Ministry of Higher Education

Higher Technological Institute

10th of Ramadan City

6th of October Branch

Electrical and Computer Engineering Department

Embedded AI medical Quadcopter

Under Supervision: Dr. Kareem Badawi

: Dr. Mahmoud Hamed

Students Name:	**Hend Emad Saber**	**32018228**
	Nada El-sayed Hassan	**32018221**
	Govani Samy George	**32018073**
	Aliaa Magdy Abd Al-Ghany	**32018135**
	Hager Ahmed Kotb	**32018224**

January /May 2023

Acknowledgement

We would like to express our deepest gratitude and appreciation to all those who have contributed to the successful completion of our graduation project book. It has been an incredible journey, and we are overwhelmed with the support and guidance we have received along the way.

First and foremost, we extend our heartfelt thanks to Dr. Kareem Badawi and Dr. Mahmoud Hamed, whose invaluable expertise and mentorship played a pivotal role in shaping our project. Their dedication, encouragement, and unwavering belief in our abilities have been instrumental in our growth as aspiring professionals.

Furthermore, we would like to extend our appreciation to all the professors at our institute who have imparted their knowledge and expertise throughout our academic journey. Their teachings, mentorship, and dedication to our education have shaped us into the individuals we are today. We are truly grateful for their guidance, patience, and unwavering commitment to our development.

Abstract

Cardiac arrest is a life-threatening condition that requires immediate medical intervention. Unfortunately, the increasing number of deaths caused by cardiac arrest highlights the need for an efficient and timely response. In this project, we propose a novel approach to overcome the challenges associated with cardiac arrest by utilizing predictive analysis, wearable technology, drones, and advanced algorithms.

The project begins by employing predictive modelling techniques to identify individuals at high risk of experiencing cardiac arrest. By analyzing relevant medical data, we can accurately predict those who are more prone to this life-threatening condition. Subsequently, these high-risk individuals are provided with a specially designed watch equipped with a NodeMCU and heart rate sensor for simultaneous pulse monitoring.

If the patient's pulse stops or drops to a critical level, the watch immediately sends an alert to a central server. The monitoring server incorporates a Raspberry Pi (RPI) and continuously receives the alerts. Upon detecting a cardiac arrest event, a drone is deployed from the monitoring server to the patient's location, utilizing the informed RRT* algorithm to determine the shortest path.

Equipped with a camera, the drone scans and recognizes a QR code placed on the patient's chest, ensuring precise landing near the patient. Once landed, the drone autonomously separates the CPR (cardiopulmonary resuscitation) component and positions it on the patient's chest. Mechanical CPR is performed to maintain circulation until the watch detects the return of a detectable pulse rate.

By integrating predictive analysis, wearable technology, drones, and advanced algorithms, our project aims to provide an efficient and timely response system for cardiac arrest cases. This innovative approach has the potential to significantly reduce response times, improve the effectiveness of medical interventions, and ultimately save lives.

Contents

Table of figures

Chapter 1 Introduction

Chapter 1

Introduction

1.1 Introduction

We here propose to use drones in a specific case of healthcare which is Out of hospital cardiac arrest (OHCA). Our project aims to reduce losses of life that resulted from cardiac arrest. This is done by using the mechanical CPR technique by applying chest compressions. Delivering chest compressions is vital to the patient. With the heart not beating, oxygen-rich blood, especially that supplied by the rescue breaths, cannot be transported around the body. Performing chest compressions becomes a manual pump, substituting the heart. It is the heart pumping the blood around the patient's body.

When someone in a hospital has a sudden cardiac arrest, trained staff can immediately use a lifesaving Push up down CPR. It mechanically provides force on the heart in the hope of restoring a normal beating rhythm. If sudden cardiac arrest happens outside a hospital, emergency teams must battle traffic and distance to deliver defibrillators to those in need. Thus, our project aims to avoid traffic & distance issues, then reach the patient in the shortest time possible and perform CPR without any need for human intervention with consideration of cardiac-arrest occurrence when someone is alone.

Algorithms are utilized for mechanical CPR, autonomous take-off and landing, route planning, and monitoring duties in a certain area. Additionally, the patient will receive a watch with a pulse sensor that detects heart rate. In order to facilitate communication between the patient's pulse sensor, the drone system, and the hospital, this design is outfitted with sensor and communication integrators. If the patient was unaccompanied, the impact of a CPR drone would be considerable.

CPR drone will have a high effect if the patient is alone.

1.2 The objective of project

Our goal is to develop a drone that can simultaneously send an ambulance and deploy from its base to the patient's location, picking the best path and avoiding obstacles. In addition, each patient receives a T-shirt with a magnetic mechanical locking mechanism and a watch with a pulse sensor and location module. When the drone arrives at the patient's position and is centered on the patient's chest, it is hooked to the T-shirt magnets and performs mechanical cardiopulmonary resuscitation, this small standalone consistently and automatically administers deep, high-quality chest compressions, while waiting for the ambulance to arrive the location and apply the required procedures to the patient.

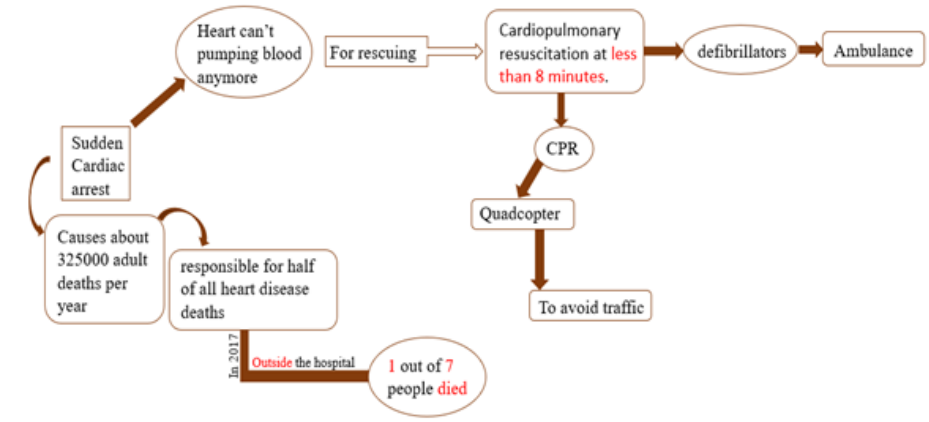


Figure 1- 1 project objective

1.3 Proposed design

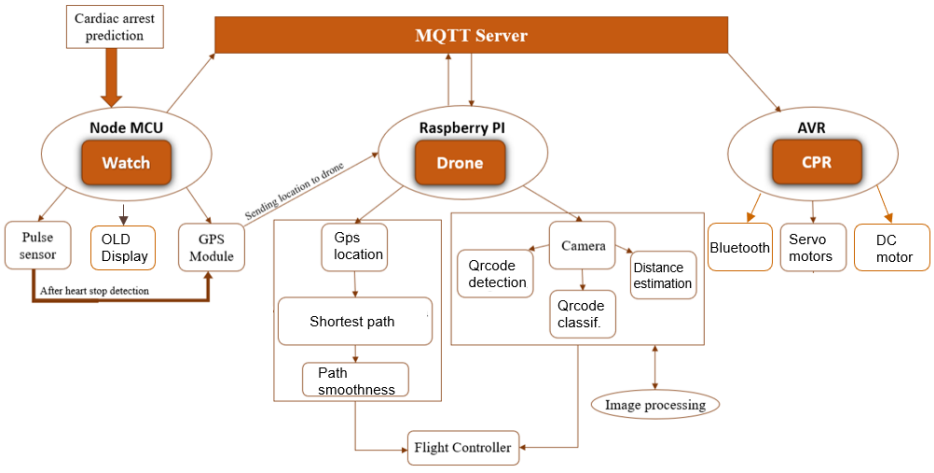


Figure 1- 2 proposed design

We will provide a "CPR medical drone system" to help resuscitate a cardiac arrest case as quickly as possible to minimize the damage to the patient's body and brain. Our system must achieve the following:

1) Choosing the shortest path to reach the patient's location and avoiding any obstacles in that path
2) Using raspberry pi will be responsible for motor motion directions depending on the predicted path by the motion planning algorithms, as well as connecting to the Pixhawk flight controller to obtain easy access to the drone's command library. As a result, the controlling algorithm can operate onboard and transmit commands to the drone via the DroneKit framework. Also, RPi will be linked to the server in order to communicate with the watch and CPR.
3) Communication between the drone system and the heart rate detecting watch at the base location is done via server.
4) Using a locking mechanism on a special t-shirt given to patients according to prediction based on data inputted by the user via a shared form to locate them and facilitate access ito them.
5) Finding the best position where the CPR is centered on the patient's chest is the first step in applying mechanical cardiopulmonary resuscitation.
6) Once the CPR reaches the right position, it starts its function, which is chest compression to mimic how the heart pumps. This compression helps keep the
7) blood flowing throughout the body.
8) The camera, sensors, and other hardware used to make the drone are cost-effective, so the drone would be affordable for the Health and Medical Department.

Chapter 2 Cardiac arrest prediction

Chapter 2

Cardiac arrest prediction

2.1 Problem definition

2.1.1 Target

To predict if someone has any possibility to have a sudden cardiac arrest.

2.1.2 Data

The data is collected from hospitals records. There are some factors that affects Death Event. This dataset contains person's information like age, sex, blood pressure, smoke, diabetes, ejection fraction, creatinine phosphokinase, serum_creatinine, serum_sodium, time and we have to predict if probability is 1 or 0.

2.1.3 Variables description

1. age: the age of the person with heart failure.

2. anemia: Decrease of red blood cells or hemoglobin (Boolean).

3. creatinine_phosphokinase: Level of the CPK enzyme in the blood (mcg/L).

4. diabetes: If the patient has diabetes (Boolean).

5. ejection_fraction: Percentage of blood leaving the heart at each contraction (percentage).

6. high_blood_pressure: If the patient has hypertension (Boolean).

7. platelets: Platelets in the blood (kiloplatelets/mL).

8. serum_creatinine: Level of serum creatinine in the blood (mg/dL).

9. serum_sodium: Level of serum sodium in the blood (mEq/L).

10. sex: Woman or man (binary).

11. smoking: If the patient smokes or not (Boolean).

12. time: Follow-up period (days).

13. is_under_risk: If the patient may have sudden cardiac arrest (Yes) or not (No).

2.1.4 Steps of the prediction

1. Data cleaning

2. Exploratory data analysis (EDA)

3. Feature engineering

4. Modelling

2.1.5 Modelling

1. Logistic regression

2. Support vector machine (SVM)

3. Decision tree

4. CatBoost

5. XGBoost

6. Random Forest

2.2 Data cleaning

This included convert variables to the suitable data type to avoid any errors in any calculations later, as well as outliers' detection and removal as it affects negatively on the models' accuracy.

2.3 Exploratory data analysis

2.3.1 Correlation matrix

It is a table that shows the correlation coefficients between the set of data features and target. In statistics, correlation refers to the degree to which two variables are related to each other. The correlation coefficient is a measure of the strength and direction of the relationship between two variables, ranging from -1 to +1. If the coefficient is about +1, this means that there is a strong positive relationship. If it is about -1, this means that the relationship is a strong negative one.

A correlation matrix typically displays the correlation between all pairs of variables in dataset. The variables are listed along both the rows and columns of the matrix, and the correlation coefficient for each pair of variables is displayed at the intersection of the corresponding row and column.

The diagonal of the correlation matrix always shows the correlation of a variable with itself, which is always equal to 1. The correlation matrix is symmetric, meaning that the correlation coefficients between variable A and variable B is the same as the correlation between variable B and variable A.

2.3.2 Univariate data analysis

Univariate data analysis is a statistical method that involves the analysis of a single variable at a time. It is a technique used to describe and summarize the characteristics of a single variable, such as its distribution, central tendency, variability, and outliers.

It is often used in exploratory data analysis to gain insights into the characteristics of a dataset. It is also used in hypothesis testing and statistical modeling to test the assumptions and conditions required for the chosen statistical tests.

It can be contrasted with multivariate analysis, which involves the analysis of two or more variables simultaneously. Multivariate analysis is often used when the relationships between variables need to be examined.

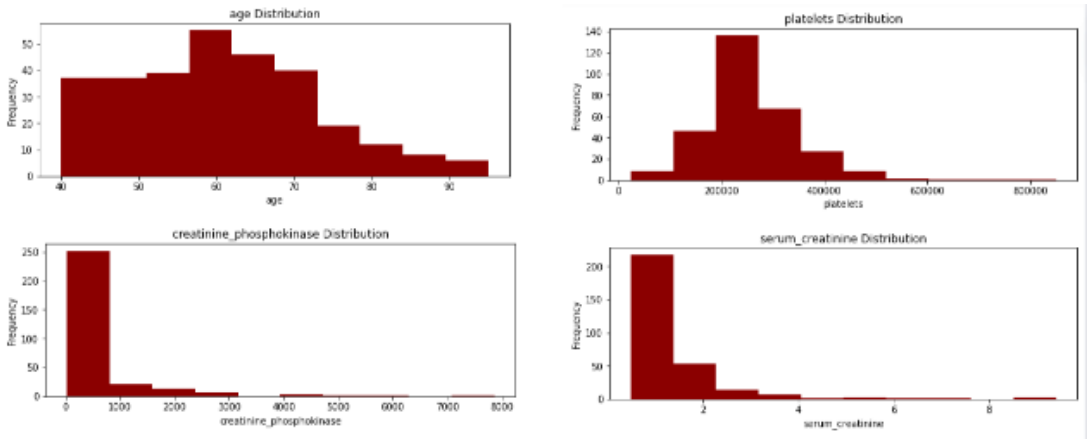


Figure 2- 1 numerical data analysis - 1

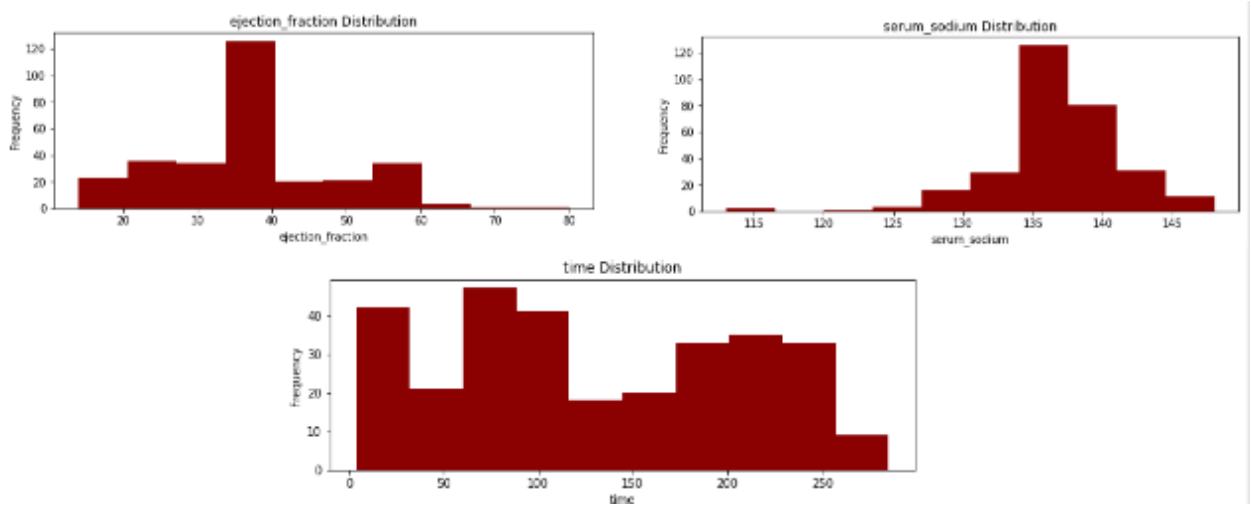


Figure 2- 2 numerical data analysis - 2

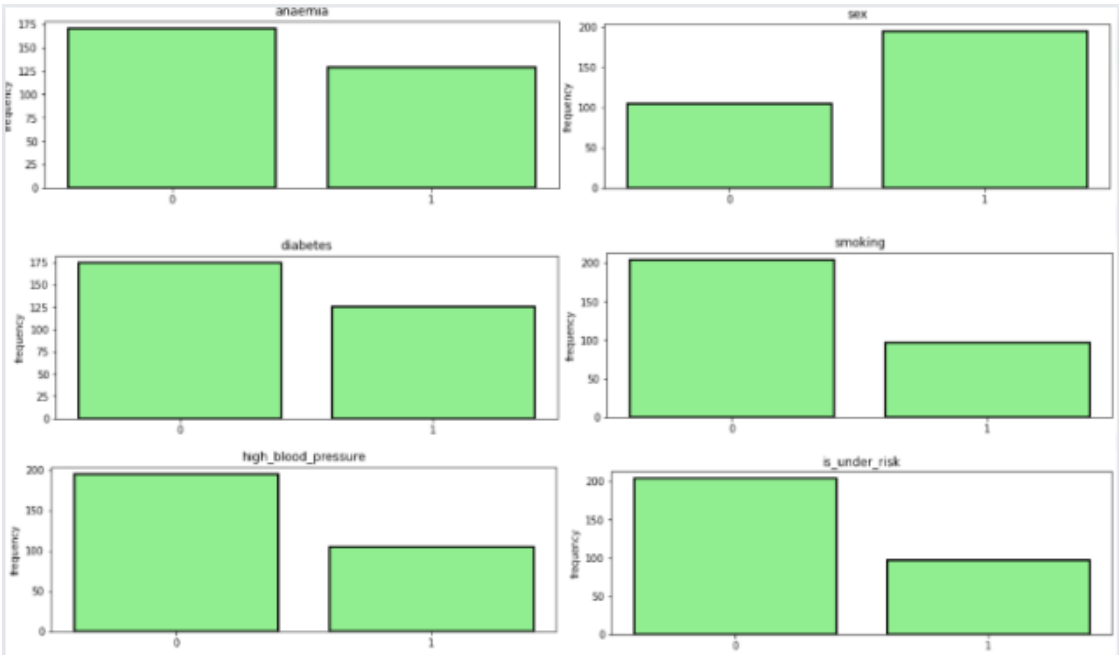


Figure 2- 3 categorical data analysis

From univariate analysis, there are unbalancing and skewness problems in the data set.

2.4 Feature engineering

2.4.1 Imbalanced data

Imbalanced data refers to a situation in which the number of observations in different classes or categories of a categorical variable is not equal. In other words, the frequency of one or more categories is much lower than the others.

It can lead to biased or inaccurate model performance, particularly when the model is trained using standard machine learning algorithms that assume balanced classes. In imbalanced data, the model may predict the majority class more frequently, resulting in poor performance on the minority class. The model may also have a high false-negative rate, failing to detect the minority class altogether.

Solution using SMOTE technique:

A data augmentation technique, referred as Synthetic minority oversample technique. Over sampling can be achieved by duplicating examples, but this will not add any information to the model.

As an improvement of only duplicating data, SMOTE technique adds data to the exist dataset by synthesis new examples from the minority class. It selects the close examples, draws a line between them, and draw then a new example at a point along that line.

SMOTE technique working logic:

A. Choose a random example from the minority class.

B. Take the k nearest neighbors for that chosen example (typically = 5).

C. Select random neighbored, and create a synthetic example between the two randomly selected points in the feature space.

2.4.2 Skewness

Skewness refers to a distortion or asymmetry that deviates from the symmetrical bell curve, or normal distribution, in a set of data. In simple terms, it is how much a variable deviate from the normal distribution.

There are two types: Right Skewed or Positive Skewed and Left Skewed or Negative Skewed.

- Right Skewed or Positive Skewed The distribution has a rightward tail with respect to the normal distribution.
- Left Skewed or Negative Skewed The distribution has a tail to the left relative to the normal distribution.

The model has difficulty in estimating the correct value at other points while focusing on the dense point while predicting on data that does not show a normal distribution. The output will result as a biased model.

What do we do, we will look at our skewness values. If it is greater than 1, there is positive skewness, if it is less than -1, there is negative skewness.

Box-Cox transformation:

Box-Cox transformation is a statistical technique used to transform a non-normal dependent variable into a normal shape. It is used to stabilize the variance of a variable and to improve the normality of the distribution.

involves applying a power transformation to the dependent variable. This power transformation is controlled by a parameter, lambda (λ), which determines the type of transformation applied. The Box-Cox transformation can be used for both positive and negative values of lambda.

The formula for the Box-Cox transformation is:

y(lambda) = (y ^ lambda - 1) / lambda if lambda != 0 & log(y) if lambda = 0

where y is the original dependent variable, and y(lambda) is the transformed variable. The parameter lambda controls the transformation. When lambda is equal to 1, the transformation is the identity transformation, and when lambda is equal to 0, the transformation is the natural logarithm.

The optimal value of lambda can be determined using maximum likelihood estimation or by minimizing the residual sum of squares. The Box-Cox transformation is commonly used in regression analysis to improve the normality of the distribution of the dependent variable and to stabilize the variance of the residuals.

2.5 Modelling

2.5.1 Logistic regression

Logistic regression is a statistical model used to analyze the relationship between a binary dependent variable (also called the outcome or response variable) and one or

more independent variables (also called predictors or explanatory variables). It is a type of regression analysis that is used when the dependent variable is dichotomous, meaning it can take one of two possible values, typically represented as 0 or 1.

It estimates the probability of the dependent variable taking the value 1 (or the positive outcome) given the values of the independent variables. The model uses a logistic or sigmoid function to convert a linear combination of the independent variables into a probability score that ranges from 0 to 1.

Mathematically, the logistic regression model can be represented as:
$P(y = 1 \mid x) = 1 / (1 + \exp(-z))$

where $p(y=1|x)$ is the predicted probability of the positive outcome given the values of the independent variables, x is a vector of the independent variables, and z is the linear combination of the independent variables and their coefficients.

It is estimated using a maximum likelihood method, which involves finding the values of the coefficients that maximize the likelihood of observing the data given the model.

It is often used in combination with other statistical techniques such as feature selection, regularization, and model evaluation to improve model performance and avoid overfitting.

2.5.2 Support vector machine (SVM)

It uses a boundary or hyperplane to separate the data into different classes. SVM can be used for both linear and nonlinear classification and regression tasks.

Mathematically, SVM works by finding the hyperplane that maximizes the margin or distance between the closest data points from different classes. The hyperplane is chosen in such a way that it separates the data into different classes with the maximum margin. The margin is defined as the distance between the hyperplane and the closest data points from different classes.

Let's assume we have a dataset with two classes, labeled as +1 and -1. Each data point is represented by a feature vector x and a label y, where y is either +1 or -1. The goal of

SVM is to find the hyperplane w*x + b = 0 that separates the two classes with the maximum margin. Here, w is the weight vector and b are the bias term.

The distance between a data point x and the hyperplane w*x + b = 0 is given by the formula: distance (x, w, b) = |w*x + b| / ||w||, where ||w|| is the Euclidean norm of the weight vector w. The sign of the value w*x + b determines which side of the hyperplane the data point x belongs to.

The margin of the hyperplane is given by the distance between the closest data points from different classes. Let's assume that the closest data points from each class lie on the hyperplanes wx + b = 1 and wx + b = -1, respectively. Then, the margin of the hyperplane is given by the formula: margin (w, b) = distance (x+, w, b) + distance (x-, w, b) = 2/||w||, where x+ and x- are the closest data points from the +1 and -1 class, respectively.

To find the hyperplane that maximizes the margin, we need to solve the following optimization problem: maximize margin (w, b) subject to y_i (w*x_i + b) >= 1 for all i, where y_i is the label of the i-th data point and x_i is its feature vector. This is a constrained optimization problem that can be solved using Lagrange multipliers. The solution involves finding the Lagrange multipliers alpha_i that satisfy the following conditions:

1) alpha_i >= 0 for all i
2) sum_i alpha_i*y_i = 0
3) w = sum_i alpha_iy_ix_i

The Lagrange multipliers alpha_i represent the importance or weight of each data point in the solution. Only the data points with non-zero alpha_i values are called support vectors, and they lie on the margin or on the hyperplanes w*x_i + b = +/- 1.

After finding the Lagrange multipliers alpha_i, the optimal hyperplane can be obtained by calculating the weight vector w and the bias term b using the equations:

- w = sum_i alpha_iy_ix_i
- b = (1/|S|) sum_i (y_i - wx_i), where S is the set of support vectors.

If the data is not linearly separable, we can use a kernel function to map the data into a higher-dimensional space where it is linearly separable. The most commonly used kernel functions are the linear kernel, the polynomial kernel, and the radial basis function (RBF) kernel.

2.5.3 Decision tree

A decision tree is a tree-like model that represents a sequence of decisions and their possible consequences. The algorithm works by recursively splitting the data into subsets based on the values of the input features until a stopping criterion is met or the tree is fully grown.

Mathematically, it can be represented as a set of if-then rules that partition the feature space into regions. Each internal node of the tree corresponds to a test of a feature, and each leaf node corresponds to a class label or a regression value.

Let's assume we have a dataset with n data points and m features. The goal of decision tree is to find a set of if-then rules that minimize the impurity of the resulting regions. Impurity is a measure of the homogeneity or purity of the class labels or regression values within a region. The most commonly used impurity measures are Gini impurity, entropy, and classification error.

The impurity of a region R is given by the formula: impurity(R) = sum_k p_k*(1 - p_k), where p_k is the fraction of data points in R that belong to class k.

The total impurity of a split of the data based on a feature j and a threshold s is given by the formula: impurity (j, s) = (n_l/n) *impurity(R_l) + (n_r/n) *impurity(R_r), where R_l and R_r are the two resulting regions of the split, n_l and n_r are the number of data points in each region, and n is the total number of data points. The split is chosen in such a way that it maximizes the information gain or the reduction in impurity compared to the parent node.

The information gain of a split based on a feature j and a threshold s is given by the formula: information_gain (j, s) = impurity (parent) – impurity (j, s), where impurity(parent) is the impurity of the parent node.

It works by recursively splitting the data based on the features that provide the highest information gain. At each node of the tree, the algorithm selects the feature and the threshold that maximize the information gain and creates two child nodes representing the resulting regions. The process is repeated until a stopping criterion is met or the tree is fully grown.

The stopping criterion can be based on several factors, such as the maximum depth of the tree, the minimum number of data points in a leaf node, or the minimum information gain required for a split. The tree is pruned after it is fully grown to reduce overfitting and improve its generalization performance.

2.5.4 CatBoost

It is a gradient boosting algorithm that is designed to handle categorical features and provide high accuracy and efficiency in machine learning tasks. CatBoost is based on the gradient boosting framework and uses a novel algorithm for handling categorical features, a custom loss function that can handle missing values, and a method for optimizing the ordering of categorical variables.

Mathematically, CatBoost works by building an ensemble of decision trees using gradient boosting. The objective of CatBoost is to minimize a loss function that measures the difference between the predicted values and the actual values of the target variable.

Let's assume we have a dataset with n data points and m features, including categorical features. The goal of CatBoost is to build a set of decision trees that can predict the target variable y based on the input features x.

The prediction of the i-th data point by the CatBoost model is given by the formula: y_i = sum_j f_j(x_i), where f_j is the j-th decision tree, and x_i is the feature vector of the i-th data point.

It works by building a set of decision trees iteratively. At each iteration, the model fits a new decision tree to the negative gradient of the loss function with respect to the predicted values. The new tree is added to the ensemble, and the model updates the predicted values by adding the predictions of the new tree to the previous predictions. The process is repeated until a stopping criterion is met or the maximum number of trees

is reached. It uses a novel algorithm called Ordered Boosting to handle categorical features. This algorithm assigns a numerical value to each category based on its frequency and its relationship with the target variable. The numerical values are used to split the data and build the decision trees. The Ordered Boosting algorithm can improve the accuracy of the model by reducing the bias introduced by the one-hot encoding of categorical features.

It provides a custom loss function called Logarithmic Loss that can handle missing values in the target variable. The Logarithmic Loss function penalizes the model for incorrect predictions and adjusts the weights of the data points based on the presence or absence of the target value. This approach can improve the accuracy of the model by taking into account the missing values in the target variable. As well as, a method called Best-Gain Split to optimize the ordering of categorical variables. This method selects the best split point for each feature based on its impact on the loss function and the number of data points in each split. This approach can improve the accuracy of the model by reducing the variance introduced by the arbitrary ordering of categorical variables.

2.5.5 XGBoost

It stands for Extreme Gradient Boosting, which is a popular gradient boosting algorithm used for classification and regression analysis. XGBoost is based on the gradient boosting framework and uses a novel algorithm for handling missing values, a method for regularizing the tree structure, and a technique for parallelizing the computations.

Mathematically, it works by building an ensemble of decision trees using gradient boosting. The objective of XGBoost is to minimize a loss function that measures the difference between the predicted values and the actual values of the target variable.

Let's assume we have a dataset with n data points and m features. The goal of XGBoost is to build a set of decision trees that can predict the target variable y based on the input features x.

The prediction of the i-th data point by the XGBoost model is given by the formula: y_i = sum_j f_j(x_i), where f_j is the j-th decision tree, and x_i is the feature vector of the i-th data point.

The algorithm works by building a set of decision trees iteratively. At each iteration, the model fits a new decision tree to the negative gradient of the loss function with respect to the predicted values. The new tree is added to the ensemble, and the model updates the predicted values by adding the predictions of the new tree to the previous predictions. The process is repeated until a stopping criterion is met or the maximum number of trees is reached.

This algorithm can handle missing values by assigning them to the left or right child of a split based on the direction that gives the maximum reduction in the loss function. It also provides a method for regularizing the tree structure called Tree Pruning. This method prunes the branches of the decision trees that do not contribute significantly to the reduction of the loss function. This approach can improve the accuracy of the model by reducing the variance introduced by the overfitting of the decision trees, as well as a technique for parallelizing the computations called Distributed and Out-of-Core Computing. This technique allows XGBoost to process large datasets that do not fit into memory by splitting the data and the computations across multiple nodes or disks.

2.5.6 Random Forest

It is an ensemble learning method that combines multiple decision trees and uses to improve the accuracy and generalization performance of the model.

Mathematically, it works by building an ensemble of decision trees using random sampling and random feature selection. The objective of Random Forest is to minimize a loss function that measures the difference between the predicted values and the actual values of the target variable.

Let's assume we have a dataset with n data points and m features. The goal of Random Forest is to build an ensemble of decision trees that can predict the target variable y based on the input features x.

The prediction of the i-th data point by the Random Forest model is given by the formula: y_i = 1/T*sum_j f_j(x_i), where f_j is the j-th decision tree, T is the total number of trees, and x_i is the feature vector of the i-th data point.

It works by building multiple decision trees using random sampling and random feature selection. At each iteration, the model selects a random subset of the data and a random subset of the features to build a decision tree. The process is repeated until a stopping criterion is met or the maximum number of trees is reached.

Furthermore, it uses a method called Bagging to improve the accuracy and stability of the model. Bagging involves randomly selecting subsets of the data with replacement and training a decision tree on each subset. This approach can reduce the variance introduced by the overfitting of the decision trees and improve the generalization performance of the model, as well as a method called Feature Randomness to improve the robustness and diversity of the model. Feature Randomness involves randomly selecting a subset of the features at each split of the decision tree. This approach can reduce the correlation between the decision trees and improve the accuracy of the model by capturing different aspects of the data.

The prediction of the Random Forest model is obtained by averaging the predictions of the individual decision trees. This ensemble approach can improve the accuracy and stability of the model by reducing the impact of the individual errors or biases of the decision trees.

Chapter 3 Heart rate monitor Watch

Chapter 3

Heart rate monitor Watch

3.1 Introduction

A watch is a wearable technology that is designed to monitor the wearer's heart rate in real-time which can be an early warning sign of cardiac arrest. When a cardiac arrest patient experiences an abrupt loss of heart function, the heart rate monitor watch can detect the absence of pulsatile blood flow then sends the patient's location to a server that can be a valuable tool for helping the drone locate the patient quickly. once the drone locates the patient, it can arrive the patient’s location and perform immediate cardiopulmonary resuscitation (CPR).

3.2 Hardware components

To build a watch, the following hardware components are needed:

- NodeMCU microcontroller: This microcontroller is used to control the sensors and display the heart rate and GPS data.
- Heart Rate sensor (MAX30102 Heart Rate and Pulse Oximeter Sensor): This sensor is used to measure the activity of the heart and detect abnormal heart rhythms.
- GPS module (u-blox NEO-M8N): This module is used to track the location of the wearer.
- OLED display: This display is used to show the heart rate and GPS information.
- Battery: A rechargeable battery is needed to power the device.

3.3 Software components

The software components include:

- Adafruit IO platform: This platform is used to monitor and analyze the heart rate and GPS data from the watch.
- Arduino IDE: This integrated development environment is used to program the NodeMCU microcontroller using C++ programming language.

3.4 Adafruit IO

3.4.1 Introduction

Adafruit.io is cloud service needed to display the IOT project's data online in real-time. It is a cloud server that can be used to connect to IoT devices through wifi and to control these devices through a dashboard. It can be used as a free service and it has got a simple easy-to-use interface to design dashboards. The key for authentication can be generated directly from the user's account and can be included in the program to connect the IOT components to the respective dashboard. The feeds are important to make connections, using the program that connects the IOT circuit to the Adafruit dashboards. The feeds are holders for the data and related information that is transferred to and fro from the dashboard and IOT circuit.

3.4.2 Connecting a watch to Adafruit.IO

To connect a watch to Adafruit this involve establishing a connection between the NODEMCU and the Adafruit platform. Once the connection is established, we can monitor patient's heart rate. Connecting a NodeMCU with Adafruit IO involves setting up a wireless connection between the NodeMCU and Adafruit IO, and then using MQTT as a lightweight messaging protocol to transmit data between the two devices. MQTT uses a publish-subscribe model, where each client can publish messages to a central broker, and other clients can subscribe to receive messages from the broker. This allows multiple devices to communicate with each other through a single broker, without the need for direct communication between the devices. Using MQTT for wireless connectivity can offer several benefits for connecting a heart rate monitor watch to Adafruit. MQTT is a lightweight protocol that is designed for use in low-bandwidth, high-latency networks, making it well-suited for IoT applications.

3.4.3 Working Principle

The heart rate sensor is placed on the user's wrist and is used to measure the user's heart rate. The sensor sends data to the NodeMCU board, which processes the data and calculates the heart rate. The NodeMCU board is also connected to a GPS module, which provides location data that can be used to track the user's location in case of an emergency. The OLED display is used to display the user's heart rate, time, Battery

charge percentage, displays a warning if the watch is unable to measure the pulse and other relevant information.

The NodeMCU board is connected to the Adafruit.io cloud platform. In the case of the heart rate sensor detect stopping patient's heartbeat, the NodeMCU publishes 1 and the patient's location to the Adafruit.io. in the otherwise it publishes 0 and patient's heart rate.

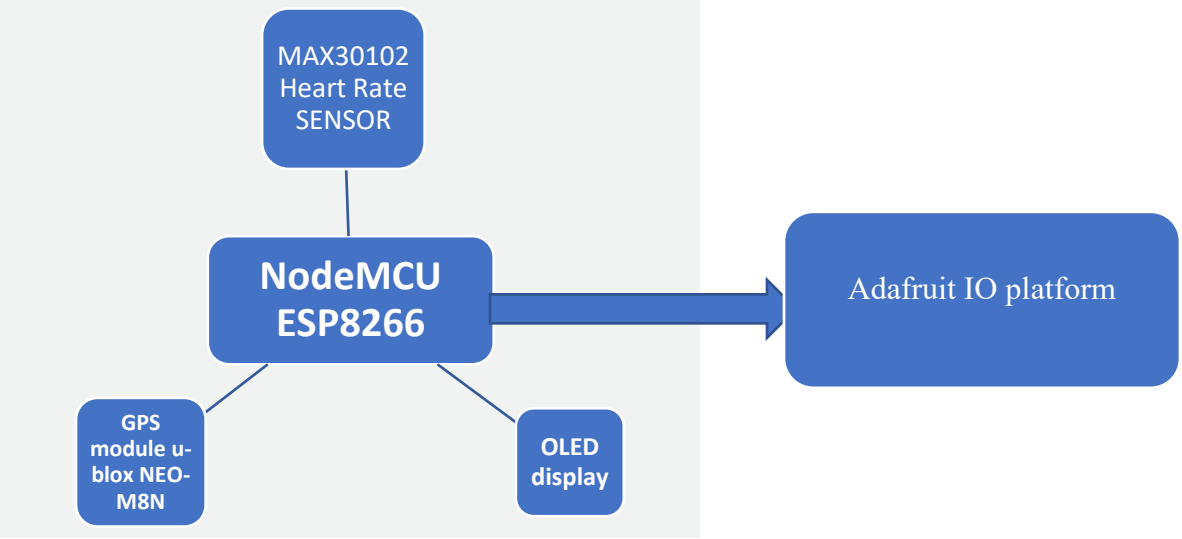


Figure 3- 1heart rate monitor watch diagram

3.5 NodeMCU ESP8266

3.5.1 Introduction

The NodeMCU is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266 contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for Internet of Things (IoT) projects of all kinds.

However, as a chip, the ESP8266 is also hard to access and use. You must solder wires, with the appropriate analog voltage, to its pins for the simplest tasks such as powering it on or sending a keystroke to the "computer" on the chip. You also have to program it in low-level machine instructions that can be interpreted by the chip hardware. This level of integration is not a problem using the ESP8266 as an embedded controller chip in mass-

produced electronics. It is a huge burden for hobbyists, hackers, or students who want to experiment with it in their own IoT projects.

3.5.2 ESP8266 Peripherals

The ESP8266 peripherals include:

- 17 GPIOs
- SPI
- I2C (implemented on software)
- I2S interfaces with DMA
- UART
- 10-bit ADC

3.6 NodeMCU

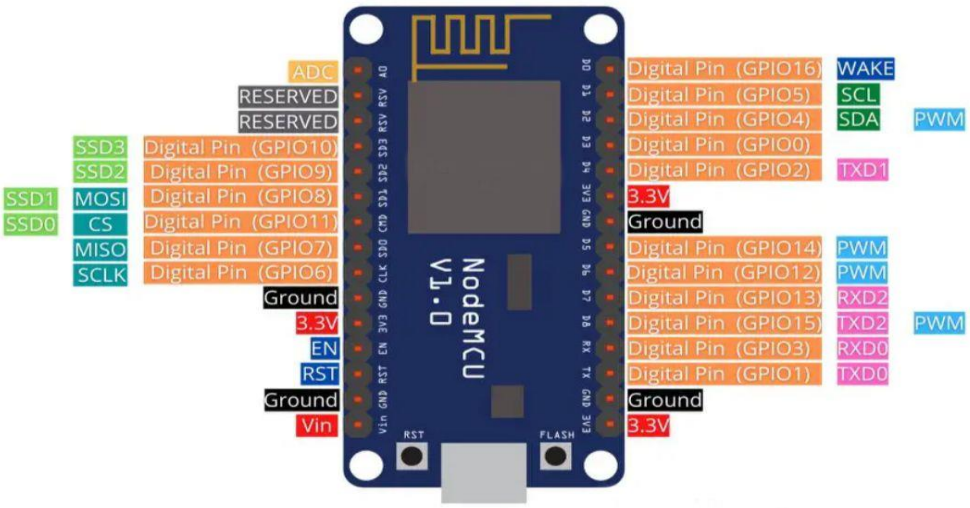


Figure 3- 2 nodemcu pinout

- Power Pins: Vin which used to power the esp8266 board. The board has three 3.3V pins and There are GND a total of 4 pins.
- ADC: there is an inbuilt 10-bit analog to digital converter with only one ADC.
- UART pins: there are 2 UART interfaces– RX0 and TX0, RX2 and TX2
- SPI Pins: four pins available for SPI (one set of SPI).

- I2C Pins: may use any GPIO pins as I2C as long as you are aware of the I2C programming. Only two pins are required for I2C: SDA and SCLK. Usually, the following GPIOs are used as I2C pins (GPIO 4, GPIO 5)
- PWM Pins: there are 4 PWM-enabled pins which can be used for driving digital motors and LEDs.
- EN or Enable Pin: EN which enable or disable the board
- RST or Reset Pin: which resets the board when pulled low. It works like the onboard reset button.
- Wake Pin(D0): This pin is used to wake up the esp8266 chip from a deep sleep.

3.7 MAX30102 Heart Rate and Pulse Oximeter Sensor

3.7.1 Introduction

The MAX30102 pulse oximeter and heart rate sensor is an I2C-based low-power plug-and-play biometric sensor; used as both a heart rate monitor and a pulse oximeter. These features are enabled by constructing this sensor which consists of two LEDs, a photodetector, optimized optics, and low noise signal processing components.

The MAX30102 sensor module consists of two LDO regulators. This is because the MAX30100 IC requires 1.8V and the LEDs require 3.3V to function properly. With the addition of the voltage regulators, we can safely use microcontrollers that use 5/3.3/1.8V level input/outputs. In addition to solder jumper feature to select the voltage logic level. By default, it is set to 3.3V but you can also change it to 1.8V according to your microcontroller's logic requirements. The module uses a simple two-wire I2C interface for communication with the microcontroller. It has a fixed I2C address: 0xAEHEX (for write operation) and 0xAFHEX (for read operation).

The MAX30102 sensor provides accurate measurements and operates at a low power consumption, making it ideal for battery-powered devices. therefore, a great choice to use in wearable devices such as heart rate monitor watches.

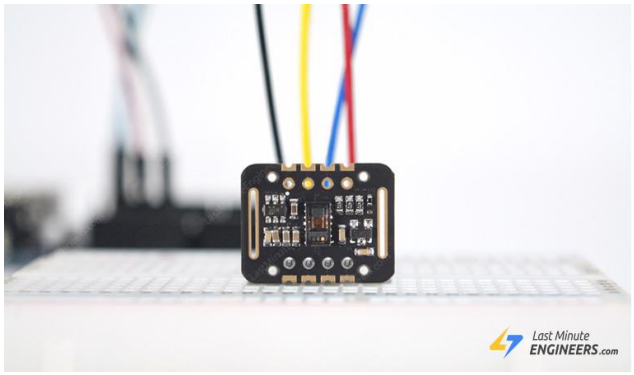


Figure 3- 3 max30102 sensor

3.7.2 Max30102 module working

The MAX30102 sensor is a pulse oximeter and heart rate sensor module designed to provide accurate measurements of oxygen saturation and heart rate.it designed to measure changes in blood volume using a technique called photoplethysmography (PPG) which relies on the pulsatile nature of blood flow caused by the heartbeat. It is based on the MAX30102 sensor chip, which integrates a red LED, an infrared LED, and a photodetector to measure the absorption of light by blood vessels. The red and infrared LEDs emit light into the skin, and the photodetector measures the amount of light that is absorbed by the blood vessels. As the heart beats and blood flows through the vessels, the amount of light absorbed changes, and these changes can be used to determine the heart rate and blood oxygen saturation levels. if there is no blood flow, there will be no oxygenated or deoxygenated hemoglobin to measure. Which means the heartbeat has stopped.

3.7.3 Interfacing MAX30100 Pulse Oximeter sensor with NodeMCU ESP8266

Connect the default I2C pins of ESP8266 (D1, D2) with the SCL and SDA pins of the module. Additionally, the sensor is powered by 3.3V from the ESP8266 and both grounds are in common.

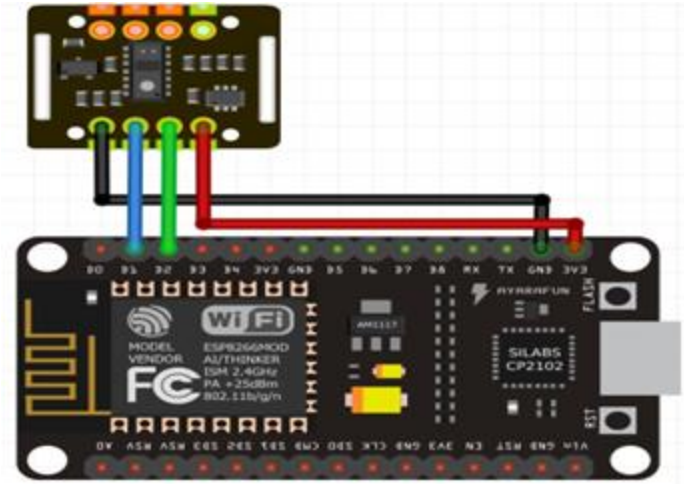


Figure 3- 4 connecting MAX30100 sensor with NodeMCU ESP8266 diagram

Configuring the Sensor

The MAX30102 sensor is a highly configurable sensor that provides several options for adjusting the performance and behavior of the sensor. it helps to optimize the accuracy of the heart rate measurements while balancing the trade-offs between power consumption, processing time, and other factors. To configure the MAX30102 sensor, the sensor's settings can be adjusted using the MAX30102 library, which provides functions for configuring the sensor and reading the data. The configuration settings can be adjusted by calling the appropriate functions provided by the library.

The configuration parameters include:

- ledBrightness
- sampleAverage
- ledMode
- sampleRate
- pulseWidth
- adcRange.

These settings may need to be adjusted depending on the specific application and the characteristics of the user's skin and blood vessels.

3.8 GPS module (Ublox NEO-M8N)

3.8.1 Introduction

The NEO-M8N GPS module is a high-performance GPS receiver module that provides accurate positioning with a horizontal accuracy of up to 2.5 meters and timing information. It uses the Global Navigation Satellite System (GNSS) to receive signals from multiple satellites and provide accurate location and timing information. The NEO-M8N GPS module supports various GNSS systems, including GPS, GLONASS, Galileo, and BeiDou. This module has a rechargeable battery and supports UART communication protocol with active antenna. NEO-8M can receive information and then calculate the geographical position with fast speed and has an internal memory to save settings.

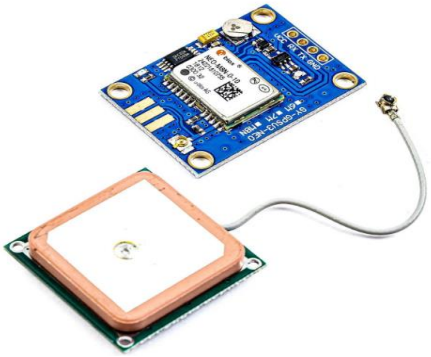


Figure 3- 5 Ublox NEO-M8N GPS

3.8.2 GPS module working

The global positioning system receives signals from satellites. If the signals of at least four satellites are received the position of the device can be calculated. With four satellites, the receiver can determine its three-dimensional position (latitude, longitude, and altitude) and its clock offset. It can do this by comparing small differences in the time the signal took to reach the receiver. Since the satellites broadcast these signals with relatively weak signal the receiver requires direct line-of-sight to the satellites and this means pure GPS often doesn't work well indoors.

3.8.3 Interfacing NEO-M8N with NodeMCU

The module is powered using a 3.3V or 5V power supply, depending on the module's specifications. The module's data can be read using a serial communication protocol, such as UART or SPI. In our project we used UART communication protocol. By default, it is set to 9600 baud but this value can be changed also through the serial interface.

This sensor has 4 pins:

- VIN: Module power supply
- GND: Ground
- RX: Receive data via serial protocol
- TX: Sending data via serial protocol

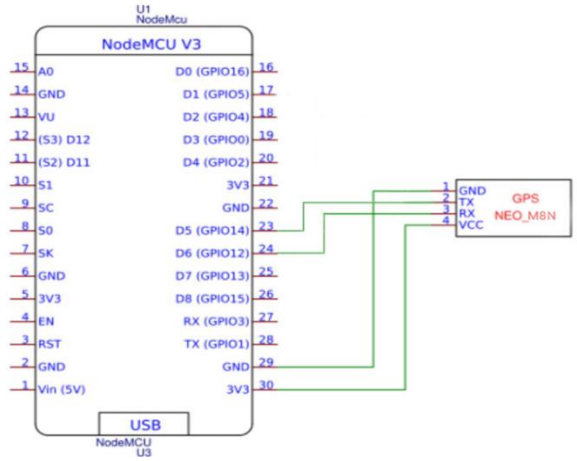


Figure 3- 6 connecting NEO-M8N with NodeMCU diagram

Connect the GPS module directly to NodeMCU board by connecting GND pin of neo m8n to GND pin of NodeMCU and VCC pin to 3v3 pin. Also connect RXD to D5 and TXD to D6 (GPIO 14, GPIO12).

3.8.4 Received data by GPS

GPS module sends the Real time tracking position data in NMEA format. NMEA format consist several sentences, in which four important sentences are given below.

- $GPGGA: Global Positioning System Fix Data
- $GPGSV: GPS satellites in view
- $GPGSA: GPS DOP and active satellites
- $GPRMC: Recommended minimum specific GPS/Transit data

We can get to the core of the GPS information: latitude, longitude, altitude, speed,

By installing The TinyGPS+ library. TinyGPS library has inbuilt function to get the required data from the NMEA string.

3.9 OLED Display Module

3.9.1 Introduction

OLED (Organic Light-Emitting Diode) displays are available in a range of sizes (such as 128×64, 128×32) and colors (such as white, blue, and dual-color OLEDs). Some OLED displays have an I2C interface, while others have an SPI interface. In our project we used 128×64 I2C OLED display.

At the heart of the module is a powerful single-chip CMOS OLED driver controller – SSD1306, which handles all RAM buffering, requiring very little work from your ESP8266. Also, the SSD1306 controller's operating voltage ranges from 1.65V to 3.3V, making it ideal for interfacing with 3.3V microcontrollers such as the ESP8266.

An OLED display, unlike a character LCD display, does not require a backlight because it generates its own light. This explains the display's high contrast, extremely wide viewing angle, and ability to display deep black levels.

3.9.2 control the OLED display

In order to control the display, it is crucial to understand the memory map of the OLED display. Regardless of the size of the OLED display, the SSD1306 driver includes a 1KB Graphic Display Data RAM (GDDRAM) that stores the bit pattern to be displayed on the screen. This 1 KB memory area is divided into 8 pages (from 0 to 7). Each page has 128 columns/segments (block 0 to 127). And, each column can store 8 bits of data (from 0 to 7). That certainly proves that we have:

8 pages x 128 segments x 8 bits of data = 8192 bits = 1024 bytes = 1KB memory

Each bit represents a single OLED pixel on the screen that can be turned ON or OFF programmatically.

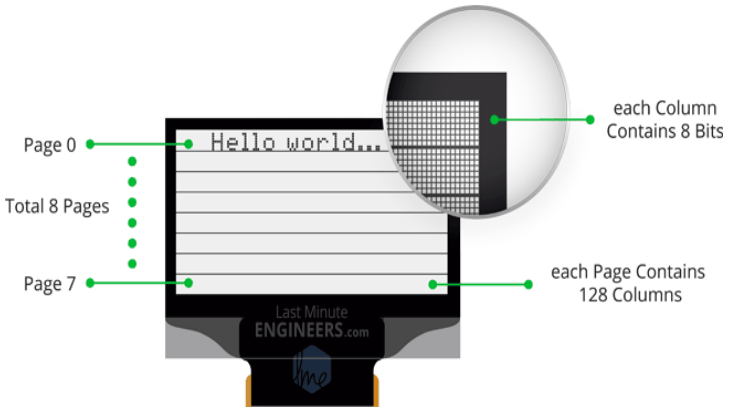


Figure 3- 7 OLED memory map

3.9.3 Interfacing OLED display module to an ESP8266 NodeMCU

Connections are straightforward. Begin by connecting the VCC pin to the NodeMCU's 3.3V output and the GND pin to ground. Finally, connect the SCL and SDA pins to the ESP8266's I2C pins D1 and D2, respectively.

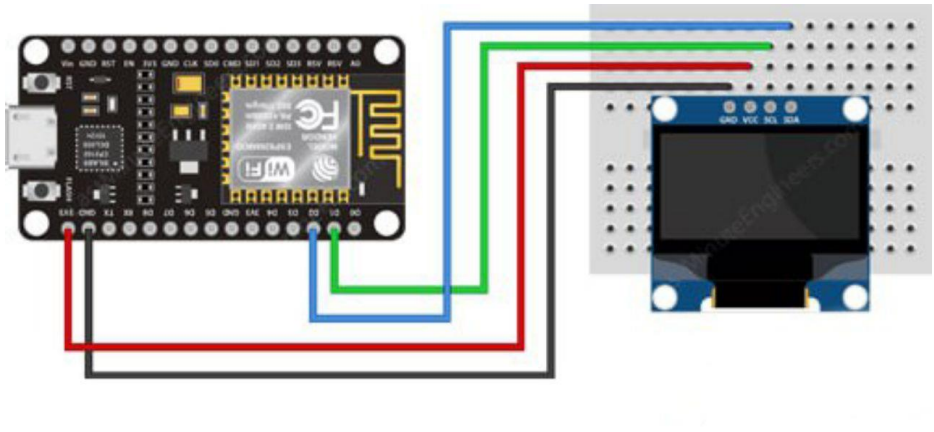


Figure 3- 8 Interfacing OLED display module to an ESP8266 NodeMCU

3.9.4 Library Installation

The SSD1306 controller of the OLED display has flexible but complex drivers. To use the SSD1306 controller, extensive knowledge of memory addressing is required. Fortunately, the Adafruit SSD1306 library was written to hide the complexities of the SSD1306 controller, allowing us to control the display with simple commands.

Chapter 4 Path planning

Chapter 4

Path planning

4.1 Introduction

Path planning is the process of finding a feasible and optimal path from a starting point to a goal point, while avoiding obstacles or other constraints in the environment.

To obtain the path planning software, there are 5 steps to follow:

1. Define the problem: This step involves specifying the start and goal locations whether if they are constant or GPS locations, as well as any constrains or obstacles/collisions in the environment.
2. Create the representation of the environment: This step involves creating a model of the environment that the quadcopter will navigate through. This can include a map or a grid of the environment, as well as information about obstacles, and other features that may affect the path.
3. Generate a search graph: This involves creating a graph or other data structure that represents the possible paths that the quadcopter could take through the environment, given the constraints and objectives.
4. Apply a search algorithm: This involves searching the graph to find the optimal path between the start and goal locations. This may involve exploring different paths, evaluating their feasibility and optimality, and updating the search space accordingly. There are different search algorithms with different logics to explore the graph and find the optimal path. Choosing the most suitable algorithm for our problem depends on several factors that must be taken under consideration including the characteristics of the environment, the capabilities of the quadcopter, and the specific requirements such as time, speed, or safety.
5. Execute the path: Once the path has been found, the quadcopter can follow it through the environment regardless the applied control way.

4.2 Path planning for the quadcopter

1. The start and goal point will be the output of the GPS through the sever feed. The output is in spherical coordinate system, so it has to be converted to the cartesian coordinate system to be suitable input to the path planning search algorithm. The environment will include two rectangle obstacles which may represent apartment blocks. This may be done using many approaches, but the simplest and what we used is through considering that the earth is in spherical shape whose radius is 6371km. The conversion is done using these equations:

$$x = R * \cos(latitude) * \cos(longtuide)$$

$$y = R * \cos(latitude) * \sin(logtuide)$$

2. The algorithm will not have a static map to be more general to pass any GPS location value. The constant dimensions of the constant map will deny the algorithm to prevent the algorithm to run on points outside its map. This will deny the quadcopter to move from and to any location.
3. For now, there are 2 obstacles to be taken into account. In addition, the environment will be divided using the graph data structure as this will be the simplest way to put the algorithm on. The simplicity of the discretization of our environment will affect the time, power, and memory consumption. The simpler the environment discretization, the less time, power, and memory consumption.
4. Our quadcopter characteristics:
   A) Our quadcopter is assumed to fly in a simple environment but with obstacles, these obstacles may be apartment blocks, banners, signs, and many other obstacles in Egypt.
   B) The quadcopter is highly maneuverable and can move in many directions.
   C) The primary concern of the software is the time constraints to reach the goal as quickly as possible, so we need an aggressive algorithm that prioritizes speed.
   D) The memory for this project is enough, so there is no problem if the algorithm's computational memory is a bit high.
   E) One more thing to be noticed, the algorithm should return accurate and optimal path in real-time as well.

5. For those characteristics, the algorithm which will be used for our quadcopter path planning is the rapidly exploring random tree (RRT) algorithm. In addition, some optimization techniques will be applied to reduce its consumption and to produce a smooth path. There are many other path planning algorithms that could be suitable for our scenario include A* and D* algorithms. However, these algorithms are less suited for real-time performance and may not be as aggressive as the RRT algorithm in prioritizing speed.
6. The algorithm will be applied on a flight controller, which will send the motion orders to the quadcopter.

4.3 Tree data structure

In graph data structure, each node can have many parents and many children, but the tree can have only one parent and many children.

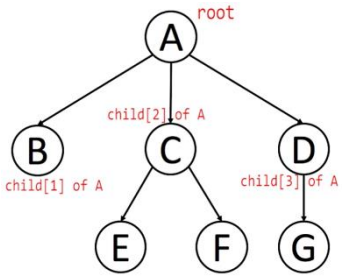


Figure 4- 1 tree data structure

The tree is a simplified graph with no cycles, which means there is only one path to reach any node.

- A is the root, which is the starting of the tree. B, C, D, E, F, and G are the nodes. Produced path is the way between the two points/nodes.
- Parent(B) = A

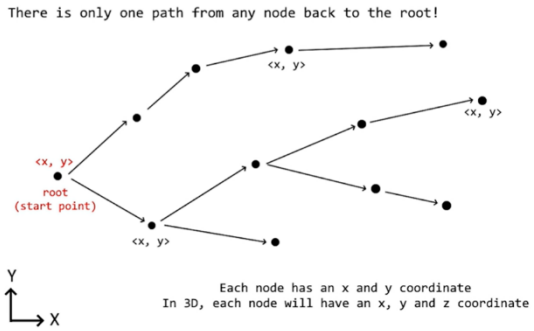


Figure 4- 2 Rapidly exploring random tree

Rapidly exploring random tree:

Each node has an X, Y location, one parent, and has many children.

4.4 Rapidly exploring random tree (RRT algorithm):

The tree is generated purely by sampling, and it expands put words in all directions, filling up the space. This is how it is done in case of no obstacles exist.

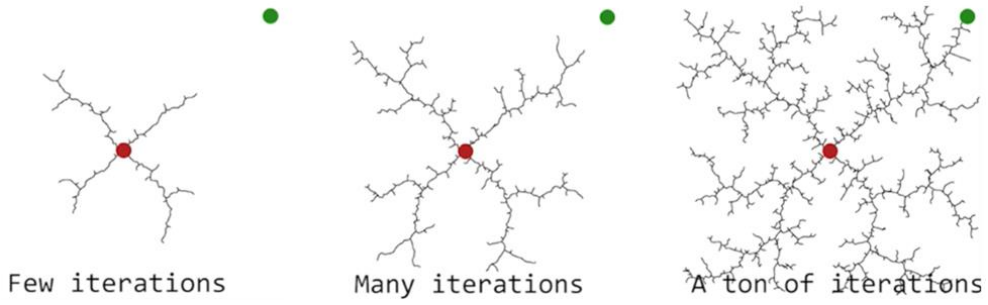


Figure 4- 3 rrt algorithm

4.4.1 Logic

Given two points (Start and goal) with one obstacle between them:

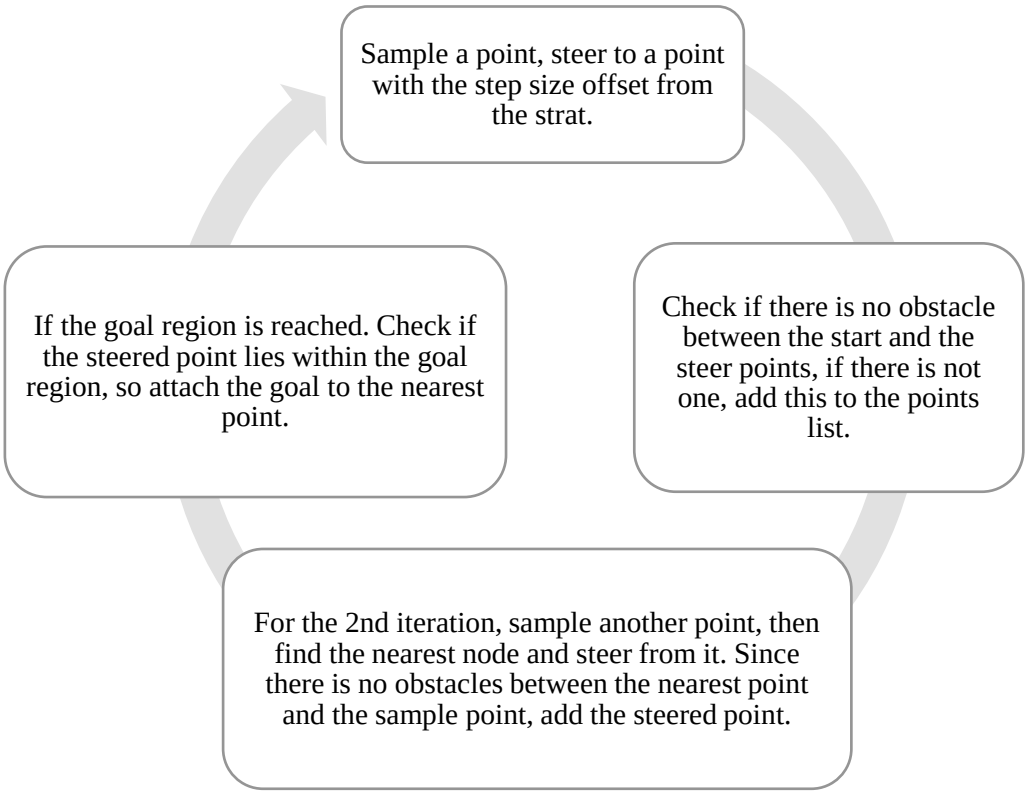


Figure 4- 4 rrt logic diagram

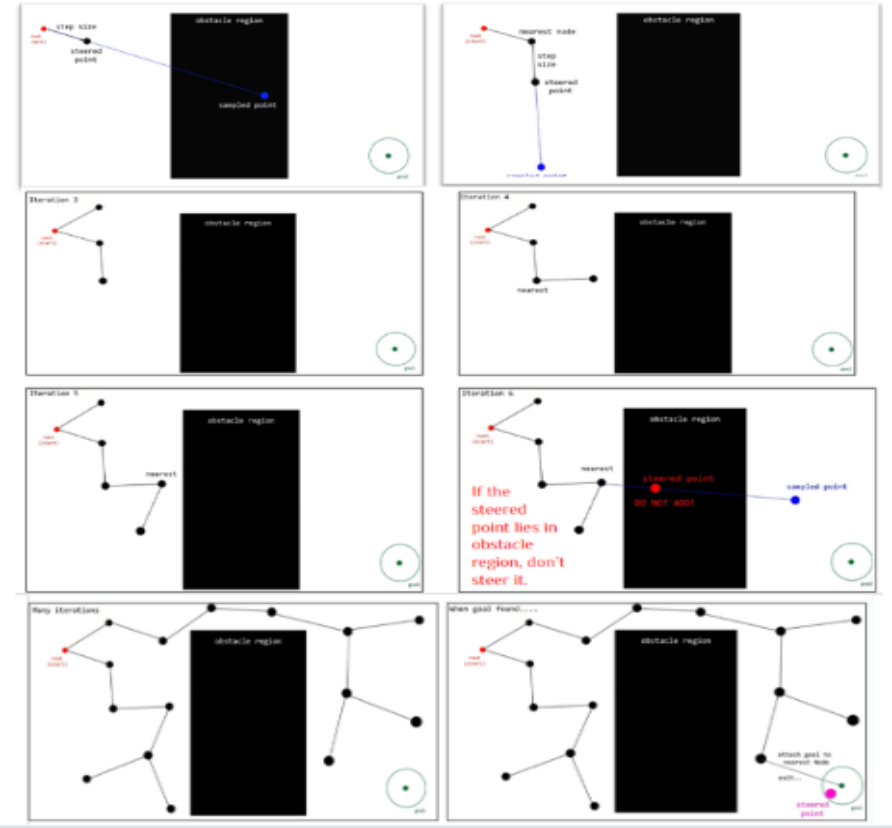


Figure 4- 5 rrt algorithm iterations

1) After reaching the goal, the last step is tree tracing, which means finding the full path. Each node can have many children, but it has only one parent. Since the goal parent is the nearest node, keep tracing the parent back until reaching the start point. Going from start to goal from the goal itself is a mistake, as there is only one path.

- The RRT algorithm can find a path between two nodes, but not the shortest path. SO, there is an updated version of RRT called RRT* which aims to find a shorter spot by optimization. It generates a lot more straighter paths in an obstacle free space.
- The key difference is that at each iteration, search within a neighboring region and find all nodes that are in that region from the nearest point.
- Nearest point is found the same way as RRT algorithm.

4.5 RRT* algorithm

1) In iteration 1: There is no any optimization to do; because there is only a start node which will be connected later.

2) In iteration 2:
   a. Sample a point ‘p’.
   b. Search a neighborhood region centered at ‘p’.
   c. Find that nodes are n1(start node), n2. So, the nearest point will be n2, but the path is not appended since it is obstacle free only. There are two additional steps must be done:

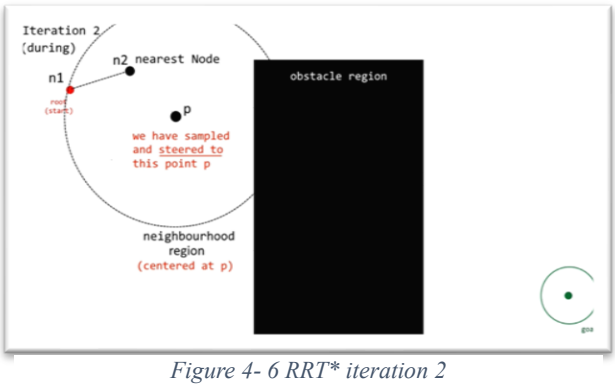


Figure 4- 6 RRT iteration 2*

1. Connect along the minimum cost:
   - The distance between n1, n2, and p is longer than the distance between n1 and p (directly). So, if the straight line from n1 to p is drawn, it will be shorter than going from n1 to n2 to p. this is one of the key differences in RRT*.

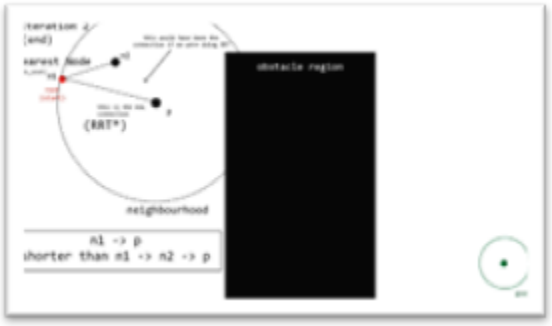


Figure 4- 7 rrt connect along the minimum cost step*

   - If there is a node outside the region as the goal, nothing will be done; because only the nodes within the neighboring region are ones who be looked at using the algorithm. Any nodes outside the neighboring region are an out of scope and the algorithm will not look at it.
   - For the extended nodes in figure 4- 8:

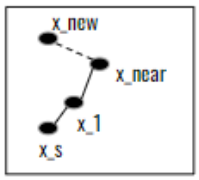


Figure 4- 8 RRT extended nodes*

$g_{x_{new}} = g_{x_{nearest}} + c_{x_{nearest},x_{new}}$

$G_{x_{new}}$ ➔ The cost of x_{new}(minimum cost to this node).

$G_{x_{nearest}}$ ➔ Distance between x_{near} an x_1+Distance between $x_1$and x_s.

$C_{x_{nearest}}$, x_{new}➔ Distance between x_{near} and x_{new}

So, total cost = Distance between x_s and x_1 + Distance between $x_1$and x_{near} + Distance between x_{near} and x_{new}.

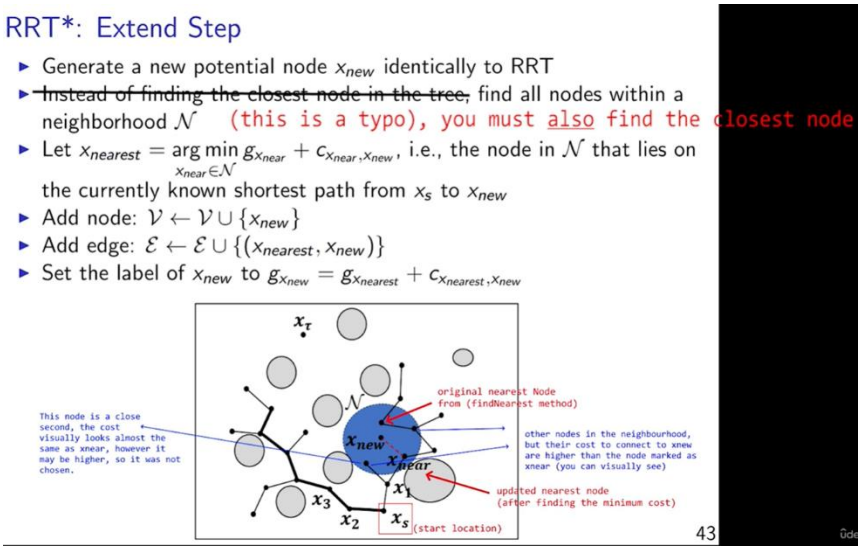


Figure 4- 9 RRT extend step*

2. Review step:

If the cost (distance) of x_{new}(This is resulted from finding the minimum path cost step) + the cost (distance) of the $x_{new}$to x_{near} is lower than the cost distance between x_{near} and the x_s(start point), then remove the edge between x near and its parent, and add a new edge between x_{near} and x_{new}.

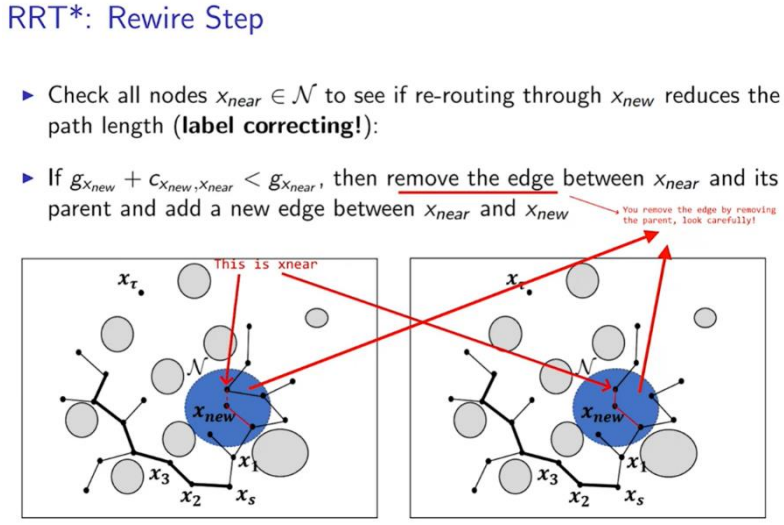


Figure 4- 10 RRT rewire step*

4.6 Informed RRT*

After finding the shortest path using RRT* algorithm. The algorithm can be optimized through sampling within an ellipse region to reduce the time consumed.

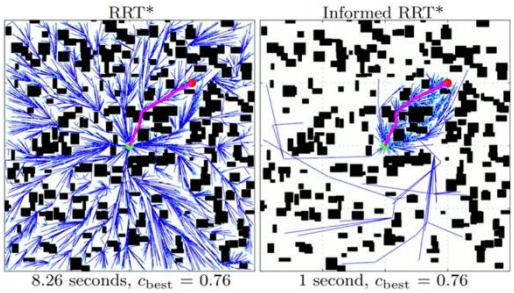


*Figure 4- 11 informed rrt**

The ellipse region covers the path and lets the sampling be done within these. That means that we don't need to sample anywhere else in the space, thus saving the time and finding shorter paths.

The ellipse itself has to be declared by some parameters, as not any ellipse matches the optimization.

- The minimum cost of the ellipse is the eucledian distance between the start and goal (c_{min}). It is not matter if there are any obstacles between them or nor. The value of the c_{min} is always the eucledian distance.

c_{min}➔ The minimum possible cost && c_{best}➔ The best solution found (the cost of RRT*).

As we go along, the distance between the start and the goal(c_{min}) decreases, as we come close to the goal point. So, the cost will decrease (c_{best}) as the ellipse will get smaller and smaller.

c_{best} will be close to be a line.

If we don't have any obstacles at all, c_{best}will pretty much be almost the same as c_{min}.

So, informed RRT* is RRT* algorithm result + sampling within an ellipse when the initial path is found using RRT* algorithm.

The result of more optimal path and reducing the time and power consumption makes it the best choice of RRTs algorithms to be used in our application.

Practically, there is need to know the ellipse equation to be able to represent its sampling inside it in coding.

$b^2 + c^2 = a^2$

$b^2 = a^2 - c^2$

Area = pi * a * b

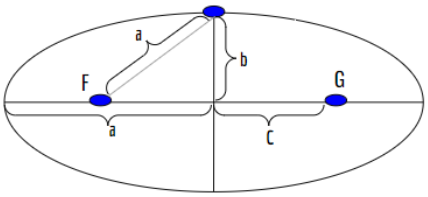


Figure 4- 12 ellipse axes

Where: a is the semi-major axis && b is the semi minor axis && Semi-major axis (the longest diamaeter) = ½ major axis && semi-minor axis (the shortest diameter)= ½ minor axis.

Ellipse equation

$$\sqrt{(x+c)^2 + (y-0)^2} + \sqrt{(x-c)^2 + (y-0)^2} = 2a$$

$$\sqrt{(x-c)^2 + (y-0)^2} = (2a - \sqrt{(x+c)^2 + (y-0)^2})$$

$$(x-c)^2 + y^2 = 4a^2 - 4a\sqrt{(x+c)^2 + y^2} + (x+c)^2 + y^2$$

$$x^2 - 2xc + c^2 + y^2 = 4a^2 - 4a\sqrt{(x+c)^2 + y^2} + x^2 + 2xc + c^2 + y^2$$

$$\left(4a\sqrt{(x+c)^2 + y^2}\right)^2 = (4a^2 + 4cx)^2$$

$$a^2[(x+c)^2 + y^2] = (a^2 + cx)^2$$

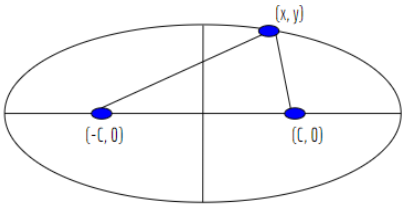
$$a^2x^2 + 2a^2xc + a^2c^2 + a^2y^2 = (a^2 + cx)^2$$

$$a^2x^2 + 2a^2xc + a^2c^2 + a^2y^2 = a^2 + 2a^2cx + c^2x^2$$

$$a^2x^2 - x^2c^2 + a^2y^2 = a^2 - a^2c^2$$

$$x^2(a^2 - c^2) + a^2y^2 = a^2(a^2 - c^2)$$

$$\frac{x^2}{a^2} + \frac{a^2y^2}{a^2(a^2 - c^2)} = 1$$



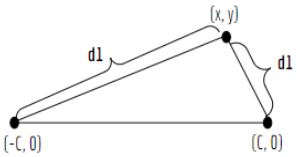


Figure 4- 13 ellipse equation

$$\frac{x^2}{a^2} + \frac{y^2}{(a^2 - c^2)} = 1$$

$$b^2 = a^2 - c^2$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

4.7 Algorithm sampling optimization:

The algorithm is based on the sampling idea. Sampling in the state space without any limitations of the random samples generated will consume power, time, and speed. So, there are two optimization techniques for generating random samples are applied:

1) Uniform sampling: It involves generating samples that are uniformly distributed within a specific range or domain.
2) Ellipse sampling: It involves generating samples that are distributed according to an ellipse shape.

Combining these two techniques together can potentially provide better optimization to generate samples that are both uniformly distributes and biased towards a particular region of the distribution.

In the code, this optimization is performed in `get_random_node` function. This function generates a random node object within a state space using ellipse sampling. The state space is defined as a rectangular domain with minimum and maximum x & y values, as well as a start and goal points which define the path to be optimized.

This function generated a random number between 0 and 1 using the `random.random()` function:

- If the random number is greater than 0.5, then the function generates a random point within a rectangular domain using the `random.uniform()` function.
- If the random number is less than or equal to 0.5, the function generates a random point on an ellipse centered between the start and goal points. The major axis of the ellipse is aligned with the line connecting the start and goal points, and the minor axis is half of the length of the major axis. To generate a random point on the ellipse, the function generates

a random angle `t` and a random radius `u` in the range [0, 1], and then maps these values onto the ellipse using the ellipse equation. The resulting x and y coordinates represent a random point on the ellipse, and are used to create a new node object which is returned by the function.

4.8 Generated path smoothness:

The smoothness of the path generated can have a significant effect on the motors of a quadcopter which are responsible for controlling its movement in space, including its speed, direction, and orientation. The smoother path generated by the algorithm, the less abrupt and sudden changes in speed, direction, and orientation will be, which can help reduce the stress on the motors and improve their lifespan.

In addition, smoother paths can also improve the stability and control of the quadcopter, particularly in the presence of external disturbances such as wind. A smoother path can reduce the amount of oscillation or vibration in the quadcopter's movement, which can help prevent instability and improve control.

The path smoothness in the project is done using the `weighted moving average/ weighted smoothing` technique. This function takes four arguments: `path` which is a list of 2D points, `weighted_data` which is a weight factor that controls how much the smoothed path should be adjusted towards the original path; `weight_smooth`, which is a weight factor that controls how much the smoothed path should be adjusted towards neighboring points; and tolerance, which is the tolerance threshold that determines when the smoothing process should stop.

The function iteratively adjusts the `newPath` list until it becomes smoother, using a weighted moving average approach. For each point in the path, the function calculates a new smoothed position by taking a weighted average of the original position (path[i][j]) and the current smoothed position (newPath[i][j]), with a weight factor `weight_data`. The function when calculates a new smoothed position for the same point by taking a weighted average of the neighbouring points' smoothed positions (newPath[i+1] [j] and newPath[i-1] [j]), with a weight factor `weight_smooth`. The function iteratively applies these adjustments until the change in the smoothed path falls below the specified tolerance threshold. This iterative process helps to gradually smooth out the original path, reducing any sharp turns or sudden changes in direction.

Overall, the path smoothness can be helpful for improving the quality and safety of a trajectory for our quadcopter. By reducing any sudden changes in direction or speed, the smoothed path can help to minimize stress on the motors and other components, and improve stability and control over the flight path.

In summary, Path smoothness is important for several reasons:

1. Efficiency: A smooth path allows the quadcopter to move more efficiently through the environment, reducing the time and energy required to complete the task. A jagged path with abrupt changes in direction or curvature would require the quadcopter to constantly adjust its speed and orientation, leading to wasted energy and slower movement.
2. Stability: A smooth path helps to ensure the stability of the quadcopter during flight. Any abrupt changes in direction or curvature can cause the quadcopter to become unstable and potentially crash. A smooth path reduces the likelihood of sudden changes in direction, making it easier for the quadcopter to maintain stability.
3. Safety: A smooth path can help to ensure the safety of the quadcopter and those around it. A jagged path with sudden changes in direction or curvature can cause the quadcopter to collide with obstacles or other objects in the environment. A smooth path reduces the risk of collisions and other accidents.
4. Aesthetics: A smooth path can make the quadcopter's flight more aesthetically pleasing to observe. A quadcopter that moves smoothly and gracefully through the environment is more visually appealing thanone that moves in a jerky or erratic manner. This is particularly important in applications such as aerial photography or videography, where the quadcopter's movement is being recorded for later viewing.

To achieve path smoothness in quadcopter path planning, various techniques can be used. One approach is to use smooth trajectory planning algorithms such as cubic splines, which generate a continuous curve that passes through a set of predefined waypoints. Another approach is to use path smoothing techniques that adjust the trajectory to reduce any abrupt changes in direction or curvature. This can be done using optimization-based methods or through the use of smoothing filters such as the Savitzky-Golay filter.

Chapter 5 Landing image processing

Chapter 5

Landing image processing

5.1 Landing system

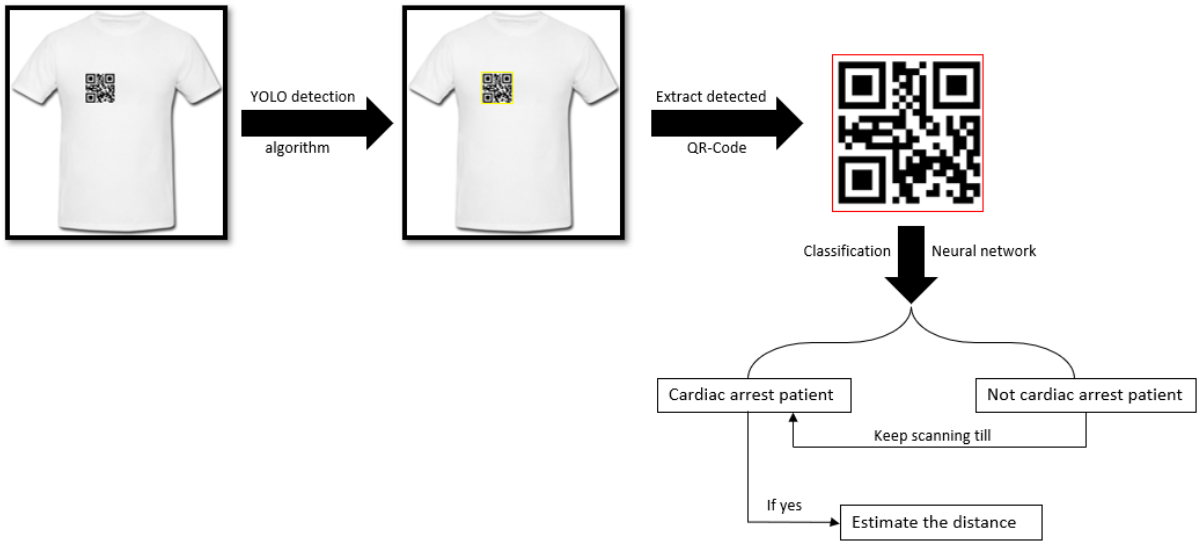


Figure 5- 1 landing system

The task is to recognize a specific QR-code generated specifically for cardiac arrest patients which will be put on their t-shirts. This task aims to enhance the quadcopter hardware to reach the actual destination location and avoid the GPS errors which mostly exceeds 3m, and to reach the proper location to start applying the CPR function on as well. This problem can be broken into 3 small parts:

1. QR code detection
2. QR code recognition and classification
3. Distance estimation

5.2 YOLO Algorithm

5.2.1 Introduction

YOLO (You Only Look Once) is ais a single-shot detection algorithm, which means that it can detect objects in an image with a single forward pass-through neural

network. This makes it faster than other object detection algorithms that require multiple passes through the network.

5.2.2 Algorithm logic steps

1) Input image: It takes an input image as its input.
2) Divide image into grid: The input image is divided into a grid cells. Each cell is responsible for detecting objects that appear within it.
3) Extract features: A convolutional neural network (CNN) is used to extract features from the input image. The network consists of several convolutional layers followed by a few fully connected layers. The convolutional layers are responsible for detecting low-level features such as edges and corners, while the fully connected layers are responsible for high-level feature extraction.
4) Predict bounding boxes and abjectness score: For each cell in the grid, YOLO predicts a set of bounding boxes and corresponding abjectness scores. Each bounding box is represented by the five parameters (x, y, w, h confidence). The (x, y) coordinates represent the center of the bounding box and (w, b) represent its width and height. The confidence score represents the probability that the bounding box contains an object.
5) Predict class probabilities: For each bounding box, YOLO predicts the class probabilities for the object within the box. The class probabilities represent the probability that the object belongs to a specific class, such as a car or a person.
6) Non-Maximum Suppression: The final step of the YOLO algorithm is non-maximum suppression (NMS), which is used to eliminate redundant bounding boxes. NMS works by selecting the bounding box with the highest abjectness score and suppressing any overlapping bounding boxes with lower abjectness scores.

5.2.3 Confidence score

It plays an important role in determining the likelihood that a bounding box contains an object of interest. The confidence score is a value between 0 and 1 that represents the probability that the corresponding bounding box contains an object.

During the object detection process in YOLO, the algorithm predicts multiple bounding boxes for each cell in the grid. These bounding boxes may overlap or be redundant, and it

is necessary to determine which bounding box is the most likely to contain an object of interest.

The confidence score is used to determine the likelihood that each bounding box contains an object. Bounding boxes with high confidence scores are more likely to contain objects than those with low confidence scores. Therefore, during the non-maximum suppression step of the algorithm, redundant bounding boxes with lower confidence scores are suppressed in favor of those with higher confidence scores.

By using the confidence score to filter out redundant or false-positive detections, the YOLO algorithm is able to improve the accuracy of object detection. It is worth noting that the threshold for the confidence score can be adjusted based on the specific requirements of the application. A higher threshold will result in fewer but more accurate detections, while a lower threshold will result in more detections but with potentially lower accuracy.

5.2.4 non-maximum suppression (NMS) technique

A technique used to eliminate redundant bounding boxes and improve the accuracy of the final detections. The basic idea behind NMS is to select the bounding box with the highest abjectness score and suppress any overlapping bounding boxes with lower abjectness scores.

Steps:

1. Sort Bounding Boxes by Objectless Score: The first step in NMS is to sort all the bounding boxes predicted by YOLO for each cell in the grid by their objectless score in descending order. The bounding box with the highest objectless score is selected as the starting point.
2. Calculate Intersection over Union (IoU): For each pair of overlapping bounding boxes, the IoU is calculated. IoU is a measure of the overlap between two bounding boxes and is calculated as the ratio of the area of their intersection to the area of their union.
3. Suppress Overlapping Bounding Boxes: For each bounding box with an objectless score lower than the currently selected bounding box, if the IoU is above a certain threshold, it is suppressed and removed from the list of detections. The threshold is typically set to a value between 0.3 and 0.5, depending on the specific requirements of the application.

4. Select Next Highest Scoring Bounding Box: The next highest scoring bounding box is selected from the remaining list of detections, and the processor calculating IoU and suppressing overlapping bounding boxes is repeated until all the bounding boxes have been examined.
5. Repeat for All Cells in the Grid: Steps 1-4 are repeated for each cell in the grid, resulting in a final set of non-redundant bounding boxes with high objectless scores and accurate object locations.

5.3 QR code detection

The aim of this part is to get the camera closer to the QR-Code. This is considered as object detection problem which requires 3 steps to be done, these steps are:

1. Data collection and labeling, this is done by zipping google images includes different QR-Codes on t-shirts, then labeling them using labelImg library.
2. This dataset is huge, so it will be uploaded to google drive and then will be trained on the GPU of google colab. The training is done using YOLO v.3 as it is very fast and accurate; as it can detect from only one scan while the other architectures need many scans to be able to detect the object. This is why YOLO is used here. The training is done using custom command lines that are:

```
# Check if NVIDIA GPU is enabled
!nvidia-smi

# mount google drive
from google.colab import drive
drive.mount('/content/gdrive')
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive

# 1) Clone the darknet
!git clone https://github.com/AlexeyAB/darknet

# 2) Compile Darknet using Nvidia GPU
# change makefile to have GPU and OPENCV enabled
%cd darknet
```

```
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!make

# 3) Configure Darknet network for training YOLO V3
!cp cfg/yolov3.cfg cfg/yolov3_training.cfg

!sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg
!sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3_training.cfg
!sed -i 's/max_batches = 500200/max_batches = 4000/'
cfg/yolov3_training.cfg
!sed -i '610 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '696 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '783 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '603 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '689 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '776 s@filters=255@filters=18@' cfg/yolov3_training.cfg

# Create folder on google drive so that we can save there the weights
!mkdir "/mydrive/yolov3"

!echo "qrcode" > data/obj.names
!echo -e 'classes= 1\ntrain  = data/train.txt\nvalid  =
data/test.txt\nnames = data/obj.names\nbackup = /mydrive/yolov3' >
data/obj.data
!mkdir data/obj

# Download weights darknet model 53
!wget https://pjreddie.com/media/files/darknet53.conv.74

# 4) Extract Images
!unzip /mydrive/yolov3/images.zip -d data/obj
```

```
# 6) Start the training
# Start the training
!./darknet detector train data/obj.data cfg/yolov3_training.cfg
darknet53.conv.74 -dont_show
```

3. The training will produce a weights file for each 1000 iterations; as the darknet makes a backup of the model each 1000 iterations. These weights will be used for testing task.

5.4 QR code recognition and classification

5.4.1 Neural network architecture

The aim of this part is to classify our specific QR-Code to decide whether this is a QR code for a cardiac arrest patient or for any other usage. This is done by building a convolutional neural network architecture which is commonly used for many image classification problems. This architecture consists of many layers ordered in a specific way to enable the neural network to distinguish our QR code.

The architecture consists of several convolutional layers, which are capable of automatically learning hierarchical representations of image features. Each convolutional layer applies a set of learned filters to the previous layer's output, creating a new set of feature maps that capture increasingly complex image features.

- The MaxPooling2D layers down sample the feature maps to reduce the number of parameters and computation required by the network.
- The Flatten layer converts the 2D feature maps into a 1D vector, which is then fed into a fully connected Dense layer. This layer applies a linear transformation to the input vector, followed by a ReLU activation function, which introduces non-linearity into the model. The final Dense layer has a single output unit with a sigmoid activation function, which produces a probability between 0 and 1 indicating the likelihood that the input image belongs to the positive class (cardiac_arrest_patient).
- The architecture ordered layers:
  - Conv2D layer with 32 filters and a 3x3 kernel, followed by a ReLU activation function.
  - MaxPooling2D layer with a 2x2 pool size.

  - Conv2D layer with 64 filters and a 3x3 kernel, followed by a ReLU activation function.
  - MaxPooling2D layer with a 2x2 pool size.
  - Conv2D layer with 128 filters and a 3x3 kernel, followed by a ReLU activation function.
  - MaxPooling2D layer with a 2x2 pool size.
  - Flatten layer.
  - Dense layer with 128 units and a ReLU activation function.
  - Dense layer with 1 unit and a sigmoid activation function.
- The input to the network is a 3-channel image (224 x 224 pixels). The first layer is a Conv2D layer with 32 filters and a 3x3 kernel, followed by a ReLU activation function. The output of this layer is then passed to a MaxPooling2D layer with a 2x2 pool size.
- The next two layers are identical to the first, except that they have 64 and 128 filters respectively. Each layer is followed by a ReLU activation function and a MaxPooling2D layer.
- The output of the final MaxPooling2D layer is then passed to a Flatten layer, which converts the 2D feature maps into a 1D vector. This vector is then passed to a fully connected Dense layer with 128 units and a ReLU activation function.
- Finally, the output of the Dense layer is passed to the final Dense layer with a single output unit and a sigmoid activation function. This produces a probability between 0 and 1 indicating the likelihood that the input image belongs to the positive class (cardiac_arrest_patient in our case).
- Each layer function:
  - Conv2D layer: This layer applies a set of filters to the input image to extract features. Each filter slides over the input image and produces a new output feature map. The specific filters are learned during training and can capture various image features such as edges, corners, and textures.
  - ReLU activation function: This activation function is applied element-wise to the output of each Conv2D layer. It introduces non-linearity into the model and helps to make the model more expressive.

- MaxPooling2D layer: This layer down-samples the feature maps produced by the previous layer. It slides a window over the feature maps and outputs the maximum value within each window. This reduces the spatial dimensionality of the feature maps and helps to reduce overfitting.
- Flatten layer: This layer flattens the 2D feature maps produced by the previous layer into a 1D vector. This allows the feature maps to be fed into a fully connected Dense layer.
- Dense layer: This layer applies a linear transformation to the input vector, followed by an activation function. The ReLU activation function introduces non-linearity into the model and helps to make the model more expressive. The final Dense layer has a single output unit with a sigmoid activation function, which produces a probability between 0 and 1 indicating the likelihood that the input image belongs to the positive class (cardiac_arrest_patient).

5.4.2 Compressing the model from .h5 format into. tflite format

This step has many several benefits when deploying the model on Raspberry pi as a limited memory capacity embedded device:

Reduced Memory Usage: The. tflite format is optimized for deployment on mobile and embedded devices, which often have limited memory capacity. By compressing the model file into the. tflite format, its memory usage can be reduced, making it easier to deploy on a Raspberry Pi.

Improved Inference Performance: The. tflite format is designed to be efficient for mobile and embedded devices, which often have limited processing power. By compressing the model file into the. tflite format, its structure can be optimized for inference on a Raspberry Pi, resulting in improved inference performance and lower latency.

Easy Integration with TensorFlow Lite: The. tflite format is the preferred format for deployment on TensorFlow Lite, which is a lightweight framework for mobile and embedded devices. By converting a model file to the. tflite format, it can be easily integrated with TensorFlow Lite and run on a Raspberry Pi.

Reduced Power Consumption: The smaller size and optimized structure of the. tflite model can reduce the power consumption required to run the model on a Raspberry Pi. This can be particularly important for battery-powered applications and can help to extend the battery life of the device.

5.5 Distance estimation

By passing the camera parameters into the distance estimation equation:

Focal length: It is a fixed value parameter varies from each camera to another. For Pi camera v2.1, focal length is equal to 3.06mm. Pi camera v2.1 has a sensor size of ¼ inches (3.76mm x 2.74mm), and has a resolution of 3280 x 2464 pixels (8 megapixels). From this, we can calculate the sensor width and pixel size in mm:

Pixel size(mm) = sensor size(pixels) * pixel size(mm) / focal length (mm)

Pixel size(mm) = sensor width(mm) / sensor width(pixels)

Thus:

Sensor Width (mm) = (3.76 mm * 3280 pixels) / (3.09 mm * 2464 pixels) = 4.62 mm

Pixel Size (mm) = 4.62 mm / 3280 pixels = 0.00141097 mm/pixel

Then, the estimated distance can be calculated by passing these parameters and the QR code width into the following equation:

Distance in m = [(QR code size in cm * focal length in mm) / (object width in pixels * pixel size in mm)] / 1000000

Chapter 6 CPR based on embedded AVR microcontroller

Chapter 6

CPR based on embedded AVR microcontroller

6.1 Introduction

Cardiopulmonary resuscitation (CPR) is a life-saving technique used to revive a person who has stopped breathing or whose heart has stopped. CPR involves a combination of chest compressions and rescue breaths to manually circulate blood and oxygen throughout the body until emergency medical services arrive.

An embedded AVR microcontroller is a small computer chip that is built into an electronic device to control its functions. In the context of CPR, an embedded AVR microcontroller can be used to create a device that helps guide rescuers through the steps of performing CPR correctly.

To create a CPR device using an embedded AVR microcontroller, a combination of a DC motor for pushing the chest up and down and a servo motor for separating the drone and CPR parts can be used. The motors can be controlled using a Bluetooth module, which allows the device to be remotely controlled by a rescuer or medical professional. Additionally, the use of a drone can help provide faster access to emergency medical services in remote or hard-to-reach areas.

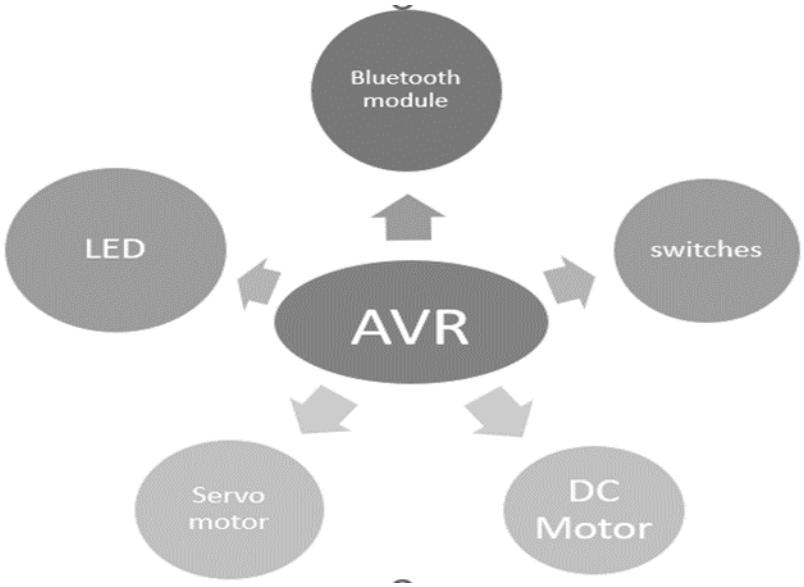


Figure 6- 1 CPR components

6.2 ATmega32 Microcontroller

The ATmega32 is an 8-bit microcontroller from the AVR family, designed and manufactured by Microchip Technology (previously Atmel Corporation). It is a popular choice for embedded systems projects due to its rich feature set, versatility, and ease of use.

Key features:

1. Architecture: The ATmega32 is based on the Harvard architecture, which separates the program memory and data memory. It features an 8-bit RISC (Reduced Instruction Set Computer) CPU with a rich set of instructions, making it efficient for a wide range of applications.
2. Flash Memory: The ATmega32 has 32KB of in-system self-programmable flash memory. This allows you to easily write and update the program code directly on the microcontroller, making it flexible for firmware development.
3. RAM and EEPROM: It has 2KB of SRAM (Static Random-Access Memory) for storing data during runtime and 1KB of EEPROM (Electrically Erasable Programmable Read-Only Memory) for non-volatile data storage.
4. I/O Ports: The ATmega32 offers a total of 32 general-purpose I/O (GPIO) pins, organized into four ports (Port A, B, C, and D). These pins can be configured as inputs or outputs, and can be used for interfacing with various external devices and peripherals.
5. Timers/Counters: It features three 16-bit timers/counters (Timer/Counter0, Timer/Counter1, and Timer/Counter2) with various operating modes. These timers are useful for tasks such as generating PWM signals, measuring time intervals, and creating accurate timing delays.
6. Serial Communication: The microcontroller supports multiple serial communication interfaces, including USART (Universal Synchronous and Asynchronous Receiver and Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit) for communication with other devices or peripherals.
7. Analog-to-Digital Converter (ADC): The ATmega32 includes a 10-bit ADC with eight channels, allowing you to convert analog signals to digital values for processing.

8. Interrupt System: It has an efficient interrupt handling mechanism that supports both external and internal interrupts, allowing the microcontroller to respond quickly to real-time events or external signals.
9. Power Management: The ATmega32 incorporates power-saving features such as sleep modes, allowing you to optimize power consumption in battery-powered applications.
10. Development Tools: Microchip provides a comprehensive set of development tools, including an Integrated Development Environment (IDE), compilers, programmers, and debuggers, making it easy to develop, program, and debug applications for the ATmega32.

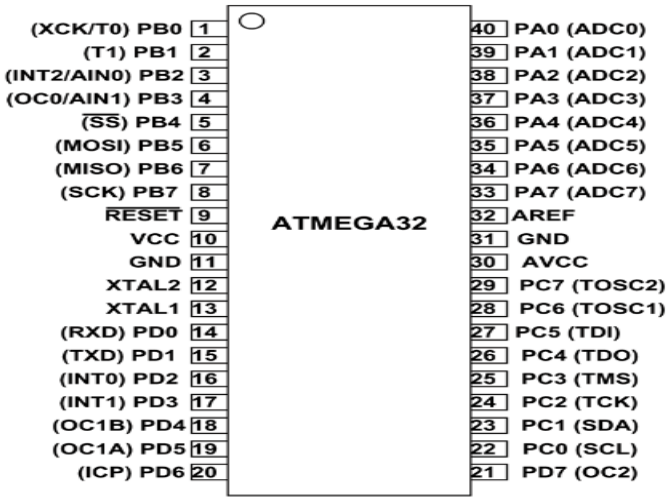


Figure 6- 2 ATmega32 pinout

6.3 DC Motor

6.3.1 Introduction

The function of a DC motor is to convert electrical energy into mechanical energy. This is achieved through the interaction between the magnetic fields of the stator and rotor.

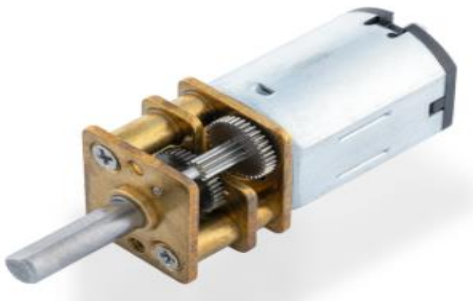


Figure 6- 3 DC Motor

When a current is passed through the stator, it creates a magnetic field. This magnetic field interacts with the magnetic field of the rotor, causing it to rotate. The commutator ensures that the current flowing through the rotor

changes direction at the appropriate times, allowing the rotor to continue rotating in the same direction.

The speed and torque of the DC motor can be controlled by adjusting the voltage supplied to the motor. The greater the voltage supplied to the motor, the faster it will rotate and the greater the torque it will produce. Conversely, reducing the voltage supplied to the motor will slow down the rotation speed and reduce the torque.

6.3.2 Function of DC motor in project

In CPR, a DC motor can be used to perform the chest compressions. The DC motor is connected to the embedded AVR microcontroller, which controls the speed and direction of the motor to push the chest up and down. The motor is typically connected to a mechanical arm that is used to apply the compressions to the patient's chest.

6.3.3 Interfacing DC Motor with AVR

The ULN2003 is a popular Darlington transistor array that can be used to interface DC motors with an AVR microcontroller. Interfacing a DC motor with an AVR microcontroller via ULN2003 Darlington pair is a simple and effective method. The ULN2003 can drive up to 500mA and 50V, which makes it suitable for a wide range of DC motors.

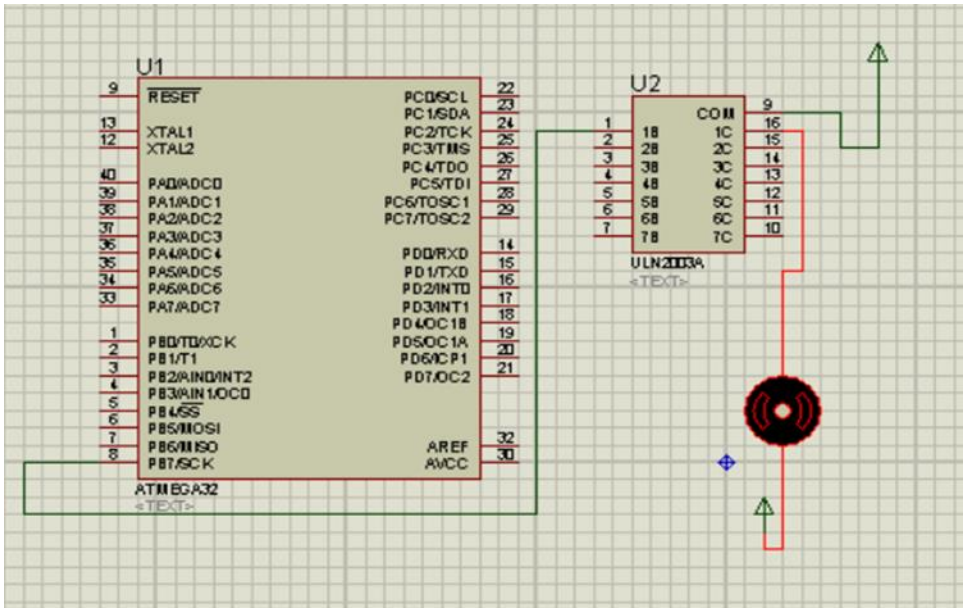


Figure 6- 4 circuit diagram of AVR with DC motor

In this circuit, the ULN2003 Darlington transistor array is used as a motor driver IC. The positive terminal of the DC motor is connected to the output pin 1 of the ULN2003, while the other terminal is connected to 5V. The ULN2003 is then connected to the ATmega32 microcontroller pin B7, which sends the signals to the motor driver to control the motor.

Make sure to properly connect the ULN2003 to the ATmega32 and to provide the necessary power supply for the motor and the microcontroller.

6.4 Bluetooth module

6.4.1 Basic concept

Bluetooth BLE module is a technology that acts as an interface that aids the wireless Bluetooth Low energy connection of any two devices and establishes a protocol for the communication of data between the devices. Bluetooth low energy module's mediated data communication range is usually an average of tens of meters and data is communicated in specified frequency bands.

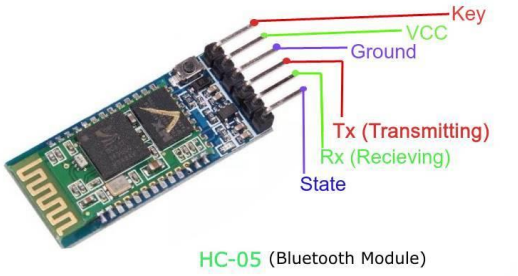


Figure 6- 5 Bluetooth module

There are various brands, types, models and classifications of Bluetooth modules. Bluetooth Modules' diversity in application makes them one of the most widely-accepted Internet of things (IoT) connectivity protocols.

HC-05 is a Bluetooth device used for wireless communication. It works on serial communication (USART).

It is a 6 pin module.

The device can be used in 2 modes; data mode and command mode.

The data mode is used for data transfer between devices whereas command mode is used for changing the settings of the Bluetooth module.

AT commands are required in command mode.

The module works on 5V or 3.3V. It has an onboard 5V to 3.3V regulator.

As the HC-05 Bluetooth module has a 3.3 V level for RX/TX and the microcontroller can detect 3.3 V level, so, no need to shift the transmit level of the HC-05 module. But we need to shift the transmit voltage level from the microcontroller to RX of the HC-05 module.

Interfacing Bluetooth module with AVR microcontroller via UART

Interfacing a Bluetooth module with an AVR microcontroller using UART (Universal Asynchronous Receiver-Transmitter) communication is a common way to enable wireless communication between the microcontroller and other devices. The AVR microcontroller can send and receive data wirelessly, allowing for more flexibility and mobility in applications.

UART communication is a simple and efficient serial communication protocol that enables two devices to communicate with each other by sending and receiving data one bit at a time. It uses a common ground for reference and two separate signal wires for transmitting and receiving data. The AVR microcontroller has built-in hardware USART (Universal Synchronous/Asynchronous Receiver-Transmitter) modules that can be configured to communicate using UART protocol.

To interface a Bluetooth module with an AVR microcontroller via UART protocol, the microcontroller must be programmed to initialize the UART communication and handle incoming and outgoing data. The Bluetooth module must also be configured to set the device name, baud rate, and other parameters. Once the Bluetooth module is properly configured and the microcontroller code is written, the two devices can communicate wirelessly using UART protocol.

In the following circuit diagram, the Bluetooth module is connected to the AVR microcontroller using two wires for UART communication: RXD (receive data) and TXD (transmit data). The RXD pin of the Bluetooth module is connected to the TXD pin of the AVR microcontroller, and the TXD pin of the Bluetooth module is connected to the RXD pin of the AVR microcontroller. The Bluetooth module and AVR microcontroller share a common ground. The Bluetooth module is also powered by +5V and GND pins.

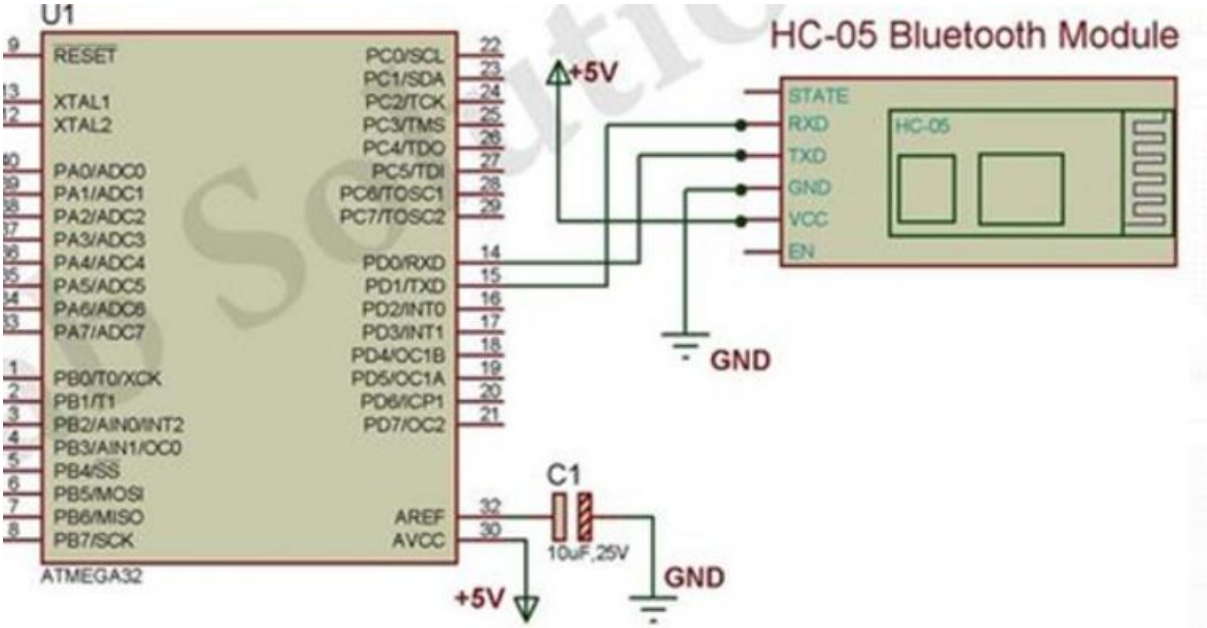


Figure 6- 6 circuit diagram of Bluetooth module with AVR

6.4.2 Function of Bluetooth in project

The function of the Bluetooth module in this project is to provide a wireless communication link between the Raspberry Pi and the AVR microcontroller, enabling remote control of the drone's CPR and push up/down functions.

Specifically, the Bluetooth module receives commands from the Raspberry Pi over UART, then sends these commands wirelessly to the AVR microcontroller. This allows the Raspberry Pi to control the drone without needing a physical connection to the drone itself.

The Bluetooth module acts as a wireless serial port, converting data received from the Raspberry Pi over UART into a wireless format that can be transmitted to AVR microcontroller using Bluetooth.

6.5 Servo motor

6.5.1 Basic concept

Figure 6- 7 Servo motor

A servo motor is a type of motor commonly used in robotics, automation, and other applications that require precise control of

angular position. The basic concept of a servo motor is to use a feedback mechanism to accurately position the motor shaft to a desired angle.

A typical servo motor consists of a DC motor, a gear train, a control circuit, and a potentiometer. The motor provides the rotational force, while the gear train reduces the speed and increases the torque of the motor output. The potentiometer is typically attached to the motor shaft and provides feedback to the control circuit about the motor's position.

The control circuit is responsible for sending signals to the motor to move it to a specific position. These signals are typically in the form of pulse-width modulated (PWM) signals, where the width of the pulse determines the desired position of the motor. The control circuit compares the desired position with the feedback from the potentiometer, and adjusts the motor output to move it to the desired position.
One of the key features of servo motors is their ability to maintain their position even when an external force is applied. This is due to the feedback mechanism, which constantly monitors the motor's position and adjusts the output accordingly. Servo motors are also known for their high accuracy and repeatability, making them suitable for applications that require precise positioning.

6.5.2 Interfacing Servo motor with AVR

To interface a servo motor with an AVR microcontroller, we'll need to connect the servo motor to one of the microcontroller's output pins and generate appropriate control signals to control the position of the servo motor.

Steps:

1) Determine the servo motor's power requirements: Check the voltage and current requirements of the servo motor. Make sure the power supply we use can provide sufficient voltage and current to drive the motor.
2) Connect the servo motor: Connect the servo motor to the microcontroller's output pin. Typically, servo motors have three wires: power (+5V), ground (GND), and control signal. Connect the servo motor's power and ground wires to the appropriate power

and ground pins on the microcontroller or an external power supply. Connect the control signal wire to one of the microcontroller's output pins.

3) Configure the microcontroller's output pin: Set the corresponding output pin as an output of AVR microcontroller's code. This allows us to control the servo motor's position by generating the appropriate control signals.
4) Generate control signals: Servo motors use a control signal known as a pulse-width modulation (PWM) signal to determine the desired position. The width of the PWM signal's high pulse corresponds to the desired position of the servo motor. Use the microcontroller's timers or software-generated PWM to generate the control signal. The specific width of the PWM signal's high pulse will depend on the servo motor's specifications and the desired position range.
5) Write the control signal to the output pin: In our code, calculate and write the appropriate PWM signal to the microcontroller's output pin connected to the servo motor. The duration of the high pulse will determine the servo motor's position.
6) Repeat the process: Continuously update the PWM signal to maintain the desired position of the servo motor. we can vary the width of the high pulse to change the position of the servo motor.

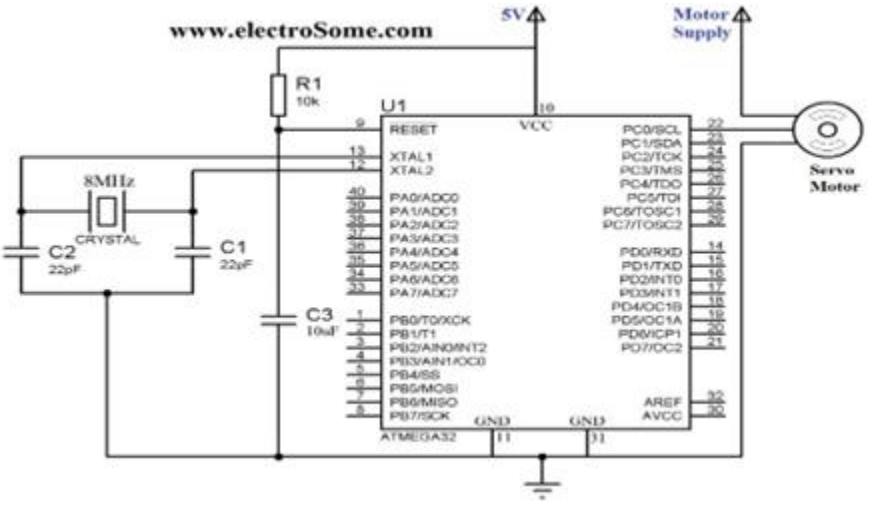


Figure 6- 8 Circuit diagram of servo motor with AVR

6.5.3 Usage in the quadcopter

In our project, we are utilizing a servo motor to separate the CPR (Cardiopulmonary Resuscitation) part from the drone. The purpose of this separation is to provide a modular and flexible design that allows the drone to perform its primary functions while also enabling the CPR functionality when required.

The servo motor serves as a mechanical actuator in this setup. It is responsible for controlling the movement of a physical barrier or partition within the drone. When the CPR functionality is activated, the servo motor adjusts the position to create a physical separation between the drone's primary functions and the CPR module.

This separation ensures that the CPR mechanism operates independently from the rest of the drone. By physically isolating the CPR part, we can focus on performing chest compressions and other life-saving procedures without interference or disruption from the drone's other components.

6.6 PWM generation

A PWM (Pulse Width Modulation) signal is a type of signal that is commonly used to control the average power or voltage delivered to a load. It achieves this by varying the width of pulses in a periodic signal. In the context of AVR microcontrollers, PWM signals are often used to control the position or speed of motors, including servo motors.

To generate a PWM signal in AVR microcontrollers, we used the built-in hardware support for PWM generation provided by the timers. Here's a basic approach to generate PWM signals in AVR microcontrollers:

1) Configure the Timer: Choose a suitable timer (e.g., Timer/Counter1, Timer/Counter2) and configure its mode of operation. Set the timer to operate in a PWM mode, which allows generating PWM signals on specific output pins.
2) Set the PWM Frequency: Set the desired frequency for the PWM signal by configuring the timer's prescaler and the compare match values. The frequency is determined by the timer's clock source and the prescaler value.
3) Set the Duty Cycle: Determine the desired duty cycle for the PWM signal. The duty cycle represents the percentage of time the signal is high (ON) within a complete cycle.

Calculate the appropriate compare match value based on the desired duty cycle and the timer's resolution.

4) Configure the Output Pin: Set the corresponding output pin as an output in the microcontroller's code. This pin will generate the PWM signal.
5) Enable PWM Generation: Enable the timer's PWM generation mode and configure the necessary registers to control the PWM signal's properties, such as polarity and mode of operation.
6) Adjust the Duty Cycle: To change the position or speed of the motor, update the compare match value periodically according to the desired duty cycle. This can be done in the main program loop or through interrupt service routines.

6.7 Switches

6.7.1 Introduction

A switch push button, also known as a push button switch or momentary switch, is a type of switch that remains in its actuated state only as long as force is applied to it. When the force is released, the switch returns to its original position. Push button switches are widely used in electronic devices and systems for momentary control functions and user input.

Figure 6- 9 Switch

6.7.2 Interfacing switch with ATmega32

To interface a push button, switch with the ATmega32 microcontroller, utilize the internal pull-up resistor, and generate an interrupt when the button is pressed.

1. Circuit Connection:
   - Connect one terminal of the push button switch to a digital input pin (e.g., PORTx) of the ATmega32.
   - Connect the other terminal of the push button switch to a common ground (GND) pin of the ATmega32.
2. Enable Internal Pull-up Resistor:
   - Configure the respective digital input pin as an input.

- Enable the internal pull-up resistor for that pin by setting the corresponding bit in the PORT register.

3. Enable Interrupt:
   - Configure the interrupt settings to generate an interrupt when the button is pressed or released.
   - Choose the appropriate interrupt mode. For example, we can use the INT0 (external interrupt 0) or INT1 (external interrupt 1) pin on the ATmega32.
   - Enable the specific interrupt and set the corresponding interrupt control registers (EIMSK, EICRA, EIFR) accordingly. Refer to the ATmega32 datasheet for specific register settings and configurations.
4. Implement Interrupt Service Routines (ISRs):
   - Define separate ISRs for each switch interrupt.
   - Inside each ISR, write code to handle the desired actions based on the switch states and requirements.
5. Enable Global Interrupts:
   - Enable global interrupts to ensure that interrupts are processed by setting the Global Interrupt Enable (GIE) bit in the Status Register (SREG).

6.7.3 Usage of switch in project

In our project with the AVR microcontroller, switches are used to generate interrupts and control the operation of a DC motor for initiating or stopping the CPR function, as well as separating a servo motor (CPR part) from a drone. The switches are connected to specific digital input pins, and their states trigger interrupts. When the DC motor switch is pressed, an interrupt starts the motor for CPR initiation, and releasing the switch triggers another interrupt to stop the motor. Similarly, the servo motor switch triggers an interrupt to separate or reconnect the servo motor from the drone. By utilizing interrupts and switches in this way, we can precisely control the CPR function and separate the CPR part from the drone, enhancing the effectiveness and integration of our project.

Chapter 7 Drone control

Chapter 7

Drone control

7.1 Introduction

In this project, the goal is to control a drone using a Raspberry Pi 4 (Rpi4) and a control script written using DroneKit source code. The project involves implementing several algorithms such as perception, shortest path, QR code detection and recognition, and distance estimation.

Drone control part starts with building a drone control script using DroneKit source code and documentation. The script is then tested using software in the loop (SITL) to ensure that all systems are working correctly and that the implemented algorithms are properly calibrated.

The project also involves monitoring the battery health and EKF parameters of the drone and identifying potential barriers or problems that may arise during testing.

Next, the Rpi4 is connected to a camera and Xserver to enable QR code landing. The final step in the project is to connect all remaining parts of the project using a server and Bluetooth.

The server part of the project involves subscribing to the heartbeat data of a patient watch on an Adafruit server to detect if an arrhythmic case occurs. If an arrhythmic case is detected, the drone is given permission to launch and access the watch's GPS data (longitude, latitude) to use as a goal point for the shortest path algorithm.

Once the drone reaches the detected location, the camera is opened to allow the distance estimation algorithm to work. Finally, the drone is moved closer to the QR code at the same altitude to enable landing.

Accessing CPR is achieved using Bluetooth, and a message is sent to start its function.

In summary, this part aims to control a drone using a Raspberry Pi 4 and a control script written using DroneKit source code. The part involves implementing several algorithms, including perception, shortest path, QR code detection and recognition, and distance estimation. The project is carried out in several stages, starting with the development of the drone control script and testing it using software in the loop. The project also involves monitoring the battery health and EKF parameters of the drone and identifying potential barriers or problems that may arise

during testing. The Rpi4 is then connected to a camera and Xserver to enable QR code landing. Finally, the project is completed by connecting all remaining parts of the project using a server and Bluetooth, which includes subscribing to the heartbeat data of a patient watch and detecting arrhythmic cases, using the watch's GPS data as a goal point for the shortest path algorithm, and landing the drone using the QR code detection and recognition algorithm.

7.2 ArduPilot source code

To install ardupilot from github:

1. Install the required dependencies: Before installing ArduPilot, install some dependencies. By opening a terminal window and running the following command to install the required dependencies:

```
sudo apt-get update
sudo apt-get install git python python-dev python-pip python-matplotlib
python-serial python-wxgtk3.0 python-wxtools python-lxml python-scipy
python-opencv ccache gawk
```

2. Clone the ArduPilot repository: Clone the ArduPilot repository from GitHub by running the following command:

```
git clone https://github.com/ArduPilot/ardupilot.git
```

3. Checkout the latest stable release: Change to the ardupilot directory and checkout the latest stable release by running the following commands:

```
cd ardupilot
git submodule update --init --recursive
git checkout Copter-4.1
```

4. Compile the firmware: Compile the ArduPilot firmware by running the following commands:

```
./waf configure --board sitl
./waf copter
```

The above commands will configure and compile the ArduPilot firmware for the SITL (Software in The Loop) simulation environment.

5. Test the installation: Once the compilation is complete, test the installation by running the following command:

```
sim_vehicle.py -w
```

This command will start the SITL simulator and launch a console window. You can use the console window to test the installation and run your ArduPilot scripts.

7.3 Drone script

Scripting a drone with a Raspberry Pi (RPI) 4 involves writing code to control the drone's flight behavior and integrating it with the drone's flight controller. Here are the steps to script a drone dronekit

Dronekit is a Python library that provides a high-level API for controlling drones and other unmanned vehicles. It is built on top of the MAVLink messaging protocol and provides a simple and intuitive interface for sending commands and receiving telemetry data from the drone's flight controller.

With Dronekit, you can write Python scripts to control the drone's behaviour and execute missions autonomously. Some of the features and functions of Dronekit include:

1. Vehicle control: Dronekit provides a set of functions for controlling the drone's behaviour, such as setting its altitude, speed, and heading. so can use these functions to create autonomous missions or manually control the drone's flight.
2. Telemetry data: Dronekit provides a way to receive real-time telemetry data from the drone's flight controller, such as GPS coordinates, altitude, and battery voltage. You can use this data to monitor the drone's behaviour and make decisions about its flight path and behaviour.
3. Events and call-backs: Dronekit allow you to register event handlers and call-backs to respond to specific events, such as when the drone reaches a waypoint or when its battery level is low.

4. Mission planning: Dronekit provides a set of functions for defining and executing complex missions, such as surveying or mapping. You can use these functions to create custom missions that fit your specific needs.
5. Simulation: Dronekit provides a way to simulate the behaviour of a drone in a virtual environment using SITL (Software in The Loop). This allows you to test your code and mission plans in a safe and controlled environment before deploying them on a real drone.

Dronekit is a powerful and flexible library for controlling drones and other unmanned vehicles. It is widely used in the drone community and has a large and active user community that provides support and resources for getting started with the library.

functions to command our drone with velocities

To command a drone with velocities using Dronekit, you can use the send_velocity function, which allows you to specify the drone's velocity in the North, East, and Down directions. Here is an example of how to use the send_velocity function to command a drone to move forward:

```
from dronekit import connect, VehicleMode
import time
# Connect to the drone at (SITL)
vehicle = connect('udp:127.0.0.1:14550', wait_ready=True)
# Arm the drone and set it to guided mode
vehicle.mode = VehicleMode("GUIDED")
vehicle.armed = True
# Set the drone's velocity
velocity_x = 1.0 # m/s
velocity_y = 0.0 # m/s
velocity_z = 0.0 # m/s
duration = 5 # seconds
# Send the velocity command
vehicle.send_velocity(velocity_x, velocity_y, velocity_z, duration)
# Disarm the drone and close the connection
```

```
vehicle.armed = False
vehicle.close()
```

In this example, we first connect to the drone and set it to guided mode. We then arm the drone and specify its velocity in the send_velocity function. The function takes four arguments: the velocity in the North, East, and Down directions, and the duration of the velocity command.

After sending the velocity command, we wait for the drone to move by sleeping for the duration of the command. Finally, we disarm the drone and close the connection.

7.3.1 Function to waypoint for drone path

To command a drone to follow a path defined by waypoints using Dronekit, this can be applied by using the MissionItem class to define each waypoint and add it to the drone's mission queue.

```
from dronekit import connect, VehicleMode, LocationGlobalRelative, Command
from pymavlink import mavutil
import time
```

Define the waypoints

```
waypoints = [
      LocationGlobalRelative(-35.363261, 149.165230, 20),
      LocationGlobalRelative(-35.362998, 149.165327, 20),
      LocationGlobalRelative(-35.363166, 149.165580, 20)
]
```

Upload the waypoints to the drone

```
cmds = vehicle.commands
cmds.clear()
for i, wp in enumerate(waypoints):
    cmd = Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, wp.lat, wp.lon,
wp.alt)
```

```
    cmds.add(cmd)
cmds.upload()
```

Wait for the drone to execute the mission

```
while vehicle.commands.next:
    time.sleep(1)
```

In this example, we first connect to the drone and set it to guided mode. We then define a set of waypoints as LocationGlobalRelative objects, which specify the latitude, longitude, and altitude of each waypoint.

We then upload the waypoints to the drone's mission queue using the Command class. This class takes several arguments, including the frame of reference for the waypoint (in this case, global relative altitude), the command to be executed at the waypoint (in this case, MAV_CMD_NAV_WAYPOINT), and the latitude, longitude, and altitude of the waypoint.

After uploading the waypoints, we wait for the drone to execute the mission by waiting for the vehicle.commands.next property to become None. Finally, we disarm the drone and close the connection.

Note that this is a simple example and does not include error handling or other safety precautions. When flying a drone, it is important to follow all safety guidelines and regulations and to test your code thoroughly before deploying it in a real-world environment.

So, to apply a similar script need some steps first:

1. Choose a programming language (we used python) then Install the required libraries: Depending on the programming language, then, need to install some libraries or packages to interface with the drone's flight controller. For example, dronekit library using the following command:

   ```
   sudo pip install dronekit
   ```

2. Connect to the drone: Use the dronekit library to establish a connection between the RPI and the drone's flight controller. Using USB cable or a telemetry module to establish the connection.

3. Write the code: Write the code to control the drone's flight behaviour. This can include setting flight parameters such as altitude, speed, and heading, and controlling the drone's movement using waypoints or joystick inputs.

4. Test the code: Once you have written the code, test it on a simulation or test environment to ensure that it is functioning correctly. You can use SITL (Software in The Loop) or a physical drone for testing.

5. Deploy the code: Once testing the code, deploy it to the drone and test it in a real-world environment. Without forgetting follow all safety guidelines and regulations when flying the drone.

By scripting the drone with an RPI, so can customize its behavior and add advanced features that can help to achieve specific goals and solve real-world problems. However, it is important to follow all safety guidelines and regulations when flying the drone, and to test the code thoroughly before deploying it in a real-world environment.

7.3.2 Launch our simulated drone

To launch a simulated drone, you can use SITL (Software In The Loop), which is a powerful tool that allows you to simulate the behavior of a drone without the need for physical hardware. Here are the steps to launch a simulated drone using SITL:

1. Install ArduPilot: Install the ArduPilot software by following the instructions on the ArduPilot website. ArduPilot is a popular flight controller software that can be used to control the behaviour of the simulated drone.

2. Install SITL: Install SITL by running the following command in a terminal window:

```
sudo apt-get install sim_vehicle.py
```

3. Launch SITL: Launch SITL by running the following command in a terminal window:

```
sim_vehicle.py -v ArduCopter
```

This command launches the SITL simulator and starts a virtual drone using the ArduCopter firmware.

4. Connect to the virtual drone: Use a ground control station software such as Mission Planner or QGroundControl to connect to the virtual drone. Then can connect to the virtual drone by selecting the appropriate serial port and baud rate.
5. Test the virtual drone: Once connecting to the virtual drone, so can test its behaviour by sending commands and monitoring its telemetry data. You can use the ground control station software to control the drone's flight behaviour, such as setting waypoints or changing flight modes.

By launching a simulated drone using SITL. This can help to identify and fix bugs and ensure that the code is functioning correctly before deploying it on a real drone.

SITL can help you identify a variety of bugs and errors in your drone software, including:

1. Navigation errors: SITL can simulate GPS data and test your drone's ability to navigate to waypoints accurately. You can use SITL to test for navigation errors such as incorrect heading, altitude, or speed.
2. Control errors: SITL can help you test your drone's stability and control in different flight modes. You can use SITL to test for control errors such as oscillations, overshoot, or instability.
3. Sensor errors: SITL can simulate sensor data and test your drone's ability to respond to changes in the environment. You can use SITL to test for sensor errors such as incorrect readings, noise, or bias.
4. Communication errors: SITL can help to test the drone's ability to communicate with ground control stations and other devices. we can use SITL to test for communication errors such as packet loss, latency, or interference.
5. Performance issues: SITL can help to test the drone's performance under different conditions such as wind, temperature, or humidity. we can use SITL to test

for performance issues such as reduced flight time, decreased speed, or overheating.

7.3.3 Sim_vehicle.py function?

sim_vehicle.py is a command-line tool that is part of the ArduPilot software suite. It is used to launch a simulated drone in a Software in The Loop (SITL) environment for testing and simulation purposes. sim_vehicle.py provides an easy way to start a simulated drone and provides a console interface for interacting with the drone.
When running sim_vehicle.py, it launches a simulated drone using the ArduPilot software and starts a console window that provides a command-line interface for interacting with the drone. The console window provides real-time telemetry data such as GPS coordinates, altitude, and battery voltage, and allows you to send commands to the drone using the MAVLink protocol.

Some of the common options that can be used with sim_vehicle.py include:

- -v: Specifies the vehicle type to simulate. For example, you can simulate a quadcopter using the ArduCopter option.
- -f: Specifies the path to the ArduPilot firmware. This option is useful if you have a custom firmware that you want to test.
- -w: Starts the drone in a GUI window using the SDL library.
- --console: Starts the drone in a console window.

By using sim_vehicle.py, you can easily launch a simulated drone and test your code and flight behavior in a controlled environment without the need for physical hardware. This can save you time and money, and help you identify and fix bugs before deploying your code on a real drone.

7.3.4 Ground control station

Raspberry Pi (RPI) is a popular single-board computer that can be used for a wide range of projects, including drone development. To build a drone with an RPI, you will need to select the appropriate hardware components such as motors, ESCs, propellers,

and a flight controller that is compatible with the RPI. Once you have assembled the hardware, you will need to write software that allows the RPI to control the drone's flight.

To create a ground control station (GCS), you will need a computer with an internet connection and a web browser. By using a laptop, desktop, or even a heart rate monitor phone to act as the GCS. The GCS software can be written in a variety of programming languages including Python, Java, or JavaScript.

To establish communication between the drone and the GCS, you can use a wireless communication protocol such as Wi-Fi or Bluetooth. The RPI can act as a Wi-Fi access point, allowing the GCS to connect to the drone. You can also use a telemetry module to transmit data between the drone and the GCS.

Once you have established communication between the drone and the GCS, you can use the GCS to send commands to the drone and receive telemetry data such as GPS coordinates, altitude, and battery voltage. The GCS can also display a live video feed from the drone's camera, allowing you to monitor the drone's flight in real-time.

There are many open-source projects and tutorials available online that can help you get started with building a drone with an RPI and GCS. Some popular projects include ArduPilot, DroneKit, and PX4.

7.4 How ground control station communicates with Ardupilot software?

The ground control station (GCS) communicates with the ArduPilot software running on the drone's flight controller using a telemetry link. The telemetry link establishes a two-way communication channel between the GCS and the drone, allowing the GCS to send commands to the drone and receive telemetry data such as GPS coordinates, altitude, and battery voltage.

Figure 7- 1 Communication between ground control station and Ardupilot

There are several types of telemetry links that can be used for communication between the GCS and the ArduPilot software, including:

1. Radio telemetry: A radio telemetry link uses a wireless radio transmitter and receiver to transmit data between the GCS and the drone. Radio telemetry links operate on a specific frequency band and can have a range of several kilometers.
2. Wi-Fi telemetry: A Wi-Fi telemetry link uses a wireless Wi-Fi connection to transmit data between the GCS and the drone. The drone's flight controller acts as a Wi-Fi access point, allowing the GCS to connect to the drone.
3. Cellular telemetry: A cellular telemetry link uses a cellular network connection to transmit data between the GCS and the drone. Cellular data plans can be used to establish a reliable and long-range communication link between the GCS and the drone.

Once the telemetry link is established, the GCS can send commands to the ArduPilot software using the MAVLink protocol. MAVLink is a lightweight messaging protocol that is used to exchange data between the GCS and the drone. The ArduPilot software interprets the MAVLink messages and executes the appropriate commands to control the drone's flight.

The ArduPilot software also sends telemetry data back to the GCS using the MAVLink protocol. The GCS can use this data to display real-time information about the drone's flight, such as GPS coordinates, altitude, and battery voltage. The GCS can also use this data to monitor the drone's performance and make decisions about its flight path and behavior.

7.4.1 What are some common MAVLink commands that can be sent to the drone?

MAVLink is a messaging protocol used to communicate with drones and other unmanned vehicles. MAVLink commands are used to send instructions and information to the drone's flight controller. Here are some common MAVLink commands that can be sent to the drone:

1. ARM/DISARM: The ARM command is used to arm the drone's motors, while the DISARM command is used to disarm them. These commands require the drone to be in a specific state, such as landed or disarmed.
2. TAKEOFF/LAND: The TAKEOFF command is used to instruct the drone to take off from its current position, while the LAND command is used to instruct the drone to land at its current position.
3. SET_MODE: The SET_MODE command is used to set the drone's flight mode, such as manual, stabilize, or auto. This command is useful for changing the drone's behavior depending on the situation.
4. SET_PARAMETER: The SET_PARAMETER command is used to set a specific parameter of the drone's flight controller, such as altitude or speed. This command is useful for customizing the drone's behavior to fit specific needs.
5. MISSION_ITEM: The MISSION_ITEM command is used to define a mission item, such as a waypoint or a camera action. This command is used to create a mission that the drone can execute autonomously.
6. REQUEST_DATA_STREAM: The REQUEST_DATA_STREAM command is used to request a specific data stream from the drone's flight controller, such as GPS or attitude information. This command is useful for monitoring the drone's behavior in real-time.

These are just a few examples of the many MAVLink commands that can be sent to the drone. MAVLink commands can be customized and extended depending on the specific needs of the drone and the application.

7.5 Mission Planner

Mission Planner is an open-source software application that can be used as a ground control station (GCS) for unmanned aerial vehicles (UAVs) such as drones. It is compatible with the Raspberry Pi (RPI) and can be used to plan, monitor, and control a drone's mission.

Figure 7- 2 Mission planner

To use Mission Planner with an RPI drone, we will need to install the software on a computer that is connected to the drone's flight controller. The computer can be a laptop, desktop, or even a Raspberry Pi. Once we have installed the Mission Planner software, we can connect to the drone's flight controller using a USB cable or a telemetry module.

Mission Planner provides a user-friendly interface that allows to plan and execute autonomous drone missions. Then can define waypoints, set flight parameters such as altitude, speed, and camera orientation, and monitor the drone's progress in real-time. Mission Planner also provides features such as live video feed, telemetry data, and flight logs that can help to analyze the drone's performance.

To use Mission Planner with an RPI drone, we will need to ensure that the drone's flight controller is compatible with the software. Most popular flight controllers such as the Pixhawk and ArduPilot are compatible with Mission Planner. we will also need to ensure that the RPI is compatible with the telemetry module used for communication between the drone and the GCS.

Overall, Mission Planner is an excellent software application for planning and executing autonomous drone missions. It provides a user-friendly interface and a wide range of features.

7.6 Connection between pixhawk and Raspberry pi 4

The Pixhawk is a popular open-source flight controller that can be used to control the flight of a drone. It is compatible with the Raspberry Pi (RPI) and can be used to build a powerful and customizable drone.

To use the Pixhawk with an RPI 4, you will need to connect the Pixhawk to the RPI using a serial connection. The Pixhawk has a serial port that can be used to communicate with the RPI, and the RPI has several serial ports that can be used for this purpose.

Here are the steps to connect the Pixhawk to the RPI 4:

1. Connect the Pixhawk to the RPI using a serial cable. The serial cable should be connected to the Pixhawk's TELEM1 or TELEM2 port and the RPI's serial port. You can use a USB-to-serial adapter if your RPI does not have a serial port.
2. Install MAVProxy on your RPI. MAVProxy is a ground control station software that can be used to communicate with the Pixhawk. You can install MAVProxy using the following command:

```
sudo apt-get install python-mavproxy
```

3. Launch MAVProxy by running the following command:

```
mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --aircraft MyCopter
```

Replace **/dev/ttyAMA0** with the **serial port** connected to the Pixhawk and MyCopter with the name of your aircraft.

4. Connect to the Pixhawk using a ground control station software such as Mission Planner or QGroundControl. To connect the Pixhawk should select the appropriate serial port and baud rate.

Once connecting the Pixhawk to the RPI and launched MAVProxy, you can use your ground control station software to control the drone's flight. The RPI can also be used to execute scripts and programs that can perform advanced functions such as computer vision or machine learning.

7.7 Interfacing pi camera for QR code landing

Using a Raspberry Pi camera with a drone can be a useful way to capture images and data during flight. In order to do this, you'll need to connect the camera to the Raspberry Pi and use its

GPIO pins to communicate with the drone's flight controller. Here are the high-level steps you'll need to follow:

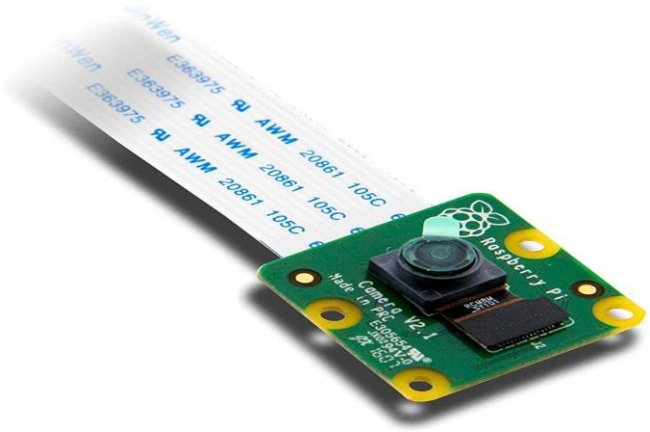


Figure 7- 3 pi camera

1. Obtaining a Raspberry Pi camera module version 2.1:
   The Raspberry Pi camera module version 2.1 is a high-quality camera that is specifically designed for use with the Raspberry Pi. It includes a 8MP sensor and can capture video at 1080p at 30fps, or 720p at 60fps. You can purchase the camera module from various online retailers.

2. Enabling the camera module in the Raspberry Pi configurations:
   To use the camera module, you'll need to enable it in the Raspberry Pi configurations. This can be done by running the command "sudo raspi-config" in the terminal, selecting "Interfacing Options", selecting "Camera", and then selecting "Enable". This will enable the camera module and allow you to use it with your drone.

3. Cloning the necessary algorithms:
   The distance estimation, QR code detection and recognition algorithms that you'll need to use can be found online, and can be easily cloned using Git. Once you've cloned the necessary code, you'll need to install any dependencies and configure the code to work with your specific setup.

4. Installing an X server:
   If you're using a Windows machine to connect to your Raspberry Pi, you'll need to install

an X server in order to enable SSH. Xming is a popular X server that can be used for this purpose.

5. Enabling X11 on Putty:
   Once you've installed an X server, you'll need to enable X11 forwarding on Putty before beginning the session. This can be done by opening Putty, selecting your session, clicking on "SSH", selecting "X11", and then checking the box next to "Enable X11 forwarding".

6. Setting the video capture to "0":
   In order to use the Raspberry Pi camera module with your drone, you'll need to set the video capture to "0" in the distance estimation code. This tells the code to use the camera module instead of an external camera.

7. Setting the focal length and camera resolution:
   Finally, you'll need to set the focal length for the camera module to 3.04mm and the camera resolution to 640 x 480 pixels. This will ensure that the camera is configured correctly and that you're capturing high-quality images and data during flight.

7.8 Connecting other project parts

7.8.1 Connect watch using adafruit io server

This part aim to continuously monitor a patient's heart rate and, in the event of a cardiac arrest, launch a drone immediately to the patient's location and send an email with the patient's location coordinates of a specific values of longitude and latitude on server feeds. To accomplish this, we are using several components, including a watch to monitor the patient's heart rate, a drone to respond to emergencies, and Adafruit IO for data storage and communication between components.

The first step in our system is to continuously monitor the patient's heart rate using a heart rate monitor connected to a watch. The watch will send heart rate data to Adafruit IO in real-time, allowing us to monitor the patient's heart rate remotely.

In the event of a cardiac arrest, our system will launch a drone to the patient's location. The drone will be equipped with GPS and will use the patient's location coordinates, which are also sent to Adafruit IO in real-time via two separate feeds for longitude and

latitude, as goal points for its mission. To calculate the shortest path to the patient's location, that used in a shortest path algorithm.

In addition to launching the drone, our system will also send an email with the patient's location coordinates to a server feed. This will allow emergency services to quickly locate the patient and provide assistance.

So, what is adafruit? and the way to use

Adafruit IO is a key component of our system, as it provides a platform for data storage and communication between components. We are subscribing to three separate feeds in Adafruit IO: one for heart rate data, and two for longitude and latitude data. By subscribing to these feeds, we can receive real-time updates on the patient's condition and location, and use this information to launch the drone and send the email with the patient's location coordinates.

To use the Adafruit IO platform with a Raspberry Pi, by using the Adafruit IO Python library to send and receive data from the platform. Here are the high-level steps to use the Adafruit IO platform with a Raspberry Pi:

1. Create an Adafruit IO account: Create an account on the Adafruit IO platform by visiting the Adafruit IO website and following the instructions to create an account.
2. Set up the Raspberry Pi: Set up the Raspberry Pi by installing the necessary software and libraries, such as Python 3 and the Adafruit IO Python library.
3. Connect to the Adafruit IO platform: Use the Adafruit IO Python library to connect to the Adafruit IO platform using your account credentials. This allowing to send and receive data from the platform.

```
from dronekit import connect, VehicleMode, LocationGlobalRelative,
Command
from pymavlink import mavutil
from Adafruit_IO import Client
```

4. Retrieve the latest values: Use the Adafruit IO Python library to retrieve the latest values from the latitude and longitude feeds. You can do this by using the receive method to retrieve the latest value from each feed.

```
client.subscribe('loc-lng')
client.subscribe('loc-lat')
client.subscribe('drone-deploy')
```

5. Create waypoints: Use the latitude and longitude values to create a set of waypoints for the drone to follow. You can do this by creating a list of LocationGlobalRelative objects, which specify the latitude, longitude, and altitude of each waypoint.
6. Upload the waypoints: Use the Dronekit library to upload the waypoints to the drone's mission queue. You can do this by creating a MissionItem object for each waypoint and adding it to the mission queue using the add method.
7. Execute the mission: Use the Dronekit library to execute the mission by setting the drone to guided mode and arming it. The drone will then follow the waypoints in the mission queue.

Overall, our system is designed to provide a rapid response to cardiac arrest cases by using a drone to quickly reach the patient's location and administer first aid. By integrating a heart rate monitor, a drone, and Adafruit IO, we are able to monitor the patient's condition and location in real-time and respond to emergencies quickly and effectively.

In this example, we first connect to the drone and set it to guided mode. We then connect to Adafruit IO using the Adafruit IO Python library and retrieve the latest values from the latitude and longitude feeds.

7.8.2 Connect CPR using Bluetooth

In the event that the drone reaches the patient and confirms that the patient is suffering from a cardiac arrest, our system will automatically initiate CPR. To do this, we will send a message to a CPR device after the drone lands at the patient's location. The

CPR device will then begin its function of administering chest compressions until emergency services arrive.

To ensure that the CPR device only starts its function, when necessary, we will integrate Bluetooth calibration into our system. This will allow us to detect changes in the patient's heart rate and determine whether CPR is necessary.

To calibrate Bluetooth on the Raspberry Pi, we will need to follow several steps:

1. Enable Bluetooth on the Raspberry Pi by running the command.

   ```
   sudo systemctl enable Bluetooth
   ```
2. Install the necessary Bluetooth packages by running the command.

   ```
   sudo apt-get install bluetooth bluez libbluetooth-dev
   ```
3. Pair the Raspberry Pi with the CPR device by running the command and following the prompts.

   ```
   sudo bluetoothctl
   ```
4. Test the Bluetooth connection by running the command.

   ```
   sudo rfcomm connect hci0 [Bluetooth device address]
   ```

Once Bluetooth calibration is set up, we can integrate it into our system to ensure that the CPR device only starts its function when necessary. This will involve continuously monitoring the patient's heart rate and sending a message to the CPR device if the heart rate drops below a certain threshold. If the heart rate returns to normal, we will send another message to the CPR device to stop its function.

To integrate Bluetooth calibration into our system, we will need to follow several steps:

1. Set up a Bluetooth connection between the Raspberry Pi and the CPR device as described above.
2. To continuously monitor the patient's heart rate, send a message to the CPR device if the heart rate drops below a certain threshold.
3. Then send another message to the CPR device if the patient's heart rate returns to normal, indicating that the CPR device should stop its function.

4. Integrate these scripts into the Drone_contol script, so that they are activated once the drone lands at the patient's location and confirms that the patient is suffering from a cardiac arrest.

By integrating Bluetooth calibration into our system, we can ensure that the CPR device only starts its function, when necessary, thereby conserving battery life and reducing the risk of injury to the patient. By continuously monitoring the patient's heart rate and sending messages to the CPR device, when necessary, we can provide a rapid and effective response to cardiac arrest cases.

Chapter 8 Drone hardware components

Chapter 8

Drone hardware components

8.1 Pixhawk.

8.1.1 Introduction to pixhawk

The Pixhawk 2.4.8 is an open-source flight controller designed for autonomous vehicles such as drones, ground vehicles, and underwater vehicles. It is based on the Pixhawk FMUv3 architecture and is capable of running multiple flight control algorithms.

With its support for GPS navigation, telemetry communication, and a range of sensors and peripherals, the Pixhawk 2.4.8 is a versatile flight controller that is used in a variety of applications, including surveying, mapping, inspection, and search and rescue.

One of the main benefits of using the Pixhawk 2.4.8 is its open-source nature, which allows for community-driven development and support. The Pixhawk 2.4.8 is well-documented and has a large community of users who share resources and provide support for each other.

Compared to other popular flight controllers, such as the Ardupilot or Betaflight, the Pixhawk 2.4.8 offers a unique set of features and capabilities. While it may not be suitable for very large or complex autonomous vehicles, it offers a powerful and flexible platform for a wide range of applications.

In this book, we will explore how to connect and configure various peripherals with the Pixhawk 2.4.8, including the Raspberry Pi 4, GPS, Telemetry, BLDC Motors, ESCs, Compass, Power module, and Battery. Wewill also cover how to configure and calibrate the Pixhawk 2.4.8 and its peripherals, as well as troubleshooting tips for common issues.

8.1.2 Pixhawk 2.4.8 Hardware:

The Pixhawk 2.4.8 flight controller is a small, square board with a number of connectors and components. It measures approximately 50 x 81 x 15 mm and weighs around 40 grams. Here are some of the key components and connectors on the Pixhawk 2.4.8:

Figure 8- 1 Pixhawk

- Processor: The Pixhawk 2.4.8 is powered by a 32-bit ARM Cortex-M4F processor running at 168 MHz. This provides plenty of processing power for running flight control algorithms and communicating with peripherals.
- Sensors: The Pixhawk 2.4.8 includes a range of sensors for measuring orientation, acceleration, and other metrics. These sensors include an Inertial Measurement Unit (IMU), which consists of a gyroscope, an accelerometer, and a magnetometer, as well as a barometer.
- Connectors: The Pixhawk 2.4.8 has a number of connectors for connecting peripherals and other devices. These include:
    - Power input: The Pixhawk 2.4.8 can be powered by a 5V DC input or a 2S-6S LiPo battery.
    - ESC connectors: The Pixhawk 2.4.8 has eight connectors for connecting Electronic Speed Controllers (ESCs) for controlling Brushless DC (BLDC) motors.
    - Servo connectors: The Pixhawk 2.4.8 has six connectors for connecting servos for controlling other components, such as camera gimbals.
    - Telemetry radio connectors: The Pixhawk 2.4.8 has two connectors for connecting telemetry radios for communication with a ground station.
    - I2C connector: The Pixhawk 2.4.8 has one connector for connecting I2C devices, such as compass modules or other sensors.

  - SPI connector: The Pixhawk 2.4.8 has one connector for connecting SPI devices, such as external flash memory or other sensors.
- LEDs: The Pixhawk 2.4.8 has several LEDs for indicating status and errors. These include LEDs for power and status.
- SD card slot: The Pixhawk 2.4.8 has an SD card slot for storing data logs and other files.

Overall, the Pixhawk 2.4.8 hardware provides a powerful and flexible platform for building autonomous vehicles. Its range of sensors and connectors allows for a wide range of applications and configurations, while its compact form factor and lightweight design make it suitable for use in a variety of environments.

8.1.3 Connecting Peripherals to Pixhawk 2.4.8:

The Pixhawk 2.4.8 flight controller is designed to work with a wide range of peripherals and sensors. In this section of the book, we will explore how to connect and configure various peripherals with the Pixhawk 2.4.8 to enable autonomous control of vehicles.

Here are some of the peripherals:

1. Raspberry Pi 4: The Raspberry Pi 4 is a popular single-board computer that can be used in conjunction with the Pixhawk 2.4.8 to enable advanced computation, image processing, and communication capabilities. We will cover how to connect the Raspberry Pi 4 to the Pixhawk 2.4.8 using the serial interface and how to configure the communication between the two devices.
2. GPS: The Pixhawk 2.4.8 supports a range of GPS modules for navigation and position tracking. We will cover how to connect a GPS module to the Pixhawk 2.4.8 and how to configure the GPS module to work with the flight controller.
3. Telemetry: The Pixhawk 2.4.8 supports telemetry radios for communication with a ground station. We will cover how to connect telemetry radios to the Pixhawk 2.4.8 and how to configure the communication between the two devices.
4. Brushless DC (BLDC) Motors and ElectronicSpeed Controllers (ESCs): The Pixhawk 2.4.8 can control up to eight BLDC motors through the ESCs. We will

cover how to connect the motors and ESCs to the Pixhawk 2.4.8 and how to configure the ESCs for optimal motor control.

5. Compass: The Pixhawk 2.4.8 has a built-in magnetometer, but an external compass module can be used for improved accuracy. We will cover how to connect an external compass module to the Pixhawk 2.4.8 and how to calibrate the compass for accurate orientation measurements.
6. Power module and Battery: The Pixhawk 2.4.8 requires a power source to operate, and a power module can be used to measure the voltage and current of the battery. We will cover how to connect a power module to the Pixhawk 2.4.8 and how to configure the voltage and current sensing.

For each peripheral, we will provide step-by-step instructions on how to connect and configure it with the Pixhawk 2.4.8. We will also cover troubleshooting tips for common issues that may arise during the connection and configuration process.

By the end of this section, you will have a comprehensive understanding of how to connect and configure various peripherals with the Pixhawk 2.4.8, enabling you to build and customize your autonomous vehicle to meet your specific needs and requirements.

8.2 Connecting Raspberry Pi to Pixhawk 2.4.8:

The Raspberry Pi is a powerful single-board computer that can be used in conjunction with the Pixhawk 2.4.8 to enable advanced computation, image processing, and communication capabilities. Here are the steps to connect the Raspberry Pi to the Pixhawk 2.4.8:

1. Obtain the necessary hardware: You will need a Raspberry Pi 4 and a 4-pin cable.
2. Connect the 4-pin cable to the Raspberry Pi and the Pixhawk 2.4.8: Connect one end of the 6-pin cable to the GPIO pins on the Raspberry Pi (pins 4,6,27 and 28), and the other end of the cable to the Pixhawk 2.4.8's TELEM2 port. The wiring order should be as follows: Pin 4 to 5V, Pin 27 to RX, Pin 28 to TX, and Pin 6 to GND.
3. Configure the Raspberry Pi: Once the hardware is connected, you will need to configure the Raspberry Pi to communicate with the Pixhawk 2.4.8 through the serial port (ttyAMA1) on GPIO pins. You can do this by disabling the serial console on the Raspberry Pi and configuring the serial port to use GPIO pins.

4. Test the connection: To test the connection between the Raspberry Pi and the Pixhawk 2.4.8, you can use the MAVProxy software to connect to the flight controller and send commands to control the vehicle.

Figure

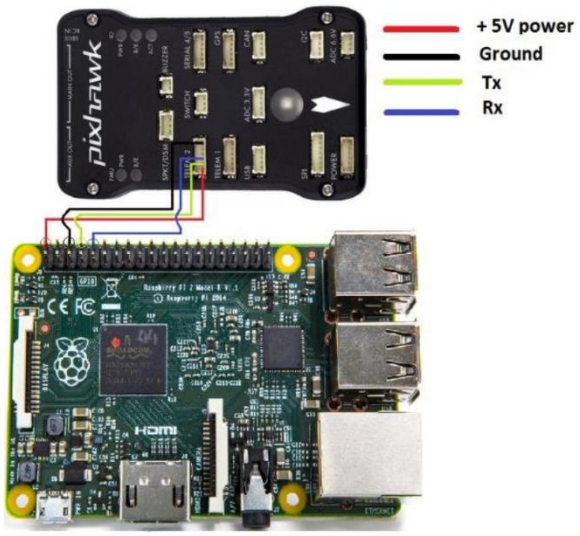


Figure 8- 2 Connection between pixhawk and raspberry pi

By connecting the Raspberry Pi to the Pixhawk 2.4.8 through the serial port (ttyAMA1) on GPIO pins via the TELEM2 port, you can take advantage of the Raspberry Pi's processing power and advanced computing capabilities to perform tasks such as image processing and machine learning. This opens up a wide range of possibilities for autonomous vehicle applications, from object detection and tracking to path planning and decision-making.

Overall, the Pixhawk 2.4.8 and Raspberry Pi combination provides a powerful and flexible platform for building autonomous vehicles with advanced capabilities and customizability.

8.3 Brushless motor and ESC

8.3.1 Introduction to brushless motor

Brushless motor with a 1000KV rating is commonly used in drones and RC applications. Here are some key specifications:

- KV Rating: 1000KV means the motor rotates at approximately 1000 RPM per volt applied.

- Speed and Torque: It offers a balance between speed and torque.
- Voltage Range: Typically used with 2S to 4S LiPo batteries (7.4V to 14.8V).

Figure 8- 3 Brushless motor

Here are some key reasons why brushless motors are preferred in drones:

1. Efficiency: Brushless motors are highly efficient compared to brushed motors. They convert electrical energy into mechanical energy with minimal energy losses, resulting in longer flight times and improved battery life for drones. The efficient operation allows drones to fly for extended periods while conserving energy.
2. Power-to-Weight Ratio: BLDC motors offer a high power-to-weight ratio, making them ideal for drones. Their compact size and lightweight construction allow drones to generate significant thrust while keeping the overall weight of the aircraft low. This characteristic enhances maneuverability, agility, and overall flight performance.
3. High Torque and Speed Control: Brushless motors deliver high torque output, which is essential for drones to achieve the required lift and control during takeoff, flight, and landing. They offer precise speed control, enabling drones to adjust their propulsion system for various flight conditions, including maintaining stability, ascending, descending, or maneuvering at different speeds.
4. Durability and Low Maintenance: The absence of brushes in brushless motors eliminates mechanical friction and wear, resulting in increased durability and

reduced maintenance requirements. Brushless motors have a longer lifespan compared to brushed motors, making them more reliable and cost-effective in the long run.

5. Enhanced Control and Stability: Brushless motors offer better control and stability for drones. They respond quickly to changes in speed and direction, allowing for precise adjustments and responsive flight maneuvers. This control enables drones to maintain stability, hover accurately, and perform complex aerial tasks with ease.

6. Reduced Noise and Vibration: Brushless motors operate with reduced noise and vibration compared to brushed motors. The elimination of physical brushes and commutation mechanisms contributes to quieter operation, making brushless motors suitable for applications where noise reduction is important, such as aerial photography and videography.

7. High RPM Capability: BLDC motors can achieve high rotational speeds, allowing drones to generate the necessary thrust for fast acceleration and agile maneuvers. This capability enables drones to perform aerobatic stunts, fly at high speeds, and respond quickly to pilot commands.

8. Electronic Control Compatibility: Brushless motors work seamlessly with electronic speed controllers (ESCs) and flight control systems. They can be easily integrated and controlled using digital signals, enabling precise and synchronized motor control for optimized flight performance.

8.3.2 Introduction to ESC and its usage in drone

The Electronic Speed Controller (ESC) is a crucial component in a drone's propulsion system.

Figure 8- 4 ESC

It serves the following primary purposes:

1. Motor Control: The ESC is responsible for controlling the speed and direction of the brushless motors in a drone. It receives control signals from the flight controller and converts them into appropriate electrical signals to adjust the motor's rotational speed. By varying the voltage and frequency of these signals, the ESC regulates the motor's RPM (Rotations Per Minute) and hence controls the thrust generated by the propellers.

2. Brushless Motor Compatibility: Brushless motors require electronic commutation, which is handled by the ESC. The ESC sends electric pulses to different motor windings at precise intervals, ensuring that the motor rotates smoothly and efficiently. This eliminates the need for physical brushes and commutation mechanisms used in brushed motors, making brushless motors more reliable and durable.

3. Electronic Speed Stabilization: The ESC plays a vital role in stabilizing the rotational speed of the motors. It actively monitors the motor's RPM and adjusts the electrical signals accordingly to maintain a consistent speed. This stabilization contributes to a smoother and more stable flight experience, allowing the drone to hover steadily and perform controlled maneuvers.

4. Throttle Response and Motor Protection: The ESC ensures rapid and precise response to throttle commands from the flight controller. It translates the pilot's inputs or autonomous control signals into corresponding changes in motor speed, enabling quick acceleration or deceleration. Additionally, the ESC incorporates

various safety features to protect the motor and electronics, such as overcurrent and overtemperature protection, preventing damage in case of abnormal operating conditions.

5. Calibration and Configuration: ESCs often require calibration to establish the minimum and maximum throttle values and to synchronize with the flight controller. This calibration process ensures accurate and consistent motor control. Additionally, ESCs may offer configuration options, such as motor timing, braking settings, and motor direction reversal, allowing users to fine-tune the motor's behavior based on specific requirements.

8.3.3 Connection between brushless motor and ESC with Pixhawk

The connection process typically involves the following steps:

1. Power Supply: Connect the power source, usually a battery, to the ESC to provide electrical power to the system. The ESC is typically powered directly from the battery.

2. Motor Connections: Connect the three motor wires from the brushless motor to the corresponding three motor output ports on the ESC. Ensure that the wires are connected in the correct order (A, B, and C) to ensure proper motor rotation direction. The ESC will have labels indicating the correct wiring configuration.

3. ESC Connections: Connect the signal wire from the ESC to the appropriate motor output port on the Pixhawk flight controller. The Pixhawk flight controller provides PWM (Pulse Width Modulation) signals to the ESC to control the motor's speed and direction.

4. Calibration: Perform the necessary calibration procedures for the ESC as per the manufacturer's instructions. This calibration ensures that the ESC recognizes the minimum and maximum throttle values accurately.

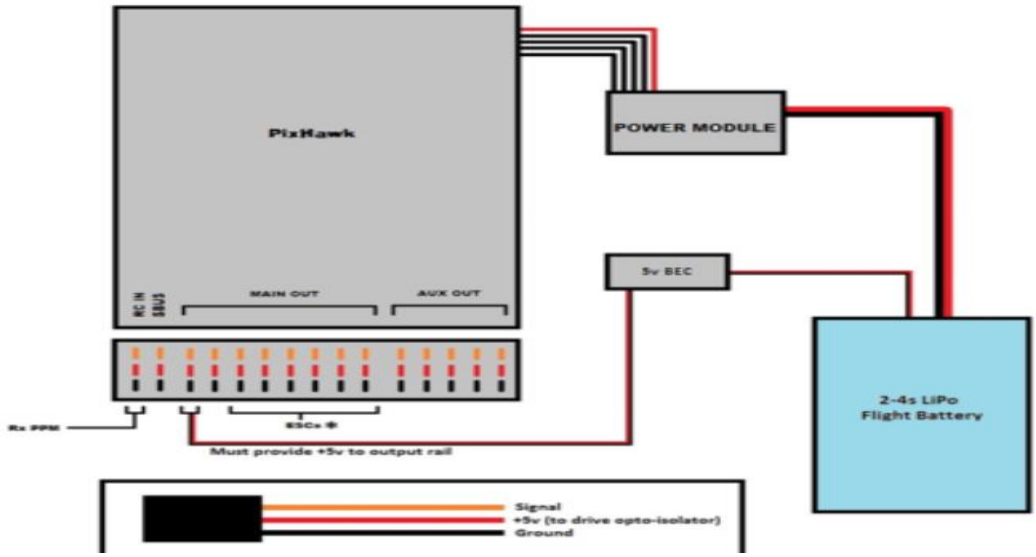


Figure 8- 5 Connection between brushless motor and ESC with Pixhawk

8.4 Power module

A power module in a drone is an essential component that helps monitor and regulate the power supply to the flight controller and other electronic components. It typically includes a voltage regulator and current sensor to provide accurate power measurements and protection. The power module is connected to the drone's power distribution system and the flight controller.

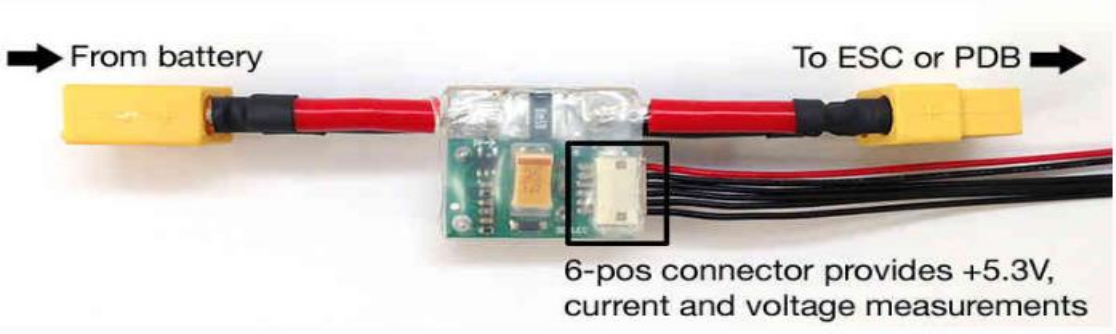


Figure 8- 6 power module

8.4.1 Power module components

- Voltage Regulator: The voltage regulator converts the input voltage from the drone's battery to a stable and regulated voltage suitable for powering the flight controller and other onboard electronics. It ensures a consistent power supply to prevent voltage fluctuations that could affect the performance of the drone's components.

- Current Sensor: The current sensor measures the current flowing through the power module, providing information on the drone's power consumption. This data is often used for flight logging, monitoring battery usage, and implementing safety features like low battery warnings.
- Connectors: The power module has input and output connectors to connect to the drone's power distribution system and the flight controller. The input connector is usually connected to the drone's battery, while the output connector is connected to the flight controller.

8.4.2 Power module connection.

- Battery Connection: Connect the drone's battery to the input connector of the power module. Ensure that the positive (+) and negative (-) terminals of the battery match the corresponding terminals on the power module.
- Power Distribution System: Connect the power module's output connector to the appropriate power input port on the flight controller or power distribution board. The specific connection points may vary depending on the drone's setup and flight controller used.
- Data Connection (Optional): Some power modules provide a data connection that allows the flight controller to receive voltage and current measurements from the power module. This data can be used for monitoring battery levels, estimating flight time, and triggering low battery warnings. If available, connect the data line of the power module to the designated data input port on the flight controller.
- Calibration and Configuration: Depending on the power module and flight controller used, you may need to perform calibration and configuration steps to ensure accurate voltage and current measurements. Follow the instructions provided by the power module manufacturer and flight controller documentation for the specific calibration process.

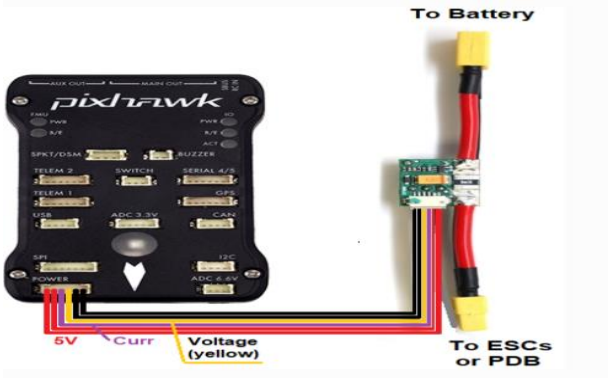


Figure 8- 7 connection between pixhawk and power module

8.5 GPS

8.5.1 Introduction

Having GPS on a drone makes a big difference in how drones perform and enables sophisticated and powerful drones that can perform a wide range of tasks with accuracy and reliability. GPS on a drone is essential for accurate and reliable autonomous flight. GPS provides critical position data that enables precise navigation to a specific location and allows the drone to perform complex tasks with ease. GPS also enables real-time tracking of the drone's location and flight path, providing valuable information for monitoring the drone's position during complex missions or in challenging environments. Additionally, the use of Real-Time Kinematics (RTK) improves GPS accuracy, which is important for precise control of the drone during flight.

8.5.2 GPS usage

The GPS module allows us to know the location relative to a network of orbiting satellites. Connecting to signals from these satellites allows the drone to perform functions such as position hold, autonomous flight, return to home, and waypoint navigation. Here are some ways that GPS can improve drone performance:

1. Accurate navigation: GPS provides accurate position data that enables precise navigation to a specific location. This is critical for autonomous flight, where the drone must navigate to a specific waypoint or follow a pre-defined flight path.

2. Autonomous flight modes: GPS enables autonomous flight modes, such as GPS lock, GPS hold, GPS waypoint navigation, and Return to Home (RTH). These modes rely on GPS data to enable accurate and reliable autonomous flight.
3. Real-time tracking: GPS data enables real-time tracking of the drone's location and flight path. This is particularly useful for monitoring the drone's position during complex missions or in challenging environments, such as urban areas or mountainous terrain.
4. Improved accuracy: Pixhawk uses a technique called Real-Time Kinematics (RTK) to improve GPS accuracy. RTK uses a base station to provide correctional data to the GPS module, resulting in much more accurate position data.
5. Telemetry data transmission: GPS data enables telemetry data transmission, which provides real-time information about the drone's position, altitude, speed, and battery life to the ground control station. This information is critical for monitoring the drone's flight path and ensuring safe and successful flights.

8.5.3 Connecting the NEO M8 GPS Module:

To connect GPS NEO-M8 with Pixhawk 2.4.8, you need to connect the GPS module to the Pixhawk's GPS port using a 6-pin JST-GH cable. Once connected, the Pixhawk will automatically detect the GPS module and start using its position data to enable GPS-based flight modes, such as Return to Home (RTH) and GPS waypoint navigation. The GPS module can be connected to the Pixhawk using a 6-pin JST-GH cable. The cable has a locking mechanism to prevent it from coming loose during flight. Connect the cable to the GPS port on the Pixhawk and the appropriate port on the GPS module.

8.5.4 Configuring a GPS with Pixhawk

Configuring a GPS with Pixhawk using Mission Planner software is a critical step to ensure that the GPS module is working properly and providing accurate position data for autonomous flight. It could be done by selecting Standard Param, select the appropriate communication protocol for your GPS module (usually "Auto" or "UBLOX"), adjust the GPS update rate and accuracy settings as needed.

8.6 Compass

8.6.1 Introduction

Compass is an important peripheral used with Pixhawk, as it helps the drone to maintain its heading and orientation. An external compass provides additional redundancy and accuracy to the Pixhawk's internal compass. It can also help reduce interference from other electronics on the drone, resulting in more accurate position data and better performance.

8.6.2 Connect the compass to the Pixhawk

- Connect the compass module to the Pixhawk's I2C port using a 4-wire cable. The 4-wire cable usually consists of a red wire for power, a black wire for ground, and two other wires for the I2C communication.
- Magnetic interference from other electronics on the drone, such as power cables or motors, can cause compass issues. So Compass calibration should be performed in a location that is free from sources of magnetic interference

8.6.3 Compass issues and troubleshoot

Compass issues can cause significant problems with a drone's navigation and flight performance, so it's essential to understand the common issues and how to troubleshoot them.

1. Magnetic interference: Magnetic interference from other electronics on the drone, such as power cables or motors, can cause compass issues. The drone's frame and other metal components can also cause magnetic interference.
2. Mounting issues: The compass must be mounted in a location that provides an unobstructed view of the sky and is away from sources of interference. Ensure that the compass is mounted correctly and that it is pointing in the same direction as the Pixhawk's arrow.
3. Calibration issues: Calibration is an essential step in ensuring accurate and reliable compass readings. If the compass is not calibrated correctly, it may provide inaccurate readings.

4. Hardware issues: The compass module or the Pixhawk flight controller may be damaged or faulty, which can cause compass issues.
5. Firmware issues: If the firmware for the Pixhawk flight controller or the compass module is outdated or incompatible, it can cause compass issues.
6. Power supply issues: The compass module must receive a stable power supply. Voltage fluctuations can cause compass issues.
7. Location issues: Magnetic interference can be caused by nearby buildings, power lines, or other sources of electromagnetic radiation. Testing the drone in a different location can help identify whether the issue is location-specific.

To troubleshoot compass issues, it's essential to check the calibration, orientation, sources of interference, firmware updates, power supply, and hardware issues. ensure that the compass is functioning correctly, as inaccurate compass readings can cause the drone to fly off course or lose stability in flight.

8.7 Telemetry

8.7.1 Introduction to Telemetry

Telemetry refers to the remote measurement and transmission of data from a drone to a ground station or other devices. It plays a crucial role in drone operations by providing real-time information about the aircraft's flight parameters and status.

Figure 8- 8 Telemetry

8.7.2 Function of Telemetry in drone.

1. Data Monitoring: Telemetry allows for the monitoring of various flight parameters and sensor data in real-time. This includes information such as altitude, GPS position, speed, battery voltage, motor RPM, and attitude (pitch, roll, and yaw). Monitoring these data points helps pilots and operators to have a comprehensive understanding of the drone's performance and status during flight.
2. Flight Control and Navigation: Telemetry enables two-way communication between the drone and the ground station, allowing for remote control and navigation of the aircraft. Commands from the ground station, such as adjusting flight modes, changing waypoints, or initiating automated actions, can be sent to the drone via telemetry links, providing control and guidance during the flight.
3. Safety and Security: Telemetry data plays a vital role in ensuring the safety and security of the drone operation. Real-time updates on battery voltage and remaining flight time help pilots to manage the drone's power resources effectively, preventing unexpected power failures. Additionally, telemetry can provide alerts or warnings for critical events like high winds, GPS signal loss, or low signal strength, enabling timely decision-making to ensure safe flight operations.
4. Flight Planning and Analysis: Telemetry data is valuable for flight planning and post-flight analysis. By logging and recording telemetry data, operators can analyse flight performance, review flight paths, and evaluate the drone's behaviour. This information aids in optimizing flight routes, identifying areas for improvement, and assessing mission success.
5. Remote Diagnostics and Maintenance: Telemetry allows for remote diagnostics and troubleshooting of the drone's systems. By analysing telemetry data, operators can identify potential issues or anomalies in real-time, such as motor or sensor malfunctions. This helps in diagnosing problems quickly, implementing corrective actions, and reducing downtime for maintenance.

6. Telemetry Range and Link Quality: Telemetry systems typically utilize radio frequency (RF) communication links, such as Wi-Fi, Bluetooth, or dedicated RF modules. The range and link quality of the telemetry system are important factors to consider for reliable and uninterrupted communication between the drone and ground station.

8.7.3 Connection of Telemetry with drone

The connection of telemetry with a drone involves establishing a communication link between the drone and a ground station or control device to transmit and receive data. Here's an overview of how telemetry is connected to a drone:

1. Telemetry Transmitter: The drone is equipped with a telemetry transmitter, which is a module or component responsible for collecting data from various sensors and systems on the drone. This transmitter converts the data into a format suitable for transmission.

2. Telemetry Receiver: The ground station or control device, such as a computer or handheld device, is equipped with a telemetry receiver. This receiver is responsible for receiving the transmitted telemetry data from the drone.

3. Communication Link: The telemetry system utilizes a communication link to establish the connection between the drone and the ground station. Common communication methods include radio frequency (RF) technologies like Wi-Fi, Bluetooth, or dedicated RF modules. These links can be wireless or wired, depending on the specific telemetry system used.

4. Telemetry Protocol: Telemetry systems use specific protocols to transmit and interpret the data. Common protocols include MAVLink (Micro Air Vehicle Link) and FrSky telemetry protocol, among others. These protocols define the structure, format, and encoding of the telemetry data.

5. Data Transmission: The telemetry transmitter on the drone continuously collects data from various sensors, flight controllers, and other onboard systems. It then packages the data according to the telemetry protocol and transmits it via the communication link.

6. Data Reception and Display: The telemetry receiver on the ground station or control device receives the transmitted data from the drone. The data is then decoded and displayed on the ground station's user interface or telemetry software, providing real-time information about the drone's flight parameters and status.

7. Two-Way Communication: Telemetry systems often support two-way communication, allowing commands and instructions to be sent from the ground station to the drone. This enables remote control functionalities, such as changing flight modes, adjusting parameters, or initiating automated actions.

8.8 Battery:

Selecting and connecting a suitable battery is crucial for safe and efficient operation of the Pixhawk 2.4.8 and its peripherals. Here are some guidelines to follow when selecting and connecting a battery:

1. Battery voltage: The Pixhawk 2.4.8 requires a battery with a voltage between 10V and 50V. It's important to choose a battery with a voltage that is within this range. If the battery voltage is too low,

Figure 8- 9 battery

The Pixhawk may not receive enough power to operate. If the battery voltage is too high, it may damage the Pixhawk and its peripherals.

2. Capacity: The capacity of the battery determines how long the drone can fly before the battery needs to be recharged. The capacity is usually measured in milliampere-hours

(mAh). The capacity required depends on the weight of the drone, the size of the propellers, and the flight time required. It's important to choose a battery with a capacity that is sufficient for your needs.

3. Connector type: The battery should have a connector that is compatible with the power module and the Pixhawk. The most common connector types are XT60 and Dean's T-plug. Ensure that the connector is securely fastened to the power module and the Pixhawk to prevent it from disconnecting during flight.
4. Wiring: When connecting the battery to the power module, ensure that the wiring is correct and secure. Connect the red wire from the battery to the positive (+) terminal of the power module, and the black wire from the battery to the negative (-) terminal of the power module. A properly connected battery will provide power to the Pixhawk and its peripherals through the power module.
5. Battery safety: Always follow the manufacturer's instructions for handling and charging the battery. Use a fireproof bag when charging the battery to reduce the risk of fire. Inspect the battery regularly for signs of damage or swelling, and dispose of it properly if it becomes damaged.

For our setup, we used a 3300mAh 3-cell 11.1V battery with a T-plug connector. This battery provided a suitable voltage and capacity for our drone, and the T-plug connector was compatible with our power module and Pixhawk.

By selecting and connecting a suitable battery to the power module of the Pixhawk, you can ensure that your drone has a reliable power source that is compatible with the Pixhawk 2.4.8 and its peripherals. A properly chosen and calibrated battery can help you achieve safe and efficient flights with the Pixhawk. Additionally, following battery safety guidelines can help minimize the risk of accidents and ensure that your equipment lasts for a long time.

Overall, battery selection and connection to the power module of the Pixhawk is an important aspect of setting up the Pixhawk 2.4.8 for flight. By following these guidelines and considering the specific requirements of your drone, you can choose a battery that is suitable for your needs and ensure that it is connected safely and correctly to the power module of the Pixhawk for reliable and stable operation.

8.9 Configuration and Calibration of Pixhawk 2.4.8:

Configuring and calibrating the Pixhawk 2.4.8 and its peripherals is an essential step in setting up the drone for safe and efficient flight operations. Here's an overview of the software tools and processes for configuring and calibrating the Pixhawk:

1. Mission Planner: Mission Planner is a free and open-source ground control station software that allows you to configure and calibrate the Pixhawk. It provides a graphical user interface (GUI) that displays the status and parameters of the Pixhawk, and allows you to modify its settings.
2. Configuration: To configure the Pixhawk, you will need to connect it to your computer and open Mission Planner. From there, you can access the Configuration tab and modify the parameters of the Pixhawk. Some of the parameters that you may need to configure include the flight modes, radio settings, and telemetry settings.
3. Calibration: To calibrate the Pixhawk, you will need to perform a series of procedures to ensure that the sensors and peripherals are working correctly. Some of the calibrations that you may need to perform include the compass calibration, accelerometer calibration, and radio calibration. You can access the calibration procedures from the Initial Setup tab in Mission Planner.
4. Sensor testing: After configuring and calibrating the Pixhawk, you can perform sensor testing to ensure that the sensors are functioning correctly. Mission Planner provides a built-in sensor testing tool that allows you to check the readings of the sensors and verify their accuracy.
5. Firmware updates: It's important to keep the firmware of the Pixhawk up-to-date to ensure that it has the latest features and bug fixes. You can update the firmware of the Pixhawk using Mission Planner by accessing the Install Firmware tab.
6. Data logging: Mission Planner also provides a data logging feature that allows you to record flight data and analyze it later. Data logging can help you identify any issues with the drone and improve its performance.

Overall, configuring and calibrating the Pixhawk 2.4.8 is an essential step in setting up the drone for safe and efficient flight operations. By using Mission Planner and following the calibration procedures, you can ensure that the sensors and peripherals are working correctly and that the

drone is properly configured. Additionally, keeping the firmware up-to-date and performing sensor testing and data logging can help you maintain the drone's performance and identify any issues that may arise.

8.10 Advanced Topics of Pixhawk 2.4.8:

The Pixhawk 2.4.8 is a versatile and powerful flight controller that can be used for a range of advanced applications and features beyond the basic setup and calibration. Here are some advanced topics related to the Pixhawk 2.4.8 that you may want to explore:

1. Integration with other sensors or peripherals: The Pixhawk 2.4.8 can be integrated with a range of other sensors or peripherals to enhance its functionality and performance. For example, you can integrate a LiDAR sensor for obstacle detection or a secondary GPS for increased accuracy. To integrate additional sensors or peripherals, you may need to modify the firmware or use additional software tools.
2. Advanced control features: The Pixhawk 2.4.8 has a range of advanced control features that can be used to enhance the flight performance of the drone. For example, it supports advanced control modes such as altitude hold, loiter, and auto-landing. You can also use advanced control features such as waypoint navigation or follow-me mode to automate the drone's flight path.
3. Customization and programming: The Pixhawk 2.4.8 are highly customizable and can be programmed using a range of programming languages such as C++, Python, or MATLAB. You can use programming to customize the flight behavior of the drone, integrate new features or sensors, or create custom user interfaces for ground control.
4. Telemetry and communication: The Pixhawk 2.4.8 support advanced telemetry and communication features that allow you to monitor and control the drone remotely. For example, you can use telemetry to transmit real-time flight data to a ground control station, or use communication protocols such as MAVLink or ROS to communicate with other devices or drones.
5. Advanced safety features: The Pixhawk 2.4.8 has a range of advanced safety features that can be used to ensure safe and stable flight operations. For example, it supports failsafe modes such as return-to-home or auto-landing in case of a lost connection or other emergencies. You can also use advanced safety features such as geofencing or altitude

limits to restrict the drone's flight path and prevent it from flying into restricted or dangerous areas.

6. Payload integration: The Pixhawk 2.4.8 can be used to integrate and control a range of payloads such as cameras, sensors, or actuators. For example, you can use the Pixhawk to control a camera gimbal or a drop mechanism for payload delivery. To integrate and control payloads, you may need to use additional hardware or software tools.

Overall, the Pixhawk 2.4.8 is a highly capable flight controller that can be used for a range of advanced applications and features beyond the basic setup and calibration. By exploring these advanced topics and features, youcan enhance the functionality and performance of your drone and take advantage of its full potential. However, it's important to approach advanced topics with caution and ensure that you have the necessary knowledge and expertise to implement them safely and effectively. Always follow best practices and guidelines, consult the documentation and community resources, and test your modifications or integrations in a controlled environment before deploying them in a real-world scenario. With careful planning and execution, you can leverage the advanced capabilities of the Pixhawk 2.4.8 to achieve your desired goals and push the boundaries of drone technology.

8.11 Drone safety Considerations

Ensuring that the drone is safe to fly is essential to prevent accidents and protect people, animals, and property. Tips to ensure that the drone is safe to fly:

1. Check the drone before each flight: Make sure that your drone is in good condition and that all the components are securely attached. Check the propellers for any damage, and make sure they are tightened properly. Verify that the battery is fully charged and that the drone is calibrated correctly.
2. Choose a safe flying location: Select a location that is safe and legal for flying a drone. Avoid flying near airports, government buildings, and crowded areas. Also, be mindful of any hazards such as power lines, trees, or bodies of water.
3. Follow local regulations: Being aware of the local regulations and laws related to flying drones. Check for any airspace restrictions, flight height limits, and other regulations that apply to home location.

4. Keep the drone in sight: Always keep the drone within line of sight while flying. Losing sight of drone can lead to accidents and make it difficult to control the drone.
5. Be aware of weather conditions: Avoid flying the drone in windy or rainy conditions as it can affect the stability and control of the drone. Check the weather forecast before flying and avoid flying in adverse weather conditions.
6. Respect people's privacy: Avoid flying the drone over people's houses or private property without their permission. Respect people's privacy and avoid capturing images or videos that can invade their privacy.

Chapter 9 Testing and evaluation

Chapter 9

Testing and evaluation

9.1 Watch

The watch in 3d design:

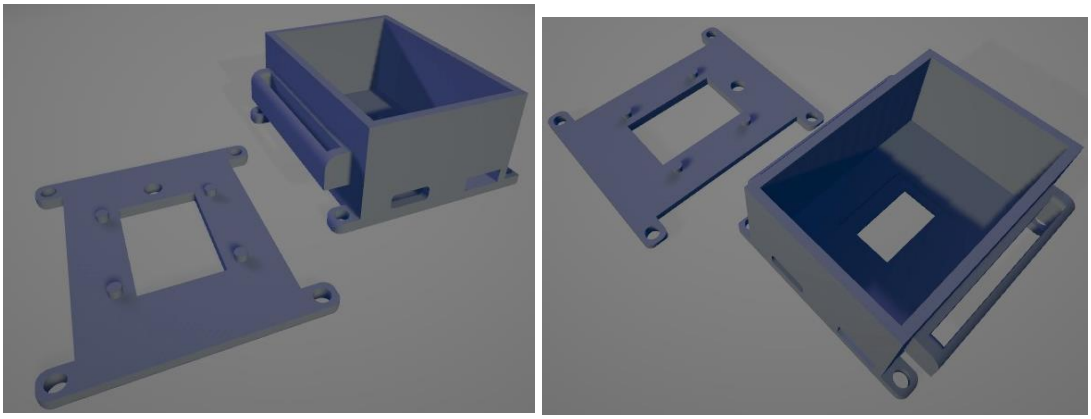


Figure 9- 1 Watch in 3d design

The watch PCB:

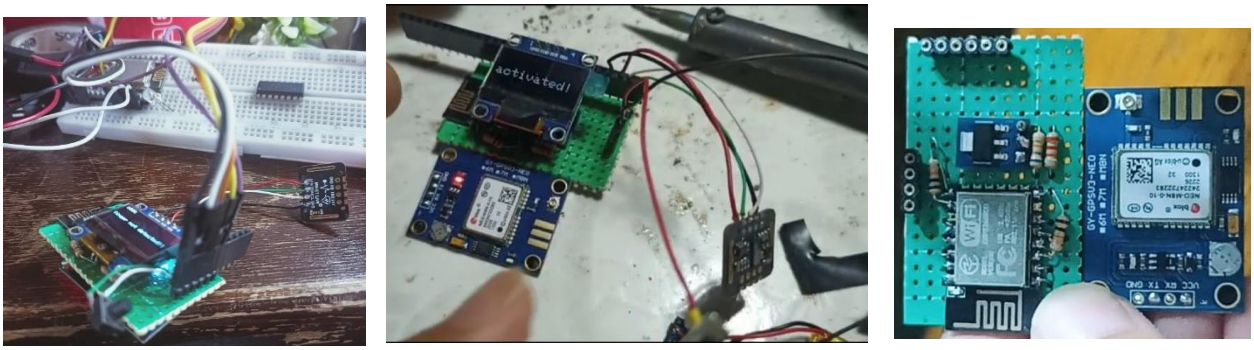


Figure 9- 2 Watch PCB

The watch in real:

Figure 9- 3 Watch in real

9.2 Path planning simulation

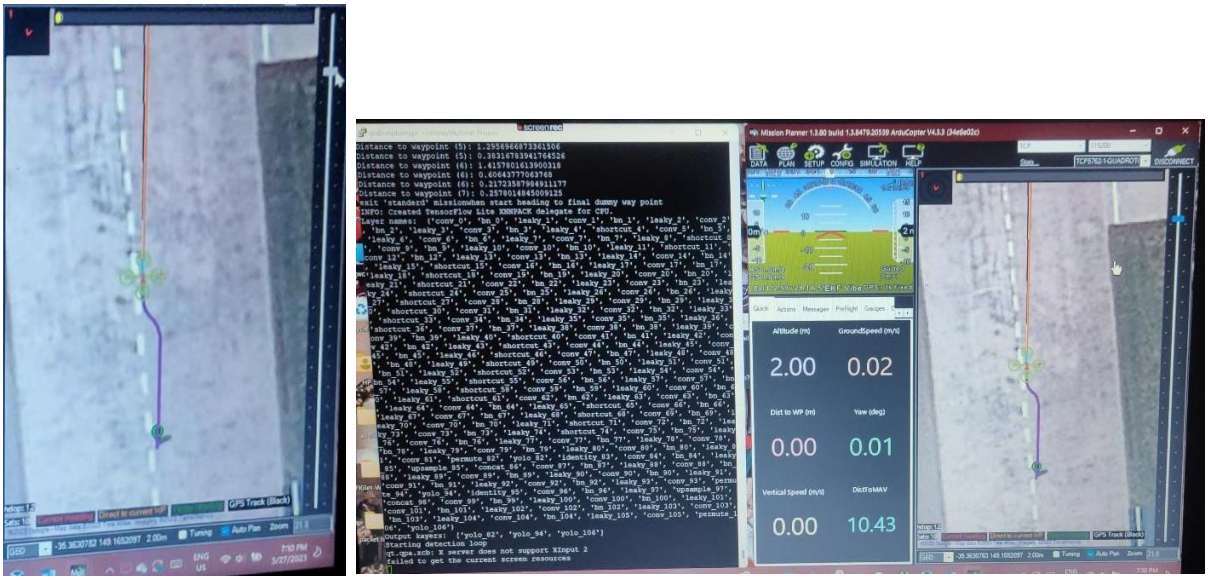


Figure 9- 4 path planning simulation

9.3 Landing image processing

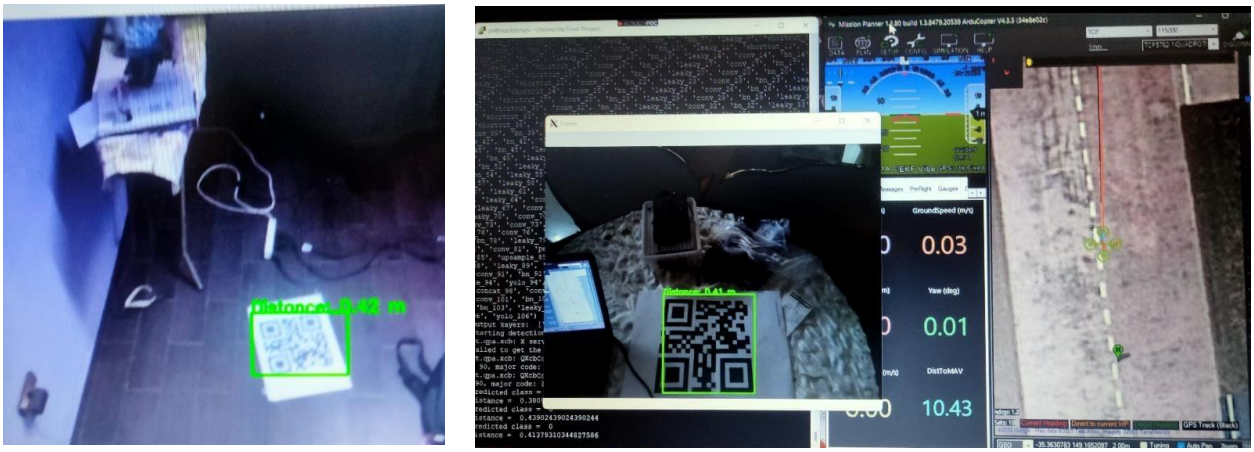


Figure 9- 5 landing image processing

9.4 CPR Mechanism

CPR main circuit:

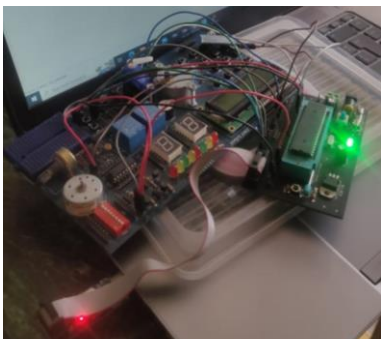


Figure 9- 6 CPR main circuit

CPR PCB:

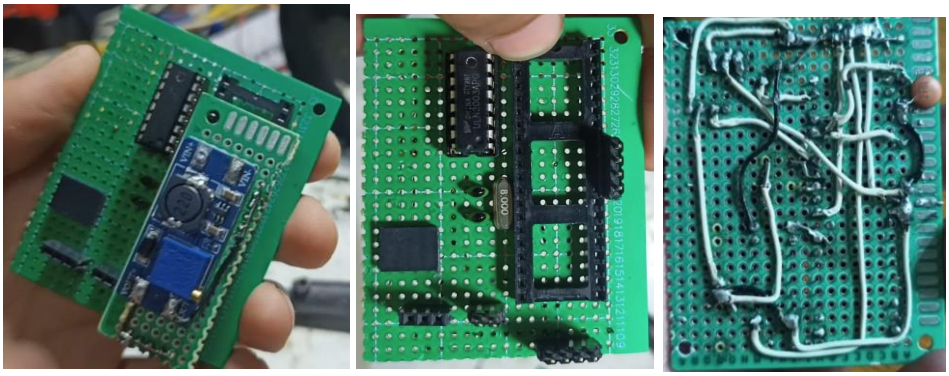


Figure 9- 7 CPR PCB

CPR 3d design:

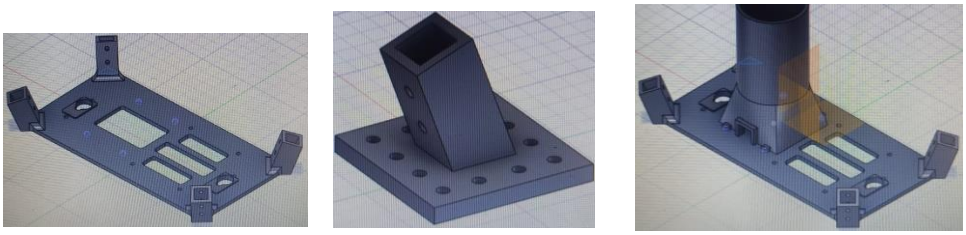


Figure 9- 8 CPR PCB design

CPR in real:

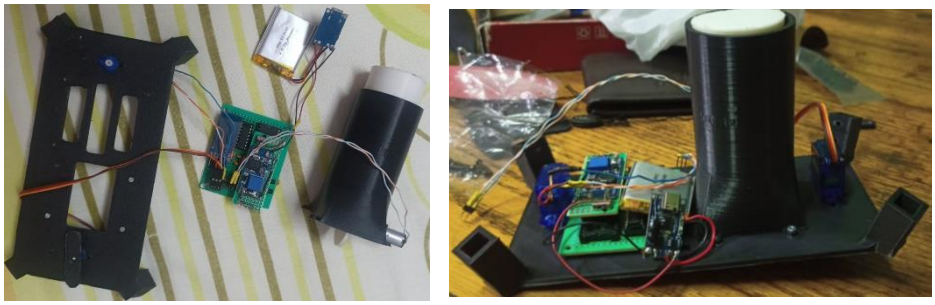


Figure 9- 9 CPR in real

9.5 Quadcopter body and hardware

Figure 9- 10 Quadcopter and hardware

9.6 Quadcopter flight in real

Figure 9- 11 Quadcopter in real

9.8 Flight time:

Path flight:

Distance estimation:

Full system:

Figure 9- 12 Flight time

Chapter 10 Conclusion and Future Work

Chapter 10

Conclusion & Future Work

10.1 Conclusion

The project begins with the development of an AI prediction model that can accurately detect patients at high risk of cardiac arrest based on relevant medical data. This model enhances the proactive identification and monitoring of individuals who may require immediate medical intervention.

Once identified, the patients wear a watch equipped with a heart rate sensor and GPS module, programmed using NodeMCU. The watch continuously collects and transmits vital data, including heart rate and location information, to an online server. This server acts as a central hub for data communication between the watch and the quadcopter.

The quadcopter, controlled by a Raspberry Pi, employs the RRT* (Rapidly-exploring Random Tree) algorithm to calculate the shortest path to reach the patient's location. This algorithm enables efficient and optimized navigation, ensuring quick response times.

Upon arriving at the patient's location, the quadcopter utilizes image processing techniques to detect and decode a QR code on the patient's chest. This information can provide additional critical patient data or medical history for the emergency responders.

In the event that the heart rate sensor on the watch detects a cessation of pulses, a CPR mechanism is activated. A servo motor, controlled by an AVR microcontroller, separates the CPR apparatus from the quadcopter. Real-time data exchange between the watch and the quadcopter is facilitated by a Bluetooth module.

The CPR mechanism, driven by a DC motor, performs mechanical push-ups and push-downs until the pulse is detected to have returned, at which point the motor automatically stops.

10.2 Future work

- The quadcopter can be used in unknown and more complex environment by applying SLAM algorithms and by using an RPLidar. It can be used to draw new unknown maps

for the environment. It may also include features extractions to be able to act in dynamic complex environments with much obstacles.

- For high accuracy of distance estimation, a 2d lidar can be used for this rather than the camera; as the complex calculations and parameters of it. In addition, the camera is computationally expensive and has a high time, power, and memory consumption, as well as not having a very high accuracy.
- More advanced deep learning techniques, architectures. As well as computer vision algorithms can be used for quicker localization and recognition.
- More advanced path planning algorithms can be used for optimization to reduce time, power, and memory consumptions which in turns reduce the overall cost.
- More accurate and higher quality sensors can be used to enhance the hardware integration with the advanced algorithms.
- Enhance the watch function with implementing a quick cardiac arrest prediction before heart rate stopping to increase the percentage of rescuing.
- Reinforcement algorithms and techniques can be used to enable the quadcopter to take the right movement decisions against the wind storms.
- The way of applying the CPR for the patient may be improved by using RF instead of the QR code recognition.
- Add safety checks to prevent taking off or landing in unsafe conditions.
- Add redundant sensors to ensure that the drone can continue flying even if one system fails.
- Distribute the processing across multiple boards on the Raspberry Pi 4 to improve performance in case of processing large amount of data or perform complex calculations.
- Add fail-safes to ensure the drone can safely land in case of a failure.
- Add a physical emergency stop button to your drone that immediately stops the motors and lands the drone safely to handle emergency or uncontrollable situations.
- Implement real-time monitoring of critical drone parameters such as battery level, altitude, and speed; to alert you to any potential safety issues. This is done using GUI.
- Use backup communication if the primary communication link between the drone and the ground station fails to be able to maintain control of the drone.

- Select appropriate sensors: The CPR device will need to include sensors to detect whether compressions are being performed correctly and to monitor the patient's vital signs and send feed back to the emergency department. Some potential sensors to consider include force sensors to measure the force of compressions
- Develop the motor control system: The CPR device will need a motor to perform the compressions. This will require developing a motor control system that can drive the motor at the appropriate speed and force to perform effective compressions. The motor control system will need to be designed to work with the AVR microcontroller and the selected sensors.
- Communication capabilities: Enable communication features such as calls, text messages, and notifications. This allows users to stay connected without having to reach for their smartphone. With voice recognition implemented for hands-free operation.
- Connect the watch with a smartphone via Bluetooth technology.

References

[1] ul Husnain, A., Mokhtar, N.B., Shah, N.B.M., Dahari, M.B., Ikmal, M.S., Ramlee, B., Rajagopal, H. and Iwahashi, M., 2023. A Study on the Impact of Hardware Limitations in Multi-Rotor UAVs on Coverage Path Planning Models.

[2] Gugan, G. and Haque, A., 2023. Path Planning for Autonomous Drones: Challenges and Future Directions. Drones, 7(3), p.169.

[3] Voxid, F., Xolbek, X. and Kamoliddin, X., 2023. Sorting the object based on neural networks computer vision algorithm of the system and software. ijtimoiy fanlarda innovasiya onlayn ilmiy jurnali, 3(1), pp.67-69.

[4] Zhu, C., Liang, J. and Zhou, F., 2023. Transfer learning-based YOLOv3 model for road dense object detection. Journal of Electronic Imaging, 32(6), p.062505.

[5] Uc-Cetina, V., Navarro-Guerrero, N., Martin-Gonzalez, A., Weber, C. and Wermter, S., 2023. Survey on reinforcement learning for language processing. Artificial Intelligence Review, 56(2), pp.1543-1575.

[6] Gadre, Dhananjay V., “Programming and customizing the AVR microcontroller“ McGraw-Hill, Inc., 2002, pages 112–135.

[7] Bailey, Tim.,"An Introduction to the C Programming Language and Software Design", July-2005.

[8] Mazidi Muhammad, Mckinlay Rolin, Mazidi Janice,” The 8051 Microcontroller and Embedded Systems Using Assembly and C,” Pearson, 2007.

[9] Vierhauser, M., Garmendia, A., Stadler, M., Wimmer, M. and Cleland-Huang, J., 2023. GRuM–A flexible model-driven runtime monitoring framework and its application to automated aerial and ground vehicles. Journal of Systems and Software, p.111733.

[10] Weitbrecht, P., Monteiro, C., Jacomussi, L., Borin, M. and Jardim, C., 2023, February. MEMs Based Low-Cost Urban Noise Monitoring: Tests and Case Study. In INTER-NOISE and NOISE-CON Congress and Conference Proceedings (Vol. 265, No. 1, pp. 6266-6276). Institute of Noise Control Engineerin

Appendix

Cardiac arrest prediction code

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("patients data.csv")
data.head()
print(data.shape)
print(data.columns)
data.info()
data['age'] = data['age'].astype(int)
data['platelets'] = data['platelets'].astype('int64')
data.describe()
# Cardiac arrest distribution
import plotly.express as px
fig = px.pie(data, names='is_under_risk', title='Distribution of
cardiac arrest probability', width=600, height=400)
fig.show()
# Mean values of all the features for cases of is_under_risk or not
yes = data[data['is_under_risk'] == 1].describe().T
no = data[data['is_under_risk'] == 0].describe().T


colors = ['#2596be','#FFFFFF']

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(5, 5),
gridspec_kw={'wspace': 2.5})
plt.subplot(1, 2, 1)
sns.heatmap(yes[['mean']],annot = True,cmap = colors,linewidths =
0.4,linecolor = 'black',cbar = False,fmt = '.3f')
plt.title('Cardiac arrest Occurred');

plt.subplot(1,2,2)
sns.heatmap(no[['mean']],annot = True,cmap = colors,linewidths =
0.4,linecolor = 'black',cbar = False,fmt = '.3f')
plt.title('No cardiac arrest Occurred');

plt.tight_layout(pad=-3)
def plot_hist(variable):
    plt.figure(figsize=(9, 3))
    plt.hist(data[variable], color='darkred')
    plt.xlabel(variable)
```

```
    plt.ylabel('Frequency')
    plt.title("{} Distribution".format(variable))
    plt.show()

numericVars = ['age', 'creatinine_phosphokinase',
'ejection_fraction',
               'platelets', 'serum_creatinine', 'serum_sodium',
'time']
for n in numericVars:
    plot_hist(n)
def bar_plot(variable):
    # get feature
    var = data[variable]
    #count number of categorical variable (value/sample)
    varValue = var.value_counts()

    #visualize
    plt.figure(figsize=(9,3))
    plt.bar(varValue.index, varValue,color = "lightgreen", edgecolor
= "black", linewidth = 2)
    plt.xticks(varValue.index, varValue.index.values)
    plt.ylabel("frequency")
    plt.title(variable)
    plt.show()
    print("{}: \n {}".format(variable,varValue))
category =
["anaemia","diabetes","high_blood_pressure","sex","smoking","is_under
_risk"]
for c in category:
    bar_plot(c)
# check correlation --> heatmap
corr = data.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
corr[abs(corr['is_under_risk']) > 0.1]['is_under_risk']
from collections import Counter
def detect_outliers(df, features):
    outlier_indices = []

    for c in features:
        Q1 = np.percentile(df[c], 25)  # 1st quartile
        Q3 = np.percentile(df[c], 75)  # 3rd quartile
        IQR = Q3 - Q1
        outlier_step = IQR * 1.5
        outlier_list_col = df[(df[c] < Q1 - outlier_step) | (df[c] >
```

```
Q3 + outlier_step)].index  # detect quartiles indices
        outlier_indices.extend(outlier_list_col)

    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(i for i, v in outlier_indices.items() if
v > 1)

    return multiple_outliers
data =
data.drop(detect_outliers(data,["age","creatinine_phosphokinase","eje
ction_fraction","platelets","serum_creatinine","serum_sodium","time"]
),axis = 0).reset_index(drop=True)
from scipy.stats import skew, norm, boxcox
# chcek skewness
skewed_features = data.apply(lambda x:
skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame(skewed_features, columns=['skewed'])
skewness
data["creatinine_phosphokinase"], lam =
boxcox(data["creatinine_phosphokinase"])
data["serum_creatinine"], lam_serum_creatine =
boxcox(data["serum_creatinine"])
data["ejection_fraction"], lam_serum_creatine =
boxcox(data["ejection_fraction"])
data["platelets"], lam_serum_creatine = boxcox(data["platelets"])
numericVars = ['creatinine_phosphokinase','ejection_fraction',
               'platelets', 'serum_creatinine']
for n in numericVars:
    plot_hist(n)
# target and features
X = data.drop("is_under_risk", axis = 1)
y = data.is_under_risk
# Balancing
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_sm, y_sm = sm.fit_resample(X, y)
print("Before Smote: \n", y.value_counts())
print("After Smote: \n", y_sm.value_counts())
# Train- Test splitting
from sklearn.model_selection import train_test_split,
StratifiedKFold, GridSearchCV
test_size = 0.2
X_train, X_test, Y_train, Y_test = train_test_split(X_sm, y_sm,
test_size=test_size, random_state=42)
result_acc = []
```

```
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error, confusion_matrix,
accuracy_score
XGB = XGBClassifier(max_depth = 1)
XGB.fit(X_train, Y_train)
y_pred_xgb = XGB.predict(X_test)
cm_xgb = confusion_matrix(y_pred_xgb, Y_test)
acc_xgb = accuracy_score(Y_test, y_pred_xgb)
result_acc.append(acc_xgb)
print("RESULT")
print("XGBoost Model Acc : ",acc_xgb)
print("XGBoost Model Cm : ",cm_xgb)
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
model_rnd = RandomForestClassifier()
model_rnd.fit(X_train, Y_train)
importance = model_rnd.feature_importances_

# plot feature importance
plt.bar([x for x in range(len(importance))], importance, color =
"red")
plt.show()
x_train_random_forest =
X_train[["age","creatinine_phosphokinase","ejection_fraction","serum_
creatinine","time"]]
x_test_random_forest =
X_test[["age","creatinine_phosphokinase","ejection_fraction","serum_c
reatinine","time"]]
random_forest_model = RandomForestClassifier(max_depth=7,
random_state=25)
random_forest_model.fit(x_train_random_forest, Y_train)
y_pred_random_forest =
random_forest_model.predict(x_test_random_forest)
cm_random_forest = confusion_matrix(y_pred_random_forest, Y_test)
acc_random_forest = accuracy_score(Y_test, y_pred_random_forest)
result_acc.append(acc_random_forest)
print("RESULT")
print("Random Forest Model Acc : ",acc_random_forest)
print("Random Forest Model Cm : ",cm_random_forest)
from sklearn.linear_model import LogisticRegression
model_log_reg = LogisticRegression()
model_log_reg.fit(X_train, Y_train)
importance = model_log_reg.coef_[0]

plt.bar([x for x in range(len(importance))], importance, color =
"orange")
```

```
plt.show()
x_train_log_reg =
X_train[["creatinine_phosphokinase","ejection_fraction","serum_creati
nine","sex"]]
x_test_log_reg =
X_test[["creatinine_phosphokinase","ejection_fraction","serum_creatin
ine","sex"]]
log_reg = LogisticRegression()
log_reg.fit(x_train_log_reg, Y_train)
y_pred_log = log_reg.predict(x_test_log_reg)
cm_log_reg = confusion_matrix(y_pred_log, Y_test)
acc_log_reg = accuracy_score(Y_test, y_pred_log)
result_acc.append(acc_log_reg)
print("RESULT")
print("Logistic Regression Model Acc : ",acc_log_reg)
print("Logistic Regression Model Cm : ",cm_log_reg)
from sklearn.tree import DecisionTreeClassifier
model_decision_tree = DecisionTreeClassifier()
model_decision_tree.fit(X_train, Y_train)
importance = model_decision_tree.feature_importances_

plt.bar([x for x in range(len(importance))], importance, color =
"blue")
plt.show()
x_train_dec =
X_train[["creatinine_phosphokinase","ejection_fraction","time"]]
x_test_dec =
X_test[["creatinine_phosphokinase","ejection_fraction","time"]]
dt_param_grid = {"min_samples_split" : range(10,500,20),
                 "max_depth": range(1,20)}

decision_tree = DecisionTreeClassifier()
clf = GridSearchCV(decision_tree, param_grid=dt_param_grid, cv =
StratifiedKFold(n_splits = 10), scoring = "accuracy", n_jobs = -
1,verbose = 1)
clf.fit(x_train_dec,Y_train)

y_pred_decision_tree = clf.predict(x_test_dec)
cm_y_pred_decision_tree = confusion_matrix(y_pred_decision_tree,
Y_test)
acc_y_pred_decision_tree = accuracy_score(Y_test,
y_pred_decision_tree)
result_acc.append(acc_y_pred_decision_tree)
print("RESULT")
print("Decision Tree Model Acc : ",acc_y_pred_decision_tree)
```

```
print("Decision Tree Model Cm : ",cm_y_pred_decision_tree)
from sklearn.svm import SVC
model_svm = SVC(kernel="linear")
model_svm.fit(X_train, Y_train)
importance = model_svm.coef_[0]

plt.bar([x for x in range(len(importance))], importance, color =
"brown")
plt.show()
x_train_svm =
X_train[["creatinine_phosphokinase","ejection_fraction","serum_creati
nine","sex"]]
x_test_svm =
X_test[["creatinine_phosphokinase","ejection_fraction","serum_creatin
ine","sex"]]
svm = SVC()
svm.fit(x_train_svm, Y_train)
y_pred_svm = svm.predict(x_test_svm)
cm_svm = confusion_matrix(y_pred_svm, Y_test)
acc_svm = accuracy_score(Y_test, y_pred_svm)
result_acc.append(acc_svm)
print("RESULT")
print("SVM Model Acc : ",acc_svm)
print("SVM Model Cm : ",cm_svm)
from catboost import CatBoostClassifier, Pool
best_params = {'bagging_temperature': 0.8,
               'depth': 5,
               'iterations': 500,
               'l2_leaf_reg': 30,
               'learning_rate': 0.05,
               'random_strength': 0.8}

model_cat_boost = CatBoostClassifier(
        **best_params,
        loss_function='Logloss',
        eval_metric='Accuracy',
        nan_mode='Min',
        verbose=False
    )

model_cat_boost.fit(
        X_train, Y_train,
        verbose_eval=100,
        early_stopping_rounds=50,
        eval_set=(X_test, Y_test),
```

```
        use_best_model=False,
        plot=True
)

y_pred_cat_boost = model_cat_boost.predict(X_test)

cm_cat_boost = confusion_matrix(y_pred_cat_boost, Y_test)
acc_cat_boost = accuracy_score(Y_test, y_pred_cat_boost)
result_acc.append(acc_cat_boost)
print("RESULT")
print("Cat Boost Model Acc : ",acc_cat_boost)
print("Cat Boost Model Cm : ",cm_cat_boost)
results = pd.DataFrame({"Model Result":result_acc,
                        "Models":["XGBoost",
                                  "RandomForest",
                                  "LogisticRegression",
                                  "DecisionTree",
                                  "SVM",
                                  "CatBoost"]})
print(results)
```

Watch code

```
#include <TinyGPS++.h>
#include <TinyGPSPlus.h>
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define wifi_ssid "magdy helal"
#define wifi_password "A732000A"
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define AIO_USERNAME  "hagerahmed"
#define AIO_KEY       "aio_fXRG98LfedlipeeqGZaUu5mvztJu"

//*************************oled decleration*********************
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET     -1 // Reset pin
// Declaration for SSD1306 display connected using I2C
```

```
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
//create an object of Adafruit_SSD1306.h


WiFiClient client;
Adafruit_MQTT_Client mqtt(&client ,AIO_SERVER,AIO_SERVERPORT,AIO_USERNAME,
AIO_KEY);
Adafruit_MQTT_Publish AverageBeat = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME
"/feeds/avgBeat");
Adafruit_MQTT_Publish Location_lat = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME
"/feeds/Loc_Lat");
Adafruit_MQTT_Publish Location_lng = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME
"/feeds/Loc_Lng");
Adafruit_MQTT_Publish Drone_Deploy = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME
"/feeds/Drone_Deploy");

TinyGPSPlus gps;
SoftwareSerial ss(14,12);


MAX30105 particleSensor;

unsigned long previousmillis = 0;
#define interval 1000
#define Button1 13

/*******************Battery Setup*********************/
const int analogInPin = A0;  // ESP8266 Analog Pin ADC0 = A0

int BatteryValue = 0;  // value read from the pot
int BatteryPercent = 0; // percentage of the battery
const int BatteryEmpty = 902; // Digital value of the 3.7v of li-ion battery
const int BatteryRange = 1024 - BatteryEmpty ; //the range of battery charge
(used to devide the value of battery)

const byte RATE_SIZE = 4; //number of average times
byte rates[RATE_SIZE];    // array of heart rates
byte rateSpot =0;
long lastBeat = 0 ;    // time at which the last beat occurred

byte ledBrightness = 60; //Options: 0=Off to 255=50mA
byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR +
Green
byte sampleRate = 100; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
int pulseWidth = 411; //Options: 69, 118, 215, 411
int adcRange = 4096; //Options: 2048, 4096, 8192, 16384

float beatsPerMinute;
```

```
int beatAvg; // averadge of 4 beats
int drone_deploy = 0;

float longitude,latitude;
String lat_str,lng_str,time_str,date_str;
int sec5counter = 0;
int
hour,second,minute,date,month,year,pm,am;


void BatteryPercentage () // a function that changes the value of the battery
percentage displayed in oled display
{
  BatteryValue = analogRead(analogInPin);
  BatteryPercent = (float)(BatteryValue - BatteryEmpty) / (float)BatteryRange
* 100.0;
  //return BatteryPercent;
}


void setup() {

  pinMode(Button1, INPUT_PULLUP);



  Serial.begin(115200);
  // initialize the OLED object..(SSD1306_SWITCHCAPVCC turns on the internal
charge pump circuitry),(OLED display's I2C address)
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Clear the buffer.
  display.clearDisplay();

  // Display Text
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,28);
  display.println("activated!");
  display.display();
  delay(2000);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,28);
  display.println("Connecting ");
  display.println("To WiFi ...");
```

```
   display.display();
  display.clearDisplay();


  Serial.print(F("Connecting to "));
  Serial.println(wifi_ssid);
  WiFi.begin(wifi_ssid, wifi_password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));

  }
  Serial.println();
  Serial.println(F("WiFi connected"));
  Serial.println(F("IP address: "));
  Serial.println(WiFi.localIP());
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,28);
  display.println("Connected to ");
  display.println(wifi_ssid);
   display.display();
  display.clearDisplay();
  delay(1000);



  // connect to adafruit io
  connect();
  ss.begin(9600);

  Serial.println("initializing...");
  while(!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port,
400kHz speed
  {
    Serial.println("sensor not detected , please chech wiring or power");
    while(1);
  }
  Serial.println("place your finger steadly on the sensor ");
  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate,
pulseWidth, adcRange); //Configure sensor with these settings
  particleSensor.setPulseAmplitudeRed(0x0B);
  particleSensor.setPulseAmplitudeGreen(0);



}

void connect() {
```

```
  Serial.print(F("Connecting to Adafruit IO... "));
  int8_t ret;
  while ((ret = mqtt.connect()) != 0) {
    switch (ret) {
      case 1: Serial.println(F("Wrong protocol")); break;
      case 2: Serial.println(F("ID rejected")); break;
      case 3: Serial.println(F("Server unavail")); break;
      case 4: Serial.println(F("Bad user/pass")); break;
      case 5: Serial.println(F("Not authed")); break;
      case 6: Serial.println(F("Failed to subscribe")); break;
      default: Serial.println(F("Connection failed")); break;
    }

    if(ret >= 0)
      mqtt.disconnect();

    Serial.println(F("Retrying connection..."));
    delay(10000);
  }
  Serial.println(F("Adafruit IO Connected!"));
}

void loop() {

  BatteryPercentage(); // get the battery percentage
  unsigned long currentmillis = millis();
  long irValue = particleSensor.getIR();


  while(ss.available() > 0)
  {if (gps.encode(ss.read() ))
  {
    if (gps.location.isValid())
    {
      latitude = gps.location.lat();
      lat_str = String(latitude,8);
      longitude = gps.location.lng();
      lng_str = String(longitude,8);
    }
    if (gps.date.isValid()) //check whether gps date is valid
      {
        date_str = "";
        date = gps.date.day();
        month = gps.date.month();
        year = gps.date.year();
        if (date < 10)
          date_str = '0';
        date_str += String(date);// values of date,month and year are stored
in a string
        date_str += "/";
```

```
        if (month < 10)
          date_str += '0';
        date_str += String(month); // values of date,month and year are
stored in a string
        date_str += "/";
        if (year < 10)
          date_str += '0';
        date_str += String(year); // values of date,month and year are stored
in a string
      }
      if (gps.time.isValid())  //check whether gps time is valid
      {
        time_str = "";
        hour = gps.time.hour();
        minute = gps.time.minute();
        second = gps.time.second();
        minute = (minute + 30); // converting to IST
        if (minute > 59)
        {
          minute = minute - 60;
          hour = hour + 1;
        }
        hour = (hour + 5) ;
        if (hour > 23)
          hour = hour - 24;   // converting to IST
        if (hour >= 12)  // checking whether AM or PM
          pm = 1;
        else
          pm = 0;
        hour = hour % 12;
        if (hour < 10)
          time_str = '0';
        time_str += String(hour); //values of hour,minute and time are stored
in a string
        time_str += ":";
        if (minute < 10)
          time_str += '0';
        time_str += String(minute); //values of hour,minute and time are
stored in a string
        if (pm == 1)
          time_str += " PM ";
        else
          time_str += " AM ";
      }
  }
  }
  const char* Newtime = time_str.c_str();  //convert string to char for mqtt
  const char* Newdate = date_str.c_str();  //convert string to char for mqtt
```

```
  if (checkForBeat(irValue)==true )
  {
    // we sensed a beat
    long delta = millis() - lastBeat;
    lastBeat = millis();

     beatsPerMinute = 60 / (delta / 1000.0);

     if (beatsPerMinute<255 && beatsPerMinute >= 0)  //edit as needed
     {
      rates[rateSpot++] = (byte)beatsPerMinute;  // store this reading in the
array
      rateSpot %= RATE_SIZE;  //wrap the varible

      //take the avg of reading
      beatAvg = 0 ;
      for (byte i = 0;i < RATE_SIZE ; i++)
      {
        beatAvg+=rates[i];
      }
      beatAvg /= RATE_SIZE;
     }
  }

  Serial.print("IR=");
  Serial.print(irValue);
  Serial.print(", BPM=");
  Serial.print(beatsPerMinute);
  Serial.print(", Avg BPM=");
  Serial.print(beatAvg);
  Serial.println("");
  Serial.print("latitude :");
  Serial.print(lat_str);
  Serial.print(", longitude :");
  Serial.println(lng_str);
  //num_splitter(beatAvg);
  if (irValue < 50000)
  {
      display.clearDisplay();

display.setTextSize(1);
display.fillRect(96, 0, 6, 8, WHITE);
    display.setCursor(103,0);
    display.print(BatteryPercent);
    display.println("%");

display.setCursor(0,11);
    display.println(date_str);
display.setCursor(0,0);
    display.println(time_str);
```

```
    display.setTextSize(1.5);
    display.setCursor(0,28);
    display.println("finger not detected!!");
    display.display();
    delay(500);

  }
   // Serial.print(" No finger?");

else
{
  if (digitalRead(Button1)==0)
  {
     drone_deploy =1;
     beatAvg=0;
     Serial.println("Button pressed");
    Drone_Deploy.publish(drone_deploy);
    AverageBeat.publish(beatAvg);
    Location_lat.publish(lat_str.c_str());
    Location_lng.publish(lng_str.c_str());
    display.clearDisplay();
    display.setTextColor(WHITE);
    display.setTextSize(1.5);
    display.setCursor(0,25);
    display.println("cardiac arrest!!!");
    display.display();
display.startscrollright(0x00, 0x07);
delay(2000);
display.stopscroll();
delay(1000);
display.startscrollleft(0x00, 0x07);
delay(2000);
display.stopscroll();
delay(1000);
  }
  else
  {
  if(currentmillis - previousmillis >= interval){
  previousmillis =currentmillis;

display.clearDisplay();

display.setTextSize(1);
display.fillRect(96, 0, 6, 8, WHITE);
    display.setCursor(103,0);
    display.print(BatteryPercent);
    display.println("%");

display.setCursor(0,11);
```

```
    display.println(date_str);
display.setCursor(0,0);
    display.println(time_str);

    display.setTextSize(1);
    display.setCursor(0,28);
    display.print("heart beat:");
    display.print(beatAvg);
    display.display();
  sec5counter++;
  if (sec5counter >=9)
  {
  AverageBeat.publish(beatAvg);
    drone_deploy =0;
    Drone_Deploy.publish(drone_deploy);
  }
  sec5counter=0;
  }
  }
}
```

Path planning

```
import math
import random
import numpy as np
import server5


# A class to represent a node in the RRT* tree.
class Node:
   def __init__(self, x, y):
       self.x = x
       self.y = y
       self.cost = 0
       self.parent = None


# A class to represent the Informed RRT* algorithm.
class InformedRRTStar:
   def __init__(self, start, goal, x_min=1500, x_max=2000, y_min=-5000,
y_max=-2000, max_iter=3000, step_size=200, goal_tol=50):
       self.start = Node(start[0], start[1])
       self.goal = Node(goal[0], goal[1])
       # self.obstacles = obstacles
       self.x_min = x_min
       self.x_max = x_max
       self.y_min = y_min
       self.y_max = y_max
```

```
        self.max_iter = max_iter
        self.step_size = step_size
        self.goal_tol = goal_tol
        self.nodes = []
        self.path = []

    # Plan a path from the start to the goal using Informed RRT* algorithm.
    def plan(self):
        self.nodes.append(self.start)
        for i in range(self.max_iter):
            q_rand = self.get_random_node()
            q_near = self.get_nearest_node(q_rand)
            q_new = self.extend(q_near, q_rand)
            if q_new and not self.is_collision(q_near, q_new):
                self.nodes.append(q_new)
                self.rewire(q_new)
                if self.calculate_distance(q_new, self.goal) <= self.goal_tol:
                    self.path = self.get_path()
                    return self.path
        return None

    # Generate a random node within the state space using ellipse sampling.
    def get_random_node(self):
        if random.random() > 0.5:
            q_rand = Node(random.uniform(self.x_min, self.x_max),
random.uniform(self.y_min, self.y_max))
        else:
            # Ellipse parameters
            a = self.calculate_distance(self.start, self.goal) / 2
            b = a / 2

            # Ellipse center
            xc = (self.start.x + self.goal.x) / 2
            yc = (self.start.y + self.goal.y) / 2

            # Angle of major axis
            theta = math.atan2(self.goal.y - self.start.y, self.goal.x -
self.start.x)

            # Generate random point on ellipse
            t = 2 * math.pi * random.random()
            u = random.random() + random.random()
            r = u if u < 1 else 2 - u
            x = xc + r * a * math.cos(t) * math.cos(theta) - r * b *
math.sin(t) * math.sin(theta)
            y = yc + r * a * math.cos(t) * math.sin(theta) + r * b *
math.sin(t) * math.cos(theta)

            q_rand = Node(x, y)
        return q_rand
```

```
    # Get the nearest node in the tree to the randomly generated node.
    def get_nearest_node(self, q_rand):
        distances = [self.calculate_distance(q_rand, n) for n in self.nodes]
        min_index = distances.index(min(distances))
        return self.nodes[min_index]

    # Extend the tree towards the randomly generated node.
    def extend(self, q_near, q_rand):
        theta = math.atan2(q_rand.y - q_near.y, q_rand.x - q_near.x)
        q_new = Node(q_near.x + self.step_size * math.cos(theta), q_near.y +
self.step_size * math.sin(theta))
        q_new.parent = q_near
        q_new.cost = q_near.cost + self.calculate_distance(q_near, q_new)
        return q_new

    # Rewire the tree if a new node provides a shorter path to its neighbors.
    def rewire(self, q_new):
        for node in self.nodes:
            if node == q_new or node.parent is q_new or
self.is_collision(node, q_new):
                continue
            if self.calculate_distance(q_new, node) < self.step_size and
node.cost > q_new.cost + self.calculate_distance(q_new, node):
                node.parent = q_new
                node.cost = q_new.cost + self.calculate_distance(q_new, node)

    # Return the optimal path from the start to the goal.
    def get_path(self):
        path = []
        node = self.nodes[-1]
        while node.parent is not None:
            path.append([node.x, node.y])
            node = node.parent
        path.append([self.start.x, self.start.y])
        path.reverse()
        return path

    # Check if the path between two nodes collides with any obstacles.
    def is_collision(self, q1, q2):
        return False
        '''
        for obstacle in self.obstacles:
            if self.line_rectangle_intersect(q1.x, q1.y, q2.x, q2.y,
obstacle):
                return True
        return False
        '''

    # Check if a line segment intersects with a rectangle.
```

```
    def line_rectangle_intersect(self, x1, y1, x2, y2, rect):
        x_min, y_min, width, height = rect
        x_max = x_min + width
        y_max = y_min + height

        if x1 > x_max and x2 > x_max:
            return False
        if x1 < x_min and x2 < x_min:
            return False
        if y1 > y_max and y2 > y_max:
            return False
        if y1 < y_min and y2 < y_min:
            return False

        m = (y2 - y1) / (x2 - x1)
        y = m * (x_min - x1) + y1
        if y_min <= y <= y_max:
            return True

        y = m * (x_max - x1) + y1
        if y_min <= y <= y_max:
            return True

        x = (y_min - y1) / m + x1
        if x_min <= x <= x_max:
            return True

        x = (y_max - y1) / m + x1
        if x_min <= x <= x_max:
            return True

        return False

    # Calculate the Euclidean distance between two nodes.
    def calculate_distance(self, q1, q2):
        return math.sqrt((q1.x - q2.x) ** 2 + (q1.y - q2.y) ** 2)


#start = (1991, -23)
goal_x = server5.x / 1000
goal_y = server5.y / 1000
goal = (goal_x, goal_y)
# obstacles = [(1000, -200, 20, 20)]
rrt = InformedRRTStar(start, goal)
path = rrt.plan()
if path is not None:
    print("First path: ", path)
    print("First path length = ", len(path))
```

```
    def smooth(path, weight_data=0.5, weight_smooth=0.1, tolerance=0.00001):
        newPath = [[0 for col in range(len(path[0]))] for row in
range(len(path))]
        for i in range(len(path)):
            for j in range(len(path[0])):
                newPath[i][j] = path[i][j]

        change = 1
        while change > tolerance:
            change = 0
            for i in range(1, len(path) - 1):
                for j in range(len(path[0])):
                    ori = newPath[i][j]
                    newPath[i][j] = newPath[i][j] + weight_data * (path[i][j]
- newPath[i][j])
                    newPath[i][j] = newPath[i][j] + weight_smooth * (
                            newPath[i + 1][j] + newPath[i - 1][j] - 2 *
newPath[i][j])
                    change += abs(ori - newPath[i][j])
        return newPath


    smoothed_path = smooth(path)

    final_path = []
    for i in smoothed_path:
        i[0] /= 100
        i[1] /= 100
        final_path.append([i[0], i[1]])

    distance_togo = []
    for i in range(len(final_path)-1):
        distance = (float(final_path[i + 1][0]) - float(final_path[i][0]),
float(final_path[i + 1][1]) - float(final_path[i][1]))
        distance_togo.append(distance)

    print("Final distance = ", distance_togo)
    print(len(distance_togo))
else:
    print("no path found!")
```

landing system code

QR code recognition and classification

```
import os
import tensorflow as tf
from tensorflow import keras
from keras import layers
```

```
import cv2

# Define the dataset directories
train_dir = 'data/'
train_datagen =
keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255)

# Define the model architecture
model = keras.Sequential([
   layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224,
224, 3)),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(64, (3, 3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Conv2D(128, (3, 3), activation='relu'),
   layers.MaxPooling2D((2, 2)),
   layers.Flatten(),
   layers.Dense(128, activation='relu'),
   layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
train_data = train_datagen.flow_from_directory(train_dir,
target_size=(224, 224), batch_size=32, class_mode='binary')
model.fit(train_data, epochs=10)

# Save the model
model_version=max([int(i) for i in os.listdir("models") + [0]])+1
model.save(f"models/{model_version}")

model.save("my_model.h5")
```

Compression code

```
import tensorflow as tf
from keras.models import load_model

model = load_model('my_model.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

with open('my_model.tflite', 'wb') as f:
 f.write(tflite_model)
```

QR Code localization

```
import cv2
import numpy as np
import glob
import random
import shutil

# Load YOLO
net = cv2.dnn.readNet("yolov3_training_last.weights",
"yolov3_testing.cfg")

# Name custom object
classes = ["QRCode"]

# Images_path
images_path = glob.glob(r"E:\\qrdata\\*.jpg")

layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Loading image
img = cv2.imread("QR_Tshirt.jpg")
img = cv2.resize(img, None, fx=0.4, fy=0.4)
height, width, channels = img.shape

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True,
crop=False)
net.setInput(blob)
outs = net.forward(output_layers)

# Showing information on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
   for detection in out:
       scores = detection[5:]
       class_id = np.argmax(scores)
       confidence = scores[class_id]
       if confidence > 0.3:
           # Object detected
           print(class_id)
           center_x = int(detection[0] * width)
           center_y = int(detection[1] * height)
           w = int(detection[2] * width)
           h = int(detection[3] * height)

           # Rectangle coordinates
```

```
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
# print(indexes)
font = cv2.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
   if i in indexes:
       x, y, w, h = boxes[i]
       print(x, y, w, h)
       label = str(classes[class_ids[i]])
       color = colors[class_ids[i]]
       cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
       # cv2.putText(img, label, (x, y-5), font, 3, color, 2)
       specific_qrcode = img[y-5:y+h+5, x-5:x+w+5]
cv2.imwrite("localized_qrcode.jpg", img)
# cv2.imshow("Image", img)
# cv2.imshow("Specific_qrcode", specific_qrcode)
cv2.imwrite("Specified_qrcode.jpg", specific_qrcode)
# key = cv2.waitKey(0)
# cv2.destroyAllWindows()

# for i in range(9750):
#     shutil.copy2('Specified_qrcode.jpg', 'Specified_qrcode' + str(i)
+ '.jpg')
```

Distance estimation

```
import cv2
import numpy as np
import tensorflow as tf
from keras.models import load_model

# Load classification model
# model = load_model('my_model.tflite')
model = tf.lite.Interpreter('my_model.tflite')
model.allocate_tensors()
input_details = model.get_input_details()
output_details = model.get_output_details()

# Load detection model
net = cv2.dnn.readNet('yolov3_testing.cfg',
'yolov3_training_last.weights')

# print(model.summary())
```

```
# Set up the camera
cap = cv2.VideoCapture(1)
distances = []
frame_count = 0
max_frames = 10

layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]
print("Layer names: ", layer_names)
print("Output kayers: ", output_layers)

# camera parameters
qrcode_size_cm = 24  # 9.5
focal_length_mm = 0.99 # 0.99 # 3.04  # 0.99
sensor_width_mm = 6.35# 4.62 # 6.35
pixel_size_mm = 0.00009  # 0.00000112 # 0.109

print("Starting detection loop")
while True:
   ret, frame = cap.read()
   height, width, channels = frame.shape
   if ret:
       frame_count += 1

   if frame_count >= max_frames:
       break

   # Perform object detection
   blob = cv2.dnn.blobFromImage(frame, 1/255, (416, 416), swapRB=True,
crop=False)
   net.setInput(blob)
   outs = net.forward(output_layers)

   # Loop over detected objects
   class_ids = []
   confidences = []
   boxes = []
   for out in outs:
       for detection in out:
           scores = detection[5:]
           class_id = np.argmax(scores)
           confidence = scores[class_id]
           if confidence > 0.5:
               # Extract ROI from input image
               x, y, w, h = detection[:4] * np.array([frame.shape[1],
frame.shape[0],
                                                      frame.shape[1],
frame.shape[0]])
               x, y, w, h = int(x - w / 2), int(y - h / 2), int(w),
```

```
int(h)
                if x < frame.shape[1] and y < frame.shape[0] and x+w <
frame.shape[1] and y+h < frame.shape[0]:
                    roi = frame[y : y+h, x : x+w]
                    if roi.size != 0:
                        ROI = cv2.resize(roi, (224, 224))

                    # Perform classification on ROI
                    input_tensor =
tf.keras.preprocessing.image.img_to_array(ROI)
                    input_tensor /= 255.
                    input_tensor = np.expand_dims(input_tensor, axis=0)

                    model.set_tensor(input_details[0]['index'],
input_tensor)
                    model.invoke()
                    predictions =
model.get_tensor(output_details[0]['index'])

                    # predictions = model.predict(input_tensor)
                    predicted_class = np.argmax(predictions)
                    print("Predicted class = ", predicted_class)

                    if predicted_class == 0:
                        object_width_pixels = w
                        # distance_m = (qrcode_size_cm *
focal_length_mm) / object_width_pixels
                        # distance_m = (focal_length_mm * qrcode_size_cm
* height) / (width * pixel_size_mm * width)
                        distance_m = (qrcode_size_cm * focal_length_mm)
/ (object_width_pixels * pixel_size_mm)
                        cv2.putText(frame, f"Distance: {distance_m:.2f}
m", (x, y),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)
                        cv2.rectangle(frame, (x, y), (x + w, y + h), (0,
255, 0), 2)
                    distances.append(distance_m)
                    print("Distance = ", distance_m)

    # Display frame
    cv2.imshow('Frame', frame)
    if cv2.waitKey(1) == ord('q'):
            break

sum = 0
average_distance = 0
for i in distances:
    sum += i
    average_distance = sum / len(distances)
```

```
print(average_distance)

# Release resources
cap.release()
cv2.destroyAllWindows()
```

YOLO3 testing configuration

```
[net]
# Testing
batch=8
subdivisions=1
# Training
# batch=64
# subdivisions=166
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 4000
policy=steps
steps=400000,450000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky
```

```
[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
```

```
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
```

```
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
```

```
activation=leaky

[shortcut]
from=-3
activation=linear

######################

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 6,7,8
anchors =
10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373
,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1


[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61



[convolutional]
batch_normalize=1
filters=256
size=1
```

```
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear
```

```
[yolo]
mask = 3,4,5
anchors =
10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373
,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1



[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36



[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
```

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 0,1,2
anchors =
10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373
,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
```

```
truth_thresh = 1
random=1
```

CPR based on AVR code

1. BIT_MATH.H

```
#define SET_BIT(VAR,BIT_NO)     VAR=(VAR)|(1<<BIT_NO)
#define CLR_BIT(VAR,BIT_NO)     VAR=(VAR)&(~(1<<BIT_NO))
#define TOGGLE_BIT(VAR,BIT_NO)  VAR=(VAR)^(1<<BIT_NO)
#define GET_BIT(VAR,BIT_NO)     (VAR>>BIT_NO)&1
```

2. STD_TYPES.H

```
typedef unsigned char u8;
typedef signed char s8;

typedef unsigned short int u16;
typedef signed short int s16;

typedef unsigned long int u32;
typedef signed long int s32;

typedef float f32;
typedef double f64;

#define NULL 0

#define OK   0
#define NOK  1

#define NULL_POINTER  2
```

3. DIO_program.c

```
#include "STD_TYPES.H"
#include "BIT_MATH.H"

#include "DIO_interface.h"
#include "DIO_private.h"
#include "DIO_register.h"
#include "DIO_config.h"

u8 DIO_u8SetPinValue(u8 Copy_u8Port,u8 Copy_u8Pin,u8 Copy_u8Value)
{
        u8 Local_u8ErrorState=0;
        if(Copy_u8Pin<=DIO_u8PIN7)
        {
              if(Copy_u8Value==DIO_u8PIN_HIGH)
              {
```

```
                        switch(Copy_u8Port)
                        {
                        case DIO_u8PORTA:SET_BIT(PORTA,Copy_u8Pin);break;
                        case DIO_u8PORTB:SET_BIT(PORTB,Copy_u8Pin);break;
                        case DIO_u8PORTC:SET_BIT(PORTC,Copy_u8Pin);break;
                        case DIO_u8PORTD:SET_BIT(PORTD,Copy_u8Pin);break;
                        default:Local_u8ErrorState=1;
                        }
                  }
                  else if(Copy_u8Value==DIO_u8PIN_LOW)
                  {
                        switch(Copy_u8Port)
                        {
                        case DIO_u8PORTA:CLR_BIT(PORTA,Copy_u8Pin);break;
                        case DIO_u8PORTB:CLR_BIT(PORTB,Copy_u8Pin);break;
                        case DIO_u8PORTC:CLR_BIT(PORTC,Copy_u8Pin);break;
                        case DIO_u8PORTD:CLR_BIT(PORTD,Copy_u8Pin);break;
                        default:Local_u8ErrorState=1;

                        }
                  }
                  else
                  {
                        Local_u8ErrorState=1;
                  }
            }
            else
            {
                  Local_u8ErrorState=1;
            }
            return Local_u8ErrorState;
}
u8 DIO_u8SetPortValue(u8 Copy_u8Port,u8 Copy_u8Value)
{
            u8 Local_u8ErrorState=0;
            switch(Copy_u8Port)
            {
            case DIO_u8PORTA:PORTA=Copy_u8Value;break;
            case DIO_u8PORTB:PORTB=Copy_u8Value;break;
            case DIO_u8PORTC:PORTC=Copy_u8Value;break;
            case DIO_u8PORTD:PORTD=Copy_u8Value;break;
            default:Local_u8ErrorState=1;
            }
            return Local_u8ErrorState;
}

u8 DIO_GetPinValue(u8 Copy_u8Port,u8 Copy_u8Pin,u8*Copy_pu8Value)
{
            u8 Local_u8ErrorState=0;
            if((Copy_pu8Value!=NULL) && (Copy_u8Pin<=DIO_u8PIN7))
```

```
        {
                switch(Copy_u8Port)
                {
                case
DIO_u8PORTA:*Copy_pu8Value=GET_BIT(PINA,Copy_u8Pin);break;
                case
DIO_u8PORTB:*Copy_pu8Value=GET_BIT(PINB,Copy_u8Pin);break;
                case
DIO_u8PORTC:*Copy_pu8Value=GET_BIT(PINC,Copy_u8Pin);break;
                case
DIO_u8PORTD:*Copy_pu8Value=GET_BIT(PIND,Copy_u8Pin);break;
                default:Local_u8ErrorState=1;
                }
        }
        else
        {
                Local_u8ErrorState=1;
        }

        return Local_u8ErrorState;
}
```

4. DIO_interface.h

```
#ifndef  DIO_INTERFACE_H_
#define  DIO_INTERFACE_H_

#define DIO_u8PIN_HIGH     1
#define DIO_u8PIN_LOW      0

#define DIO_u8PORT_HIGH    0xff
#define DIO_u8PORT_LOW     0

#define DIO_u8PIN0         0
#define DIO_u8PIN1         1
#define DIO_u8PIN2         2
#define DIO_u8PIN3         3
#define DIO_u8PIN4         4
#define DIO_u8PIN5         5
#define DIO_u8PIN6         6
#define DIO_u8PIN7         7

#define DIO_u8PORTA        0
#define DIO_u8PORTB        1
#define DIO_u8PORTC        2
#define DIO_u8PORTD        3


u8 DIO_u8SetPinValue(u8 Copy_u8Port,u8 Copy_u8Pin,u8 Copy_u8Value);
```

```
u8 DIO_u8SetPortValue(u8 Copy_u8Port,u8 Copy_u8Value);

u8 DIO_GetPinValue(u8 Copy_u8Port,u8 Copy_u8Pin,u8*Copy_pu8Value);


#endif
```

5. DIO_register.h

```
#ifndef  DIO_register_H_
#define  DIO_register_H_

#define  PORTA       *((volatile u8*)0x3b)

#define  PINA        *((volatile u8*)0x39)

#define  PORTB       *((volatile u8*)0x38)

#define  PINB        *((volatile u8*)0x36)

#define  PORTC       *((volatile u8*)0x35)

#define  PINC        *((volatile u8*)0x33)

#define  PORTD       *((volatile u8*)0x32)

#define  PIND        *((volatile u8*)0x30)

#endif
```

6. PORT_program.c

```
#include "STD_TYPES.h"

/*here config file must be before private file*/
#include "PORT_config.h"
#include "PORT_private.h"
#include "PORT_interface.h"
#include "PORT_register.h"

void PORT_voidInit(void)
{
         DDRA=PORTA_DIR;
         DDRB=PORTB_DIR;
         DDRC=PORTC_DIR;
         DDRD=PORTD_DIR;

         PORTA=PORTA_INITIAL_VALUE;
         PORTB=PORTB_INITIAL_VALUE;
         PORTC=PORTC_INITIAL_VALUE;
```

```
        PORTD=PORTD_INITIAL_VALUE;
}
```

7. PORT_interface.h

```
#ifndef  PORT_INTERFACE_H_
#define  PORT_INTERFACE_H_

void PORT_voidInit(void);

#endif
```

8. PORT_config.h

```
#ifndef PORT_CONFIG_H_
#define PORT_CONFIG_H_

#define PORTA_PIN0_DIR                1
#define PORTA_PIN1_DIR                1
#define PORTA_PIN2_DIR                0
#define PORTA_PIN3_DIR                0
#define PORTA_PIN4_DIR                0
#define PORTA_PIN5_DIR                0
#define PORTA_PIN6_DIR                0
#define PORTA_PIN7_DIR                0

#define PORTB_PIN0_DIR                1
#define PORTB_PIN1_DIR                1
#define PORTB_PIN2_DIR                0
#define PORTB_PIN3_DIR                0
#define PORTB_PIN4_DIR                0
#define PORTB_PIN5_DIR                0
#define PORTB_PIN6_DIR                0
#define PORTB_PIN7_DIR                1

#define PORTC_PIN0_DIR                0
#define PORTC_PIN1_DIR                0
#define PORTC_PIN2_DIR                0
#define PORTC_PIN3_DIR                0
#define PORTC_PIN4_DIR                0
#define PORTC_PIN5_DIR                0
#define PORTC_PIN6_DIR                0
#define PORTC_PIN7_DIR                0

#define PORTD_PIN0_DIR                0
#define PORTD_PIN1_DIR                1
#define PORTD_PIN2_DIR                0
#define PORTD_PIN3_DIR                0
#define PORTD_PIN4_DIR                0
#define PORTD_PIN5_DIR                1
#define PORTD_PIN6_DIR                0
```

```
#define PORTD_PIN7_DIR                0

/*choose :0 for floating if input, low if o/p
         :1 for pulled up if input, high if o/p */

#define PORTA_PIN0_INITIAL_VALUE      0
#define PORTA_PIN1_INITIAL_VALUE      0
#define PORTA_PIN2_INITIAL_VALUE      0
#define PORTA_PIN3_INITIAL_VALUE      0
#define PORTA_PIN4_INITIAL_VALUE      0
#define PORTA_PIN5_INITIAL_VALUE      0
#define PORTA_PIN6_INITIAL_VALUE      0
#define PORTA_PIN7_INITIAL_VALUE      0

#define PORTB_PIN0_INITIAL_VALUE      0
#define PORTB_PIN1_INITIAL_VALUE      0
#define PORTB_PIN2_INITIAL_VALUE      1
#define PORTB_PIN3_INITIAL_VALUE      0
#define PORTB_PIN4_INITIAL_VALUE      0
#define PORTB_PIN5_INITIAL_VALUE      0
#define PORTB_PIN6_INITIAL_VALUE      0
#define PORTB_PIN7_INITIAL_VALUE      0

#define PORTC_PIN0_INITIAL_VALUE      0
#define PORTC_PIN1_INITIAL_VALUE      0
#define PORTC_PIN2_INITIAL_VALUE      0
#define PORTC_PIN3_INITIAL_VALUE      0
#define PORTC_PIN4_INITIAL_VALUE      0
#define PORTC_PIN5_INITIAL_VALUE      0
#define PORTC_PIN6_INITIAL_VALUE      0
#define PORTC_PIN7_INITIAL_VALUE      0

#define PORTD_PIN0_INITIAL_VALUE      0
#define PORTD_PIN1_INITIAL_VALUE      0
#define PORTD_PIN2_INITIAL_VALUE      1
#define PORTD_PIN3_INITIAL_VALUE      1
#define PORTD_PIN4_INITIAL_VALUE      0
#define PORTD_PIN5_INITIAL_VALUE      0
#define PORTD_PIN6_INITIAL_VALUE      0
#define PORTD_PIN7_INITIAL_VALUE      0

#endif
```

9. PORT_private.h

```
#ifndef PORT_PRIVATE_H_
#define PORT_PRIVATE_H_

#define
CONC(b7,b6,b5,b4,b3,b2,b1,b0)          CONC_HELPER(b7,b6,b5,b4,b3,b2,b1,b0
```

```
)
#define  CONC_HELPER(b7,b6,b5,b4,b3,b2,b1,b0)   0b##b7##b6##b5##b4##b3##b2
##b1##b0

#define
PORTA_DIR                CONC(PORTA_PIN7_DIR,PORTA_PIN6_DIR,PORTA_PIN5_DIR
,PORTA_PIN4_DIR,PORTA_PIN3_DIR,PORTA_PIN2_DIR,PORTA_PIN1_DIR,PORTA_PIN0_DI
R)
#define
PORTB_DIR                CONC(PORTB_PIN7_DIR,PORTB_PIN6_DIR,PORTB_PIN5_DIR
,PORTB_PIN4_DIR,PORTB_PIN3_DIR,PORTB_PIN2_DIR,PORTB_PIN1_DIR,PORTB_PIN0_DI
R)
#define
PORTC_DIR                CONC(PORTC_PIN7_DIR,PORTC_PIN6_DIR,PORTC_PIN5_DIR
,PORTC_PIN4_DIR,PORTC_PIN3_DIR,PORTC_PIN2_DIR,PORTC_PIN1_DIR,PORTC_PIN0_DI
R)
#define
PORTD_DIR                CONC(PORTD_PIN7_DIR,PORTD_PIN6_DIR,PORTD_PIN5_DIR
,PORTD_PIN4_DIR,PORTD_PIN3_DIR,PORTD_PIN2_DIR,PORTD_PIN1_DIR,PORTD_PIN0_DI
R)

#define  PORTA_INITIAL_VALUE     CONC(PORTA_PIN7_INITIAL_VALUE,PORTA_PIN6_
INITIAL_VALUE,PORTA_PIN5_INITIAL_VALUE,PORTA_PIN4_INITIAL_VALUE,PORTA_PIN3
_INITIAL_VALUE,PORTA_PIN2_INITIAL_VALUE,PORTA_PIN1_INITIAL_VALUE,PORTA_PIN
0_INITIAL_VALUE)
#define  PORTB_INITIAL_VALUE     CONC(PORTB_PIN7_INITIAL_VALUE,PORTB_PIN6_
INITIAL_VALUE,PORTB_PIN5_INITIAL_VALUE,PORTB_PIN4_INITIAL_VALUE,PORTB_PIN3
_INITIAL_VALUE,PORTB_PIN2_INITIAL_VALUE,PORTB_PIN1_INITIAL_VALUE,PORTB_PIN
0_INITIAL_VALUE)
#define  PORTC_INITIAL_VALUE     CONC(PORTC_PIN7_INITIAL_VALUE,PORTC_PIN6_
INITIAL_VALUE,PORTC_PIN5_INITIAL_VALUE,PORTC_PIN4_INITIAL_VALUE,PORTC_PIN3
_INITIAL_VALUE,PORTC_PIN2_INITIAL_VALUE,PORTC_PIN1_INITIAL_VALUE,PORTC_PIN
0_INITIAL_VALUE)
#define  PORTD_INITIAL_VALUE     CONC(PORTD_PIN7_INITIAL_VALUE,PORTD_PIN6_
INITIAL_VALUE,PORTD_PIN5_INITIAL_VALUE,PORTD_PIN4_INITIAL_VALUE,PORTD_PIN3
_INITIAL_VALUE,PORTD_PIN2_INITIAL_VALUE,PORTD_PIN1_INITIAL_VALUE,PORTD_PIN
0_INITIAL_VALUE)

#endif
```

10. PORT_register.h

```
#ifndef  PORT_register_H_
#define  PORT_register_H_

#define  DDRA        *((volatile u8*)0x3a)
#define  DDRB        *((volatile u8*)0x37)
#define  DDRC        *((volatile u8*)0x34)
#define  DDRD        *((volatile u8*)0x31)
```

```
#define  PORTA       *((volatile u8*)0x3b)
#define  PORTB       *((volatile u8*)0x38)
#define  PORTC       *((volatile u8*)0x35)
#define  PORTD       *((volatile u8*)0x32)
#endif
```

11. USART_program.c

```
#include "STD_TYPES.H"
#include "BIT_MATH.H"
#include <stdlib.h>

#include "USART_private.h"
#include "USART_config.h"
#include "USART_interface.h"
#include "USART_register.h"

void USART_INIT(void)
{
          u8 Local_u8USRC=0;
          SET_BIT(Local_u8USRC,UCSRC_URSEL); //To write value in UCSRC

          /*use only 1 stop bit*/
          CLR_BIT(Local_u8USRC,UCSRC_USBS);

          /*select 8 data bits*/
          SET_BIT(Local_u8USRC,UCSRC_UCSZ0);
          SET_BIT(Local_u8USRC,UCSRC_UCSZ1);
          CLR_BIT(Local_u8USRC,UCSRB_UCSZ2);

          /*to select asynchronous mode*/
          CLR_BIT(Local_u8USRC,UCSRC_UMSEL);

          /*no parity selected*/
          CLR_BIT(Local_u8USRC,UCSRC_UPM0);
          CLR_BIT(Local_u8USRC,UCSRC_UPM1);

          UCSRC=Local_u8USRC;

          /*SET 9600 baud rate*/
          CLR_BIT(UCSRA,UCSRA_U2X);
          UBRRL=51;

          /*ENABLING transmission AND RECEIVING*/
          SET_BIT(UCSRB,UCSRB_TXEN);
          SET_BIT(UCSRB,UCSRB_RXEN);

}
void USART_voidSend(u8 Copy_u8Data)
{
```

```
        while((GET_BIT(UCSRA,UCSRA_UDRE))==0); //POLLING

        /*Clear The Flag*/
        SET_BIT(UCSRA, UCSRA_UDRE);

        UDR=Copy_u8Data;
}
u8 USART_u8Receive(void)
{
        while((GET_BIT(UCSRA,UCSRA_RXC))==0);

        /*Clear The RXC Flag*/
        SET_BIT(UCSRA, UCSRA_RXC);

        return UDR;
}

void USART_SEND_string(char *Copy_u8DataArray)
{
        int Local_u8Counter=0;
   while(Copy_u8DataArray[Local_u8Counter]!='\0')
   {
      USART_voidSend(Copy_u8DataArray[Local_u8Counter]);
      Local_u8Counter++;
   }
}
```

12. USART_interface.h

```
void USART_INIT(void);
void USART_voidSend(u8 Copy_u8Data);
u8 USART_u8Receive(void);
void USART_SEND_string(char *Copy_u8DataArray);
```

13. USART_private.h

```
#define BAUD 115200
#define BAUD_PRESCALE (((F_CPU / (BAUD * 16UL))) - 1)
```

14. USART_register.h

```
#ifndef USART_REGISTER_H_
#define USART_REGISTER_H_

#define UDR                     *((u8*volatile)0x2C)

#define UCSRA                   *((u8*volatile)0x2B)
#define UCSRA_RXC               7
#define UCSRA_TXC               6
#define UCSRA_UDRE              5
#define UCSRA_FE                4
```

```
#define UCSRA_DOR               3
#define UCSRA_PE                2
#define UCSRA_U2X               1
#define UCSRA_MPCM              0


#define UCSRB          *((u8*volatile)0x2A)
#define UCSRB_RXCIE             7
#define UCSRB_TXCIE             6
#define UCSRB_UDRIE             5
#define UCSRB_RXEN              4
#define UCSRB_TXEN              3
#define UCSRB_UCSZ2             2
#define UCSRB_RXB8              1
#define UCSRB_TXB8              0


#define UCSRC                   *((u8*volatile)0x40)
#define UCSRC_URSEL             7
#define UCSRC_UMSEL             6
#define UCSRC_UPM1              5
#define UCSRC_UPM0              4
#define UCSRC_USBS              3
#define UCSRC_UCSZ1             2
#define UCSRC_UCSZ0             1
#define UCSRC_UCPOL             0

#define UBRRL                   *((u8*volatile)0x29)

#endif
```

15. EXTI_program.c

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"


#include "EXTI_interface.h"
#include "EXTI_private.h"
#include "EXTI_register.h"
#include "EXTI_config.h"

/*global variable pointer to function (ISR function in main.c)*/
void (*EXTI_pvInt0Func)(void)=NULL;
void (*EXTI_pvInt1Func)(void)=NULL;
void (*EXTI_pvInt2Func)(void)=NULL;

void EXTI_voidInt0Init(void)
{
        /*SET SENSE CONTROL FOR INT0 TO FALLING EDGE*/
```

```
#if INT0_SENSE==LOW_LEVEL
        CLR_BIT(MCUCR,MCUCR_ISC00);
        CLR_BIT(MCUCR,MCUCR_ISC01);

#elif INT0_SENSE==ON_CHANGE
        SET_BIT(MCUCR,MCUCR_ISC00);
        CLR_BIT(MCUCR,MCUCR_ISC01);

#elif INT0_SENSE==FALLING_EDGE
        CLR_BIT(MCUCR,MCUCR_ISC00);
        SET_BIT(MCUCR,MCUCR_ISC01);

#elif INT0_SENSE==RISING_EDGE
        SET_BIT(MCUCR,MCUCR_ISC00);
        SET_BIT(MCUCR,MCUCR_ISC01);

#else
#error "wrong INT0_SENSE configuration options"   //or hash warning but
hash error stop the program
#endif

/*check peripheral interrupt enable intial state*/
#if INT0_INITIAL_STATE==ENABLED
        SET_BIT(GICR,GICR_INT0);
#elif INT0_INITIAL_STATE==DISABLED
        CLR_BIT(GICR,GICR_INT0);
#else
#error "wrong INT0_INITIAL_STATE configuration options"
#endif
}

void EXTI_voidInt1Init(void)
{
        /*SET SENSE CONTROL FOR INT0 TO FALLING EDGE*/
#if INT1_SENSE==LOW_LEVEL
        CLR_BIT(MCUCR,MCUCR_ISC10);
        CLR_BIT(MCUCR,MCUCR_ISC11);

#elif INT1_SENSE==ON_CHANGE
        SET_BIT(MCUCR,MCUCR_ISC10);
        CLR_BIT(MCUCR,MCUCR_ISC11);

#elif INT1_SENSE==FALLING_EDGE
        CLR_BIT(MCUCR,MCUCR_ISC10);
        SET_BIT(MCUCR,MCUCR_ISC11);

#elif INT1_SENSE==RISING_EDGE
        SET_BIT(MCUCR,MCUCR_ISC10);
        SET_BIT(MCUCR,MCUCR_ISC11);
```

```
#else
#error "wrong INT1_SENSE configuration options"   //or hash warning but
hash error stop the program
#endif

/*check peripheral interrupt enable initial state*/
#if INT1_INITIAL_STATE==ENABLED
        SET_BIT(GICR,GICR_INT1);
#elif INT1_INITIAL_STATE==DISABLED
        CLR_BIT(GICR,GICR_INT1);
#else
#error "wrong INT1_INITIAL_STATE configuration options"
#endif
}

void EXTI_voidInt2Init(void)
{
        /*SET SENSE CONTROL FOR INT2 TO FALLING EDGE*/
#if INT2_SENSE==FALLING_EDGE
        CLR_BIT(MCUCSR,MCUCSR_INT2);

#elif INT2_SENSE==RISING_EDGE
        SET_BIT(MCUCSR,MCUCSR_INT2);

#else
#error "wrong INT2_SENSE configuration options"   //or hash warning but
hash error stop the program
#endif

/*check peripheral interrupt enable initial state*/
#if INT2_INITIAL_STATE==ENABLED
        SET_BIT(GICR,GICR_INT2);
#elif INT2_INITIAL_STATE==DISABLED
        CLR_BIT(GICR,GICR_INT2);
#else
#error "wrong INT2_INITIAL_STATE configuration options"
#endif
}

u8 EXTI_u8Int0SetSenseControl(u8 Copy_u8Sense)
{
        u8 Local_u8ErrorState=OK;
        switch(Copy_u8Sense)
        {
        case LOW_LEVEL:CLR_BIT(MCUCR,MCUCR_ISC00);
CLR_BIT(MCUCR,MCUCR_ISC01); break;
        case ON_CHANGE:SET_BIT(MCUCR,MCUCR_ISC00);
CLR_BIT(MCUCR,MCUCR_ISC01); break;
        case FALLING_EDGE: CLR_BIT(MCUCR,MCUCR_ISC00);
SET_BIT(MCUCR,MCUCR_ISC01); break;
```

```
		case RISING_EDGE: SET_BIT(MCUCR,MCUCR_ISC00);
SET_BIT(MCUCR,MCUCR_ISC01); break;
		default:Local_u8ErrorState=NOK;
		}
		return Local_u8ErrorState;
}

u8 EXTI_u8Int0SetCallBack(void(*Copy_pvInt0Func)(void))
{
		u8 Local_u8ErrorStatus=OK;
		if(Copy_pvInt0Func!=NULL)
		{
			EXTI_pvInt0Func=Copy_pvInt0Func;
		}
		else
		{
			Local_u8ErrorStatus=NULL_POINTER;
		}
		return Local_u8ErrorStatus;
}

u8 EXTI_u8Int1SetCallBack(void(*Copy_pvInt1Func)(void))
{
		u8 Local_u8ErrorStatus=OK;
		if(Copy_pvInt1Func!=NULL)
		{
			EXTI_pvInt1Func=Copy_pvInt1Func;
		}
		else
		{
			Local_u8ErrorStatus=NULL_POINTER;
		}
		return Local_u8ErrorStatus;
}

u8 EXTI_u8Int2SetCallBack(void(*Copy_pvInt2Func)(void))
{
		u8 Local_u8ErrorStatus=OK;
		if(Copy_pvInt2Func!=NULL)
		{
			EXTI_pvInt2Func=Copy_pvInt2Func;
		}
		else
		{
			Local_u8ErrorStatus=NULL_POINTER;
		}
		return Local_u8ErrorStatus;
}
/*ISR of INT0*/
void __vector_1 (void) __attribute__((signal));
```

```
void __vector_1 (void)
{
	if(EXTI_pvInt0Func!=NULL)
	{
		EXTI_pvInt0Func();
	}
	else
	{
		/*do nothing*/
	}
}

/*ISR of INT1*/
void __vector_2 (void) __attribute__((signal));
void __vector_2 (void)
{
	if(EXTI_pvInt1Func!=NULL)
	{
		EXTI_pvInt1Func();
	}
	else
	{
		/*do nothing*/
	}
}

/*ISR of INT2*/
void __vector_3 (void) __attribute__((signal));
void __vector_3 (void)
{
	if(EXTI_pvInt2Func!=NULL)
	{
		EXTI_pvInt2Func();
	}
	else
	{
		/*do nothing*/
	}
}
```

16. EXTI_interface.h

```
#ifndef  EXTI_interface_H_
#define  EXTI_interface_H_

#define LOW_LEVEL    1
#define ON_CHANGE    2
#define FALLING_EDGE 3
#define RISING_EDGE  4
```

```
/*fn to set required sense control of int0 using prebuild configuration */
void EXTI_voidInt0Init(void);
void EXTI_voidInt1Init(void);
void EXTI_voidInt2Init(void);

/*fn to set required sense control of int0 using postbuild configuration
*/
u8 EXTI_u8Int0SetSenseControl(u8 Copy_u8Sense);

u8 EXTI_u8Int1SetSenseControl(u8 Copy_u8Sense);

u8 EXTI_u8Int2SetSenseControl(u8 Copy_u8Sense);

u8 EXTI_u8IntEnable(u8 Copy_u8Int);

u8 EXTI_u8IntDisable(u8 Copy_u8Int);

u8 EXTI_u8Int0SetCallBack(void(*Copy_pvInt0Func)(void));
u8 EXTI_u8Int1SetCallBack(void(*Copy_pvInt1Func)(void));
u8 EXTI_u8Int2SetCallBack(void(*Copy_pvInt2Func)(void));


#endif
```

17. EXTI_config.h

```
#ifndef  EXTI_CONFIG_H_
#define  EXTI_CONFIG_H_

/*OPTIONS
 * 1-LOW_LEVEL
 * 2-ON_CHANGE
 * 3-FALLING_EDGE
 * 4-RISING_EDGE
 * */
#define INT0_SENSE         FALLING_EDGE

/*OPTIONS
 * 1-ENABLED
 * 2-DISABLED
 * */
#define INT0_INITIAL_STATE ENABLED

#define INT1_SENSE         FALLING_EDGE
#define INT1_INITIAL_STATE ENABLED

#define INT2_SENSE         FALLING_EDGE
#define INT2_INITIAL_STATE ENABLED

#endif
```

18. EXTI_private.h

```
#ifndef  EXTI_PRIVATE_H_
#define  EXTI_PRIVATE_H_

#define ENABLED      1
#define DISABLED      2

#endif
```

19. EXTI_register.h

```
#ifndef  EXTI_register_H_
#define  EXTI_register_H_

#define GICR          *((volatile u8*)0x5B)
#define GICR_INT1           7
#define GICR_INT0           6
#define GICR_INT2           5

#define GIFR                *((volatile u8*)0X5A)

#define MCUCR         *((volatile u8*)0x55)
#define MCUCR_ISC11         3
#define MCUCR_ISC10         2
#define MCUCR_ISC01         1
#define MCUCR_ISC00      0

#define MCUCSR              *((volatile u8*)0x54)
#define MCUCSR_INT2        6
#endif
```

20. GIE_program.c

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"

#include "GIE_interface.h"
#include "GIE_register.h"

void GIE_voidEnable(void)
{
        SET_BIT(SREG,SREG_I);
}
void GIE_voidDisable(void)
{
        CLR_BIT(SREG,SREG_I);
}
```

21. GIE_interface.h

```
#ifndef  GIE_INTERFACE_H_
#define  GIE_INTERFACE_H_

void GIE_voidEnable(void);
void GIE_voidDisable(void);

#endif
```

22. TIMER_program.c

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"

#include "TIMER_interface.h"
#include "Timer_private.h"
#include "Timer_config.h"
#include "Timer_register.h"

static void (*TIMER0_pvCallBackFunc)(void)=NULL;

void TIMER_voidTimer1Init(void)
{
        /*setting pwm non inverting mode FAST PWM*/
        CLR_BIT(TCCR1A,TCCR1A_COM1A0);
        SET_BIT(TCCR1A,TCCR1A_COM1A1);

        /*set waveform generation mode for fast pwm*/
        CLR_BIT(TCCR1A,TCCR1A_WGM10);
        SET_BIT(TCCR1A,TCCR1A_WGM11);
        SET_BIT(TCCR1B,TCCR1B_WGM12);
        SET_BIT(TCCR1B,TCCR1B_WGM13);

        /*Set The Prescaler to be 8*/
        TCCR1B&=PRESCALER_MASK;
        TCCR1B|=DIVIDE_BY_8;
}
void TIMER1_voidSetICR(u16 Copy_u16TOP)
{
        ICR1=Copy_u16TOP;
}
void TIMER1_voidSetChannelCompMatch(u16 Copy_u16CompareMatch)
{
        OCR1A=Copy_u16CompareMatch;
}
u8 TIMER_voidTimer0SetCallBack(void (*Copy_CallBackFunc)(void))
{
        u8 Local_u8ErrorStatus=OK;
        /*Assign the function address to the global pointer to function*/
        if(Copy_CallBackFunc!=NULL)
        {
```

```
                TIMER0_pvCallBackFunc = Copy_CallBackFunc;
        }
        else
        {
                Local_u8ErrorStatus=NULL_POINTER;
        }
        return Local_u8ErrorStatus;
}
void TIMER0_voidSetCompMatchValue(u8 Copy_u8Value)
{
        OCR0=Copy_u8Value;
}

/*ISR for Timer0 Compare Match Event */
void __vector_10 (void) __attribute__((signal));
void __vector_10 (void)
{
        /*Check if the global pointer to function is changed or not*/
        if(TIMER0_pvCallBackFunc != NULL)
        {
                /*Execute the global pointer to function*/
                TIMER0_pvCallBackFunc();
        }
}
```

23. TIMER_interface.h

```
#ifndef  TIMER_INTERFACE_H_
#define  TIMER_INTERFACE_H_
void TIMER_voidTimer1Init(void);
void TIMER1_voidSetICR(u16 Copy_u16TOP);
void TIMER1_voidSetChannelCompMatch(u16 Copy_u16CompareMatch);
u8 TIMER_voidTimer0SetCallBack(void (*Copy_CallBackFunc)(void));
void TIMER0_voidSetCompMatchValue(u8 Copy_u8Value);
#endif
```

24. TIMER_register.h

```
#define  TCCR0            *((volatile u8 *)(0x53))      // Timer/Counter0
Control Register
#define  TCCR0_FOC0       7
   // Force On Compare
#define  TCCR0_WGM00      6                             // Wave
Generation Mode 0
#define  TCCR0_COM01      5                              // Compare match
output mode 1
#define  TCCR0_COM00      4                             // Compare match
output mode 0
#define  TCCR0_WGM01      3                             // Wave
Generation Mode 1
#define  TCCR0_CS02       2                             // Clock Select
```

```
02
#define  TCCR0_CS01        1                              // Clock Select
01
#define  TCCR0_CS00        0                              // Clock Select
00

#define  TCNT0             *((volatile u8 *)(0x52))       // Timer/Counter0
Register

#define  OCR0              *((volatile u8 *)(0x5C))       // Output
Compare0 Register

#define  TIMSK             *((volatile u8 *)(0x59))       // Timer0 Mask
Register
#define  TIMSK_TOIE0       0
   // Timer Overflow Interrupt Enable
#define  TIMSK_OCIE0       1                              // Timer On
Compare Interrupt Enable
#define              TIMSK_TOIE1         2
                     // Timer/Counter1 Overflow Interrupt Enable
#define              TIMSK_OCIE1B   3
               // Timer/Counter1 Output Compare B Interrupt Enable
#define              TIMSK_OCIE1A   4
               // Timer/Counter1 Output Compare A Interrupt Enable
#define              TIMSK_TICIE1   5
               // Timer/Counter1 Input Capture Interrupt Enable
#define  TIMSK_TOIE2       6
   // Timer/Counter2 Overflow Interrupt Enable
#define  TIMSK_OCIE2       7
   // Timer/Counter2 Output Compare Interrupt Enable

#define  TIFR              *((volatile u8 *)(0x58))            //
Timer/Counter0 Interrupt Flag
#define  TIFR_TOV0         0
   // Timer/Counter0 Overflow flag
#define  TIFR_OCF0         1                              // Timer/Counter0
Output Compare flag
#define              TIFR_TOV1           2
                     // Timer/Counter1 Overflow Flag
#define              TIFR_OCF1B          3
                     // Timer/Counter1 Output Compare B Flag
#define              TIFR_OCF1A          4
                     // Timer/Counter1 Output Compare A Flag
#define              TIFR_ICF1           5
                     // Timer/Counter1 Input Capture Flag
#define  TIFR_TOV2         6
   // Timer/Counter2 Overflow flag
#define  TIFR_OCF2         7                              // Timer/Counter2
Output Compare flag
```

```
/************************************** TIMER 1 REGISTERS
***********************************/
#define             TCCR1A                  *((volatile u8
*)(0x4F))           // Timer/Counter 1 Control Register A
#define  TCCR1A_WGM10           0
               // Wave Generation Mode 0 Bit
#define  TCCR1A_WGM11           1
               // Wave Generation Mode 1 Bit
#define  TCCR1A_FOC1B           2
               // Force Output Compare B Bit
#define  TCCR1A_FOC1A           3
               // Force Output Compare A Bit
#define  TCCR1A_COM1B0          4
               // Compare Output Mode Channel B 0 Bit
#define  TCCR1A_COM1B1          5
               // Compare Output Mode Channel B 1 Bit
#define  TCCR1A_COM1A0          6
               // Compare Output Mode Channel A 0 Bit
#define  TCCR1A_COM1A1          7
               // Compare Output Mode Channel A 1 Bit
#ifndef  TIMER_REGISTER_H_
#define  TIMER_REGISTER_H_

#define             TCCR1B                  *((volatile u8
*)(0x4E))           // Timer/Counter 1 Control Register B
#define  TCCR1B_CS10                        0
                         // Clock Selection 0 Bit
#define  TCCR1B_CS11                        1
                         // Clock Selection 1 Bit
#define  TCCR1B_CS12                        2
                         // Clock Selection 2 Bit
#define  TCCR1B_WGM12                   3
                    // Wave Generation Mode 2 Bit
#define  TCCR1B_WGM13                   4
                    // Wave Generation Mode 3 Bit
#define  TCCR1B_ICES1                   6
                    // Input Capture Edge Select
#define  TCCR1B_ICNC1                   7
                    // Input Capture Noise Canceler


#define             TCNT1L                      *((volatile u8
*)(0x4C))           // Timer/Counter 1 Low Register
#define             TCNT1H                      *((volatile u8
*)(0x4D))           // Timer/Counter 1 High Register
#define             TCNT1                       *((volatile u16
*)(0x4C))           // Timer/Counter 1 Register

#define             OCR1AL                      *((volatile u8
```

```
*)(0x4A))           // Output Compare 1 Channel A Low Register
#define             OCR1AH                        *((volatile u8
*)(0x4B))           // Output Compare 1 Channel A High Register

#define        OCR1A                          *((volatile u16
*)(0x4A))
// Output Compare 1 Channel A Register

#define             OCR1BL                        *((volatile u8
*)(0x48))           // Output Compare 1 Channel B Low Register

#define             OCR1BH                        *((volatile u8
*)(0x49))           // Output Compare 1 Channel B High Register

#define        OCR1B                          *((volatile u16
*)(0x48))
// Output Compare 1 Channel B Register

#define             ICR1L                         *((volatile u8
*)(0x46))           // Input Capture 1 Low Register

#define             ICR1H                         *((volatile u8
*)(0x47))           // Input Capture 1 High Register

#define  ICR1                                     *((volatile u16
*)(0x46))
// Input Capture 1 Register

#define PRESCALER_MASK     0b11111000
#define DIVIDE_BY_8        2

#endif
```

25. Main.c

```
#include "STD_TYPES.H"
#include "BIT_MATH.H"

#include "DIO_interface.h"
#include "USART_interface.h"
#include "PORT_interface.h"
#include "TIMER_interface.h"
#include <util/delay.h>
#include "EXTI_interface.h"
#include "GIE_interface.h"
void INT0_ISR(void);
void INT1_ISR(void);
void servo_rotate(u8 angle);
u16 Local_u16ON_Off_DCMotor;
u16 Local_u16RotatingServo;
```

```
void main(void)
{
	u8 data=0;

	PORT_voidInit();
	USART_INIT();
	TIMER_voidTimer1Init();
	TIMER1_voidSetICR(20000);
	EXTI_voidInt0Init();
	EXTI_voidInt1Init();
	EXTI_u8Int0SetCallBack(&INT0_ISR);
	EXTI_u8Int1SetCallBack(&INT1_ISR);

	GIE_voidEnable();

	while(1)
	{
		data=USART_u8Receive();
		/*if the drone reach patient's location RPI send 1 to
Bluetooth module*/
		if (data=='1')
		{
			/*rotate servo motor 90 degree*/
			servo_rotate(90);
			_delay_ms(3000);


   DIO_u8SetPinValue(DIO_u8PORTA,DIO_u8PIN1,DIO_u8PIN_HIGH);

			/*turn DC motor on for CPR action*/

   DIO_u8SetPinValue(DIO_u8PORTB,DIO_u8PIN7,DIO_u8PIN_HIGH);

		}

		/*if patient's pulse return RPI sends 2 to Bluetooth */
		if (data=='2')
		{

   DIO_u8SetPinValue(DIO_u8PORTA,DIO_u8PIN1,DIO_u8PIN_LOW);

			/*turn off DC motor*/

   DIO_u8SetPinValue(DIO_u8PORTB,DIO_u8PIN7,DIO_u8PIN_LOW);
		}

	}

}
```

```
/*INT0 connected to switch1 if pressed turn off DC motor*/
void INT0_ISR()
{       TOGGLE_BIT(Local_u16ON_Off_DCMotor,0);

   DIO_u8SetPinValue(DIO_u8PORTB,DIO_u8PIN7,Local_u16ON_Off_DCMotor);
        _delay_ms(10);

}

/*INT1 connected to switch2 if pressed rotate servo motor to 90 then if
pressed again rotate it to 0 degree*/
void INT1_ISR()
{
        TOGGLE_BIT(Local_u16RotatingServo,0);
             if(Local_u16RotatingServo==0)
             {
                  servo_rotate(90);

             }
             if(Local_u16RotatingServo==1)
             {
                  servo_rotate(0);

             }
             _delay_ms(10);

}

void servo_rotate(u8 angle)
{
        // Convert angle to pulse width value
   u8 pulse_width = (angle * 10) + 500;

   // Generate PWM signal on SERVO_PIN using Timer/Counter1
    TIMER1_voidSetChannelCompMatch(pulse_width);

   // Delay to allow the servo to reach the desired position
   _delay_ms(15);
}
```

Server code

```
import math
import sys
import time
import os
import smtplib
import time
from time import sleep
from Adafruit_IO import MQTTClient, Client, Feed, Data
```

```
smtpUser = 'user1.python@gmail.com'
smtpPass = 'hwwa fgzb pkag lifk'
toAdd = ['hendegypt7@gmail.com']
fromAdd = smtpUser
ADAFRUIT_IO_KEY = 'aio_fXRG98LfedlipeeqGZaUu5mvztJu'

ADAFRUIT_IO_USERNAME = 'hagerahmed'

current_state = 0

a, b = 0, 0


def connected(client):
    print('Connected to Adafruit IO!  Listening for changes...')
    client.subscribe('loc-lng')
    client.subscribe('loc-lat')
    client.subscribe('drone-deploy')


def disconnected(client):
    print('Disconnected from Adafruit IO!')
    sys.exit(1)
def message(client, feed_id, payload):
    global current_state, a, b
    if feed_id == 'drone-deploy':
        current_state = payload
    elif feed_id == 'loc-lng':
        a = payload
        print("Feed {0} received new value: {1}".format(feed_id, a))
    elif feed_id == 'loc-lat':
        b = payload
        print("Feed {0} received new value: {1}".format(feed_id, b))


client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
client.on_connect = connected
client.on_disconnect = disconnected
client.on_message = message

client.connect()
client.loop_background()
while True:
    if current_state == '1':
        R = 6371000
        # lat ==> b, long ==> a
        x = R * math.cos(float(b)) * math.cos(float(a))
        y = R * math.cos(float(b)) * math.sin(float(a))
        subject = 'Sudden Cardiac Arrest Case'
```

```
        header = "To : " + str(toAdd) + "\n" + "From : " + str(fromAdd) +
"\n" + "Subject: " +  str(subject)
        body= "there is a sudden cardiac arrest case located in:
https://www.google.com/maps/dir//"+str(b)+","+str(a)+" \nthat was happened
at"+ time.ctime()
        print (header + '\n' + body)

        s= smtplib.SMTP('smtp.gmail.com',587)
        s.ehlo()
        s.starttls()
        s.ehlo()

        s.login(smtpUser , smtpPass)
        s.sendmail(fromAdd, toAdd, header + '\n\n' + body)
        # s.quit()
        # sys.exit(1)
        break
```

Drone control code

```
from dronekit import connect,VehicleMode
,LocationGlobalRelative,APIException, LocationGlobal, Command
import time
import bluetooth
import socket
import math
import argparse
from pymavlink import mavutil
import server5
import final_pathPlanning as path
#from final_pathPlanning import get_start_point

#################functions#############################

def connectMyCopter():
    parser = argparse.ArgumentParser(description='commands')
    parser.add_argument('--connect')
    args = parser.parse_args()
    connection_string=args.connect
    if not connection_string:
        import dronekit_sitl
        sitl = dronekit_sitl.start_default()
        connection_string=sitl.connection_string()

    vehicle = connect(connection_string,baud=57600,wait_ready=True)
    return vehicle

def arm_and_takeoff(aTargetAltitude):
    while vehicle.is_armable==False:
```

```
        print("waiting for vehicle to be armable")
        time.sleep(1)

    vehicle.mode=VehicleMode("GUIDED")
    while vehicle.mode!="GUIDED":
        print("Waiting for vehicle to enter GUIDE mode")
        time.sleep(1)


    vehicle.armed=True
    while vehicle.armed==False:
        print("waiting for vehicle to be armed")
        time.sleep(1)

    vehicle.simple_takeoff(aTargetAltitude)

    while True:
        print("Current Altitude :
%d"%vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt >=
aTargetAltitude*0.95:
            break
        time.sleep(1)
    print("Target Altitude reached!!")
    return None


def send_local_ned_velocity(vx,vy,vz):
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,  #time boot ms not used
        0,0,    #target system, target component
        mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED,  # frame
        0b0000111111000111, #bit mask only speeds enabled
        0,0,0,    #position (not used)
        vx,vy,vz, #velocities  in m/s
        0,0,0,      #accelerations (not usable yet?!)(not supported yet,
ignoered in GCS Mavlink)
        0,0)        #yaw,yaw rate (not supported yet, ignoered in GCS
Mavlink)
    vehicle.send_mavlink(msg)
    vehicle.flush()


def send_global_ned_velocity(vx,vy,vz):
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,  #time boot ms not used
        0,0,    #target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED,  # frame
        0b0000111111000111, #bit mask only speeds enabled
        0,0,0,    #position (not used)
```

```
        vx,vy,vz, #velocities  in m/s
        0,0,0,      #accelerations (not usable yet?!)(not supported yet,
ignoered in GCS Mavlink)
        0,0)        #yaw,yaw rate (not supported yet, ignoered in GCS
Mavlink)
    vehicle.send_mavlink(msg)
    vehicle.flush()


def goto_position_target_global_int(aLocation):
    """
    Send SET_POSITION_TARGET_GLOBAL_INT command to request the vehicle
fly to a specified LocationGlobal.

    For more information see:
https://pixhawk.ethz.ch/mavlink/#SET_POSITION_TARGET_GLOBAL_INT

    See the above link for information on the type_mask (0=enable,
1=ignore).
    At time of writing, acceleration and yaw bits are ignored.
    """
    msg = vehicle.message_factory.set_position_target_global_int_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, # frame
        0b0000111111111000, # type_mask (only speeds enabled)
        aLocation.lat*1e7, # lat_int - X Position in WGS84 frame in 1e7
* meters
        aLocation.lon*1e7, # lon_int - Y Position in WGS84 frame in 1e7
* meters
        aLocation.alt, # alt - Altitude in meters in AMSL altitude, not
WGS84 if absolute or relative, above terrain if GLOBAL_TERRAIN_ALT_INT
        0, # X velocity in NED frame in m/s
        0, # Y velocity in NED frame in m/s
        0, # Z velocity in NED frame in m/s
        0, 0, 0, # afx, afy, afz acceleration (not supported yet,
ignored in GCS_Mavlink)
        0, 0)    # yaw, yaw_rate (not supported yet, ignored in
GCS_Mavlink)
    # send command to vehicle
    vehicle.send_mavlink(msg)

def goto_position_target_local_ned(north,east,down):
    # working , relative to the north of the drone the drone has to move
its north alot and around it self
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,  #time boot ms not used
        0,0,    #target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED,  # frame
        0b0000111111111000, #bit mask only speeds enabled
```

```
        north,east,down,    #position (not used)
        0,0,0, #velocities  in m/s
        0,0,0,      #accelerations (not usable yet?!)(not supported yet,
ignoered in GCS Mavlink)
        0,0)        #yaw,yaw rate (not supported yet, ignoered in GCS
Mavlink)
    vehicle.send_mavlink(msg)
    vehicle.flush()

def get_location_metres(original_location, dNorth, dEast):

    earth_radius = 6378137.0 # radius of "spherical" earth
    #coordinate  offsets in radians
    dLat = dNorth/earth_radius
    dLon =
dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position on decimal degree
    newlat=original_location.lat + (dLat * 180/math.pi)
    newlon=original_location.lon + (dLon * 180/math.pi)
    if type(original_location) is LocationGlobal :
        targetlocation=LocationGlobal(newlat,newlon,original_location.al
t)
    elif type(original_location) is LocationGlobalRelative:
        targetlocation=LocationGlobalRelative(newlat,newlon,original_loc
ation.alt)
    else:
        raise Exception("Invalid Location Object passed")

    return targetlocation

def get_distance_metres(aLocation1,aLocation2):

    dLat = aLocation2.lat - aLocation1.lat
    dLon = aLocation2.lon - aLocation1.lon
    return math.sqrt((dLat*dLat) + (dLon*dLon)) * 1.113195e5

def add_new_path_mission(patharray):

    cmds = vehicle.commands

    print("clear any existing commands")
    cmds.clear()

    print(" Define / add new commands")
    # add new commands

    # Add MAV_CMD_NAV_TAKEOFF command. this is ignored if the vehicle is
in the AIR , the 10 here is the alt of the drone
    cmds.add(Command( 0, 0, 0,
```

```
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavutil.mavlink.MAV_CMD_N
AV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, 2))

   #define the array of waypoints of the path (MAV_CMD_NAV_WAYPOINT)
commands
   for i in range(len(patharray)):
       cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0,0,0,0,0,0, patharray[i].lat
,patharray[i].lon, 2))

   # add dummy waypoint at the last point (let us know when have
reached destination)
   cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0,0,0,0,0,0, patharray[i].lat
,patharray[i].lon, 2))

   print(" Upload the new commands to vehicle")
   cmds.upload()

def distance_to_cuuent_waypoint():

   nextwaypoint = vehicle.commands.next
   if nextwaypoint == 0:
       return None
   missionitem = vehicle.commands[nextwaypoint-1]  #commands are zero
indexed
   lat = missionitem.x
   lon = missionitem.y
   alt = missionitem.z
   targetWaypointLocation = LocationGlobalRelative(lat,lon,alt)
   distancetopoint = get_distance_metres(vehicle.location.global_frame,
targetWaypointLocation)
   return distancetopoint

vehicle=connectMyCopter()

def goto(dNorth ,dEast, gotoFunction = vehicle.simple_goto):

   currentLocation = vehicle.location.global_relative_frame
   targetLocation = get_location_metres(currentLocation,dNorth,dEast)
   targetDistance = get_distance_metres(currentLocation,targetLocation)
   gotoFunction(targetLocation)

   #print "DEBUG: targetLocation: %s" % targetLocation
   #print "DEBUG: targetLocation: %s" % targetDistance
   firstremainingdistance =
get_distance_metres(vehicle.location.global_relative_frame,
targetLocation)
```

```
   while vehicle.mode.name == "GUIDED":
       if firstremainingdistance > 17 : #if the distance is less than
12 meters , then we the test distance is going to be 1 meter only

           remainingDistance =
get_distance_metres(vehicle.location.global_relative_frame,
targetLocation)
           print("Distance  to target : ", remainingDistance)
           if remainingDistance <= targetDistance*0.08:
               print("Target reached")
               break
           time.sleep(2) # gets location every 2 seconds
       else :
           remainingDistance =
get_distance_metres(vehicle.location.global_relative_frame,
targetLocation)
           print("Distance  to target : ", remainingDistance)
           if remainingDistance <= 1.5:
               print("Target reached")
               break
           time.sleep(2)

##################### main ##########################
#metresarray=[[1,1],[2,3],[4,5],[5,6],[7,5],[6,3]]
#metresarray=[[1.0, 1.0], [1.8254639257468783, 1.7147918562740645],
[2.7135909148720443, 2.4074717842158124], [4.086489639505206,
3.4843458877380256], [5.1619968764705915, 4.819897995538652],
[6.219899497003026, 5.60869164344931], [7.287596584403505,
6.102307481110323], [9.028230314196062, 6.547525687800578],
[10.92720841017478, 6.638477160792514], [12.804782020716104,
6.854920347677928], [14.506721879230009, 7.117327648435041], [16.0,
7.5]]
#get_start_point(vehicle.location.global_relative_frame)
metresarray = path.distance_togo
def path_location_array(metrearray,homeLocation =
vehicle.location.global_relative_frame ):
   locationarray=list()
   for i in range(len(metrearray)):
       if i == 0:

           locationarray.append(get_location_metres(homeLocation,metrea
rray[i][0],metrearray[i][1])) # first location after the home location
of the drone
       else:
           locationarray.append(get_location_metres(locationarray[i-
1],metrearray[i][0],metrearray[i][1]))
   return locationarray
print(metresarray)
locationarray=path_location_array(metresarray)
```

```
print("the defined path according to the given meters array is :" )
for j in locationarray :
   print(j.lat , j.lon)

print("creat a new mission for current location")
add_new_path_mission(locationarray)
arm_and_takeoff(2)


print("starting mission")
# reset mission set to first (0) waypoint
vehicle.commands.next=0

# set mode to auto to start mission
vehicle.mode = VehicleMode("AUTO")

start_position = vehicle.location.global_relative_frame
latitude = vehicle.location.global_relative_frame.lat
longitude = vehicle.location.global_relative_frame.lon
R = 6371000
x = R * math.cos(float(latitude)) * math.cos(float(longitude))
y = R * math.cos(float(latitude)) * math.sin(float(longitude))
initial_location=(x/1000,y/1000)

print ("the starting point is:",start_position)
while True:
   nextwaypoint = vehicle.commands.next
   print("Distance to waypoint (%s): %s " % (nextwaypoint,
distance_to_cuuent_waypoint()))
   if nextwaypoint == len(locationarray)+1 :
       print("exit 'standerd' missionwhen start heading to final dummy
way point")
       break;
   time.sleep(1)
vehicle.mode = VehicleMode("GUIDED")
import  dist_estimate
# Define the distance to qr code
distance = dist_estimate.average_distance
# Define the drone's speed
speed = 0.5
# Move the drone to the position
current_position = vehicle.location.global_relative_frame
print ("staring a new misssion to: ",current_position)
target_position = get_location_metres(current_position,-distance,0)
vehicle.simple_goto(target_position, groundspeed=speed)
time.sleep(3)
current_position = vehicle.location.global_relative_frame
print ("staring a new misssion to: ",current_position)
print ("Landing !!")
```

```
vehicle.mode = VehicleMode("LAND")


bd_addr = "00:21:13:00:6C:43"
# Create a Bluetooth socket and connect to the device
port = 1
sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
sock.connect((bd_addr, port))
currentt_state=0
previous_state=1
while True:
   time.sleep(5)
   if server5.current_state == "1":
    if  previous_state != currentt_state:
       mission_completed="1"
       sock.send(mission_completed)
       print("CPR working")
       time.sleep(1)
       currentt_state=1
   elif server5.current_state == "0":
       for i in range(3):
            sock.send("2")
            time.sleep(2)
       print("Stopping CPR Function")
       time.sleep(3)
       break
sock.close()
```