UNIVERSITY *of the* WESTERN CAPE

Multi Drone Task Allocation (MDTA)

Project presented in partial fulfillment
of the requirements for the degree of
Bachelor of Science (Honours)
at the University of the Western Cape

Author: Takudzwa Chakanyuka

Supervisor: Prof Antoine Bagula
Co-Supervisor: Mr Mehrdad Ghaziasgar

November 20, 2015

Abstract

In this paper we explore autonomous control of multiple Unmanned Ariel Vehicles(UAV) [1] and task distribution among the Unmanned Ariel Vehicles. This paper is based on our project, Multi-Drone Task Allocatio (MDTA). The motivation for this project includes the fact that UAVs can perform tasks that are too dangerous, expensive or difficult to be performed by humans. Also the fact that it is especially advantageous to use multiple robots when tasks are not order-dependant, because then multiple tasks can be performed simultaneously and the mission will take less time to complete.
The key challenges are:

- Developing an overal system that can perform optimal control of the Unmanned Ariel Vehicles
- Evaluation of the performance of the UAVs
- Collision avoidance among UAVs
- Reconfiguration of the UAVs in cases of failure
- Avoidance of restricted

Therefore an overview of the issues to be solved includes:

- UAV path planning
- UAV collision detection and avoidance
- UAV task allocation and restricted area avoidance

Our main contribution is to provide a solution that solves the issues that we have just outlined and other contributions are investigating the field readiness of UAVs, specifically Parrot Bebob drones. In order to get an understanding of the field readiness of the drones we conduct tests to uncover drone autonomy, maximum range of the drones in terms of distance travelled and altitude, drone throughput by using drone as a data mule and the effects of weather conditions especially wind on drone performance. The results will be shown in later sections of this paper.

[1]Through out this paper the terms Unmanned Ariel vehicle and drone are interchangeable

Key words

Unammaned Ariel Vehicle(UAV) or Drone

Autonomous UAV control

UAV collision detection

UAV collision avoidance

Restricted area avoidance

Path planning

Task allocation

Acknowledgments

I would like to thank my supervisors Prof Antoinne Bagula and Mr Mehrdad Ghaziasgar for their support, knowledge and patience during my project and studies. I also want to thank Freedwell Shingange with whom I am doing my project with, for his excellent work ethic and dedication to the success of our project. Without the help of these individuals I would certainly not be where I am today, once again thank you. Special thanks to Mr Andries Kruger, Mr Daniel Leënderts and Mr Andre Henney for thier support and contributions to my project. I also want to thank all the other teaching and non-teaching staff in the computer science department for their support throughout my studies. Lastly I would like to thank my family and the Lord God Almighty for helping me through the trying times and their support throughout my life.

Contents

1 Project Proposal **1**
1.1 Introduction 1
1.2 Motivation 1
1.3 Objectives 3
1.4 Deliverables 3
1.5 Literature review 3

2 User Requirements **6**
2.1 User's view of the problem 6
2.2 Description of the problem domain 6
2.3 What is expected from a software solution 6
2.4 What is not expected from the software solution 7

3 Requirements Analysis Design **8**
3.1 Interpretation of the problem 8
3.2 Breakdown of Problem 8
3.3 Deep Analysis 10

4 Existing And Alternative Solutions **11**
4.1 Autonomous drone flight by Lehata 11
4.2 A Distributed Tasks Allocation Scheme in Multi-UAV Context . 11
4.3 Complexity in UAV cooperative control 11
4.4 Cooperative Real-Time Search and Task Allocation in UAV Teams 12
4.5 Coordination and Control Of Multiple UAVs 12
4.6 Multiple UAV Task Allocation using Negotiation 13

5 Methodology To Test the Solution **14**

6 User Interface Specification **15**
6.1 Introduction 15
6.2 Description of the user interface 15

6.3 GUI elements 15
6.4 Description of GUI elements 16
6.4.1 Welcome screen 16
6.4.2 Output display section 17
6.4.3 Information and Hints sections 17
6.4.4 Navigation links 17
6.4.5 Search and retrieval interface 18
6.4.6 Data entry and verification interface 18
6.5 How the user interacts with the interface 19
6.6 How the user interface behaves 19

7 High Level Design (HLD) 20
7.1 Description of Subsystems and Operations 20
7.1.1 Input of mission planning data 20
7.1.2 Target search 20
7.1.3 Task execution 21
7.1.4 Drone monitoring 21
7.1.5 Data display 21
7.2 Interaction between Subsystems 21
7.3 Low Level Design 21
7.3.1 Get Location 21
7.3.2 Get Restricted 23
7.3.3 Verify Data 23
7.3.4 Calculate Distance 23
7.3.5 Determine Shortest Path 23
7.3.6 Recalculate Shortest Path 23
7.3.7 Connect Drones 24
7.3.8 Navigate drone 24
7.3.9 Collision Avoidance 24
7.3.10 Check drone status 24
7.3.11 Set up Transfer 24
7.3.12 Transfer Data 24
7.3.13 Task reallocation 25
7.3.14 Task Division 25
7.3.15 Process Data 25
7.3.16 Display Data 25

8 Code Documentation 26
8.1 Introduction 26
8.2 Travelling salesman pseudo code 26
8.3 Function documentation 27

8.4 Collision Avoidance Pseudo-code 29
8.4.1 Predetermined collission avoidance 29
8.4.2 Real time collision avoidance 29
8.5 Restricted Area Avoidance 29

9 Performance Testing And Reporting 31
9.1 Drone GPS Vs Waspmote GPS 31
9.2 Data mule scenario testing 32
9.2.1 Factors affecting upload and download speed 32
9.3 Drone Camera Sensor vs Mobile Camera 32
9.4 Unit Testing 34
9.4.1 Testing drone forward movement 34
9.4.2 Testing drone rotation 34
9.4.3 Test drone collision avoidance 35
9.4.4 Video stream testing 35
9.5 Travelling salesman performance 36
9.6 Integration testing 38
9.6.1 Single Thread testing 38
9.6.2 Master thread testing 39
9.7 Destructive and stress testing 41
9.7.1 Testing the range at which drone flies out of range 41
9.7.2 Testing drone power failure 41
9.7.3 Fault injection: GPS coordinate testing 43

10 Conclusion 44
10.1 Observations 44
10.2 Future work 45

11 User Guide 46
11.1 Pre-requisites 46
11.2 Drone setup 46
11.2.1 Connecting drone to pc 46
11.2.2 Change drone static IP address 47
11.2.3 Drone battery 47
11.3 Installing an extra wifi dapter 47
11.4 Requirements before flying drones 48
11.4.1 Open space 48
11.4.2 Outdoor operation 48
11.5 Running our program 48

List of Figures

6.1 Welcome screen 16
6.2 Output screen 17
6.3 Input screen 18

7.1 Interaction between Subsystems 22
7.2 Low level Design 22

8.1 Function Documenation 28
8.2 Restricted Area Avoidance 30

9.1 GPS Comparison 31
9.2 UAVcam 33
9.3 CellCam 33
9.4 Testing drone rotation 35
9.5 Snap of the video stream 36
9.6 Drone Status and Stream 36
9.7 Single drone path 39
9.8 Multiple drone path 40

Glossary

MDTA	Multi Drone Task Allocation
GPS	Global Positioning System
UAV	Unmanned Aerial Vehicle

Chapter 1

Project Proposal

1.1 Introduction

The capabilities and roles of Unmanned Aerial Vehicles (UAVs) are evolving, and require new concepts for their control. In order to control a fleet of drones in a coorperative effort to finish a given task we typically require several operators, but in this paper we propose a software solution that can perform optimal coordination of drones, monitor the drones during task execution, and quickly reconfigure to account for changes in the fleet. Our proposed software solution will basically be conducting the task of Multi drone task allocation. Where we define Multi drone task allocation (MDTA) as the processes involved in coordinating a team of drones to cover any given set of areas in a near-optimal manner with the objective of completing tasks such as environment mapping [1], surveillance [2, 3], sensor deployment [4], acting as communications hubs for immobile wireless sensor networks or aerobiological sampling.

1.2 Motivation

The use of drones is justified in situations where the task to be performed is too dangerous, expensive or difficult to be performed by humans, or can be performed more cheaply or efficiently. There are many such applications, which are the subject of ongoing research and development.
The main motivation for using multiple drones is seen as illustrated by advantages listed below:

1. Efficiency:
   It is especially advantageous to use multiple robots when tasks are not

order-dependant, because then multiple tasks can be performed simultaneously and the mission will take less time to complete.

2. Robustness through redundancy:
   The energy and computation required to perform the mission is distributed across team members, so if one member ceases to function, its role can be assigned to another.

3. Flexibility:
   The required functionality can be distributed across robots, although this advantage only becomes apparent when the mission is composed of diverse tasks: a team of robots with different specialisations, i.e. a heterogeneous team, can be engineered more cheaply and easily than a single robot capable of performing each kind of task.

The interest in drone technology and the relevence of research in drones can be seen in the applications of drones as of today as listed below:

1. Search and rescue:
   In search and rescue operations calling drone as air support has become a possibilty.When firefighting, for example, you can determine the proportion of a particular gas present using sophisticated measuring equipment and react appropriately.

2. Inspections:
   Saving on the high cost of inspections with drones is also another possibility. Wind turbines, power lines, pipelines, etc. can be inspected using a drone. No need to engage an external contractor.

3. Security:
   Many authorities worldwide are already using aerial platforms, for example to help coordinate security operations or preserve evidence.

4. Aerial video and photography:
   Creating professionals film from a bird's eye view or creating videos from unusual perspectives, it's all possible with a drone.

5. Delivery:
   As seen in the case of amazon, delivery of goods using drones is now a possibility.

All of the above has inspired us to get into drone research and tackle our topic of Multi-Drone Task allocation.

1.3 Objectives

The project aims to derive a solution in the form of a program or set of programs that can be used by drones to do the following:

- Conduct target search using multiple drones
- Divide a specific task between multiple drones
- Avoid drone collision
- Avoid restricted areas

1.4 Deliverables

- software solution for MDTA
- User manual

1.5 Literature review

Aaron Gage focused on Multi-Robot Task Allocation. This paper addresses simular isuues that we will be addressing with Multi Drone Task Allocation. Gage's literature defines Nulti-Robot Task Allocation as the assignment of tasks to mobile robots in a cooperative team. This is simular to our Multi Drone Task Allocation problem, which we can define as the assignment of tasks to drones and management of task execution. The only difference is that in Gage's project they are using various diffferent robots and we are using simular robots, namely ar parrot drones. The robot team used by Gage consists of at least one unmanned aerial vehicle (UAV) and multiple unmanned ground vehicles, while our project uses unmanned aerial vehicles only. Gauge identified several issues that make MRTA challenging, the issues that apply to our project are as follows. The first issue is unreliable communication or control channels e.g wireless networks, that may periodically fail, and can saturate if used heavily. Another isuues is that information required to make informed decisions is often spread across the robot team. It may not be possible to reassign a robot to a new task once it has a task. This means that if one robot fails, we are not guranteed to find another robot to do the task that the failed robot was doing. This is because the robots in the team might be preoccupied with their own tasks.

The approach taken by Gage is that, one robot requests the assistance of another robot in order to complete a task. In Gage's approach the task allocation is based on the fitness of the robot for the task and a concept they call affective recruitment. Fitness of the robot for the task basically enquires if the robot is done with it's own task and can be assigned a new one, also checks the distance between the robot and objective location where it needs to conduct the task.It also cheks if the robot is capable of doing the task, in our project this will always hold true. All of our robots are pressumed to be able to conduct the task. In oder to understand affective recruitment, we need to understand affective computing. Affective computing is the study and development of systems and devices that can recognize, interpret, process, and simulate human affects. Where affect is the experience of feeling or emotion. Gage simulates emotions such as hunger, shame and frustratuion. The robots get frustrated if they can not complete a task in a given time and trigger them to increase their speed in excution of tasks. The robot feels hungrary if it is out of power. The robot feels shame level increases when it does not help others. In affective recruitment gage checks the enotion of the robots in order to make task allocations. For example he uses the shame level to force a robot to respond to help request from other robots. If a robot's shame level get to a certain point, it will be forced to help other robots.Gage also checks the fitness of the robot before allocating a task to it. If a robot is too far from the location it has to do a task, he will skip it and assign the task to another robot. Gage's approach can be used in our research but instead of making robots send help request to each other, we can make them send help requests to an application and the application will do the task allocation making use of fitness of robot and affects.

Complexity in UAV cooperative control addresses complexity and coupling issues in cooperative decision and control of distributed autonomous UAV teams. In particular, the results obtained by the inhouse research team are presented. Hierarchical decomposition is implemented where team vehicles are allocated to subteams using set partition theory. Results are presented for single assignment and multiple assignment using network flow and auction algorithms. Simulation results are presented for wide area search munitions where complexity and coupling are incrementally addressed in the decision system, yielding radically improved team performance.

Distributed Tasks Allocation Scheme in Multi-UAV Context focuses on the task allocation problem in multi-robot systems. In this project they identified two distinct problems that needed to be solved, namely allocation and planning of navigation tasks and the problem of constraints on tasks schedules which enable the system to deal with complex tasks that need synchronization of robots activities. This project can be used for the task allocation in our project.[5]

Cooperative Real-Time Search and Task Allocation in UAV Teams considers a heterogeneous team of UAVs drawn from several distinct classes and need to engage in a search and destroy mission over an extended battlefield. Several different types of targets are considered. Some target locations are suspected with a certain probability, while the others are initially unknown. During the mission, the UAVs perform Search, Confirm, Attack and Battle Damage Assessment (BDA) tasks at various locations. The target locations are detected gradually through search, while the tasks are determined in real-time by the actions of all UAVs and their results (e.g.,sensor readings), which makes the task dynamics stochastic. The tasks must, therefore, be allocated to UAVs in real-time as they arise. Each class of UAVs has its own sensing and attack capabilities with respect to the different target types, so the need for appropriate and efficient assignment is paramount.

Coordination and Control Of Multiple UAVs addresses the problems of autonomous task allocation and trajectory planning for a fleet of UAVs. Two methods are compared for solving the optimization that combines task assignment, subjected to UAV capability constraints, and path planning, subjected to dynamics, avoidance and timing constraints. Both sub-problems are non-convex and the two are strongly-coupled. The first method expresses the entire problem as a single mixed-integer linear program (MILP) that can be solved using available software. This method is guaranteed to find the globally-optimal solution to the problem, but is computationally intensive. The second method employs an approximation for rapid computation of the cost of many different trajectories. This enables the assignment and trajectory problems to be decoupled and partially distributed, offering much faster computation.

Chapter 2

User Requirements

2.1 User's view of the problem

Multi drone task allocation is the process of controlling multiple drones, dividing a specific workload among the drones and the continuous coordination and management of the drones during task execution. The user needs a system that will automate the process of coordinating multiple drones to do a specific task.

2.2 Description of the problem domain

A system is required that can automatically fly multiple drones to a specific location, given the location's GPS coordinates. The system must coordinate the drones to avoid drone collision and avoid locations marked as restricted points by the user. The drones must get to specified locations in optimal time and using best route. Upon the drones' arrival to the location, the system must divide the given task among the drones and ensure the overall task is done efficiently.

2.3 What is expected from a software solution

- The software should be able to move a drone in all directions (up, down, left, right, forward and backwards).
- The software should make allowances for collision detection and avoidance.

- The software should make allowances for restricted area avoidance.
- The software must be able to divide a given task among the drones.

2.4 What is not expected from the software solution

- Object detection and avoidance

Chapter 3

Requirements Analysis Design

3.1 Interpretation of the problem

The solution has to complete the task of target searching or visitation using multiple drones. The solution also has got to conduct multi drone task allocation. During the target searching task, the system will be given number of points denoted by a specific set of GPS coordinates in an area and the system must find or map the optimal route to get to the points. The solution must then provide each drone a route to the given points and they must get to the points in optimal time.
For the multi drone task allocation the solution must be able to divide a given task among the drones and ensure that the overall task is still completed in optimal time and efficiently. This entails that the solution will have to make sure drones are not repeating tasks that have already been done by the other drone. Also in the event of one of the drones failing, the solution must detect the point of failure and rectify it by sending another drone to complete the task of the drone that has failed. The solution must make sure there is no drone collision and all restricted areas as defined by the user, are avoided during target search and afterwards.[6]

3.2 Breakdown of Problem

The problem can be broken down into the following parts:

1. Conducting target search using multiple drones
2. Dividing tasks between multiple drones

3. Avoiding collision between the drones
4. Avoiding points marked as restricted areas
5. Drone failure detection
6. Drone task reassignment
7. Data collection

3.3 Deep Analysis

1. Conducting target search using multiple drones:
   The user needs to provide GPS coordinates of the locations that the drones need to visit. The user must then clearly define restricted points and provide their locations and GPS coordinates too. The desired solution should then map out a route for the drones to follow in order to get to the desired location input by the user. The mapped out route must ensure that drones do not enter or move in restricted areas.

2. Dividing tasks between multiple drones:
   The desired solution must be able to divide a given task between the drones, ensuring that the overall task is completed in optimal time and efficiently. This entails that the drones must not repeat tasks that have already been done by other drones.

3. Avoiding collision between the drones:
   This part of the problem requires the desired solution to implement a method that will allow the drones to detect one another when they are at a close proximity to one another. The solution should then ensure that the drones stay a safe distance from one another to avoid collision.

4. Avoiding points marked as restricted areas:
   The desired solution has got to track the position of drones and ensure that they do not enter into restricted zones.

5. Drone failure detection:
   The desired solution has to implement a method to monitor drones, to check if they are still functional.

6. Drone task reassignment:
   This requires drone failure detection and counter measure provision. The solution will need to detect the drone that has failed and then map out which of the drones are still functional. The system then needs to assign the task of the failed drone to one of the functional drones.

7. Data collection:
   This entails retrieving data from the drones once they have completed their task.

Chapter 4

Existing And Alternative Solutions

4.1 Autonomous drone flight by Lehata

The project aims to convert a quadcopter into an autonomous vehicle capable of using computer vision to identify and approach beacons placed in its environment. The project can be used to to help our drones identify restricted areas and target locations, if we attach beacons at these points. [7]

4.2 A Distributed Tasks Allocation Scheme in Multi-UAV Context

The project focuses on the task allocation problem in multi-robot systems. In this project they identified two distinct problems that needed to be solved, namely allocation and planning of navigation tasks and the problem of constraints on tasks schedules which enable the system to deal with complex tasks that need synchronization of robots activities. This project can be used for the task allocation in our project.[5]

4.3 Complexity in UAV cooperative control

This project addresses complexity and coupling issues in cooperative decision and control of distributed autonomous UAV teams.In particular, the results obtained by the inhouse research team are presented. Hierarchical decomposition is implemented where team vehicles are allocated to subteams using set partition theory. Results are presented for single assignment and multiple assignment using network flow and auction algorithms. Simulation results are presented for wide area search munitions where complexity and coupling are incrementally addressed in the decision system, yielding radically improved team performance.[8]

4.4 Cooperative Real-Time Search and Task Allocation in UAV Teams

They consider a heterogeneous team of UAVs drawn from several distinct classes and need to engage in a search and destroy mission over an extended battlefield [9]. Several different types of targets are considered. Some target locations are suspected with a certain probability, while the others are initially unknown. During the mission, the UAVs perform Search, Confirm, Attack and Battle Damage Assessment (BDA) tasks at various locations. The target locations are detected gradually through search, while the tasks are determined in real-time by the actions of all UAVs and their results (e.g.,sensor readings), which makes the task dynamics stochastic. The tasks must, therefore, be allocated to UAVs in real-time as they arise. Each class of UAVs has its own sensing and attack capabilities with respect to the different target types, so the need for appropriate and efficient assignment is paramount.

4.5 Coordination and Control Of Multiple UAVs

They address the problems of autonomous task allocation and trajectory planning for a fleet of UAVs [10]. Two methods are compared for solving the optimization that combines task assignment, subjected to UAV capability constraints, and path planning, subjected to dynamics, avoidance and timing constraints. Both sub-problems are non-convex and the two are strongly-coupled. The first method expresses the entire problem as a single mixed-integer linear program (MILP) that can be solved using available software. This method is guaranteed to find the globally-optimal solution to the problem, but is computationally intensive. The second method employs an approximation for rapid computation of the cost of many different trajectories. This enables the assignment and trajectory problems to be decoupled and partially distributed, offering much faster computation.

4.6 Multiple UAV Task Allocation using Negotiation

They propose a negotiation scheme that efficiently allocates tasks for multiple UAVs [11]. The performance of the negotiation based task allocation is studied on a battle field scenario. We study the effect of sensor ranges on the task allocation and compare the results with that of greedy strategy and a variation of the negotiation mechanism. The results show that the negotiation based task allocation mechanism performs far better than the greedy strategy. Negotiation with target information exchange between neighbours performs better than without target information exchange, when the sensor radius is low.

Chapter 5

Methodology To Test the Solution

In order to test our solution, we will conduct various simulations. When we get satisfactory results from the simulation, we will then do actual field testing by operating multiple drones with the developed solution and sending them to collect data from sensors placed in different locations. We will then evaluate the drones performace in both target visitation and task allocation by looking at the times taken to complete the tasks. We will also compare field test results to simulation results. Lastly we shall compare our results to the results of other existing solutions.

Chapter 6

User Interface Specification

6.1 Introduction

This section will describe exactly what the user interface is going to look like, functionality and how the user is going to interact with the program. In the application or program that we are developing, we intend to be using a graphical user interface. Therefore, our user interface specification is going to define the components that the user will see when interacting with our program, this includes all the menus, heading, submenus and all the options included in such components. We are also going to describe the presentation of data of data in the program.

6.2 Description of the user interface

The user interface for our application is designed to be provide high usability and provide excellent functionality without confusing the user. Thus the user interface has been designed with goals to ensure that the application the user will interact with, will be user-friendly and intuitive or easy to grasp. In order to fulfill those goals we have incorporated the following elements into our user interface.

6.3 GUI elements

- Welcome screen
- Information and Hints sections
- Navigation links

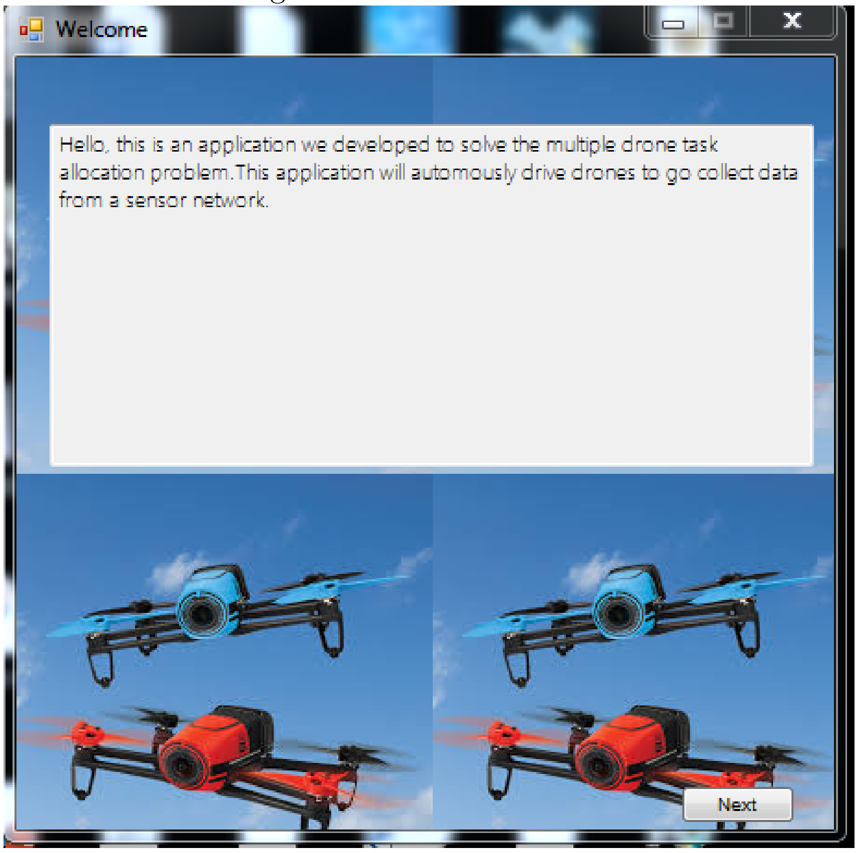
- Search and retrieval interface
- Data entry and verification interface
- Output display section
- Results storage section

6.4 Description of GUI elements

6.4.1 Welcome screen

This is the first page that the user will see when they open the interface. It will provide a brief description of the application. It will also serve the purpose of informing the user on the types of data to be used on input sections and other information the user might need to know before using the application. An example of the information will be format of GPS coordinates.

Figure 6.1: Welcome screen

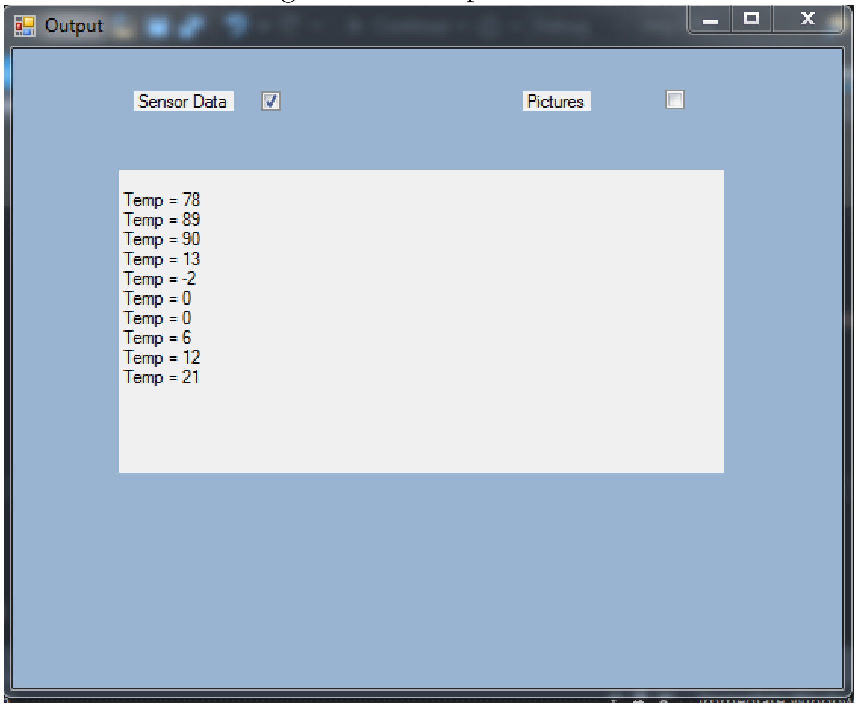


Hello, this is an application we developed to solve the multiple drone task allocation problem.This application will automously drive drones to go collect data from a sensor network.

6.4.2 Output display section

This is the part of the interface that will display the pictures taken by the drone as well serve the functionality of displaying data collected by the drones. This section will also serve as a mechanism of providing feedback to the user on things such as the status of the drones or displaying errors that can are encountered by the system, for example drones going out of range or errors during data collection.

Figure 6.2: Output screen



```
Temp = 78
Temp = 89
Temp = 90
Temp = 13
Temp = -2
Temp = 0
Temp = 0
Temp = 6
Temp = 12
Temp = 21
```

6.4.3 Information and Hints sections

The user interface will provide hints on how the user is supposed to interact with the various sections and features included in the interface. The hints provided can include things such as click a button to get data or enter the number of points to be visited by the drone etc.

6.4.4 Navigation links

The user interface will provide links to different sections of the application, at the top of current window for easy navigation to different parts of the application. Navigation links will also be provided in various positions depending on current section and functionality to be provided by link.

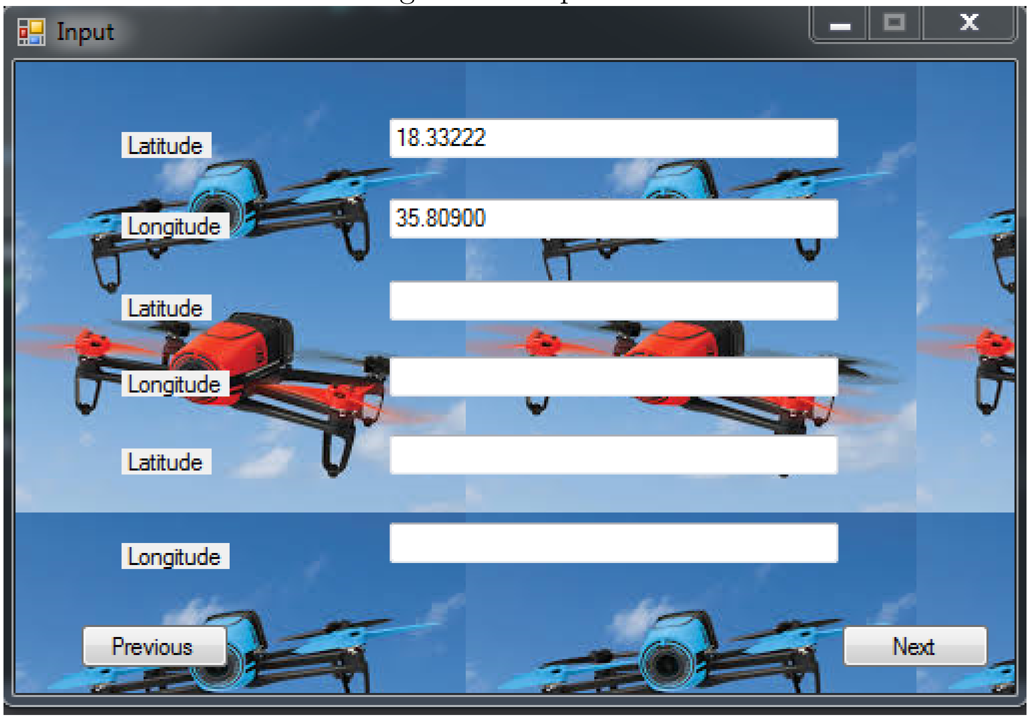
6.4.5 Search and retrieval interface

This will allow the user to get collected data easily. This section of the interface will basically allow the user to avoid sequentially searching for collected data and offer quicker data retrieval through a search interface.

6.4.6 Data entry and verification interface

This is the part of the interface that will allow the user to enter data that is required by the application. The section will allow the user to enter GPS coordinates, number of points to be visited and number of restricted areas as well as the corresponding coordinates. The interface will allow the user to verify if data entered was correct and if the data was entered incorrectly, will allow the user to reenter data to rectify the incorrect section or portions.

Figure 6.3: Input screen



Latitude	18.33222
Longitude	35.80900
Latitude	
Longitude	
Latitude	
Longitude	

6.5 How the user interacts with the interface

The application is using a graphical user interface, therefore the user interacts with the application by navigating with the input devices and clicking and checking on the various interactive components of the graphical user interface. Users will allow be able to enter data into the application through the text input boxes.

6.6 How the user interface behaves

The user interface will basically be waiting for events to be generated when users click, select check or input data into the interactive components of the graphical user interface, and the interface will provide responses by launching event handlers for the generated events. The event handlers will perform actions such as opening a new window, submitting input and writing input data to text files.

Chapter 7

High Level Design (HLD)

The HLD shows us that this application has the following key components:

- Input of mission planning data
- Target search
- Task execution
- Drone monitoring
- Data display

7.1 Description of Subsystems and Operations

7.1.1 Input of mission planning data

This part of the application will allow the user to enter the number of points to be visited as well as GPS coordinates of those locations that have to be visited. This component also allows users to enter the number of restricted areas as well as the corresponding GPS coordinates.

7.1.2 Target search

This component will be responsible for executing the code that will allow the map out the path to be taken by the drones to get to their desired locations. It will then move the drones according to the mapped out path to the target locations.

7.1.3 Task execution

It is the part of the application that is responsible for making the drones retrieve data from the sensors or take pictures when they get to the desired locations.

7.1.4 Drone monitoring

This part of the application is responsible for avoiding drone collisions. The drone monitoring module will also be responsible for drone failure detection and task reassignment in cases of failure. The module will also take the responsibility of restricted area avoidance.

7.1.5 Data display

This is the module that is responsible for displaying the data that has been retrieved for the sensors or display the pictures taken by the drones.

7.2 Interaction between Subsystems

The application starts by running the input module, then the GPS coordinates of the locations to be visited as well as restricted points, are passed on to the target search module. The target search module will then map out the drone flight plan using the data from the input module. The target search module will then call on the drone monitoring module and then move the drones according to the path. When the drones get to the target locations, the application launches the task execution module. The drone monitoring module will still be running and will continue running, so that it will now be running in parallel with the task execution module. The task execution will fulfil its goal and retrieve the data, then call target search module to drive the drones back to original location. The drone monitoring module will then be stopped and data display module will then be launched.

7.3 Low Level Design

7.3.1 Get Location

This function will get the GPS coordinates of all the locations that will need to be visited by the drone. The coordinates will be entered as latitude and longitude pairs, these will be stored in a text file and can be reused in the

Figure 7.1: Interaction between Subsystems

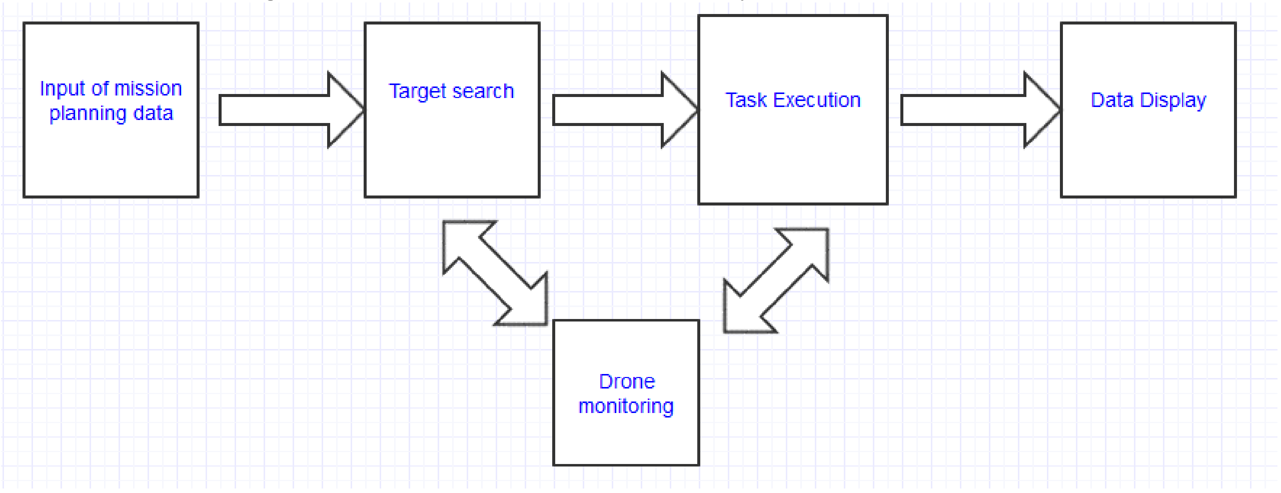
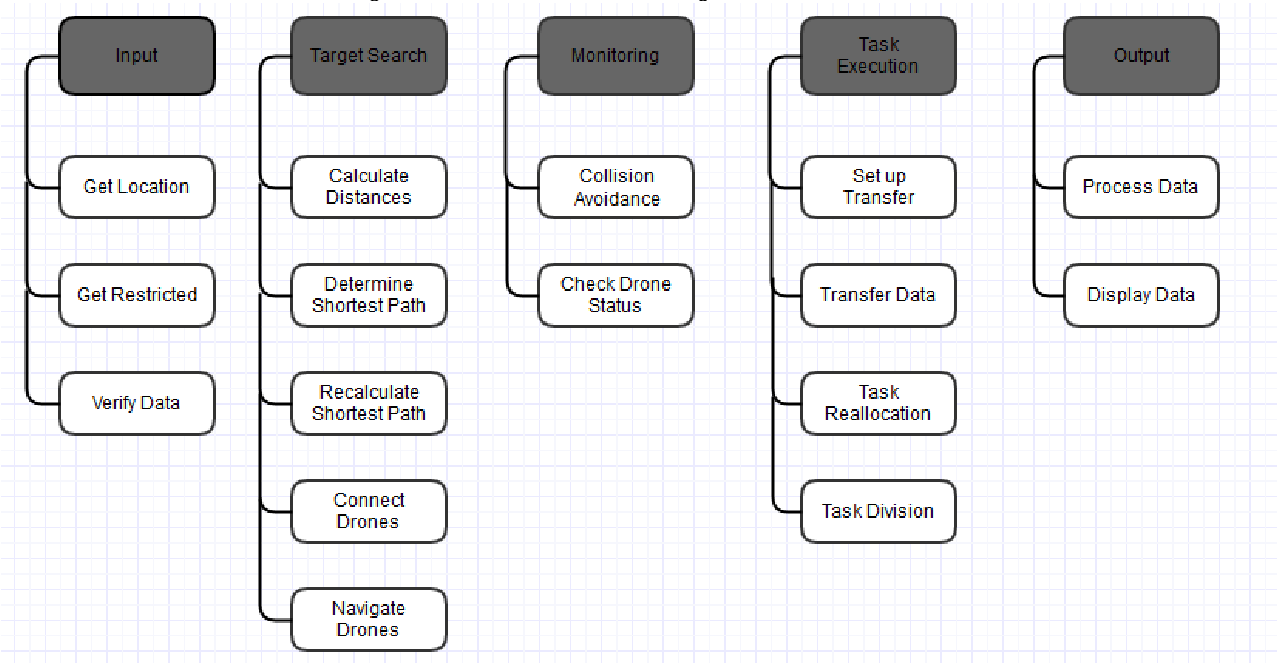


Figure 7.2: Low level Design



future. Therefore this function will also allow GPS coordinates to be entered from a text file.

7.3.2 Get Restricted

Get Restricted is a function that will have a similar functionality to Get Location, but Get Restricted will be getting the coordinates of the restricted areas. This function is neccesary because restricted points have extra information like the radius of the area they cover. This information needs to be captured and stored differently in our program, and hence the need of the Get Ristricted function.

7.3.3 Verify Data

This will allow the user to verify GPS coordinates for the locations to be visited and the restricted points. This will entail accessing the text file where the coordinates are stored and redisplaying them for the user. The user will then change the data accordingly if there were any mistakes.

7.3.4 Calculate Distance

Calculate distance will calculate the distance between all the points to be visited, these will be used as weights when making calculations for the path to be taken by the drones. The distance will be calculated using the latitude and longitude of the points and radius of the earth. We will basically be using the so called haversine formula and the great circle distance between the two locations.

7.3.5 Determine Shortest Path

This is a module that will use the travelling sales man algorithm to calculate the shortest path, which will travel back to the origin, between the points. The module will use the distance calculated using the haversine formula, as weights for the points to be visited.

7.3.6 Recalculate Shortest Path

This function will basically recalculate the shortest path, after considering the effect of restricted areas on the path to be taken by the drones. The module will calculate the distance covered by the restricted areas and add that to all the distances between points that would have had to pass through the restricted areas. The function will then calculate the shortest path with the readjusted distances using the travelling salesman algorithm.

7.3.7 Connect Drones

This will basically setup the wireless communication between the drones and the application we are developing for the MTDA.

7.3.8 Navigate drone

This is basically a module that is responsible for giving the drone all the commands necessary for it to travel to points that need to be visited. This module will call functions to take off the drones, land them and do movements along the x, y and z axis until they reach target locations and get back to their deployment point.

7.3.9 Collision Avoidance

This function will be responsible for ensuring that the drones do not collide into each other. This will be accomplished by having the drones relay their positions to one another. If a drone needs to pass through a location already occupied by another drone, that drone will just increase its altitude and thus avoid collisions.

7.3.10 Check drone status

This module is responsible for checking for drone failures and errors. If a drone stops sending data from it's on board sensors to the MTDA application, that drone will be regarged as dead and the task that it was doing will need to be reassigned to a functional drone.

7.3.11 Set up Transfer

This function will determine the appropriate distance the drone needs to be in order to start receiving data from the sensors at the target locations.

7.3.12 Transfer Data

This is a module that will retrieve data from the sensors, when the drones get in range of the sensors or when the drone lands on the sensors. The data is the written to a text file. Time and date taken must be attached to the data, in order improve differentiation during data display. The data with the attached labels will be written to a text file.

7.3.13 Task reallocation

This function basically takes a task that was left unfinished by a failed drone and assigns it to a functional drone. Therefore, this the function needs to track task execution process and mark tasks as completed or not. Incomplete tasks will need to get assigned to other drones. Task completion status can be represented by a Boolean, true or false.

7.3.14 Task Division

This function is responsible for telling the drones which location to visit. The function will pick a drone and send it to the first location to be visited as determined by the shortest path calculation. The rest of the locations will then be marked as unvisited. The next available drone will then be sent to next location to be visited in accordance with the shortest path calculation. Once a drone is finished with the task it was executing, the drone will be sent to the next location to be visited in accordance with the shortest path calculation. This continues until all locations have been visited or all drones fail.

7.3.15 Process Data

This function basically retrieves the text file containing data collected from sensors and sorts the data according to user requirements. This function also allows user to retrieve things such as max, min, day results and time based results.

7.3.16 Display Data

This function simply displays the collected data.

Chapter 8

Code Documentation

8.1 Introduction

The previous chapter discuses the low level design of the system and gives a basis on how the code should be structured. This chapter illustrates snippets of code and displays how it was documented. The language of choice was python, the editor gedit and the operating system used was Ubuntu 14.04 (LTS).

8.2 Travelling salesman pseudo code

```
1. T = 1, ..., n-1 //vertices
2. W = some large value
3. For x in T
4.    V = vertices in T
5.    U = x
6.    while V not empty
7.        u = most recently added vertex to U
8.        Find vertex v in V closest to u
9.        Add v to U and remove v from V.
10.        endwhile
11.    if total weight of U is less than W
12.        P = U
13.        W = total weight of U
14.Print P, which is the shortest path
```

8.3 Function documentation

MultiDrone Task Allocation system consist of multiple functions as illustrated in Figure 6 which represents some of the functions with code documentation. The comments describe each function and it's parameters.

Figure 8.1: Function Documenation

```
# getBearing(Lat1, Long1, Lat2, Long2):
# This method is responsible for getting the bearing between the drone's current lo
# and the target location. The parameters are as follows,
# lat1 - is the latitude of the of the drone's current location or first point
# long1 - is the longitude of the of the drone's current location or first point
# lat2 - is the latitude of the location the drone has to visit next or second poi
# long2 - is the longitude of the location the drone has to visit next or second p
# The coordinates are in degree format and will be converted to radians by this me
def getBearing(Lat1, Long1, Lat2, Long2):

# tsp(coordinates):
# This is the method that will calculate the order in which locations must be visit
# This is basically responsible for optimal path finding.
# The parameter for this method is,
# coordinates - this is an array with all gps coordinates
def tsp(coordinates):

# execTask(drone,lat,lon,angle):
# The method is responsible for moving a single drone to a specific location in or
# to collect sensor data. The method has got the following parameters,
# drone - is a number that specifies which drone is to be sent to collect the data
# lat   - is the latitude of the location to be visited in degrees
# lon   - is the longitude of the location to be visited in degrees
# angle - refers to bearing between the drones current location and target locatio
def execTask(drone,lat,lon,angle):

# checkDroneStatus():
# This method is responsible for checking is the drones are still in a condition to
# or that the drones are still doing what we expect them to do i.e are there flyin
# we gave themthe fly instruction.
# The parameter for this method is status where,
# status - is a string that reflects what we expect the drone to be doing, it can
#        flying, landed, hover, etc.
def checkDroneStatus():

# getCoordinates():
# This is the method responsible for getting the gps coordinates of the current lo
# of the drone. The cordinates are taken from a waspmote, since the parrot 2.0 doe
# have its own built in gps unit.The gps coordinates are written to file for the p
# of path tracking. The coordinates are stored in the form of <latitude,longitude>
# There are no parameters for this method
def getCoordinates():

# distance_on_unit_sphere(lat1, long1, lat2, long2):
# The parameters of this method are lat1, long1, lat2 and long2 where
# lat1 - is the latitude of the of the drone's current location or first point
# long1 - is the longitude of the of the drone's current location or first point
# lat2 - is the latitude of the location the drone has to visit next or second poi
# long2 - is the longitude of the location the drone has to visit next or second p
# The coordinates are in degree format and will be converted to radians by this me
def distance_on_unit_sphere(lat1, long1, lat2, long2):
```

8.4 Collision Avoidance Pseudo-code

8.4.1 Predetermined collission avoidance

```
1. let A and B be two drones we do not want to collide

2. let A fly at altitude Alt1

3. let B fly at altitude Alt2, which is 2m higher than Alt1

4. Keep both drones at their respective altitudes
```

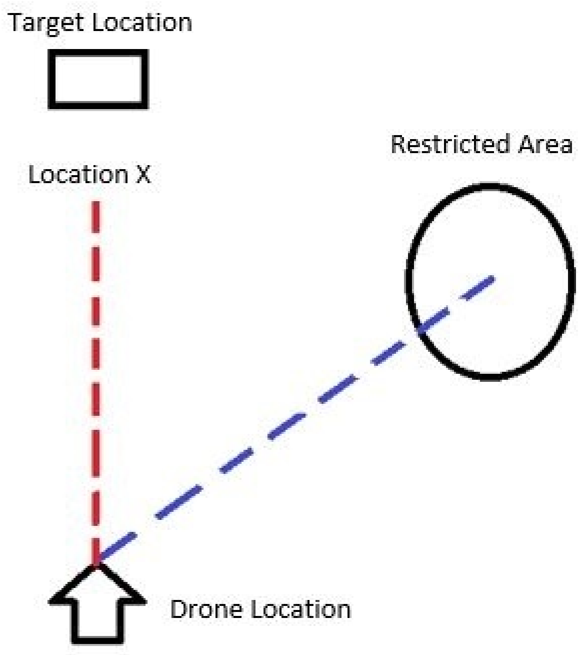
8.4.2 Real time collision avoidance

```
1. let A and B be two drones we do not want to collide

2. let posA be the current location of drone A

3. let posB be the current location of drone B

4. check if drones are too close to each other and may collide:

5.      if(distance between posA and posB <= 2m):

6.           drones may collide

7.           elavate one of the drones to a higher altitude

8. repeat 4-6 through out the flight of the two drones
```

8.5 Restricted Area Avoidance

```
1.    r = radius of the restricted area
2.    angle_T = calculate angle drone needs to rotate to get to target location
3.    distance_R = calculate shortest distance from drone's current location to
```

Figure 8.2: Restricted Area Avoidance



```
        the center of the restricted area using harvesine formula
4.    locationX = calculate GPS coordinates of the point distance_R away from
        drones current location at angle_T using harvesine formula
5.    distance_XR = calculate distance between locationX and the center of the
        restricted area
6.    if(distance_XR <= r):
7.        The path that the drone needs to traverse in order to get to the target
            location passes through the restricted area
8.        Implement restricted area avoidance:
09.           midPoint = coordinate of point 2m from total restricted area
10.           move drone from its current location to midPoint
11.           move drone from midPoint to target location
12.   else:
13.       visit target location
```

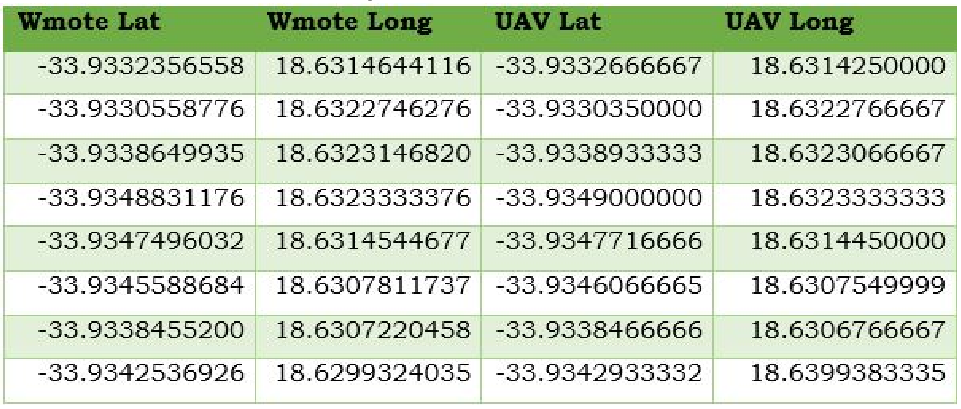
Chapter 9

Performance Testing And Reporting

9.1 Drone GPS Vs Waspmote GPS

For calibration and precision purposes, an experiment was also conducted to compare the GPS coordinates obtained with the waspmote's GPS sensor and the UAV's GPS sensor. Figure 13 presents the compilation of the data we collected at 8 different locations. Note that, the collected and compiled data in Figure 13 below, is rounded to 10 decimals, if the eleventh digit is greater than or equal to 5, we add one to the tenth digit and drop the rest of digits. It the eleventh digit is less than 5, we keep the tenth digit as is and drop the rest.

Figure 9.1: GPS Comparison



Wmote Lat	Wmote Long	UAV Lat	UAV Long
-33.9332356558	18.6314644116	-33.9332666667	18.6314250000
-33.9330558776	18.6322746276	-33.9330350000	18.6322766667
-33.9338649935	18.6323146820	-33.9338933333	18.6323066667
-33.9348831176	18.6323333376	-33.9349000000	18.6323333333
-33.9347496032	18.6314544677	-33.9347716666	18.6314450000
-33.9345588684	18.6307811737	-33.9346066665	18.6307549999
-33.9338455200	18.6307220458	-33.9338466666	18.6306766667
-33.9342536926	18.6299324035	-33.9342933332	18.6399383335

9.2 Data mule scenario testing

Table 9.1: Data mule experiment result

Capacity(gigabyte)	Upload Time	Download Time
1	4 minutes and 36 seconds	3 minutes and 27 seconds
2	9 minutes and 02 seconds	6 minutes and 33 seconds
3	13 minutes and 58 seconds	12 minutes and 27 seconds
4	17 minutes and 14 seconds	13 minutes and 52 seconds
5	22 minutes and 23 seconds	16 minutes and 43 seconds
6	26 minutes and 38 seconds	17 minutes and 28 seconds

This is basically a test designed to test the throughput of bebop drones so as to test their field readiness in terms of networking capabilities. We want to investigate whether the drones can be used to carry data in remote locations or even be used to provide internet.

9.2.1 Factors affecting upload and download speed

- Computer:
  The computer and the drone can be a bottleneck. The times above can be changed based on the quality of the computer you are using, if the computer is old the time taken can be more and if the computer is new it will probably be less. The hardware is also affecter with the speed in which it downloads and uploads.
- Distance:
  Upload and download speed is also affected by the distance between the computer and the drone. The greater the distance the slower the speed of the data transfer.
- Protocol:
  The protocol that is used for uploading and downloading a file can also have an impact on the speed of data transfer. We are using FTP which use the TCP/IP protocol for sending and receiving data.

9.3 Drone Camera Sensor vs Mobile Camera

The UAV comes with a number of sensor, and they are designed for multipurpose tasks. One of the sensor that the drone we used for this project has is a camera. Based on the specifications on the box, the camera is a high definition, and to test the resolution of the camera.

Figure 9.2: UAVcam

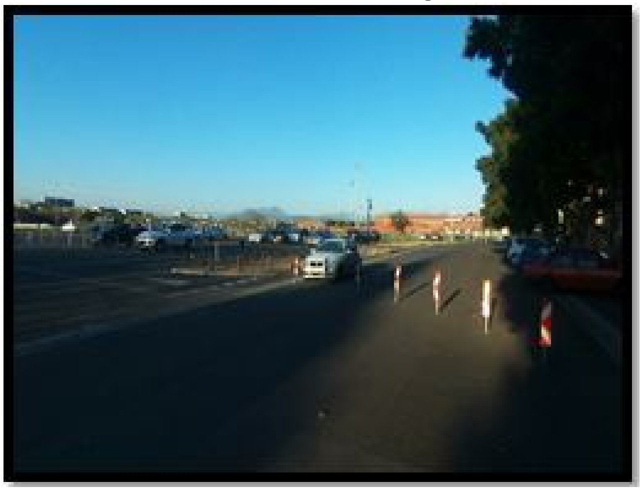
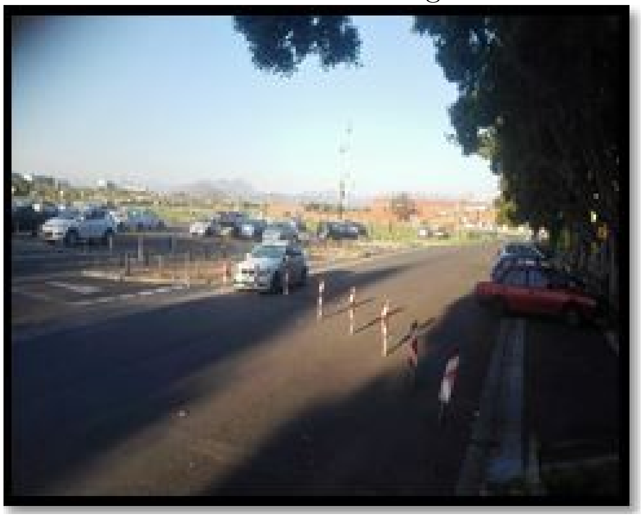


Figure 9.3: CellCam



9.4 Unit Testing

9.4.1 Testing drone forward movement

All drone movements are done by rotating the drone to face the direction we want to go to or simply face the direction of the location we want to visit. Hence, we shall be utilizing a lot of foward movement when the drone is visiting locations in the data collection scenarion and thus we deemed it necesary to test how accurate the drone is when executing forward movement. The goal of the tests here is to determine the error in terms of distance travelled forwards in our attempt to reach target locations. We want to determine how close or further away from the target locations, the drone will get when we execute forward movement.

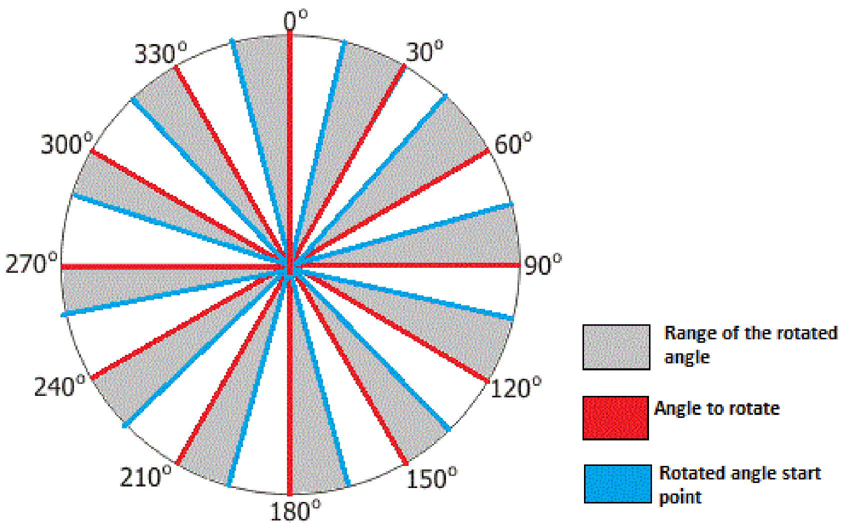
Table 9.2: Testing forward movement

Input distance (meters)	Actual distance travelled (meters)	Percentage Error
20	18.2	9
19	19.7	3.7
18	19.5	8.3
17	16.8	1.1
16	18.1	13.1
15	13.7	8.7
14	15.3	9.3
13	15.1	16.2
12	12.6	5.0
11	11.2	1.8
10	10.4	4.0
9	11.3	25.6
8	10.1	26.3
7	7.2	2.9
6	8.7	45.0
5	7.1	42.0
4	6.3	57.5
3	5.5	83.3
2	5.1	155
1	5.3	430

9.4.2 Testing drone rotation

All drone are dependent on rotating the drone into an orientation that is facing the location of the target point when the drone is visiting locations in the data collection scenarion. Thus we have deemed it necessary to determine the error in terms of the angle that the drone is actually rotating against the angle that the drone was supposed to rotate in order to face the direction of the target location.

Figure 9.4: Testing drone rotation



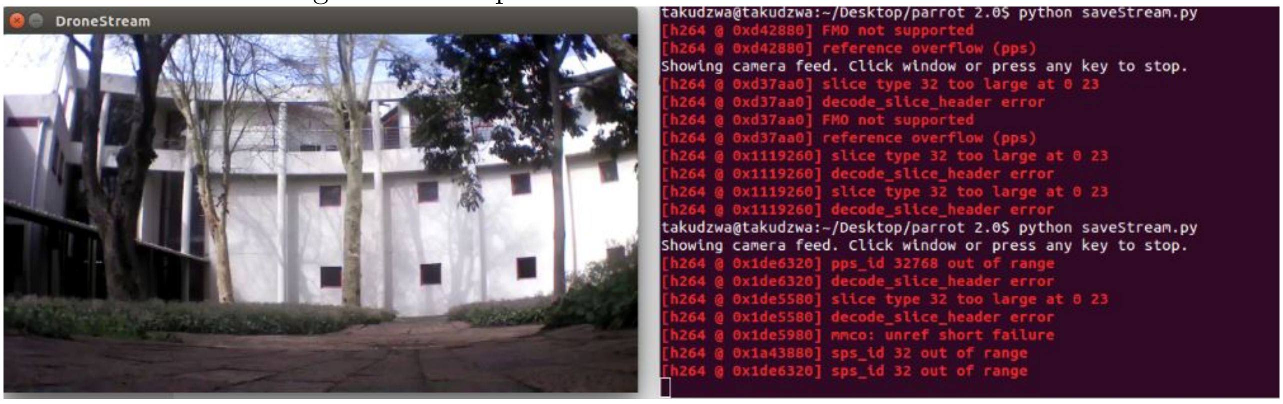
9.4.3 Test drone collision avoidance

The objective here is to test the module that is responsible for drone collision. The goal is to find the distance at which we can safely move the drones away from each other just before they collide.
The testing will be done by implementing predetermined collision avoidance. Here we will alevate the drones to different altitudes and then start flying them in the direction facing each other. We will measure the distance between the drones as they fly towards the direction of one another and then we move one of the drones to a higher altitude as soon as the distance between them becomes equal to or less than a set limit. The first time we run the experiment the limit will be five meters and we will make the limit smaller and smaller as the testing continues in decrements of halve a meter.

9.4.4 Video stream testing

The above figure shows a snap of the video stream module. The terminal with title DroneStream, on the left side, shows what the video stream from the drone would look like. The panel on the right side shows show the output from the stream module as video is recorded from the drone. The module makes use of openCV version 2 to get the video output from the drone.

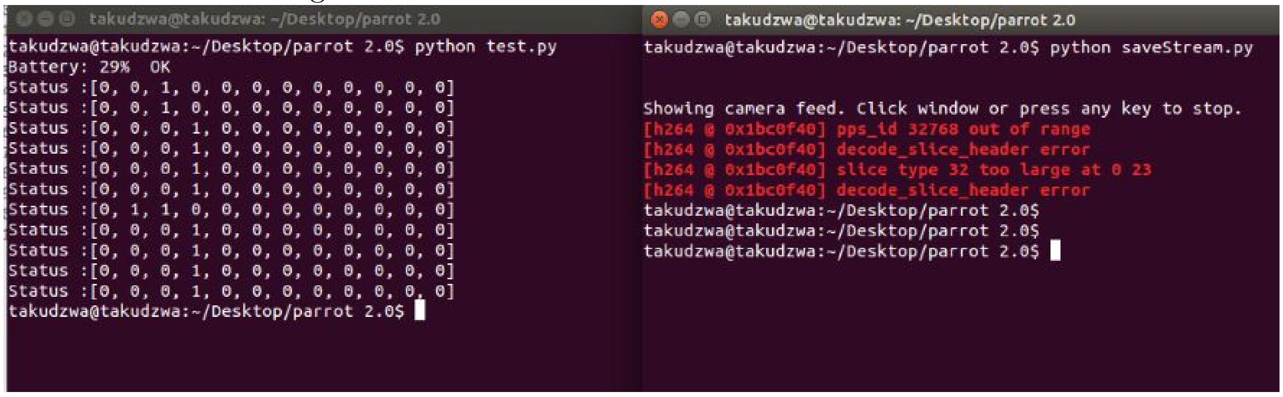
Figure 9.5: Snap of the video stream



```
takudzwa@takudzwa:~/Desktop/parrot 2.0$ python saveStream.py
[h264 @ 0xd42880] FMO not supported
[h264 @ 0xd42880] reference overflow (pps)
Showing camera feed. Click window or press any key to stop.
[h264 @ 0xd37aa0] slice type 32 too large at 0 23
[h264 @ 0xd37aa0] decode_slice_header error
[h264 @ 0xd37aa0] FMO not supported
[h264 @ 0xd37aa0] reference overflow (pps)
[h264 @ 0x1119260] slice type 32 too large at 0 23
[h264 @ 0x1119260] decode_slice_header error
[h264 @ 0x1119260] slice type 32 too large at 0 23
[h264 @ 0x1119260] decode_slice_header error
takudzwa@takudzwa:~/Desktop/parrot 2.0$ python saveStream.py
Showing camera feed. Click window or press any key to stop.
[h264 @ 0x1de6320] pps_id 32768 out of range
[h264 @ 0x1de6320] decode_slice_header error
[h264 @ 0x1de5580] slice type 32 too large at 0 23
[h264 @ 0x1de5580] decode_slice_header error
[h264 @ 0x1de5980] mmco: unref short failure
[h264 @ 0x1a43880] sps_id 32 out of range
[h264 @ 0x1de6320] sps_id 32 out of range
```

The above figure shows a snap of the autonamous control module and video

Figure 9.6: Drone Status and Stream


takudzwa@takudzwa: ~/Desktop/parrot 2.0
takudzwa@takudzwa: ~/Desktop/parrot 2.0

```
takudzwa@takudzwa:~/Desktop/parrot 2.0$ python test.py
Battery: 29%  OK
Status :[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Status :[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
takudzwa@takudzwa:~/Desktop/parrot 2.0$
```

```
takudzwa@takudzwa:~/Desktop/parrot 2.0$ python saveStream.py


Showing camera feed. Click window or press any key to stop.
[h264 @ 0x1bc0f40] pps_id 32768 out of range
[h264 @ 0x1bc0f40] decode_slice_header error
[h264 @ 0x1bc0f40] slice type 32 too large at 0 23
[h264 @ 0x1bc0f40] decode_slice_header error
takudzwa@takudzwa:~/Desktop/parrot 2.0$
takudzwa@takudzwa:~/Desktop/parrot 2.0$
takudzwa@takudzwa:~/Desktop/parrot 2.0$
```

stream module. The terminal on the left side shows output during navigation.The panel shows battery status at the start of the drone fight and later on we retrieve status codes of the drone which reflect whether drone is still flying, has landed, etc. The panel on the right side shows the output from the stream module as video is recorded from the drone.

9.5 Travelling salesman performance

The travelling salesman algorith is the core for our path planning. The algorithm will be responsible for mapping out the optimal path that the drone has to traverse during target search and visitation. Finding the optimal path is very important in order to minimize battery power consumption during target

search and visitation. Finding the optimal path will enable use to maximize the number of targets that can be visited by the drone based on the drone's autonomy. It is therefore important that our travelling salesman solution give out the best perfomance in terms of both the time taken to map out path and also finding the optimal path. As such, this section explores methods we took to test the performance of our travelling salesman algorithm.
The testing will be conducted by timing the execution of our greedy solution to the travelling salesman algorithm given varying inputs. Once we have the execution times of our solution, we compare it to the execution times of a dynamic solution to travelling salesman problem. We shall then test the performance of our task allocation algorithm, specifically the Hungarian method.

Table 9.3: Greedy travelling salesman performance

Number of inputs	Greedy Travelling Salesman
1000	0.98s
1500	2.113s
2000	4.319s
2500	7.044s
3000	10.340s
3500	14.307s
4000	19.015s
4500	26.016s
5000	33.375s

Table 9.4: Comparison of algorithm performance

Number of inputs	Dynamic Travelling salesman	Greedy Travelling Salesman
5	0.000132s	6.69956207275e-05s
10	0.015s	0.00012s
15	1.324s	0.00022s
20	1 min 36s	0.00036s
25	stopped	0.000481s
50	stopped	0.0020s
100	stopped	0.0080s

Table 9.5: Hungarian algorithm performance

Number of inputs	Hungarian method
5	0.00223s
10	0.0287s
15	0.153s
20	0.457s
25	1.229s
50	22.751s
100	8 min 31s

The inputs to the travelling salesman algorithms are the gps coordinates to points we want drones to visit. While the inputs to the hungarian method is the number of drones and we use the distance that they still need to cover to get to a location as weighting so that we can decide which drone will get the next task.
Dynamic travelling salesman had to be stopped due to the fact that the execution time was taking too long. We stopped the dynamic travelling sales execution when the running time was over eight minutes. We had to stop the execution because a runtime of over eight minutes is unacceptable when we consider that the drones only have an autonomy of eleven minutes. A run time of eight minutes would waste a lot of the drone power leading to low efficiency of drone task execution. Our greedy solution to the travelling salesman provided the best execution times in terms of speed and will thus lead to a higher efficiency of drone task execution.

9.6 Integration testing

9.6.1 Single Thread testing

The gooal here is to test the thread that is responsible for running all the functions that are require to execute the full task of collecting data from a single GPS coordinate. The testing will be conducting by having the thread call the GPS input function and check for errors. Then we will have the thread call the fuction to execute target search and visitation. We will then check for any errors and verify that the drone has visited the target location by extracting the drone's location at the end of execution of the target search and visitation function and comparing that location to the provided input GPS location.

9.6.2 Master thread testing

Our program is structured so that there is a thread for monitoring drones and two threads that will each allow a single drone to conduct single target search and visitation, then lastly there is a master thread whose main responsibility is to run the first three threads. The master thread is also responsible for task distribution between our drones. Thus the goal of master thread testing is to verify that it can do all it's given responsibilities without the program crashing. Testing will be done by making the master thread conduct task distribution and call the two threads responsible for single target search and visitation, also simultaneously call the drone monitoring thread. The end goal of the testing is to ensure all given coordinates have been visited and there were no errors. The deliverable at the end of the testing will be a path taken to visit input GPS cordinates and a log of errors if any.

Figure 9.7: Single drone path

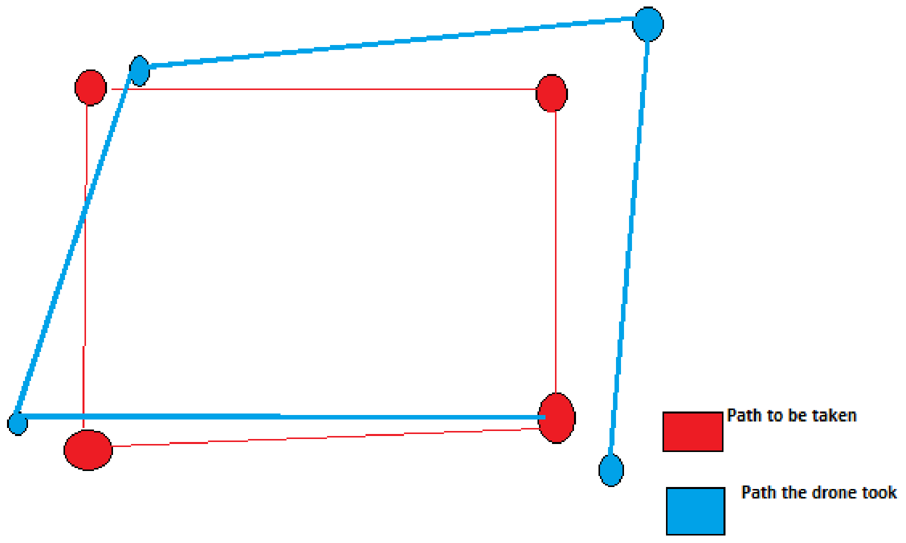
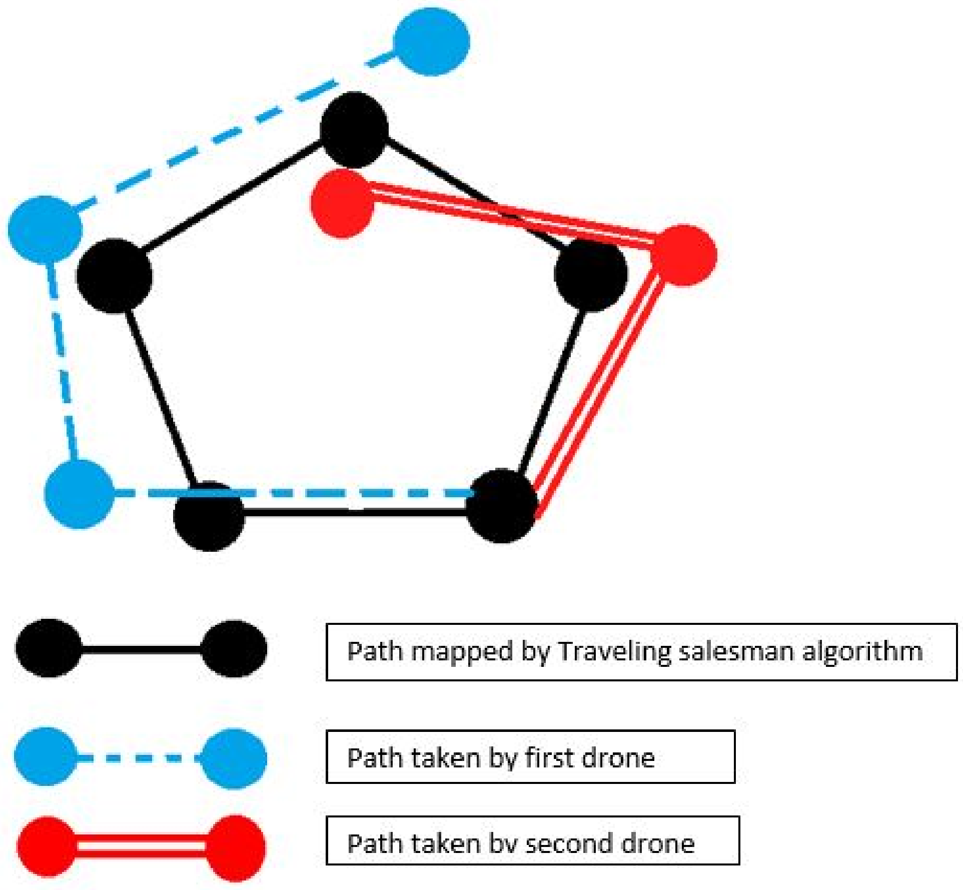


Figure 9.8: Multiple drone path


Path mapped by Traveling salesman algorithm
Path taken by first drone
Path taken bv second drone

9.7 Destructive and stress testing

9.7.1 Testing the range at which drone flies out of range

Here the goal is to determine the distance at which the drone flies out of range and record the behaviour that we get from our program. We want to determine if the program will crash or giving out other errors that can result in termination of program before completion of given tasks.
The testing will be conducted by attemping to fly the drone at distances that are hundred meters and above. We chose to start testing at a hundred meters because the quoted value for the range according to the parrot bebop manufacture specifications is two hundred and fifty meters and we assumed that the drone will be able to fly in ranges that are below forty percent of the qouted range by drone manufacturer. Our goal is then to orbserve drone and program behaviour as we test the drone range, by looking out for errors and sudden termination of the drone control program, as well as looking out for loosing control of the drone.

9.7.2 Testing drone power failure

Here the objective is to determine the level of battery power at which our drone dies out and loses all functionality. This is important in order to avoid drone power failure during task executing and also as a measure to prevent drone damage in the case where the drone dies out due to low battery amid flight and just crashes to the ground or worse. This test is critical for the drone monitoring module as we need to determine the cut of point in terms of the level of battery power left, where we will have to terminate task execution and land the drone so as to avoid power failure and damages to the drone.
Testing will be done by attempting to fly the drone when the level of battery power left is twenty percent and below. We start testing at twenty percent and if the drone can not be flown at that battery power will attempt to fly the drone at a higher power level in increaments of two percent. However, if the drone can be flown at that power we will attempt to fly the drone at a lower power level in decreaments of two percent.

Table 9.6: Testing drone power failure

Power level (percentage)	Drone Behaviour
20	responsive
18	responsive
16	responsive
14	responsive
12	responsive
10	responsive
8	responsive
6	responsive
4	unresponsive

9.7.3 Fault injection: GPS coordinate testing

Target search and visitation of locations in the data collection scenario depends on the program being supplied with GPS coordinates of the locations we need to visit as well as the coiordinates of the restricted areas, if any. It is therefore important that the GPS coordinates are correctly formated.
The format of GPS coordinates as required by our program is as follows:

- decimal degrees format
- no spaces
- floating points
- use the period instead of a decimal comma

Thus our testing methodology will attempt to introduce faults in the programming by giving wrongly formatted GPS coordinate inputs as follows

- providing coordinates in degrees, minutes, seconds format
- coordinates with spaces
- use a decimal comma instead of a period

Chapter 10

Conclusion

The objective was to develop a system that will allow autonomous control of multiple drones. The system had to include collision detection and avoidance, restricted area avoidance, efficient task allocation and drone path planning. We developed a program as a solution that fulfils the objectives that we outlined and now we will point out our orbservations and future work that can be done to improve the program we developed.

10.1 Observations

The main orbservation was the effect of wind on the drones. Almost all the actions that are performed by the drone in order to archieve movement were severely affected by wind. This can be seen from the results of the path taken by drones. There is a significant difference between the actual path mapped out by the travelling salesman algorithm and the path taken by the drones. Starting at the moment of drone take off we orbserve the drone experiencing a pushing force, moving the drone out of position if we were to draw an imaginery line from the initial point of take off. We also see other drone actions being affected by the wind in a simular fashion. Actions affected by the wind include drone rotation, hovering and general movements in any direction.
We also orbserved a delay in the reaction time of the drone when we flew it at altitudes higher than 50m. When we tried to perform actions such as rotating the drone, we would issue the command to perform the action and then only get a reaction from the drone a significant time after the command has been issued. The delay ranged from 4 to 5 seconds. This time is significant because if we were issuing a command to perform a critical action like trying to move away from a building to avoid crushing, by the time we would get a response from the drone it will be too late, the drone would have crushed.

10.2 Future work

- Object detection and avoidance
- Avoiding multiple restricted areas between current drone location and target location
- Autonomous indoor navigation
- Path planning in a changing environment
- Use machine learning to recognize important structures and objects in an environment

Chapter 11

User Guide

This chapter will feature a detailed user guide relating to the project. It will mainly focus on setting up the drones, the equipment need and it's setup process. Then we will describe the actual steps required to run the software we developed to solve the Multi-Drone task allocation problem.

11.1 Pre-requisites

- Two parrot bebop drones
- pc running ubuntu
- Two wifi adaptors on your pc
- Open CV version 2

11.2 Drone setup

11.2.1 Connecting drone to pc

The parrot bebop provides a wireless network access point and we can connect our computer to the drone's network by using the same steps as those we use to connect any wifi access point as describe in the steps below.

1. Make sure your wifi card is switched on
2. Click network icon, to view all available networks
3. Select the network: BebopDrone_xxxxx

4. Wait for pc to notify you that it is now connected to BebopDrone_xxxxx network

11.2.2 Change drone static IP address

Our program requires that we connect to multiple drones simultaneously but all parrot bebop drones come with the same static IP address 192.168.42.1, therefore we have to change the IP addresses of the drones we wish to use with our program so that each drone has a unique IP address. Outlined below are the steps to change a parrot bebop's static IP address.

1. switch the drone on and connect your pc to the drone access point
2. open a terminal
3. telnet 192.168.42.1
4. cd sbin
5. vi broadcom_setup.sh
6. change field IFACE_IP_AP="192.168.42.1" to IFACE_IP_AP="192.168.42.x", where x represents any number which you have not assigned to other drones
7. save the file and exit

11.2.3 Drone battery

Due to the low autonomy of drones, we recommend only using batteries that are fully charged. The drones have a fly time of 11 minutes after a full charge, using batteries that are not fully charged means lesser fly time. Using batteries that are not fully charged increases the risk of not finishing task execution due to low power.

11.3 Installing an extra wifi dapter

In order to connect our pc that will be running our program to two drone access points simultaneuosly we decided to add extra usb wireless adapter to our pc. The steps required are as follows:

1. install wireless adaptor drivers

2. switch off pc

3. insert wireless adaptor into usb port

4. switch on pc and check that you see an extra wireless adaptor in network setting

5. if you do not see the extra wirelsess adaptor, switch off pc and remove the wifi adaptor then repeat 1 - 4

11.4 Requirements before flying drones

11.4.1 Open space

The program we developed currently does not support detection of objects, this includes all static and non-static objects. Therefore, we highly recommend using the program to control drones in an area that is clear of objects like trees, cars, building and pedestrians. Due to the lack of object detection and avoidance measures in our program, if the you try to run the program in an area with static and non-static objects, you are risking crushing the drones into the objects as well as colliding with moving objects.

11.4.2 Outdoor operation

The program we developed uses GPS coordinates for navigation. Therefore, we require that the program be used in outdoor open spaces where the drones can connect to GPS satellites. It is very important that the outdoor area be far from buildings and other things that can block connection to gps satellites. It is highly recommended that you verify access to gps connection in the area where you will use the program.

11.5 Running our program

- Connect your pc to the networks of your drones
- Open a terminal
- Go to the folder with our source code
- Type ”python prog.py <data.txt”, where data.txt is the file with gps coordinates of the location you want the drones to visit

Bibliography

[1] R. Zlot, A. (tony Stentz, M. B. Dias, and S. Thayer, “Multi-robot exploration controlled by a market economy,” in *Multi-robot exploration controlled by a market economy*, 2002, pp. 3016–3023.

[2] X. Ding, A. Rahmani, and M. Egerstedt, “Optimal multi-uav convoy protection,” in *Robot Communication and Coordination, 2009. ROBOCOMM’09. Second International Conference on.* IEEE, 2009, pp. 1–6.

[3] N. Nigam and I. Kroo, “Persistent surveillance using multiple unmanned air vehicles,” in *Aerospace Conference, 2008 IEEE.* IEEE, 2008, pp. 1–14.

[4] C. T. Cunningham and R. S. Roberts, “An adaptive path planning algorithm for cooperating unmanned air vehicles,” in *ICRA*, 2001, pp. 3981–3986.

[5] T. Lemaire, R. Alami, and S. Lacroix, “A distributed tasks allocation scheme in multi-uav context,” in *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*, vol. 4. IEEE, 2004, pp. 3622–3627.

[6] W. Zhu and S. Choi, “An auction-based approach with closed-loop bid adjustment to dynamic task allocation in robot teams,” in *Proceedings of the world congress on engineering*, vol. 2. IAENG., 2011, pp. 1061–1066.

[7] M. M. Lehata, “Autonomous drone flight,” Published as partial fulfilment of the requirements for the degree of Baccalaureus Scientae (Honours) Computer Science University of the Western Cape, 11 2013.

[8] P. R. Chandler, M. Pachter, D. Swaroop, J. M. Fowler, J. K. Howlett, S. Rasmussen, C. Schumacher, and K. Nygard, “Complexity in uav cooperative control,” in *American Control Conference, 2002. Proceedings of the 2002*, vol. 3. IEEE, 2002, pp. 1831–1836.

[9] Y. Jin, A. Minai, M. M. Polycarpou *et al.*, "Cooperative real-time search and task allocation in uav teams," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1. IEEE, 2003, pp. 7–12.

[10] A. Richards, J. Bellingham, M. Tillerson, and J. How, "Coordination and control of multiple uavs," in *AIAA guidance, navigation, and control conference, Monterey, CA*, 2002.

[11] P. Sujit, A. Sinha, and D. Ghose, "Multiple uav task allocation using negotiation," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems.* ACM, 2006, pp. 471–478.

[12] R. Henriques, F. Bação, and V. Lobo, "Uav path planning based on event density detection," in *Advanced Geographic Information Systems & Web Services, 2009. GEOWS'09. International Conference on.* IEEE, 2009, pp. 112–116.

[13] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *AAAI/IAAI*, 2000, pp. 852–858.

[14] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, "Multi-task allocation and path planning for cooperating uavs," in *Cooperative Control: Models, Applications and Algorithms.* Springer, 2003, pp. 23–41.

[15] G. Chen and J. B. Cruz Jr, "Genetic algorithm for task allocation in uav cooperative control," in *Proceedings, AIAA Guidance, Navigation, and Control Conference, Austin, Texas*, 2003.