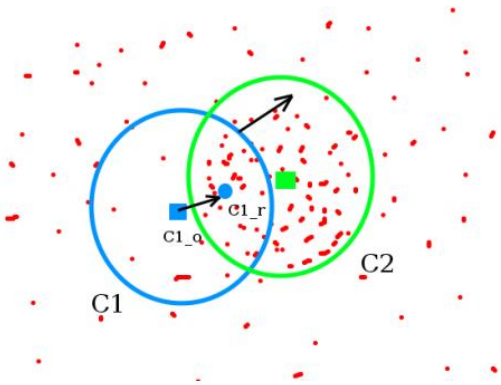



Figure 32: Graphic representation of mean shift algorithm

The first step for camshift algorithm is to get the target distribution from initial location and do histogram backproject to generate the probability image. Besides, weight and ratio should be done on the histogram to decrease the influence of background. Then meanshift is implement in each frame. The process of meanshift is shown in Fig.32. The initial window is shown in blue circle and the origin center is marked in blue rectangle. However, the centroid of pixel in the circle is the blue circle. So in meanshift, it will keep move the center of your window to the centroid of the pixels in the window until the centriod and the center of your window are on the same location. So when the object moves, the movement will be reflected in the histogram and thus the backprojected image. Therefroe, meanshift will keep move the window with the movement of our traget object. In this process, $0th$ moment $M_{00}$ should be recorded. Once the meanshif converges, we should update the size of window as $s = 2 \times \sqrt{\frac{M_{00}}{256}}$. Then we will get the new window of tracking object.

## 4.3 *Accuracy Detection*

According to the test result, the compute speed of Edison is fast enough to do some simple detection method like SIFT and SURF. However, when comes to some accurate detection algorithm like HOG or fast-rcnn, the compute speed of Edison is far from enough. It can not do real-time accurate detetcion on Edison. To solve this problem, we proposed a method as shown in Fig.**??**.

The first step for accuracy detection is searching model. In this model our drone can patrol in a specific area. At the same time, Edison will keep sending images and GPS information taken from camera to our server. Then our server can do the accurate detection algorithm like fast-rcnn to detect the object we want. Once it find the object, it will store the window of the object as the reference image. A command and a GPS information will be sent to our Edison (if we have multiple drones, we can send the command to drones, which are near to the GPS location of the traget)/ The Edison will entering the detecting and following model introduced in 4.1. It will start to detect nearby area according to the GPS information sent by our server. Using this method, our drone can do a more accurate detection task.

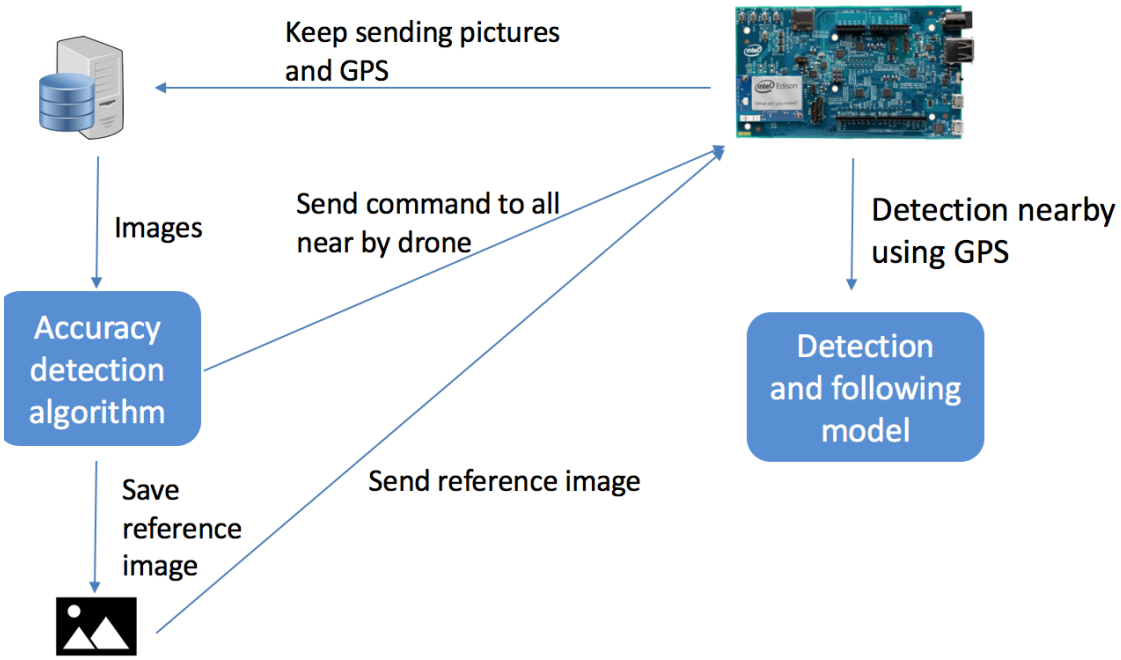



Figure 33: Process of accurate detection and following algorithm

## 5 SYSTEM SOURCE CODE

All the source codes and readme descriptions are provided in `https://github.com/peidong/drone`

Below is some essential codes in different modules.

1. /main/include/beaglebone_drone_include.h

   (a) process_message

   Decode the messages that are sent from Edison board to BeagleBone Black via UART.

Listing 1: Key code in the function

```
for (n_temp_command_index = 0; n_temp_command_index <= 2; n_temp_command_index++){
arrc_command_index[n_temp_command_index] = arrc_buffer[n_temp_command_index];
}
arrc_command_index[3] = '\0';
n_command_index = atoi(arrc_command_index);

```

   (b) communication_with_edison_uart

   Generate and receive UART messages between Edison and BeagleBone Black.

Listing 2: Key code in the function

```
if (mraa_uart_data_available(beaglebone_uart, 50) == 1){
    mraa_uart_read(beaglebone_uart, c_flag, 1);
    if (c_flag[0] == '~'){
```

```
        nflag_find_beginning = 1;
        n_receive_message_index = 0;
        while (nflag_find_end != 1){
            if (pT_drone->nflag_stop_all != 0){
                break;
            }
            if (mraa_uart_data_available(beaglebone_uart, 50) == 1){
                mraa_uart_read(beaglebone_uart, arrc_buffer + n_receive_message_index, 1);
                if (arrc_buffer[n_receive_message_index] == '$'){
                    arrc_buffer[n_receive_message_index] = '\0';
                    nflag_find_end = 1;
                    //break;
                }else if (arrc_buffer[n_receive_message_index] == '~'){
                    nflag_find_end = -1;
                    nflag_find_beginning = 1;
                    n_receive_message_index = 0;
                    //continue;
                }else{
                    n_receive_message_index++;
                    nflag_find_end = 0;
                }
            }
        }
    }
}

```

(c) update_T_drone_gps

Read GPS data value via UART from the sensor to BeagleBone Black.

Listing 3: Key code in the function

```
mraa_uart_read(gps_uart, arrc_search, 1);
if (arrc_search[0] == '$'){
    for(i=1; i<7;i++){
        mraa_uart_read(gps_uart, arrc_search+i, 1);
    }
    if (strstr(arrc_search, "$GPRMC,")){
        for(i=0; i<100;i++){
            mraa_uart_read(gps_uart, arrc_buf+i, 1);
            if (arrc_buf[i] == '\n'){
                arrc_buf[i]='\0';
                break;
            }
        }
        // printf("%s\n", arrc_buf);
        nmea_parse_gprmc(arrc_buf, &readGPS);
        gps_convert_deg_to_dec(&(readGPS.latitude), readGPS.lat, &(readGPS.longitude), readGPS.lon);

```

(d) update_T_drone_arrd_yaw_pitch_roll

Read Gyroscope's data value via I2C from the MPU9250 sensor to BeagleBone Black.

Listing 4: Key code in the function

```
mraa_i2c_read_bytes_data(mpu, 59, Buf, 14);
// Accelerometer
arawx = -(Buf[0] << 8 | Buf[1]);
arawy = -(Buf[2] << 8 | Buf[3]);
arawz = Buf[4] << 8 | Buf[5];
// Gyroscope
grawx = (Buf[8] << 8 | Buf[9]) - 50;
grawy = (Buf[10] << 8 | Buf[11]) + 0;
grawz = (Buf[12] << 8 | Buf[13]) + 30;
// Magnetometer
mraa_i2c_read_bytes_data(mpu, 73, Buf, 6);
mrawx = (Buf[1] << 8 | Buf[0]);//-213;// + mag_offset_x;
mrawy = (Buf[3] << 8 | Buf[2]);//-92;// + mag_offset_y;
mrawz = (Buf[5] << 8 | Buf[4]);//+200;// + mag_offset_z;
```

```
ax = (float)arawx*aRes;
ay = (float)arawy*aRes;
az = (float)arawz*aRes;
gx = (float)grawx*gRes;
gy = (float)grawy*gRes;
gz = (float)grawz*gRes;
mx = (float)mrawx*mRes*magCalibration[0] - 292.5;  // get actual magnetometer value, this depends on scale being set
my = (float)mrawy*mRes*magCalibration[1] - 32;
mz = (float)mrawz*mRes*magCalibration[2] + 89;
MadgwickAHRSupdate(ax, ay, az, gx*PI / 180.0f, gy*PI / 180.0f, gz*PI / 180.0f, my, mx, mz); //my, mx, mz
// Calculate yaw pitch roll
yaw = atan2(2.0f * (q1 * q2 + q0 * q3), q0 * q0 + q1 * q1 - q2 * q2 - q3 * q3);
pitch = -asin(2.0f * (q1 * q3 - q0 * q2));
roll = atan2(2.0f * (q0 * q1 + q2 * q3), q0 * q0 - q1 * q1 - q2 * q2 + q3 * q3);
yaw *= 180.0f / PI;
pitch *= 180.0f / PI;
roll *= 180.0f / PI;
if (yaw<0) yaw += 360;

```

(e) update_T_drone_arrd_pid

Compute the feedback of gyroscope's values to balance the drone's yaw, pitch and roll values.

Listing 5: Key code in the function

```
//first loop
Pid_SetTunings(pidData_yaw, kp_yaw, ki_yaw, kd_yaw);
Pid_SetTunings(pidData_pitch, kp_pitch*10, ki_pitch, kd_pitch);
Pid_SetTunings(pidData_roll, kp_roll*10, ki_roll, kd_roll);
Pid_SetSetPoint(pidData_yaw, 0);
Pid_Run(pidData_yaw, (int)pT_drone->arrd_yaw_pitch_roll[0]/1.0, 0);
Pid_SetSetPoint(pidData_pitch, 0);
Pid_Run(pidData_pitch, (int)pT_drone->arrd_yaw_pitch_roll[1]/1.0, 0);
Pid_SetSetPoint(pidData_roll, 40);
Pid_Run(pidData_roll, (int)pT_drone->arrd_yaw_pitch_roll[2]/1.0, 0);
//second loop
d_rate_yaw = pidData_yaw->output/10.0;
d_rate_pitch = pidData_pitch->output/10.0;
d_rate_roll = pidData_roll->output/10.0;
Pid_SetTunings(pidData_second_yaw, kp_second_yaw*10, 0, kd_second_yaw);
Pid_SetTunings(pidData_second_pitch, kp_second_pitch*10, 0.001, kd_second_pitch);
Pid_SetTunings(pidData_second_roll, kp_second_roll*10, 0.001, kd_second_roll);
Pid_SetSetPoint(pidData_second_yaw, d_rate_yaw);
Pid_SetSetPoint(pidData_second_pitch, d_rate_pitch);
Pid_SetSetPoint(pidData_second_roll, d_rate_roll);
Pid_Run(pidData_second_yaw, -Pid_rm_noise(pT_drone->n_grawz/32768.0)/1.0, 0);
Pid_Run(pidData_second_pitch, -Pid_rm_noise(pT_drone->n_grawy/32768.0)/1.0, 0);
Pid_Run(pidData_second_roll, -Pid_rm_noise(pT_drone->n_grawx/32768.0)/1.0, 0);

```

(f) GeneratePwm

Generate pwm wave from beaglebone black to control the speed of motors.

Listing 6: Key code in the function

```
pwm_set_duty_cycle("P9_14", 100 * pT_drone->arrd_current_pwm[0]);
pwm_set_duty_cycle("P9_16", 100 * pT_drone->arrd_current_pwm[1]);
pwm_set_duty_cycle("P8_13", 100 * pT_drone->arrd_current_pwm[2]);
pwm_set_duty_cycle("P8_19", 100 * pT_drone->arrd_current_pwm[3]);

```

(g) CalibrateEsc

Calibrate Esc from beaglebone black, this is for Esc initialization.

Listing 7: Key code in the function

```
printf("Setting the pwm duty cycle to max 0.1\n");
pwm_start("P9_14", 100 * 0.1, 50, 0);//pwm3
pwm_start("P9_16", 100 * 0.1, 50, 0);//pwm4
pwm_start("P8_13", 100 * 0.1, 50, 0);//pwm6
pwm_start("P8_19", 100 * 0.1, 50, 0);//pwm5
pwm_set_duty_cycle("P9_14", 100 * pT_drone->arrd_current_pwm[0]);
pwm_set_duty_cycle("P9_16", 100 * pT_drone->arrd_current_pwm[1]);
pwm_set_duty_cycle("P8_13", 100 * pT_drone->arrd_current_pwm[2]);
pwm_set_duty_cycle("P8_19", 100 * pT_drone->arrd_current_pwm[3]);
printf("Setting the pwm duty cycle to min 0.0001\n");

```

2. /main/include/drone_include.h

   (a) update_T_drone_http_gps_ubidots_post

   Update the GPS value which is received from BeagleBone to the server.

Listing 8: Key code in the function

```
char *sz_url_post_gps_ubidots = "http://128.97.89.181/rest/api/gps_ubidots/post/";
char arrc_post_data[100];
//store the post data
sprintf(arrc_post_data, "face_direction=%d&latitude=%f&longitude=%f", pT_drone->n_face_direction, pT_drone->
    d_current_latitude, pT_drone->d_current_longitude);
//http post
http_post(sz_url_post_gps_ubidots, arrc_post_data);

```

   (b) process_message

   Decode the messages(e.g. GPS value) that are sent from Edison board to BeagleBone Black via UART.

Listing 9: Key code in the function

```
while (nflag_load_face_direction == 0 || nflag_load_latitude == 0 || nflag_load_longitude == 0){
    if (nflag_load_face_direction == 0){
        if (arrc_buffer[n_message_index] == '|'){
            arrc_face_direction[n_face_direction_index] = '\0';
            nflag_load_face_direction = 1;
            n_message_index++;
        }else{
            arrc_face_direction[n_face_direction_index] = arrc_buffer[n_message_index];
            n_face_direction_index++;
            n_message_index++;
            nflag_load_face_direction = 0;
        }
    }else if (nflag_load_latitude == 0){
        if (arrc_buffer[n_message_index] == '|'){
            arrc_latitude[n_latitude_index] = '\0';
            nflag_load_latitude = 1;
            n_message_index++;
        }else{
            arrc_latitude[n_latitude_index] = arrc_buffer[n_message_index];
            n_latitude_index++;
            n_message_index++;
            nflag_load_latitude = 0;
        }
    }else if (nflag_load_longitude == 0){
        if (arrc_buffer[n_message_index] == '$'){
            arrc_longitude[n_longitude_index] = '\0';
            nflag_load_longitude = 1;
            n_message_index++;
        }else{
            arrc_longitude[n_longitude_index] = arrc_buffer[n_message_index];
            n_longitude_index++;
```

```
            n_message_index++;
            nflag_load_longitude = 0;
        }
    }
}

```

(c) communication_with_beaglebone_uart

Encode the message that is to be sent to BeagleBone Black, and receive the messages from BeagleBone Black.

Listing 10: Key code in the function

```
if (n_command_index == 0){
    char arrc_message[6] = "~000$";
    arrc_message[5] = '\0';
    mraa_uart_write(edison_uart, arrc_message, 5);
    usleep(1000);
}else if (n_command_index >= 100 && n_command_index <= 299){
    char arrc_message[6] = {'\0'};
    sprintf(arrc_message, "~%d$", n_command_index);
    arrc_message[5] = '\0';
    mraa_uart_write(edison_uart, arrc_message, 5);
    usleep(1000);
}else if (n_command_index == 301){
    char arrc_message[14] = {'\0'};
    sprintf(arrc_message, "~%d%.6f$", n_command_index, pT_drone->d_kp_pitch);
    arrc_message[13] = '\0';
    mraa_uart_write(edison_uart, arrc_message, 13);
    usleep(1000);
}
if (mraa_uart_data_available(edison_uart, 0) == 1){
mraa_uart_read(edison_uart, arrc_buffer + n_receive_message_index, 1);
if (arrc_buffer[n_receive_message_index] == '$'){
    n_receive_message_index++;
    arrc_buffer[n_receive_message_index] = '\0';
    nflag_find_end = 1;
}else if (arrc_buffer[n_receive_message_index] == '~'){
    nflag_find_end = -1;
    nflag_find_beginning = 1;
    n_receive_message_index = 0;
    arrc_buffer[n_receive_message_index] = '~';
}else{
    n_receive_message_index++;
    nflag_find_end = 0;
}
}

```

(d) update_T_drone_http_pid_tuning_get

Get the pid_tuning parameters' value that are sent from iPad/iPhone.

Listing 11: Key code in the function

```
char *sz_url_get_pid_tuning = "http://128.97.89.181/rest/api/pid_tuning/get/";
sz_http_response = http_get(sz_url_get_pid_tuning);
pT_json_object_whole_response = json_tokener_parse(sz_http_response);

n_json_response = json_object_object_get_ex(pT_json_object_whole_response, "data", &pT_json_object_data);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_pitch", &ppT_json_object_pid_tuning[0]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "ki_pitch", &ppT_json_object_pid_tuning[1]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_pitch", &ppT_json_object_pid_tuning[2]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_roll", &ppT_json_object_pid_tuning[3]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "ki_roll", &ppT_json_object_pid_tuning[4]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_roll", &ppT_json_object_pid_tuning[5]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_yaw", &ppT_json_object_pid_tuning[6]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "ki_yaw", &ppT_json_object_pid_tuning[7]);
n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_yaw", &ppT_json_object_pid_tuning[8]);
```

```
15
16 n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_second_pitch", &
       ppT_json_object_pid_second_tuning[0]);
17 n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_second_pitch", &
       ppT_json_object_pid_second_tuning[1]);
18 n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_second_roll", &
       ppT_json_object_pid_second_tuning[2]);
19 n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_second_roll", &
       ppT_json_object_pid_second_tuning[3]);
20 n_json_response = json_object_object_get_ex(pT_json_object_data, "kp_second_yaw", &ppT_json_object_pid_second_tuning
       [4]);
21 n_json_response = json_object_object_get_ex(pT_json_object_data, "kd_second_yaw", &ppT_json_object_pid_second_tuning
       [5]);
22 n_json_response = json_object_object_get_ex(pT_json_object_data, "update_time", &pT_json_object_update_time);
23
24 pT_drone->d_kp_pitch = json_object_get_double(*(ppT_json_object_pid_tuning + 0));
25 pT_drone->d_ki_pitch = json_object_get_double(*(ppT_json_object_pid_tuning + 1));
26 pT_drone->d_kd_pitch = json_object_get_double(*(ppT_json_object_pid_tuning + 2));
27 pT_drone->d_kp_roll = json_object_get_double(*(ppT_json_object_pid_tuning + 3));
28 pT_drone->d_ki_roll = json_object_get_double(*(ppT_json_object_pid_tuning + 4));
29 pT_drone->d_kd_roll = json_object_get_double(*(ppT_json_object_pid_tuning + 5));
30 pT_drone->d_kp_yaw = json_object_get_double(*(ppT_json_object_pid_tuning + 6));
31 pT_drone->d_ki_yaw = json_object_get_double(*(ppT_json_object_pid_tuning + 7));
32 pT_drone->d_kd_yaw = json_object_get_double(*(ppT_json_object_pid_tuning + 8));
33
34 pT_drone->d_kp_second_pitch = json_object_get_double(*(ppT_json_object_pid_second_tuning + 0));
35 pT_drone->d_kd_second_pitch = json_object_get_double(*(ppT_json_object_pid_second_tuning + 1));
36 pT_drone->d_kp_second_roll = json_object_get_double(*(ppT_json_object_pid_second_tuning + 2));
37 pT_drone->d_kd_second_roll = json_object_get_double(*(ppT_json_object_pid_second_tuning + 3));
38 pT_drone->d_kp_second_yaw = json_object_get_double(*(ppT_json_object_pid_second_tuning + 4));
39 pT_drone->d_kd_second_yaw = json_object_get_double(*(ppT_json_object_pid_second_tuning + 5));
40
```

(e) update_T_drone_http

Get the control commands that are sent from iPhone/iPad.

Listing 12: Key code in the function

```
1 char *sz_url_get_control_part1 = "http://128.97.89.181/rest/api/control/get/?mac_address=fc:c2:de:3d:7f:af";
2
3 sz_http_response = http_get(sz_url_get_control);
4
5 pT_json_object_whole_response = json_tokener_parse(sz_http_response);
6
7 n_json_response = json_object_object_get_ex(pT_json_object_whole_response, "data", &pT_json_object_data);
8 n_json_response = json_object_object_get_ex(pT_json_object_data, "control_type", &pT_json_object_control_type);
9 n_json_response = json_object_object_get_ex(pT_json_object_data, "auto_control_command", &
      pT_json_object_auto_control_command);
10 n_json_response = json_object_object_get_ex(pT_json_object_data, "manual_control_command", &
      pT_json_object_manual_control_command);
11 n_json_response = json_object_object_get_ex(pT_json_object_data, "suspend_pwm1", &ppT_json_object_suspend_pwm[0]);
12 n_json_response = json_object_object_get_ex(pT_json_object_data, "suspend_pwm2", &ppT_json_object_suspend_pwm[1]);
13 n_json_response = json_object_object_get_ex(pT_json_object_data, "suspend_pwm3", &ppT_json_object_suspend_pwm[2]);
14 n_json_response = json_object_object_get_ex(pT_json_object_data, "suspend_pwm4", &ppT_json_object_suspend_pwm[3]);
15 n_json_response = json_object_object_get_ex(pT_json_object_data,"update_time",&pT_json_object_update_time);
16
17 pT_drone->n_control_type = json_object_get_int(pT_json_object_control_type);
18 pT_drone->n_auto_control_command = json_object_get_int(pT_json_object_auto_control_command);
19 pT_drone->n_manual_control_command = json_object_get_int(pT_json_object_manual_control_command);
20 for(n_index = 0; n_index < 4; n_index++){
21     pT_drone->arrd_suspend_pwm[n_index] = json_object_get_double(*(ppT_json_object_suspend_pwm + n_index));
22 }
23
```

(f) update_T_drone_http_gps

Get the destination GPS location from the server.

Listing 13: Key code in the function

```
char *sz_url_get_gps_destination = "http://128.97.89.181/rest/api/gps/get/?location_type=0";//get destination gps
sz_http_response = http_get(sz_url_get_gps_destination);
pT_json_object_whole_response = json_tokener_parse(sz_http_response);
n_json_response = json_object_object_get_ex(pT_json_object_whole_response, "data", &pT_json_object_data);
n_json_response = json_object_object_get_ex(pT_json_object_data, "face_direction", &pT_json_object_face_direction);
n_json_response = json_object_object_get_ex(pT_json_object_data, "latitude", &pT_json_object_latitude);
n_json_response = json_object_object_get_ex(pT_json_object_data, "longitude", &pT_json_object_longitude);
n_json_response = json_object_object_get_ex(pT_json_object_data, "update_time",&pT_json_object_update_time);
pT_drone->d_destination_latitude = json_object_get_double(pT_json_object_latitude);
pT_drone->d_destination_longitude = json_object_get_double(pT_json_object_longitude);

```

3. /main/include/http/http.h

   (a) http_get/http_post

   Do get/post http requests using libcurl.

Listing 14: Key code in the function

```
curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, url);
curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);
res = curl_easy_perform(curl);

```

4. /main/include/gps/nmea.h

   (a) nmea_parse_gprmc

   Get the GPS sensor's values: latitude, longitude and direction.

Listing 15: Key code in the function

```
p = strchr(p, ',')+1; //skip time
p = strchr(p, ',')+1; //skip status
p = strchr(p, ',')+1;
loc->latitude = atof(p);
p = strchr(p, ',')+1;
switch (p[0]) {
    case 'N':
        loc->lat = 'N';
        break;
    case 'S':
        loc->lat = 'S';
        break;
    case ',':
        loc->lat = '\0';
        break;
}
p = strchr(p, ',')+1;
loc->longitude = atof(p);
p = strchr(p, ',')+1;
switch (p[0]) {
    case 'W':
        loc->lon = 'W';
        break;
    case 'E':
        loc->lon = 'E';
        break;
    case ',':
        loc->lon = '\0';
        break;
}
p = strchr(p, ',')+1;
loc->speed = atof(p);
```

```
p = strchr(p, ',')+1;
loc->course = atof(p);

```

5. /main/include/mpu9250/mpu9250_bbb.h

   (a) MadgwickAHRSupdate

   Update the gyroscope's value.

Listing 16: Key code in the function

```
// Rate of change of quaternion from gyroscope
qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);
qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);
qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);
qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);
// Reference direction of Earth's magnetic field
hx = mx * q0q0 - _2q0my * q3 + _2q0mz * q2 + mx * q1q1 + _2q1 * my * q2 + _2q1 * mz * q3 - mx * q2q2 - mx * q3q3;
hy = _2q0mx * q3 + my * q0q0 - _2q0mz * q1 + _2q1mx * q2 - my * q1q1 + my * q2q2 + _2q2 * mz * q3 - my * q3q3;
_2bx = sqrt(hx * hx + hy * hy);
_2bz = -_2q0mx * q2 + _2q0my * q1 + mz * q0q0 + _2q1mx * q3 - mz * q1q1 + _2q2 * my * q3 - mz * q2q2 + mz * q3q3;
_4bx = 2.0f * _2bx;
_4bz = 2.0f * _2bz;
// Gradient decent algorithm corrective step
s0 = -_2q2 * (2.0f * q1q3 - _2q0q2 - ax) + _2q1 * (2.0f * q0q1 + _2q2q3 - ay) - _2bz * q2 * (_2bx * (0.5f - q2q2 -
     q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q3 + _2bz * q1) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) -
     my) + _2bx * q2 * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
s1 = _2q3 * (2.0f * q1q3 - _2q0q2 - ax) + _2q0 * (2.0f * q0q1 + _2q2q3 - ay) - 4.0f * q1 * (1 - 2.0f * q1q1 - 2.0f *
      q2q2 - az) + _2bz * q3 * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (_2bx * q2 + _2bz * q0) *
      (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q3 - _4bz * q1) * (_2bx * (q0q2 + q1q3) + _2bz *
     (0.5f - q1q1 - q2q2) - mz);
s2 = -_2q0 * (2.0f * q1q3 - _2q0q2 - ax) + _2q3 * (2.0f * q0q1 + _2q2q3 - ay) - 4.0f * q2 * (1 - 2.0f * q1q1 - 2.0f
     * q2q2 - az) + (-_4bx * q2 - _2bz * q0) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (_2bx *
     q1 + _2bz * q3) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q0 - _4bz * q2) * (_2bx * (q0q2
     + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
s3 = _2q1 * (2.0f * q1q3 - _2q0q2 - ax) + _2q2 * (2.0f * q0q1 + _2q2q3 - ay) + (-_4bx * q3 + _2bz * q1) * (_2bx *
     (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q0 + _2bz * q2) * (_2bx * (q1q2 - q0q3) + _2bz * (
     q0q1 + q2q3) - my) + _2bx * q1 * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);
recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3); // normalise step magnitude

```

6. /main/include/pid/pid.h

   (a) Pid_Run

   Run the pid feedback function.

Listing 17: Key code in the function

```
pidData->error = pidData->setPoint - input;

// Integral calcs

pidData->iTerm += (pidData->zKi * pidData->error);
// Perform min/max bound checking on integral term
if (pidData->iTerm > pidData->outMax)
    pidData->iTerm = pidData->outMax;
else if (pidData->iTerm < pidData->outMin)
    pidData->iTerm = pidData->outMin;

    pidData->inputChange = (input - pidData->prevInput);
    pidData->dTerm = -pidData->zKd*pidData->inputChange;

    // Compute PID Output
    pidData->output = pidData->prevOutput + pidData->zKp*pidData->error + pidData->iTerm + pidData->dTerm;

if (pidData->output > pidData->outMax)
```

```
    pidData->output = pidData->outMax;
else if (pidData->output < pidData->outMin)
    pidData->output = pidData->outMin;

    // Remember input value to next call
    pidData->prevInput = input;
    // Remember last output for next call
    pidData->prevOutput = pidData->output;

```

7. /main/include/python/send_image.py

   (a) send_image_main

   Send images to the server for image tracking.

Listing 18: Key code in the function

```
while 1:
    get_flag_value("http://fryer.ee.ucla.edu/rest/api/camera/get/")
time.sleep(1)
# print("get flag value success")
    if n_image_flag == 0:
    continue
    str_filename = get_str_filename()
CaptureImage(str_filename)
    UploadFile(str_filename, "http://fryer.ee.ucla.edu/rest/api/upload/")
    print("upload file success")

```

8. /main/car/car_control.h

   (a) GpsNavigationMove

   Auto navigate using the GPS sensor's data.

Listing 19: Key code in the function

```
d_west_to_east_distance = get_latitude_distance(pT_drone->d_current_latitude, pT_drone->d_destination_latitude,
     pT_drone->d_current_longitude);
d_south_to_north_distance = get_longitude_distance(pT_drone->d_current_longitude, pT_drone->d_destination_longitude,
      pT_drone->d_current_latitude);

if ((pT_drone->d_destination_longitude - pT_drone->d_current_longitude) > 0)
{
    pT_drone->d_move_direction = 90 - atan(d_south_to_north_distance / d_west_to_east_distance);
}else if ((pT_drone->d_destination_longitude - pT_drone->d_current_longitude) < 0)
{
    pT_drone->d_move_direction = 270 - atan(d_south_to_north_distance / d_west_to_east_distance);
}else if ((pT_drone->d_destination_longitude - pT_drone->d_current_longitude) == 0)
{
    if ((pT_drone->d_destination_latitude - pT_drone->d_current_latitude) >= 0)
    {
        pT_drone->d_move_direction = 0;
    }else if ((pT_drone->d_destination_latitude - pT_drone->d_current_latitude) < 0)
    {
        pT_drone->d_move_direction = 180;
    }
}
double tmp = abs(pT_drone->d_move_direction - pT_drone->d_face_direction);
double tmp_distance = sqrt(pow(d_west_to_east_distance, 2) + pow(d_south_to_north_distance, 2));
if (tmp > 15)
{
    //turn_direction(pT_drone->d_move_direction - pT_drone->d_face_direction);
    if(pT_drone->d_move_direction - pT_drone->d_face_direction > 0){
        turn_right();
    }else if(pT_drone->d_move_direction - pT_drone->d_face_direction < 0)
    {
```

```
        turn_left();
    }
}else if (tmp_distance > 10)
{
    printf("move forward\n");
    move_forward();
}

```

9. /Server/control

   (a) index.php

   Get or post the commands from or to the server.

Listing 20: Key code in the function

```
$mac_address = $_GET['mac_address'];
$control_type = $_POST['control_type'];
$auto_control_command = $_POST['auto_control_command'];
$manual_control_command = $_POST['manual_control_command'];
$suspend_pwm1 = $_POST['suspend_pwm1'];
$suspend_pwm2 = $_POST['suspend_pwm2'];
$suspend_pwm3 = $_POST['suspend_pwm3'];
$suspend_pwm4 = $_POST['suspend_pwm4'];

$conn = mysql_connect('localhost', 'webmaster', '');
mysql_select_db('edison', $conn);
$query = "UPDATE control
SET control_type = '$control_type', auto_control_command = '$auto_control_command', manual_control_command = '
    $manual_control_command', suspend_pwm1 = '$suspend_pwm1', suspend_pwm2 = '$suspend_pwm2', suspend_pwm3 = '
    $suspend_pwm3', suspend_pwm4 = '$suspend_pwm4', update_time = now()
WHERE mac_address = '$mac_address'";
$result = mysql_query($query);

```

10. /Server/gps_ubidots

    (a) index.php

    Post the GPS data from Edison to the server.

Listing 21: Key code in the function

```
$url = "http://things.ubidots.com/api/v1.6/variables/566328417625427550 9fc2ad/values";
$url = "http://things.ubidots.com/api/v1.6/variables/566328417625427550 9fc2ad/values";
$headers = array("Content-Type: application/json", "Accept: application/json; indent=4", "X-Auth-Token: 55
    QTnmUYv4kLQERtSqD3XAa27n8qnX");
$ch = curl_init();
$timeout = 30;
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $json_data);
$server_output = curl_exec($ch);
curl_close($ch);

```

11. /Server/ip_address

    (a) index.php

    Get or post the ip address of Edison from or to the server.

Listing 22: Key code in the function

```
$ip_address = $_POST['ip_address'];
$mac_address = $_POST['mac_address'];
$network_name = $_POST['network_name'];
$conn = mysql_connect('localhost', 'webmaster', '');
mysql_select_db('edison', $conn);
$query = "UPDATE ip_address
    SET ip_address = '$ip_address', mac_address = '$mac_address', network_name = '$network_name', update_time = now
      ()
    WHERE id = '1'";
$result = mysql_query($query);
```

12. /Server/pid_tuning

(a) index.php

Get or post the pid tuning parameters for pid feedback from or to the server.

Listing 23: Key code in the function

```
$kp_pitch = $_POST['kp_pitch'];
$ki_pitch = $_POST['ki_pitch'];
$kd_pitch = $_POST['kd_pitch'];
$kp_roll = $_POST['kp_roll'];
$ki_roll = $_POST['ki_roll'];
$kd_roll = $_POST['kd_roll'];
$kp_yaw = $_POST['kp_yaw'];
$ki_yaw = $_POST['ki_yaw'];
$kd_yaw = $_POST['kd_yaw'];
$kp_second_pitch = $_POST['kp_second_pitch'];
$kd_second_pitch = $_POST['kd_second_pitch'];
$kp_second_roll = $_POST['kp_second_roll'];
$kd_second_roll = $_POST['kd_second_roll'];
$kp_second_yaw = $_POST['kp_second_yaw'];
$kd_second_yaw = $_POST['kd_second_yaw'];
$conn = mysql_connect('localhost', 'webmaster', '');
mysql_select_db('edison', $conn);
deliver_response(200, "The pid_tuning commands have been updated", $response);
```

13. /Server/upload

(a) index.php

Post the image files from Edison to the server.

Listing 24: Key code in the function

```
//$reference_filename = $_GET['reference_filename'];
$image_filename = $_GET['image_filename'];
$targetfolder = "files/";
$targetfolder = $targetfolder.basename($_FILES['file']['name']);
if(move_uploaded_file($_FILES['file']['tmp_name'], $targetfolder)){
    echo "The file ".basename($_FILES['file']['name'])." is uploaded";
}
else{
    echo "Problem uploading file";
}
// close the session
session_write_close();
echo "<br><br>";
//$command = 'make -C opencv && ./opencv/symbol_detection';
$command = './opencv/symbol_detection reference.jpg ';
$command .= $image_filename;
exec($command, $out, $status);
$scene_corners_bottom_right_x = $out[0];
$scene_corners_bottom_right_y = $out[1];
$scene_corners_top_right_x = $out[2];
$scene_corners_top_right_y = $out[3];
```

```
$scene_corners_top_left_x = $out[4];
$scene_corners_top_left_y = $out[5];
$scene_corners_bottom_left_x = $out[6];
$scene_corners_bottom_left_y = $out[7];
```

14. /iOSApp

(a) Drone_control.xcode.project

Send commands and pid parameters from iPad/iPhone to server.

Listing 25: Key code in the function

```
forwardData = (int)(_slider1.value * 4000);
backwardData = (int)([_slider2 value] * 4000);
leftData = (int)([_slider3 value] * 4000);
rightData = (int)([_slider4 value] * 4000);
_stepper1.value = (int)([_slider1 value] * 4000);
_stepper2.value = (int)([_slider2 value] * 4000);
_stepper3.value = (int)([_slider3 value] * 4000);
_stepper4.value = (int)([_slider4 value] * 4000);
fpwm = forwardData * 0.1 /4000;
bpwm = backwardData * 0.1/4000;
lpwm = leftData * 0.1/4000;
rpwm = rightData * 0.1/4000;
total = (int)([_stepper_total value]);
AFHTTPRequestOperationManager *manager = [AFHTTPRequestOperationManager manager];
```

(b) Drone_onair.xcode.project

Send iPhone's GPS location data to server.

(c) control.xcode.project

Send commands and pid parameters from iPad/iPhone to server.

**Yang Yang:**

On air IOS application (used to get and send GPS information, taking pictures) and off-air remote control/test iOS (used to send control or test signal) application design; Smart drone state (according to sensor information) analysis and motor control signal output module implementation; Car's evadable system design and GPS navidation; Object detecting and tracking algorithm.

**Peidong Chen:**

Server side design, iOS application design, Edison and BeagleBone system code design. Automatic drone state analysis and motor control signal output module implementation; Car's evadable system design and GPS navigation; Object detecting and tracking algorithm.

**Yitian Hu:**

Data acquisition and analysis from MPU9025 9-axis sensor module; Apply AHRS sensor fusion algorithm based on 9-axis data from MPU9250. Smart drone state analysis (according to sensor information) and motor control state machine algorithm design; Drone assembly.

**Jiayu Guo:**

Algorithm programming for flight loop control; Sensor signal processing and noise calibration; Test the functionality of drone system; Ultrasonic module for distance detection; GPS module for drone's localization.

[1] robots that fly and cooperate. https://www.ted.com/talks/vijay_kumar_robots_that_fly_and_cooperate?language=en.

[2] David Gossow, Peter Decker, and Dietrich Paulus. An evaluation of open source surf implementations. In *RoboCup 2010: Robot Soccer World Cup XIV*, pages 169–179. Springer, 2011.

[3] OULD-DRIS Nouar, Ganoun Ali, and Canals Raphaël. Improved object tracking with camshift algorithm. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II. IEEE, 2006.

[4] Introduction to surf (speeded-up robust features). http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.