CHAPTER ONE
INTRODUCTION

1.1. Introduction:

The rising of human demand for machines capable of doing his work autonomously and precisely in most of his modern life's needs, manufacturing, aviation, military and transportation...etc. has inspired scientists and engineers to develop autonomous control systems.Autonomous control has become a fascinating field for its endless applications which work in difficult areas that need a very precise and careful control scenario[1].

One of the promising applications of the autonomous control systems is the quadcopter. The quadcopter is a multi-rotor helicopter that is lifted and propelled by four rotors. It represents an excellent platform for the autonomous control because it is a small, agile and maneuverable robot.

Quadcopterunmanned aerial vehicles areused in many civilian and military applications. They are considered as the best solution for intelligence, surveillance and reconnaissance by military and law enforcement agencies, as well as their use in suicidal missions. In addition they have lots of civilian applications such as search and rescue missions, precision agriculture/remote farming, inspection and transportation [2].

1.2. Problem Statement:

There are many limitations in the current methodof controlling the quadcopter manually by human using radio controller (RC). RC has a limited range which makes it impossible to control the quadcopter out of

that range. Human ability to control the quadcopter is limited especially when dealing with a dangerous situation or flying in a complex environment.

1.3. Proposed Solution:

The advances in the capabilities of microcomputer boards as well as the growing in the development of affordable and high precision state sensors such as inertial measurement unit and GPS sensor made it possible to develop an autonomous control system for the quadcopter that is able to drive the quadcopter in higher range and precisely in complex dynamicenvironments.

1.4. Methodology:

1.4.1. Quadcopter Dynamic:

The quadcopter consists of four rotors that are mounted at the end of two perpendicular axes. Rotors at opposite ends of an arm turn in the same direction while rotors on a perpendicular axis rotate in the opposite direction. When all four motors are spinning at the same speed, the rotors create thrust that lifts the quadcopter into the air. As there are pairs of rotors spinning in opposite directions, the torque produced in each direction around the yaw axis cancels out and the yaw angle remains constant[2].

1.4.2The Control Architecture:

- Perception and Data Fusion:

There are different sensors used to give the quadcopter information about the environment. The first one is IMU (9 DOF) motion sensors, their readings are integrated (fused) by Kalman filter to measure the orientation

of the robot in form of three angles (yaw, Roll and Pitch). The second one is the GPS sensor which is used to localize the position of the robot in outdoor environments. The last one is the ranging sensors which are used to measure distances from the robot to the nearby objects [3].

- Localization:

The localization process consists of a number of steps that use the environment to update the position of the robot. Since the odometery (the dynamic model) of the robot is often erroneous SLAMdoesn't rely directly on it. It also usessensors' measurements to correct the position of therobot. This is accomplished by extracting featuresfrom the environment and re-observing them while the robot is exploring the environment [3].

- Motion Planning:

Motion planning breaks down a desired movement task (goal state) into discrete motions that satisfy movement. Motion planning first calculates the direction to goal. After that it checks whether this direction is clean from potential collision or not; if it is clean then it sends this direction to the motion controller to drive the quadcopter toward that direction, but if it is not the motion planning finds an alternative path that drives the quadcopter away from collision [3].

- Motion Control:

The motion control calculates a suitable motion to follow every sub-goal. Then sends these calculations to the Electronic Speed Controller (ESC) of each motor as PWM signal to alter the speed of the motors in a way making it perform the required motion [3].

PID controllers are responsible for the calculation. The control scheme is the cascaded control where the outer loop is a position controller and the inner

loop is an attitude controller. The position controller receives the current position from the GPS sensor and the sub-goal position from motion planning then it computes the angles for the attitude controller which is then try to stabilize at these angels [3].

- On-board Computer:

Raspberry Pi 3 is used as the on-board computer; it has a built-in Wi-Fi that will be used to receive the goal state. Pythonis considered as the main programming language in the project.

1.5. Aim and Objectives:

The main aim of this project is to design an autonomous quadcopter that can go from a current location to a desired location autonomously and safely. Where the stated objectives are:

- To design a navigation system capable of acting rationally within complex environments.
- To develop a control system using a cascaded PID controller.
- To design a ground station for sending goal locations to the quadcopter as well as making the user able to monitor the quadcopter flight data.
- To implement a prototype model for the system using Raspberry Pi 3 that has high computational power and tests its ability to process a number of autonomy algorithms gracefullyto satisfactorilydrive the model.
- To simulate the different autonomy components in a quadcopter on Virtual Robot Experimentation Platform (V-REP).

1.6. Research Outlines:

Chapter one is an introduction that gives a background about the project, its aims and objectives, the problem statement and proposed solutions. It also gives a brief description on how to achieve those goals in the methodology.

Chapter two is the literature review that first gives an overall look on the mobility management schemes. The second part of the chapter is related works which include the analysis of several papers that were in the field of mobility management highlighting the pros and cons of each.

Chapter three is the system design (Methodology) contains all the methods and steps in great details that were undertaken to achieve the project's objectives.

Chapter four is results include simulation parameters, a discussion of the simulation and the resulted outcome from it, which are also justified.

Chapter five the conclusion and recommendation is the achieved goals from the project and the recommendations for future studies.

CHAPTER TWO

LITERATURE REVIEW

2.1Background:

This section introduces the basic knowledge needed before starting the dissertation. Here the used concepts throughout this document are explained in details. The concepts of the autonomous navigation in dynamic environments are explained first which are: perception, localization and mapping, planning, and navigation. After thatthe autonomous quadcopter will be explained too.

2.1.1 Autonomous navigating in dynamic environments:

Autonomous mobile robots design follows four-layer architecture design paradigm [1]which ensures a modular architecture by default. A layer is defined as repository of software and hardware components that serves a very well defined function; these layers are:

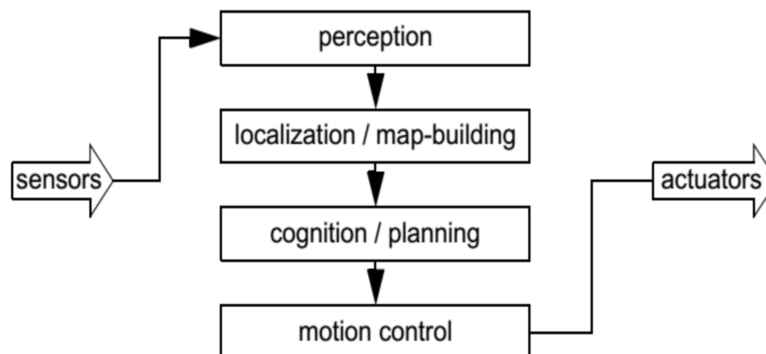


Figure 2-1: Conceptual architecture block diagram

The system is designed in a layer-based architecture so that each layer contains one or more algorithm that do a specific function. This way if we change the algorithm used in one layer the other layer would not be affected. Belowis a detailed description of the different layers.

- **Perception**:

One of the most important tasks of the autonomous system of any kind is to acquire knowledge about its environment. This is done by taking measurements using various sensors andthen extracting meaningful information from those measurements[4].

There are a wide variety of sensors used in mobile robots. Some sensors are used to measure simple values like the internal temperature of a robot's electronics or the rotational speed of the motors. Other, more sophisticated sensors can be used to acquire information about the robot's environment or even to directly measure a robot's global position. We classify sensors using two important functional axes;proprioceptive/exteroceptive and passive/active. The proprioceptive sensors measure values internal to the system (robot) while exteroceptive sensors acquire information from the robot's environment; for example, distance measurements, light intensity, sound amplitude. Hence exteroceptive sensor measurements are interpreted by the robot in order to extract meaningful environmental features.

Table2-1: Sensor classification

General classification (typical use)	Sensor Sensor system	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers Optical batteries Noncontact proximity sensor	EC EC EC	P A A
Wheel/ motor sensors (wheel/motor speed and position)	Burch encoder Potentiometer Optical encoders capacitive encoders	PC PC PC PC	P P A A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes inclinometer	EC PC EC	P P A/P
Ground-based beacons(localization in a fixed reference frame)	GPS Active optical or RF beacons Active ultrasonic beacons Reflective beacons	EC EC EC EC	A A A A
Active ranging(reflectivity ,time-of-light ,and geometry triangulation)	Reflectivity sensors Ultrasonic sensor Laser range finder	EC EC EC	A A A
Motion/speed sensor (speed relative to fixed or moving objects)	Doppler radar Doppler sound	EC EC	A A
Vision-based sensors(visual ranging, whole-image analysis	CCD/CMOS camera(s) Visual ranging packages	EC	P

A=active; P=passive; P/A=passive active; PC=proprioceptive; EC= exteroceptive.

The perception of nearby objects distance is done using ultrasonic sensors by triggering it. Then the estimation of the distance is done from the time elapseduntil receiving the echo signal using the following equation:

$$D = \frac{S*T}{2} \quad [2.1]$$

Where D is the distance, S is sound speed and T is round trip time between the sensor and the object.

The perception of the altitude is done using the barometer sensorfrom the measured pressureby the following international barometric formula:

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.2\ 55}}\right) \quad [2.2]$$

Where P is current pressure and P_0 is sea level pressure.

Thus, a pressure change of Δp= 1hPa corresponds to 8.43m at sea level.

- **Localization and mapping:**

Localization is the main issue that is needed in order to perform autonomous navigation. The robot needs to know its global locationrelative to some landmarks and should be able to recalculate (re-localize) its position during the navigation[4].

Localization has different techniques depending on the characteristics of the robot. The techniques that work fine for one robot in one environment may not work well or at all for another robot or in another environment. For example, localizations which work well in an outdoors environment using GPS signal may be useless indoors where there is poor or no signal. In general all localization techniques provide two basic pieces of information:

- The current location of the robot in some environment: X, Y and Z.
- The robot's current orientation in that same environment: Roll, Yaw and Pitch.

Mapping is needed in autonomous mobile robots (AMRs) to integrate the information gathered with the robot's sensors into a given representation so that it can be useful in the planning process. There are different types of mapping schemes;

- Topological maps:

A topological map describes the environment based on its utility to the robot, i.e. what in the scope of the robot operations can be performed there. The maps are seen as a graph where; nodes represent places, such as rooms; edges represent links between places like hallways or doors. Moreover, each node contains a description of the place or its abilities. A room may, for example, contain a printer; this would augment the respective node with the ability to give access to a printer. This kind of maps is clearly very useful for high level deliberation. It is easy to plan for a goal on this description. However when computing the trajectory from A to B (where A and B refer to spatial coordinates) these maps are not enough, as they do not contain geometric information about the environment.

- Feature maps:

A feature map is a list of features extracted from the environment and with known positions, this mapping technique offers a good geometric description of the environment as, by observing a feature and computing its relative position, it is possible to calculate the global position of the robot. The short coming here are three: the number of unique features may be too small, i.e. the environment may be too simplistic; the difference from one feature to

another may not be enough for the sensors to understand; the feature themselves do not give any more information about the environment like the topological maps might give.

- Grid maps:

A grid map divides the map into subspaces, each position is either occupied or free and the robot calculates his position by evaluating the grid around. This approach has the advantage of reducing the path-planning problem to a search and trajectory smoothing algorithm, however the updating process to the entire grid is very computational intense and the grid usually fill a lot of memory.

Simultaneous Localization and Mapping (SLAM) is the main concept which is used in localization[5]. SLAM is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time navigating the environment using the same map. It consists of multiple parts; Landmark extraction, data association, state estimation, state update and landmark update.

Landmarks are features that can be easily re-observed and distinguished from the surroundings. They are used by the robot to find out where it is. Landmarks should be easily re-observable, distinguishable from each other, plentiful in the environment and stationary.

Data association is used to match observed landmarks with ones existing in the map. State estimation is the process of estimating the state (position) of the robot from odometry data and landmark observations. There are different methods for estimating the state i.e. Extended Kalman Filter.

•Motion planning

The movement of an object seems easy, but finding the path to move through is much more complex. Motion planning is the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.

Motion planning task is to produce a continuous motion that connects a start configuration S and a goal configuration G, while avoiding collision with nearby obstacles. Motion planning can be divided from three different perspectives; from the time of the data that used in the planning process to online planning and offline planning. Online planning uses just the current state data and the surrounding objectsdata fused from different sensors to calculate the next movement, while offline planning usesa saved version of the data about previously visited locations[6].

From the goal seeking perspective motion planning is divided into path planning in which the planner seeks for the path that leads to the goal and obstacle avoidance in which the planner ignores the goal and seeks for the occurrence of objects near the quadcopter only and calculate an avoidance action. From the location of the motion planning in the control loop it is divided into dynamic planning in which the motion planner is inside the control loop supervising the execution of the motion by re-planning the path every motion-executioniteration and static planning in which the planned motion should be executed completely before starting a new planning process.
Modern algorithms have been fairly successful in addressing hard instances of the basic geometric problem and a lot of effort is devoted to extend their capabilities to more challenginginstances[7]. The below summary discusses some of these algorithms:

- A* search algorithm:

  Is a computer algorithm that is widely used in path-finding and graph traversal, the process of plotting an efficiently traversable path between multiple points is called nodes. Noted for its performance and accuracy, it enjoys widespread use. However, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, although other work has found A* to be superior to other approaches.

- Rapidly exploring random tree (RRT):

  Is an algorithm designed to efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem.

- Probabilistic roadmap (PRM):

  Is a motion planning algorithm in robotics, which takes random samples from the configuration space of the robot, testing them for whether they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations. The starting and goal configurations are added in, and a graph search algorithm is applied to the resulting graph to determine a path between the starting and goal configurations.

- Attractive and repulsive: is a mixture of path planning and obstacle avoidance which consists of an attractive component and a repulsive component. The attractive potential pulls the robot toward the goal but the repulsive potential pushes the robot away from the obstacles.It is

considered very robust against control and sensing errors and quite an efficient algorithm.

- **Navigation and Control**

Given partial knowledge about its environment and a goal position or series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible[4].

There are two types of the navigation systems: GPS navigation system and inertial navigation system (INS). The former uses the received GPS positions during the navigation to both localize and correct robot's position. The INS integrates the IMU measurements to produce position, velocity, and attitude estimates. INSs are self-contained and are not sensitive to external signals. Since an INS is an integrative process, measurement errors within the IMU can result in navigation errors that will grow without bound. INS errors (and calibrations) can be corrected through a well-designed data fusion procedure.

Despite the huge literature of the mobile robot navigation, the development of intelligent robots able to navigate in unknown and dynamic environment is still a challenging task [8]. Therefore, developing techniques for robust navigation of mobile robots is both important and needed.

The control of the robot is forcing the equipped hardware to take an action that is required of the robot, to move between the planned points. Effective control of a robot's hardware faculties and making use of sensor feedback are extremely important.

There are many types of controllers in robotics vary depending on the characteristics of the robot actuating system; LQG, PID …etc. There are five properties should be existed in the good controller:

- Stability and Robustness.
- Tracking and Optimality.
- Disturbance rejection.

The most famous controller is the Proportional-Integral-Derivative controller (PID). Although it is relatively simple, it can provide a satisfactory performance in many process control tasks. The PID controlcommand can be calculated from the error between the desired state and the current state using the following equation:

$$u = k_p * e(t) + k_i * \int_0^t e(t)\, dt + k_d * \frac{d}{dt} e(t) \quad [2.3]$$

Where k_p, k_i *and* k_d proportional, integral and derivative are gains respectively and $e(t)$ is the process error at time (t).

2.2Related Works:

This section describes the history and methodology used intwo long running projects. Both of them already had many researches achievements and have contributed enormously in the development of autonomous quadcopters.

Flying Machine Arena (FMA) [7]is a part of Swiss Federal Institute of Technology in Zürich. The project intended to allow the quadcopter to localize itself indoor by fusing the information from the on-board sensors and a vision system installed on the room; this

project was never implemented but was an idea. Many years passed and the (FMA) researchers were stillresearching but at a slower rate, until (2007) when D'Andrea returned to the academic world andpushed FMA forward. Starting from (2009), we can identify many important research results in quadcopters control and autonomy such as Iterative Learning Control(ILC)[8] that allows a quadcopter to perform aggressive maneuvers without the need to precisely model the entire environment as such would be too costly. This algorithm was light enough to runonline on the robot and was tested in the FMA quadcopters. At the time, during tests, researchers found that the vision-based localization system used in the FMA gave some misalignments due to impacts or hand manipulating of the quadcopters. With this in mind they developed a system that could recalibrate the system automatically even when there are multiple robots[9].

Two years later FMA published an article where they report the successful co-ordination of a group of quadcopter in catching and throwing a ball[10]. In the same year another paper was published, it describes a controller to safeguard mechanical failures in the quadcopters or the vision system[11]. As this technology is getting more public such measures are needed to prevent disastrous events.

Another related work that has been getting much attention in the last years is the GRASPLab at the University of Pennsylvania. They also have a broad range of subjects but focus their applications to quadcopters. In (2010), they published a paper describing amethod to control quadcopters landing on difficult situation, like upside down platforms[12]. During their research they found that this method also

allows the quadcopters to pick up objects withthe use of a claw. But what is more important is that although the global localization is given by a vision system, the quadcopters have to identify the landing surfaces on their own and thusstarting a path to autonomy.

By (2012) Daniel Mellinger reported methods to both single and multiple quadcopter systems, which allowed the quadcopters to generate and follow aggressive trajectories[13]. Lastly in [14]a fleet of small quadcopters flies in formation with less than a body length of separation, they overcome obstacles without ever crashing into each other or theenvironment.

CHAPTER THREE

METHEDOLOGY

In this chapter the proposed architecture and its implementation will be described in details in addition to the ground station that handles the interface with the user. For a better understanding, it should be noted that software architecture refers to the idle design of the system, while hardware architecture refer to the actual implementation based on the available components in the market.

3.1 The Design of the Autonomous Quadcopter:

The quadcopter can be implemented to be an autonomous robot through the explained design architecture and can be utilized to do several jobs in both civilian and military applications.

The focus of this thesis is to design the quadcopter for civilian applications, specifically for search and rescue missions. In the past ten years there have been a large number of urban disasters throughout the world due to weather and earthquakes Hurricane Katrina in (2005), the (2010) Haiti earthquake, or the (2011) Tohoku earthquake and tsunami. In urban disaster scenarios, USAR (Urban Search and Rescue) teams respond to find and save victims. Unfortunately, rescue teams typically have less than 48 hours to rescue victims before their chance of survival decreases dramatically. The quadcopter may be the most effective solution for such time constrained and dangerous rescue missions.

- **Quadcopter dynamics:**

The quadcopter consists of four rotors that are mounted at the end of two perpendicular axes driven by a DC brushless electric motor.

The quadcopter has two configuration of rotors' rotation:

- Plus configuration (+): where rotors at opposite ends of an arm turn in the same direction and rotors on a perpendicular axis rotates in the opposite direction. This concept is illustrated in Figure 3-1.

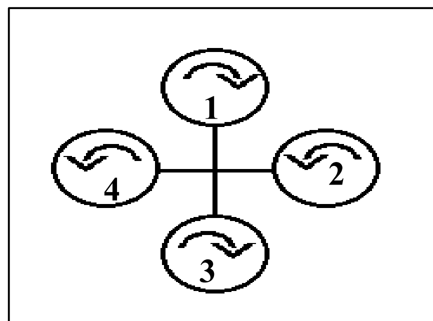


Figure 3-1: Plus quadcopter schematic.

When all four motors spine at the same speed, the rotors create thrust that lifts the quadcopter into the air [9]. As there are pairs of rotors spinning in opposite directions, the torque produced in each direction around the yaw axis cancels out and the yaw angle remains constant. To change the pitch attitude, the speed of motor (1) is reduced while the speed of motor (3) is increased, or vice versa, creating a non-zero pitch angle. As both motor (1) and motor (3) are rotating in the same direction the total counteracting torque provided is not changed so the quadcopter maintains its yaw angle. The roll attitude is adjusted in a similar manner. To adjust the yaw angle the speed of motors (1) and (3) are increased while the speed of motors (2) and (4) are decreased, or vice versa. This creates imbalance in the total torque in the yaw axis and so the quadcopter will change yaw angle.

The quadcopter should maintain a relatively constant thrust during yaw and the height of the aircraft should remain constant.

- X configuration: is quite similar to the plus configuration in everything except that the pitch movement is achieved by increasing the speed of motor (1) and (2) at the same time and decreasing the speed of motor (3) and (4) or vice versa; the roll movement is achieved by increasing the speed of motor (2) and (3) at the same time and decreasing the speed of motor (4) and (1) or vice versa.

X configuration is preferred when using a camera because the camera will have enough space to take clear photos without interfering with the rotors

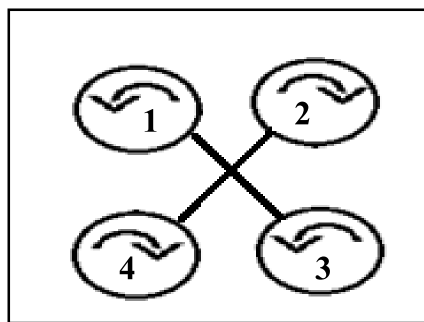


Figure 3-2: X quadcopter schematic.

While the above description provides a simplified overview of how a quadcopter maneuvers, the dynamics of the quadcopter are complex and tightly coupled. These dynamics make it extremely difficult for a human to control the quadcopter without an onboard flight augmentation system to reduce the unwanted response down to an acceptable level.

- **Quadcopter control:**

Roll, pitch, and yaw dynamics are controlled by increasing or decreasing the speed of four motors by the controller to achieve the desired value.

Different control strategies of the quadcopter have been studied in commercial, academic, and military platforms such as PD-PID controller, inverse control, back stepping control, and sliding mode control [4].

Four rotors increase the maneuverability of the vehicle. Having four rotors increases load carrying capacity, on the other hand, constrains it to consume more energy.

- **The quadcopter design consideration:**
  - Take-off throttle:

It is the throttle that should be applied to every motor to make the quadcopter leave the ground. Take-off throttle can be calculated from the basic static laws of beams; which state that the beam will remain static if the resulting forces in the opposite directions are the same. For example (figure3-3) if the beam in the ground the downward force of it is its weight which can be calculated from the equation[3.1] [10]:

$$w = m * g \quad [3.1]$$

Where ***w*** is the weight, ***m*** is the mass of the beam and ***g*** is earth gravity. To make the beam move in up direction, the resulting upward force should exceed the weight.

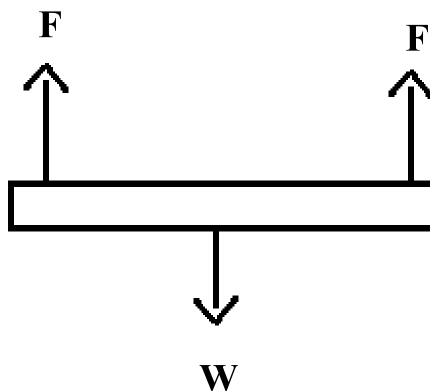


Figure 3-3: Illustration of the forces on a beam

In the quadcopter the total forces generated by the motors should exceeds its weight in order to take off from the ground. Every motor has a specification of the maximum thrust that can be generated (in grams) written in the data sheet of the manufacturer. The thrust of the motors is controlled as a percentage of the maximum thrust. For example sending (40%) throttle means that the motor will generate (40%) of the maximum thrust. And the take-off throttle in percentage can be calculated according to the following equation:

$$take\ off\ throttle = \frac{total\ mass\ of\ the\ quadcopter}{4 * maximum\ thrust\ of\ the\ motor} * 100\% \quad [3.2]$$

- The center of the gravity:

  It is the point in the body of an object that the object rotates around. For example when throwing a spoon in the air it rotates around a fixed point that point is the center of gravity [9].

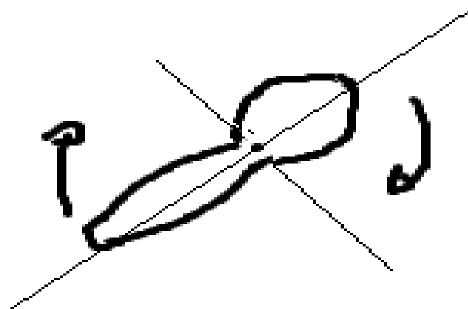


Figure 3-4: The center of the gravity of a spoon

Around the center of gravity the mass is distributed equally. If the center of gravity is in the middle of the body then applying equal

forces in the corners of the body will not rotate it because the resulting rotation momentum is zero;

Rotation momentum = Force*Length/2 – Force*Length/2 = 0.

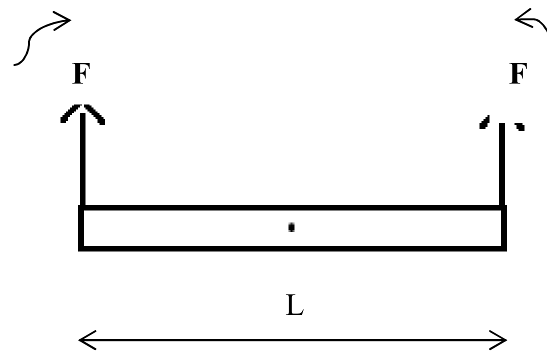


Figure 3-5: Illustration of the rotation momentum

Adjusting the center of gravity position to match it with the center of the body is very important to make the quadcopter stable and not making any rotation when sending equal throttle to all the motors. It is done by suspending the quadcopter from its body center and adjusting the distance of the components from the center until the body agrees with the horizontal plane.

- **The advantages of the quadcopter**:

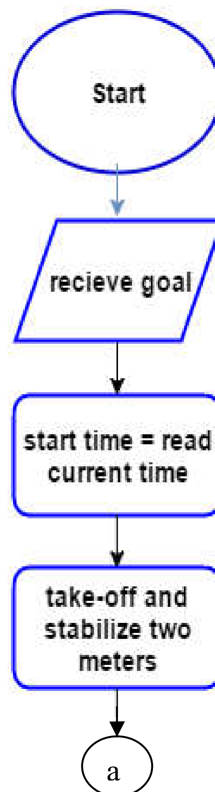
Despite the complex control systems required for a quadcopter aircraft, there are many benefits to this aircraft over other platforms. Having four rotors as opposed to a single rotor in a traditional helicopter allows each of the rotors on a quadcopter to be smaller and lighter, hence carrying less kinetic energy [2]. This is advantageous when working within indoor environments. For example, in the undesired case of a blade striking an object, due to the

design of a quadcopter, much less damage will result when compared to a helicopter in the same situation. It is also possible to mount the rotors within a duct or shroud to protect both the aircraft and any object if contact occurs.

The mechanics of the quadcopter are relatively simple and the aircraft is able to use fixed pitch propellers. This reduces setup, maintenance, and manufacturing costs and time associated with a quadcopter. The relatively simple mechanical setup of a quadcopter also leads to limited vibration making it a friendly environment for inertial sensors and cameras.

3.2 Software architecture:

The software architecture should be modular and flexible to allow researchers to change only certain parts of the system and still get a working deployment[1]. The overall design architecture is presented in Figure 3-6.



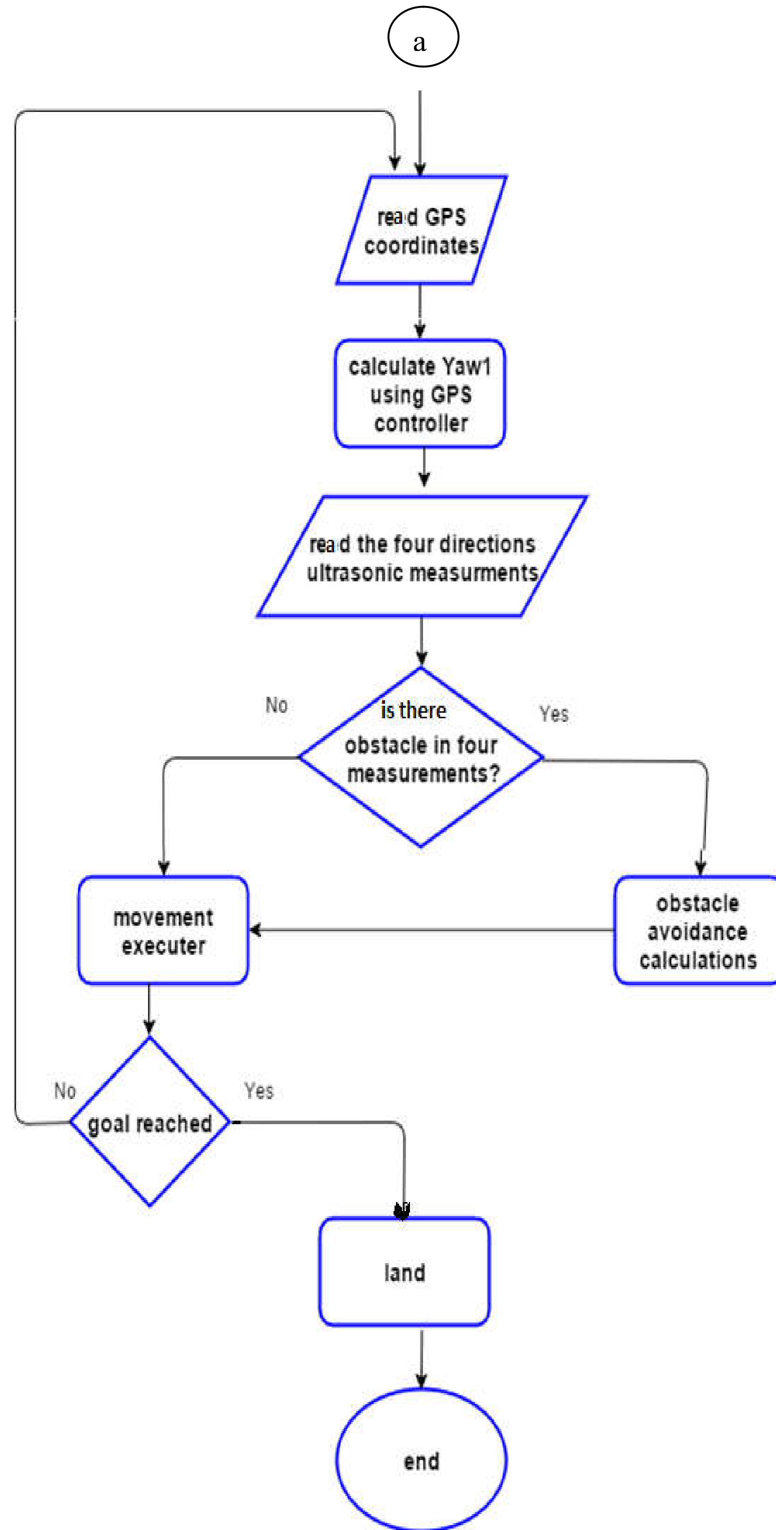


Figure 3-6: Main software algorithm

3.2.1 Perception layer:

The perception of nearby objects is done using four ultrasonic sensors. The maximum measurement rate of the ultrasonic sensor is (15) readings per second; reading above this rate causes ultrasonic buffer to overflow[4]; ten readings per second is found satisfactory and chosen for the fusion algorithm.

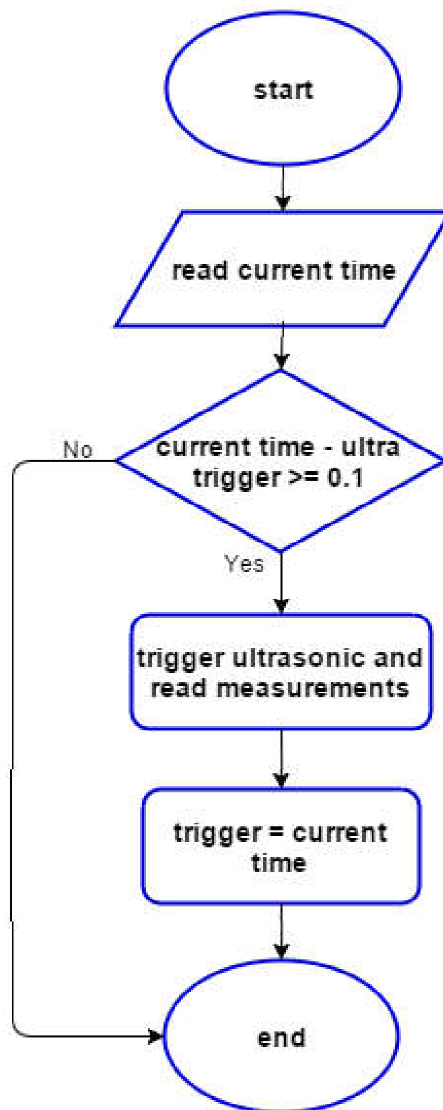


Figure 3-7: Ultrasonicfusion algorithm

GPS sensor also has a low measurement rate so the above algorithm is used for its readings too.

3.2.2 Localization/map building layer:

Localization in outdoor environments gives the position of the robot using GPS sensor readings[1]. The proposed implementation does not use any algorithm like SLAM-based algorithms for localization.

3.2.3 Cognition/planning layer:

In planning layer there are two main algorithms, attitude planner and attractive and repulsive obstacle avoidance planner. The path to the goal position is calculated by an attitude planner then if there is nearby obstacles the attractive and repulsive planner calculates the best movement that keep the quadcopter away from collision and at same time tracking its goal[14].

Attitude planner first finds the quarter that the goal is in assuming an x, y Cartesian axes that the quadcopter is the center of it as shown in figure (3-3). Then it calculates the angle to the goal by the method explained in the algorithm flowchart (figure 3-9). Attractive and repulsive working mechanism is also explained in the algorithm flowchart (figure 3-10).

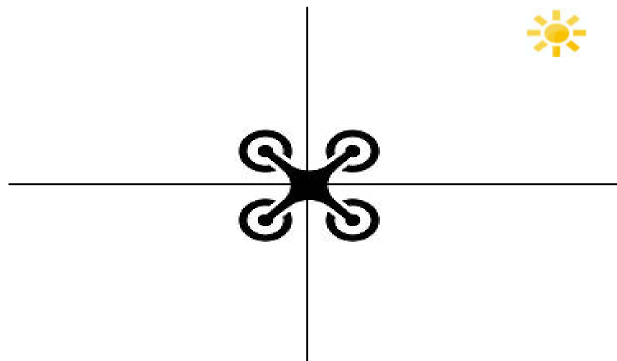


Figure 3-8: Calculating the quarter

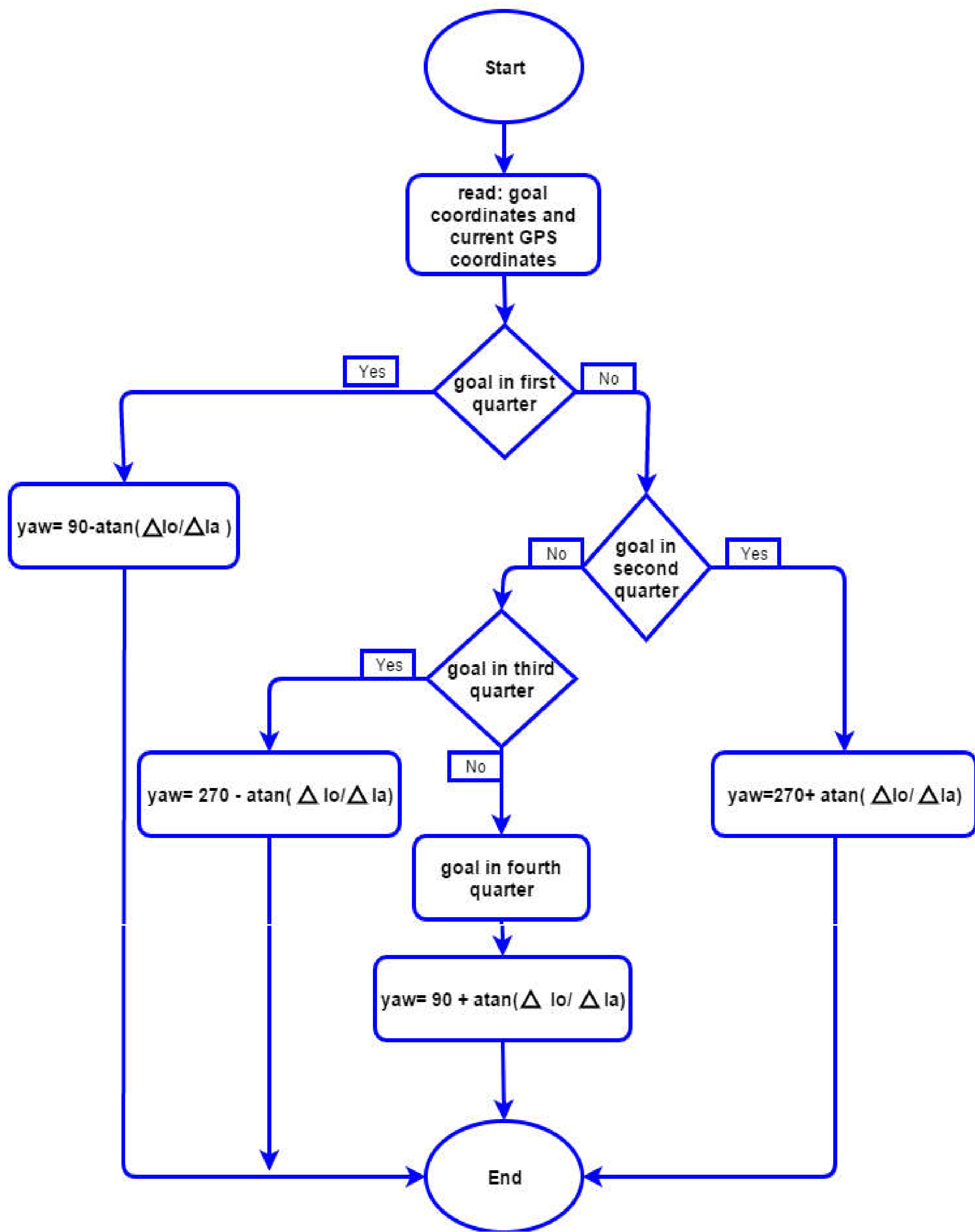


Figure 3-9: Attitude algorithm

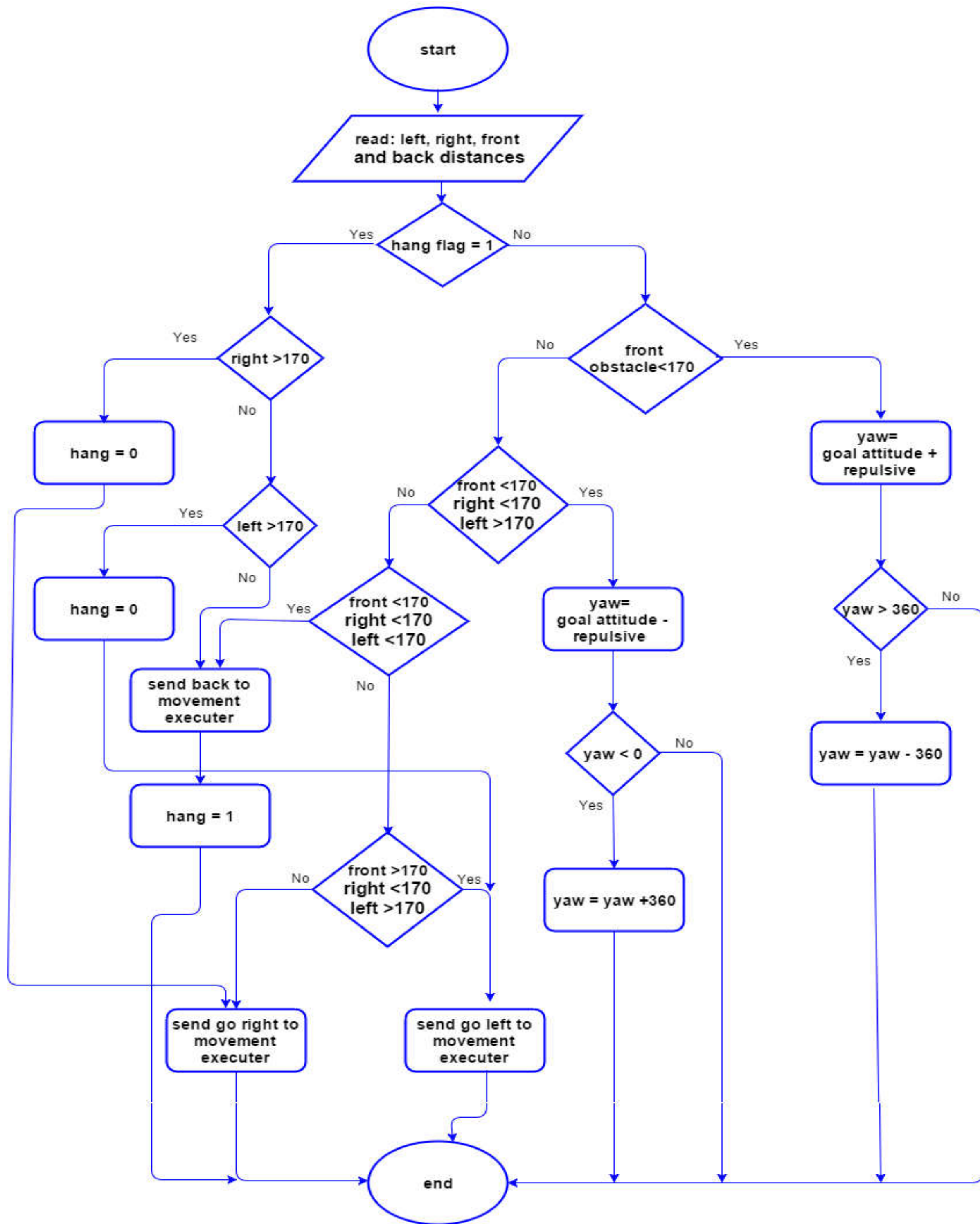


Figure 3-10: Obstacle avoidance algorithm

3.2.4 Motion control layer:

Because of the difficulties of designing all the controllers from scratch a flight controller is used to do the final stabilization control but the attitude and altitude controllersas well as the position controller will be designed as acascaded loop P-controller in the on-board computer.

Attitude and altitude inner loop controller receives the current altitude from the barometer and attitude from the IMU sensor fusionthen it calculates the error between the current and desired values and calculates the control command to decrease that error[3]. If all the errors become below deified thresholds for every process then the controller declares completing the mission (static planning) but the thresholds is going to be selected large so that to reduce dramatically the number of the correction iterations (semi-dynamic planning). Figure 3-11 illustrates the attitude and altitude controller loop.

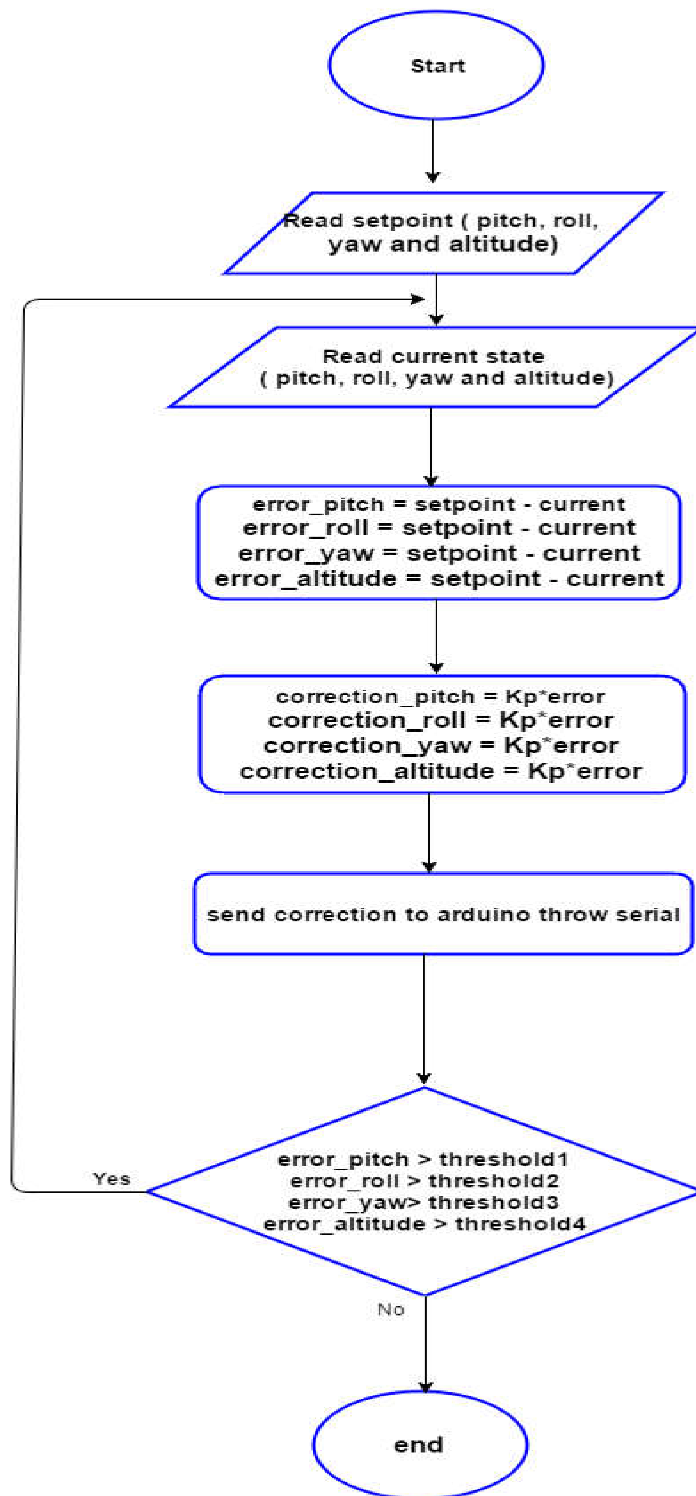


Figure 3-11: Attitude and altitude controller algorithm

3.3 Hardwarearchitecture:

This section describes the system hardware thatthe quadcopter is made of which has been classified into four categories: sensors, actuating system, power system and processing units.

3.3.1 Sensors

The quadcopter has a set of sensors that help in identifying its state and the environments around it. These sensors are: ultrasonic, barometer, IMU, GPS sensor and a camera.

- **Ultrasonic sensor (**HC-SR04 Module**)**

Ultrasonic distance sensors are designed to measure distance between the source and target using ultrasonic waves. It uses sonar to measure distance with high accuracy and stable readings.

Five ultrasonic sensors are used in order to measure the obstacles distance from five directions: down, right, left, forward and backward to help the quadcopter to avoid obstacles.

Ultrasonic ranging module (HC-SR04) provides (2cm – 400cm) non-contact measurement function, the ranging accuracy is(3mm). The module includes ultrasonic transmitters, receiver and control circuit. The transmitter transmits short bursts which gets reflected by target and are picked up by the receiver. The time difference between transmission and reception of ultrasonic signals is calculated. Using the speed of sound and (‘Speed = Distance/Time’) equation, the distance between the source and target can be easily calculated.

The basic principles of work:

(1) Using IO trigger for at least (10us) high level signal;

(2) The Module automatically sends eight (40 kHz) anddetect whether there is a pulse signal back.

(3) If the signal back, through high level, time of high output IO duration is the time from sending ultrasonic to returning.

$$\text{Test distance} = \frac{\text{high level time} \times \text{speed of sound (340m/s)}}{2}$$

Table 3-1: Ultrasonic (HC-SR04) distance sensor module pins

Name	Function
VCC	5V, input power
TRIG	Trigger Input
ECHO	Echo Output
GND	Ground

The ECHO output is of (5V). The input pin of Raspberry Pi GPIO is rated at (3.3V). So (5V) cannot be directly given to the unprotected (3.3V) input pin. Therefore, we use a voltage divider circuit to bring down the voltage to (3.3V).

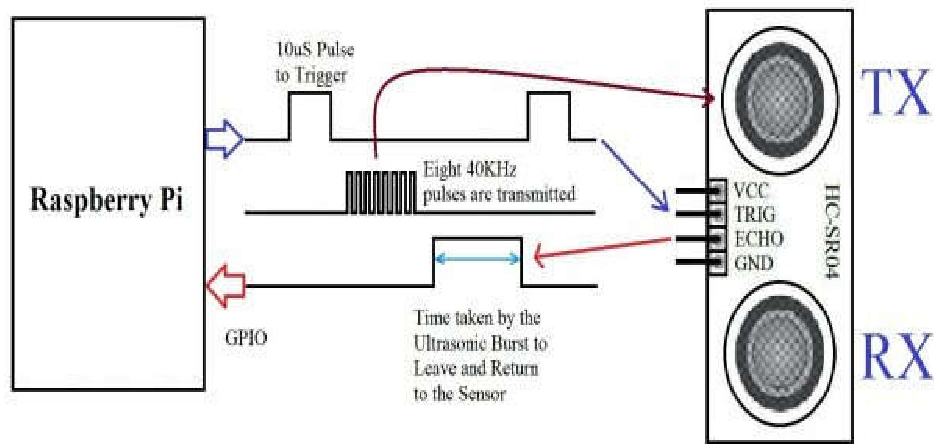


Figure 3-12: Interfacing Raspberry Pi with ultrasonic (HC-SR04)

- **Barometer (BMP180)**

The barometer sensor is used to measure the altitude from sea level. An I2C bus is used to control the sensor, to read calibration data from the EEPROM and to readthe measurement data when A/D conversion is finished. SDA (serial data) and SCL (serialclock) have open-drain outputs. TheI2C is a digital two wire interface and has Clock frequencies up to (3.4Mbit/sec).

Pin configuration of BMP180 is shown in below table:

Table 3-2: Pin configuration of Barometer (BMP180)

Name	Function
CSB*	Chip Select
VDD	Power Supply
VDDIO	Digital Power Supply
SDO*	SPI output
SCL	I2C serial bus clock input
SDA	I2C serial bus data (or SPI input)
GND	Ground

A pin compatible product variant with SPI interface is possible upon customer's request. For I2C (standard case) CSB and SDO are not used, they have to be left open.

- **IMU (MPU-9250)**

An inertial-measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometer, gyroscope and

magnetometer. The accelerometer measures acceleration and also force, so the downwards gravity will also be sensed. The gyroscope measures angular velocity, in other words the rotational speed around the three axes.A magnetometer measures the directions and strength of the magnetic field. This magnetic sensor can be used to determine which way is south and north. The pole locations are then used as a reference together with the Yaw angular velocity around from the gyroscope, to calculate a stable Yaw angle.

(MPU-9250) features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer out puts.

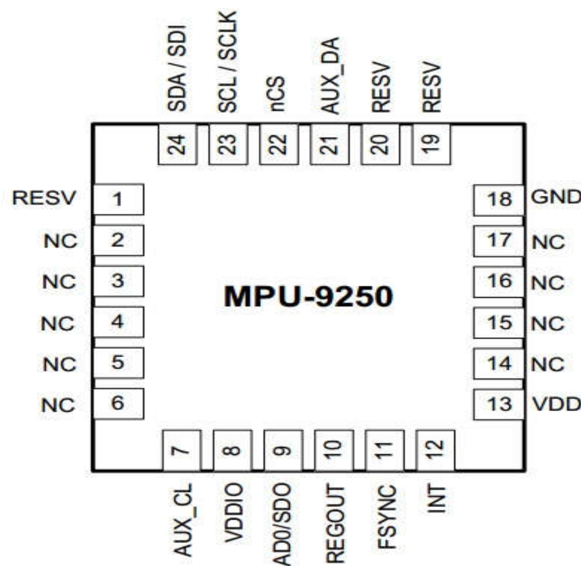


Figure 3-13: Pin out Diagram for IMU (MPU-9250)

Table 3-3: IMU (MPU-9250) pin layout

Pin number	Pin name	Pin description
1	RESV	Reserved, connected to VDDIO.
7	AUX_CL	I2C master serial clock(external sensors)
8	VDDIO	Digital I/O supply voltage
9	ADO/SDO	I2C Slave address (AD0);SPI serial data o/p
10	REGOUT	Regulator filter capacitor connection
11	FSYNC	Frame Synchronization digital input
12	INT	Interrupt digital output
13	VDD	Power supply voltage and digital I/O voltage
18	GND	Power supply ground
19	RESV	Reserved, do not connect
20	RESV	Reserved connect to GND
21	AUX_DA	I2C master serial data (external sensors)
22	n-CS	Chip select (SPI mode only)
23	SCL/SCLK	I2C serial clock(SCL);SPI serial clock(SCLK)
24	SDA/SDI	I2C serial data (SDA);SPI serial data input(SDI)
2-6, 14-17	NC	Not internally connected.

- **GPS (NEO-6)**

The GPS module retrieves the location information from the near satellites. The (NEO-6) module series brings the high performance of the (u-blox 6) position engine to the miniature (NEO) form factor. (U-blox 6) has been designed with low power consumption and low costs.

The main components of (NEO-6) module are EEPROM to save the setting and enable Display Data Channel (DDC) mode, backup battery to allow the system to warm start and antenna for communications purpose. The I2C compatible (DDC) interface can be used either to access external devices with a serial interface EEPROM or to interface with a host CPU.

Table 3-4:GPS (NEO-6) pin layout

Pin name	Function
VCC	Power supply
TX	Digital pin for Transmit data
RX	Digital pin for receive data
GND	Ground

- **The camera**

The camera was used to stream image capturesfrom the environment that the quadcopter navigates.

3.3.2 Actuating system:

The Actuating systemis responsiblefor performing the movement by adjusting the forces and the torques around the quadcopter, it consists of three components: brushless motors, electronic speed controllers (ESCs) and propellers.

- **Brushless motors**

Brushless DC motors provide the necessary thrust to the propellers. Each rotor needs to be controlled separately by a speed controller.

Brushless motors are a bit similar to normal DC motors in the way that coils and magnets are used to drive the shaft. Though the brushless motors do not have a brush on the shaft which takes care of switching the power direction in the coils, and this is why they are called brushless. Instead the brushless motors have three coils on the inner (center) of the motor, which is fixed to the mounting. On the outer side it contains a number of magnets

mounted to a cylinder that is attached to the rotating shaft. So the coils are fixed which means wires can go directly to them and therefor there is no need for a brush.

Generally brushless motors spin in much higher speed and use less power at the same speed than DC motors. Also brushless motors don't lose power in the brush-transition like the DC motors do, so it's more energy efficient. Brushless motors come in many different varieties, where the size and the current consumption differ. To select brushless motor the KV-rating,weight, thrust per motor, size, type of propeller should be put in consideration.

- **Electronic Speed Controller (ESC)**

The brushless motors are multi-phased, normally 3 phases, so direct supply of DC power will not turn the motors on. That where the Electronic Speed Controllers (ESC) comes into play. The ESC generates three high frequency signals with different but controllable phases continually to keep the motor turning. The ESC is also able to source a lot of current as the motors can draw a lot of power.It has three input ports (two for the battery and one for the PWM) and three output ports to the motor as shown in below Figure.

Figure 3-14: Typical Electronic Speed Controller

- **The propellers:**

A propeller is a type of fan that transmits power by converting rotational motion into thrust. A pressure difference is produced between the forward and rear surfaces of the airfoil-shaped blade, and a fluid (such as air or water) is accelerated behind the blade.

The propellers come in different diameters and pitches (tilting) according to the frame size and the type of the motors.

3.3.3 The Power System:

Power system supplies the different components with power it consists of: (11.1V) battery, a power distribution board and a buzzer.

- **The battery:**

Lithium-polymer battery (11.1V 2200mAh)is used to supply the power. Lithium batteries are batteries that have lithium as an anode. They stand apart from other batteries in their high charge density (long life) and high cost per unit. Depending on the design and chemical compounds used, lithium cells can produce a voltage of(3.7 V)per cell.

- **The power distribution board:**

It is used to distribute the power of the battery to the motors. It has two (5V)voltage regulator outputsone of them is used to supply the on-board computer.

- **The buzzer**:

The buzzer is used to measure the voltage of the battery and make alarm when battery cells are below a critical voltage level (specified by the user).

3.3.4 TheProcessing units:

Three processing units are used, the first one is Raspberry Pi 3 for performing the top level computation and control,the second one is CC3D flight controllerto perform the down level control by stabilizing the system at inputs pushed by the RaspberryPi 3 and last oneis Arduino which acts as an intermediate communication unit to pass the inputs to the flight controllerfrom the Raspberry Pi.

- **Raspberry Pi:**

The Raspberry Pi is a single-board small computer that has computational power could be compared with the big PCs. It is used to perform the top level perception, localization, planning and control tasks.

- **Arduino:**

Arduino is used as a PWM generator because it has 6 stable PWM pins which is a limitation of the Raspberry Pi which has only one and the (CC3D) needs at least 4 PWM inputs to perform the stabilization process.

The board features serial communication interfaces such as Universal Serial Bus (USB) which will be used for the communication with the Raspberry Pi.

- **Flight controller (CC3D):**

CC3D flight controller is used to stabilize the movement of the quadcopter by stabilizing at the Raspberry Pi commands and compensating for any unusual movement caused by the air currents or any other external force.

3.4 The ground station system

It is used to form the interface between the user and the autonomous quadcopter in which the user canmonitor the state of the quadcopter (GPS coordinates, altitude and attitude); also he can access the quadcopter camera to see what the autonomous quadcopter sees.

In addition the ground station allows the user to send the goal GPS coordinates to the quadcopter.The communication is done via Wi-Fi, and the packets are sent over a TCP connection to/from the ground station.

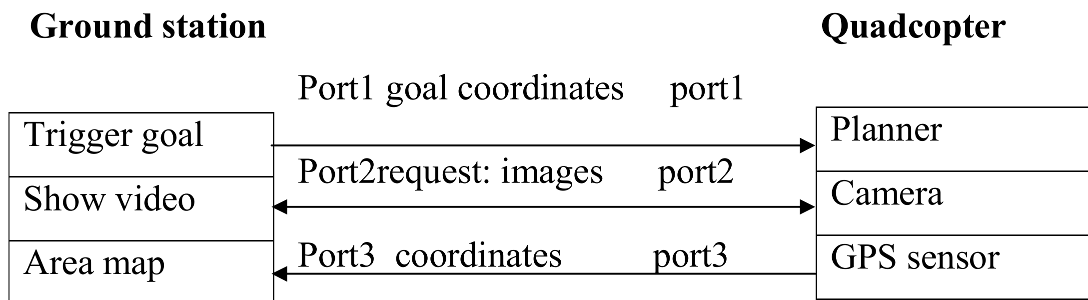


Figure 3-15: The communication architecture between the ground station and the quadcopter.

3.5 The simulation

This section contains a description of the environment used to simulate the autonomous quadcopter, the simulation parameters, the simulation process as well as a brief discussion about the results.

3.5.1 The simulation environment

The Virtual Robot Experimentation Platform (V-REP) simulation environment was used to simulate the movement of the autonomous quadcopter in an indoor environment. V-REP is a general purpose robot simulator with integrated development environment. Lots of sensors,

mechanisms and robots packages can be added to the environment and the whole systems can bemodeled and simulated in various ways. It has many advantages such as fast prototyping and verification, easy to use, fast algorithm development, and its compatibility with different programing languages such as C, C++, Lua and python.

Table 3-5: Illustrates the simulation environment parameters

The parameter	Comment
The sensors	8 proximity sensors and 3 line following sensors
The robot	X configuration quadcopter
The environment	Indoor environment with a plant and 3 cylinders
The view of the environment	4 room cameras; a left side, right side, upper side and a moving camera. 1 front robot camera
The goal of the robot	Following a black line drawn in the floor
Path planning algorithm	Stay-on-the line algorithm
Obstacle avoidance algorithm	Attractive and repulsive algorithm
The programming language	Lua programming language
The safe distance	0.16*(size factor of the robot body)

3.5.2 The simulation process:

The simulation uses four cameras (1-4) to monitor the movement of the quadcopter in a 3D indoor environment. The quadcopter different sensors and its front camera are monitored too in (5). A black line path with some obstacles in it is drawn in the floor in a form of a closed loop. The quadcopter should be able to navigate within the line using the line following sensors while avoiding the collision with the plant and the cylinders in that path.

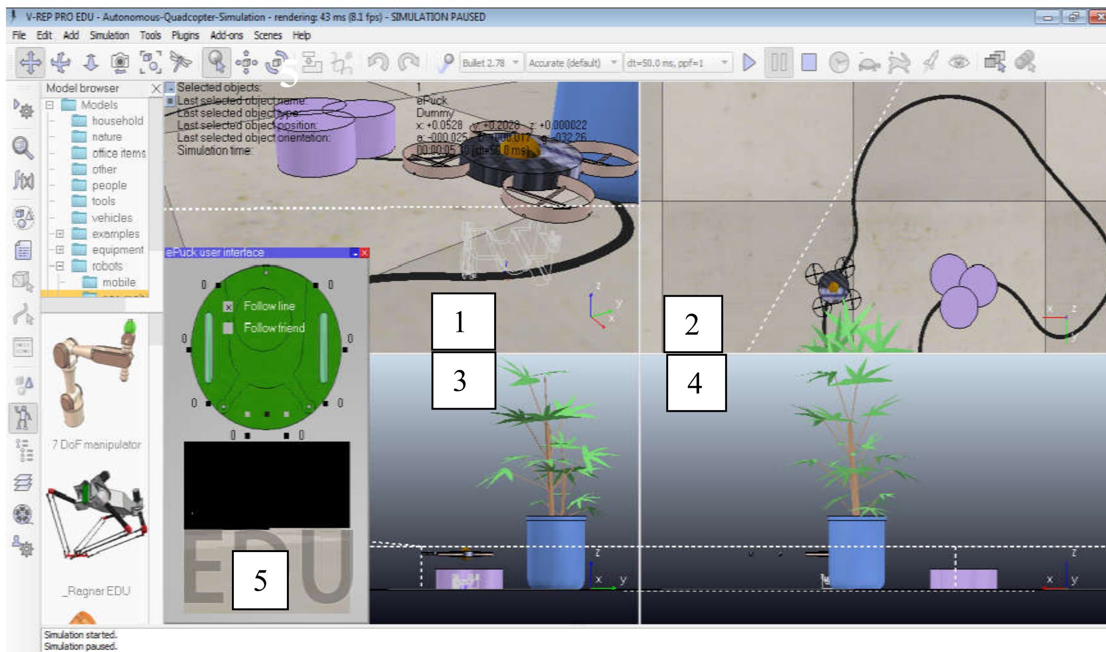


Figure 3-16: The autonomous quadcopter simulation

The quadcopter finds the line by using three line following sensors that return 'True' when detecting a black line. If the middle sensor sees the line this is an indication that the quadcopter is within the path and the path planner command the quadcopter to continue the movement without steering, but if the right sensor sees the line this is an indication that the quadcopter is in the left to the line and the path planner command the quadcopter to steer to the right and vice versa.

The attractive and repulsive obstacle avoidance then reads the distances of eight sensors distributed around the quadcopter's body and compares these distances to a safe distance. In case all the eight measurements is above the safe distance the path planner command is passed to the motor controller to execute the path, but if there is a measurement or more below the safe distance an avoidance path is calculated based on the measurement that is function of the collision distance and the path to the goal.

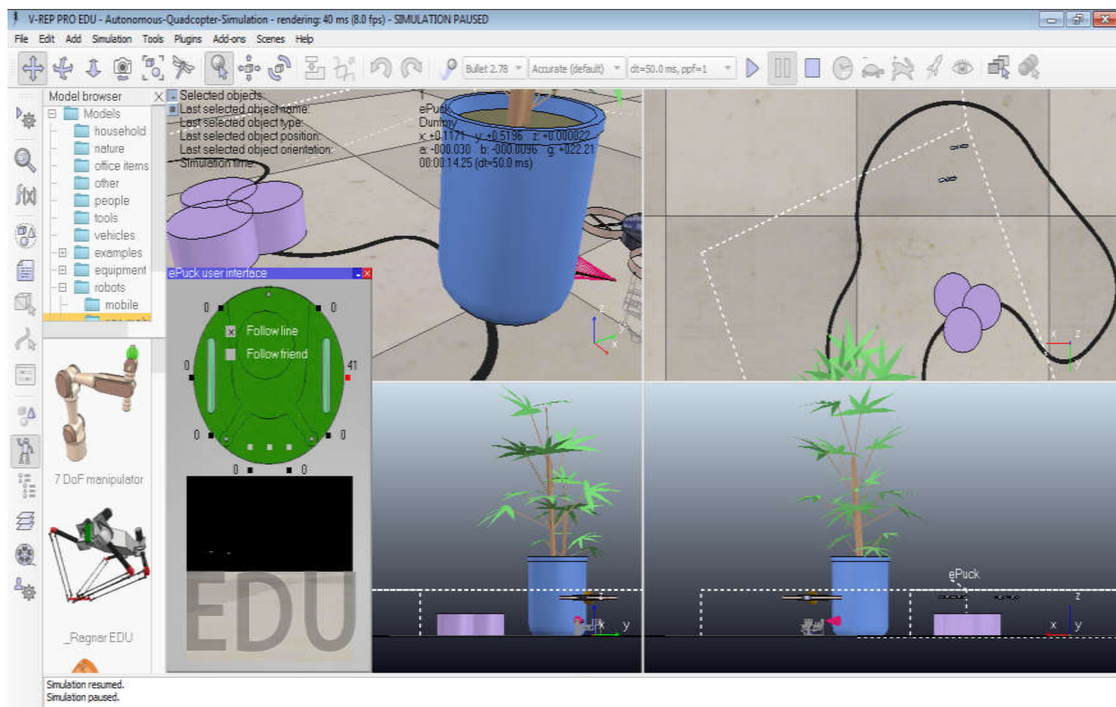


Figure 3-17: Detection of an obstacle shown in the five views

3.6 Prototype Implementation

This section describes the implementation of the proposed model of the quadcopter described in the previous sections.

3.6.1 Frame design

The frame is designed in a form of layers that contain the main components. Our main focus was to make the frame as small and light as possible. The components dimensions have been measured to compute the spacing between every two layers in addition to the diameter of each layer considering our main focus. The layer diameters as well as the spacing between every two layers are chosen based on components' dimensions to allow the component to be put conveniently.

Table 3-6: The specifications of the Quadcopter components

Component	Length/cm	Width/cm	Height/cm
Arduino	6.8	5.8	1.2
Raspberry pi	8.8	5.8	1.8
CC3D	3.5	3.5	1.7
Ultrasonic	4.6	2.0	2.0
Camera	2.5	2.4	1.0
Battery	10.5	3.3	2.4
Buzzer	3.5	2.4	1.1
GPS sensor	3.5	2.5	0.5
Power distribution board	5.2	5.2	0.2

Table 3-7: Dimensions of the Quadcopter components

Components	Dimensions(cm)
Layer1 diameter	11.5
Layer2 diameter	9.7
Layer3 diameter	9.7
Spacing between layer1 and layer2	3
Spacing between layer2 and layer3	3

Polycarbonate-molds are used to fabricate the layers for their small weight and their strength. Layers have been cut by a laser cutting machine and connected together by copper spacers. A plastic cover is attached to the upper layer which contains the CC3D flight controller for protection purposes.

3.6.2 Prototype design phases

- **Phase 1:**

After doing the frame dimension calculations and having the layers cut, we marked the layers in the places we want to place the components at and the places we want to connect them to each other. After that we drilled the layers and assembled everything together.

Figure 3-18: Drilling the layers

\

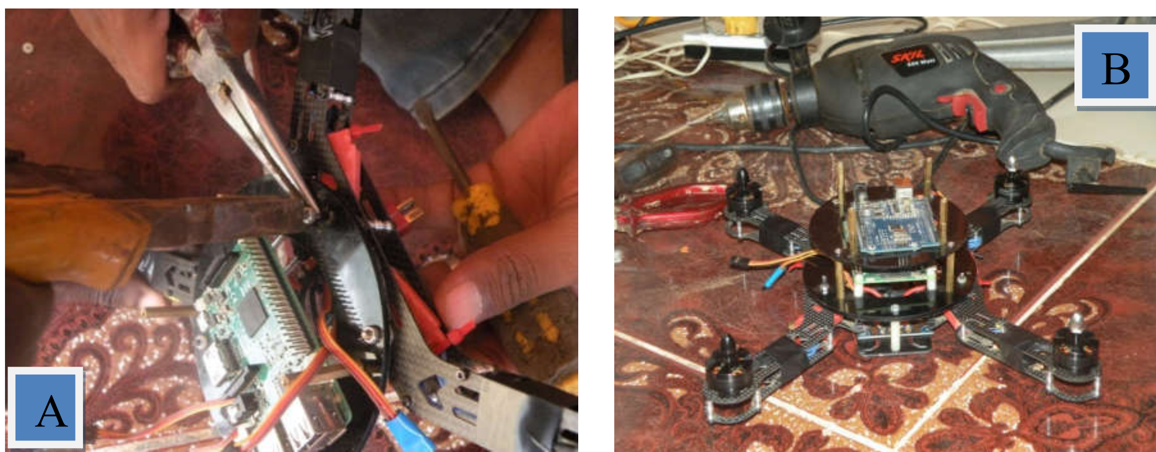


Figure 3-19(A and B): Connecting everything together

- **Phase 2:**

All the motors, ESCs and processing units have been tested, and verified that everything within the frame is working fine without any power or space problems.

Figure 3-20: Testing the different components

CHAPTER FOUR

RESULTS

Thischapter discusses the results related to the implemented design including the hardware calibration and the building and testing phases.

4.1 Results of the simulation:

An experiment has been done to the autonomous quadcopter to evaluate the performance of path planning and attractive and repulsive obstacle avoidance algorithms using V-REP in an indoor environment. The indoor environment has been designed as shown in Figure 4-1.

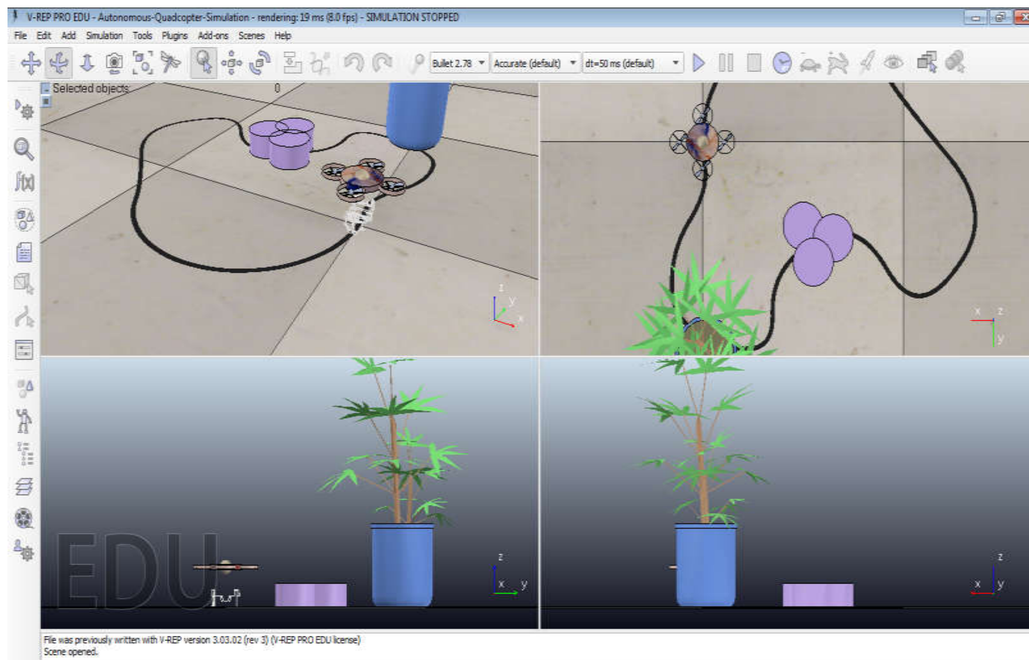


Figure 4-1: Show the quadcopter in the designed indoor environment

The path planner read the readings of the three line following sensors, the leftmost sensor returned ‘True’. The planner then successfully commanded the quadcopter to adjust the movement toward the left. After the execution of the movement the path has been tracked again as shown in Figure 4-2

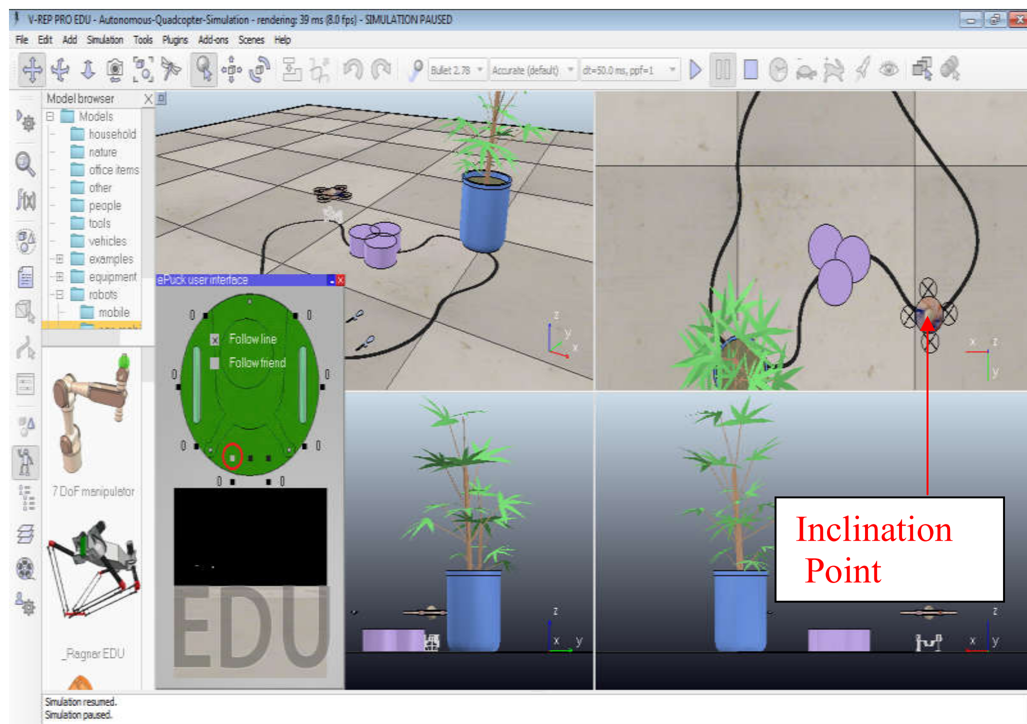


Figure 4-2: The quadcopter succeeded to adjust its path in a turn

Along the path an obstacle has been observed by one of the left proximity sensors. The obstacle avoidance algorithm calculated the avoidance action and commanded the qaudcopter to turn into the right. The quadcopter successfully avoided the collision with the obstacle as shown in Figure 4-3

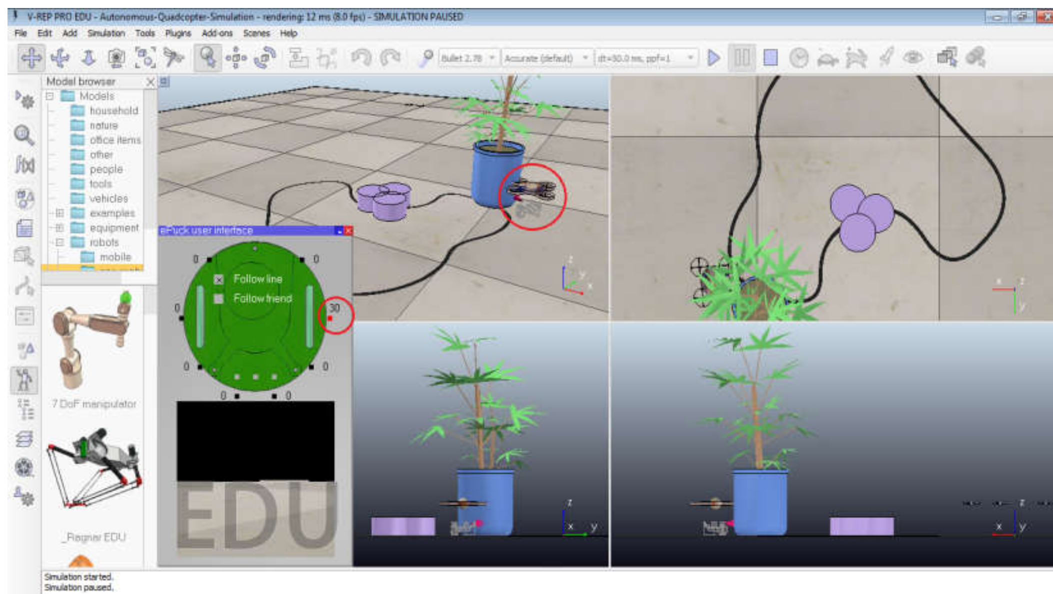


Figure 4-3: The quadcopter successful to read and avoid obstacle

4.2 Hardware calibration:

- Motors calibration

All Motors have been calibrated to figure out the stating throttle of each motor which is shown in the following table.

Table 4-1: Motors calibration

Motor NO.	Starting duty cycle
1	1116 us
2	1116 us
3	1116 us
4	1116 us

- **IMU calibration:**

The IMU has been calibrated to find the minimum and the maximum values of the magnetic field in the three axes measured by the compass and the minimum and maximum values of the rate of change of the velocity in the three axis measured by the accelerometer.

The calibration data is saved in a file and supplied to the fusion algorithm to calculate the angles of the quadcopter in the three axes with respect to the world. Table (4-2) and Table (4-3)show the calibration data.

Table 4-2: Compass calibration

Compass axis	Value
CompassMinX	-3.063191
CompassMinY	-74.754135
CompassMinZ	-52.087727
CompassMaxX	75.672134
CompassMaxY	2.896713
CompassMaxZ	19.115725

Table 4-3: Accelerometer calibration

Accelerometer axis	Value
Accelerometer Min X	-1.153897
Accelerometer Min Y	-1.066652
Accelerometer Min Z	-1.111786
Accelerometer Max X	1.259294
Accelerometer Max Y	1.099110
Accelerometer Max Z	1.032464

4.3 Prototype testing

The response of the quadcopter controller has been tested while following the required safety procedures by connecting the quadcopter by four ropes from four different sides to pull it in case it does any unusual behavior and by wearing gloves to protect the tester.

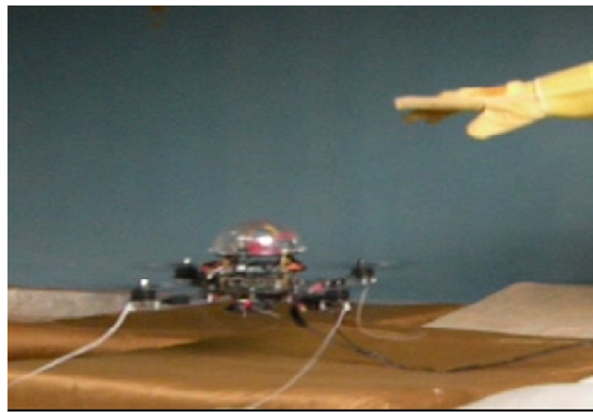


Figure 4.4: The quadcopter test bed

4.4 ComponentsReliability

4.4.1 Sensors

After testing all the sensors we found that:

- IMU: was reliable under all testing circumstances and provided accurate readings
- GPS: our first GPS sensor (u-blox NEO 6M) was reliable and giving accurate readings outdoor. But the current one (APM 2.6 GPS module) is unreliable and giving inaccurate readings.
- Ultrasonic: our first ultrasonic sensors were reliable under all circumstances and giving accurate readings for different

objects. But the current are unreliable and giving inaccurate readings for some objects.

- Barometer: our first and second barometer sensors were unreliable at all and give totally false readings.

4.4.2 Actuating system

Our first actuating system (A2212 brushless motors, 30 A ESCs and (1045) propellers) was unreliable, motors stop spinning above 40% throttle and ESCs were responding differently to the PWM signal. Propellers were easy to crash.

Our second one (Elite 2204-2300KV brushless motors, 12 A ESCs and (6045)three wings propellers) was unreliable too but when we have changed the motors to Emax-MT2204-2300KV and the propellers to (6045) two wings propellers the actuating system became reliable and responsive.

4.4.3 Power system

Along the project we had many issues with the power system. First we had problems providing the processing and sensing units with sufficient power when we was using (7805-5V) linear regulator. Issue has been resolved using the power distributing board regulator.

The second issue was the power failure due to the insufficient motors current delivered by an impaired battery. Issue has been resolved by changing the battery.

The current power system reliability is acceptable.

4.4.4 The processing units

The processing units were totally reliable and provide acceptable performance under all circumstances.

4.5 System stability

The final flight system was unstable due to three contributing factors:

- The difficulty of matching the center of gravity with the body center.
- The difficulty of leveling the flight controller with the horizontal plane of the actuating system.
- The difficulty of tuning the PID controllers using trial and error method

4.6 The ground station testing

The ground station has been tested to evaluate its performance against its range and the result shows that the performance of the communication between the ground station and the quadcopter is pretty good (up to 50m).

CHAPTER FIVE

CONOCLUTION AND RECOMONDITATION

In this chapter we will take some conclusions about the developed project, the chosen path, andthe problems faced throughout this thesis.

5.1 Final Remarks:

After studying a lot about robotics and autonomous agents it is clear that the modular architecture is the best solution to designing autonomous mobile robots because it enables the designer to modify completely any part of the system without affecting the other parts. Also it allows the researchers to focus their effort on only one part of the autonomous system components (perception, localization, planning or control).

After a comprehensive study about different algorithms in the different autonomous system layers we made a trade-off between the complexity and the performance when choosing the suitable algorithm in each layer for our model.

in the localization layer we have studied lots of algorithms and we found that all of them are complex, have too many parameters need to be adjusted to reach satisfactory performance so we used the direct GPS sensor measurement to localize our platform but we compensated for the GPS measurement noise by a powerful obstacle avoidance algorithm, Attractive

and repulsive obstacle avoidance algorithm which found very robust and simple at the same time.

The control loop we have designed is a semi-dynamic planning and control loop in which the planner may be considered part of the control loop. The control loop is basically static which means it should execute the planned path completely before planning again but we have adjusted the thresholds of the controller to return to the planner quickly which increased the supervision of the planner.

The following table shows the things which were helpful to achieve our goals and the things which have found harmful to achieve our goals

Table 5-1: Different remarks about the project

Helpful things	Harmful things
The modular system architecture	Developing our own PID rate stabilizer
CC3D flight controller	Power failures due to insufficient power
The safety procedures	Using complex control commands
The test platform	Using sensors without calibration

5.2 Recommendations:

The main purpose of this project was to launch a quadcopter platform that uses 4 ultrasonic sensors to detect potential collision; we recommend using laser range finding sensor for future implementations because it has a wide angle of detection (up to 360°) and can detect any number of objects within its angle of view.

Also we canceled the use of any localization algorithm in the localization layer due to the time constraint, we recommend using large scale direct monocular SLAM (LSD-monocular SLAM) because of its high performance in building large map of the environment using a single camera and at low computational power compared to its counterparts.

References

[1] I. Nourbaskhsh and R. Siegwart, "Introduction to autonomous mobile robots," *The MIT Press, Cambridge, Massachusetts, England, ISBN 0,* vol. 262, pp. 142-150, 2004.

[2] A. Güçlü, "Attitude and altitude control of an outdoor quadrotor," ATILIM UNIVERSITY, 2012.

[3] S. S. Ge, *Autonomous mobile robots: sensing, control, decision making and applications* vol. 22: CRC press, 2006.

[4] S. Riisgaard and M. R. Blas, "SLAM for Dummies," *A Tutorial Approach to Simultaneous Localization and Mapping,* vol. 22, p. 126, 2003.

[5] Y. B. Sebbane, *Planning and decision making for aerial robots*: Springer, 2014.

[6] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Transactions on robotics and automation,* vol. 16, pp. 615-620, 2000.

[7] S. Lupashin, A. Schöllig, M. Hehn, and R. D'Andrea, "The flying machine arena as of 2010," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2970-2971.

[8] O. Purwin and R. D. Andrea, "Performing aggressive maneuvers using iterative learning control," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2009, pp. 1731-1736.

[9] G. Ducard and R. D. Andrea, "Autonomous quadrotor flight using a vision system and accommodating frames misalignment," in *Industrial embedded systems, 2009. SIES'09. IEEE international symposium on*, 2009, pp. 261-264.

[10] R. Ritz, M. W. Muller, M. Hehn, and R. D'Andrea, "Cooperative quadrocopter ball throwing and catching," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 4972-4978.

[11] M. W. Mueller and R. D'Andrea, "Critical subsystem failure mitigation in an indoor UAV testbed," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 780-785.

[12] D. Mellinger, M. Shomin, and V. Kumar, "Control of quadrotors for robust perching and landing," in *International Powered Lift Conference*, 2010, pp. 5-7.

[13] D. W. Mellinger, "Trajectory generation and control for quadrotors," 2012.

[14] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots,* vol. 35, pp. 287-300, 2013.

Appendix A

The Main Quadcopter Code (python)

```
import Adafruit_BMP.BMP085 as BMP085
import serial
import time
import math

ser = serial.Serial('/dev/ttyACM0', 115200)
ch1,ch2,ch3,ch4,ch5=1500,1500,1000,1500,1500
th1,th2,th3,th4=
hang_flag=0
setpoint_alt,setpoint_roll,setpoint_pitch=
floor_alt=sensor.read_altitude()
# outside scripts connections
hostName='127.0.0.1'
socket_goal = socket.socket( AF_INET, SOCK_DGRAM ) # connect
with the Ground station goal_sender
socket_goal.bind( ('0.0.0.0', 5010) )
socket_imu = socket.socket( AF_INET, SOCK_DGRAM )  # connect
with IMU fusion
socket_imu.bind( (hostName, 5005) )
socket_GPS = socket.socket( AF_INET, SOCK_DGRAM )   # connect
with GPS fusion
socket_GPS.bind( (hostName, 5055) )
socket_fr = socket.socket( AF_INET, SOCK_DGRAM )   # connect
with the front ultrasonic fusion
socket_fr.bind( (hostName, 5015) )
socket_bc = socket.socket( AF_INET, SOCK_DGRAM )   # connect
with the back ultrasonic fusion
socket_bc.bind( (hostName, 5020) )
socket_rs = socket.socket( AF_INET, SOCK_DGRAM )   # connect
with the right ultrasonic fusion
socket_rs.bind( (hostName, 5025) )
socket_ls = socket.socket( AF_INET, SOCK_DGRAM )   # connect
with the left ultrasonic fusion
socket_ls.bind( (hostName, 5015) )



# receive the goal from the ground station goal_sender
goal, addr = socket_goal.recvfrom(1024)
start_time=time.time()
desired_la=float(goal[0:goal.find(',')])
desired_lo=float(goal[goal.find(',''+1:)])
```

```
def
stabilize(setpoint_roll,setpoint_pitch,setpoint_alt,setpoint_yaw
):
    while True:

        # Read current values
        data, addr = sock_imu.recvfrom(1024)
        current_pitch=int(data[6:data.find(',')])

current_roll=int(data[data.find("roll")+5:data.find(',',data.fin
d('r'))])
        current_yaw=int(data[data.find("yaw")+4:])
        current_alt=sensor.read_altitude()-floor_alt   # to read
the altitude with respect to the ground

        if setpoint_yaw==365:
            setpoint_yaw=current_yaw

        # Calculate the processes errors
        alt_error=setpoint_alt-current_alt
        pitch_error=setpoint_pitch-current_pitch
        roll_error=setpoint_roll-current_roll
        yaw_error=setpoint_yaw-current_yaw

        # Calculate the correction
        if  alt_error>th1 and pitch_error>th2 and
roll_error>th3 and  yaw_error>th4:
            additive_throttle=kp*alt_error #kp is detirmined
experimentally
            additive_pitch=kp*pitch_error
            additive_roll=kp*roll_error
            additive_yaw=kp*yaw_error

            # Calculate the final channels values
            ch1+=int(round(additive_roll))
            ch2+=int(round(additive_pitch))
            ch3+=int(round(additive_throttle))
            ch4+=int(round(additive_yaw))

            # send corrections to Arduino
            ser.write(str(ch1)+str(ch2)+str(ch3)+str(ch4))
            time.sleep(.01)
        else :
            break

def
get_goal_attitude(current_la,current_lo,desired_la,desired_lo):
    if (desired_la-current_la>0)and(desired_lo-current_lo>0):
        goal_attitude=90-math.atan((1.03754*abs(desired_la-
current_la))/abs(desired_lo-current_lo))*180/math.pi #
multiplying by (180/pi) because the angle is returned in radian
    elif (desired_la-current_la>0)and(desired_lo-current_lo<=0):
```

```
        goal_attitude=270+math.atan((1.03754*abs(desired_la-
current_la))/abs(desired_lo-current_lo))*180/math.pi
    elif (desired_la-current_la<=0)and(desired_lo-current_lo<0):
        goal_attitude=270-math.atan((1.03754*abs(desired_la-
current_la))/abs(desired_lo-current_lo))*180/math.pi
    else:
        goal_attitude=90+math.atan((1.03754*abs(desired_la-
current_la))/abs(desired_lo-current_lo))*180/math.pi
    return int(round(goal_attitude))

def goal_reached(current_la,current_lo):

    if (abs(current_la-desired_la)<=.00005)and(abs(current_lo-
desired_lo)<=.00005):
        return True
    else:
        return False
def
obs_avoidance(goal_attitude,front_obstacle,back_obstacle,Rside_o
bstacle,Lside_obstacle):
    if hang_flag==0:
        if (front_obstacle<170)and(Rside_obstacle>170):
            yaw=goal_attitude+int(round(((170-
front_obstacle)/170)*90))
            if yaw>360:
                yaw=yaw-360
            return "steer "+str(yaw)
        elif
(front_obstacle<170)and(Rside_obstacle<170)and(Lside_obstacle>17
0):
            yaw=goal_attitude-int(round(((170-
front_obstacle)/170)*90))
            if yaw<0:
                yaw=yaw+360
            return "steer "+str(yaw)
        elif
(front_obstacle<170)and(Rside_obstacle<170)and(Lside_obstacle<17
0):
            hang_flag=1
            return "back"
        elif
(front_obstacle>170)and(Rside_obstacle<30)and(Lside_obstacle>150
):
            return "left"
        elif
(front_obstacle>170)and(Rside_obstacle>170)and(Lside_obstacle<30
):
            return "right"
    else:
        if Rside_obstacle>170:
            hang_flag=0
            return "right"
```

```
        elif Lside_obstacle>170:
            hang_flag=0
            return "left"
        else:
            return "back"


# taking-off
ser.write('1500150012501500')
stabilize(0,0,sensor.read_altitude()-floor_alt,365)
stabilize(0,0,200,365)

# main control an planning loop
while True:
    # Read GPS coordinates
    current_location, addr = socket_GPS.recvfrom(1024)

current_la=float(current_location[0:current_location.find(',')])

current_lo=float(current_location[current_location.find(',')+1:)]
)
    # Calculate the goal attitude

goal_attitude=get_goal_attitude(current_la,current_lo,desired_la
,desired_lo)

    # Read Ultrasonic sistances
    front, addr = socket_fr.recvfrom(1024)
    back, addr = socket_bc.recvfrom(1024)
    right, addr = socket_rs.recvfrom(1024)
    left, addr = socket_ls.recvfrom(1024)

    # is there nearby obstacles
    if front<170 or back<170 or right<170 or left<170:

avoidance_data=obs_avoidance(goal_attitude,front,back,right,left
)  # Obstacle avoidance calculation
        obst_flag=1
    else:
        yaw=goal_attitude
        obst_flag=0

    # motion excuter
        if obst_flag==1:
            if avoidance_data[0]=='s':
                yaw=int(avoidance_data[6:])
                stabilize(0,setpoint_pitch/2,setpoint_alt,yaw)
            elif avoidance_data[0]=="b":
                stabilize(0,-setpoint_pitch,setpoint_alt,365) #
send 365 in yaw to indicate stablize at the current yaw
            elif avoidance_data[0]=="l":
                stabilize(-setpoint_roll,0,setpoint_alt,365)
```

```
            else:
                stabilize(setpoint_roll,0,setpoint_alt,365)
        else:
            stabilize(0,setpoint_pitch,setpoint_alt,yaw)

    # is goal reached

    if goal_reached(current_la,current_lo):
        break
    else:
        continue

# Landing
ser.write('1500150012501500')
time.sleep(1)
ser.write('1500150011001500')
time.sleep(1)
ser.write('1500150010001500')
```

Appendix B

Arduino Code (C)

```
#include <Servo.h>
String readString, ch_input1, ch_input2, ch_input3, ch_input4;
Servo ch1;
Servo ch2;
Servo ch3;
Servo ch4;
Servo ch5;

void setup() {
  Serial.begin(115200);
  ch1.attach(5);  //attaching PWM pins
  ch2.attach(6);
  ch3.attach(9);
  ch4.attach(10);
  ch5.attach(11);


  Serial.println("Arming motors ./.");
  delay(1000);
      ch1.writeMicroseconds(1500); // Arming the motors
      ch2.writeMicroseconds(1500);
      ch3.writeMicroseconds(1000);
      ch4.writeMicroseconds(2000);
      ch5.writeMicroseconds(1500);
   delay(1000);

   Serial.println("Arming done");




}

void loop() {

  while (Serial.available()) {
    delay(3);  //delay to allow buffer to fill
    if (Serial.available() >0) {
      char c = Serial.read();  //gets one byte from serial
buffer
      readString += c; //makes the string readString
    }
  }
```

```
  if (readString.length() >0) {
      Serial.println(readString); //see what was received

      // expect a string like 07002100 containing the two
ch_input positions
      ch_input1 = readString.substring(0, 4); //get the first
four characters
      ch_input2 = readString.substring(4, 8); //get the next
four characters
      ch_input3 = readString.substring(8, 12); //get the next
four characters
      ch_input4 = readString.substring(12, 16); //get the next
four characters


      Serial.println(ch_input1);  //print to serial monitor to
see parsed results
      Serial.println(ch_input2);
      Serial.println(ch_input3);
      Serial.println(ch_input4);

      int n1 = ch_input1.toInt();
      int n2 = ch_input2.toInt();
      int n3 = ch_input3.toInt();
      int n4 = ch_input4.toInt();


      ch1.writeMicroseconds(n1); // sending the PWM commands to
CC3D
      ch2.writeMicroseconds(n2);
      ch3.writeMicroseconds(n3);
      ch4.writeMicroseconds(n4);


    readString="";  // emptying the buffer for the next command
  }
}
```

Appendix C

The Ground Station Communication Codes (python)

1. Streaming video server:

```
import socket
import cv2
import numpy

def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
        buf += newbuf
        count -= len(newbuf)
    return buf

TCP_IP = '0.0.0.0'
TCP_PORT = 5050

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(True)
conn, addr = s.accept()

length = recvall(conn,16)
stringData = recvall(conn, int(length))
data = numpy.fromstring(stringData, dtype='uint8')
s.close()

decimg=cv2.imdecode(data,1)
cv2.imshow('SERVER',decimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Streaming video client:

```
import socket
import cv2
import numpy

def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
        buf += newbuf
        count -= len(newbuf)
    return buf

TCP_IP = '0.0.0.0'
TCP_PORT = 5050

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(True)
conn, addr = s.accept()

length = recvall(conn,16)
stringData = recvall(conn, int(length))
data = numpy.fromstring(stringData, dtype='uint8')
s.close()

decimg=cv2.imdecode(data,1)
cv2.imshow('SERVER',decimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. Receiving current position and showing it on a map

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread
from socket import socket, gethostbyname, AF_INET,
SOCK_DGRAM
import sys
PORT_NUMBER = 5000
SIZE = 1024

hostName = '0.0.0.0'
```

```
mySocket = socket( AF_INET, SOCK_DGRAM )
mySocket.bind( (hostName, PORT_NUMBER) )

np.random.seed(0)
x =977 #np.random.uniform(0.0,15.0,20)
y =619 #np.random.uniform(0.0,15.0,20)
edge_lo=32.498085
edge_la=15.542107
lo_range=0.07596
la_range=0.046385
while True:
    #Show that data was received:
    (data, addr) = mySocket.recvfrom( SIZE )
    lo=float(data[:data.find(',')])
    la=float(data[data.find(',')+1:])
    plotted_lo=int(round((lo-edge_lo)*x/lo_range))
    plotted_la=y-int(round((la-edge_la)*y/la_range))

    img = imread("map.png")
    plt.scatter(plotted_lo,plotted_la,zorder=1)
    plt.imshow(img,zorder=0)
    plt.show()
```

Appendix D

The Simulation Code (Lua)

```
-- This is the autonomous quadcopter principal control
script. It is threaded
actualizeLEDs=function()
    if (relLedPositions==nil) then
        relLedPositions={{-0.0343,0,0.0394},{-
0.0297,0.0171,0.0394},{0,0.0343,0.0394},

{0.0297,0.0171,0.0394},{0.0343,0,0.0394},{0.0243,-
0.0243,0.0394},
                    {0.006,-0.0338,0.0394},{-0.006,-
0.0338,0.0394},{-0.0243, -0.0243,0.0394}}
    end
    if (drawingObject) then
        simRemoveDrawingObject(drawingObject)
    end

type=sim_drawing_painttag+sim_drawing_followparentvisibilit
y+sim_drawing_spherepoints+

sim_drawing_50percenttransparency+sim_drawing_itemcolors+si
m_drawing_itemsizes+

sim_drawing_backfaceculling+sim_drawing_emissioncolor

drawingObject=simAddDrawingObject(type,0,0,bodyElements,27)
    m=simGetObjectMatrix(ePuck,-1)
    itemData={0,0,0,0,0,0,0}
    simSetLightParameters(ledLight,0)
    for i=1,9,1 do
        if
(ledColors[i][1]+ledColors[i][2]+ledColors[i][3]~=0) then
            p=simMultiplyVector(m,relLedPositions[i])
            itemData[1]=p[1]
            itemData[2]=p[2]
            itemData[3]=p[3]
            itemData[4]=ledColors[i][1]
            itemData[5]=ledColors[i][2]
            itemData[6]=ledColors[i][3]

simSetLightParameters(ledLight,1,{ledColors[i][1],ledColors
[i][2],ledColors[i][3]})
            for j=1,3,1 do
                itemData[7]=j*0.003

simAddDrawingObjectItem(drawingObject,itemData)
```

```
            end
        end
    end
end

getLightSensors=function()
    data=simReceiveData(0,'EPUCK_lightSens')
    if (data) then
        lightSens=simUnpackFloats(data)
    end
    return lightSens
end

threadFunction=function()
    while
simGetSimulationState()~=sim_simulation_advancing_abouttost
op do
        st=simGetSimulationTime()
        velLeft=0
        velRight=0

opMode=simGetScriptSimulationParameter(sim_handle_self,'opM
ode')
        lightSens=getLightSensors()
        s=simGetObjectSizeFactor(bodyElements) -- make sure
that if we scale the robot during simulation, other values
are scaled too!
        noDetectionDistance=0.16*s

proxSensDist={noDetectionDistance,noDetectionDistance,noDet
ectionDistance,noDetectionDistance,noDetectionDistance,noDe
tectionDistance,noDetectionDistance,noDetectionDistance}
        for i=1,8,1 do
            res,dist=simReadProximitySensor(proxSens[i])
            if (res>0) and (dist<noDetectionDistance) then
                proxSensDist[i]=dist
            end
        end
        if (opMode==0) then -- We wanna follow the line
            if (math.mod(st,2)>1.5) then
                intensity=1
            else
                intensity=0
            end
            for i=1,9,1 do
                ledColors[i]={intensity,0,0} -- red
            end
            -- Now make sure the light sensors have been
read, we have a line and the 4 front prox. sensors didn't
detect anything:
            if lightSens and
((lightSens[1]<0.5)or(lightSens[2]<0.5)or(lightSens[3]<0.5)
```

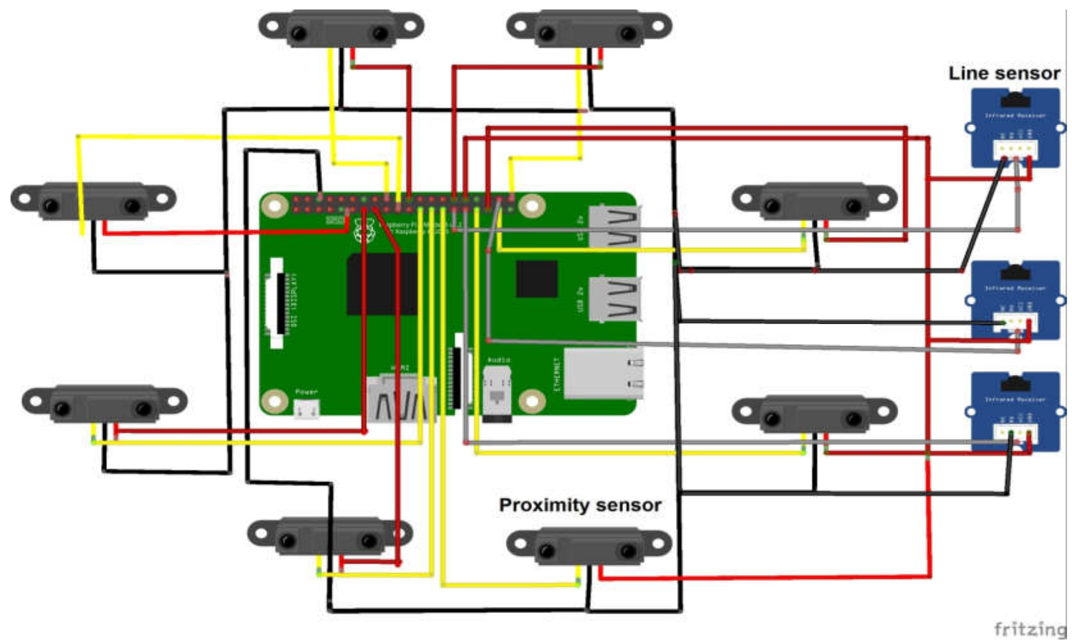
```
) and
(proxSensDist[2]+proxSensDist[3]+proxSensDist[4]+proxSensDi
st[5]==noDetectionDistance*4) then
                if (lightSens[1]>0.5) then
                    velLeft=maxVel
                else
                    velLeft=maxVel*0.25
                end
                if (lightSens[3]>0.5) then
                    velRight=maxVel
                else
                    velRight=maxVel*0.25
                end
            else
                velRight=maxVel
                velLeft=maxVel
                if
(proxSensDist[2]+proxSensDist[3]+proxSensDist[4]+proxSensDi
st[5]==noDetectionDistance*4) then
                    -- Nothing in front. Maybe we have an
obstacle on the side, in which case we wanna keep a
constant distance with it:
                    if
(proxSensDist[1]>0.25*noDetectionDistance) then

velLeft=velLeft+maxVel*braitSideSens_leftMotor[1]*(1-
(proxSensDist[1]/noDetectionDistance))

velRight=velRight+maxVel*braitSideSens_leftMotor[2]*(1-
(proxSensDist[1]/noDetectionDistance))
                    end
                    if
(proxSensDist[6]>0.25*noDetectionDistance) then

velLeft=velLeft+maxVel*braitSideSens_leftMotor[2]*(1-
(proxSensDist[6]/noDetectionDistance))

velRight=velRight+maxVel*braitSideSens_leftMotor[1]*(1-
(proxSensDist[6]/noDetectionDistance))
                    end
                else
                    -- Obstacle in front. Use Braitenberg
to avoid it
                    for i=1,4,1 do

velLeft=velLeft+maxVel*braitFrontSens_leftMotor[i]*(1-
(proxSensDist[1+i]/noDetectionDistance))

velRight=velRight+maxVel*braitFrontSens_leftMotor[5-i]*(1-
(proxSensDist[1+i]/noDetectionDistance))
                    end
                end
```

```
            end
        end
        if (opMode==1) then -- We wanna follow something!
            index=math.floor(1+math.mod(st*3,9))
            for i=1,9,1 do
                if (index==i) then
                    ledColors[i]={0,0.5,1} -- light blue
                else
                    ledColors[i]={0,0,0} -- off
                end
            end
            velRightFollow=maxVel
            velLeftFollow=maxVel
            minDist=1000
            for i=1,8,1 do

velLeftFollow=velLeftFollow+maxVel*braitAllSensFollow_leftM
otor[i]*(1-(proxSensDist[i]/noDetectionDistance))

velRightFollow=velRightFollow+maxVel*braitAllSensFollow_rig
htMotor[i]*(1-(proxSensDist[i]/noDetectionDistance))
                if (proxSensDist[i]<minDist) then
                    minDist=proxSensDist[i]
                end
            end

            velRightAvoid=0
            velLeftAvoid=0
            for i=1,8,1 do

velLeftAvoid=velLeftAvoid+maxVel*braitAllSensAvoid_leftMoto
r[i]*(1-(proxSensDist[i]/noDetectionDistance))

velRightAvoid=velRightAvoid+maxVel*braitAllSensAvoid_rightM
otor[i]*(1-(proxSensDist[i]/noDetectionDistance))
            end
            if (minDist>0.025*s) then minDist=0.025*s end
            t=minDist/(0.025*s)
            velLeft=velLeftFollow*t+velLeftAvoid*(1-t)
            velRight=velRightFollow*t+velRightAvoid*(1-t)
        end
        simSetJointTargetVelocity(leftMotor,velLeft)
        simSetJointTargetVelocity(rightMotor,velRight)
        actualizeLEDs()
        simSwitchThread() -- Don't waste too much time in
here (simulation time will anyway only change in next
thread switch)
    end
end

-- Initialization:
```

```
simSetThreadSwitchTiming(200) -- We will manually switch in
the main loop
bodyElements=simGetObjectHandle('ePuck_bodyElements')
leftMotor=simGetObjectHandle('ePuck_leftJoint')
rightMotor=simGetObjectHandle('ePuck_rightJoint')
ePuck=simGetObjectHandle('ePuck')
ledLight=simGetObjectHandle('ePuck_ledLight')
proxSens={-1,-1,-1,-1,-1,-1,-1,-1}
for i=1,8,1 do
    proxSens[i]=simGetObjectHandle('ePuck_proxSensor'..i)
end
maxVel=120*math.pi/180
ledColors={{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
{0,0,0},{0,0,0},{0,0,0}}

-- Braitenberg weights for the 4 front prox sensors
(avoidance):
braitFrontSens_leftMotor={1,2,-2,-1}
-- Braitenberg weights for the 2 side prox sensors
(following):
braitSideSens_leftMotor={-1,0}
-- Braitenberg weights for the 8 sensors (following):
braitAllSensFollow_leftMotor={-3,-1.5,-0.5,0.8,1,0,0,-4}
braitAllSensFollow_rightMotor={0,1,0.8,-0.5,-1.5,-3,-4,0}
braitAllSensAvoid_leftMotor={0,0.5,1,-1,-0.5,-0.5,0,0}
braitAllSensAvoid_rightMotor={-0.5,-0.5,-1,1,0.5,0,0,0}

-- Execute the thread function:
res,err=xpcall(threadFunction,function(err) return
debug.traceback(err) end)
if not res then
    simAddStatusbarMessage('Lua runtime error: '..err)
end

-- Clean-up:

for i=1,9,1 do
    ledColors[i]={0,0,0} -- no light
end
actualizeLEDs()
```

Appendix E

Simulation Circuit Diagram:



Prototype Circuit Diagram: