NC STATE UNIVERSITY

Final Report
Control Team 3

Firefighting Drone Challenge NCSU
Fall’14

Nachiappan Chockalingam
Ashitha Velayudhan
Rasika Kalwit
Nitin Sharma

Click to open Website

Table of Contents

2. Team

Name	**Contact**	**About**
Ashitha Velayudhan	**avelayu**@ncsu.edu	First year MCS student at NC State. Areas of interest include Algorithms, Cryptography and Computer Networks.
Nachiappan Chockalingam	**nchocka**@ncsu.edu	First year MS EE student at NC State. Areas of interest include control systems and optimization.
Nitin Sharma	**nsharm10**@ncsu.edu	First year MCS student at NC State. Areas of interest include AI and machine learning.
Rasika Kalwit	**rskalwit**@ncsu.edu	First year MS EE student at NC State. Areas of interest include optimization, gas turbines and modern control theory.

3. Challenge Overview

A drone is made up of many complex components, and it would take a lot of time to develop the whole system. Also, there are different components in a drone that can be worked on independently and require different skill sets. So, to design the drone faster and allow more time to perfect smaller modules, the drone has been divided into four components. These components will be developed simultaneously in Fall 2015 and later integrated in Spring 2015 to complete the challenge.

To facilitate the integration process, a few guidelines have been put in place that govern the communication between modules. A brief description of components is given below:

- **Sensor Module:** As the name of the module suggests, the sensor module is responsible for sensing the environment drone is operating in. It is this module that will detect objects in all directions and provide a clear area of operation under the roof. This data will be used by the control module to control drone movement indoors without collision. The module senses the environment in six - 90 degree cones (front, back, right, left, top, down). This gives a clear cuboid in which the drone can fly without hitting any obstacle. The output of this module will be six distances : front, back, right, left, top and down , and will give distance to the closest obstacle in that direction respectively.

- **Control Module:** The control module is the backbone of the system as it allows operation in indoor environment without collision. The job of this module is to control the drone and stop it from collision for any level of user operation (beginner or advanced). The inputs to the control module are : Operator commands - pitch, roll, yaw & throttle, and sensor distance readings : front, back, right, left, top & down. The output of the module is modified values of : pitch, roll, yaw & throttle , to the airframe. These modified values prevent the drone from crashing.

- **Airframe:** The main goal of this module is to design an airframe optimized for the challenge. The airframe should not be very large or heavy, as that would restrict movement of drone indoors (especially through doorframes). The airframe has to be stable, should be able to carry payload (sensors and camera) and should have some feature to sustain impact. Airframe receives four inputs from the control team - pitch, yaw, roll, throttle.

- **Operator Control and Feedback:** There are two main goals of this module:
  - Allow operator to control the drone through something like a RF operator, etc.
  - Provide operator visual feedback from the drone. This includes a real time video stream and a feature to highlight humans on the video.

The operator team gives the values of pitch, roll, yaw, throttle to the control module as input. The input is in the range of +100 to -100 for pitch, roll, yaw and 0 to 100 for throttle.

4. Goal

4.1 Goal of the challenge

The goal of this challenge is to build a working prototype of a drone capable of operating in conditions of indoor fire. Fire and smoke inside a burning building pose threats to safety of firefighters, who go in looking for victims.

This challenge strives to provide the firefighters the technological edge for scanning the building and perform rescue operations quickly and mitigate the hazards. The drone can go in the building ahead of the firefighters and provide them with video surveillance to identify danger areas.

4.2 Goal of Control Module

The control module is responsible to acclimatize the drone for indoor operation. In indoor situations, most essential feature for a drone to have is obstacle avoidance. Collisions can not only damage the drone and render it inoperable but also compromise the speed of operation.

The main goal of control module is to avoid collisions under all circumstances while giving maximum control of the drone to human operator. Work on completion of module is divided into two phases: development phase and integration phase. Development phase is to develop and test a prototype of the module on simulation. This phase is to be completed in Fall 2014. The integration phase is to integrate all components of drone together and complete the challenge. This phase is to be completed in Spring 2015.

5. Detailed Description(Control Module)

5.1 Hardware Used & Estimated Cost:

The computing chip we use is BeagleBone Black. BeagleBone Black is a low-cost, community supported development platform. It offers better processing, more memory and more IO pins for communication purposes. For technical specifications, see here.

The cost of the chip is 55 USD (Check here).

5.2 Software Requirements:

Linux is the OS on beaglebone black. To install linux follow the instructions here.

Matlab is a widely used interactive environment used to explore and visualize ideas. We use matlab for our simulation. More about matlab can be found here.

5.3 Simulation Environment:

We are using “Integrated Simulation Platform for indoor Quadrotor Applications” [1] based on Matlab. The simulation takes into account inertia and is designed specific to indoor operation.

The simulation works as : operator input and sensor values are fed to beaglebone black through matlab. The control code resides on beaglebone and after doing the necessary computations, board sends the output back to Matlab and this is used as the input to simulation environment.

5.4 Drone’s Environment:

The Figure 1 below shows the environment of the drone in the room. The outer walls as shown in the figure are given by the sensor readings in those directions. So, from the sensor readings we get a cuboid, with drone inside it (not necessarily at the center of cuboid).
This gives us a measure of clear distance in all six directions. These distances go into our algorithm (explained in section 5.6) as inputs.

This report uses the terms “red zone”, “yellow zone” and “green zone” as:

- Red Zone: This is a buffer area closest to the obstacle. The drone should not enter this area.
- Yellow Zone: As soon as the drone enters this zone, our algorithm kicks in. It starts feeding modified values of pitch, roll and throttle to the airframe and the drone eventually slows down and stops before entering the red zone.
- Green Zone: This is the area of free movement for the drone. The operators commands are directly relayed to the airframe.

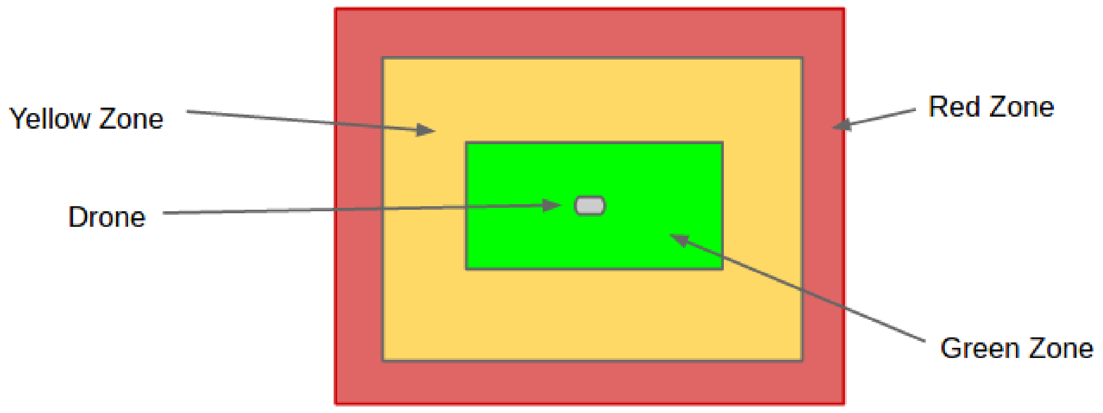


Figure 1

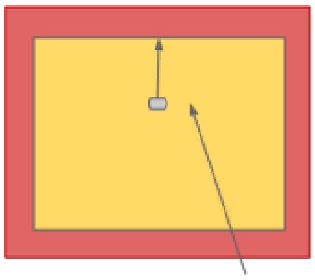
5.5 Breakdown of Tasks:

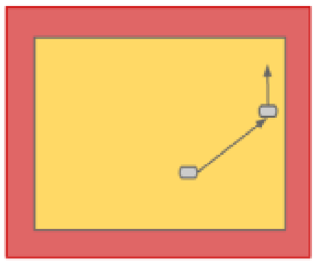
To better understand the problem at hand, a breakdown of scenarios to be supported by Control Module:

5.5.1 Avoiding Collision:

The main goal of the control module is to avoid drone from colliding. The movement of drone can be divided into **two components**: first, where the drone moves around the room (controlled using pitch and roll) and the second where the drone moves up and down (controlled using throttle).

- When the drone has just pitch or roll input given to it and it travels in the direction of the obstacle Figure 2 a: we reduce pitch or roll respectively according to the ‘negation factor’ (more about negation factor in section 5.6.3**.** Now, when the drone has both pitch and roll components Figure 2 b, we reduce both of them independently based on distance from obstacle in respective directions of movement. This way, we can stop the drone from colliding with an obstacle in one direction and yet keep moving in the other direction that operator wants it to go.
- Throttle is used to increase/ decrease altitude of drone.


When one of pitch or roll is applied, the respective values are modified to stop the drone before red zone.


When both pitch and roll are applied, the drone moves at an angle. For this example, the roll is modified to stop the drone from entering the red zone on the right but it continues moving forward.

a **b**

Figure 2

5.5.2 Maneuver through doors

For indoor operation, the most important part is getting through doors. Going through doors poses a challenge as the space is confined and getting through that space is tough even for experienced operators. Our algorithm takes pitch and roll as different components and also prevents drone from crashing in any direction. We can make use of these features to assist movement through doors (Figure 3). For example, if increasing the pitch moves the drone closer to the wall and giving roll moves the drone parallel to the wall, then to get through a wall, we need to place the drone in front of the door and move it forward by increasing the pitch. For assistance in this and mitigate the need to place the drone directly in front of the door, we can move the drone to the wall and give it small roll towards the door (drone should move slow across the wall and should have higher pitch going in as input). This way, as soon as the drone finds an open space in form of the door, it will go forward and into the room. One this to keep in mind here is that movement across the wall should be slow otherwise by the time drone senses the opening and reacts, it will be past the door.

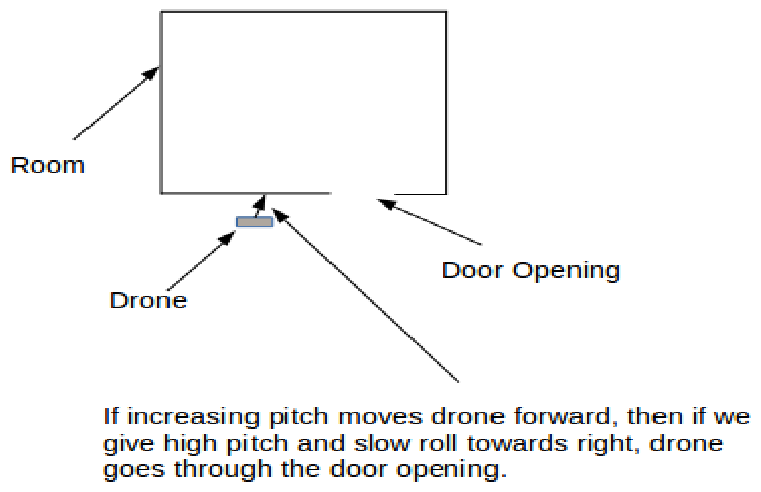

Room
Drone
Door Opening
If increasing pitch moves drone forward, then if we give high pitch and slow roll towards right, drone goes through the door opening.

Figure 3

5.5.3 Giving maximum possible control to the operator

Our algorithm allows operator to control the speed of the drone throughout (figure 4). From the environment described above, drone operates as per operator commands in the green zone. In the yellow zone, modified values of pitch and roll are given to the drone. Even in this region, the operator may speed up or slow down the drone with restriction on upper limits of pitch, roll and throttle. This upper limit varies with distance of obstacle from the drone, as the distance decreases, the upper limit values of pitch, roll and throttle come down. Finally, the drone stops before entering the red zone.

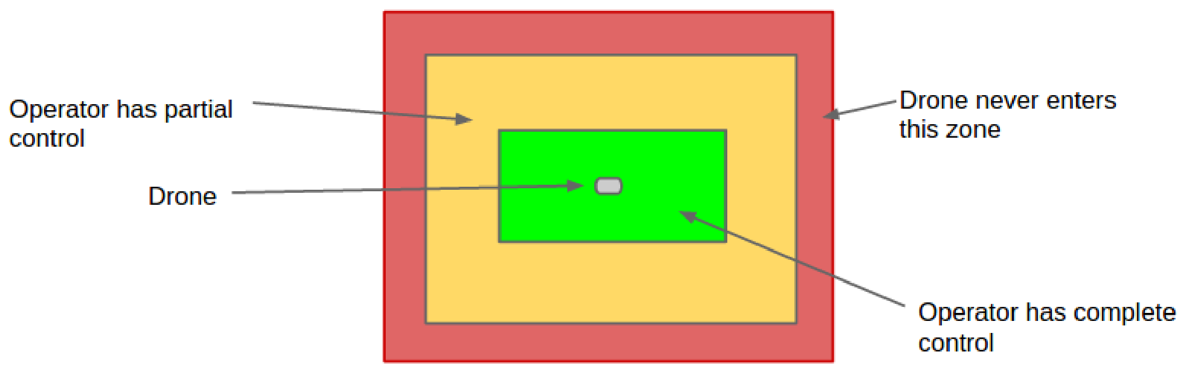

Operator has partial control
Drone
Drone never enters this zone
Operator has complete control

Figure 4

5.5.4 Avoiding moving objects

The algorithm we developed does not allow the drone to get into the red zone. So when the drone tries to get into the red zone, it is pushed away from the object. This feature enables the drone to save itself from moving objects, that might damage its operation if left unattended.

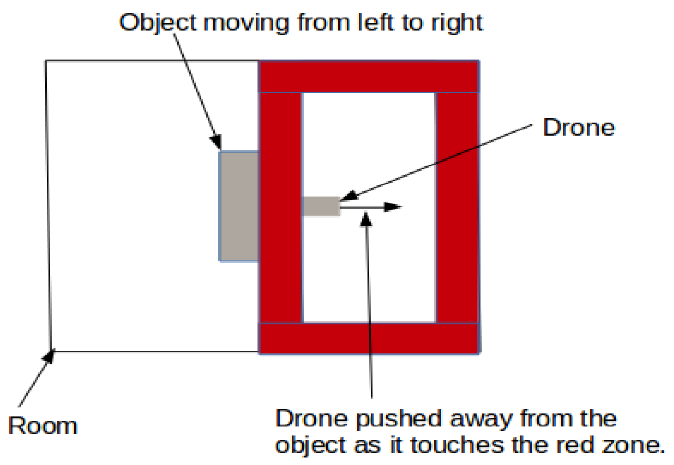

Object moving from left to right
Drone
Room
Drone pushed away from the object as it touches the red zone.

Figure 5

5.6 Control Algorithm

5.6.1 Introduction

The control algorithm that we implemented for the challenge has following features:

- The red zone from figure 1 is 0.3 m. So the drone never goes closer than 0.3 m from the wall.
- We start modifying operator command as soon as drone enters the yellow zone. The sensors can sense objects as far as 5 m. So, we have taken our yellow zone to be 5 m from the obstacle. So algorithm kicks in as soon as the drone is within 5 m from obstacle.
- Since we are operating the drone in closed room conditions, we are placing an upper limit on permitted angles of operation. The speed of drone is directly proportional to pitch and roll angles. Therefore, it limits the speed of operation. We limit the angles to 18 degrees (this can be calibrated to different values as needed).
- Frequency of Sensor data = 30 Hz.

5.6.2 Evolution of the Algorithm

We started off with two sets of inputs : sensor distance readings and airframe operator control commands. The sensor readings were distances in six directions: front, back, right, left, up and down. And operator control commands were: pitch, roll, yaw and throttle. With these two inputs, we had to figure out a way to change the operator commands to prevent drone from colliding with giving maximum control to the operator.

Given these inputs, the **first approach** we thought of is explained below:

Velocity based Control

In this method the drone is controlled based on the velocity calculated by using the distance measures from the sensor input. Since we do not have GPS or other means of measurement for actual speed/velocity of the drone, we had to estimate the velocity of the drone by using the distance measures from the sensor module and time difference between each sensor data update.

$$v_{actual} = d_{k+1} - d_k \text{ , where k is the no. of measurement, } v - \text{velocity, } d - \text{distance}$$

The maximum allowed pitch attainable by the drone is θ and with this pitch the maximum acceleration/ deceleration possible is estimated by:

$$a_{max} = g\sin\theta$$

where, g = 9.8 m/s^2.

Assuming this as the maximum deceleration/acceleration attainable by the drone we calculate a safe velocity i.e. the velocity at which we can apply the maximum deceleration and stop the motion of the drone when it reaches the outer boundary of the red zone.

$$vsafe_k = \sqrt{2 * a_{max} * (d_k - d_{safe})}$$

where, $vsafe_k$ and d_k are the safe velocity and distance, respectively, from the obstacle at the k^{th} sensor input.

Now the vsafe calculated at each sample time is used as the reference set point for the actual velocity to track, if the velocity of the drone is greater than the safe velocity. In case the actual velocity is less than the safe velocity at estimated at that time instant no control action is taken.

Incase the actual velocity is greater than the safe velocity a deceleration in the direction of motion is applied either through the pitch or roll commands . This deceleration is proportional to the difference between the vsafe and vactual .

$$\Delta a = (v_{safe} - v_{actual})^2 / 2 * (d_k - d_{safe})$$
$$\theta = \sin^{-1}(\Delta a / g)$$

this calculated θ , the pitch or roll angle is sent to the airframe module overriding the user input.

Pros

This approach would allow the drone to fly at much higher speeds, before starting to decelerate, which would thereby increase the total area covered by the drone.

Cons

Oscillations: Since we are controlling the drone to make the velocity track a calculated safe velocity, the drone tries to accelerate as and when the velocity drops below the safe velocity which results in large number of oscillations.

Stability: At the boundaries of the zones, the control action is maximum negative (Figure 6) and this happens within one time step which would drive the real drone to instability. At the boundary the control action is aggressive, which is not desirable.

We tried correcting the issues, but nothing seemed to work. As soon as we put in acceleration element, jitters came back into the system. At this point, we thought of **eliminating the acceleration and velocity components** completely.

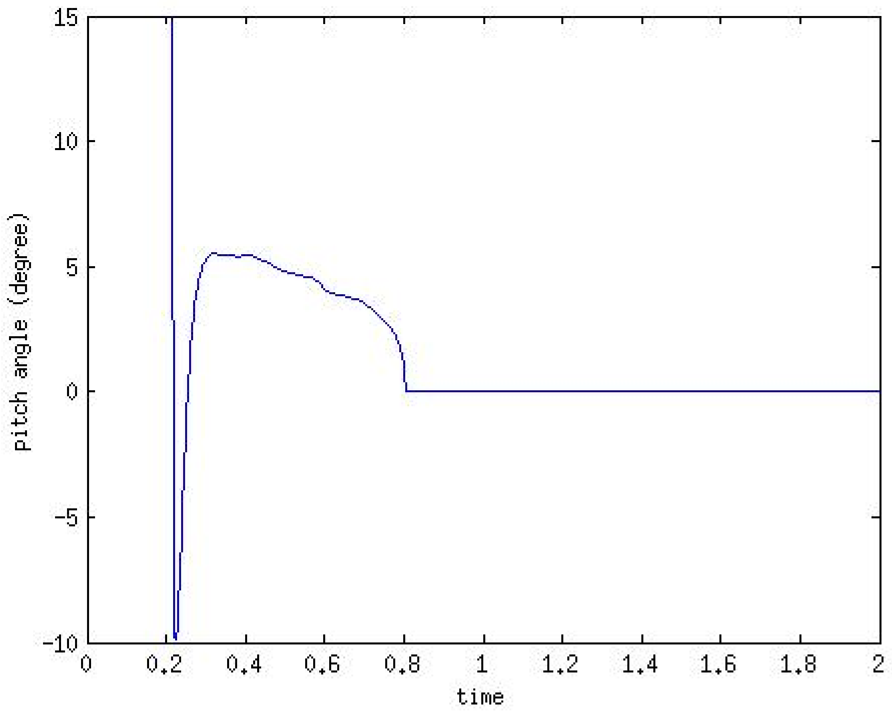


Figure 6

Second Approach (used as final):

Distance based control

We analysed the flight log data from working drones to figure out a model to help us achieve the goal. The analysis of the logs gave some insight into flying dynamics of the drone. The most important finding was : To stop a drone going with a pitch or roll angle of X, a maximum negative angle that will be needed, if we reduce the angle of operation slowly, is much less than X.

We started working on a function of pitch/roll angle with respect to distance from obstacle. What we need was to increase the retardation force as we go closer to the wall. This along with the observation from flight logs, helped us to get an equation to the negation factor (explained in section 5.6.3).

A detailed performance evaluation is given in section 5.6.5.

Third Approach

Distance from obstacle and velocity based control

As explained in the second approach, retardation is achieved by reducing operator command by a negation factor which is inversely proportional to distance from obstacle. But this negation factor only takes into consideration current distance from obstacle.

In this method, even past readings are taken into consideration. If drone is moving faster, the difference between two consecutive distance readings will be higher. In this case, speed of drone should be reduced faster by increasing negation factor.

Negation Factor = m* (angle/ (x_k +c)) + n* angle*(x_k - x_{k-1})

x_k is distance from wall on k^{th} instance
x_{k-1} is distance from wall on $(k-1)^{th}$ instance

where, m, n and c are constants that can be calibrated to get drone to work at different speeds.

Pros:

This algorithm reduces the speed of drone considering speed at which drone enters yellow zone. This eliminates the need to put limits on pitch and roll angles.
If the speed of drone at which drone enters yellow zone is higher, control algorithm takes aggressive action at beginning of yellow zone rather than at the boundary of yellow and red zone.

Cons:

Oscillations: If at any point, control action is negative that is pitch/roll angle is opposite to previous value of pitch/ roll angle, velocity at that instance becomes negative. This actually increases negation factor. Due to which a lot of oscillations are introduced. Sometimes drone went unstable due to aggressive corrective action.

5.6.3 Negation Factor

Negation Factor is the backbone of the algorithm. This factor is subtracted from the given pitch / roll angle at every sensor reading.

Negation Factor = m* (angle/ (distance from wall +c))

where, m and c are constants that can be calibrated to get drone to work at different speeds.

5.6.4 Calibrating Negation Factor

Retardation of drone is directly proportional to the negation factor. Therefore, more the negation factor, faster will the drone slow down. This gives us liberty to alter maximum operation speed. So, **at higher speeds the value of m increases and the value of c can be used to fine tune the distance from the wall at which the drone settles down**. The values can be calibrated. A few examples of effect of these values is given in the table below:

m	c	Maximum operating angle (pitch & roll)	Closest distance from the wall
1.1	2	18°	0.3m

2	1	18°	0.4m
1	0.2	8°	0.3m

5.6.5 Performance

To measure the effectiveness of the algorithm, we consider three performance criteria:

Speed

Speed is one crucial requirement for the fire fighting drone. It is necessary to maintain a good speed in order to cover more area within the limited battery time. But still we have a hard limit on the maximum speed which we can attain in the green region, in order to be able to stop the drone without crashing ,given a particular sensor range.

From our simulations assuming the sensor range is 5m and sensor input frequency is 30Hz and the maximum pitch angle restricted at 18°, we observed a maximum speed of **1.5m/s** i.e. **5.4 kmph**, which is normal human walking speed.

Higher speeds can be attained by either increasing the sensor detection range or sensor input frequency.

Proximity

This is the closest distance the drone can get to an obstacle/wall. Ideally it is required for the drone to get very close to the walls in order to maneuver through a room and get through doors or hallways. But the limitation here is that the drone needs to maintain certain minimum distance from the obstacle (walls or objects) so that it can take corrective action and avoid colliding with moving objects.

Through our simulations we were able to get as close as **0.4m** to the wall, which is evident in the graphs shown below in Figure 7. The graphs were generated for the final presentation. The minimum distance for the code submitted is 0.3m.

Stability

We measure the stability of the drone by considering the number of oscillations induced by the control algorithm. The control algorithm reduces the speed of the drone right from the yellow region gradually and some point it would start moving in the opposite direction and when the sensor data starts to increase again the drone tends to move forward again resulting in an oscillatory motion which will settle down close to the red region boundary.

In Ideal conditions we would prefer to have zero oscillations, i.e the drone stops exactly at the boundary of the red and the yellow region. From our simulations (Figure 7) we have observed that the drone settles down to a zero velocity or rest position near the red boundary within **one observable oscillation**.

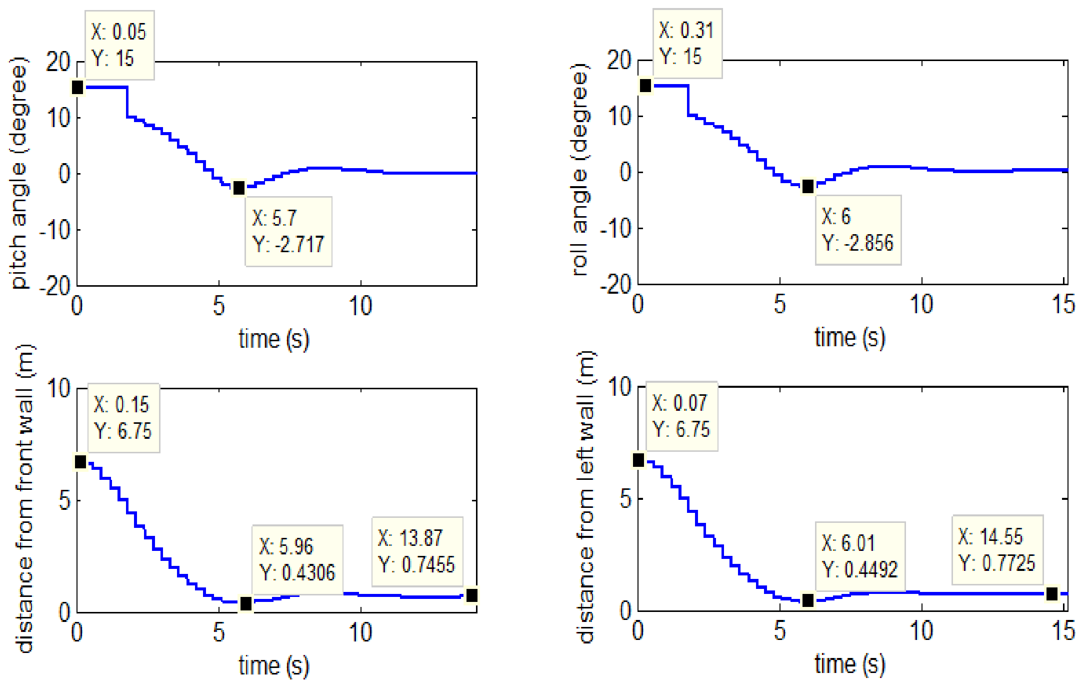


Fig.7.a) Controlled pitch command when user input pitch angle is 15 and roll =0.. b) Controlled roll command when user input pitch =0 and roll =15. (in both cases the sensor input frequency considered is 30Hz)

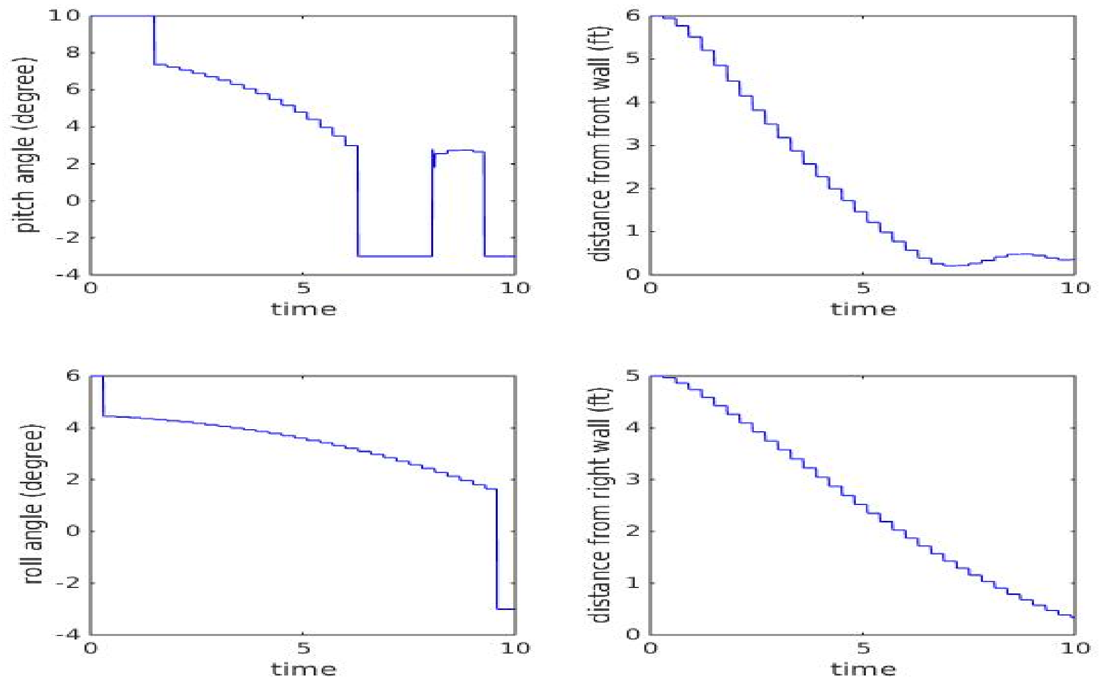


Fig 8.a) control commands when the pitch = 10 and roll = 6.This is the case when the drone slides on the wall cancelling out the pitch but keeps going right. The two graphs on the right show that. The drone stops before the red zone at less than 5 sec but the roll stops much after that..

5.7 Hardware Implementation

In the drone we suggest to use the beaglebone black as the processing device for the control algorithm. We implemented the control algorithm on beaglebone black and performed a hardware in loop simulation, where we had the operator commands and sensor inputs fed from the matlab simulation environment to the hardware controller through UDP packets and calculated airframe control commands were sent back to the simulation through UDP packets.

5.7.1 Set up BeagleBone Black

A step by step guide to re-imaging beaglebone can be found here. Beaglebone is connected to laptop using a USB cable. A UDP server resides on beaglebone and a client code on matlab. When MATLAB is run, it opens a UDP connection to beaglebone and sends data through udp objects. The operator input device and sensor data is setup in MATLAB and these are fed to the chip. The control algorithm is programmed in C and is run from the beaglebone black. This set of airframe commands are sent to MATLAB through UDP.

5.7.2 Create a UDP object

In matlab, a UDP object can be created easily using the udp function. Documentation can be found here.
An example: u=udp('192.168.7.2', 8225,'LocalPort', 9091);
creates a UDP object with destination : 192.168.7.2:8225 to source: localhost:9091.

5.7.3 Inputs to Beaglebone and Output from Beaglebone

Beaglebone is set up in this way:

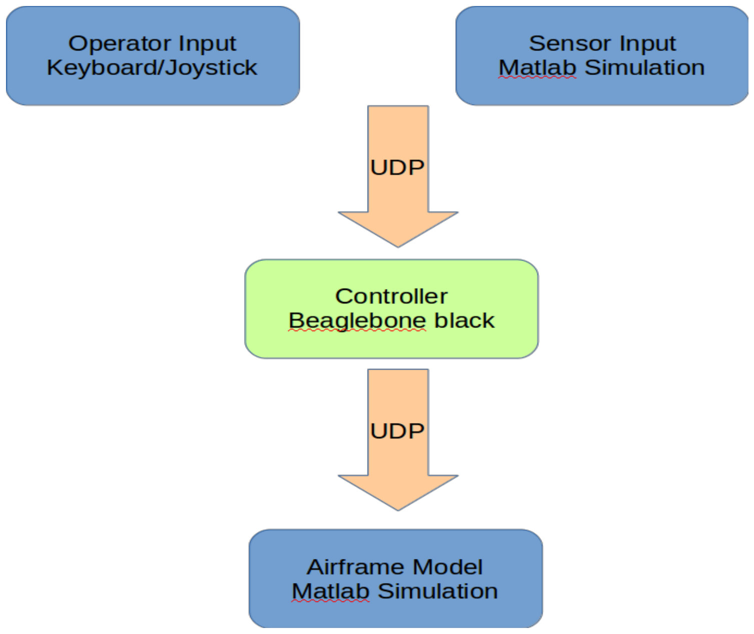


Figure 9

5.8 Operator Input

For simulation purposes, we are using keyboard and joystick (both are supported) to get input from the operator. Joystick support for matlab is available here [2].

6. Workload Distribution

Component	**Team member**
Velocity based Control Algorithm	Nachiappan
Velocity and distance based Algorithm	Rasika
Final Algorithm	Nitin
Algorithm Implementation	Nitin
Simulation & Testing	Nitin
Graph Generation	Rasika
Operator Input	Ashitha
Hardware Implementation	Ashitha & Nachiappan
Report Writing	Nitin & Nachiappan
Website	Nitin

7. Future Works

Bleaglebone supports only front movement of the drone. So first thing we need to do next semester is to extend the same for other directions.Also, different components will be integrated together to make a working prototype of a drone capable of assisting firefighting operations.

8. References

[1] "Integrated Platform for Indoor Quadrotor Applications" by Muhannad A.R. Al-Omari, Mohammad A. Jaradat and Mohammad Jarrah. Link here.

[2] Joystick support for MATLAB. Source here.