


Cairo University

Medical Drone with Object Detection and Recognition

Abdelrahman Ashraf Mahmoud
Khaled Ali
Mahmoud Hassan
Mostafa Marey
Yousef Abdelghany

Supervised by:
Prof. Hassan Mostafa
Prof. Samah El-Shafiey

A thesis submitted in the fulfilment of the requirements
for the degree of Bachelor of Science in Engineering by Research

in the

Faculty of Engineering of Cairo University
Electronics and Communication Department

August 2020

Abstract

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year. Regular methods nowadays, such as an ambulance, are not effective. Currently, the survival rates are between 5% to 8%. Therefore, what is needed is to provide necessary aid as swiftly as possible in less than 9 minutes. This can be made possible by sending a flying drone carrying necessary medical kits, such as the automated external defibrillator which is key to saving a cardiac arrest patient. This will increase the survival rate from 8% to 67%.

Table of Contents

Introduction

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year. However, what is more concerning is that 85% of the CVD victims are due to heart strokes or cardiac arrest[1].

Regular methods nowadays, such as an ambulance, are not effective. Currently, the survival rates are between 5% to 8%. Such low survival rates are due to the absence of immediate help within the time period within which the patient could be saved. A cardiac arrest victim has a time approximate of 9 minutes, any later than that the patient suffers from irreversible brain damage and then a coma and ultimately death. Such a patient, if saved after 9 minutes , will suffer from irreversible brain damage, left unable to function as before[1].

The goal in an emergency is to stabilize the patient, and then to move them where they need to be. As 9 minutes is such a short period of time, it is unwise to use regular methods. According to the statistics, the average time of arrival of an ambulance is 15 minutes. In Egypt, the average time of arrival is 45 minutes, and in some cases 3.8 hours[2].

Therefore, what is needed is to provide necessary aid as swiftly as possible, which is less than 9 minutes. This can be made possible by sending a flying drone carrying necessary medical kits, such as the automated external defibrillator which is key to saving a cardiac arrest patient, to the emergency scene, avoiding all obstacles on the ground and reaching through the shortest path.

The goal is to design and implement a fully functioning drone solely made for such purpose. A system capable of receiving coordinates to the emergency scene, calculate the shortest path and autonomously drive the drone to the scene while being supported with machine learning algorithms in order to predict and avoid potential obstacles along the way, ensuring safe arrival of the drone and the rescue of the patient.

In this thesis, it is going to be explained in fine details how this project came to life. Provided that all the components used were standalone and each component was handpicked and customly built for this purpose. Keeping in mind that no readymade libraries were used such as PX4 or products of this sort.

1. Literature Review

However novel the idea might be, others have worked on this concept and achieved significant milestones. In this section, different projects of the same target are going to be mentioned.

1.1. TU Delft Ambulance Drone

A project led by “Alec Momont”, a student of TU Delft. His project features a custom design drone for the purpose of high speed flights in a straight line. “With the Ambulance Drone, we want to dramatically increase this survival rate. The incorporation of a two-way, video supported, communication channel in the drone between 112 operators and the first responders will improve first care. Successful AED usage by lay-persons is currently at 20%. With personalised instructions and communication on the Ambulance Drone, this can be increased to 90%. In short, the Ambulance Drone helps to save lives by extending existing emergency infrastructure with a network of fast and compact UAVs capable of bringing emergency supplies and establishing communication, anywhere”[3].

1.2. SOS Alarm

SOS Alarm, the company that operates Sweden’s 112 emergency number, is to use drones to deliver Automated External Defibrillators (AED) for out-of-hospital cardiac arrests (OHCA) for the first time in a trial that began in June of 2020. The drones will complement existing ambulance dispatching, and be activated when a 112 call taker suspects an OHCA. When this happens, the drone will use GPS technology and advanced camera systems to navigate to the scene of the incident, delivering the AED exactly where it is needed – lowered by a winching device while the drone hovers 30m above. The trial will run between June and September, when the results will be evaluated before further expansion[4].

The above mentioned projects are the two most significant efforts in the idea of utilizing drones to save lives. It was only as of recent that the Swedish government took initiative into implementing such a drone network as stated above. Professor Mouhammed Derawi also predicts that within the next five years, drones will be able to transport patients[5].

2. Overall System Architecture

This project is divided into two main parts, the drone programming and path finding coupled with object detection and recognition. The drone is custom build design, which means that the components used are standalone and no ready made processors, libraries or code were used. However, it is difficult to achieve such goals in a small period of time without relying on special tools and simulations which are going to be mentioned in detail in the following subsections.

2.1. Assembly

2.1.1. Processor

Starting at the core of this project, the processor is the main driver for this UAV. The processor used was STM32 of the F1 family, chip 3c8t6 which features a clock frequency of 72 Mhz. Unfortunately, the decision made to use this processor was not out of choice but simply as it is the only kind available locally. Therefore, the results and numbers associated with tests done on the drone are exclusive to this processor and better results will arise upon switching this processor with a better one.

2.1.2. Motors

For our case, it is important to take into consideration the heavy payload that the drone has to support along its journey. A professional calculator[6] was used to roughly estimate how a combination of components would behave as a group and measures it in terms of flight time, thrust to weight ratio, estimated temperature and power dissipation. Brushless DC motors are used, specifically for their ability to achieve high RPM under heavy payload. 1250 KV motors were used which weigh about 40 gm each. The number 1250 translates to how much is the RPM per volt. There are two kinds of motors: High KV and Low KV, high KV motors are fast and suitable for light drones that are mainly used for racing which might need to maneuver frequently. Low KV motors are more associated with stability and heavy lifting as they produce higher torque but less speed. At 1250 KV, 21 minutes of flight time are achievable at an average speed of 51 Km/hr with a thrust-to-weight ratio of 3.6.

2.1.3. Battery

To supply the needed power, a large battery is needed. Although it adds more weight however it adds more into our flight time, however, there is a limit as to how large of a battery can be used as the motors can carry a limited weight. In addition, the motors are specifically chosen for heavy lifting, which directly

translates to high power consumption, and therefore high demand for electricity which is an important factor while choosing the battery. In our case, each motor needs 25 Amps in order to operate freely between minimum and maximum speeds. Hence, for four motors, a battery that can discharge 100 Amps at a time is needed. To be able to achieve that, a battery's capacity and C-rating should be able to provide us with that. Therefore, a 5000 mAh battery was chosen with a 40C rating. To check whether it would be enough or not, the capacity in Ampere hours is multiplied with the C-rating, the result should be in Amps, it is expected to be more than 100 amperes, keeping in mind that a motor can suddenly ask for more and the battery should sustain that to avoid burning of the motor.

example: 5 x 40 = 200 Amps

2.1.4. Electronic speed controller

Due to the complicated nature of brushless motors, it would be difficult to control it through the main processor. Therefore, each motor requires a processor of its own that can interface with and feed it the required current. These processors are called ESCs, electronic speed controllers, which mainly are ARM processors specifically designed to interface with brushless motors. To choose a proper ESC, it is essential to take into consideration the amount of current a motor might need and expect the worst case scenario. Consequently, the ESCs that were used can operate up to 30 amperes.

2.1.5. Sensors

To feed back the drone with accurate reading about its whereabouts, several sensors were used including Accelerometer, Gyroscope, Barometer, Magnetometer and GPS. These sensors are described with more details in section 6.

2.1.6. Propellers

As our frame is 40cm in diagonal, the most suitable propellers are called "1045", which translates to 10 inch in length and 4.5 inch in pitch. With such propellers it enables us to achieve the numbers mentioned previously. However, the kinds of materials and quality that make the propellers are key to a better flight.

2.1.7. Computer

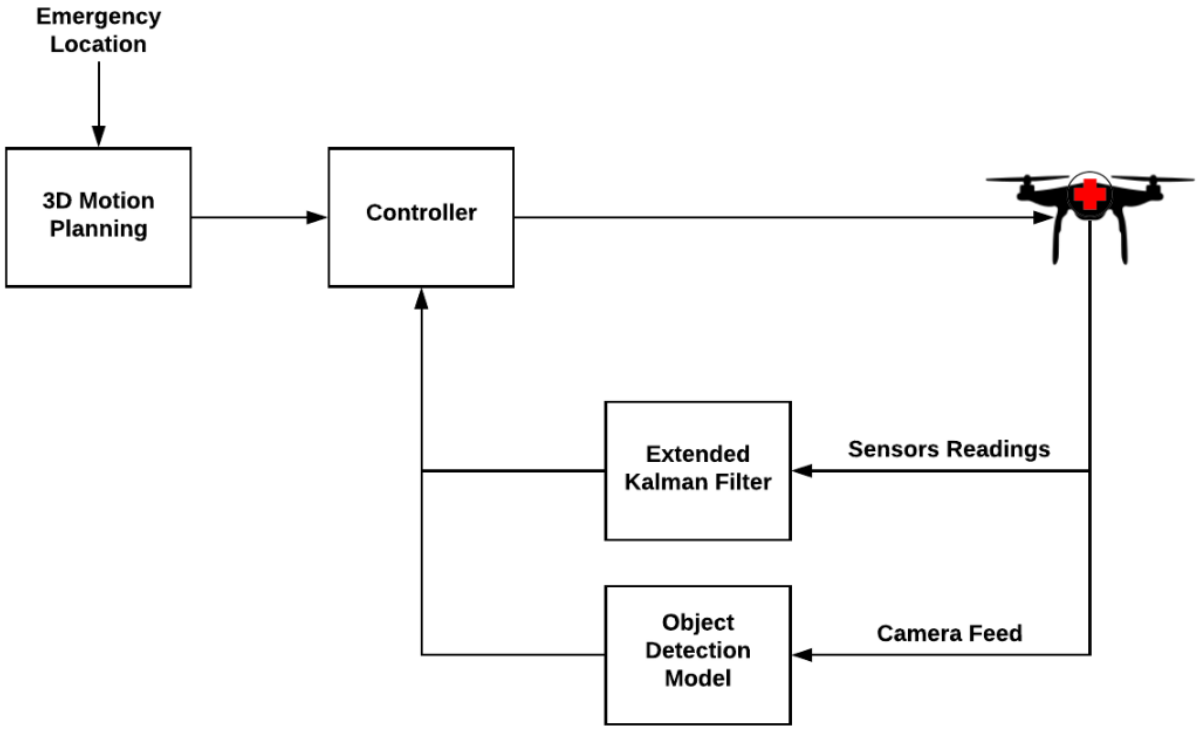
Raspberry Pi 3 was first used in order to capture a video with Pi Camera and run a tensorflow model on it, but due to the slow operation of Raspberry Pi 3 in comparison to Raspberry Pi 4, Raspberry Pi 4 was used with Broadcom BCM2711

Quad core Cortex-A72 @ 1.5GHz and memory of 4GB LPDDR4-2400 SDRAM, this will be illustrated in details in section 9.

2.1.8. 4G module

A Raspberry Pi HAT (Hardware attached on top) is an expansion board that is specifically made and tested to work with a certain version of Raspberry Pi. They are the same as shields in the Arduino universe. HATs can be quite expensive, so a simple consumer-grade USB modem was used to get the Raspberry Pi connected. There is a wide variety of models out there to choose from. This allowed connecting the Pi in the same way any user would at home.

3. Implementation Details



overview of the project

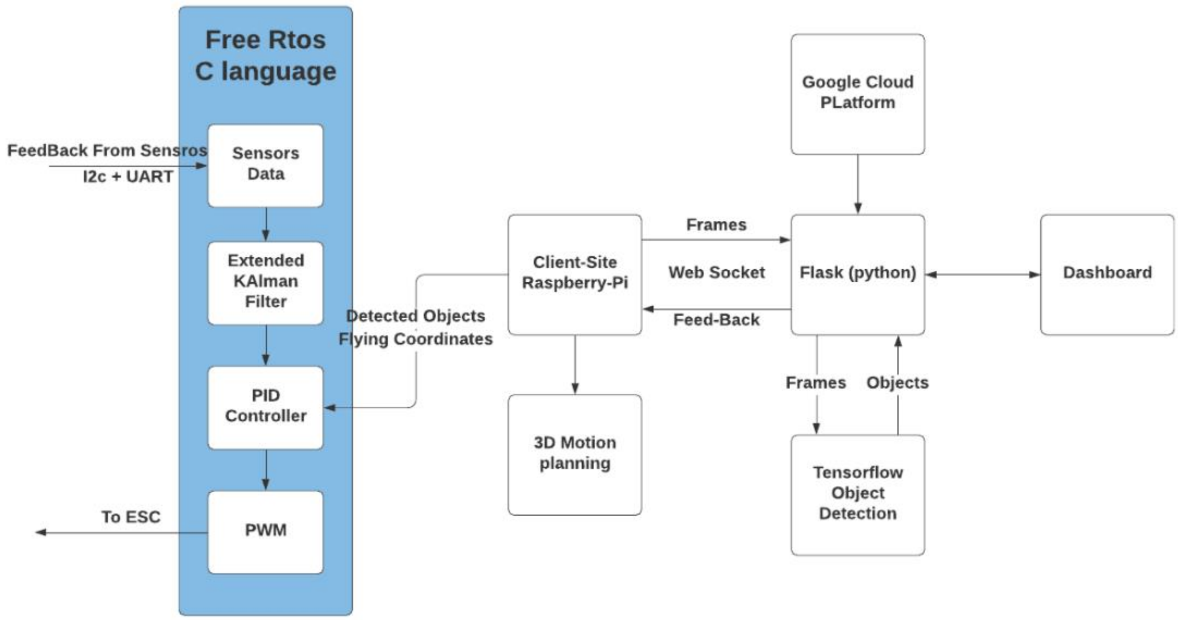
3.1. Overview

The main goal is to deliver the necessary medical equipment to the place of emergency. The PID controller, which is the main pilot of our drone, uses the target coordinates given by the 3D motion planning model along with current coordinates given by the feedback from the sensors through the kalman filter.

To organize these blocks together, time becomes critical. As the presence of multiple integration processes in the program, it becomes necessary to keep the timing on point and accurate. Therefore, a real time operating system was used and a total of 10 tasks work in harmony which will be explained later.

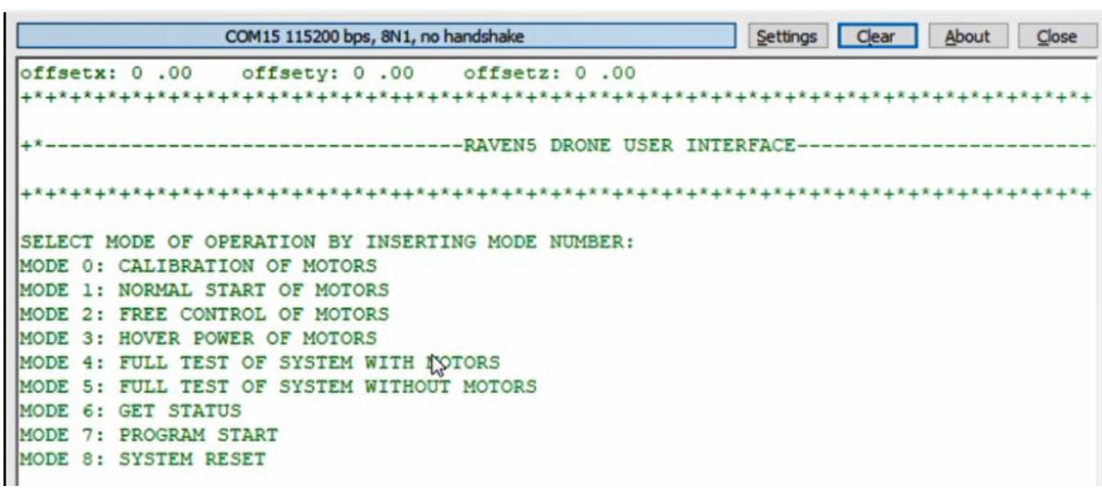
3.2. Software

To get a better intuition of how the project is like from the inside, the following diagram illustrates the key elements of the system.



System controlling the project

The program of the drone was written in embedded C, as it is flexible, low level and allows us to get creative while also being compatible with microcontrollers such as ARM microprocessors as in our case. With the specifications mentioned before, the block diagram above is capable of running 1000 times per second, which means that the drone makes 1000 decisions per second giving it a high response rate to changes and quick reaction to obstacles.



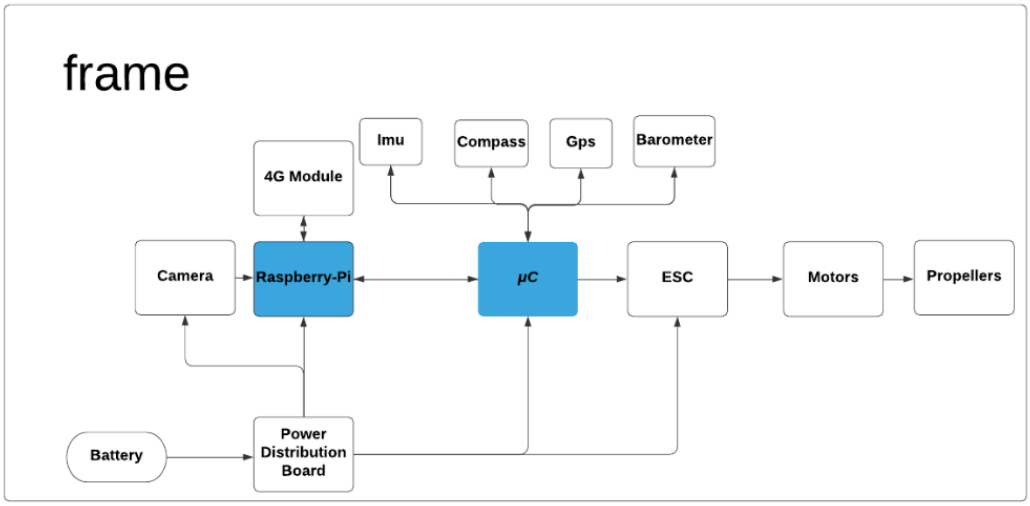
User interface

The program starts by prompting the user between 9 different modes of operation. One of them for running normally and 4 are for testing purposes while the rest are for overriding

the system with a forced reset, calibrating the motors, arming the motors or simply get a look over the system flags at a certain moment. Hence, it becomes easier for the verification process.

After a mode is chosen, given that it allows the program to run fully, the sensors start storing readings which are then passed to the kalman filter for sensor fusion. Consequently, these readings are then passed as feedback to the PID controller which in turn produces the required forces to the motors which depends on the difference between its current position and orientation and the target position. More will be explained in the concerned sections.

3.3. Hardware



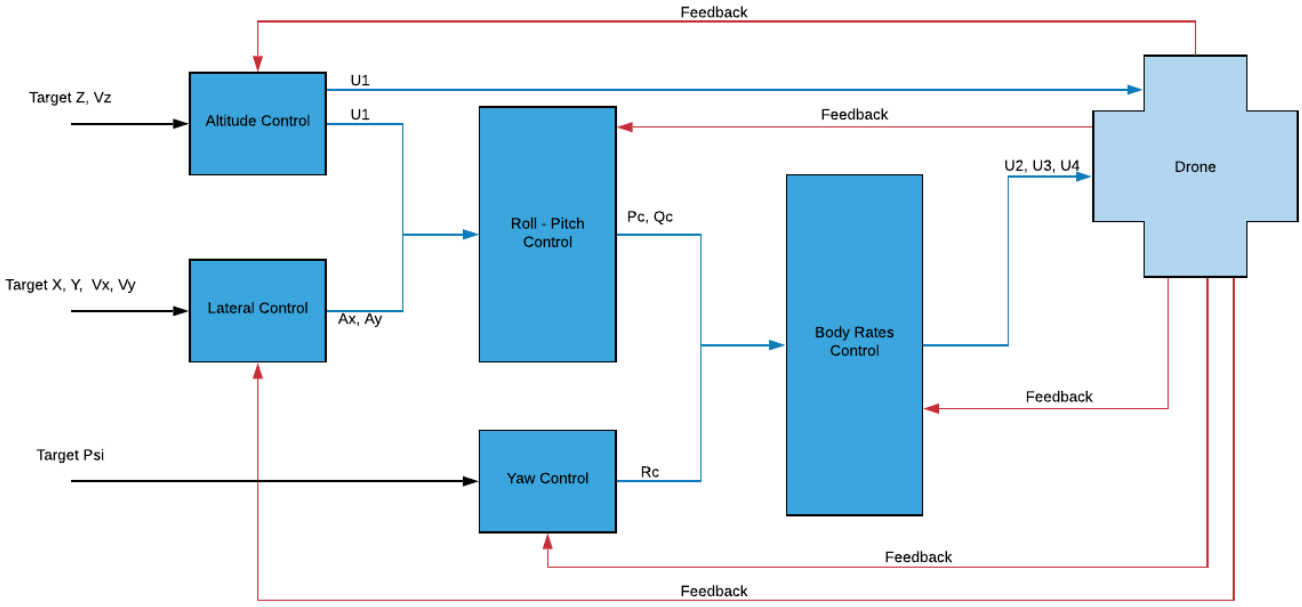
Project Hardware

The battery feeds the power distribution boards which then feeds the entire circuit. Mainly the motors, which require most of the battery, and the camera along with the transmitter and receiver. Then comes the microprocessors, the STM32 and the Raspberry Pi 4, which in turn feed their peripherals and connected components. The STM32 is connected to all the sensors as shown, while the RP4 is concerned mainly with the 4G module which in our case is a USB modem. The STM32 interfaces with the ESCs through a PWM signal that decides the speed of the motors based on the width of the pulse, keeping in consideration that the ESC can deal with signals of 50 - 60 Hz.

4. Controller

The last part of the project is the PID controller, however, it is important to design it firsthand to understand its requirements and meet at a common ground. Starting with the big picture, the PID controller needs pure and accurate readings of the current position and orientation in order to be capable of estimating the required forces needed by the motors to achieve the required motion.

The PID controller in this case is not exactly a PID, but a combination of PD and P controllers as they are sufficient to do nearly the same results and much less computationally expensive. Following is a block diagram of the blocks forming the PID controller:



PID Controller

4.1. Altitude Control

In this block, only one goal is of concern, it takes the target position in the Z-direction and calculates the difference in distance between the drone and the target, inserts this difference in a PD equation which depends on unknown variable gains to be multiplied by this difference in order to produce the result U1, which is the upward thrust, the equation is as follows:

$$A_z = K_p \times (Z_{Target} - Z_{current}) + K_d \times (Vz_{target} - Vz_{current})$$

The PD controller results in an acceleration, therefore, following this step is transforming the acceleration into a thrust force which is done through this equation:

$$R = \frac{\cos(phi)}{\cos(theta)}, \; U1 = mass \times \frac{(A_z - 1)}{R}$$

The subtracted 1 in the equation is to remove the effect of gravity, as it is assumed that the drone is hovering.

4.2. Lateral Control

The first block was concerned with vertical movement, in this block the 2D plane perpendicular to the Z-axis is handled as the motion in this plane is different conceptually. For the drone to move in this plane, it needs to create a moment around these axes in order to generate a tilt in a proposed direction which will consequently lead to a motion, an accelerated motion, in this direction. Therefore, the same PD control concept is used, however, they are not directly transformed into forces, but use the acquired accelerations as an input to the Roll-Pitch block that is concerned with the transformation of accelerations into angular velocities.

$$A_x = K_p \times (X_{Target} - X_{current}) + K_d \times (Vx_{target} - Vx_{current})$$
$$A_y = K_p \times (Y_{Target} - Y_{current}) + K_d \times (Vy_{target} - Vy_{current})$$

4.3. Roll-Pitch Control

This block is the core block of this controller along with body rate control as it transforms the accelerations into angular velocities which are then passed to the body rate controller to output the required moments.

$$P_c = \frac{1}{R_{33}} \times (R_{21} \times b_{Vx} - R_{11} \times b_{Vy}),\ where\ R_{21} = 0,\ R_{11} = 1$$

$$Q_c = \frac{1}{R_{33}} \times (R_{22} \times b_{Vx} - R_{12} \times b_{Vy}),\ where\ R_{22} = \cos(phi),\ R_{12} = \sin(phi) \times \frac{\sin(theta)}{\cos(theta)}$$

$$where,\ R_{33} = \frac{\cos(phi)}{\cos(theta)},\ b_{Vx} = \frac{\left(\cos(phi) \times \frac{\sin(theta)}{\cos(theta)} - A_x \times \frac{mass}{U1}\right)}{\tau}$$

$$b_{Vy} = \frac{\left(-\sin(phi) - A_y \times \frac{mass}{U1}\right)}{\tau}\ and\ \tau = \frac{1}{K_{bank}}$$

4.4. Yaw Control

The simplest block of the Controller as it is only concerned with transforming the required Psi (heading) into an angular velocity which is a simple P equation.

$$R_c = K_p \times (\psi_{target} - \psi_{current})$$

4.5. Body Rate Control

Finally after acquiring the needed angular velocities, it is possible to calculate the corresponding moments around each axis of the drone in order to produce the required tilt which should consequently lead to the desired motion.

$$U2 = I_{xx} \times \dot{P}, \text{ where } \dot{P} = K_{pp} \times (P_c - P)$$

$$U3 = I_{yy} \times \dot{Q}, \text{where } \dot{Q} = K_{pq} \times (Q_c - Q)$$

$$U4 = I_{zz} \times \dot{R}, \text{where } \dot{R} = K_{pr} \times (R_c - R) \text{ while } I_{xx}, I_{yy}, I_{zz} \text{ are design constants}$$

Having acquired the desired upward thrust and moments around each axis, it is needed to find the required force for each motor.

$$L = \frac{\text{Length of arm}}{\sqrt{2}}$$

$$F_1 = \frac{U2}{L}, F_2 = \frac{U3}{L}, F_3 = -\frac{U4}{k_{thrust}}, F_4 = U1$$

$$\text{therefore, } Motor_1 = \frac{(F_1 + F_2 + F_3 + F_4)}{4}$$

$$Motor_2 = \frac{(-F_1 + F_2 - F_3 + F_4)}{4}$$

$$Motor_3 = \frac{(F_1 - F_2 + F_3 - F_4)}{4}$$

$$Motor_4 = \frac{(-F_1 - F_2 + F_3 + F_4)}{4}$$

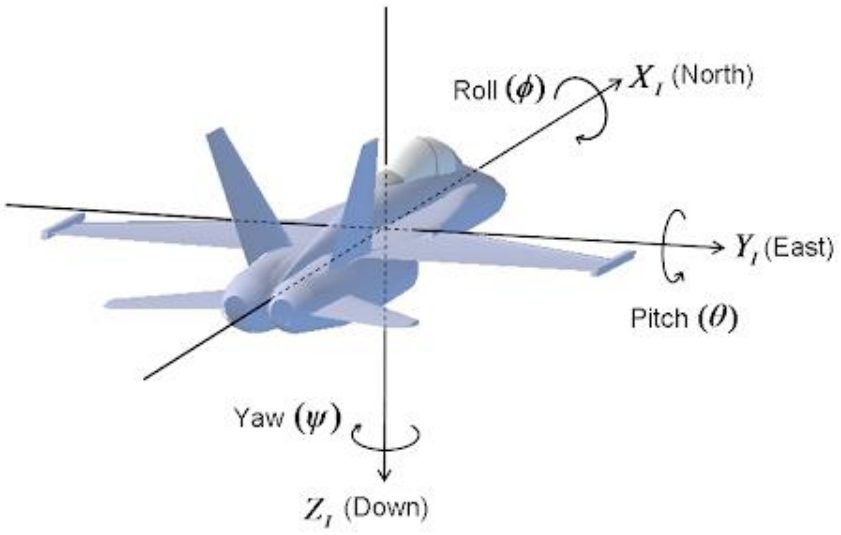
5. Sensors

This project has utilized a number of sensors to get rigid info about both the drone's position and orientation. For all the sensors ahead, calibration should be done in the beginning of the codes to get the offset and then subtract it from all the future readings.

5.1. Acceleration

Acceleration is the rate of change of the velocity of an object with respect to time. It can be obtained from the Inertial Measurement Unit (IMU6050) in 3 Axis. When the drone isn't moving, the Acceleration in the Z Axis will always be pointing upwards with the value 9.8 m/s^2. This can be used to get the Roll angle of the drone around both x axis (phi) and Pitch angle around y axis (theta) using these formulas:

$$\text{phi} = \text{atan2}\left(\frac{Accy}{\sqrt{Accx^2 + accy^2}}\right) \qquad \text{theta} = \text{atan2}\left(\frac{-Accx}{\sqrt{Accz^2 + accy^2}}\right)$$



Although this method cannot be accurate while the drone is accelerating in the x and y axes, it has an essential advantage that it is drift free due to the absence of any integrations.

These readings can be helpful for the drone by utilizing it in calculating both the drone Velocity and Displacement in three dimensions using the equations ahead:

$$velocity = \int Acc*dt \qquad Position = \iint Acc*dt$$

These equations can be used if the sensor's readings are ideal and have no errors, in real life that is impossible to achieve. For the IMU chip used in the project, it has an error equal to 1 degree which will result in a high error accumulation with time. Google has mentioned in a talk regarding the IMU chips in android phones, that these sensors cannot be used to track the users displacement and showcased the errors accumulated statistics as follow:

Angle Error (degrees)	Acceleration Error (m/s/s)	Velocity Error (m/s) at 10 seconds	Position Error (m) at 10 seconds	Position Error (m) at 1 minute	Position Error (m) at 10 minutes	Position Error (m) at 1 hour
0.1	0.017	0.17	1.7	61.2	6120	220 e 3
0.5	0.086	0.86	8.6	309.6	30960	1.1 e 6
1.0	0.17	1.7	17	612	61200	2.2 e 6
1.5	0.256	2.56	25.6	921.6	92160	3.3 e 6
2.0	0.342	3.42	34.2	1231.2	123120	4.4 e 6
3.0	0.513	5.13	51.3	1846.8	184680	6.6 e 6
5.0	0.854	8.54	85.4	3074.4	307440	11 e 6

Accumulated error with time

From the previous table, it can be noticed that it will have an accumulation equal to 17 meters in only 10 seconds which is not acceptable for a moving vehicle.

5.2. Angular Velocity

angular velocity refers to how fast an object rotates or revolves relative to another point, in this project, it refers to the speed of the drone rotation around its x,y and z axes.These readings can also be obtained using the IMU6050 chip. They are useful to get the tilting angles roll and pitch which were mentioned before and also the yaw (angle around z axis).
The following integrations should be calculated to obtain these angles:

$$\text{phi} = \int p\,dt \qquad \text{theta} = \int q\,dt \qquad \text{psi} = \int r\,dt$$

Any integration should be done with a constant time. For example if the *dt* equals to 1ms, the integration functions should be called every one millisecond to get accurate results.

Calculating angles with the angular velocities isa very accurate compared to the acceleration method, the only disadvantage is that it is prone to drift because of the integration error.

Due to the high vibration, anti vibration pads should be attached between the controller and the drone to allow the IMU chip to work correctly. If no pads are attached, the readings will start fluctuating with an error around 10 degrees.

It can be noticed now that there are 2 methods of calculating the tilting angles of the drone. Which one should be used? This will be discussed in section 7.1.

5.3. Body frame to world frame

All the previous calculations were regarding the drone’s Axes and not the world’s north and East axes. This is totally acceptable if the drone was to be controlled manually, but in order for the drone to be able to understand the trajectories that's obtained from the GPS, the drone’s axes should be transformed to the world's axes. This is a mandatory step to make the drone autonomous.

Firstly, it is needed to transform 3 angles from the body frame to the world frame. However, it is important to note that this cannot be done directly from the Euler form using the following transformation matrix due to several reasons:

$$D(\phi,\theta,\psi) = \begin{pmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{pmatrix}$$

Rotation matrix for Euler form[7]

It can be seen from the element (3,2) and (3,3) that if theta happens to be 90 degrees, as cos(90) is 0 which will lead to division by zero. This problem is called “Gimbal lock” and it happens when 2 different axes align with each other. Therefore, to avoid this issue, it becomes necessary to transform the angles to Quaternion form using the following matrix:

$$
\begin{aligned}
q_i &= \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\
q_j &= \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\
q_k &= \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \\
q_r &= \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2}
\end{aligned}
$$

Transformation matrix from Euler to Quaternion form[7]

The obtained angles represented in 4 variables, which will be multiplied with the rotational matrix for the quaternions:

$$Q = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw \\ 2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2xw \\ 2xz - 2yw & 2yz + 2xw & 1 - 2x^2 - 2y^2 \end{bmatrix}.$$

Rotation matrix for quaternion from Body frame to world frame[7]

This Matrix can also be used to get the Accelerations to the world frame so both the velocity and the displacement can be obtained with respect to the world axes.

5.4. Heading Angle

The drone should always be able to identify its heading in order to face the camera to the side it's flying towards. For this purpose, the magnetometer HMC5883 was used. This sensor feeds back the magnetic field in the 3 axes with respect to the body frame. The ratio between each axis in a specific direction is constant, this can be used to know the heading of the chip.

Using the *atan(magnetic.y/magnetic.x)*, the heading (yaw) can be obtained if the drone is not tilted. Once the drone tilts horizontally, the ratios between the magnetic fields in x and y directions will start to change although the drone is still facing the same angle. To compensate for that, The magnetic field in the z direction should be analyzed, with the help of both Phi and Theta angles to get the new x and y components for the atan using the following equations:

$$X = mag.x * \cos(pitch) + mag.z * \sin(pitch)$$

$$Y = mag.x * \sin(roll) * \sin(pitch) + mag.y * \cos(roll) + mag.z * \sin(roll) * \cos(pitch)$$

It should be mentioned that not all magnetometer chips have the same z axis direction, one should take into consideration if the chip's z axis points upwards or downwards.

Magnetometers should be calibrated regularly to get the new surrounding magnetic fields. This step can be done by measuring the maximum and minimum fields in each axis while rotating the chip in all directions. Add the max and min and divide by 2 to get the offset and then start subtracting it from all the future measurements.

Due to the difference between the geographical north direction and magnetic north. There will be a difference known as the “declination angle”. To compensate for this angle, it should be subtracted from the final yaw angle.

5.5. GPS

As mentioned before, obtaining the displacement using the IMU sensor is not efficient, apart from the readings not being with respect to the world's axes. For the first step, Neo-7m GPS sensor was used, which can communicate using UART protocol. Following the NMEA format, time, longitude altitude and number of fixed satellites can be extracted.

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
```

To change the formats being received from the module, U-centre Application should be used which can be found in u-blox site (the manufacturer of the module). The frequency was set to 10Hz instead of the default 1Hz and to receive the format as a NAV-PVT.

This step should be done every time the GPS is configured, so instead, the configuration data can be extracted while being sent from the U-centre to the module and use them as an initial function in the code.

39.7 NAV-PVT (0x01 0x07)

39.7.1 Navigation Position Velocity Time Solution

Message	**NAV-PVT**				
Description	**Navigation Position Velocity Time Solution**				
Firmware	Supported on: • u-blox 7 firmware version 1.00				
Type	Periodic/Polled				
Comment	**Note that during a leap second there may be more (or less) than 60 seconds in a minute; see the description of leap seconds for details.** This message combines position, velocity and time solution, including accuracy figures				
Message Structure	*Header* 0xB5 0x62	*ID* 0x01 0x07	*Length (Bytes)* 84	*Payload* *see below*	*Checksum* CK_A CK_B

Payload Contents:

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U4	-	iTOW	ms	GPS time of week of the navigation epoch. See the description of iTOW for details.
4	U2	-	year	y	Year (UTC)
6	U1	-	month	month	Month, range 1..12 (UTC)
7	U1	-	day	d	Day of month, range 1..31 (UTC)
8	U1	-	hour	h	Hour of day, range 0..23 (UTC)
9	U1	-	min	min	Minute of hour, range 0..59 (UTC)
10	U1	-	sec	s	Seconds of minute, range 0..60 (UTC)
11	X1	-	valid	-	Validity Flags (see graphic below)
12	U4	-	tAcc	ns	Time accuracy estimate (UTC)
16	I4	-	nano	ns	Fraction of second, range -1e9 .. 1e9 (UTC)
20	U1	-	fixType	-	GNSSfix Type, range 0..5 0x00 = No Fix 0x01 = Dead Reckoning only 0x02 = 2D-Fix 0x03 = 3D-Fix 0x04 = GNSS + dead reckoning combined 0x05 = Time only fix 0x06..0xff: reserved
21	X1	-	flags	-	Fix Status Flags (see graphic below)
22	U1	-	reserved1	-	Reserved
23	U1	-	numSV	-	Number of satellites used in Nav Solution

U-blox NAV-PVT packet section

From the Table above, the required data can be extracted from the packet sent based on its byte number, but first it is mandatory to add 6 bytes for the packet header.

Receiving the packet from UART requires a periodic check on the UART buffer which will suspend the operation systems tasks. This is inefficient because the operating system is supposed to be a real time operating system. To solve this, ARM microcontrollers offer DMA (Direct Memory Access) inside its chips which can be used to store the reading from the UART directly to a specific array of your choice without the need of your CPU interaction. This data can be accessed whenever needed.

5.6. Altitude

Measuring the altitude cannot be done with neither the IMU nor the GPS due to its 2 meters error, therefore an extra altitude sensor should be used which is the Barometer (Bmp180) that can feedback the current pressure and temperature. After obtaining the pressure, the following equation can be used to obtain the height with respect to the sea level:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Where P_0 represents the pressure of the sea surface. Measuring the altitude of the drone was one of the challenging processes due to the inaccuracy of the barometer which is ± 1 meter.

7. Filters

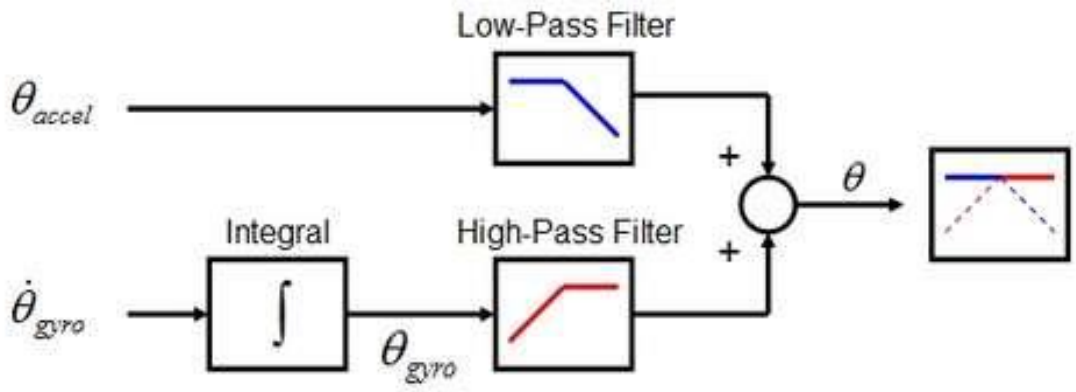
For the PID controller to work, it depends mainly on the feedback given by the drone. However, these feedback readings cannot be used as they are, but need to be passed through filters of sorts in order to refine them into the required information.

The challenging problem is that multiple sensors have the same output, but each has a different problem in addition to all of them being noisy, therefore, it is needed to estimate the drone's state (position, velocity, orientation) from these sensors.

7.1. Complementary Filter

It is noticed from the previous section that there are two methods for obtaining roll and pitch angles and another two for the yaw angle. As mentioned before the advantage of the angles obtained from the accelerations is that they are accumulation-free, so what is needed is to extract the stable part using a low-pass filter.

For the angles from the gyroscope, the advantage was that they provide accurate measurements but keep accumulating. To get rid of this accumulation, which can be represented as a DC part, a high pass filter will be used.



Complementary Filter

The previous process can be done in code by expressing the low pass filter as a 1% of the total value, while the highpass filter as a 99% of the total value as follows:

$$Angle = 0.99 * GyroAngle + 0.01 * Acc\ Angle$$

7.2. Kalman Filter

The best approach for estimation is through Bayes filters. Bayes filters estimate the drone's state based on the drone's movements. Intuitively, if the drone was told to go up, for instance, then it is believed that the drone's position should also go up. The Bayes Filter gives a way to incorporate motion prediction into the state estimate. First the next state is predicted, given a control input, the current state, and a model of how the system evolves over time. It does not maintain a point estimate but rather a belief or distribution of the estimate. Then, revise the prediction with an update from an observation. The update method takes the previous estimate from prediction, and an observed sensor value. It returns a new distribution that takes into account the sensor value, using a model of how the sensor works.[8]

The bayes filter that was used in this project is The Kalman Filter(KF). But there are 12 variables (positions, velocities, orientation, angular velocities) about the drone state that are needed as feedback to the controller which will be computationally intensive mathematically as well as time consuming. Therefore, by distributing the variables, obtaining the p, q and r directly from the gyroscope as they do not need any computations to obtain them.

The orientation$\Phi, \theta \; and \; \psi$are going to be obtained as mentioned before using a complementary filter to estimate their values because the errors in their measurements are not severe unlike the positions. This leaves only the positions and velocities to be estimated using the KF and that does not just reduce the input state vector to the KF but rather reduces much of the nonlinearity between the orientation and the other variables. In fact, this makes the equations completely linear which makes things much easier because otherwise a different filter would have to be used. The Extended Kalman Filter (EKF) which is similar to the KF but deals with nonlinear equations and harder to implement than the KF.

The Kalman Filter assumes that all the variables and sensor measurements are gaussian distributions with their means as the most probable values for each state and their variance to represent how trustworthy the estimated value is.

A pseudocode for the KF algorithm:

1: **function** PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
2: $\quad \bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\quad G_t = g'(u_t, x_t, \Delta t)$
4: $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
5: $\quad$ **return** $\bar{\mu}_t, \bar{\Sigma}_t$
6: **function** UPDATE($\bar{\mu}_t, \bar{\Sigma}_t, z_t$)
7: $\quad H_t = h'(\bar{\mu}_t)$
8: $\quad K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$
9: $\quad \mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
10: $\quad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
11: $\quad$ **return** μ_t, Σ_t
12: **function** EXTENDEDKALMANFILTER
13: $\quad u_t = \text{COMPUTECONTROL}(\mu_{t-1}, \Sigma_{t-1})$
14: $\quad \bar{\mu}_t, \bar{\Sigma}_t = \text{PREDICT}(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t)$
15: $\quad z_t = \text{READSENSOR}()$
16: $\quad \mu_t, \Sigma_t = \text{UPDATE}(\bar{\mu}_t, \bar{\Sigma}_t, z_t)$

Kalman Filter algorithm

u_t	The control input at time t.
x_t	The state at time t.
z_t	The observation at time t.
$x_{t,\text{variable}}$	The scalar value of the state vector at the index corresponding to variable. Similar notation for u_t and z_t.
Δt	The elapsed time between updates in seconds.
Q_t	Transition model covariance
R_t	Measurement model covariance
$g(x_t, u_t, \Delta t)$	The transition function.
$h(x_t)$	The observation function.

The way the algorithm works is by computing the control input as a start which in this case is the acceleration in the x, y and z directions measured by the IMU. Then the predict function is called and passed the previous drone state as input at time t -1 in the form of μ_{t-1} and $\sum_{t-1}$. Since everything is in gaussian along with the control input that was just computed and the time Δt. Firstly, the function uses the transition model function ‘g’ which returns the predicted state at time t and in this case that is done simply by integrating the velocity, adding it to the position and integrating the control input(the acceleration) to add it to the velocity.

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_{t,x} + x_{t,\dot{x}}\Delta t \\ x_{t,y} + x_{t,\dot{y}}\Delta t \\ x_{t,z} + x_{t,\dot{z}}\Delta t \\ x_{t,\dot{x}} \\ x_{t,\dot{y}} \\ x_{t,\dot{z}} - g\Delta t \end{bmatrix}$$

Since the equation is linear it is possible to rewire the equation as:

$$g(x_t, u_t, \Delta t) = A\,x_t + Bu_t$$

Then $g'(x_t, u_t, \Delta t)$ is simply the matrix A which will be used to get the predicted covariance matrix.

$$g' = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Consequently, to obtain the sensor measurement which in this case is the GPS measurements of the drone state as well as the barometer to get the measurement of the altitude in particular. Lastly, the update function is called and passed the predicted state that was obtained from the predict function along with the GPS measurements to calculate the estimated values for the state.

8. 3D Motion Planning

8.1. Overview

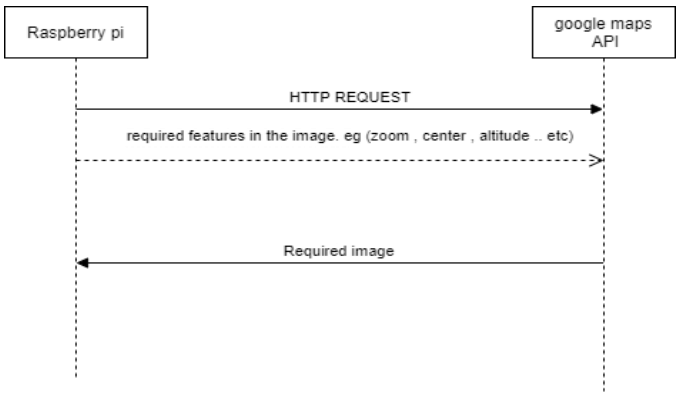
In order for the drone to be capable of flying autonomously and arrive on time, path planning was the solution. Path planning is a computational problem to find a sequence of valid trajectory points in the space that moves the drone from the start to destination. satellite imagery was used to get an image from a satellite at the required location, this image is then discretized into a grid, and each grid point (node) is labeled as a path or an obstacle. Search algorithms are used to find a path from the start to the goal.

There are many search algorithms that find this path. The fastest calculating algorithm was not the highest first priority, but the shortest path. A computationally efficient search will not affect the overall time as the fastest path.

Therefore, optimized map images had to be used for the computations not to be intensive and with a quality that provides the right path. Therefore, a suitable algorithm was chosen which will compute the path in a short time with an optimized path.

8.2. Satellite imagery

The best way for us to get a satellite image was through google. Google provides an API that is easy to use and adjust for software developers. The image is acquired with an HTTP request, providing the required specifications of the image and in the query parameters. And the result is returned to a .jpg formatted image that is ready to be used in planning the path.



8.3. Path planning

After receiving the required image with the needed specifications, this image is then discretized into a grid, and each grid node is labeled as a path or an obstacle. The size of the grid nodes mustn't be too big or too small. A big node will miss out

some of the details in the image. A small node means too many nodes will be considered, which results in intensive computations and longer computation time.

Each grid node in the image is mapped into obstacles and valid paths. The obstacles are listed in a list of (x,y) points in the grid. Now after acquiring the obstacles, start and destination, the algorithm can start running.

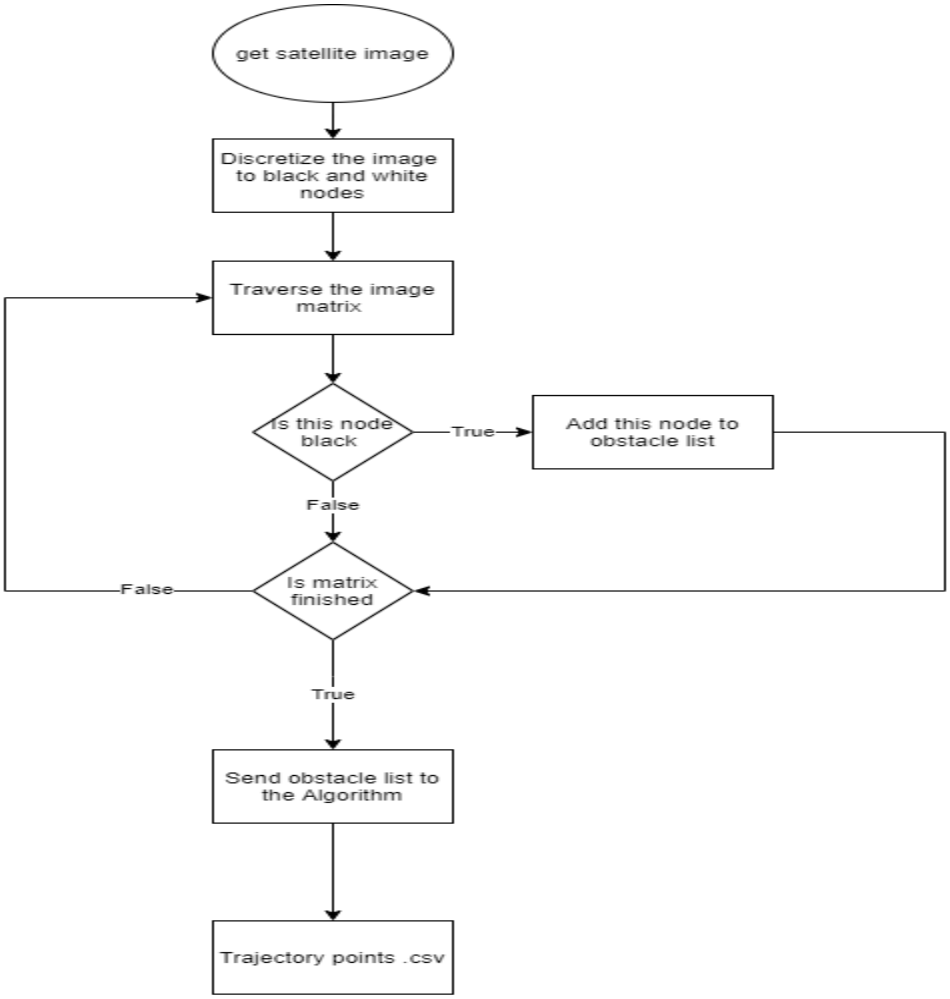


Fig. 2. Flowchart for analyzing data to get obstacles

8.4. RRT

RRT constructs a tree using random sampling in search space. The tree starts from an initial state say init z and expands to find a path towards goal state say goal z . The tree gradually expands as the iteration continues. During each iteration, a random state say rand z is selected from configuration space Z . If a random sample rand z lies in an obstacle free region, then a nearest node, say

nearest z is searched in a tree according to a defined metric ρ . If rand z is accessible to nearest z according to predefined step size, then tree is expanded by connecting rand z and nearest z . Otherwise, it returns a new node new z by using a steering function, thus expanding the tree by connecting new z with nearest z . A Boolean collision checking process is performed to ensure collision free connection between tree nodes new z and nearest z . When an initial path is found even then the process continues until a predefined time period expires or a fixed number of iterations are executed[9]. Node expansion process is described in Figure 1.

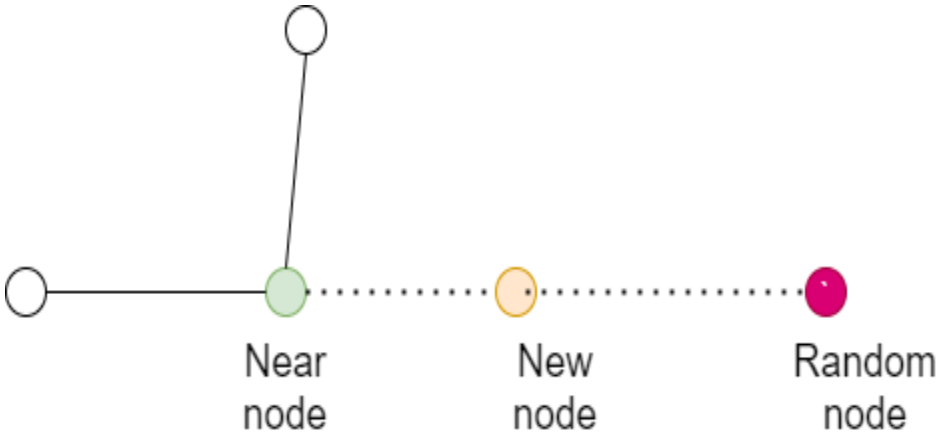


Figure 3:Tree expansion process.

RRT algorithm[x]

T ← InsertNode(Ø, z_{init}, T)

for i=0 to i=N do 4 z_{rand} ← Sample(i)

$Z_{nearest}$ ← Nearest(T, z_{rand})

(z_{new}, U_{new}) ← Steer ($Z_{nearest}$, z_{rand})

if Obstacle_free(z_{new}) then 8 T ← InsertNode(z_{min}, z_{new}, T)

return T

8.5. RRT*

RRT* inherits all the properties of RRT and works similar to RRT. However, it introduced two promising features called near neighbor search and rewiring tree operations. Near neighbor operations find the best parent node for the new node before its insertion in the tree. This process is performed within the area of a ball of radius defined by

$$k = \gamma \left(\frac{log(n)}{n} \right)^{\frac{1}{b}}$$

where d is the search space dimension and γ is the planning constant based on the environment. Rewiring operation rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections as shown in Figure 2 . Space exploration and improvement of path quality is shown in Figure 3. As the number of iterations increase, RRT* improves its path cost gradually due to its asymptotic quality, whereas RRT does not improve its jaggy and suboptimal path[9].

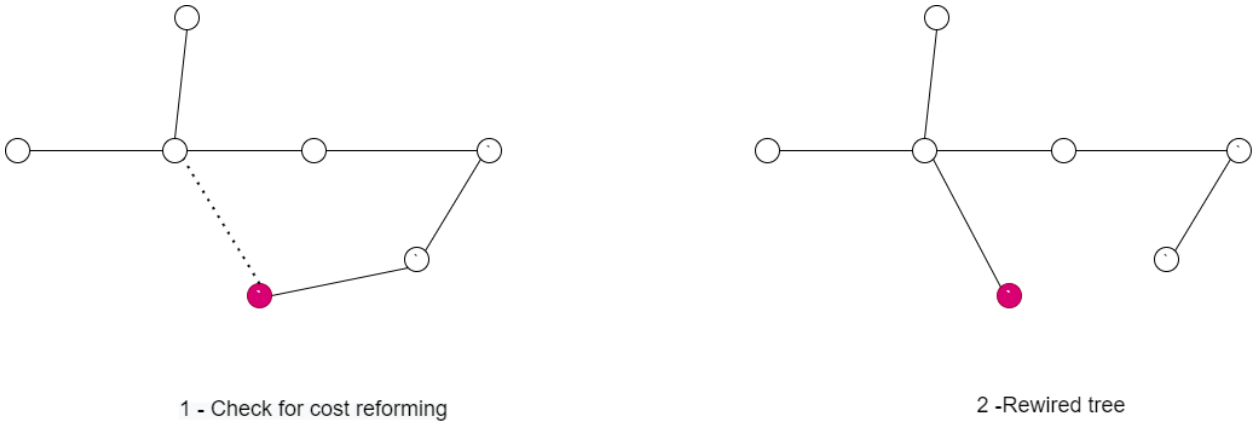


Figure 4: Near neighbour search and rewiring operations in RRT*

RRT* algorithm[x]

T= (V, E) ⟵ RRT*(z_{init}) 1 T ⟵ Initialize Tree()

T ⟵ InsertNode(Ø, z_{init}, T)

for i=0 to i=N do

z_{rand} ← Sample(i)

$Z_{nearest}$ ← Nearest(T, z_{rand})

(z_{new}, Unew) ← Steer $Z_{nearest}$, z_{rand})

if Obstacle_free(z_{new}) then

z_{near} ← Near(T, z_{new}, |V|)

z_{min} ← Chooseparent (z_{near}, $Z_{nearest}$, z_{new})

T ⟵ InsertNode(z_{min}, z_{new}, T)

Rewire (T, z_{near}, z_{min}, z_{new})

9. Deep Learning

9.1. Why deep learning

After guaranteeing fast arrival of the drone, the goal was to make the trip safe by avoiding any obstacles by working on a deep learning model that will help the drone to detect any nearby object and avoid it.

9.2. Main Idea

Tensor flow was used for object detection model running on the server to detect objects in the frames sent by the raspberry pi and after detecting the objects in the frame the server sends the detected objects and locations of these objects to the drone, upon receiving the feedback from the server the drone takes all the necessary actions to avoid any obstacle.

9.3. First Implementation

Firstly, a pre-trained tensor flow object detection model was downloaded which was SSD mobile net v1 and here is the specs of that model.

Model name	Speed (ms)	COCO mAP	Output
SSD MobileNet V1 FPN	48	29.1	Boxes

Specs of the used object detection model [10]

The above mentioned model is a pretrained model on the COCO 2017 dataset, and the categories in these datasets were interesting. Therefore, it was used in this project.

For the sake of testing a python script was written to capture frames from a recorded video or from laptop's webcam using python library cv2 and send these frames as an input to the model, then the model processes these frames and outputs a video with boxes around each detecting object in the frame indicating percentage of confidence, the result of the previous implementation was acceptable. The handled frames per second were acceptable as 20 FPS was reached and the model detected all the objects in the frames successfully but after testing the previous implementation on a higher quality videos as 720P or 1080P or even higher, it was noticeable that as the quality of the video increases the processing takes longer time and the FPS decreases. Therefore, some compression techniques on the video had to be made before sending it to the object detection model to process it to guarantee high speed processing of the model and live object detection to

guarantee safe arrival of the drone to the patient site,to assure that, the quality of the captured frames was decreased using CV2.

After testing the model on a laptop with high CPU and RAM, it was time to test the previous mentioned implementation on Raspberry pi 3. Therefore, a tensorflow model was downloaded and the previous mentioned python script was used to capture the frames coming from a pi cam or a video, unfortunately, there was a severe problem, which is the low processing time of the Raspberry pi compared to that of a laptop, therefore, the model runs too slowly and the FPS reached 1 even on a extremely inadequate video quality.

At first, it was thought that the problem is with the tensorflow model and what was needed is to work on tensorflow lite as it is lighter and does not need much processing compared to tensorflow. However, after testing tensorflow lite on raspberry pi 3 the performance did not get much better.

After upgrading the used Raspberry pi to Raspberry pi 4 as it has higher specs and it might help us with the pre mentioned problem, and here is a detailed comparison between Raspberry pi 3 and Raspberry pi 4.

	Raspberry pi 4	Raspberry pi 3 B+

CPU	Broadcom BCM2711 Quad core Cortex-A72 @ 1.5GHz	Broadcom BCM2837B0 Quad core Cortex-A53 @ 1.4GHz
GPU	VideoCore VI @ 500Mhz	VideoCore IV @ 250-400MHz
RAM	1GB, 2GB or 4GB LPDDR4-2400 SDRAM	1GB LPDDR2 SDRAM
USB	2x USB-a 2.0, 2x USB-A 3.0, 1x USB-C	4x USB-A 2.0 ports
Display ports	2x microHDMI	Single Full-size HDMI
Connectivity	802.11ac Wi-Fi, Gigabit Ethernet, Bluetooth 5.0	802.11ac Wi-Fi, 300Mbps Ethernet, Bluetooth 4.0
Misc	40-pin GPIO header, 3.5mm audio port, camera module support, composite video	40-pin GPIO header, 3.5mm audio port, camera module support, composite video

Comparison between Raspberry pi 4 and Raspberry pi 3[11]

Although the performance of the object detection model on Raspberry pi 4 improved compared to that on Raspberry pi 3 but this improvement didn't fill our need as the processes frames per second reached 4 FPS but this number is very far from live object detection as by this FPS the drone could hit an obstacle before noticing it.

Nvidia Jetson nano is a well known choice for deep learning models as it has a high performance GPU and it will definitely help us to achieve higher FPS and here is the specs of Nvidia Jetson nano.

GPU	128-core NVIDIA Maxwell™ architecture-based GPU
CPU	Quad-core ARM® A57
Video	4K @ 30 fps (H.264/H.265) / 4K @ 60 fps (H.264/H.265) encode and decode
Camera	MIPI CSI-2 DPHY lanes, 12x (Module) and 1x (Developer Kit)
Memory	4 GB 64-bit LPDDR4; 25.6 gigabytes/second
Connectivity	Gigabit Ethernet
OS Support	Linux for Tegra®

Module Size	70mm x 45mm
Developer Kit Size	100mm x 80mm

Nvidia Jetson nano specs[12]

The main problem with Jetson nano is its weight as it weighs 249.6 gm compared to Raspberry pi which weighs 43 gm only, given that the weight is very crucial in this case, an alternative had to be found.

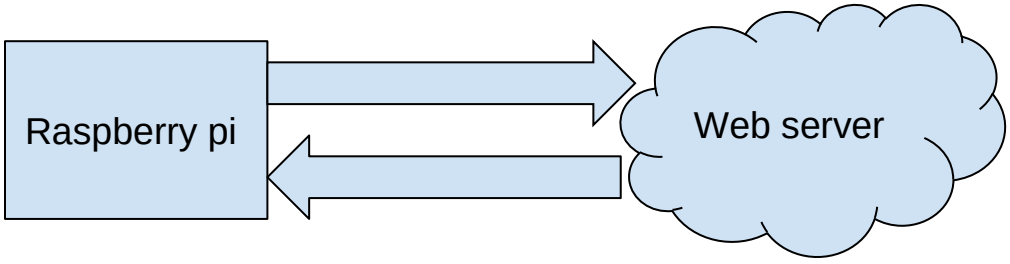
Following some research, it was found that the solution is to use a USB Coral accelerator. It acts as an external processor for the object detection model where the model would run faster and get the results in real time but due to COVID-19 acquiring such a device would require more time than the scope of the project.

9.4. Using web server

After trying to implement the object detection model locally, it is obvious that it's impossible to achieve live object detection on a raspberry pi locally without using USB coral accelerator, the alternative solution was using a web server where the raspberry pi will capture the frames using the pi cam and send them to the web server using 4G module, and the web server will process these frame and send the feedback to the drone.

The main advantage of using a web server is moving the whole processing unit off the drone, hence, no more constraint on the weight again and choosing a very powerful server without any load on the drone. Although using the web server is a very powerful solution, this approach introduced three main challenges.

The first one is building the backend that will accept the frames and send it to the tensorflow model. The second one is choosing the protocol that will connect the drone with the cloud to achieve live object detection and with minimum overhead. Finally, in terms of security one as using a server can be an immense vulnerability, as it is quite important to keep this connection safe from attackers, the upcoming subsections will show how it was managed to handle all these challenges.



9.5. Building backend on web server

After deciding on using the web server approach, comes the part of building a backend server where the raspberry pi will send the captured frames using the pi cam.

Firstly, building the backend using node js. Node js has an asynchronous background as it assigns a single thread to handle incoming requests. This thread is not blocked until the request is handled, for example if the Raspberry pi needs to read something from the database it sends a get request to the node js server. the server node assigns a single thread to handle the GET request and sends a request to read the required part from the database. This will take some time, so this thread is not blocked. But after sending the requests to read the required part from the database, it will handle other incoming requests, and when the database finishes its work, it puts a message in the event queue. Node js continuously monitors this queue in the background, that is, when it finds an event in that queue it takes it out and handles it. This kind of architecture makes node js ideally for applications which require intensive network access so it can serve more clients without the need of extra hardware.

In contrast, Node should not be used in cpu-intensive applications like video encoding or image manipulation service or object detection model, as these applications require high calculation, and since node is single threaded, this processing will block the thread and other clients have to wait. For that reason, using node js in this case will be inefficient.

Secondly, building the backend using flask, as it doesn't have the single thread nature as node, it's a lightweight framework, easy to use, has built in development server and debugger, it supports integrated unit testing, dispatching RESTful requests and extensively documented

Here is a comparison between Flask and Node Js showing the pros and cons of each framework.

	Node.js	Flask
Pros	● Npm ● Javascript ● Great libraries ● High-performance ● Open source ● Great for apis ● Asynchronous ● Non blocking IO ● Can be used as a proxy	● Lightweight ● Python ● Open source
Cons	● Bound to a single CPU ● Asynchronous programming is the worst ● Low computational power	● Not async-friendly

Comparison between Node.js and Flask[13]

9.6. Protocol used to send frames to the backend

For the protocol used to connect between the front-end which is the raspberry pi and the backend which is implemented using flask, a protocol that guarantees live object detection had to be used by connecting between the raspberry pi and web server using HTTP protocol, but after testing sending the frames to the server, using HTTP protocol allows sending a frame every 2 seconds which led to a disappointing performance for live object detection. 20 frames have to be sent and processed per second, but sending these frames using http takes much longer time as every time the client sends a frame to the server, a TCP connection is opened between the raspberry pi and the server. when the server sends the response back to the Raspberry PI, this tcp connection is terminated. If 20 frames per second were to be sent, it would require opening 20 tcp connection which is a large overhead. Thus, http protocol cannot be used to connect the raspberry pi to the drone and an alternative way is needed.

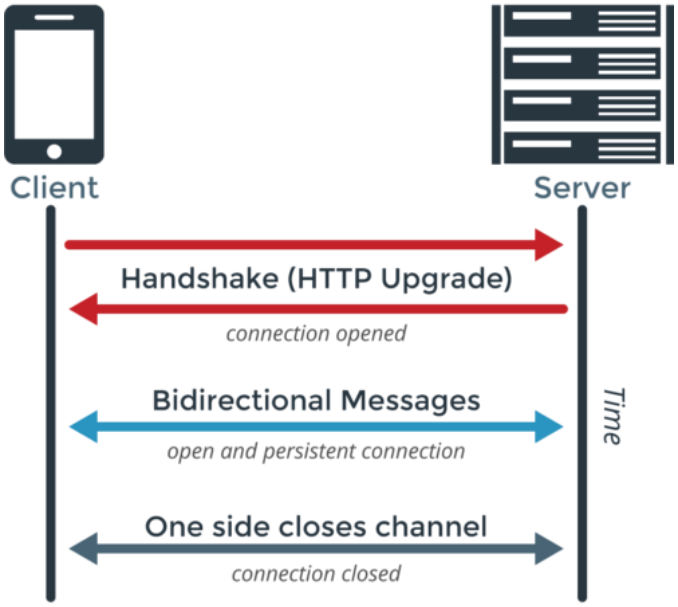
Websocket protocol will help achieving live object detection without much overhead as in the case of http, a websocket protocol is an excellent protocol in live streaming between client side and server side as whenever the client wants to send data to the server, a TCP connection is opened and this connection is not terminated when the server sends the

response back to the client but it continues to be opened until the client terminates this connection. Therefore, the overhead of opening a tcp connection with each frame as in the case of http protocol is not a problem anymore.

WebSocket solves few issues with HTTP:

- Bi-directional protocol — either client/server can send a message to the other party (In HTTP, the request is always initiated by the client and the response is processed by the server — making HTTP a uni-directional protocol)
- Full-duplex communication — client and server can talk to each other independently at the same time.
- Single TCP connection — After upgrading the HTTP connection in the beginning, client and server communicate over that same TCP connection throughout the lifecycle of WebSocket connection.

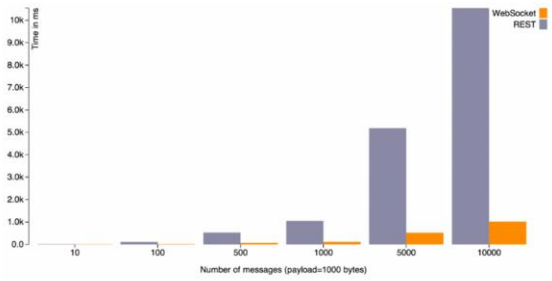
Here is a diagram showing how websocket protocol works:



Steps of websocket protocol [14]

If a performance test for REST and Websockets was run, Websockets might do better when high loads are presented. This does not necessarily mean that REST is inefficient. It's just that these two features solve two different problems and cannot be compared with a simple performance test like this:

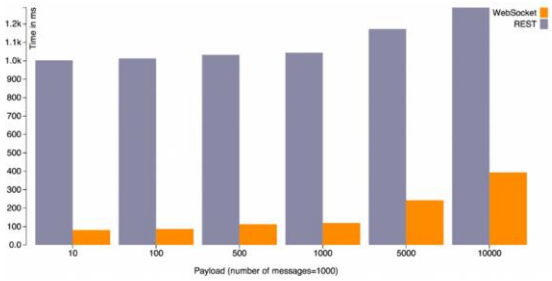
The first graph shows the time (in milliseconds) taken to process N messages for a constant payload size.



Here is the raw data that feeds this graph:

Constant payload, increasing number of messages			
Messages	REST (in ms)	WebSocket (in ms)	x times
10	17	13	1.31
100	112	20	5.60
500	529	68	7.78
1000	1050	115	9.13
5000	5183	522	9.93
10000	10547	1019	10.35

The second graph shows the time taken to process a fixed number of messages by varying the payload size.



Here is the raw data that feeds this graph:

Constant number of messages, increasing payload			
Payload (in bytes)	REST (in ms)	WebSocket (in ms)	x times
10	1003	81	12.38
100	1013	87	11.64
500	1032	113	9.13
1000	1044	119	8.77
5000	1173	243	4.83
10000	1289	394	3.27

Performance test between Websocket and REST [15]

By using websocket protocol to connect the Raspberry Pi to the server, a live object detection model was successfully implemented which reached 20 FPS. This number is improved after compressing the sent frames using cv2 python library where there is a parameter in the function which captures the frames from the pi cam this parameter was tuned in order to decrease the quality of each frame to speed up uploading the process and achieve a higher FPS.

9.7. Security of the server

After deciding to use a web server, handling the security is no more a luxury but it's a necessity, as without handling the security any one can connect to the server and send commands to the drone and control it, this will create a huge obstacle that will hinder the drone from reaching the patient.

Therefore JWT was used to make sure the connection is secure by carrying out the following steps:

1. The server sends a JWT to the drone.

2. Whenever the drone wants to open a connection with the server, the drone has to send the JWT with the request to the server.
3. On receiving the token the server verifies it with the secret key. if it’s not valid, the server will send a 403 response which indicates that the server understood the request but refuses to authorize it.
4. If the token is valid the server will open the connection with the drone and give it a permission to send frames to the server and receive the feedback from it.

9.8. JSON Web Token

JSON Web Token is an Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key. For example, a server could generate a token that has the claim "logged in as admin" and provide that to a client. The client could then use that token to prove that it is logged in as admin. The tokens can be signed by one party's private key *(usually the server's)* so that party can subsequently verify the token is legitimate. If the other party, by some suitable and trustworthy means, is in possession of the corresponding public key, they too are able to verify the token's legitimacy. The tokens are designed to be compact, URL-safe, and usable especially in a web-browser single-sign-on (SSO) context. JWT claims can typically be used to pass identity of authenticated users between an identity provider and a service provider, or any other type of claims as required by business processes.

JSON web token consists of three parts which are:

Header	{
	"alg" : "HS256",
	"typ" : "JWT"
	}
		Identifies which algorithm is used to generate the signature
		HS256 indicates that this token is signed using HMAC-SHA256.
		Typical cryptographic algorithms used are HMAC with SHA-256 (HS256) and RSA signature with SHA-256 (RS256). JWA (JSON Web Algorithms) RFC 7518 introduces many more for both authentication and encryption.

Payload	{
	"loggedInAs" : "admin",
	"iat" : 1422779638
	}
		Contains a set of claims. The JWT specification defines seven Registered Claim Names which are the standard fields commonly included in tokens. Custom claims are usually also included, depending on the purpose of the token.
		This example has the standard Issued At Time claim (`iat`) and a custom claim (`loggedInAs`).

Signature	HMAC-SHA256(secret, base64urlEncoding(header) + '.' + base64urlEncoding(payload))	Securely validates the token. The signature is calculated by encoding the header and payload using Base64url Encoding and concatenating the two together with a period separator. That string is then run through the cryptographic algorithm specified in the header, in this case HMAC-SHA256. The *Base64url Encoding* is similar to base64, but uses different non-alphanumeric characters and omits padding.

Structure of JSON Web Token [16]

9.9. Final implementation

After all the above mentioned trials, the final implementation is a drone containing a raspberry pi. This raspberry pi is connected to a web server where an object detection model is ready for processing any received frame through web socket protocol. whenever pi cam captures a frame, this frame is sent to the server where the object detection model is run to determine 3 things. Firstly, what objects are in the frame , the position of these objects in the frame and the score of each object. Finally, to speed up the process, the server will send feedback only for the objects with scores higher than 50%. Upon receiving the feedback from the server, the raspberry pi should notify the PID controller to take the appropriate action to avoid any nearby obstacle and complete a safe trip to the patient, this is how the drone drives autonomously and achieves live object detection.

9.10. Next step

After all what had been done, there is still plenty of room for improvement to make the trip even more safe, and achieve a faster and more accurate object detection model.

Firstly, To work on a deep learning model that will predict the distance between the object and the drone, as in the above-mentioned implementation, the drone can determine the objects in front of it, and use an ultrasonic sensor to determine the distance between the drone and any object. However,in order to make it even more reliable, the distance will be determined even more accurately by developing a new model to predict it.

After implementing the previous-mentioned model, we will work on an algorithm that will be executed whenever a drone receives feedback from the server. This algorithm will include the required procedures that the drone must take to avoid any nearby obstacles. Also, how the Raspberry pi will communicate with the PID controller to notify it with any nearby obstacles. Upon this notification, the PID will take the appropriate actions to avoid the obstacles and keep the drone safe.

10. Testing and User Interface

As for any software, it needs to be tested and verified especially if it is related to the automotive industry where lives can be at risk. Therefore, a simple user interface was designed to facilitate the testing process in a way that manually coding the different scenarios into the program itself is no longer needed and instead it takes the input from the user as to which scenario to try out and see the results on a terminal.

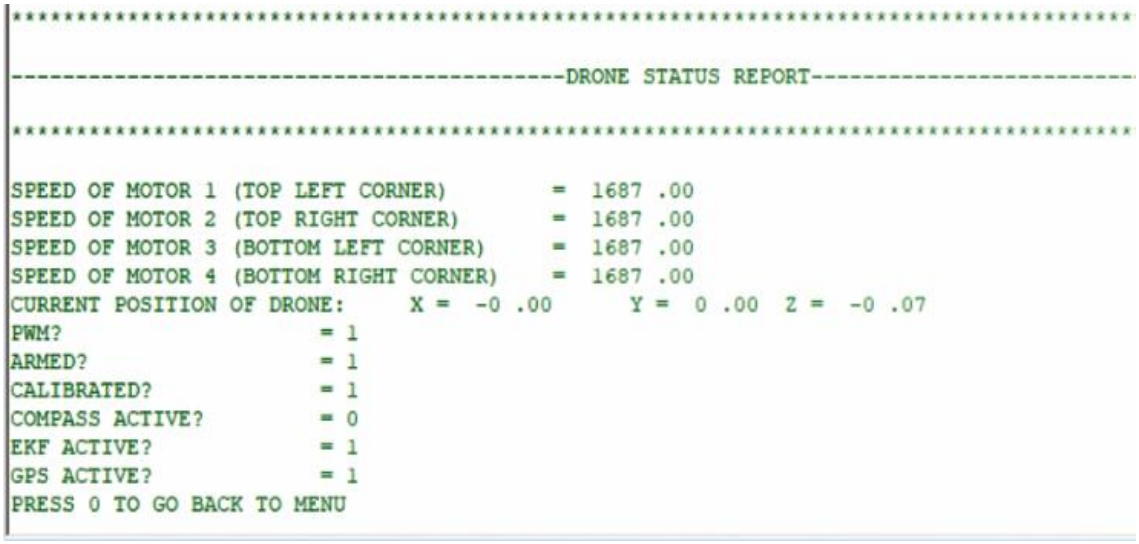
As explained before, there are 9 modes of operation. Most of these modes are for controlling different aspects of the drone such as the motors. However, modes 4 and 5 are solely made for testing purposes. They are similar except one runs the motors during the test and the other does not. The reason for that is simply as some tests do not require motors to be running per say.

```
COM15 115200 bps, 8N1, no handshake        Settings   Clear   About   Close
offsetx: 0 .00    offsety: 0 .00    offsetz: 0 .00
+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+

+*----------------------------------RAVEN5 DRONE USER INTERFACE------------------------

+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+

SELECT MODE OF OPERATION BY INSERTING MODE NUMBER:
MODE 0: CALIBRATION OF MOTORS
MODE 1: NORMAL START OF MOTORS
MODE 2: FREE CONTROL OF MOTORS
MODE 3: HOVER POWER OF MOTORS
MODE 4: FULL TEST OF SYSTEM WITH MOTORS
MODE 5: FULL TEST OF SYSTEM WITHOUT MOTORS
MODE 6: GET STATUS
MODE 7: PROGRAM START
MODE 8: SYSTEM RESET
```

Upon choosing a mode, the user is prompted to enter values of the target position, velocity and orientation. Every input is associated with an output that shows whether the system works or not. This enabled the testers to assess the drone's performance without manually coding different scenarios.

```
**************************************************************************************

----------------------------------------DRONE TESTING PLATFORM--------------------------

**************************************************************************************

TO INSERT PARAMETER VALUES INTO THE DRONE: FOLLOW THE GUIDE BELOW

FOR TARGET X Y Z: INSERT 1
FOR TARGET X Y Z VELOCITIES: INSERT 2
FOR TARGET PSI: INSERT 3
TO RETURN TO PREVIOUS MENU INSERT 0
INSERT TARGET X (-128 ~ 127), 'r' TO DISPLAY CURRENT MENU OR 'e' TO GO TO MAIN MENU
```

At some instances it is important to take a look at the system flags to see whether the different sensors are initialized correctly or not. This is useful when running multiple sensors and the readings are incorrect, it would be difficult to pinpoint the error if it was from a certain sensor.



Finally, during the testing process at the early stages, it was expected that the drone might be difficult to handle. Hence, an option was added to force the system to land and reset (Mode 8). Moreover, a long wire with a switch on its end and the drone on the other end was implemented to force the drone to shutdown with a non-maskable hardware interrupt in case of software failure.

11. Future Work

These are some ideas that can be added to the project for future work:

1. Develop a mobile application that requests the drone that can access the location of the user in order to make it easier to provide the coordinates without manually feeding it.
2. Make a network of drones and algorithms to decide which drone flies to which location across a whole country/city.
3. Develop a mechanism to handle the AED device delivering without hovering close to pedestrians.
4. Adding another camera to be able to distinguish distances between the drone and the surrounding objects.

12. Bibliography

12.1. Cardiovascular diseases WHO/Quinn Mattingly Credits Cardiovascular Diseases, (May 17, 2017). [Online].
Accessible: <https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1>

12.2. Admission delays' magnitude of traumatized patients in the emergency department of a hospital in Egypt: a cross-sectional study
Eur J Trauma Emerg Surg (April 2018). [Online]
Accessible: <https://pubmed.ncbi.nlm.nih.gov/28255612>

12.3. Ambulance Drone, TU Delft
by Alec Momont (October 24, 2014). [Online]
Accessible: <https://www.tudelft.nl/en/ide/research/research-labs/applied-labs/ambulance-drone/>

12.4. Drones join emergency care front line in Sweden with defibrillator drops
By Piers Ford (May 19, 2020). [Online].
Accessible: <https://www.mobihealthnews.com/>

12.5. Ambulance drones on the horizon
By Mohamed Derawi. [Online].
Accessible: <https://norwegianscitechnews.com/2018/05/ambulance-drones-horizon/>

12.6. Drone Calculator
Accessible: <https://www.ecalc.ch/>

12.7. Rotation matrix
By Wikipedia. [Online].
Accessible: <https://en.wikipedia.org/wiki/Rotation_matrix>

12.8. Estimation for Quadrotors
By udacity. Inc. [Online].
Accessible: <https://www.overleaf.com/read/vymfngphcccj>

12.9. 3D motion
By Iram Noreen , Amna Khan, Zulfiqar Habib (October 2016)
Accessible: <http://paper.ijcsns.org/07_book/201610/20161004.pdf>

12.10. Tensorflow object detection model
By Tensorflow [online]
Accessible:

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md>

12.11. Raspberry pi 4 vs Raspberry pi 3
By Jerry Hildenbrand (10 may 2020). [online]
Accessible: <https://www.androidcentral.com/raspberry-pi-4-vs-raspberry-pi-3>

12.12. Nvidia Jetson nano
By Nvidia [online]
Accessible: <https://nvidianews.nvidia.com/news/nvidia-announces-jetson-nano-99-tiny-yet-mighty-nvidia-cuda-x-ai-computer-that-runs-all-ai-models>

12.13. Node.js vs Flask
By Stackshare (11 Aug 2020). [online]
Accessible: <https://stackshare.io/stackups/flask-vs-nodejs#:~:text=Flask%20is%20intended%20for%20getting,fast%2C%20scalable%20network%20applications%22.>

12.14. Steps of websocket protocol
By Thilina Ashen Gamage (19 Nov 2017). [online]
Accessible: <https://medium.com/platform-engineer/web-api-design-35df8167460>

12.15. Performance test between Websocket and REST
By Thilina Ashen Gamage (19 Nov 2017). [online]
Accessible: <https://medium.com/platform-engineer/web-api-design-35df8167460>

12.16. Structure of Json Web Token
By Wikipedia (28 Dec 2010). [online]
Accessible: <https://en.wikipedia.org/wiki/JSON_Web_Token>

13. Appendix

This project was sponsored by Mentor Graphics and Dell.
Dell Competition Video:
Saving Cardiac Arrest Patients With a Medical Drone!