CS407 GROUP REPORT

# OCTODRONE: SIMULATION AND DEPLOYMENT FOR AUTONOMOUS DRONE NETWORKS

*Authors:*

Alex HENSON,

Ben DE IVEY,

Jonathan GIBSON,

William SEYMOUR

*Supervisor:*

Dr. Arshad JHUMKA

*Secondary Marker:*

Dr. Andrzej MURAWSKI

Department of Computer Science

University of Warwick

Summer 2016

# Contents

# List of Figures

# List of Tables

## Abstract

Drones and quadcopters are becoming increasingly prevalent in modern society, leading to increased research and development in the field of mobile sensor networks. This project, octoDrone, aimed to create a high quality network simulator targeted at quadcopters, with the aim of being accessible to academics and students alike. Through a focus on being abstract, we have created an application which is powerful, expressive, and can be deployed to almost all currently available hardware. This ability to use the same code for simulations and deployments allows for accurate benchmarking and verification of drone programs.

CHAPTER 1

# Opening

*"Once you have tasted flight, you will forever walk the earth with your eyes turned skyward, for there you have been, and there you will always long to return"*

*- Leonardo da Vinci*

## 1.1 Key Words

Autonomous Drones, Sensor Networks, Network Simulation, Communications Routing, Quadcopters.

## 1.2 License

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. This means that you are free to:

- **Share** - Copy and redistribute the material in any medium or format.
- **Adapt** - Remix, transform, and build upon the material.

for any purpose (even commercially), under the following terms:

## 1.3 Acknowledgements

We would like to thank a few people in particular for helping us through the vast amount of research and development that was undertaken as part of this project. Our project supervisor Arshad Jhumka has helped guide us, and the advice he gave us when we were considering changing the direction of the project was invaluable. In addition, campus security were very understanding when asked to retrieve one of the drones used for testing from the roof of the maths building, and spared us what could have been a lot of embarrassment and trouble. On a lighter note, we would also like to thank USB Man and Will's rubber duck, who both made development immeasurably smoother.

## 1.4 Introduction

Drones have exploded into the public conciousness in recent years, for reasons both good and bad. With companies such as Amazon using drones for delivery, and many studios using them for filming, it is clear that they will become and remain a part of everyday life. Research into networks of drones is still young, and requires a specific set of tools. This project aims to create a new application for simulating drone networks to aid both academic progress and outreach. With a focus on adaptability, it is hoped that octoDrone can be used as a domain specific testing tool which is generic enough to be deployable to most types of consumer and commercial hardware available.

This report will provide a comprehensive analysis of the project undertaken by our group. There will be a background summary of the key components in this field, as well as a discussion of the ongoing research, development, and production being carried out. An analysis of the potential problems for which a solution can be found in drone networks will be supplied, and justification given for the resulting aims and objectives of our group. The report will detail the design, implementation, and testing of the solution, including considerations for the management of the project. Finally, the project outcome will be evaluated, followed by a conclusion reflecting on the success of the project and considerations for future works.

CHAPTER 2

# Background

*This section will introduce the components which will be researched into that form the basis of the project. The aim of this section is to provide the reader with definitions for keywords which will appear on numerous occasions throughout the project, as well as helping to lead into a definition of the problem space and the objectives of the project as a result.*

## 2.1 Drones

### 2.1.1 Definition

Unmanned Aerial Vehicles (UAVs), also known as drones, are aircraft which are either 'piloted', or perform autonomously using pre-programmed flight paths and objectives [9]. Consumer-level drones are typically small in size, and take the form of quadcopters, which are multi-rotor helicopters with four rotors. These types of drones are very lightweight, and are powered by batteries; power consumption is almost completely attributed to fight time, although a more advanced drone will be required to exhaust power on its additional parts. We will be focusing on the use of these drones for the scale of this project.

### 2.1.2 Sensor Capabilities

Drones are typically equipped with cameras, as well as additional sensors, which vary depending on the type of drone, or its purpose. In the case of military drones, sensors such as multi-spectral targeting systems, night vision, infrared imaging and GPS are an absolute must [15]. However, mounted weaponry may also be included for direct warfare, unless the drone is designed specifically for intelligence, surveillance or reconnaissance, as power consumption is a primary concern, and must be limited. Military drones are controlled via satellite from a military base, although drones may also be controlled by Wi-Fi, radio or remote. Consumer-level drones have a wide range of usages, and the sensors that they require to accommodate these tasks are largely dependent on the price. It is possible to attach a large multitude of different sensors to drones; however the primary function of an aerial (consumer-level) drone is to collect high-quality imagery, often beyond the level of detail that the human eye can process. These types of sensors include stereoscopic, thermal imaging, near-infrared and infrared, but other sensors such as thermal sensors and proximity sensors may also be used [54].

### 2.1.3 Usages

Consumer-level drones have become increasingly popular in the past few years as they have become more affordable, capable, and reliable to use. Due to their ability to capture high-quality imagery from impossible-to-reach locations, drones are incredibly useful in areas such as real estate, to take aerial shots of properties, or as a cheap alternative to conventional helicopters for capturing news, such as high speed chases [12]. Drones can also be employed for services such as delivery; there have been several initiatives for drone-based delivery of food or packaged goods by famous companies such as Amazon and Domino's Pizza [13]. Another possible usage for drones is in emergency services, such as the detection of forest fires, or search and rescue. It is these areas of drone research and development which can be considered to display the strength and importance of drones, as they are able to perform dangerous tasks which humans are incapable of and/or with no physical risk to human beings themselves. Compared to other types of mobile sensing, drones offer direct control over where to sample the environment, such that they can be explicitly told where to move to.

## 2.2 Sensor Networks

### 2.2.1 Definition

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to monitor physical or environmental conditions. These devices are referred to as nodes, which are able to communicate with each other and optionally with a base node, commonly referred to as a gateway, which provides connectivity between itself, the nodes, and the rest of the wired world [20]. Nodes may vary in size and number depending on the network, but will typically contain transceivers, a battery, an electronic circuit for interfacing with sensors, and an energy source. The ability to cooperatively pass sensor data to a main location has implications in many different industries, as well as military applications.

### 2.2.2 Drone Networks

In the context of drones, this refers to a wirelessly connected network of autonomous drones (possibly with a base station), which can share information or commands with each other, as well as facilitate communication between them remote systems. Given that autonomous drones are emerging as a powerful new breed of mobile sensing system which can carry rich sensor payloads with various methods of control, a collaborative network of drones has considerable potential, and can greatly extend the capabilities of traditional sensing systems [27].

## 2.3 Network Simulation

### 2.3.1 Definition

As the name suggests, network simulation is a technique for modelling the behaviour of a network without performing a real deployment, in order to test the effectiveness of the network and assess how the network will behave under different conditions. Therefore, a simulation refers to software that predicts the behaviour of a network so that performance can be analysed. By emulating an existing network, unexpected problems can be addressed or prevented prior to the deployment of the network.

### 2.3.2 Structure and Testing

A network simulation typically produces output in text form, as well as a visual representation in a GUI. This can be useful to see how nodes interact, how data is sent, and where connections go out of range or experience interference: visualisations make it is possible to study the actual performance of a network and its protocols against the conceptual design. The simulation must be careful to provide an adequate level of detail to test the network without affecting its running time [6].

## 2.4 Routing

### 2.4.1 Physical Routing

In order for mobile sensor networks to gather data about the environment, they will be required to navigate freely using predetermined pathfinding algorithms. For a drone network, a drone will be required to navigate 3-D airspace and collect sensing information, whilst being careful to maintain an efficient route and avoid problems such as collision with its neighbours or the limitations of its physical components (such as battery life). Pathfinding must take into account the possibility of nonlinear dynamics, various constraints, and changing environments [44].

### 2.4.2 Communications Routing

One of the core aspects of a sensor network is the ability for nodes to communicate with each other, relaying data back to the gateway. For an optimal routing algorithm, the exchange of data must be robust, avoiding congestion and maintaining connectivity when faced with mobility, whilst trying to maximise the duration for which the sensing task can be performed [10]. The properties of communications routing which are to be optimised are dependent upon the type of sensor network; a drone network where battery life is a key constraint must likely be optimised for energy consumption.

CHAPTER 3

# Specification

*In this section of the report, there will be an introduction to the problem space which is solvable using consumer-level drone networks (as defined in the previous section). After describing the problem, the objectives which need to be achieved in order to provide a solution for the problem space will be clearly laid out as a means of outlining the foundation of the project. Justification for why the project solution to the aforementioned problem is both necessary and valid will also be given. There will be an analysis of the stakeholders in the project, followed by a feasibility study where a brief discussion of the scope of the problem which is to be handled will be provided, including the level of depth with which drone networks will be explored and implemented throughout the project.*

*This study also allows us to identify the possible problems which may arise throughout the project and analyse its economic implications, as well as give a brief introduction to the management of the project. In doing so, it can be shown that the project is actually feasible to complete with the time and resources available. Having defined the objectives which must be completed to provide a solution for the project, the subsequent functional and non-functional requirements must be identified to ensure that the project deliverables are measurable and well-defined. Finally, any changes from the original specification at the beginning of the project will be briefly discussed, with justification for these changes.*

## 3.1 Description of the Problem

The problem, in the context of this project, is that there are many situations in which a drone sensor network could be well implemented to effectively carry out a task, but there is a lack of an environment for testing before going through the costly process of deployment. Simulators exist for the task of modelling sensor networks, but a generic network simulator does not exist which can be readily adapted to various hardware for use in multiple projects. For this project, the group has decided to focus on the task of creating a simulator which can be used for both modelling a drone sensor network through virtual stimulations, and as a framework for the code which implements physical deployment on a real network. An arbitrary problem for user tasking, to be used as a demonstration, can be defined later. Ideal candidates are those which are impossible or risky for humans, such as search and rescue or high altitude photography, as well as problems which are feasible for humans, but can be more readily solved by UAVs, such as automated delivery.

For this project, the objective that the group decided to focus on was the creation of a stimulator which would be built from first principles using the C++ programming language. Research into pre-existing network simulator libraries was performed in the literature review in section 4.2.2. Considerations for the use of pre-existing network simulator libraries will be discussed in chapter 6.1, however it is sufficient to say that they were found to be difficult to tailor to the specific project requirements and the decision was made to focus on a project-specific solution of the group's own design. The implementation of a simulator for a drone sensor network can hypothetically be applied to any similar problem area by altering the hardware arbitrarily (such as drone sensory inputs and outputs), which allows a solution to be provided for the general use case, and applied to various other situations.

The solution to this problem can be split into four main areas: the network simulation, the physical routing, the communications routing, and the physical deployment. In order to successfully create a solution to the problem described above, it is necessary to construct a stimulator which will accurately model the performance of the drone network, with associated physical and communications routing algorithms for optimal performance, before finally transferring this model to the physical drones and deploying them. The general solution can then be tested in a real situation, and then possibly extended to handle specific problem-solving tasks with user input, such as the detection of forest fires.

## 3.2 Objectives

The aim of this project, then, is to design and implement a general-use simulator for a drone sensor network in order to provide a generic solution to the problem of modelling sensor networks, and the establishment of a framework for implementing physical deployment. This solution will provide the basis for research and extension into any potential situation in the field of drone sensor networks. The major components of the project were outlined in the initial project specification during the early stages of the project life cycle, and can be summarised as follows:

1. **Implement a network simulator**. Either using a pre-existing set of libraries, or by creating our own, which are specifically tailored to our domain.

2. **Create a set of communication implementations**. Nodes must be able to communicate with each other easily using industry standard algorithms and protocols.
3. **Provide a reference implementation**. To demonstrate the correctness of the simulator, and also to act as a starting point for future development, a relevant sample application should be created.
4. **Allow the simulator to be adapted for use in physical hardware**. In order to allow for the same programs to be used for simulation and deployment, simulated programs should be able to run on real drones.
5. **Provide a reference deployment**. To show how simulations can be run on physical hardware, a case study should be presented where the simulator runs on real drones.
6. **Ensure that the product is accessible**. The simulator should be easily usable to those most likely to benefit from it.

These tasks have been scheduled using a Gantt chart which can be seen in 10.1, where tasks were assigned to each group member based on their individual skills (as necessary). Certain tasks, such as communication and demonstration development, could be developed in parallel. Delegation of tasks and group roles will be discussed in depth in chapter 10.

## 3.3 Justification

The benefits of carrying out a project involved in creating an efficient, extensible solution to network simulators are relatively self-explanatory. The project is justifiable in its ability to produce a framework for extensive research and development in the field of sensor networks, with potentially life-saving results. Drone networks themselves are an area of research which is growing in popularity, so the project interacts well with the state-of-the-art and could have considerable impact on future developments. A general use solution based on the project software would be easy to use and deploy, and incredibly extensible.

## 3.4 Stakeholder Analysis

As previously discussed, the implementation of a drone sensor network has implications for a wide variety of industries. This section will therefore give a formal outline of the prospective stakeholders in the project and the justification for them. Firstly, given that drone sensor networks are a relatively new field of research, those parties interested in research and development of sensor networks would also benefit from the project. The results may extend the field of research, and the project solution could be adapted for use in other areas. Example applications for indirect stakeholders could include detection and response to forest fires, emergency services such as firefighters, ambulance services, and search and rescue, who would benefit greatly by providing the ability to pre-empt danger and respond to crises much more rapidly, as well as reducing the risk to human lives in combating fires. In the same way, commercial businesses such as real estate and delivery services in pursuit of more efficient business may also benefit from the project. Finally, the project can be used for academic outreach by teaching the concepts surrounding simulators and drone networks, as well as the algorithms which have been implemented in the code.

## 3.5 Feasibility Study

While the merits of the project are justifiable, it should also be noted that the project implementation carries a considerable technical difficulty. Individual consumer-level drones must be adapted to be programmable, autonomous drones which function as a sensor network capable of communications and pathfinding. These will then have to be simulated and physically deployed. Therefore, it is important to analyse the feasibility of the project given time, hardware, and other constraints, which will be discussed in the following sections.

### 3.5.1 Problem Scope

It is important to consider the scope of the problem, with regards to how far the solution can be extended into the domain of, in this case, network simulators. Given that drone sensor networks are a relatively new domain for research and development, there is no commonly used and accepted standard for drone networks in the context of network simulation. In other words, the solution to the problem is not an extension of a previously existing solution, or of a set of rules governing how the problem can and should be solved using drone sensor networks.

As a result, the scope must be carefully defined such that an effective solution can be reached for the problem without expanding too far into the problem domain. By implementing a drone sensor network which can be adopted into any general use case and can be easily extended to take any required sensory information and applied to solve a problem, the project avoids becoming overextended. At the same time, in order to show that the network can effectively handle user input tasking, one piece of sensor information was focused on, to demonstrate an accurate model for problem detection and response (as defined by the user) through well-established communications.

### 3.5.2 Project Scope

Having defined the scope of the problem, it is also necessary to examine the scope of the project itself in terms of how far the project can extend into the domain of drone sensor networks. The implementation of any sensor network requires a lot of concise testing and a solid formation of protocols. There are many areas to consider with physical routing and communications, as well as physical deployment and use tasking. When creating the network simulator, it can be tailored to project-specific requirements in order to minimise the amount of considerations for network protocols and possible problems (discussed later), whilst accurately modelling physical deployment.

Considering the time constraints that are imposed on the project, it will also be necessary to adapt pre-existing algorithms to establish our network communications and physical routing as opposed to creating our own algorithm, which would be suboptimal. In terms of physical deployment, the project is limited to two drones, so it is only possible to implement a minimum working example of a network to test the solution. Nonetheless, it is possible to show that the network is theoretically scalable and accurately demonstrates the ability for drones to communicate and use pathfinding effectively.

### 3.5.3 Financial Analysis

The basic requirements for a physical drone network are the drones themselves, the sensors which will be attached to the drones, as well as equipment for programming the drones and communications. All of these are provided by the Department of Computer Science, so financial constraints for physical deployment are limited to (a budget of) what the Department is able to provide. While it is possible to purchase more drones, the same result can be achieved, provided that at least two drones are available. Additionally, the project will not include the use of any bespoke software or libraries in the development of the simulator or communications; they will be open source, so there are no costs incurred in the development of the project. This also ties in with the license the project uses, and means that the entirety of the project can be released as free and open source software.

### 3.5.4 Market Analysis

For the project to be used by third parties, there must be licensing associated with the final product. As the project is intended to be open source and not commercialised, the project software will be using the GNU General Public License v3.0 (or later), which stipulates that our project is open source, with the freedom to use or change the software, as well as distribute it and share changes. However, in the event that the project is used, the software creators are not accountable for any problems with the project software. As the project is not commercial, with no intention to monetise it, the use of this license allows us to appease any and all stakeholders whilst protecting ourselves from potential threats. The license also dictates that modified versions of the project must be released under the same license, thus ensuring that octoDrone remains free for everyone.

### 3.5.5 Project Management

There are many challenges associated with undertaking a project such as this, particularly with regards to administration, scheduling, and the division of tasks. Therefore, it is crucial that the project is managed efficiently, and that progress is carefully monitored and assessed throughout the life-cycle of the project. The issues associated with project management and how they were handled throughout the project will be discussed later on in chapter 10. However, there are several challenges that the project group will be presented with in the context of project management, which are summarised below:

- **Development methodology**. There must be consideration for a development methodology which accommodates the size of the group and the time each group member has available, as well as how to track the progress of the project.

- **Deadlines and scheduling**. Meetings and deadlines must be scheduled such that the group is fully aware of what stage of the project must be completed and when. Interested parties such as the project supervisor must also be regularly informed of the progress of the project to allow for appropriate advisory actions where necessary.

- **Group roles**. Each member of the group must have a clear and distinct role in the group, with one member taking the role of project manager, dedicated to handling each group member and their associated tasks, as well as managing the state of the project to ensure that it is completed effectively and within constraints.

- **Managing project materials**. It is necessary to manage the project material to ensure consistency amongst each members' work, as well as maintaining previous iterations of code to avoid problems such as losing previous functionality.

## 3.6 Requirements Identification

This section will contain a list of the functional and non-functional requirements identified at the beginning of the project which will be necessary to implement to ensure that the project software adequately represents a solution to the problem as defined in this chapter. A functional requirement refers to quantitative requirements which can be easily measured and tested to ensure correct functionality. Non-functional requirements are those which cannot be measured, as they are subjective, but are nonetheless important for a successful project outcome.

### 3.6.1 Functional Requirements

A list of the functional requirements for the project can be found in table 3.1.

### 3.6.2 Non-functional Requirements

A list of non-functional requirements can be found in table 3.2.

## 3.7 Project Deliverables

Throughout the course of the project, there were several deadlines for which the project will be required to deliver a product. The project specification was submitted in Term 1, which introduces the project and its initial planning and methodology (appended to the report for reference). At the end of Term 1, a poster presentation was given by the project group to select supervisors, invigilators, and interested students. A live demo of the project software and summary of this report was delivered at the beginning of Term 3. The demo displayed the proposed solution to the problem defined at the start of the project, and showcased the features of the working solution. Alongside this report, the code for the project, including the simulation, communications routing, physical routing and deployment code will also be submitted.

## 3.8 Changes From Original Specification

In the original specification it was stated that ns-3, a set of network simulator libraries, would serve as the core of the project simulation efforts, and that the code for physical deployment would then be built off of this as a result. However, it was decided to instead build a simulator from scratch,

Table 3.1: List of functional requirements for the project software

| # | Description |
|---|---|
| R1 | Users must be able to run an executable which contains the simulation environment and communications module |
| R2 | A class must be instantiated for the nodes of the simulator (i.e. drones and base station) and a communications module must be installed on each of them |
| R3 | The simulation must run, such that each node begins to operate and 'travel' within the environment |
| R4 | Each simulated node must be capable of sending, listening for, and receiving messages through the environment |
| R5 | The environment must incorporate and handle multithreading for each node |
| R6 | The simulation must be able to accept interference functions for message passing |
| R7 | The user must be able to see output from each node as messages are sent and received |
| R8 | The user must be able to visualise the output of the simulation on a graphical interface |
| R9 | The base station node must be able to take user input which can be used to control other nodes |
| R10 | The simulation must be capable of communications from (preselected) algorithms |
| R11 | The environment and its respective nodes should stop running when a command to do so is received |
| R12 | Nodes must be capable of moving through the environment autonomously using pathfinding algorithms |
| R13 | Nodes must be able to recognise useful sensory information, and broadcast the information back to the base station |
| R14 | The simulation code must be adaptable to the physical drones for real deployment |

Table 3.2: Table of non-functional requirements for the project

| # | Description |
|---|---|
| R1 | Must follow existing simulator conventions so as to be easy to use by researchers |
| R2 | Simulation must be able to apply interference rules accurately |
| R3 | Must be accessible to students with limited technical knowledge |
| R4 | Users should be able to interpret the output from the environment |
| R5 | The simulator must be extensible for more specific use cases |

after discovering that ns-3 is relatively bloated and difficult to use. The benefits to creating our own stimulator are that the code can then be directly translated onto the drones during deployment, as well as being directly tailored to the project domain. As a result, the functionality provided by ns-3 must be manually generated, which may increase the complexity of the project. This decision is explained in more detail in section 6.1.

CHAPTER 4

# Literature Review

*This chapter of the report focuses on examination and analysis of current research and development in the field, in order to find out more about the structure of mobile ad hoc networks (MANETs) and corresponding simulators, how drones can be employed in this field, as well as existing solutions for drone sensor networks. Then, the specific algorithms and characteristics of sensor networks must be researched, as this dictates the core functionality of the project network and how this can be achieved. This chapter will cover the topics of learning about the current state-of-the-art, and allowing that to influence our design and implementation decisions.*

## 4.1 MANETs

### 4.1.1 Structure

It is necessary for a mobile ad hoc network (MANET) to be capable of self-configuring without an infrastructure, where mobile devices can connect without wires. This is because mobile nodes communicate by forming a network dynamically, and so they lack a fixed infrastructure or centralised control. Nodes are independent, and the network topology is temporary as each node dynamically self-organises. This means that each device must be a router, and each node must be aware of the organisation when forwarding packets [50]. When nodes wish to communicate with one another they can use a variety of different mediums, such as Wi-Fi or radio waves, but they must depend on intermediate nodes to route their data packets in the event that the source and destination are not in range of one another [42].

MANETs may employ a specific network model, or a general approach in which nodes are aware of other nodes in range and their task only. Various methods of forwarding data exist, such as flooding data by broadcasting to all neighbours, or by using predefined metrics to determine the optimal path [50]. It is in stark contrast to a centralised wireless network, as there are no access points in a managed infrastructure. Forwarding of data is dependent on the presence of network connectivity, so nodes must

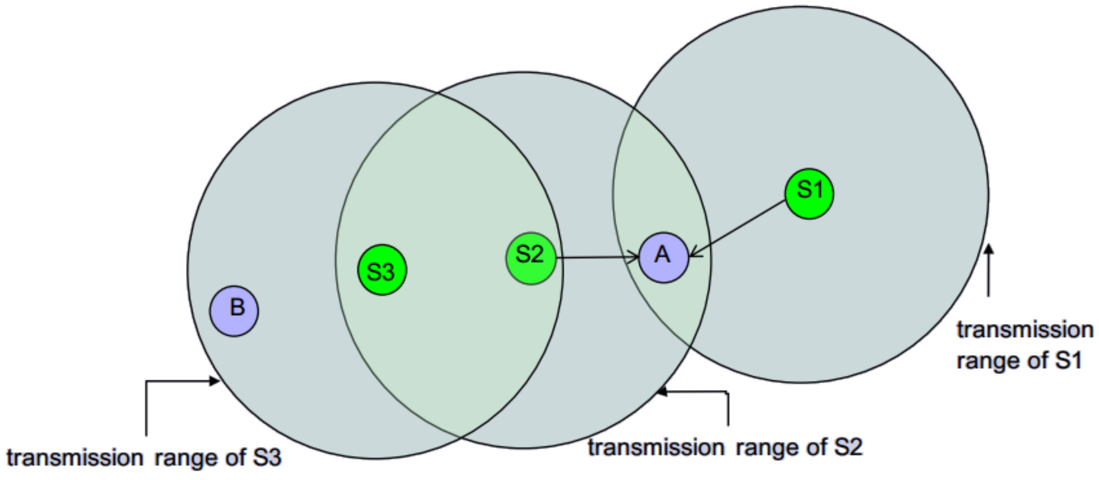


Figure 4.1: Example topology of a MANET [49]

be able to create and join networks arbitrarily, which requires high adaptability to constant changes in topology. MANET systems must also consider problems such as limited power and storage for each individual node; the network must be fault tolerant such that it can deal with problems such as when connectivity is lost or a node powers down. Energy and power consumption are priority concerns which are directly related to the hardware resources available to each node, and the environment in which the network is deployed [48].

### 4.1.2 Data Mining and Monitoring

The core functionality of MANETs is to provide a network of mobile nodes which can communicate with one another and pass information by routing through intermediate nodes where necessary. In a sensor network, each node is tasked with collecting specialised data through the use of sensory peripherals, and this data must be passed effectively to its destination. Given the dynamic nature of MANETS, considerations must be made for the constant changing topology in order to ensure that the networks's integrity and availability is not compromised and that nodes can effectively communicate with one another at any time [42]. This includes problems such as losing connection or going out of range, which will cause the loss of packets when communications links are inevitably cut and messages lost.

Connections between nodes and algorithms for data mining must be carefully established and monitored in order to ensure secure and efficient routing. For example, the data collected by each individual sensor node which is monitoring an environmental feature will typically register similar values

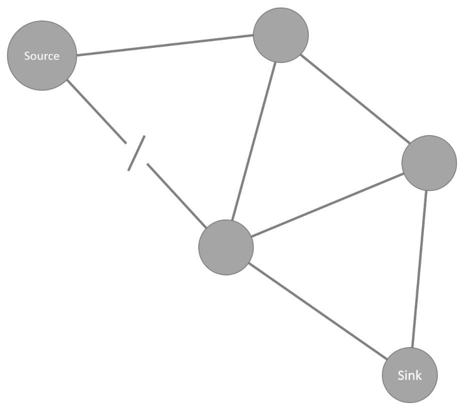


Figure 4.2: Example of a base station network with a broken link

due to spatial correlation, so there is a danger for unnecessary overheads as a result of unneeded data. By measuring spatial correlation between data sampled by different sensors, specialised algorithms can be developed to implement more efficient data mining, as well as optimising routing strategies [28]. In order to aggregate data more effectively, a stationary base station node may be included which acts as the data sink, and may be capable of passing user input to nodes, which would allow for some stability and interaction with the network by the user.

### 4.1.3 Sensor Networks

A sensor network is a network in which each node has sensors to monitor physical or environmental conditions such as temperature, sound or pressure. This data can be routed through other nodes, and it is possible to control sensor activity as networks are bidirectional [25]. While sensor networks themselves can range from wired to wireless, or have predefined topologies such as a simple star or mesh network, a drone sensor network would be classified as a Mobile Wireless Sensor Network (MWSN) for obvious reasons.

The biggest challenges to MWSNs, in general terms, are hardware and environmental. Power consumption of sensing devices should be minimised and sensor nodes should be energy-efficient, since their limited energy resource determines their lifetime. As with any mobile network, the varying topology due to the mobility of nodes results in further usage of the already limited resources in terms of battery power. If it is possible, nodes could consider powering off unused parts to save power, like wireless communications as a part of physical layer algorithms, or sensors immediately after takeoff, where information gathering may not be required en route to the target area. However, this is also dependent on the environment, which may require additional precautions in the event that it is hazardous or requires heavy monitoring. One of the most effective method for energy efficiency is the use of optimal routing protocols, which will be discussed in later sections [51].

## 4.2 Simulations

Network simulators are software applications that predict the behaviour of a computer network, making them a popular method for experimentation with models for MANETs due to their ability to cheaply and flexibly model a solution. These simulation tools make it possible to model fault tolerance and scalability, make monitoring easy, and provide reproducibility [53]. It should be noted that there are significant variations in the way that individual simulators operate, which cannot be expressed in terms of accuracy. At best, they can be said to be dependable and realistic with reasonable proficiency at modelling unpredictable error in a real environment. Simulation events can be split into continuous and discrete simulation, where continuous simulation makes use of analytic models, and so can hardly be applied in practice. Conversely, discrete simulation allows for parallelism, scalability, and speed up, as well as developing trace files containing a description of each event that occurred [26]. These trace files can then be used to generate a visualisation of the simulation, so they would be preferred for our project (or they will at least influence how we build our simulation and visualise it). This section gives an overview of the potential solutions available for simulations, and the alternatives to using a simulator.

### 4.2.1 ns-2

ns-2 (network simulator 2) is a discrete-event network simulator for research and educational use. Its software infrastructure encourages the development of simulation models which are sufficiently realistic to allow it to be used as a real-time network emulator, and is the basis for many existing real-world implementations [37]. In particular, it provides a set of randomised mobility models, including random waypoints, such that the advanced node mobility constitutes a progression towards realistic simulation. ns-2 programs are scripted in OTcl, with some components written in C++ . Despite its popularity, ns-2 suffers for its lack of modularity and inherent complexity, but does provide excellent documentation. It is the predecessor of ns-3, although it has a greater diversity of models given its age. ns-2 also allows for visualisation of the simulation through its GUI out.nam with the use of trace files, as shown in figure 4.3 [26].

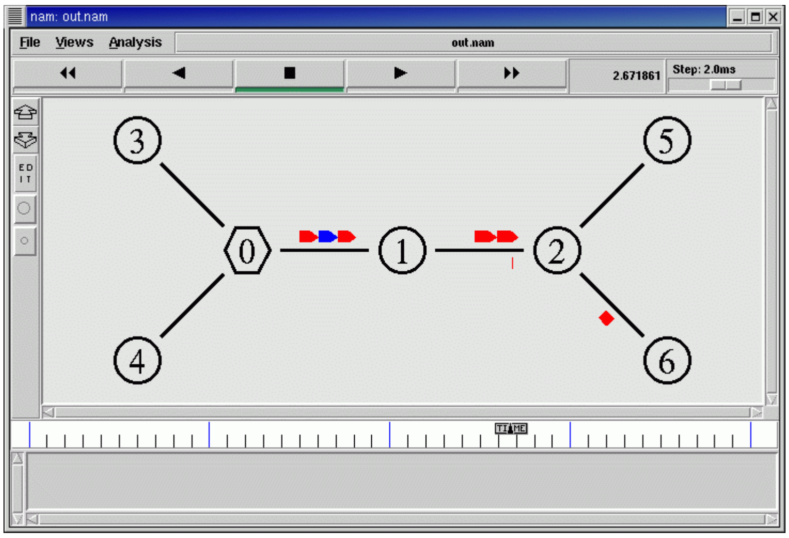


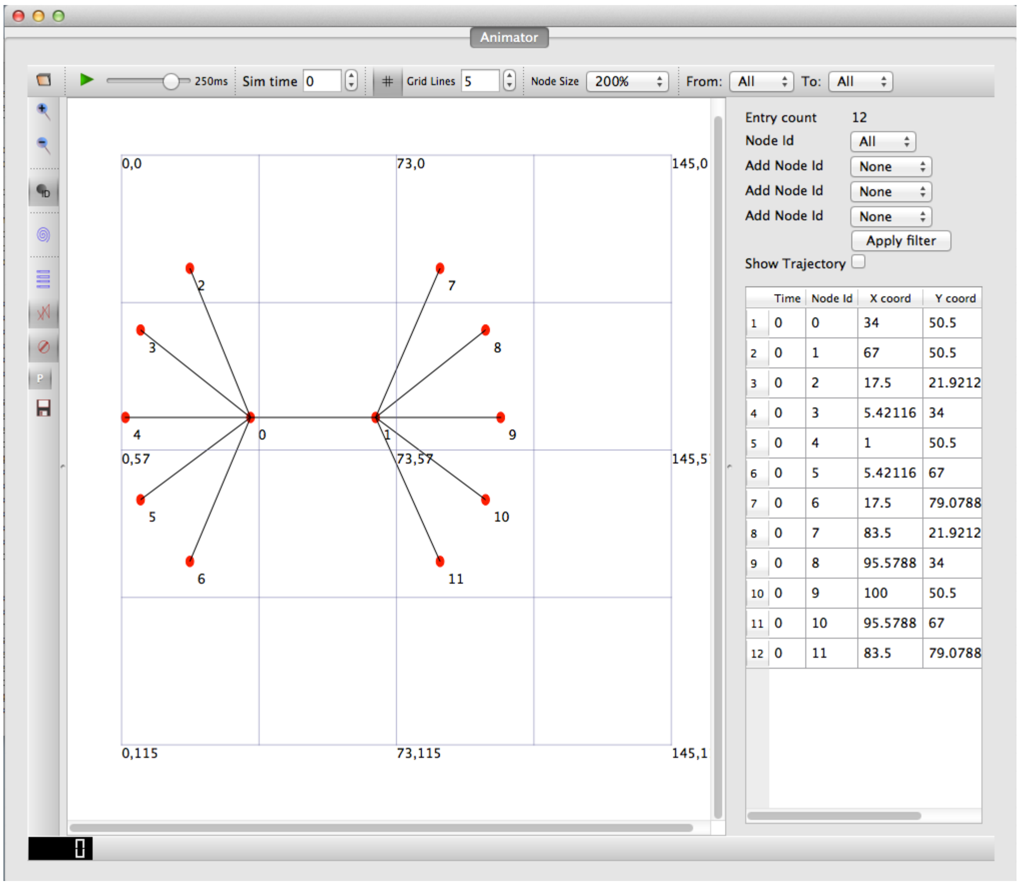
Figure 4.3: Visualisation for ns-2 trace files in out.nam [26]

### 4.2.2 ns-3

ns-3 is a newer software application which followes ns-2, focusing its efforts on improving the core architecture, software integration, models, and educational components of ns-2 [33]. ns-3 differs from ns-2 in that it is purely written in C++, with optional Python bindings. New scripts can therefore be written in C++ or in Python. This version is recommended over ns-2 by the creators of ns due to features which are not available in the previous version, such as an implementation code execution environment which allows users to run real implementation code in the simulator. Additionally, it has more detailed models, as well as being actively maintained and developed. ns-3 is not backwards compatible with ns-2; it is an entirely new simulator, with several models which were written for it in C++, and many which have been ported [37]. The visualisation platform for ns-3 is the animator NetAnim, shown in figure 4.4 and based on the Qt toolkit.

### 4.2.3 GloMoSim

Short for Global Mobile Information System Simulator, GloMoSim is a network protocol simulation software for parallel simulation of wireless and wired network systems using the parallel programming language PARSEC [59]. It uses a message-based approach to discrete-event simulation, where network layers are represented as objects called entities, which handle events represented by timestamped messages. The library has been designed to be extensible and composable, and its modular imple-



Sim time 0 | Grid Lines 5 | Node Size 200% | From: All | To: All

Entry count 12
Node Id All
Add Node Id None
Add Node Id None
Add Node Id None
Show Trajectory ☐

| | Time | Node Id | X coord | Y coord |
|---|---|---|---|---|
| 1 | 0 | 0 | 34 | 50.5 |
| 2 | 0 | 1 | 67 | 50.5 |
| 3 | 0 | 2 | 17.5 | 21.9212 |
| 4 | 0 | 3 | 5.42116 | 34 |
| 5 | 0 | 4 | 1 | 50.5 |
| 6 | 0 | 5 | 5.42116 | 67 |
| 7 | 0 | 6 | 17.5 | 79.0788 |
| 8 | 0 | 7 | 83.5 | 21.9212 |
| 9 | 0 | 8 | 95.5788 | 34 |
| 10 | 0 | 9 | 100 | 50.5 |
| 11 | 0 | 10 | 95.5788 | 67 |
| 12 | 0 | 11 | 83.5 | 79.0788 |

Figure 4.4: An example of NetAnim simulation animation [34]

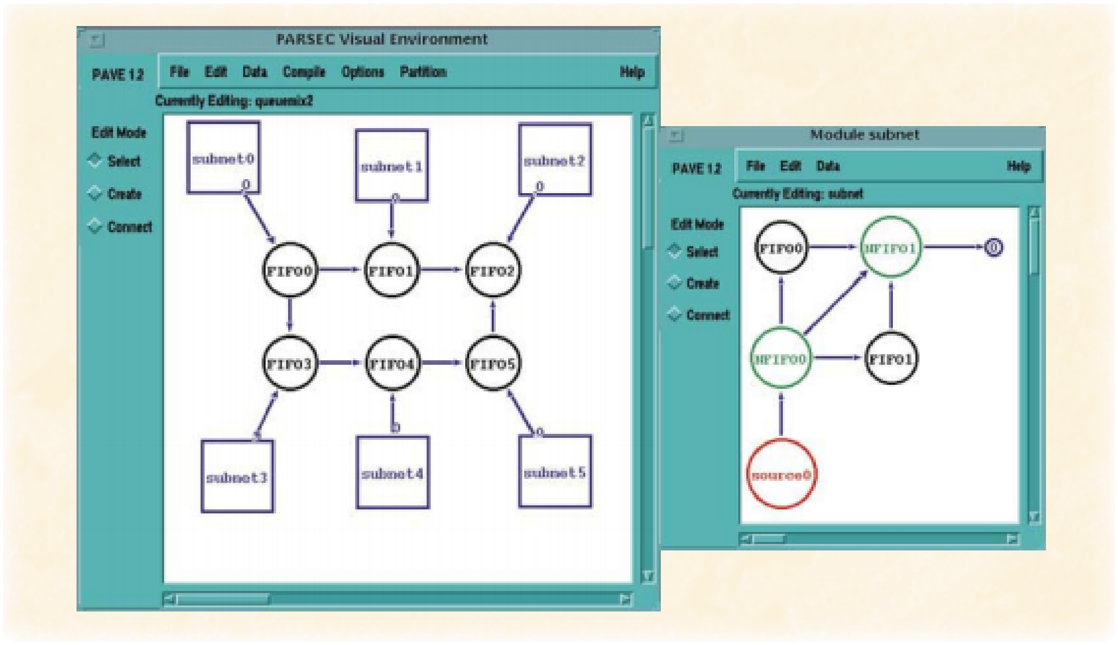
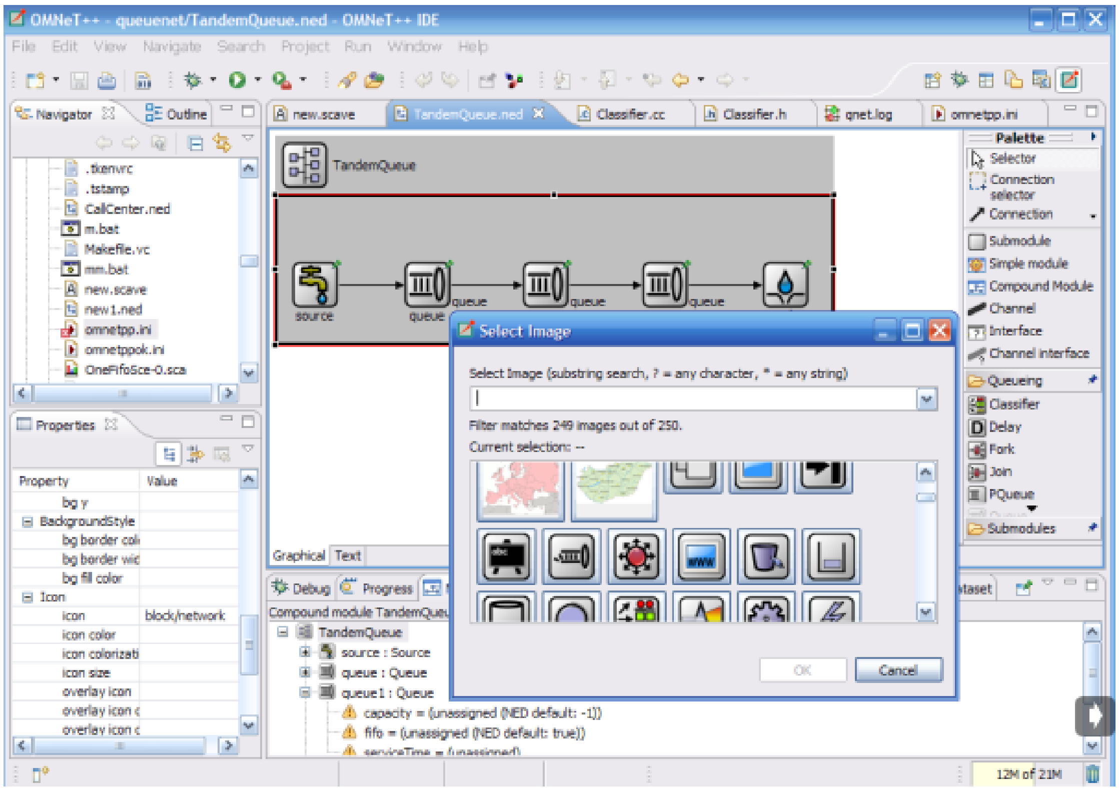


Figure 4.5: Example output from the PARSEC Visual Environment[26]

mentation enables consistent comparison of protocols at a given layer. While research into parallel simulation is relatively limited, and areas such as signal interference and attenuation are inherently more complicated, there are clear implications for a significant increase in performance in terms of spreading out the simulation overheads, which may outweigh the inherent complexity. The PARSEC Visual Environment (PAVE) shown in figure 4.5 has also been developed to support the visual design of GlomoSim event simulations [26].

### 4.2.4 OMNet++

OMNet++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators [35]. This includes wired and wireless communication networks for sensor networks, ad hoc networks, Internet protocols, and so on. OMNet++ is not a network simulator in and of itself, but provides a wide range of tools and extensions, including an Eclipse-based IDE (Integrated Development Environment). It provides a component architecture for models; components are programmed in C++ and then assembled into larger components and models using a high-level language (NED) which also has GUI support, as shown in figure 4.6. OMNet++ was designed for the general use case, as opposed to being a specialised simulator, and its modular architecture allows for



Select Image (substring search, ? = any character, * = any string)

Filter matches 249 images out of 250.

Current selection: --

Compound module TandemQueue
- TandemQueue
  - source : Source
  - queue : Queue
  - queue1 : Queue
    - capacity = (unassigned (NED default: -1))
    - fifo = (unassigned (NED default: true))

Figure 4.6: Example of the OMNet++ NED editor IDE [35]

the simulation kernel (and models) to be easily integrated into wider systems [55].

### 4.2.5 Testbeds

As an alternative to simulations, testbeds are in-lab networks built and used by researchers, which involves creating and connecting physical devices for testing the application of real networks. As a result, the cost of hardware coupled with the difficulty of managing applications in terms of deployment and monitoring creates limitations on the complexity and scalability of the network. While it can be said that the physical testing environment is much more effective and realistic than a simulation, the lack of larger testbeds of over 50 nodes prevents this from becoming a major component for validating new concepts and applications for MANETs [26].

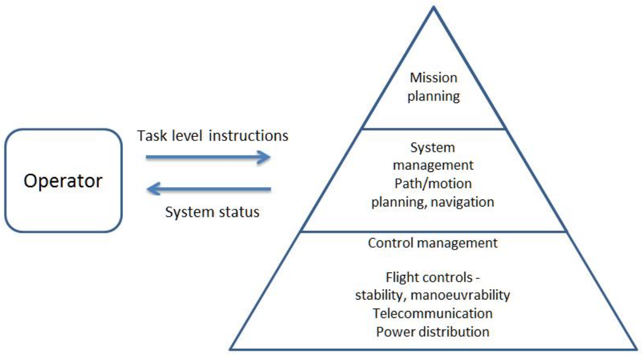


Figure 4.7: Hierarchy of drone autonomy and user control [57]

## 4.3 Drones

### 4.3.1 UAV Components

As has been discussed in previous chapters, drones have a variety of sizes, utilities, costs, and applications, from commercial to military purposes. However, they will all have to make use of sensors, actuators, software, and a power supply in battery form. The body includes a fuselage and wings for planes, tail rotor for helicopters, and a frame and arms for multirotors [23]. The combination of different layers of computer software with different time requirements, also referred to as the autopilot or flight stack, may require specialised externally supported hardware such as Raspberry Pis or Beagleboards. However, on-board classical operating systems may be too intensive for some embedded systems, presenting problems if the aircraft is dependent on fast response times [36].

Depending on its usage, a drone will also range from being fully remotely piloted to complete autonomy such that the drone will function without any user input. Is important to note that the degree of autonomy i.e. the extent to which the aircraft is independent from operator assistance, is different to the level of autonomy in terms of the scope of tasks it can perform and their sophistication [58].

### 4.3.2 Drone Autonomy

Autonomy is split into different layers of control, from basic manoeuvrability and flight controls to system management, navigation and mission planning. It is assumed that the user will be able to provide task level instructions which the drone is capable of processing. Figure 4.3.2 shows an example of the basic autonomy controls for a UAV.

An autonomous drone would be expected to have various features for flight control and communications, such as:

- **Autonomous takeoff and landing**
- **The ability to return to its starting location**
- **A failsafe to land the drone in the event of a fault**
- **The ability to stabilise and maintain altitude in unstable conditions i.e. hover and self-level in wind**
- **Possible GPS waypoint navigation**

Some drones will already contain a microcontroller with pre-programmed instructions for navigation, including tricks such as rolls and loops; however, an external small computer may be required to implement mid-level algorithms such as those involved in physical routing and communications. For example, determining an optimal path whilst avoiding obstacles, or passing messages in a network to other nodes. The attachment of an external computer such as a Raspberry Pi would allow for the drone to be programmed directly, which is the most common solution achieving autonomy, which will be discussed later.

Reactive autonomy, such as real-time collision avoidance, relies on situational awareness, which is typically provided by range sensors. However, a network of drones with GPS capabilities would be able to utilise its own spatial awareness, knowledge of other node positions, and protocols designed to avoid collision.

### 4.3.3 Classifications

UAVs have been classified by such areas as weight, functionality, range and usage. Given that this project focuses on the use of consumer-level drones, we will focus on the classification of drones by usage [18]. The fundamental types of drones are as follows:

- **Commercial off-the-shelf (COTS).** This refers to drones which are ready to fly immediately with no prior or specialised knowledge required
- **Bind-and-Fly (BNF) and Almost-Ready-to-Fly (ARF).** BNF and ARF drones typically contain most or only some of the parts required for the drone, but may be missing components such as the controller or battery, and require some knowledge to assemble and configure

- **Midsize Military and Commercial Drones.** These drones are more expensive, with greater utilities and/or bespoke functionality
- **Large Military-Specific Drones.** Military drones may be used for combat, logistics, research and development, reconnaissance or as target and decoy drones
- **Stealth Combat Drones.** Military drones which are used for stealth operations, including direct combat and reconnaissance

For a drone sensor network, it is important to be aware of power consumption, the weight, and the range of the drone in order to maximise its efficiency and understand the limitations of a single drone in the network. COTS drones (or quadcopters) will typically carry a payload (such as a camera), and additional peripherals such as the external computer required for implementing autonomy and sensing equipment will add to the weight of the drone, so considerations must be made for load capacity. Additionally, range will be directly influenced by the choice of medium, such as Wi-Fi or radio. All of these additions to the drone will affect the so-called sensor 'Degrees of Freedom', which is the number of axis and sensors combined for balancing a plane, helicopter or robot, which typically is equated with the complexity (and therefore, price) of a UAV [3].

## 4.4 Existing Solutions

This chapter has researched the structure and characteristics of sensor networks, including implementation and testing through simulators, as well as examining different varieties of drones and their components, identifying how this can be related to sensor networks. This section will highlight some of the advances and complications which arise within sensor networks, autonomous drones, and explicit combinations of both, by examining existing implementations and research into the field.

### 4.4.1 Autonomous Drone Implementation

UAVs and autonomous aircraft have become increasingly popular as an area of research and development, and a number of open source autopilot projects exists which can facilitate autonomy, allowing hobbyists and businesses to use them on small, remotely piloted aircraft inexpensively. Among these are GPS-based projects such as Paparazzi and ArduPilot, which will be analysed to see how they can be applied to drones, and how they differ in terms of their choice of operating system and implementation. Other methods of achieving drone autonomy will also be discussed in this section.

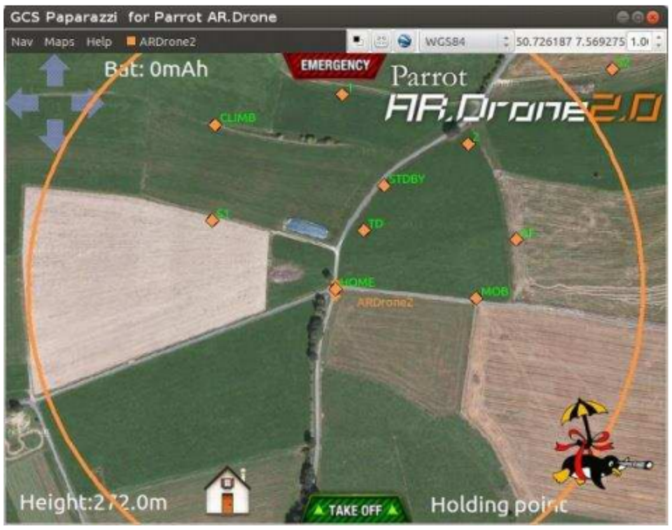


Figure 4.8: The Ground Control Station of Paparazzi for the Parrot AR drone 2[43]

**Paparazzi**

Remes et al. [43] conducted studies on the use of a GPS-based open source autopilot system known as Paparazzi in order to create a swarm of autonomous Parrot AR drones. The Parrot firmware can be enhanced with Paparazzi firmware, or a standalone version of the Paparazzi firmware can be used. The autopilot will lock onto sufficient GPS satellites, establish a signal and enact a user constructed flight plan.

Paparazzi includes a base station GUI known as the Ground Control Station (GCS) shown in figure 4.8 for the purpose of preparation, simulation, monitoring and analysis, and each drone can be located on the map provided. The overlay includes information about the drones such as height, waypoints for the flight plan and controls. The addition of more drones is arbitrary; installing the firmware on each drone will allow it to be located on the GCS and interacted with.

Both inner and outer loop control is performed by Paparazzi, shown in figure 4.9, which means that the flight plan controls the thrust and attitude for achieving the right speed and position, and the safety pilot has control over the vehicle via the RC link, which refers to the joystick connected to the GCS. The control loops are open source, however, and can be changed by the user.

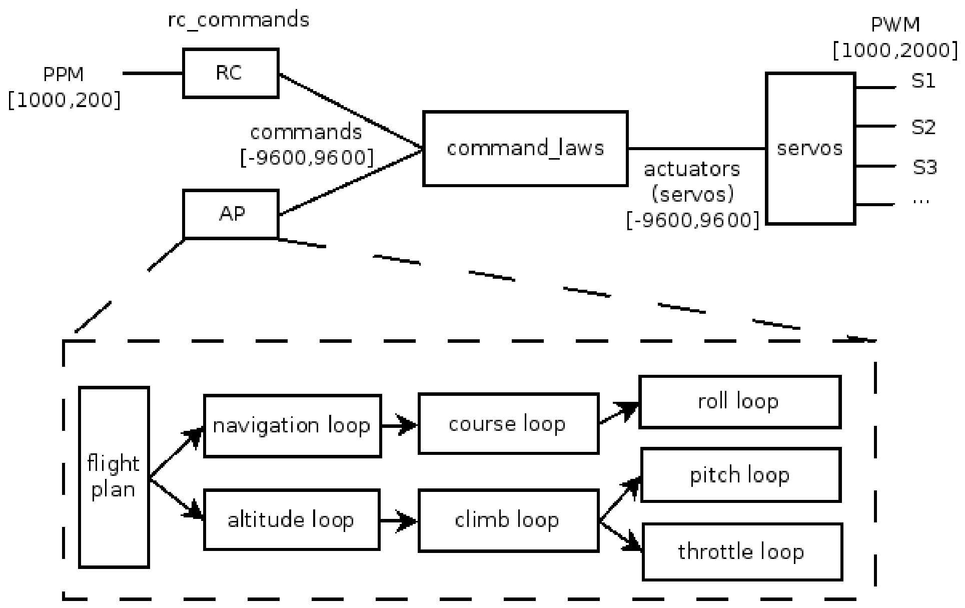


Figure 4.9: Global view of the control loops used by the Paparazzi airborne code for navigation, guidance and control

**Ardupilot**

He Bin et. al [5] discuss the design, modelling, implementation and testing of an autonomous UAV with the use of Ardupilot, which is another open source UAV autopilot platform. Ardupilot is a portmanteau of Arduino and pilot, the former being open source electronics prototyping platform upon which Ardupilot is based. It is similar to the Paparazzi project in terms of its free software approach and GPS-based implementation, shown in figure 4.10.

The Ardupilot is a custom PCB with an embedded processor combined with a built-in hardware failsafe to switch between RC control and autopilot control, such that the user can decide the amount of direct control over the UAV themselves. The autopilot controls navigation (by following GPS waypoints) and altitude by controlling the rudder and throttle. The Ardupilot software is an open source Arduino environment which can be easily run and edited on Windows, Mac OS X and Linux, in contrast to Paparazzi, which runs exclusively on Ubuntu, and requires more specialist knowledge (including a reasonable grasp of the command line). However, it requires for waypoints to be set manually and before takeoff, which limits the level of autonomy which can be achieved. Ardupilot is widely regarded as the 'breakout' model for autopilots, due to its ease-of-use, with somewhat reduced functionality and lower specs.

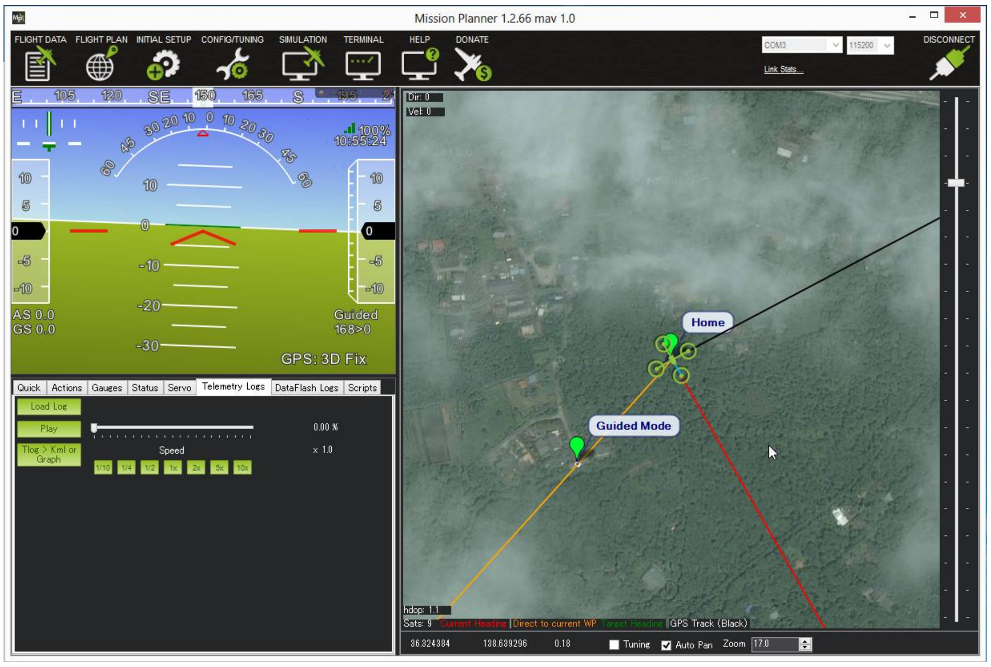


Figure 4.10: Example output of the Ardupilot Mission Planner software [5]

### Map Stitching

In order to solve the problem of autonomous drone control, Visser et al. [56] introduce a method of navigation using image stitching to build a visual map of an indoor environment. Using the frames from the Parrot AR Drone's (which we are using for the project) downward-facing camera, sets of matches between two camera frames are used to estimate a tomographic model for the apparent image motion. Thus, a model can be composed with the estimated motion of the rest of the images in order to build a mosaic. Additionally, it makes use of sonar sensor input to detect obstacles which are then added to the camera image during the image stitching process. An example result of the map stitching is shown in figure 4.11.

Calculations for frames are further influenced by a number of additional sensors such as inertial sensor data, which can be used to estimate current position, calculate an estimate for the resulting position, and use this as input for the map stitching algorithm.

Results showed that the visual map created by a simulated AR drone contained fewer errors than a map from the real drone, which could be attributed to the effect of automatic white balancing of the real camera. However, the map stitching method is able to map small areas visually with sufficient quality for human navigation.

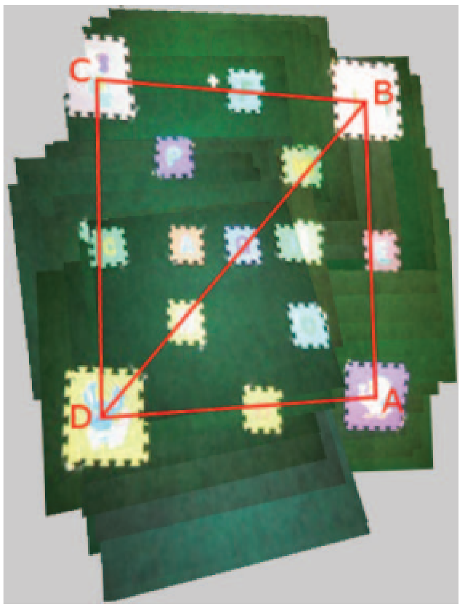


Figure 4.11: Map created by the map stitching method using camera images taken by the AR drone [56]

### Drone Localisation

Dijkshoorn et al. [11] introduce a similar method for autonomous navigation which uses sensor and motion models to localise (determine the location of) an autonomous Parrot AR drone. This method involves creating a visual map using texture and feature mapping derived from the position and attitude of the drone in the world, whereupon the map can be used for absolute position estimates and subsequent navigation. By creating a visual map, it is possible to sufficiently conduct human and artificial navigation through an indoor space. Simulation of the AR drone is also performed to aid in the estimation and testing of localisation methods and mapping.

Firstly, to initialise the visual localisation and mapping algorithm, the internal sensors of the drone are used to give information about the position and movement of the drone. The inertial sensor measures body acceleration and angular velocities, which can be converted to an estimated attitude. Sonar sensor is used to measure the altitude of the drone, and an estimation of body velocity is calculated using inertia measurements and optical flow from the relative motion between camera frames. This information is used to build a visual map of the environment.

Results showed that the mapping method is able to map areas visually with sufficient quality for both human and artificial navigation purposes. The simulation contained fewer errors than the real Parrot AR drone, which were attributed to measurement noise, however, the localisation and

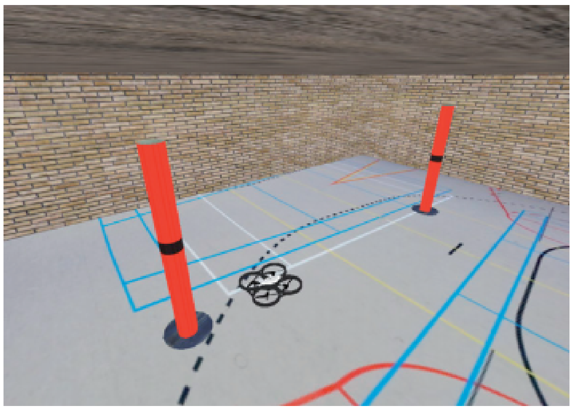


Figure 4.12: 3D model of a gym with realistic ground and wall textures for AR drone simulation [11]

visualisation methods were demonstrated to make autonomous navigation possible. This method performs similarly to the map stitching method shown above, as it uses GPS to find the location of the drone in local space and build a map of the area. This suggests that implementing localisation using GPS are a reasonable method of providing autonomy in terms of navigation.

### 4.4.2 UAV Sensor Networks

#### Collaborative Drone Tasking

Recognising the difficulty in coordinating multiple drones to perform a task collaboratively in a sensor network, Mottola et al. [30] introduce and provide an analysis of the concept of team-level programming, which can express collaborative sensing tasks without the complexity of such areas as concurrent programming or parallel execution. They create the VOLTRON programming system to explore this concept and enable coordination of multiple drones with a specific focus on active sensing applications, and dynamic re-tasking.

This method of team-level programming was developed in contrast to drone-level programming and swarm programming, which focus on addressing and tasking each individual drones with navigation, communication, and sensing, or executing a single set of basic rules and operating only on their own local state, which is difficult to scale up to multiple drones. Team level programming allows for sophisticated collaborative sensing tasks without resorting to individual addressing, enables simpler code execution, and produces marginal overhead in terms of CPU, memory and network utilisation.

Mottola et al. concluded that the implementation of a team-level programming model in VOLTRON provided a significant improvement in execution time for a given task when using multiple drones.

Application level performance of real-time drone applications were assumed to depend greatly on hardware design choices and execution times could be variable depending on the environment. Nonetheless, the use of a testbed, as well as real deployment, produced results showing that VOLTRON does introduce overhead, but that memory consumption and CPU utilisation are independent of the number of drones, and this system would likely scale effectively beyond 100 drones before reaching resource limits.

#### Emergency Services

One of the most common applications of UAVs aside from commercial or military is in the field of disaster prevention, management and relief. Fuego, the Fire Urgency Estimator in Geosynchronous Orbit, is a system created by researchers at UC Berkeley which locates and identifies fires using drones, planes and satellites with special infrared cameras [7]. The network of drones will navigate through fire-prone areas of the country, taking photos at a wavelength of light that fire emits, but which is invisible to the human eye. A geospatial information system (GIS) processes the images in real time to estimate the risk that a hotspot could grow into a serious fire, where the difference in each sample can highlight the eruption of a forest fire. With this system it is possible to identify forest fires within 3 minutes of their eruption [17].

Research into the application of networked UAVs for disaster management is also being conducted to test the ability of small-scale, battery-powered, and wirelessly connected UAVs for disaster management. These UAVs can carry cameras to locate problems such as a wood fire or a large traffic accident, and deliver high-quality image or video data. Quaritsch et al.[41] discuss the potential of the tight integration of sensing and controlling UAVs, which allows for a whole new set of applications and research challenges. Following disasters such as Hurricane Katrina, UAVs were equipped with a digital camera capable of both still and video imagery for post-disaster data collection and damage inspection of multi-storey commercial buildings [1]. The most prominent challenges to UAV sensor networks are described as optimisation for area coverage, as well as concerns over limited resources, especially energy. Limited site access and landing and takeoff environments also required that there were greater flight distances to reach areas of interest, which further emphasises the importance of energy efficiency.

## 4.5 Routing

Two of the biggest challenges for UAV Sensor networks which have been introduced so far are the issues of limited resources and the environment. In order to minimise overheads, reduce computation time,

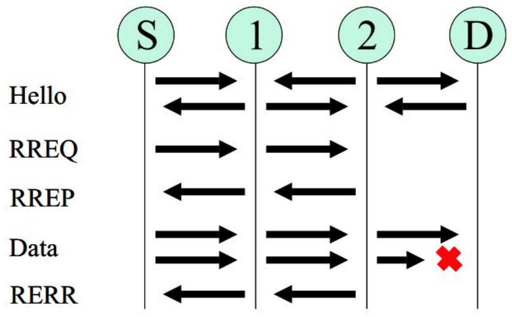


Figure 4.13: Packet delivery rate for a 50 node model (30 traffic sources)[8]

optimise pathing and battery usage, and introduce energy-efficient communication, it is essential to consider different methods of routing in sensor networks. The length of operation time and success of the network relies on firmly established protocols which are optimised for the application. In this section, we examine some of the existing algorithms for routing, giving a brief discussion of the advantages and disadvantages of each method, and how they deal with specific challenges such as robustness and energy efficiency.

### 4.5.1 Ad Hoc On-Demand Distance Vector

Ad Hoc On-Demand Distance Vector (AODV) routing is a non-position based protocol which enables dynamic, self-starting, multi-hop routing between participating mobile nodes. It operates by sending Route Request (RREQ), Route Reply (RREP) and Route Error (RERR) messages between nodes, which are used to establish endpoints of a communication connection (as shown in figure 4.13). In AODV, the source node and the intermediate nodes store the next top information corresponding to each flow for data packet transmission [38].

AODV is a reactive (on-demand) routing protocol in that it does not do anything if an active route already exists between communicating nodes, so connection setup delay is less, and destination sequence numbers are used to find the latest route to the destination. The AODV protocol also keeps the overhead of messages small, and has a great advantage in overhead over simple protocols which need to keep the entire route from source to destination in their messages. This protocol is also loop-free and avoids the counting to infinity problem through the use of sequence numbers. Despite this, such protocols have a high latency time in finding a route (because routing is done in reaction to a message being sent), and excessive flooding can lead to network clogging.

More information about AODV can be found in section 6.4.5.

### 4.5.2 Dynamic Source Routing

Like AODV, Dynamic Source Routing (DSR) is an on-demand routing protocol for mobile ad hoc networks. DSR divides the routing problem into two parts: route discovery and route maintenance. Route discovery finds routes between nodes by flooding the network with discovery packets until the destination node is reached. A route reply message is propogated back and cached. Unlike AODV, routes are retained indefinitely (unless they are broken), meaning that when nodes are static the overhead from using DSR is zero[22][21].

Route maintenance is performed whenever a data message is being sent through the network. Each node is responsible for ensuring that messages it forwards are received (this is often done through link-level protocols or passively on retransmission). In the event of a link being severed and a message being lost, the node which attempted to send over the broken link returns a route error message to the originator of the message. Routes using this link are removed from the cache.

One of the advantages of DSR is that new routes and network updates are retained by every node which witnesses them (each node a route error message passes through will remove the broken link from their routing cache). Additionally, DSR nodes will attempt to shorten routes, salvage packets sent over broken links, and reason about link uni/bi-directionality. This focus on preventing the duplication of messages means that DSR is fast with very low overhead.

### 4.5.3 Greedy Perimeter Stateless Routing

Greedy Perimeter Stateless Routing (GPSR) take a different approach to routing than AODV and DSR in that it is stateless. Instead of using caching to reduce the amount of overhead, each node memorises the geographical locations of its neighbours and uses this to greedily forward messages. The optimal (one hop) choice for routing is the neighbour which is closest to the destination node[24] (see figure 4.14).

The disadvantage of retaining as little information as possible is that there is an associated messaging overhead. GPSR nodes will routinely beacon information about their location, though can be mitigated through the practice of attaching beaconing information to all packets sent. This is exactly what AODV and DSR explicitly try and avoid through the use of caching. There are also scenarios where the optimal node to forward to is not the one chosen by the greedy algorithm. In this case, the right hand rule is used to route packets to their destination.

Benchmark simulations of GPSR and DSR reveal that the overhead incurred by beaconing is normally less than that of more stateful solutions, as shown in figure 4.15. It is worth noting that

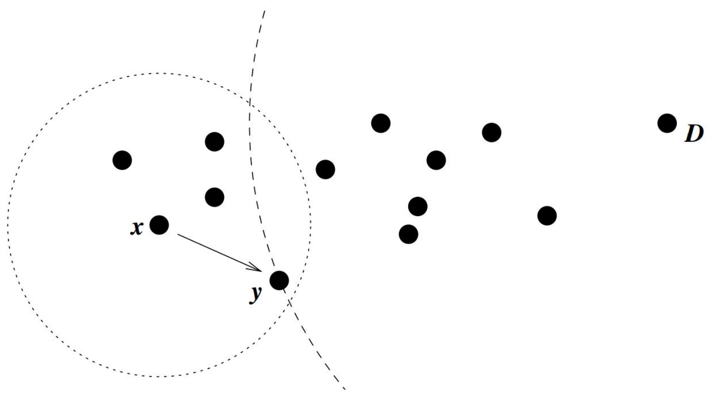


Figure 4.14: An example of greedy forwarding in GPSR[24]

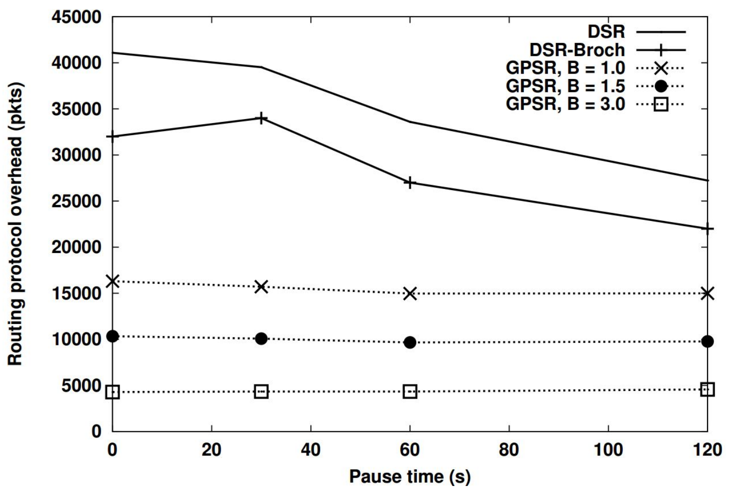


Figure 4.15: A comparison between DSR and GPSR with different beaconing intervals *b*[24]

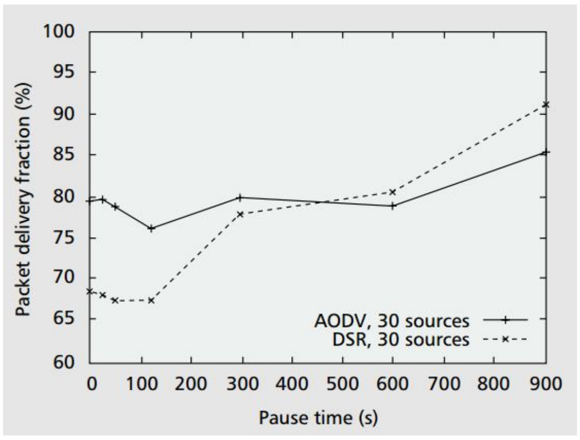


Figure 4.16: Packet delivery rate for a 50 node model (30 traffic sources)[39]

### 4.5.4 Comparison

Figures 4.16, 4.17, and 4.18 show a comparison between AODV and DSR in similar conditions. It can be seen that DSR tends to perform better in terms of overhead when considering short pause times (typical of a drone network), but that AODV results in a higher percentage of packets being delivered. Given the need for high priority messages to be delivered as quickly as possible in a quadcopter network (to prevent a collision, for example), it was decided that AODV would be a more appropriate algorithm to implement as a reference communication model in octoDrone than DSR. GPSR is very efficient but suffers from the problem that geographic routing tends to break down when nodes are moving at high speeds most (if not all) of the time. As information on neighbours becomes less certain it is harder to make efficient routing decisions without drastically decreasing the broadcast interval. After some consideration it was decided that AODV would be more broadly applicable, with GPSR being a worthwhile future addition.

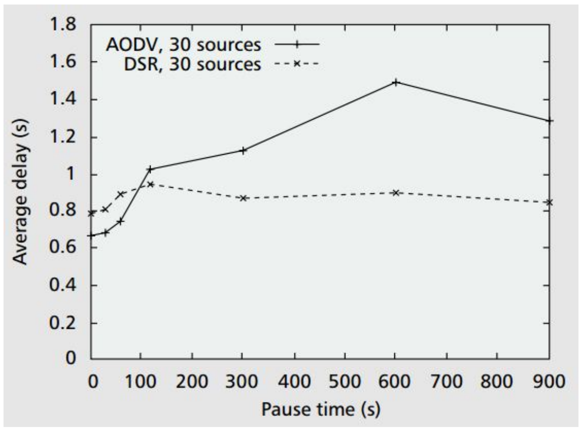


Figure 4.17: Average data packet delay for a 50 node model (30 traffic sources) [39]

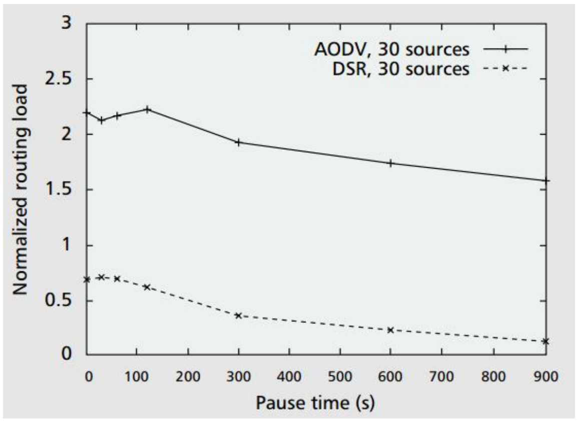


Figure 4.18: Normalised routing load for a 50 node model (30 traffic sources) [39]

CHAPTER 5

# Design

*This chapter of the report will describe in detail the design decisions which were taken when the project software was being developed. Organised by the major components and libries, the hope is that it will be clear which decisions were taken and why, and how the taking of certain decisions impacted the design choices made for others.*

## 5.1 Methodology

## 5.2 System Architecture

### 5.2.1 Network Simulation

#### Fundamental Design

The fundamental design of the simulator focusses on the concept of portability, if implemented correctly, programs written for use on the simulator should be trivially transferable to actual drones of any type, so long as those drones are configured correctly and have the appropriate environment set up. Another focus of the simulator is on simplicity, it should not be prohibitively difficult to create even simple programs. Ideally the simulator should handle as much of the work that is not relevant to actual drones as possible, allowing users of the system to focus on what the simulator is simulating rather than getting the simulator to work.

#### Implementation approach

There exist two possible approaches to the overall design of the system that were considered for this project: serial simulation and parallel simulation.

Serial simulation would define a strict ordering over which each element inside the simulation would run, each simulation element would be run sequentially, allowing a single “frame" of its program to be run. This provides several advantages, e.g: the simulation would have an inherent synchronicity; no drone or other element could ever get “out of sync" with the rest of the program, creating a serial simulation would be significantly less work than a parallel one as no thread interactions would have to be considered. However, this approach also has several disadvantages. For example, creating programs for the simulator to run would be significantly less intuitive from the users point of view, the function to be ran on each element of the simulator would have to be a single “frame" of that elements simulation, which could lead to confusion (or even the simulator hanging if the user does not realise this). Another problem is that this simulation method is not particularity realistic, in a physical deployment each drone and other element will be acting independently, not in the strict ordering implied by serial simulation.

Parallel simulation involves running each element of the simulation on its own thread of execution, each element runs its own programming independently of all the others, allowing elements to act simultaneously. The disadvantages and advantages of this method are effectively the polar opposites of the disadvantages and advantages of the serial method respectively. In other words, parallel allows for more intuitive program writing, creating the entire program at once, and the very concept of the simulation being more realistic as the drones are acting simultaneously for advantages. Lack of synchronicity (drones can be several entire loop iterations ahead of other drones for instance) and thread interactions for disadvantages.

Ultimately, parallel simulation was chosen as the method of choice, choosing the more difficult but more realistic and useful approach over the easy abstraction. This was primarily chosen due to the fact that the simulator is designed to be a testing bed for actual programs being run on actual drones. Thus the easier it is to transfer programs over from one system to the other the better, and allowing users to create the entire program in a single function rather than that function representing a single frame of execution allows this to happen more easily.

**Environment**

The environment should be the majority of the predefined segment of the simulator. The environment should handle all of the possible sensor data that is to be used by the simulator to be gathered by the drones in the simulation. The environment can also be seen as the actual simulator itself, whereas drone, base station and communications code are the “programs" that run on the simulator, thus the environment should handle any and all tasks that would not need to be handled in physical deployment. This means that the environment handles tasks such as moving drones around, passing messages between them and sending the sensor data that is requested when it is requested.

**Application Programming Interfaces (API)**

The simulator has to be as flexible as possible, allowing for arbitrary drone programs to be run with it. Due to this, the API was designed in order to maximise usability. To this end, the simulator is effectively an unfinished program, the user “finishes" this program with their own code, extending classes as necessary in order to run the code that they want the simulator to handle. This also mirrors how the code would interface with the actual physical drones as the code written in the extensions of the drone class is the code that will be running on the physical drone itself, similarly code for communications modules and base stations will be simply transferable to actual physical implementations of these systems.

### 5.2.2 Communications Modules

In order to remain as generic as possible, the simulator itself should not prescribe a set way of performing communications routing. Rather, it is up to the user to specify the routing algorithm they wish to use. This needs to be done in a way which makes it easy to specify different routing algorithms for different nodes in a simulation, i.e. there is not one set algorithm for each simulation. This is useful, for example, if one wishes to simulate a network of quadcopters backed up by a series of stationary nodes on the ground. In terms of extensibility, it should also be possible to package, distribute, and import implementations for different communications algorithms.

Starting with the first problem, that is, the ability to specify different routing algorithms for different nodes in the same simulation, it is clear that the specification of the routing protocol should apply to individual nodes rather than to simulations. To this end, it was decided to split the communications functions for each node into a separate communications module. This provides a level of abstraction for messageable programs, as they can then make use of generic send and receive functions at the application level of the OSI model, while the communications module has fine-grain control over the networking layer. In more practical terms, each messageable object must have a communications module associated with it at instantiation, and it is functions of this object that will be called when the unit wishes to send and receive messages. The main benefit of this approach is that it allows flexibility when assigning communications modules - it is possible to either have every node in the simulation use a different type of communications object, or for each node to use an instance of the same object. This provides a very obvious standard for reusing communications code between simulations.

When considering the second problem mentioned above regarding the packaging, distribution, and importing of routing functions, it was determined that this could be best achieved by combining collections of implementations into libraries and allowing the user to make use of these libraries as appropriate when writing a drone program. This makes it easy to mix and match different algorithms from different sources, and means that in order to use a particular approach, one need only link a program against the appropriate communications library. It also allows different contributors to use identical or overlapping namespaces.

### Sending and Receiving Messages

It is necessary at this stage to determine exactly what responsibilities a communications module has, as well as the interface it should expose to its messageable. The most obvious function required of a component designed to communicate is to send information. To this end the communication module must be able to take a message and broadcast it to other nodes in the network. Exactly how this is achieved is dependant on the routing algorithm involved, but it is expected that most implementations will receive a message from the messageable, do some intermediate processing (perhaps to determine the best route to take) and then call the *Environment::broadcast* function to pass the message to the simulated hardware (handled by the simulation environment).

If we are able to send messages via a communications module then it stands to reason that we should also be able to receive messages as well. Thus, there must also be a way for the communications module to transfer messages it has received from the environment to its messageable. This can be achieved by providing a callback function to the environment, which can be invoked when there is a message to deliver. An alternative method which instead provides asynchronous message passing is for each communications module to have a queue of incoming messages for it to process, which it should check at regular intervals. Both of these messages have merit, given that interrupt driven message delivery can become problematic when the network is flooded with messages (no other tasks can be done as the message interrupt is constantly called over routine code), and the asynchronous approach can lead to problems when it is paramount that a message be delivered immediately (such as a command to ground a malfunctioning drone). Given that both of these solution are optimal only some of the time, it was decided that both should be included in the simulator as methods of receiving messages.

**Processing**

The module also needs some way of performing tasks which are not triggered by the arrival of new messages. In order to facilitate these time or state driven processes, communications modules need to have a method which is run independently of the send/receive functions and either loops or is called continuously. It is also clear that one should be able to trigger the pushing out or pulling in of messages from this function, to allow for the use of non-data packets (perhaps, again, to determine the optimum route to the destination). For the purposes of simplicity the simulator calls the processing function once when the simulation is run, leaving the problem of how it is terminated to the communication module implementation.

So, there must be some condition upon which a communications module terminates, lest the simulation run indefinitely. If the threads for a messageable and its communication object were associated with each other then it would be possible to terminate both when the former exits. This can also be achieved by having the program notify the communications module that execution is complete and that it should terminate. The latter option was chosen due to the fact that it simplified the construction of the simulator, and made it easier to modify the relationship between messageable objects and communications modules in the future, for example to change the relationship from one to one to one to many (so that a base station could have separate communication interfaces to other stationary nodes and to aerial units). We devised a de facto standard for this which was to send a message with no addressing information containing only the payload “KILL”. With the aim of flexibility, developers are at liberty to devise other methods for future communication module implementations.

**Defining a Message**

Underpinning all of the above design decisions is the idea of a common specification of a message. With the view of defining this, it would be sensible to expect all messages to be serialisable as text so that they can be safely transmitted across a real world network. Beyond that it is useful to be able to label messages so that perfunctory filtering can be carried out on incoming traffic without the need for in-depth inspection. Routing model will invariably require more than this, as it does not even include space for a payload or addressing (which are both quite useful when delivering messages), but it serves as a definition of the bare minimum which can be expected.

Given the above it must be possible for the base message model to be extended by individual implementations, given that the basic object will remain simple enough that every included feature is required by *all* implementations. It is useful to have an extension of the message class which has a payload, and also one which additionally includes addressing information. These modules serve a dual purpose of being able to test the simulator functions as well as being useful to build upon for more complex communications. It is not required to plan such implementations, as it is expected that they will already have their own specifications (such as AODV in our reference code).

### 5.2.3 Physical Routing

Physical routing in this context is defined as the methods and algorithms applied by the physical entities in the system in order to manage their movement around the environment.

The design of physical routing concerns itself with a number of key problems:

- Understanding the environment.
- Finding routes between locations.
- Avoiding Obstacles.
- Avoiding other agents in the environment.

In the case of this project, the agents are the drones. The nature of the environment is unknown, although due to the scope of the project and the available hardware, it will be assumed that the environment's area is open. A discussion on this problem and possible extensions will be discussed in a later section.

Due to the autonomous nature of the system, each drone must be responsible for its own routing. It might be thought that a way to greatly simplify the problem would be to give the task of routing the drones to the base station with it keeping track of the location of every drone and its environment. However, there are a number of problems to this approach.

Firstly, if all the responsibilities for routing were on one agent, then it would introduce a critical weakness into the system by creating a single point of failure. Should the base station terminate unexpectedly or behave erratically or erroneously, then the entire system would be compromised, potentially leading to disastrous results.

Secondly, allowing the base station to perform all the routing algorithms is making the assumption that it will be in contact with every drone at all times. As is covered in the project specification, it may not be the case that the drones are all within communication distance of the base station. This would, of course, mean that any drone that left the area where it could receive messages from the base station, it would be unable to act. To counteract this, fallback algorithms could be run on the drone, but at that point, the drone may as well run the algorithms in the first place.

### Environment Representation

The first and most fundamental problem associated with routing around a physical environment is being able to represent, understand, and act on the information about that environment.

The environment, its boundaries, and locations in it will be referred to using a normal 3-dimensional Cartesian coordinate system. The exact size of a 'unit' in this coordinate system will be left intentionally undefined. This is so as to leave the scale of the implementation up to the application of the system. A mapping would be provided to convert from whatever real measurements of the environment the drone is moving through, to the x, y and z coordinates of the virtual representation. As an example, one possible scale and mapping would be in the California forest fires problem. For these drones, a GPS could be used with a mapping between the latitude and longitude provided by the GPS and the x and y coordinates of the Cartesian frame.

### Pathfinding

As essential part of physical routing is getting from point A to point B. Given the representation of the environment given above, then this provides a number of options as to how the pathfinding could be done. In most cases, little to no complex pathfinding is required, given that the environemnt is open. In this case, it is only the other drones that would need to be avoided, and this will be discussed below in section 5.2.3.

Assuming obstacles, if there are any, are stored as points in the Cartesian space, then the drone must route around them. This could be done marking an area around the obstacle as impassable, or by constructing a Voronoi map of some other variation of pathing map in order to define the areas the drone can travel through. Given the complexity of creating a Voronoi map and updating it in real time as new obstacles might be detected, simply marking areas around obstacles are restricted would appear to be the best solution.

With this, the environment's 'map' can be viewed as a 3-dimensional grid. Using this grid, and any grid areas blocked by obstacles, an A* pathfinding algorithm can be written to navigate around the environment. Given the worst case performance of A* is O(|E|) where |E| is the number of edges or connections in the map and |E| for a 3-dimensional Cartesian map is $n \times 26$, the algorithm is relatively efficient.

For most cases, however, 3 dimensions will not be needed. In most environments, if an area is blocked, it will be blocked for the entire height of the environment as obstacles such as trees, buildings and so on. Due to this, the pathfinding problem can be simplifed to only require 2-dimensions of pathfinding, with the drones adjusting their height as required once the destination has been reached. This means the worst case performance of the A* algorithm becomes $n \times 8$ for 8-way connectivity in a 2-dimensional Cartesian grid.

### Collision Avoidance

The main routing problem that is present in the specification is dealing with other drones moving about the environment. Pathfinding around immobile objects is relatively simple, as explained above in section 5.2.3. Conversely, ensuring that no collisions occur between the drones requires real-time updates on the locations of every agent in the system.

Each drone must be responsible for its own collision avoidance around it. Given the drones have no way of easily detecting the presence of other drones, then the drones must continuously broadcast their positions so that any nearby drone can react accordingly. To enable this, each drone will broadcast its location to all other drones in set intervals.

The first, and easiest check to make, is to test if the drones are within range of each other that they could feasibly collide. Conveniently, the limited range of the communication means any drone outside that range is automatically not checked for potential collisions. It is worth noting that because of this, a problem can arise if the drones can travel more distance than the range of communication in the time between location broadcasts. This can be solved by ensuring that the location broadcast is suitable small.

Each drone has a maximum speed it can travel, which is referring to the distance it is allowed to move in a certain time which, again, is mapped to real-world distances. If a message is sent every time step $t$, then the maximum distance a drone can move in that time is $t \times s_m$ where $s_m$ is the maximum speed of the drone. Therefore, a potential collision could occur if the drones are within $(t \times s_m) * 2$ distance of each other in the worst case scenario that both drones are flying towards each other at maximum speed. Using this, an initial check can be done that checks if the drone's are within this distance from each other. If they are not, then no more calculations are required to be done. This will greatly speed up the process for collision avoidance.

In the case where two drones have detected that their paths cross, they must either move to avoid each other, or one must stop. Moving to avoid each other at the same height adds a great deal of extra complication, as well as more computation time on drones' likely limited processors. Because of this, one of the drones will fly over the other.

This then presents the problem of deciding which drone should fly higher. This must be done deterministically by both drones, otherwise a collision may occur anyway. In order to solve this, a priority will be given to each drone. The drone with the higher priority will continue on its route, while the other immediately flies higher and over the top. This should mean that each drone will know whether it will be needing to change z-level without the need to communicate and agree on it each time, hence reducing wait times at potential collisions.

### 5.2.4 Physical Deployment

One thing that was established early on in the design process for the physical deployment was that each node in a deployed network should have its own simulation (Environment) thread running. This was needed so that calls to basic simulator functions could be intercepted, allowing for a seamless transition from simulated hardware to real hardware. Not structuring the physical deployment in this way would have made it incompatible with the base simulator, meaning that programs would have to be rewritten in order to use it. While not explicitly forbidden by the specification, it is clear that this situation should be avoided if at all possible - thus the choice was made to run a simulation environment on each node in a deployment. The Parrot AR 2 extension that was also designed as a case study for the physical deployment was designed with the aim of being a reference for future such extensions. To this end it was important that the different steps required to deploy octoDrone to a piece of hardware were clearly defined and delimited. A consultation of the specification suggested that the process was best divided into the following steps:

- Developing hardware specific definitions of Drone and Environment functions
- Developing an intermediate layer which translates commands between the simulator extension and actual hardware
- Ensuring that the Environment correctly starts and stops any extra processes that are required to fulfil the above steps

Focussing on the first point above, there are a number of functions in the *Environment* and *Drone* classes which must be redefined in order to send commands to hardware instead of other parts of the simulator. In *Environment* these are *Environment* (the constructors), *Broadcast*, and *Run*. In *Drone* these are *Drone, Upkeep, Kill, Move* and *Turn*.

#### Environment::Broadcast

This must be modified to send messages to other nodes in the deployment instead of other nodes in the deployment (of which there are none). In the parrot example presented in the project this would entail sending the message to some socket connection (or wrapper on top of one) that would result in the message being sent to all of the other nodes which are in range.

**Drone::Turn and Drone::Move**

Turning and moving is necessarily different when considering a physical deployment. In order to actuate movement commands need to be sent wirelessly to the quadcopter. Whether this is done directly or through a piece of middleware will depend upon the specifics of the platform being targeted, but the calls will originate from these functions. These functions also need to calculate how long the movement will take, either by querying GPS information or by using pre-existing knowledge about the drones hardware.

**Environment::Environment**

The constructor for Environment must perform any required instantiation tasks, such as starting the unit listening for incoming communications. In the sample deployment this is starting a thread to listen to the socket that the broadcast function is sending to, as well as starting a thread to communicate with the drone hardware for movement and turning.

**Drone::Upkeep**

On the subject of movement and turning, the upkeep function is normally used to update the position of the drone in the simulated environment and mark it as having stopped moving when it reaches its destination. While the actuators on a node will take care of actually moving, the hardware must still be told when to stop (and to mark it as stopped in software). This requires communication with the thread set up to communicate (described above).

**Drone::Drone and Drone::Kill**

Because there is no way in the simulator to instruct drones to take off or land (stemming from the fact that there is no concept of a drone being airborne or grounded) this must be handled by the drone constructor and kill functions. The Drone method should instruct the physical quadcopter to take off, and given that Kill is usually used to mark a drone as having finished execution for the run (either through collision, to represent a fault, or just because the program has finished running) it is the method that should be responsible for making the drone land.

**Environment::Run**

The final method that must be redefined for a physical deployment is the Run function of the simulator environment. As part of the teardown performed when a simulation finishes, any extra threads that were created during the running of the program must be stopped or waited for.

CHAPTER 6

# Simulation Software

## 6.1 The Original Choice of ns3

As the premier free and open source network simulation tool, ns3 seemed like the perfect choice for the project. The vast body of functionality that ns3 has built up over its lifetime means that most of the code that would be required could be reused, both shortening development time and reducing implementation errors.

Compared to its competitors, ns3 had the richest feature set, as well as the best API documentation. For this reason it appeared to be the easiest product to use.

## 6.2 Why ns3 was Dropped in Favour of a Bespoke Solution

Unfortunately, despite the promising findings of our preliminary research, it became apparent when ns3 was used to inform our initial design work that it would not be suitable for the project. The sheer amount of features in ns3 lead to a level of complexity which made it inaccessible. Whilst individual features were well documented and easy to use, amalgamations of

After some time spent struggling to actuate what were fairly simple tasks in the grand scheme of the project, it became clear that an alternative would need to be found. Given that even the masters level module at Warwick stops short of requiring students to perform more than simple tasks, the project would need something more accessible in order to achieve the requirement of being easy to use.

The research presented in section 4.2 shows the other network simulators that were investigated. Eventually, it was determined that there was no existing product which suited our needs to a high enough degree, and that creating our own simulator was the best option available. Of course, such a decision brought its own set of problems - primarily that a number of different algorithms and protocols would need to be implemented specifically for the application that was created (instead of being reused). Ultimately these drawbacks were outweighed by the ease of use that a domain specific application would bring.

Finally, a key advantage of creating a new simulator was that it would allow for easier deployment of programs to actual hardware. Given the breadth of drone models and software available, adapting an existing solution to be generic enough to interface with even a small subset of hardware available would have been a gargantuan task.

*This section will focus on an analysis of the implementation of the simulator, more specifically on how each element of the simulator interacts and the general structure of the simulator itself*

## 6.3 Code Structure and Development Methodology

The simulator was created to be as modular as possible with each subsystem having very clearly defined links to the other parts of the simulator. (for instance the methods of “commMod" that interact with environment) Using this design allows for far easier modification and testing of each part of the system, as each part presents a “contract" with the other pieces; so long as that “contract" remains unchanged, the other parts of the system should be able to continue working identically even if the other components are changed. This also has benefits to possible optimisations of the system as any optimisations, will be trivially intergratable so long as they do not violate this “contract".

The simulator is extensible. This is actually the key to the way that the simulator works: the point of the simulator is that users extend the classes provided to use the simulator. This also allows users to extend the simulator in order to alter the way that the simulator works. As it is, the simulator is devoid of any non-essential features, but this is by design, as the main problem identified with NS3, was feature bloat, causing the entire system to be unfocussed and thus more difficult to develop for than a more focussed system.

Despite seemingly contradicting the first point of this section, the simulator is also interconnecting, every system is related in some way to the other systems, Drone works very similarly to Base Station (which is to be expected, they both extend Messageable), and the way that broadcasts work is very similar to the ability to push visualisation elements.

Ultimately, the primary implementation decision was to use multiple threads, which was to ensure that the system could emulate many independent machines at once (the primary purpose of the system). Unfortunately, this does lead to inefficiencies on machines with a lower number of available threads as the overhead in switching to and from virtual threads of execution is expensive computationally.

### 6.3.1 The Environment

The environment code is the central hub of the simulator, handling all of the administration of tasks being performed by the system. The environment contains all of the sample sensor data, and contains references to all of the messageable components of the simulation.

The environment's thread is what can be seen as the “main" thread of the system, this is the thread that handles the movement requirements of the drones as well as keeping track of the system's internal representation of time (which is independent of real time in that the internal time is a multiplier specified by the user of the total number of movement ticks that have elapsed since the beginning of the simulation).

Another responsibility of the environment code is to handle the broadcasting of messages to the drones, in the implementation this is achieved relatively inefficiently, using a simple collision detection method to check whether every known messageable is in range or not, this could be a target for optimisation for further development of the simulator, as will be outlined in the optimisation section below.

### 6.3.2 Communication Modules

Communications modules act as intermediaries between the messageables they are attached to and the environment of the simulator. In actual deployment, the communications modules can be seen as a system handling lower-level network tasks required by the messageable. The communications modules are implemented in such a way as to reduce the workload of a developer creating messageable code, allowing for separation of labour into networking development and the actual programs to be executed on the messageables.

Because of this, the communications modules primarily interface with the environment, passing messages to attached messageables from the environment and messages from the messageable to the environment.

In terms of the actual functionality built in to the “CommMod" class that represents the communications modules, it does little but pass messages and call the environment's “broadcast" method. This is because the communications modules are not defined by the simulator but rather the user, allowing for any user defined communication protocols to be used.

The job of each communication module's thread is to run the supplied function until it terminates.

### 6.3.3 Messageables

“Messageable" is the superclass of all classes designed to be a destination for messages, in this case drones, base stations, and all of their subclasses created by the user. The primary function of the code in the “Messageable" class is to contain the virtual methods to be overwritten by the user, as well as some utility functions to facilitate communication between the running function of the messageable and its communication module.

Each messageable's thread runs the user's overwritten function in the final subclass of the messageable.

#### Drone

Drone is a subclass of Messageable. The changes made to messageable with drone are simple, introducing new functions to facilitate the movement of the drone, as well as various utility functions to query the status of the drone, most of these functions are never called in the simulator itself, but rather are designed to be called by user defined subclasses of drone in the override of the “run" function.

Functions like “move" do not actually cause the drone to move, but rather queue a movement for the drone that is then executed by the environment thread when the drone's “upkeep" function is called.

#### Base Station

The “Base Station" class is very simple as, aside from a constructor change, it is just a type transformation over Messageable. This is mainly to simplify the experience for the end user as extending messageable to create a base station would be possible, but would have been confusing.

### 6.3.4 Visualisation

The visualisation system provided a large set of implementation challenges. Because of the multi-threaded nature of the simulator, different threads can push elements to be visualised at any point. These element pushes are mainly handled through several helper functions that each call an underlying element pushing function with preset arguments. This simplifies both the implementation and use of these functions. The issue with this method is that the element list can be polled to change at any time, even when the visualiser is in the middle of a drawing operation. Due to the way C++ iterators work, and the fact the list has elements removed periodically (due to the fact that these elements are removed when they have exceeded their “lifespan"), this could cause iterators previously pointing to valid elements to become invalid.

For example, the element list consists of four elements, a base station, two drones and a broadcast. One of the drones is deactivated and thus removed from the element list, however the draw thread was drawing the broadcast at that exact moment. Because an element before the currently active element was deleted, the pointer to the active element becomes invalid. This means that the pointer can no longer produce a valid element, causing a segfault as unassigned memory is accessed.

The solution to this problem was to lock both the “step" function and the “draw" function behind the same mutex lock. Effectively, this makes the two functions mutually exclusive, allowing only one of the two functions to be running at any given time. This causes any changes to the list of elements to be done only when there are not any active pointers to the list.

The window management system used in visualisation was GLFW3, a platform independent system chosen due to its flexible nature and the fact that it can handle multi-threaded applications more easily than GLUT.

## 6.4 Implementing Communications Modules

Communications modules are responsible for performing tasks at level three of the OSI model[32] (the networking layer). Depending on the noise function used by the simulator, it is also possible to have communications modules perform tasks in layer two (such as collision detection and avoidance). Whilst the base class is a part of the simulator itself, the actual implementation of routing algorithms are part of a separate library in their own right. In this way, the communications library is intended to be distributed with **octoDrone** without being required as a constituent part.

### 6.4.1 Structure of an Individual Module

#### Sending and Receiving Messages

When a communications module wishes to transmit a message to other units it invokes the broadcast method exposed by the simulation environment. Messages that originate from the communication modules messageable collect in a queue, which must be checked at regular intervals. When there is a message that the environment determines should be delivered to the communications module, it is deposited in a similar queue. When the module has a message it needs to pass to its messageable, it stores it in a third queue.

In addition to asynchronous messaging, the project design also called for synchronous message passing. This is achieved by having a callback function in the messagable which can be invoked upon receipt of an urgent communication. It is left to the communication implementation to decide which messages are important to interrupt the normal operation of the messageable for.

**Intermediate Processing**

The virtual function *comm_function* is called when the communications module is started and is expected to return when the associated program is ready to terminate. It should be used to perform any intermediate and non-delivery driven processing such as routing or time based commands.

### 6.4.2 Structure of a Collection of Modules

In order to facilitate easy distribution and use, collections of communication implementations are combined into libraries. Given the small size of a communications modules source code (often tens of kilobytes) it might be tempting to statically link them to simulation executables. The problem with this approach, however, is that it precludes the very thing that prompted the creation of communications modules in the first place - modularity. As such, communications code is packaged into a shared library which simulator executables are then linked against dynamically at compile time. This reduces the size of produced executables and makes updating communications code possible without requiring simulations to be recompiled.

### 6.4.3 Access to Other Simulation Elements

The architecture of the simulator requires that communications modules are able to interact with the simulator environment (to dispatch messages), as well as with the messageable with which they are associated. Since it is impossible to have all of this information available when the communications module is instantiated (creating a messageable requires a reference to a communications module as well), it is necessary to set the messageable associated with a communications object at a later time (but before the simulation is started). Thus, the procedure for bootstrapping a simulation is broadly as follows:

1. Load the sensor data
2. Create a simulation environment referencing the sensor data
3. Create a communications module referencing the environment

4. Create a messageable referencing the environment and the communications module

5. Add a reference to the messageable to the communications module

How these references are used is covered in section 6.4.4.

### 6.4.4 Integration with the Simulator and other Programs

The communications module class makes use of the following functions from the simulator:

- *broadcast*

Programs for drones and base stations can make use of the following functions from *CommMod*:

- *push_in_message*
- *push_out_message*
- *comm_function* (to start the CommMod)
- *set_messageable* (used when defining simulations)

### 6.4.5 Provided Implementations

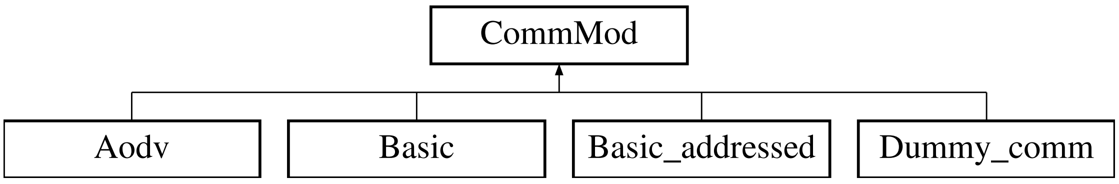


Figure 6.1: Inheritance diagram for the *CommMod* class

### Basic Messaging

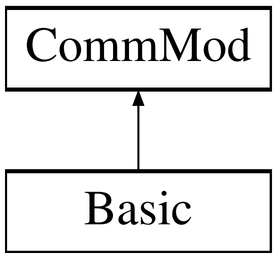


Figure 6.2: Inheritance diagram for the *Basic* class

The basic messaging implementation provides the simplest possible way of sending and receiving messages. This can be useful when creating programs for messageable units, as it removes the need to consider routing problems, units being out of range, and a whole host of other potential issues. Additionally, it provides a good method for simulating existing programs which were not written with **octoDrone** in mind and may perform their own addressing. In this way it contributes to octoDrones ability to be generic and flexible.

In this implementation, all messages that are sent are received by all nodes, regardless of range. Messages sent include payload information, but do not store information on the sender or receiver.

### Addressed Basic Messaging

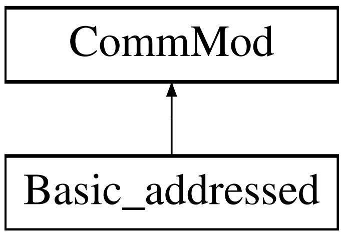


Figure 6.3: Inheritance diagram for the *Basic_addressed* class

The addressed basic messaging implementation extends the above algorithm to additionally take into account the sender and intended recipient of a message when choosing which packets to deliver and which to drop. It is intended to be used for passing messages between programs and communication modules in a way which prevents the need to include code to serialise and deserialise addresses. If appropriate, it can also be used with external programs as described above.

In this implementation, all messages that are sent to an IP address are received by all nodes with that IP address, regardless of range. Messages include payload information as well as the IP addresses of the sender and intended recipient. Messages sent to the broadcast address 255.255.255.0 will be received by all nodes in the simulation.

**Ad hoc On-Demand Distance Vector Routing**

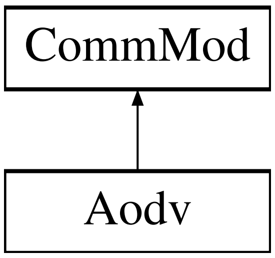


Figure 6.4: Inheritance diagram for the *Aodv* class

The Ad hoc On-Demand Distance Vector Routing (AODV) implementation provided in the communications library gives an example of how a more sophisticated routing algorithm can be used within the simulator framework. AODV is designed to be used in mobile ad hoc networks and offers loop free, just-in-time routing that is fault tolerant and capable of reacting to the movement of nodes.

**Overview**

In order to explain what is happening under the hood of the AODV implementation provided with **octoDrone** it will be useful to first cover the basics of how the algorithm operates. A node A which wishes to send a data packet to node Z transmits a Route Request (RREQ) to its neighbours, which serves the dual purpose of notifying it of the next hop in the most efficient path to Z, but also of notifying the routes along the way where they should direct messages addressed to Z[38]. If a unit receives a route request for which it does not have a valid route it broadcasts the route request to each of its neighbours in order to propagate the route discovery.

When the RREQ is received by a node which has a route to Z, either because (as in this case) it *is* node Z or because it is already part of a route to Z, that node creates and returns a Route Reply (RREP) packet. This packet is used by every node on the reverse route to store the next hop to the destination, Z, and is forwarded to the nodes that sent the RREQ. Incremented sequence numbers are used to make sure that information remains fresh as for loop prevention.

Once the sending node, A, receives an RREP it can send out its data packets via the next hop on the route that was just discovered. This route will be cached for a period of time before a new discovery period must take place. If at any time there is an error during transmission, the node encountering the issue sends out a Route Error (RERR) packet to notify other nodes that cached routes using this hop should be invalidated.

**The Routing Table** The AODV implementation has a special data structure to represent entries in an AODV routing table. As per the RFC (documents by the Internet Engineering Task Force containing technical and organizational notes about topics such as internet protocols), this contains the sequence number of the destination node, the time after which the route is no longer considered active, the hop count to the destination node, and the next hop on the route to the destination node. Each AODV communications module keeps a map of destination IP addresses to route objects, which is known as its routing table.

### Message Types

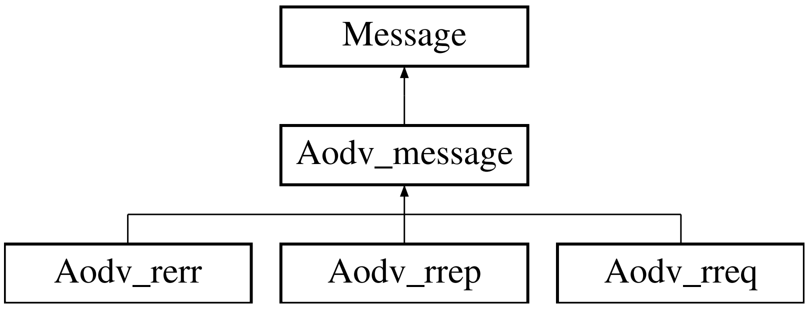


Figure 6.5: Inheritance diagram for the *Aodv_message* class

In order to easily handle the different types of messages handled by AODV (RREQ, RREP, etc.), the implementation defines a basic AODV specific message class which contains the fields common to all AODV messages. This is then extended by individual classes in order to completely define which fields are required by each message type. This is shown in figure 6.5.

**Sending Messages** With the above classes in mind, we turn to the question of how octoDrone goes about sending messages. Initially, messages which have been deposited into the outbound queue must be deserialised into a structure containing the message payload, the destination address of the message, and the source address of the message (in some situations this may be different from the address of the communications module). This happens because messages are always sent to and received from the environment in string form. While it may at first seem inconvenient, it is necessary in order to preserve the integrity of messages if they are sent over a network (which is the case if we are running a deployment on real hardware). After this the routing table is queried to find out if the node already has an active route to the destination. If it does, then a data packet is constructed immediately and dispatched via the next hop on the route to the destination.

If there does not exist a route in the cache, things become a little more complicated. Because the route discovery process requires that we send and receive a number of different packets before sending the message is considered done, we need to keep track of internal state. In octoDrone's implementation of AODV, this is done by keeping track of the current message we are sending, as well as the stage in the route discovery process we are currently at. This means that whenever something is triggered by a message being received, it can access this information (and update it if required), even when the original message object has long since gone out of scope. After the state information is created, a helper function is used to create the RREQ which will initiate the AODV protocol on other nodes. This is then sent using a call to the simulation environment. At this point there is nothing else to do but wait for messages to be received.

**Receiving Messages** When a message is received, the communications module unwraps just enough of it to determine the type of message that has been delivered, which will be one of the AODV types, such as RREQ (this type is internal to AODV, and should not be confused with the type property of the base message class which is used to denote emergency packets). This information is then used to select an appropriate deserialisation function to apply to the message in order to transform it from a string into an AODV data type. Each of the types has its own processing method, which determines what action (if any) to take given the message contents and the internal state of the communications module.

For RREQ messages, the node checks to see if the message was a hello message (described in section 6.4.6), and if so we determines if there is an active route to the sender (who will be one of its neighbours), and add a new route if it does not. A reply is sent to the sending node so that it can compile route information about other units in the network. If it is not the case that the message is a hello message, then it must be route discovery from another node. If the communications module has an active route to the destination mentioned in the RREQ, then a helper method is used to generate an RREQ packet to reply with. This is sent via a call to the simulation environment. Otherwise, the node's internal state is updated to record that it is participating in remote initiated route discovery, and the RREQ packet is forwarded after decrementing the time to live (TTL) field and incrementing the hop count. In order to keep the implementation small and simple, nodes do not respond to RREQ requests when they are part way through route discovery for themselves or another node.

When an RREP is received, the communication module first checks to see if the route it has to the sender (previous hop) is up to date, modifying the routing table if appropriate. Afterwards, it checks to see if the route was for itself (to send a data packet) or for another node (remote initiated route discovery). Multiple RREP packets for a single route indicate that there exists more than one possible path. In this case, the route with the lowest hop count is chosen (the reply window is defined by the path discovery time variable which is set at instantiation). If the former, then a data packet is constructed using the new routing information, and sent via a call to the simulation environment. Otherwise, the information is passed on (with a decremented TTL). Keeping track of hop counts is done by the helper function.

Upon receipt of an RERR packet, the communication module marks the route as invalid (achieved here by setting the active route timeout to zero), before broadcasting this information so that affected nodes can update their local routing tables.

**Broken Links** But where do these RERR packets come from? It may be that a node receives a message for which it does not have a route. It may also occur that a node loses connectivity with one of its neighbours. If either of these scenarios come to pass then the receiving node will broadcast an RERR message to the node which precedes it in the route. Given that the octoDrone implementation of AODV only stores forward routes for simplicity, all RERR messages are sent upon message receipt instead of also being sent upon link failure. This makes sense: because octoDrone does not emulate the lower layers of the OSI model, nodes do not have access to information about link statuses. One way of mitigating this is to use hello packets as described below in section 6.4.6. When a node fails to receive a hello packet for a neighbour which is listed as the next hop for one of the route in its routing table (this includes neighbours which are the next hop and destination for a route), it will mark that route as invalid and propagate an RERR the next time that route is used.

### 6.4.6 Extension: Hello Messages

The RFC for AODV also mentions the use of hello messages as a means of broadcasting connectivity information. While the document suggests that nodes should only broadcast hello messages if they are part of an active route, to account for the differences mentioned above, all nodes using the octoDrone implementation of AODV will send hello packets on a regular basis.

## 6.5 Summary

The above sections have shown how the designs for octoDrone communications have been implemented, covering the structure of communication modules, message types, and how these components integraye with the rest of the software. In the next chapter these concepts will be taken one step further with the jump to real hardware instead of simulated hardware as was discussed here.

## 6.6 Potential Optimisations

This section will highlight several areas for optimisation, how they are inefficient and why they were implemented this way. The first and most obvious area for optimisation is in the environment code, specifically the way that broadcasting works. As the code currently exists, when a broadcast occurs, the entire list of messageables is tested, each one being checked whether it is within the range of the broadcast. If it is then the message is sent to that messageable. A potential optimisation (especially for broadcast-heavy programs running on the simulator) would be to limit the messageables that receive the message to those in the general area of the broadcast perhaps through some kind of segmentation of the environment space in to a three dimensional grid which contain a subset of all elements.

Another area to be optimised could be the main threads calling of the upkeep functions on the drones The way it works in the provided implementation is that environment calls upkeep one drone at a time. This works fine, but there are no dependencies between each drone's upkeep method, so parallelising this process could improve performance, it does however cause the creation of even more threads (which this system already creates a lot of), so this optimisation could only be valid for systems that can afford to have a large number of threads running at once.

These are the obvious algorithmic optimisations that could be done, also to be considered could be reducing the amount of threads that the system creates in order to improve performance on machines with fewer hardware threads (as switching the thread of execution can be expensive).

## 6.7 Review Against Original Objectives

The only objective that pertains to this section is "Implement a network simulator" which requires analysis as to whether the implemented system fulfills this objective. The objective requires a system that is capable of supporting the creation of a network of drones which are capable of sending messages and reacting to messages they recieve. Due to the way that the system works, this requirement is fulfiled, arbirtary drone programs can be implemented through user defined Drone, CommMod, and BaseStation subclasses.

However, this objective also requires “Specifically tailored to our domain". This is where the analysis is required as the domain in which the project lies is flexible. In order to fulfil this objective the system would have to enable the creation of drone programs that can perform arbitrary tasks. The system fulfils this requrement too: the system was crafted specifically with this in mind, the simulator specifies very little, causing no bottlenecks or restrictions (with the exception of requiring exactly one base station).

## 6.8 User Manual

In order to use the simulator, users must extend BaseStation once implementing the run and message_callback methods and pass it to an instance of Environment created with the sample data for the simulation. From here, the user must implement extensions to Drone and CommMod, one for each type of drone and communications module required. Many projects will require only one of each of these, but it is possible that a project with multiple types of drones could be created. Instances of the created drones should be passed to the environment instance. In the run method for any Drone or BaseStation derived class, a user can use the “send_message" method in order to send a message to the communications module for broadcasting, the “wait_for_message" method in order to wait for the next incoming message (implementing the “message_callback" function allows for non-blocking communication), and use the “get_time" and “get_position" methods to get the time and position respectively. Drone adds the functions “move", “turn", “getMaxSpeed", “getSpeed", “getAngle" and “hasFinishedMoving" which return their respective values or perform their respective tasks. The function “sense" allows the drone to gather data of the supplied type, emulating the use of different sensors with different strings supplied to the function.

The “CommMod" extensions allow the use of the “broadcast" method as well as the “pass_message" method, with these sending messages into the environment and to the attached messageable respectively. The function of “comm_function" which should be extended by every extension of CommMod is the actual communication module code.

The purpose of the simulator is to simulate the physical deployment of code on drones. The idea is that with a properly configured drone or base station, a user can take the code in the simulator and run it on physical drones and base stations with, no work to port the code to the new machine. See the physical deployment section to see an example of this.

CHAPTER 7

# Physical Deployment

*This chapter presents the reference deployment which was carried out as part of the project. It describes the high level decisions and simulator changes which need to be made to create similar deplyoments on arbitrary hardware, as well as the specific choices which were made for the Parrot quadcopters used in the project.*

## 7.1 Objectives

The main objective of this part of the project was to enable the running of octoDrone simulations on real hardware. The move from simulated programs to real ones may seem odd when the overall goal of octoDrone was to allow for the simulation of real programs on virtual hardware. In a sense, this a decision that logically follows the move from real to simulated - once one has built a program within the simulator framework and proved that it operates as one expects, it is infuriating to then have to implement this using a different set of frameworks on real hardware. It necessitates rewriting the same software in a way which effectively prohibits simulated benchmarks: there is no guarantee that simulated programs would exhibit the same performance characteristics as their real world counterparts given that they may be written on top of different libraries (and possibly even in different languages). To this end it makes sense that one should want to run the same program in the simulator as in a real deployment - this follows the driving philosophy of octoDrone.

There were a number of options that were explored in terms of the actual components that were used for our example deployment, similarly with the ways in which we adapted the simulator software to make the process as seamless as possible. The overall goal was to obtain a minimal working example, given it was unknown how well the hardware we could acquire would interact with what we had made. It is important to distinguish our efforts to make the simulator operate on real hardware units and the actual deployment we carried out as an example. The former is a part of the product that is octoDrone, and was done to a high, professional standard. The latter was implemented using the components which were available to us through the department and, as such, does not represent what would be expected of a deployment carried out in industry. With that said, it is a testament to the flexibility of the simulator that it is possible to coerce it to run on distributed hardware components that it was never intended to support.

## 7.2 Equipment and Feasibility

The first decision to make with regards to the example deployment was to determine the type of hardware we would be targeting. There were two drones made available by the Department of Computer Science for our use - a pair of Parrot AR 2.0 Power Edition units.

**Parrot AR 2.0 specifications**[47]

- 1GHz 32 bit ARM Cortex A8 processor with 800MHz video DSP TMS320DMC64x
- Linux kernel version 2.6.32
- 1Gbit DDR2 RAM at 200MHz
- Wi-Fi b,g,n
- 3 axis gyroscope, accelerometer, and magnetometer
- Ultrasound sensors for ground altitude measurement
- 4 brushless inrunner motors. 14.5W 28,500 RPM
- 30fps horizontal HD Camera (720p)
- 60 fps vertical QVGA camera for ground speed measurement
- Total weight 380g with outdoor hull, 420g with indoor hull

### 7.2.1 Micro-controllers

The notable thing missing from the above specifications is the ability to program the quadcopter with custom code. In a commercial or industrial deployment of this type, one would expect to use drones that are capable of being programmed themselves. Unfortunately these models are rather expensive, and as such we had to find a way to make the drones autonomous without being able to change the software running on them. The obvious solution was to use a micro-controller to pilot the drone because it would be easy to run our own custom software and communicate with the drone via its WiFi network. Below is a short summary of the investigation we did into the two major micro-controllers aimed at the consumer market (and thus within the limits of what we can acquire through the Department of Computer Science).

#### Arduino

Arduino is an open-source prototyping platform with hardware and software that is designed to make it easy and simple to create and prototype electronics projects[2]. The fact that the entirety of the Arduino ecosystem is open source (including the hardware) is a large benefit given the open source nature of octoDrone. The problem with the Arduino, however, lies in the specifications of the products available. Even the Arduino "Mega" tops out with a 16MHz processor, which makes it unsuitable for use in a project which needs to be running multiple threads and potentially various interpreters.

#### Raspberry Pi

Similar to the Arduino, the Raspberry Pi is a low cost system on a chip designed for hobbyists and students. It overcomes the performance limitations experienced by its smaller rival with ARM CPUs ranging from 1x700MHz to 4x1200MHz[16], which are sufficient for the needs of octoDrone. The main drawbacks to the Raspberry Pi are its heightened power consumption (a necessary result of higher specifications) and the propriety nature of its hardware. After weighing up the pros and cons, the team decided that the Raspberry Pi was the better fit for the project, and as such, it was chosen for the sample deployment.

### 7.2.2 Inter-node Communication

In addition to communicating with the drone they were paired with, nodes would need to communicate with each other in order to pass messages and communicate with the base station. There were a number of options here which varied from true to life to more synthetic conditions.

#### Radio

In a real scenario, all communication done between nodes would be done using uni or bi-directional radio links. This allows for the use of software and protocols aimed at mobile sensor networks to facilitate the saving of power. It is obviously beneficial to create an environment which is as close as possible to real conditions, but we felt that this was less applicable for our example deployment. When we considered radio in the context of octoDrone, the abstraction of the physical and data link layers for communications meant that programs would not be able to tap into the extra power afforded by using it. In addition to this, many commercially available radio modules are designed to be used with the Arduino, which we had elected not to use.

#### WiFi

The second option we investigated was IEEE 802.11. The advantages over radio links were clear - it was easier to set up and use in in terms of program code, with many common protocols working automatically thanks to support and drivers for the linux kernel. In addition to this, there are many easily available USB adapters that work out of the box with the Raspberry Pi and Raspbian (a spin off of the Debian Linux distribution). This comes at the cost of the ability to make as many decisions at the lower OSI levels, but on balance this would make the process of developing and testing the deployment much easier. We concluded at this point that WiFi was preferred as a communication method over radio.

#### Ethernet

At this point the net was cast wider, in an attempt to see if there were any other solutions to the problem that were no immediately obvious. At this point we realised that the assumption that the micro-controllers would have to be mounted on the quadcopters themselves was a false one. Not only would having the Raspberry Pi units situated on the ground make flight more stable by reducing weight, it would also mean that we could use Ethernet to connect nodes together. This also sidestepped the issue we were encountering where the model A Raspberry Pis that the Department had given us only had two USB ports. Using two of these for WiFi adapters would have left no room for a keyboard. While not an insurmountable problem, it would have made debugging in particular more difficult than it needed to be. For the reasons above, as well as the aforementioned lack of necessity for accurate real world deployment conditions (due to the lack of budget) we chose Ethernet as the means of connectivity between nodes.

### 7.2.3 Sensor Data

We also came across the problem of how we would deal with nodes sensing the environment. This largely came down to the decision to install sensors on the quadcopters or to have this data simulated as it would be done in a simulated environment.

**Installing Physical Sensors**

The best way of testing how a real life deployment would perform would be to get real life sensor data from mobile units as a program is running. There were, however, a number of problems with this approach - because it required a processing unit to be mounted on the quadcopter to read the sensor data, choosing it ruled out Ethernet as a communication option. Another problem was that real data values are harder to reason about in terms of software correctness. Especially for a trial deployment such as this one, not being able to repeat experiments (even on the same day) and expect the same results was a real problem. The final issue with this method was that the temperature sensors provided to us by the department came only with a datasheet. A large amount of time would have had to be spent in order to get the sensors soldered correctly and interfacing with the Pi. Even then, anecdotal experience with these devices has shown that they can be finicky and inconsistent, especially given that they were almost certainly not originally intended to work with a Raspberry Pi.

**Synthetic Data**

The alternative to the above was to use the same synthetic data which we had used in the simulated versions of our programs. This had the benefit of keeping some parts of the experiment simple while we measured the effects of other variables on the performance of octoDrone (such as wind speed). As the project matures it will make sense to switch to the use of real data at some future juncture, but that is outside the scope of the project in its current state. As a result we chose to synthesise sensor readings. What this decision did allow us to do was to test a number of edge cases on the real quadcopters which would have only become apparent otherwise after hundreds of hours of testing. This approach of fixing some aspects of the deployment whilst experimenting with others has undoubtedly saved many hours of testing and debugging over the past few months.

## 7.3 Adapting the Simulator Code

With the equipment selection locked in (Parrot AR 2, Raspberry Pi model A, Ethernet connectivity and no sensors), the adaptation of the simulator so that it could run across a number of hosts could begin. Certain pieces of information could immediately be used safely in a distributed application, such as the locations of individual nodes (nodes cannot access the locations of other nodes in the simulation anyway, and there is no need to check positions for collisions as this can safely be delegated to physics).

Other properties, however, generate problems when run as part of a distributed application: the

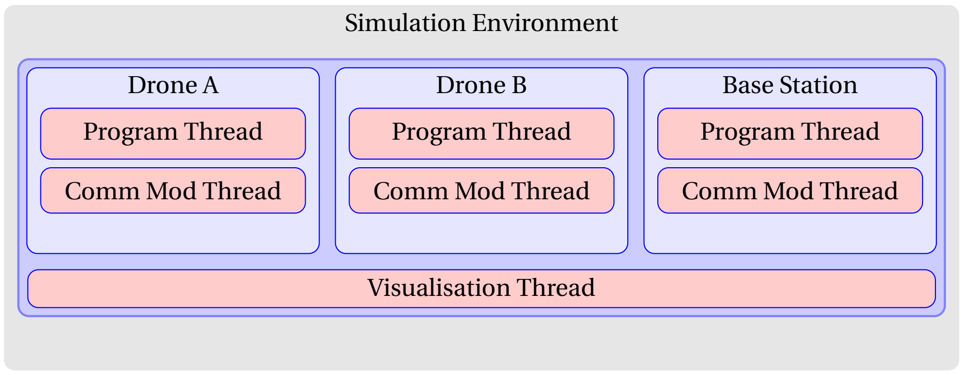


Figure 7.1: Thread diagram for octoDrone simulations

environment iterates over the list of messageables when broadcasting messages. We therefore needed to find a way to disseminate packets without knowing the other nodes present in the network (supplying each node with a list of other nodes and their addresses would be missing the point by a country mile). Movement also becomes problematic when one realises that mobile units will often move at a given speed for a given amount of time, as opposed to the speed/direction instructions that programs use to make calls to Drone::move.

In order to attempt to solve these problems we first had to decide how the simulator could be run on multiple hosts. By far the simplest way to do this was to run individual "simulations" on each physical machine and have these communicate somehow. Users would use the same simulation structure as for conventional use (an environment with drone programs and communications modules), but each environment would contain only one messageable. This meant that each hardware unit could run an individual simulation, and that one could hook in to the functions exposed by the simulator API. As such, each time a a call was made to a function that required a call to hardware (such as sending messages), this could be intercepted and run on hardware without any change to the simulator API with the exception of instantiation, which necessarily required information on network interfaces.

It was useful to visualise how the sharded environment would look - figure 7.1 shows the thread structure of a simulated octoDrone run, which can be compared to the distributed version in figure 7.2. Note that the extra threads present in the distributed version of the software hook into the functions exposed by the simulator (a call to Environment::broadcast actually ends up going to the Comm server thread to be broadcast. This meant that the difference between compiling a simulation and a deployment application is as simple as linking against a different library.

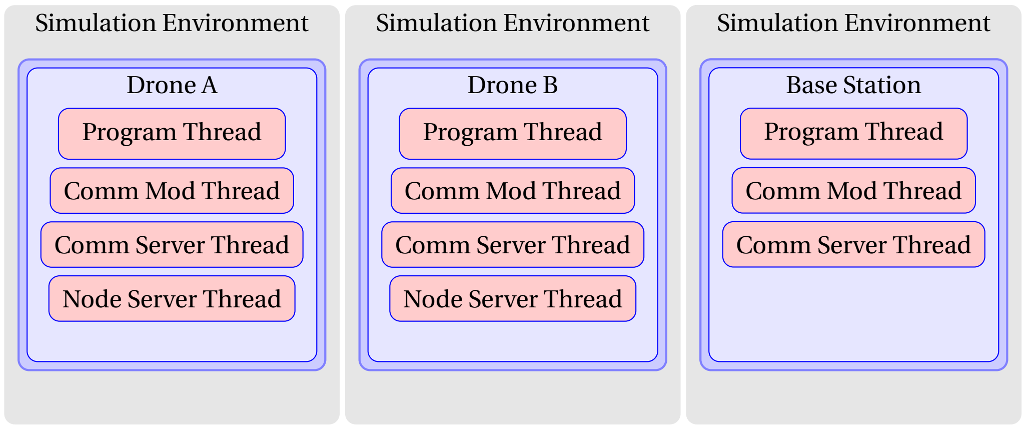


Figure 7.2: Thread diagram for octoDrone deployment

### 7.3.1 Sending and Receiving Messages

Because nodes must communicate with each other over a network (in this case Ethernet), they must be able to actually send packets to each other - a connection must be established between them. This introduces a problem in as much as nodes cannot know the addresses of other nodes in the network (as determined earlier in this section) they must use multicast or broadcast. Broadcasting is effectively deprecated in modern networks, with IPv6 removing it entirely. Add to this the fact that some devices will refuse to let broadcast packets cross the device boundary, it became clear that multicast was the only viable choice. In terms of implementation, raw C sockets were chosen due to their simplicity and speed. It would have been possible to use a wrapper on top of these (such as Boost), but would not have alleviated a significant amount of work during development and would not have made the application cross platform unless the threading library used had been changed from the linux POSIX threads (pthreads) library.

Because the receiving of messages was driven by the sending of messages in the original simulator, these actions had to be decoupled for the distributed version. Because listening for an incoming connection is a blocking action, it would require another thread to be run whilst the remainder of the application was executing. This thread would take a reference to the local messageable on instantiation that would later be used to deliver messages. Because the application would multicast packets, any outgoing messages would have to be sent to a multicast address that was known to all nodes (specified during the compilation of either the environment library or the simulation).

### 7.3.2 Movement

While there are native C bindings for the Parrot AR 2, creating an application with the open source Parrot library is quite an involved procedure. Given that the project does not require a full application for the quadcopters, just a means of making them move, it made sense to use the node js package *ardrone* as middleware. The *ardrone* package handles connecting to a drone on the same network as the device it is run on (parrot drones use a default IP address, making them easy to find), as well as sending packets to the drone which it can understand. We created a javascript server application that could be run alongside the distributed simulator that would take commands and use the package to transmit them to the drone to be carried out. This server was passed messages by the environment through a UNIX socket in the filesystem of the host device. The server program takes high level instructions from the simulator and uses the *ardrone* package to translate them into low level AT commands (see the aside on AT commands for more information on the drone API).

AT Commands

The low level instructions accepted by Parrot drones are known as AT commands. These are sent to the drone on port 5556 via UDP and contain an instruction and a number of arguments[40]. An example at command is as follows:

$AT * PCMD_M AG = 21625, 1, 0, 0, 0, 0, 0, 0 < CR > AT * REF = 21626, 290717696 < CR >$

The list of valid AT commands can be found in table 7.3, and contains three characters ("AT*") followed by a command name, an equals sign, a sequence number, and a list of arguments if appropriate. The sequence number is used to prevent a drone from following out of date instructions: any commands received with a sequence number lower than the highest received sequence number are dropped.

As mentioned above, there was an issue with how movement would be translated from the {speed, distance} format for used by the simulator and the {speed, time} format used by the drones. The first attempt at solving this problem assumed that the application would have easy access to GPS coordinates from the paired drone, and used this to perform location checks in the same way as for simulated runs. It was later learned that there was only one GPS unit available for the two quadcopters obtained, and extracting information from it at runtime was non-trivial. The fallback plan developed involved estimating the time taken for a drone to travel a given distance based on the normalised speed which the *ardrone* library requires. This had to be based on experimental evidence, and was not consistent due to the fact that is was not possible to use a linear model over short distances (see notes about drone movement in movement aside).

| AT command | Argument(s) | Description |
|---|---|---|
| AT*REF | input | Takeoff/Landing/Emergency stop command |
| AT*PCMD | flag, roll, pitch, gaz, yaw | Move the drone |
| AT*PCMD_MAG | flag, roll, pitch, gaz, yaw, psi, psi accuracy | Move the drone (with Absolute Control support) |
| AT*FTRIM | - | Sets the reference for the horizontal plane (must be on ground) |
| AT*CONFIG | key, value | Configuration of the AR.Drone 2.0 |
| AT*CONFIG_IDS | session, user, application ids | Identifiers for AT*CONFIG commands |
| AT*COMWDG | - | Reset the communication watchdog |
| AT*CALIB | device number | Ask the drone to calibrate the magnetometer (must be flying) |

Figure 7.3: Valid AT commands for Parrot quadcopters[40]

## Movement

Quadcopters move by inclining themselves relative to the horizon. This is achieved by having two of the four coplanar rotors spinning clockwise, and two spinning anticlockwise. The differing direction of rotation is to ensure that the torque (turning force) exerted by the spinning of the rotors sums to zero. This is achieved in helicopters by the use of a tail rotor. Quadcopters are underactuated in as much as they have fewer motors than they have degrees of freedom (four rotors, six degrees of freedom), meaning that they cannot follow arbitrary paths in 3D space. Put another way, there are some movements that drones are unable to achieve, such as simultaneously moving and turning, which directly stem from the lack of additional rotors.

Moving forward entails having one of the rotors which is spinning in a certain direction (say clockwise) move faster and the the opposing rotor of the same direction move slower. This has the effect of tilting the drone on the Y axis (see figure 7.4). With the drone thus inclined, the air displaced by the rotors now causes it to move forward. The same technique can be used in reverse to move the drone backwards, and applied to the rotors spinning in the opposite direction (in this case anticlockwise) to bank the drone left or right on the X axis.

This method of movement results in a period of acceleration at the start of movement as the drone inclines to its target tilt. This means that it takes a few seconds to reach a stable speed, unlike with a ground based robot where the period of acceleration is much shorter. Tilting at too steep an angle will cause the drone to flip (or cut out if there is a hardware tilt-meter with firmware support). Stopping follows a similar transition period as the motor speeds are brought back in sync.

Finally, a drone is turned on the Z axis by increasing the rotation speed (and thus torque) of a pair of rotors operating in the same direction. With the torque in one direction greater than the other the drone begins to turn. Altitude is controlled by increasing or decreasing the rotational speed of all motors in unison.

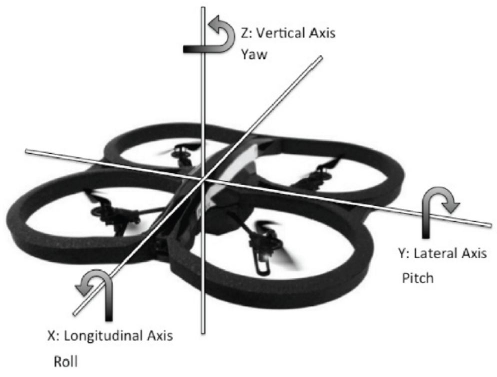


Figure 7.4: Pitch, yaw, and roll in the context of a quadcopter[19]

## 7.4 Results

The reference physical deployment of octoDrone resulted in two programs (one for the base station, and another for the drones) running on three different pieces of hardware simultaneously. All of these units were able to communicate with each other and avoid collisions. The data that was "collected" (synthesised) was relayed to the base station in a timely manner. Finally, both of the drones were able to takeoff, move, and land according to plan, with minimal drift and disturbance from environmental factors.

## 7.5 Review Against Original Objectives

The objective from the specification in section 3.2 was that "The simulation code must be adaptable to the physical drones for real deployment". This was most certainly achieved, and, in fact, exceeded in as much as the work that was done also outlines the procedure for adapting the simulator to arbitrary hardware platforms. It is probable that access to more expensive equipment would have streamlined the deployment process and improved the quality of the end result, but this is of little consequence in terms of a review against the specification.

CHAPTER 8

# Demonstration Program

*This section will outline the implementation of the demonstration program that shows a specific use case of the simulator: that of programming a drone network to measure temperature over a large area. A discussion is given on the specific methods chosen and why, as well as what this means for other applications of the simulation software. In addition to this, the section will go over the implementation of the physical routing algorithms described in the design section, as well as where they were used. This includes an explanation of the collision avoidance and area representation required for such techniques, and how they may be applied to other applications.*

## 8.1 Drone and Base Station Sample Implementation

In order to demonstrate the capabilities of the octoDrone product, both through the simulation and physical drones, a sample set of code was written. This demonstrative drone network uses heat sensors on the drones to collect temperature data across a predefined area. This does, of course, draw an immediate link with the forest fires prevalent in California. Once the 'normal' temperatures for an area are monitored, it could be extended such that any unexpected change to this temperature might mean there is the beginnings of a fire. Once alerted, wardens could confirm or refute this premise using cameras on the drone. This chapter covers the implementation of the demonstration code and physical routing algorithms used.

### 8.1.1 The Base Station

In this network, as expected, the base station will perform the task of disseminating work between the drones. In addition to this, it will be responsible for collating the data from the drones as it is being collected. Firstly, however, the base station broadcasts its address to everything nearby so that the drones know who to send temperature data to. This avoids flooding the channels with broadcasts to everything. Following this, the base station waits for a set amount of time during which it will receive the addresses of the drones in the network. These addresses are then stored as a vector on the base station.

Once this has been done, the base station determines which sections of the defined area will be allocated to each drone. This is done simply by splitting the area 1-dimensionally along the x-direction, where each drone is responsible for all points in the y-direction as shown in figure 8.1. The base station then waits messages from the drones containing temperature information.

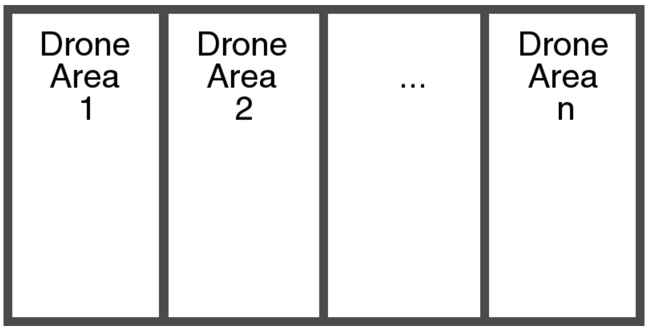


Figure 8.1: Dissemination of the area to drones.

### 8.1.2 The Drones

Similarly to the base station, the first thing the drones do is broadcast their address to everything around, followed by waiting to receive the address of the base station. Before the drones then do anything, they wait for the areas that they are to collect data on. Once this has been received, the drones systematically travel to each allocated point in space, measuring the temperature and moving onto the next. The physical routing techniques used for this are described in section 8.1.6.

Each time a drone collects data from a point in the area, it sends that data back to the base station immediately, meaning the base station receives many small messages each containing a single data point. This was done for two reasons. Firstly, the base station can spread the work of combining the data over the course of the system execution. Secondly, this means that if one of the drones fails, then its previous work is not lost.

### 8.1.3 Handling Messages

The reliable delivery of messages between the drones and base station are handled by the communication module. However, the implementation of a given system must specify how the messages are acting

| Label | Data | Action to Take |
|---|---|---|
| NEWAREA | Four values making a 2D rectangle | Set this area as the new area to collect data over. Call the function to split this into points. |
| BASEIP | An IP Address | Record this IP address as the base station's IP address. |
| DRONEIP | An IP Address | Record this IP address in the list of drone IP addresses. |
| LOC | The current location, angle, speed and routing priority of another drone | Perform collision avoidance. |

Table 8.1: Table showing the messages a drone understands

| Label | Data | Action to Take |
|---|---|---|
| DRONEIP | An IP Address | Record this IP address in the list of drone IP addresses. |
| DATUM | A point location and value | This is a data value from a drone so record it. |

Table 8.2: Table showing the messages the base station understands

upon and how they are managed. For this implementation, both the drone and base station listen for messages during their program loops. When a messages is received it is then 'interpreted' by entering into a separate function that deals with all possible actions that might need to be done regarding a messages. For example, A drone might receive a message informing it of the base station's IP address. The table in figure 8.1 shows the messages that the drone understands on top of those implemented by the base simulation software. Figure 8.2 shows the same for the base station.

### 8.1.4 What Does this Demonstrate?

This sample application demonstrates that it is very much possible for a working drone network to be built using the simulation. It shows the drones working autonomously and separate from each other, as well as a base station performing a similar task as would be expected in a real-world application. While this system only provides simple functionality, it demonstrates the possibility for a wide array of applications using fault tolerance, efficient communication, and robust physical routing.

### 8.1.5 Area Representation

As described in the physical routing section of the design, the drone's understanding of the world is in 3-dimensional Cartesian coordinates. However, for this implementation, the drone does not concern itself with the potential for obstacles in the area it is measuring as it is assumed to be completely open. From this, this means there is no need to store a representation of the 'map' on the drone, but only of the points that the drone should be travelling to and measuring.

When the drone receives the area it is to sense over, it is in the form of four values corresponding to the x-min, y-min, x-max, and y-max denoting a 2-dimensional plane. Using the sensing radius of the drone's sensing equipment, passed in as arguments to the drone when it is started, the drone determines how many measurements it will need to perform in order to cover the whole area. These points are then stored in a C++ queue so that the elements can be accessed in a strict order. The ordering walks up and down the area so that the drone has to travel the smallest distance between points. The queue is efficient and easy to iteratively add to as opposed to a vector.

### 8.1.6 Collision Avoidance

The overall physical routing technique used for collision avoidance was described in the physical routing design section. It was implemented as described there, with both drones calculating if a collision would occur and, if one would, one flying higher than the other so that the collision is avoided.

In order to detect if a collision might occur, then each drone, when sending broadcasting its location to all other drones around, also includes its current position, facing, speed, and priority. Each drone sends this broadcast with the frequency meaning that the interval between broadcasts can be used as the maximum time we need to check for collisions in. From this information, the maximum change in the x direction for this timestep is given by,

```
dx = t * 2 * s * sin(\theta)
```

and the maximum change in the y direction is given by,

```
dy = t * 2 * s * cos(\theta)
```

where t is the broadcast interval, s is the speed of the drone, and $\theta$ is the current facing of the drone.

Unfortunately, this alone leaves the issue presented if the drones are moving too fast and will not collide at the end of their movement, but rather at single point during their movement. While it is possible to extrapolate back from the collision to find the exact point of collision, for this it is not necessary to know exactly where and when the drones would collide, only that they will so it can be

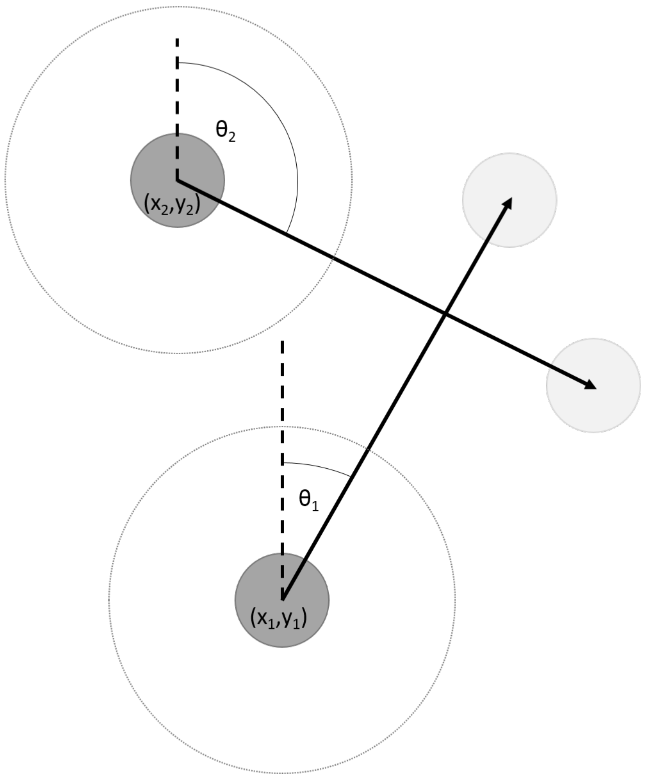


Figure 8.2: Two drones and their projected paths with $\theta$ angles shown.

avoided. Because of this, the algorithm then checks for a collision another four times along the path of the drones.

It is worth noting that this whole calculation is only done by the drone with the lower priority as only it will need to know if it should be getting out of the way. This means only the half the computation is required across the drones in total.

CHAPTER 9

# Testing

*In this chapter the different tests developed for the simulator are detailed and described. A brief description of what each test was intented to verify (with rationale) will be given, along with information on how the results of the test were interpreted, if appropriate. Please note the difference between "basic" programs and the Basic communications module. Every attempt has been made to be unambiguous in this chapter, but the underwritten sections will be easier to follow with this distinction in mind.*

## 9.1 Unit Testing

### 9.1.1 The Simulator

In order to test the individual components of the simulator, a number of different messageable programs and simulations were created.

- **Movement**
  This unit test instantiates a drone and instructs it to move in one dimension, two dimensions, and finally in three dimensions. It also attempts to turn the drone. If the resulting heading and position in 3D space of the drone is the same as the known correct value then the test passes.

- **Communications**
  This unit test instantiates two drones which uses the most basic communications module (that simply delivers every message received to every messageable). It then attempts to send a message from one drone to the other. If the message is received and it correct, then the test passes.

- **Noise Functions**
  This unit test instantiates two drones using the most basic communications module, and passes a noise function to the environment which will drop the first message sent and deliver the second.

The sending program dispatches two messages, and if the receiving program only detects the second message then the test passes.

- **Visualisation**
  The visualisation unit tests are comprised of each of the above tests, but run with the environment variable for visualisation set to true. If the results pass visual inspection (verified against the visualisation specification) then the tests are considered a success.

### 9.1.2 Communications Programs

Each communications module that was developed had its own unit test written. This served the function of testing the individual features of the communications algorithm in turn. As most of the tasks performed by these programs were heavily interlinked with others, it was decided that they would be tested as a whole (by sending a message one can test a number of co-dependent features in succession). If all of the tasks described in the test program succeed then the test passes. As there are multiple parts of the implementation being tested by a single simulation the communication module instances are set to output diagnostic information during the entirety of the process. If at any point the test fails then it is possible to determine which component is broken from this output stream.

#### Basic

The unit test for the Basic communications module verifies that messages can be passed to the communications module, broadcast to other nodes, received, interpreted (in the case of the "KILL" message), and delivered.

#### Basic Addressed

In addition to the above checks, this unit test verifies that the Basic addressed module will deliver messages only to the node identified by the IP address to which it is sent (or rather that nodes will drop any messages which they receive but are not addressed to them).

#### AODV

The testing procedure for AODV is significantly more complicated than for the above. In addition to all of the previously mentioned functionality, the unit test for the AODV communications module verifies that:

- nodes send hello messages at regular intervals

- nodes seeking to send data to a node without an active route generate and broadcast RREQ messages
- nodes seeking to send data to nodes with an active route use that route
- nodes receiving an RREQ for which they have an active route reply with an RREP for that route
- nodes receiving an RREQ for which they do not have an active route forward that RREQ to their neighbours
- nodes which receive an RREP for a route they requested record that route if it more efficient than their current active route for the destination node (or if it is the only active route they have for that node)
- nodes which receive an RREP for the route that they requested on behalf of another node return the received route if it is more efficient than the current active route they have for the destination (or it is the only active route they have for the destination)
- nodes record information about other nodes from received messages, including hello messages
- nodes receiving a data packet for which they are not the intended recipient forward that message on to the correct next hop on their active route to the destination if they have one
- nodes receiving a data packet for which they are not the intended recipient and for which they do not have an active route drop that packet and propagate an RERR packet the next upstream node on that route.
- nodes receiving an RERR packet for one of their active routes propagate that RERR message the next upstream node on that route, if any

### 9.1.3 Parrot version of the Simulator

A single test file was devised for the Parrot library which tested two the two areas of functionality which had been adapted from the original simulator - movement and communications delivery (as opposed to communications modules). The program could be run in sink or source mode, with sink mode simply waiting for a message to be delivered, and source mode sending a message. As soon as the sink version of the program received a message it performed a series of movements to make sure that the drone moved correctly (under both translation and rotation). This program was run on two hardware units, and the test failed if the drone failed to move correctly (or failed to move at all).

### 9.1.4 Demonstration Program

During the implementation of the demonstration program, text output was used as the visualisation had not been completed at that time. This involved careful integration and unit testing of new components, ensuring that the system's integrity was not compromised by any additional functionality. When the visualisation was completed, it was then used to ensure that no previously unexpected emergent behaviour existed. Specific cases were tracked in the code to ensure, for example, there was no case where two drones had collided.

The demonstration program itself acted as a test for the other components of the system, notably the communications module and underlying simulation.

## 9.2 System Testing

The nature of octoDrone's architecture was such that many features did not need to be system tested. For example, if a program was proved to be correct when using the Basic Addressed communications module, the abstracted nature of messaging meant that it would be provably correct using any other addressed communications module. Being able to reason about the components of the project in this way saved a large amount of time during development.

Another such inference was drawn about the correctness of the Parrot library and the main simulator library. If the main simulator was shown to be correct for a given set of functionality (excluding communications and movement), then the same had to have be true about the Parrot version of the simulator. This was because they shared large portions of the code base.

For those parts of the codebase that did require integration testing, the sample progam that was produced to demonstrate the capabilities of the project software made use of all of the features that were unit tested. Because of this it was the best way to ensure that the different parts of octoDrone interfaced correctly. Because this test was verifying multiple assertions, it was important that diagnostic information was emitted with enough detail to identify which part of the software had failed in the event of the test not passing.

## 9.3 Testing Against the Specification

For testing against the original specification, the demonstration program mentioned above contained all of the requirements mentioned in the specification (indeed, it is the reason that those requirements are there). As such, this was used to ensure that all of the individual parts of the simulator interoperated correctly. In the case that the test failed, diagnostic information was emitted with enough detail to identify which part of the software had failed.

CHAPTER 10

# Project Management

*There are many challenges which arise when trying to carry out a project which must be thoroughly documented and accounted for during the planning stages and subsequent design and implementation. For example, there must be considerations for how the project software will be developed, how the team will be organised, how the tasks will be scheduled and the various tools which will be used throughout the project. This chapter will discuss specific areas of project management such as these, and how the project group overcame the various challenges which arose throughout the duration of the project.*

## 10.1 Project Methodology

### 10.1.1 Project Summary

As with any software development project, it is important to first outline the aims of the project, the resources available to achieve these aims, and ensure that the stakeholders in the project have been correctly identified before the project begins. This allows for the project to have a clear view of its objectives and how to reach them. Furthermore, justification for why and how the project should be carried out is necessary to ensure that each group member is satisfied with the direction of the project and their roles within the group. This has been discussed in greater detail in the specification chapter, but will be summarised in this section to outline how the project developed from a management perspective.

The project idea to work with drone sensor networks was unanimously accepted by every group member; with the hardware freely available from the Department, each member was enthusiastic to tackle the various technical challenges which would arise in terms of hardware and software. UAVs and their applications in a sensor network is a subject which is currently a hotbed for research and development, with many different applications. The project deliverable presented an opportunity to experience working on the state-of-the-art in terms of technology and research, with the possibility of a lasting impact on the field. Using several Parrot AR drones, the objective of the group was to create a simulator for sensor networks which could initialise and control airborne nodes, allowing them to function autonomously, as well as take user input to perform a task. The platform was then to be deployed onto hardware available from the Department, including external computers and extra peripherals.

The project was decided to be made for general use case, but with a focus on making a tool which could be used to perform research and outreach activities. The project supervisor approved the project, and continued to give helpful advice and considerations to assist with the planning stages. After agreeing upon the project, considering the resources and selecting a specific use case to work towards, the following work involved researching current methods and designing a simulation for the network. Once the simulation environment had been completed, communications and physical routing would be incorporated into the design, before finally transferring the platform onto the real drones for physical deployment. Each stage of the project was carefully monitored (as described later in this chapter), and tasks were split evenly between the group members in order to ensure that the project was completed within time constraints. Monitoring also ensured that any potential problems were addressed as they appeared. In the final stages of the project, the project deliverables, including the necessary documentation, were finalised and brought to a conclusion in order to be delivered safely and in accordance with the objectives which had been established during the planning stages of the project.

### 10.1.2 Software Development Methodology

For a software-based project, it is important to consider the development methodology which will be adopted, as the lack of effective practices will lead to unpredictability, repeated error, and wasted effort. Develop methodologies range from a document-heavy, tightly-structured waterfall method, to a lighter agile method which focuses on code iterations and adaptability. In order to reflect the number of the members of the group, with fluctuating schedules and time available to work on the project, a highly flexible adapted version of the agile scrum development was adopted. Core agile principles such as frequent delivery of working software and the welcoming of continuous changing requirements, as well as close, daily cooperation between developers have been maintained [29]. However, flexibility is a very important element in the project, as students who may have other external factors to consider, such as deadlines for other assignments, would find traditional, more structured methods difficult to maintain. Waterfall methods are also more appropriate for larger businesses with long-term projects, and so would be inefficient considering the scope of the project.

The flexible approach afforded by agile development allows for adaptive development, such that milestones have been identified, but with flexibility to reach them, and allow for changes to requirements. As a result of the project group structure, compared to traditional scrum methods, sprints are not as heavily constricted by time, and there is no emphasis on providing a demo to the stakeholders during the sprint review. Additionally, the size of the development team, traditionally ranging from 3-9 people, has been downsized to two smaller teams of 1-3 individuals, and the daily scrum replaced with initiated sporadic bursts of development and discussion between members.

The scrum method is also appropriate because certain project tasks such as communications and physical routing can be completed simultaneously, such that the project software can be built up and tested modularly, and new features can be made use of immediately. This also allows for new iterations to influence and improve future iterations without extensive planning, which is suitable for a large project with many different considerations for each task in a small timeframe. Additionally, agile methods are preferred in a situation where an incremental delivery strategy based on rapid feedback is realistic [52], which is relevant to the structure of the project, with frequent meetings with the project supervisor and constant feedback available through correspondence regarding problems and requests.

When using an agile methodology, it is important to avoid the loss of information from a lack of documentation, which is common for a code-centric development methodology. This is especially dangerous for a development team with constantly changing members, as developments not being properly documented leads to a lack of knowledge and the inability to teach new members. However, this is a situation which does not apply to a static group, whose members will not change regardless of circumstance.

## 10.2 Team Structure

As mentioned in previous sections, the project group consists of four members. With such a relatively small group, it was important to ensure that the workload was balanced equally between each member, and that all of the work was accounted for such that each and every task was being handled by one or more of the group members. A group consisting of fewer members is advantageous in that organisation is simpler; discussion and dissemination of tasks is much more straightforward, and each group member is aware of their assigned work, and any possible overlap with other group members. However, the lack of members results in an overwhelming disadvantage in terms of the amount of manpower available to work on the project at any given time. This also results in group members being assigned to work which they are unfamiliar with or are unqualified to handle.

As much as possible, each team member was assigned to a role which reflected their strengths in order to maximise efficiency for each task and simplify the scheduling process. The individual members and their assigned roles are shown below:

- **Alex Henson - Coordinating the report and research**. Responsible for the bulk of research in terms of drones, sensor networks, simulations and routing protocols, as well as handling the format and content of the report.

- **William Seymour - Project manager, communications and physical deployment**. Responsible for distribution of tasks among group members and managing deadlines, implementation of communications protocols and handling transfer of the project platform to the physical drones.

- **Jon Gibson - Developing the simulator**. Responsible for the design and implementation of the drone sensor network simulator and the corresponding foundational infrastructure and protocols for the real network.

- **Ben de Ivey - Developing the demonstration simulation**. Responsible for utilising the simulator to create a simulation of the drone sensor network, as well as assisting with its development.

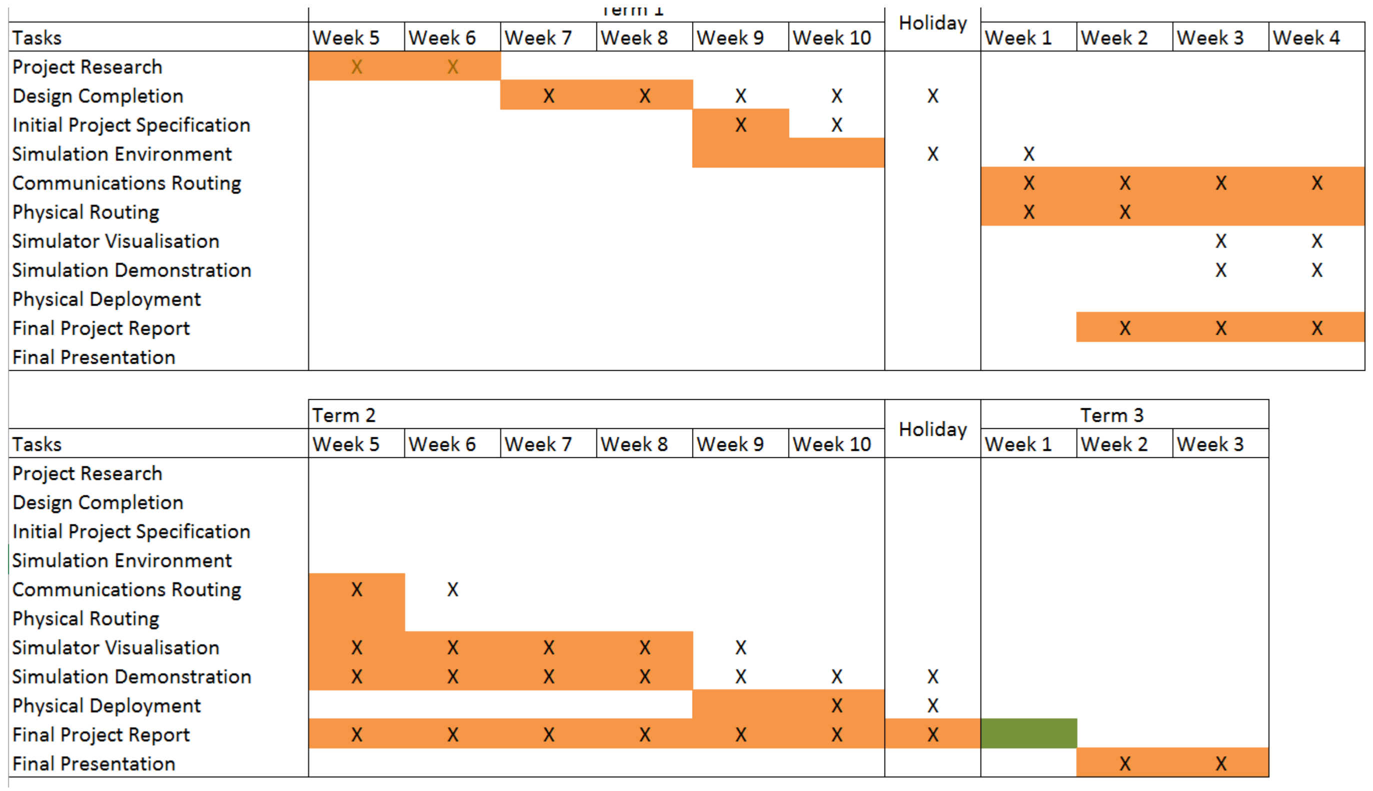
This list is not exhaustive, and many tasks were performed by more than one group member where necessary. The strength of the team could be seen in the ability to disseminate tasks easily and for each member to take responsibility for their individual tasks, using their skill sets to focus on the requirements most suited to them.

## 10.3 Time Management

One of the most important areas of project management is the ability to complete the project within time constraints. Therefore, it is necessary to accurately identify and schedule tasks to be completed within a reasonable timeframe. A projected timeline for the project is shown in figure 10.1, which outlines the fundamental tasks, and a Gantt chart is also included in figure 10.1 which shows the scheduling of those events, from the beginning of the project until delivery.

Table 10.1: Schedule of project tasks

| Time | Task | Details |
|---|---|---|
| T1 W1,2 | Group Meeting, Supervisor Meeting | Meet up to discuss project planning and schedule meeting times. |
| T1 W3,4 | Project Planning | Decide upon tasks to be completed up until project specification. |
| T1 W5,6 | Project Research | Research subject material to aid in design such as simulators and drones. |
| T1 W7,8 | Design Completion | Complete the design of each project component so as to begin implementation. |
| T1 W9 | Initial Project Specification | Write-up project specification. Have supervisor check draft and revise. |
| T1 W9,10 | Simulation Environment | Complete implementation of basic simulation environment including nodes and communication models. |
| Holiday | N/A | Temporary buffer for tasks running overtime. |
| T2 W1-5 | Communications Routing | Complete message passing using communications algorithms in simulator. |
| T2 W1-5 | Physical Routing | Complete physical routing for nodes in simulator. |
| T2 W5-8 | Simulator Visualisation | Complete graphical output for simulation execution. |
| T2 W5-8 | Simulator Demonstration | Complete demonstration of drone network simulation to be adapted to hardware |
| T2 W9,10 | Physical Deployment | Complete adaptation of simulation code to hardware. Deploy network and record results. |
| Holiday, T3 W1 | Final Project report | Complete project report following results of deployment. |
| T3 W2-3 | Final Presentation | Prepared demo video for presentation. Present results to complete the project. |



| | Term 1 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Tasks | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | Holiday | Week 1 | Week 2 | Week 3 | Week 4 |
| Project Research | X | X | | | | | | | | | |
| Design Completion | | | X | X | X | X | X | | | | |
| Initial Project Specification | | | | | X | X | | | | | |
| Simulation Environment | | | | | | | X | X | | | |
| Communications Routing | | | | | | | | X | X | X | X |
| Physical Routing | | | | | | | | X | X | | |
| Simulator Visualisation | | | | | | | | | | X | X |
| Simulation Demonstration | | | | | | | | | | X | X |
| Physical Deployment | | | | | | | | | | | |
| Final Project Report | | | | | | | | | X | X | X |
| Final Presentation | | | | | | | | | | | |

| | Term 2 | | | | | | | Term 3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tasks | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | Holiday | Week 1 | Week 2 | Week 3 |
| Project Research | | | | | | | | | | |
| Design Completion | | | | | | | | | | |
| Initial Project Specification | | | | | | | | | | |
| Simulation Environment | | | | | | | | | | |
| Communications Routing | X | X | | | | | | | | |
| Physical Routing | | | | | | | | | | |
| Simulator Visualisation | X | X | X | X | X | | | | | |
| Simulation Demonstration | X | X | X | X | X | X | X | | | |
| Physical Deployment | | | | | | X | X | | | |
| Final Project Report | X | X | X | X | X | X | X | 🟩 | | |
| Final Presentation | | | | | | | | | X | X |

Figure 10.1: Gantt chart of project schedule

## 10.4 Progress Tracking

### 10.4.1 Meetings

Regular meetings were scheduled once a week with both the project group and the project supervisor. Meetings with the supervisor consisted of sharing current progress, including any successes and challenges, as well as to seek advice on specific topics of difficulty such as simulation tools, accurately modelling error and routing techniques. Queries were also directed to the supervisor by email outside of meetings where necessary, to handle urgent requests or to check deliverables.

The general group meetings involved all four members of the group gathering at the Department to discuss progress, assign new tasks as a result of discussions with the supervisor, or re-assign current tasks to improve scheduling. The group also examined the structure of the code, discussed bug fixes and new iterations of code, and corroborated any research findings using collaboration tools, which will be explained in section 10.6. Extra meetings were also scheduled outside of the usual schedule in order to prepare for deadlines, or to coordinate workflow in the event of a task being assigned to more than one member. Certain tasks, such as physical deployment, required members to gather in one location in order to observe and work with the drones in person.

### 10.4.2 Weekly Review

In accordance with the software methodology chosen for the project, the group also informally discussed the status of the project in accordance with the schedule shown in figure 10.1, and received an update from each member as to their individual progress. By sharing the current state of each task with other members, it was possible to gauge the overall status of the project and to reassess any tasks which took up an inordinate amount of time. Instances of project review were also reserved for upcoming deadlines to ensure that project deliverables in particular were proceeding as planned. Minutes of meetings were kept in order to keep track of ongoing tasks.

## 10.5 Source Control

As a software-based project, it is important to ensure proper source control, so that code can be protected and members can be notified of new iterations, and how they differ from previous versions. While the source code itself did not have version numbers explicitly appended to it, Version Control Systems (VCS) were embedded into the collaboration tool used for code, including the ability to add messages which addressed the changes introduced by updated code, and to highlight any potential errors or bug fixes.

## 10.6 Collaboration Tools

### 10.6.1 GitHub

To manage the project material, a git repository was created to ensure consistency amongst the project members' work and safety for the code base. Github, a web-based git repository hosting service, was used to incorporate source code management and distributed revision control to the project code. Each group member had their own repository locally, and branches were also used to ensure that major changes to the code base did not affect the original version of the code. This also allowed each member to avoid overwriting work when committing changes in the same area of code, which would result in merge conflicts. Using Github, the workflow was separated safely and manageably, and previous versions were retained by the repository so that they could not be lost. This also allowed each member of the group to contribute to the same repository without requiring them to be in the same physical space, or constantly exchanging updated code.

### 10.6.2 Google Drive

Google Drive was also used in order to manage project material, as well as have a backup for the code base and documentation. This is especially useful for project deliverables such as the report and the specification, as each member could collaborate in real-time with one another on the same documents. Google Drive also allowed for access on multiple devices such as phones, tablets and laptops, which was especially convenient for group meetings.

### 10.6.3 Hastebin

During the development of the project software, Hastebin was occasionally used for individual code snippets, as the UI supports code highlighting and is designed for sharing programming code. Alternatively, a real-time format such as Etherpad could have been used, but Hastebin was chosen for its ease-of-use and due to the fact that individual group members typically worked on separate areas of the same code, so Hastebin was utilised more for checking (incorrect) code than to develop code side-by-side.

### 10.6.4 Facebook

In order to communicate with other group members, Facebook was used as the primary method of contact. Each group member was already using Facebook, which made it the most common and the easiest way for group members to communicate. Facebook also allows for sharing files, which was found to be more convenient in the case of small files which did not require to be backed up. Facebook became a useful tool for general discussion about the project, including the project schedule and the code, as well as for the dissemination of tasks on-the-fly.

## 10.7 Project Challenges

At the beginning of the chapter, a summary was given of the main elements which the project was comprised of. In this section, the various challenges which had to be overcome in order to complete the project successfully are analysed. Specifically, areas such as researching, scheduling, organisation and development will be discussed, and how these challenges were dealt with throughout the project.

### 10.7.1 Planning

The project was incredibly difficult in terms of the amount of work that had to be done in the limited amount of time available, and so effectively planning how and when each stage of the project would be completed was crucial to the project success. Each task which had to be completed was scheduled using the Gantt chart shown in figure 10.1, and the group was organised such that each member was aware of their individual tasks and responsibilities, and the timeframe available to them. The Gantt chart provided a good overview of the project activities, as well as being simple to read and understand; it was immediately clear that each member of the group what the current status of the project was, and what had to be completed at what time.

### 10.7.2 Research

One of the biggest challenges during the project was researching in order to find the proper tools and resources which were available that would be relevant to the project. A comparison of the various simulator libraries which were available was made, in order to find out their complexity, utilisation and functionality, such as ns-3 and OMNet++, and whether or not these libraries were necessary in order to create a drone sensor network. Ultimately, an original simulation was designed and developed by the group, which was influenced by the utilities provided by such libraries, and the use of ns-3 in the early stages of development was helpful to understand the various protocols which would need to be implemented into our own simulator.

### 10.7.3 Development

Following the planning stages, the project had carefully been broken down into tasks which had to be completed. According to the plan, the simulation would form the basis of physical deployment, in order to simplify the project as a whole and save time. During the development stage, the utmost care had to be taken to ensure that the simulation framework could be effectively transferred over to physical deployment during development. With this in mind, the simulation code was written in such a way that the only remaining factor to consider for physical deployment was the ability to use libraries which would interact with the drone's hardware, such as the ability to takeoff, move, turn, and land. Additionally, creating a simulation which could accurately model real-world communications without the use of network simulator libraries turned out to be a much more difficult task than had originally been scheduled for. The workload was subsequently re-evaluated and scheduled to realistically reflect the amount of time which would be taken, which involved using the time buffer of working over the holidays.

### 10.7.4 Deployment

When planning for physical deployment, there were several difficulties in terms of working with hardware which was not guaranteed to be suitable for the task. In order to provide autonomy to the drones, it was necessary to attach a Raspberry Pi to the drone, as well as GPS sensors, and then transfer the code base to the Pi in order to control the drone directly. Having no prior experience working with hardware explicitly, the group had to manage to configure the Pi in order to be able to connect and hook into the drone directly through Wi-Fi, and execute the code from the host (or base station). There were also concerns that attaching extra peripherals to the drone would potentially have weighed it down to the point that it would be incapable of sustaining its altitude, which would also affect its autonomous flight plan. The capabilities of the Parrot AR drone in terms of handling heavier payloads had to be researched and tested to confirm the maximum payload weight that it could handle.

## 10.8 Risk Management

The risks involved with the project are detailed in table 10.2.

Table 10.2: List of risks for the project

| # | Risk Event | Mitigation |
|---|---|---|
| 1 | Team member falls ill during the project | Reassign tasks where necessary and focus on schedule. Use buffer time if necessary. |
| 2 | Parrot drones have faulty hardware | Reason with Department for replacement hardware and focus on other tasks. |
| 3 | Any hardware breaks or is faulty | Same as 2 - replace where necessary or continue without if possible. |
| 4 | Lack of contribution from a team member | Attempt to encourage the team member and assist with the task if necessary. If problem persists, consult project supervisor. |
| 5 | Development turns out to be too difficult technically | Consult project supervisor for advice as to changing the project/project task/how to complete the task in question. |
| 6 | Code becomes corrupted or lost due to accident/damage | Recover backup code from git repository and continue. |
| 7 | Simulation environment is too slow to perform as needed | Consider optimisation of simulation as priority over other tasks in order to preserve core functionality |
| 8 | Research for drone sensor networks is limited | Focus on combinations of alternatives e.g. research on drones, research on sensor networks |
| 9 | Unable to implement visualisation for the simulation | Focus on ensuring that other key functionality is achieved/maintained for the simulator and consider as future work |
| 10 | Drone gets lost during deployment due to loss of signal/out of range | Liaise with campus security to arrange for retrieval of the drone |
| 11 | Communications routing is too difficult/time costly to implement | Consider finding and using pre-existing libraries which can be imported to the stimulator |
| 12 | Physical routing is too difficult/time costly to implement | Consider reverting to core functionality or pre-existing libraries where necessary |

| # | Risk Event | Mitigation |
|---|---|---|
| 13 | Drones are unable to perform collision detection | Ensure that drones manage individual, well-separated airspace only and are physically unable to go near each other |
| 14 | Physical deployment is not completed on time | Focus on functionality of the simulator for demo and present theoretical results for real deployment where possible |
| 15 | Scheduling is poorly managed and tasks are overestimated timewise | Re-evaluate schedule and use buffer time if necessary. Tasks may also be added or removed where necessary |
| 16 | Demonstration code for simulation is not completed on time | Revert to using example code used to test simulation for demo and physical deployment. |
| 17 | Use of drones on campus is deemed illegal/unsanctioned | Find open airspace in alternate location which is not monitored/free to use |

## 10.9 Legal, Social, Ethical, and Professional Issues

The subject of drones has been the focus of much controversy with regards to issues of privacy, invasion of airspace and other unsanctioned uses of UAVs with the ability to, for example, use a camera to take pictures or video. Therefore, a platform for an entire network of drones with a variety of sensors which is open source presents a multitude of possible problems. Furthermore, with the potential to be used as a foundation for military efforts, which form the basis of many current topics of research into drones and sensor networks, there are many implications for potential misuse of the project. As a result, it is necessary to research and discuss the possible methods of unlawful utilisation of drones and drone networks, in order to ensure that the project group is aware of these issues.

### 10.9.1 Legal Issues

The law is, however, very clear with regards to the use of firearms against UAVs, whether or not the UAV in question was trespassing or invading on private property. In November 2014, a man was fined $850 in damages for shooting down a neighbourâĂŹs drone while it was flying over the neighbour’s property, and then refusing to pay any compensation for destroying the drone [31]. The Federal Aviation AdministrationâĂŹs (FAA) definition of a drone as aircraft also means that, technically, shooting a drone could result in a maximum penalty of a 20 year prison sentence. Pursuing legal action in cases such as this is difficult, as it is not illegal to fly a drone in public.

There are several instances where the definition of misuse of drones has been well established and offenders have been prosecuted. In the very first case of prosecution against unsanctioned drone usage, a security guard was sentenced after repeatedly flying drones over and around Premier league football stadiums, Buckingham Palace and Parliament buildings, where he was fined and banned from flying UAVs [46]. In the US, FAA regulations state that drones must not be flown near airports and other areas with manned aircraft, as well as placing a ban on altitudes over 400ft. The Civil Aviation Authority (CAA) in the UK states that for a Small Unmanned Surveillance Aircraft (SUAS) of less than 20 kg, the operation must not endanger anyone or anything, and the aircraft must be within visual line of sight [4].

For the specific case outlined above, the aircraft being used for surveillance purposes was in violation of the rules of direct line of sight of the aircraft as well as being subjected to tighter restrictions with regard to the minimum distances that you can fly near people or properties that are not authorised. In most instances of prosecution, the owner of the drone would be required to have permission from the CAA to fly their drone in an area that would usually be forbidden access.

There have also been cases of the use of drones in order to smuggle illegal substances into high security areas. Scotland Yard has logged various different drone-related incidents, including complaints about drones being used to ferry drugs into prison [46]. It is highly likely that an individual or individuals would be responsible for such use of drones, particularly in violation of multiple laws [4], as opposed to commercial businesses. However, this implies that open source autopilot software and other similar platforms may be used unlawfully by individuals.

### 10.9.2 Social Issues

A interesting issue arises where any drone carrying a payload such as a camera has the potential for recording data unlawfully. In one instance, a drone which was found to be hovering over an ATM that seemed to be recording people entering their pin numbers into a cash machine. It can be argued that the drone was not breaking the law by simply recording a public space[46]. However, it is difficult to prove that the usage of the drone was for the express purpose of obtaining peoplesâĂŹ pin number, or if this is unlawful to begin with, in the same way that there is no law which expressly forbids someone from looking over the shoulder of an ATM user.

A full-length documentary called Speciesism: The Movie based its foundational discoveries and source of information by using drones to spy on factory farms and record evidence of environmental damage, and went on to feature major global press coverage. In this particular instance, the law with regards to prosecution was unclear, despite the activity being labelled as âĂŸspyingâĂŹ. This could suggest that such uses of drones, even on potentially private property, may not be possible to prosecute, although this may be unique to incidences of whistleblowing. It will be up to society to determine where the acceptable boundaries of drone usage lie, and how this should be represented in legislation.

### 10.9.3 Ethical Issues

As discussed in previous sections, engaging in military operations is one of the most prevalent uses for drones, as well as being a popular area for research and development of UAVs. Fortunately, the specifications of consumer-level drones are not of a high enough calibre to warrant usage in the military, which requires state-of-the-art hardware, including weaponry. However, there is an initiative to create small, hand-sized mini drones for soldiers in the US Army, for the purpose of small scale intelligence gathering and reconnaissance, in place of conventional air support [14]. Projects such as this have implications for the usage of drones of all shapes and sizes in military operations, which would suggest that implementations of drone autonomy by hobbyists and businesses may form a basis for research and development in the army.

The use of drones for spying is not only limited to consumers, militaries have also been responsible for deploying drones to spy on civilians in their home country. The Department of Defense in the United States has admitted to using Predator and Reaper military drones in the US since 2006 in order to support domestic civil authorities, which was made public in March 2016 [45]. According to the government, these domestic drone flights are stated as not being in violation of any laws, and were being employed in a âĂIJvery, very minimal way, very seldomâĂİ.

### 10.9.4 Professional Issues

Given that the project is released under an open source license, it is possible that it will be used to perpetrate acts which are illegal or unethical. If there should be an onus on creators to restrict the functionality of their projects, then this has troubling implications in terms of free speech. Additionally, there are many use cases of drones which could be classified as either ethical or unethical depending on intent (reconnaissance for a police drugs raid compared to reconnaissance for a bank heist).

CHAPTER 11

# Project Outcome

*The aim of the project was to create a generic simulation for drone networks which was capable of being extensible to a variety of different UAV hardware, and to prove its effectiveness on the specific use case of the Parrot AR drone by bridging the gap between the simulation and the hardware. This chapter will evaluate the success of the project against the original requirements, including a summary of the results of creating the simulator, the implementation of routing, and the transition to physical deployment. It will also give a summary of how the project scheduling performed compared to the projected schedule, and make suggestions as to how the project could have been extended given a larger timeframe.*

## 11.1 Project Deliverables

### 11.1.1 Simulator

One of the biggest challenges to creating the simulator was the decision to migrate from the core functionality provided by ns-3, and instead focus on creating a simulator from the ground up. As a result of research and testing, it was found that ns-3 was too bloated and rigid to create an efficient, general-purpose simulation for drone sensor networks. In place of using a pre-existing library for network simulation, it was decided that the simulator would be built using C++ from first principles. The design and development of the simulator was incredibly successful; despite concerns that the simulator would not be able to accurately reflect a real environment given the size of the workload and the scope of the project, the simulator was capable of initialising a drone sensor network with all of the functionality which was expected of it. Features of the simulator include the ability to create a network environment, instantiate drones (messageable objects), install communications modules on them such that drones (and the base station) can communicate, pass messages based on arbitrary communications protocols, and visualisation of the simulation itself. As the aim was to create a fully functional simulator which could be used to model arbitrary sensor networks, the simulator demonstrates all of the necessary components for the task.

### 11.1.2 Routing

In order to ensure that the simulation performed as expected, it had to be capable of effectively modelling communications algorithms and the ability for each node to control its own individual airspace. Additionally, it was necessary for the simulation to be capable of selecting which communications routing protocol to use at runtime, such that the simulation could be generic enough to incorporate any algorithm. This allows the end user to select the optimal algorithm to model the desired environment in terms of energy efficiency, computation time and complexity. For the implementation of physical routing, methods for collision detection and governing airspace were to be defined in the simulator, such that each drone would be able to carry out its own flightplan autonomously, which included calculating a priority in order to resolve potential collisions. Overall, both communications and physical routing were accurately represented in the simulation, and the demonstration simulation reflected the requirements of the design. Physical routing was not as complex as originally intended, as it did not include specific algorithms for pathing, which may have added to the versatility of the simulation.

### 11.1.3 Physical Deployment

Physical deployment was necessary in order to prove that the simulation could be accurately reflected in the hardware, providing a general use case for the simulation to adapt to. The main tasks for physical deployment included ensuring that the hardware, especially with regards to the peripherals for the drone, could be used correctly, that the simulator could be adapted to the library selected for hooking into the drone, and that the code could be transferred over to the external computer which was attached to the drone. Provided that the simulator accurately modelled real deployment, the assumption was that the code did not have to be re-factored for the sake of deployment. While there were some difficulties with transferring the code over to the Raspberry Pi, the library used for controlling the droneâĂŹs physical movements was sufficiently tested, and the drones actually performed according to the simulation which was built to demonstrate a real drone sensor network. Despite concerns about the weight of the drone, the peripherals were sufficiently light, and did not affect the flight of the drone.

Running distributed applications on the Raspberry Pi was made extremely easy by having the compiled simulation take care of starting and stopping any additional threads it needed to create. This ease of use was such that it piqued the interest of a number of faculty who identified it as being an excellent outreach resource.

All in all, the reference deployment of octoDrone was a great success. The final version of the parrot library was able to take simulator programs and run them on hardware with no changes and simulation files were usable with minimal changes. The quadcopters performed very similarly to our simulations, which simultaneously validates the accuracy of both the simulator and the deployment.

## 11.2 Time Management

Following the initial planning stages of the project, the group created a Gantt chart in order to schedule the tasks which had to be completed and the amount of time allotted to each task. For example, the design of the simulation environment was to be completed by week 10 of term one. Unfortunately, this schedule was too optimistic and did not reflect the allowances which had to be made for deadlines relating to each group memberâĂŹs individual work. Each group member had several coursework deadlines scheduled for the end of the first term. As a result, development did not begin until the beginning of the second term; contingency time was allotted to allow for any delays or changes to requirements, and the holiday was able to be used as a buffer for finalisation of the design stage. In order to compensate for lost time, the frequency of group meetings and collaborative coding sessions was increased, and tasks such as communications and physical routing were performed concurrently.

The project was able to progress according to schedule during the second term, as research and development, including the group project report, continued unabated until the final weeks of the term. During this time, the group focused on completing the design for the visualisation of the simulation, the implementation of communications protocols following research, and preparing the drones for physical deployment by testing the hardware and the library which would be used to pipe the simulation framework to the drone itself. The visualisation process was found to be more difficult than expected, and was not fully completed until the beginning of the third term. Nonetheless, the group did not require any extensions for deadlines and successfully delivered the project within time constraints, with minimal changes to requirements and scheduling.

## 11.3 Requirements Evaluation

In the specification chapter of the report, the functional and non-functional requirements were carefully established, such that the success of the project could be measured against the utilities which were intended for it. While it is difficult to measure non-functional requirements in terms of success, most of the core functionality such as sending, listening for, and receiving messages can be seen in the simulation. The simulation code was shown to be adaptable to the physical drones for real deployment, and the extensibility of the code as a non-functional requirement can be seen from the results of the simulation demonstration. Additionally, the functional requirements of the project do not provide an exhaustive list of the features contained within the project software, as there are many other features such as simulation timing objects for time-based events which were also included in the simulation. Given that the project is not marketable and that the stakeholders can be identified but not consulted with, it is also difficult to ensure that the project software meets requirements which have not been established by the group.

Some of the requirements for the projects were not met due to time constraints, and some requirements were adjusted to reflect the status of the project as it progressed. For example, one of the key features which was intended for the project was the ability to implement user input-tasking, where the user could specify a job for the network to carry out. While it is possible to show that the simulation can be extended to the hardware and adapted to any arbitrary task without performing the task explicitly, the project did not progress to a state where a specific demonstration could be shown of the network receiving a task from the user to complete using sensory information and carrying it out. Another functional requirement was the ability of the nodes to move through the environment using pathfinding algorithms chosen at runtime, similarly to communications routing. The drones are capable of moving through the environment by managing their own airspace, but they lack the implementation of more complex, efficient physical routing protocols, and there are no alternatives to the current implementation. However, the lack of these features does not detract from the completeness of the solution, and a result of agile development is simply that the requirements continue to change to reflect the state of the project.

## 11.4 Future Work

There are many possible extensions and improvements to the project which could have been implemented, which would have been examined given a larger timescale. In particular, there were many ideas with regards to improving the efficiency of the simulator, improving the API, and adding more communications and physical routing libraries to the system. The simulation environment is the core of the project software, and one of the biggest problems is the current runtime, which has the potential to be optimised, as discussed in section 6.2. Within the environment, it would be possible to improve the broadcasting function which currently tests the entire list of messageables. This could be improved by attempting to limit the number of messageables to those within the immediate vicinity of the broadcast through segmentation of the environment, which would greatly reduce the overheads involved in checking each messageable object. Another potential extension with regards to the simulator is the upkeep function of the drones, which currently called upkeep one drone at a time, which causes dependencies between drones. This process could be made more efficient by parallelising the upkeep function (in the case of a system that can handle a large amount of threads).

In order to improve the functionality of the simulator, the number of communications routing libraries could also be increased to provide more variety in modelling the environment. The current simulator only supports the Basic messaging protocol as well as routing based on AODV. As a result, the end user who wishes to model the system may find that it is difficult to find the optimal routing protocol for their specific use case. Adding more routing protocols such as DSR would allow for more options when modelling the environment, which would likely allow the user to improve the accuracy of the their network environment by finding the most suitable algorithm. In addition to communications routing, the current simulator code only performs physical routing by allowing each drone to manage its own airspace and avoid collisions. While the basic method of movement and avoiding collision allows for a sensor network to be modelled accurately, it may not reflect the needs of the user and could be further optimised with the use of other existing pathfinding algorithms.

On the note of the end user, it is also important for the simulator to be user-friendly, such that the functions which need to be used to tailor the stimulator to specific hardware are easy to use and understand. Currently, the API is not very readable; it is quite difficult to see which functions are supposed to be used and how. In order to improve the experience for the user, the simulator could be improved by adding prefixes for functions, so that they are more visible, or by adding another level of inheritance to the tree, which would simplify the architecture and make it more secure for the user.

With regards to physical deployment, the project has only implemented a specific use case involving the Parrot AR 2.0 Power Edition drone. In order to aid in the generic nature of the simulation and foster a more complete solution, the simulator could be adapted to more cases using different hardware. In doing so, errors which are specific to the Parrot drone could be highlighted, as well as overcoming issues related to hardware other than the Parrot which have yet to be seen. By attempting deployment on different hardware, the simulator can be shown to be portable to more than one platform and thus capable of being adapted to any arbitrary hardware. While the team envisioned the ability to extend the project software to multiple platforms, this was practically unrealistic given the budget of the project.

CHAPTER 12

# Conclusion

*This chapter will give an overall outline of the project as a whole, including a summary of the project idea, the design and implementation of project components, the issues which were encountered along the way, and the outcome of the project in its completion.*

## 12.1 Project Idea and Management

As a project, the research and development of a generic simulator for implementing drone sensor networks has been incredibly challenging and rewarding. In the project evaluation chapter (chapter 11, considerations were made for the success of the project in terms of the components required and how the project was scheduled over the seven month period of the module.

The project group, comprised of four people, was able to become acquainted and solidify themselves as a team in order to successfully overcome the tasks and challenges that came with the project. From the very beginning of the project, the team were able to establish a productive working atmosphere and an appreciation for the project concept, and the individual skills that each member would bring to the table in order to see it fulfilled. The project itself came with many hardships; drones sensor networks are a recently emerging field, which meant that it was difficult to find a large amount of material as a topic of research. The idea to create a sensor network with the use of drones in and of itself was a relatively new idea which required many considerations in terms of the feasibility of hardware, software libraries, and the existence of previous solutions to draw inspiration from. The prospect of designing and creating a simulator which could then serve as the framework for the deployment of a physical network of drones was incredibly exciting, and reflected the enthusiasm of each member to work on state-of-the-art technology.

## 12.2 Network Simulator

The creation of the simulator was originally designed to be based on the pre-existing network simulator ns-3, which was the most popular application for creating a network simulation. As originally laid out in the specification, it was planned to use ns-3 as the foundation of the network simulator. After making the decision to change the basis of the project design by creating a network simulator in C++ from scratch, the group found that it was possible to create a general use simulator which could be adapted to any hardware more readily. The software components then developed into a platform which reflected the objectives of the project more accurately. The design for the simulator environment itself was broken down into its core components: the nodes in the network (drones and base station), the communications modules installed on the nodes, the routing protocols which would be used by the network, and the code which would instantiate the environment and execute the simulation. The code for each component was meticulously planned, written, and tested, which was possible due to the modularity of the system, before being integrated into the simulator environment.

There were many issues with the development of the simulator such as having to implement communications protocols manually instead of being able to use pre-existing libraries which allow these protocols to be used automatically. While the idea to create a domain specific generic simulator from first principles was unique, it presented many challenges in practice, as the level of functionality which could be expected from using a simulator such as ns-3 would not be immediately available or feasible to introduce given the manpower and time scale of the project. However, this decision allowed for the group to create a platform which was simple and easy to manipulate - it contained only the bare minimum functionality for the task at hand, reducing the complexity and computation time significantly. Despite the technical difficulty involved, the group was able to successfully build a platform which could simulate a drone sensor network and output a visualisation of the results.

## 12.3 Physical Deployment

After the completion of the simulator, the next step was to implement an example physical deployment by transferring the simulation code over to real hardware. This reflected the objective to move from simulated programs to real hardware whilst avoiding the necessity of rewriting the same software, which is the appeal of the generic stimulator that was created. The physical deployment stage of the project went incredibly smoothly with only minor changes to the original design, such as using Ethernet for internode communication following the realisation that the Raspberry Pis did not have to be attached to the drones specifically. Despite concerns from the group that the hardware would not be compatible with the simulation, alternative solutions were found for any outstanding problems and the project was scheduled such that allowances could be made for any hindrances resulting from hardware issues.

## 12.4 Reflections

Overall, the project was completed successfully due to careful and concise planning, the enthusiasm and determination of each group member to complete their assigned tasks within their allotted time, and the well-founded management of the project in terms of scheduling and development methodology. The project itself can be said to have made contributions to multiple fields of research, with implications for future development and improved research methods surrounding drone and sensor network technology. This includes the decision of the Department to make use of this project as a demonstration to prospective students, with the aims of garnering interest in the pursuit of Computer Science and similar concepts which can be derived from this project.

## 12.5 Closing

In final conclusion, the project has been a fantastic learning experience with regards to the development of a large team project over an extended period and the difficulties which are associated with it. The results of the project show that the concept is both possible to implement and expand upon, and that, despite being a relatively new field of research and development, there exist possible applications for future projects. The support and advice from the project supervisor and among the members of the group was critical in reaching the level of success that the project achieved, and it will be exciting for each member of the team to see how similar projects in the field “take off” in the future.

APPENDIX A

# API Documentation

# octoDrone

0.1

Generated by Doxygen 1.8.11

# Contents

Chapter 1

# Bug List

**Member glfwSetMonitorCallback (GLFWmonitorfun cbfun)**

**X11:** This callback is not yet called on monitor configuration changes.

**X11:** This callback is not yet called on monitor configuration changes.

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 4

# Class Index

## 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 5

# Module Documentation

## 5.1 Context handling

### Typedefs

- typedef void(∗ GLFWglproc) (void)

  *Client API function pointer type.*
- typedef void(∗ GLFWglproc) (void)

  *Client API function pointer type.*

### Functions

- GLFWAPI void glfwMakeContextCurrent (GLFWwindow ∗window)

  *Makes the context of the specified window current for the calling thread.*
- GLFWAPI GLFWwindow ∗ glfwGetCurrentContext (void)

  *Returns the window whose context is current on the calling thread.*
- GLFWAPI void glfwSwapInterval (int interval)

  *Sets the swap interval for the current context.*
- GLFWAPI int glfwExtensionSupported (const char ∗extension)

  *Returns whether the specified extension is available.*
- GLFWAPI GLFWglproc glfwGetProcAddress (const char ∗procname)

  *Returns the address of the specified function for the current context.*

### 5.1.1 Detailed Description

This is the reference documentation for context related functions. For more information, see the Context handling.

### 5.1.2 Typedef Documentation

#### 5.1.2.1 typedef void(∗ GLFWglproc) (void)

Client API function pointer type.

Generic function pointer used for returning client API function pointers without forcing a cast from a regular pointer.

#### 5.1.2.2 typedef void(∗ GLFWglproc) (void)

Client API function pointer type.

Generic function pointer used for returning client API function pointers without forcing a cast from a regular pointer.

### 5.1.3 Function Documentation

#### 5.1.3.1 GLFWAPI int glfwExtensionSupported ( const char ∗ *extension* )

Returns whether the specified extension is available.

This function returns whether the specified client API extension is supported by the current OpenGL or OpenGL ES context. It searches both for OpenGL and OpenGL ES extension and platform-specific context creation API extensions.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↵_NO_CURRENT_CONTEXT error.

As this functions retrieves and searches one or more extension strings each call, it is recommended that you cache its results if it is going to be used frequently. The extension strings will not change during the lifetime of a context, so there is no danger in doing this.

**Parameters**

| | | |
|---|---|---|
| `in` | *extension* | The ASCII encoded name of the extension. |

**Returns**

`GL_TRUE` if the extension is available, or `GL_FALSE` otherwise.

**Thread Safety**

This function may be called from any thread.

**See also**

context_glext
glfwGetProcAddress

**Since**

Added in GLFW 1.0.

#### 5.1.3.2 GLFWAPI GLFWwindow ∗ glfwGetCurrentContext ( void )

Returns the window whose context is current on the calling thread.

This function returns the window whose OpenGL or OpenGL ES context is current on the calling thread.

**Returns**

The window whose context is current, or `NULL` if no window's context is current.

**Thread Safety**

This function may be called from any thread.

**See also**

context_current
glfwMakeContextCurrent

**Since**

Added in GLFW 3.0.

#### 5.1.3.3 GLFWAPI GLFWglproc glfwGetProcAddress ( const char * *procname* )

Returns the address of the specified function for the current context.

This function returns the address of the specified core or extension function, if it is supported by the current context.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↩_NO_CURRENT_CONTEXT error.

**Parameters**

| | | |
|---|---|---|
| `in` | *procname* | The ASCII encoded name of the function. |

**Returns**

The address of the function, or `NULL` if an [error](error_handling) occurred.

**Remarks**

The address of a given function is not guaranteed to be the same between contexts.
This function may return a non-`NULL` address despite the associated version or extension not being available. Always check the context version or extension string first.

**Pointer Lifetime**

The returned function pointer is valid until the context is destroyed or the library is terminated.

**Thread Safety**

This function may be called from any thread.

**See also**

context_glext
glfwExtensionSupported

**Since**

Added in GLFW 1.0.

#### 5.1.3.4 GLFWAPI void glfwMakeContextCurrent ( GLFWwindow ∗ *window* )

Makes the context of the specified window current for the calling thread.

This function makes the OpenGL or OpenGL ES context of the specified window current on the calling thread. A context can only be made current on a single thread at a time and each thread can have only a single current context at a time.

By default, making a context non-current implicitly forces a pipeline flush. On machines that support `GL_KHR_↩ context_flush_control`, you can control whether a context performs this flush by setting the GLFW_CO↩ NTEXT_RELEASE_BEHAVIOR window hint.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose context to make current, or `NULL` to detach the current context. |

**Thread Safety**

This function may be called from any thread.

**See also**

context_current
glfwGetCurrentContext

**Since**

Added in GLFW 3.0.

#### 5.1.3.5 GLFWAPI void glfwSwapInterval ( int *interval* )

Sets the swap interval for the current context.

This function sets the swap interval for the current context, i.e. the number of screen updates to wait from the time glfwSwapBuffers was called before swapping the buffers and returning. This is sometimes called *vertical synchronization*, *vertical retrace synchronization* or just *vsync*.

Contexts that support either of the `WGL_EXT_swap_control_tear` and `GLX_EXT_swap_control_tear` extensions also accept negative swap intervals, which allow the driver to swap even if a frame arrives a little bit late. You can check for the presence of these extensions using glfwExtensionSupported. For more information about swap tearing, see the extension specifications.

A context must be current on the calling thread. Calling this function without a current context will cause a GLFW↩ _NO_CURRENT_CONTEXT error.

**Parameters**

| | | |
|---|---|---|
| `in` | *interval* | The minimum number of screen updates to wait for until the buffers are swapped by glfwSwapBuffers. |

**Remarks**

This function is not called during context creation, leaving the swap interval set to whatever is the default on that platform. This is done because some swap interval extensions used by GLFW do not allow the swap interval to be reset to zero once it has been set to a non-zero value.
Some GPU drivers do not honor the requested swap interval, either because of a user setting that overrides the application's request or due to bugs in the driver.

**Thread Safety**

This function may be called from any thread.

**See also**

buffer_swap
glfwSwapBuffers

**Since**

Added in GLFW 1.0.

## 5.2 Initialization, version and errors

### Modules

- Error codes

### Typedefs

- typedef void(∗ GLFWerrorfun) (int, const char ∗)

  *The function signature for error callbacks.*

- typedef void(∗ GLFWerrorfun) (int, const char ∗)

  *The function signature for error callbacks.*

### Functions

- GLFWAPI int glfwInit (void)

  *Initializes the GLFW library.*
- GLFWAPI void glfwTerminate (void)

  *Terminates the GLFW library.*
- GLFWAPI void glfwGetVersion (int ∗major, int ∗minor, int ∗rev)

  *Retrieves the version of the GLFW library.*
- GLFWAPI const char ∗ glfwGetVersionString (void)

  *Returns a string describing the compile-time configuration.*
- GLFWAPI GLFWerrorfun glfwSetErrorCallback (GLFWerrorfun cbfun)

  *Sets the error callback.*

### GLFW version macros

- #define GLFW_VERSION_MAJOR 3

  *The major version number of the GLFW library.*
- #define GLFW_VERSION_MINOR 1

  *The minor version number of the GLFW library.*
- #define GLFW_VERSION_REVISION 2

  *The revision number of the GLFW library.*

### GLFW version macros

- #define GLFW_VERSION_MAJOR 3

  *The major version number of the GLFW library.*
- #define GLFW_VERSION_MINOR 1

  *The minor version number of the GLFW library.*
- #define GLFW_VERSION_REVISION 2

  *The revision number of the GLFW library.*

### 5.2.1 Detailed Description

This is the reference documentation for initialization and termination of the library, version management and error handling. For more information, see the intro.

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 #define GLFW_VERSION_MAJOR 3

The major version number of the GLFW library.

This is incremented when the API is changed in non-compatible ways.

#### 5.2.2.2 #define GLFW_VERSION_MAJOR 3

The major version number of the GLFW library.

This is incremented when the API is changed in non-compatible ways.

#### 5.2.2.3 #define GLFW_VERSION_MINOR 1

The minor version number of the GLFW library.

This is incremented when features are added to the API but it remains backward-compatible.

#### 5.2.2.4 #define GLFW_VERSION_MINOR 1

The minor version number of the GLFW library.

This is incremented when features are added to the API but it remains backward-compatible.

#### 5.2.2.5 #define GLFW_VERSION_REVISION 2

The revision number of the GLFW library.

This is incremented when a bug fix release is made that does not contain any API changes.

#### 5.2.2.6 #define GLFW_VERSION_REVISION 2

The revision number of the GLFW library.

This is incremented when a bug fix release is made that does not contain any API changes.

### 5.2.3 Typedef Documentation

#### 5.2.3.1 typedef void(∗ GLFWerrorfun) (int, const char ∗)

The function signature for error callbacks.

This is the function signature for error callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *error* | An error code. |
| in | *description* | A UTF-8 encoded string describing the error. |

**See also**

glfwSetErrorCallback

#### 5.2.3.2 typedef void(∗ GLFWerrorfun) (int, const char ∗)

The function signature for error callbacks.

This is the function signature for error callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *error* | An error code. |
| in | *description* | A UTF-8 encoded string describing the error. |

**See also**

glfwSetErrorCallback

### 5.2.4 Function Documentation

#### 5.2.4.1 GLFWAPI void glfwGetVersion ( int ∗ *major,* int ∗ *minor,* int ∗ *rev* )

Retrieves the version of the GLFW library.

This function retrieves the major, minor and revision numbers of the GLFW library. It is intended for when you are using GLFW as a shared library and want to ensure that you are using the minimum required version.

Any or all of the version arguments may be `NULL`. This function always succeeds.

**Parameters**

| | | |
|---|---|---|
| out | *major* | Where to store the major version number, or `NULL`. |
| out | *minor* | Where to store the minor version number, or `NULL`. |
| out | *rev* | Where to store the revision number, or `NULL`. |

**Remarks**

This function may be called before glfwInit.

**Thread Safety**

This function may be called from any thread.

**See also**

intro_version
glfwGetVersionString

**Since**

Added in GLFW 1.0.

#### 5.2.4.2 GLFWAPI const char ∗ glfwGetVersionString ( void )

Returns a string describing the compile-time configuration.

This function returns the compile-time generated version string of the GLFW library binary. It describes the version, platform, compiler and any platform-specific compile-time options.

**Do not use the version string** to parse the GLFW library version. The glfwGetVersion function already provides the version of the running library binary.

This function always succeeds.

**Returns**

The GLFW version string.

**Remarks**

This function may be called before glfwInit.

**Pointer Lifetime**

The returned string is static and compile-time generated.

**Thread Safety**

This function may be called from any thread.

**See also**

intro_version
glfwGetVersion

**Since**

Added in GLFW 3.0.

#### 5.2.4.3 GLFWAPI int glfwInit ( void )

Initializes the GLFW library.

This function initializes the GLFW library. Before most GLFW functions can be used, GLFW must be initialized, and before an application terminates GLFW should be terminated in order to free any resources allocated during or after initialization.

If this function fails, it calls glfwTerminate before returning. If it succeeds, you should call glfwTerminate before the application exits.

Additional calls to this function after successful initialization but before termination will return `GL_TRUE` immediately.

**Returns**

`GL_TRUE` if successful, or `GL_FALSE` if an error occurred.

**Remarks**

**OS X:** This function will change the current directory of the application to the `Contents/Resources` subdirectory of the application's bundle, if present. This can be disabled with a compile-time option.

**Thread Safety**

This function may only be called from the main thread.

**See also**

intro_init
glfwTerminate

**Since**

Added in GLFW 1.0.

#### 5.2.4.4 GLFWAPI GLFWerrorfun glfwSetErrorCallback ( GLFWerrorfun *cbfun* )

Sets the error callback.

This function sets the error callback, which is called with an error code and a human-readable description each time a GLFW error occurs.

The error callback is called on the thread where the error occurred. If you are using GLFW from multiple threads, your error callback needs to be written accordingly.

Because the description string may have been generated specifically for that error, it is not guaranteed to be valid after the callback has returned. If you wish to use it after the callback returns, you need to make a copy.

Once set, the error callback remains set even after the library has been terminated.

**Parameters**

|  |  |  |
|---|---|---|
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set.

**Remarks**

This function may be called before glfwInit.

**Thread Safety**

This function may only be called from the main thread.

**See also**

error_handling

**Since**

Added in GLFW 3.0.

#### 5.2.4.5 GLFWAPI void glfwTerminate ( void )

Terminates the GLFW library.

This function destroys all remaining windows and cursors, restores any modified gamma ramps and frees any other allocated resources. Once this function is called, you must again call glfwInit successfully before you will be able to use most GLFW functions.

If GLFW has been successfully initialized, this function should be called before the application exits. If initialization fails, there is no need to call this function, as it is called by glfwInit before it returns failure.

**Remarks**

This function may be called before glfwInit.

**Warning**

No window's context may be current on another thread when this function is called.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

intro_init
glfwInit

**Since**

Added in GLFW 1.0.

## 5.3 Input handling

### Modules

- Keyboard keys
- Modifier key flags
- Mouse buttons
- Joysticks
- Standard cursor shapes

### Typedefs

- typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

  *The function signature for mouse button callbacks.*
- typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

  *The function signature for cursor position callbacks.*
- typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

  *The function signature for cursor enter/leave callbacks.*
- typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

  *The function signature for scroll callbacks.*
- typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

  *The function signature for keyboard key callbacks.*
- typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

  *The function signature for Unicode character callbacks.*
- typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

  *The function signature for Unicode character with modifiers callbacks.*
- typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

  *The function signature for file drop callbacks.*

- typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

  *The function signature for mouse button callbacks.*
- typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

  *The function signature for cursor position callbacks.*
- typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

  *The function signature for cursor enter/leave callbacks.*
- typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

  *The function signature for scroll callbacks.*
- typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

  *The function signature for keyboard key callbacks.*
- typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

  *The function signature for Unicode character callbacks.*
- typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

  *The function signature for Unicode character with modifiers callbacks.*
- typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

  *The function signature for file drop callbacks.*

## Functions

- GLFWAPI int glfwGetInputMode (GLFWwindow ∗window, int mode)

  *Returns the value of an input option for the specified window.*
- GLFWAPI void glfwSetInputMode (GLFWwindow ∗window, int mode, int value)

  *Sets an input option for the specified window.*
- GLFWAPI int glfwGetKey (GLFWwindow ∗window, int key)

  *Returns the last reported state of a keyboard key for the specified window.*
- GLFWAPI int glfwGetMouseButton (GLFWwindow ∗window, int button)

  *Returns the last reported state of a mouse button for the specified window.*
- GLFWAPI void glfwGetCursorPos (GLFWwindow ∗window, double ∗xpos, double ∗ypos)

  *Retrieves the position of the cursor relative to the client area of the window.*
- GLFWAPI void glfwSetCursorPos (GLFWwindow ∗window, double xpos, double ypos)

  *Sets the position of the cursor, relative to the client area of the window.*
- GLFWAPI GLFWcursor ∗ glfwCreateCursor (const GLFWimage ∗image, int xhot, int yhot)

  *Creates a custom cursor.*
- GLFWAPI GLFWcursor ∗ glfwCreateStandardCursor (int shape)

  *Creates a cursor with a standard shape.*
- GLFWAPI void glfwDestroyCursor (GLFWcursor ∗cursor)

  *Destroys a cursor.*
- GLFWAPI void glfwSetCursor (GLFWwindow ∗window, GLFWcursor ∗cursor)

  *Sets the cursor for the window.*
- GLFWAPI GLFWkeyfun glfwSetKeyCallback (GLFWwindow ∗window, GLFWkeyfun cbfun)

  *Sets the key callback.*
- GLFWAPI GLFWcharfun glfwSetCharCallback (GLFWwindow ∗window, GLFWcharfun cbfun)

  *Sets the Unicode character callback.*
- GLFWAPI GLFWcharmodsfun glfwSetCharModsCallback (GLFWwindow ∗window, GLFWcharmodsfun cbfun)

  *Sets the Unicode character with modifiers callback.*
- GLFWAPI GLFWmousebuttonfun glfwSetMouseButtonCallback (GLFWwindow ∗window, GLFWmousebuttonfun cbfun)

  *Sets the mouse button callback.*
- GLFWAPI GLFWcursorposfun glfwSetCursorPosCallback (GLFWwindow ∗window, GLFWcursorposfun cbfun)

  *Sets the cursor position callback.*
- GLFWAPI GLFWcursorenterfun glfwSetCursorEnterCallback (GLFWwindow ∗window, GLFWcursorenterfun cbfun)

  *Sets the cursor enter/exit callback.*
- GLFWAPI GLFWscrollfun glfwSetScrollCallback (GLFWwindow ∗window, GLFWscrollfun cbfun)

  *Sets the scroll callback.*
- GLFWAPI GLFWdropfun glfwSetDropCallback (GLFWwindow ∗window, GLFWdropfun cbfun)

  *Sets the file drop callback.*
- GLFWAPI int glfwJoystickPresent (int joy)

  *Returns whether the specified joystick is present.*
- GLFWAPI const float ∗ glfwGetJoystickAxes (int joy, int ∗count)

  *Returns the values of all axes of the specified joystick.*
- GLFWAPI const unsigned char ∗ glfwGetJoystickButtons (int joy, int ∗count)

  *Returns the state of all buttons of the specified joystick.*
- GLFWAPI const char ∗ glfwGetJoystickName (int joy)

  *Returns the name of the specified joystick.*
- GLFWAPI void glfwSetClipboardString (GLFWwindow ∗window, const char ∗string)

*Sets the clipboard to the specified string.*
- GLFWAPI const char ∗ glfwGetClipboardString (GLFWwindow ∗window)

  *Returns the contents of the clipboard as a string.*
- GLFWAPI double glfwGetTime (void)

  *Returns the value of the GLFW timer.*
- GLFWAPI void glfwSetTime (double time)

  *Sets the GLFW timer.*

### Key and button actions

- #define GLFW_RELEASE 0

  *The key or mouse button was released.*
- #define GLFW_PRESS 1

  *The key or mouse button was pressed.*
- #define GLFW_REPEAT 2

  *The key was held down until it repeated.*

### Key and button actions

- #define GLFW_RELEASE 0

  *The key or mouse button was released.*
- #define GLFW_PRESS 1

  *The key or mouse button was pressed.*
- #define GLFW_REPEAT 2

  *The key was held down until it repeated.*

## 5.3.1 Detailed Description

This is the reference documentation for input related functions and types. For more information, see the Input handling.

## 5.3.2 Macro Definition Documentation

### 5.3.2.1 #define GLFW_PRESS 1

The key or mouse button was pressed.

The key or mouse button was pressed.

### 5.3.2.2 #define GLFW_PRESS 1

The key or mouse button was pressed.

The key or mouse button was pressed.

#### 5.3.2.3 #define GLFW_RELEASE 0

The key or mouse button was released.

The key or mouse button was released.

#### 5.3.2.4 #define GLFW_RELEASE 0

The key or mouse button was released.

The key or mouse button was released.

#### 5.3.2.5 #define GLFW_REPEAT 2

The key was held down until it repeated.

The key was held down until it repeated.

#### 5.3.2.6 #define GLFW_REPEAT 2

The key was held down until it repeated.

The key was held down until it repeated.

### 5.3.3 Typedef Documentation

#### 5.3.3.1 typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

The function signature for Unicode character callbacks.

This is the function signature for Unicode character callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *codepoint* | The Unicode code point of the character. |

**See also**

glfwSetCharCallback

#### 5.3.3.2 typedef void(∗ GLFWcharfun) (GLFWwindow ∗, unsigned int)

The function signature for Unicode character callbacks.

This is the function signature for Unicode character callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *codepoint* | The Unicode code point of the character. |

**See also**

glfwSetCharCallback

#### 5.3.3.3 typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

The function signature for Unicode character with modifiers callbacks.

This is the function signature for Unicode character with modifiers callback functions. It is called for each input character, regardless of what modifier keys are held down.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *codepoint* | The Unicode code point of the character. |
| `in` | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetCharModsCallback

#### 5.3.3.4 typedef void(∗ GLFWcharmodsfun) (GLFWwindow ∗, unsigned int, int)

The function signature for Unicode character with modifiers callbacks.

This is the function signature for Unicode character with modifiers callback functions. It is called for each input character, regardless of what modifier keys are held down.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *codepoint* | The Unicode code point of the character. |
| `in` | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetCharModsCallback

#### 5.3.3.5 typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

The function signature for cursor enter/leave callbacks.

This is the function signature for cursor enter/leave callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *entered* | `GL_TRUE` if the cursor entered the window's client area, or `GL_FALSE` if it left it. |

**See also**

glfwSetCursorEnterCallback

#### 5.3.3.6 typedef void(∗ GLFWcursorenterfun) (GLFWwindow ∗, int)

The function signature for cursor enter/leave callbacks.

This is the function signature for cursor enter/leave callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *entered* | `GL_TRUE` if the cursor entered the window's client area, or `GL_FALSE` if it left it. |

**See also**

glfwSetCursorEnterCallback

#### 5.3.3.7 typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

The function signature for cursor position callbacks.

This is the function signature for cursor position callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *xpos* | The new x-coordinate, in screen coordinates, of the cursor. |
| in | *ypos* | The new y-coordinate, in screen coordinates, of the cursor. |

**See also**

glfwSetCursorPosCallback

#### 5.3.3.8 typedef void(∗ GLFWcursorposfun) (GLFWwindow ∗, double, double)

The function signature for cursor position callbacks.

This is the function signature for cursor position callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *xpos* | The new x-coordinate, in screen coordinates, of the cursor. |
| in | *ypos* | The new y-coordinate, in screen coordinates, of the cursor. |

**See also**

glfwSetCursorPosCallback

#### 5.3.3.9 typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

The function signature for file drop callbacks.

This is the function signature for file drop callbacks.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *count* | The number of dropped files. |
| in | *paths* | The UTF-8 encoded file and/or directory path names. |

**See also**

glfwSetDropCallback

#### 5.3.3.10 typedef void(∗ GLFWdropfun) (GLFWwindow ∗, int, const char ∗∗)

The function signature for file drop callbacks.

This is the function signature for file drop callbacks.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *count* | The number of dropped files. |
| in | *paths* | The UTF-8 encoded file and/or directory path names. |

**See also**

glfwSetDropCallback

#### 5.3.3.11 typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

The function signature for keyboard key callbacks.

This is the function signature for keyboard key callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *key* | The keyboard key that was pressed or released. |
| in | *scancode* | The system-specific scancode of the key. |
| in | *action* | `GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`. |
| in | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetKeyCallback

#### 5.3.3.12 typedef void(∗ GLFWkeyfun) (GLFWwindow ∗, int, int, int, int)

The function signature for keyboard key callbacks.

This is the function signature for keyboard key callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *key* | The keyboard key that was pressed or released. |
| in | *scancode* | The system-specific scancode of the key. |
| in | *action* | `GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`. |
| in | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetKeyCallback

#### 5.3.3.13 typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

The function signature for mouse button callbacks.

This is the function signature for mouse button callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window that received the event. |
| in | *button* | The mouse button that was pressed or released. |
| in | *action* | One of `GLFW_PRESS` or `GLFW_RELEASE`. |
| in | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetMouseButtonCallback

#### 5.3.3.14 typedef void(∗ GLFWmousebuttonfun) (GLFWwindow ∗, int, int, int)

The function signature for mouse button callbacks.

This is the function signature for mouse button callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *button* | The mouse button that was pressed or released. |
| `in` | *action* | One of `GLFW_PRESS` or `GLFW_RELEASE`. |
| `in` | *mods* | Bit field describing which modifier keys were held down. |

**See also**

glfwSetMouseButtonCallback

#### 5.3.3.15 typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

The function signature for scroll callbacks.

This is the function signature for scroll callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *xoffset* | The scroll offset along the x-axis. |
| `in` | *yoffset* | The scroll offset along the y-axis. |

**See also**

glfwSetScrollCallback

#### 5.3.3.16 typedef void(∗ GLFWscrollfun) (GLFWwindow ∗, double, double)

The function signature for scroll callbacks.

This is the function signature for scroll callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that received the event. |
| `in` | *xoffset* | The scroll offset along the x-axis. |
| `in` | *yoffset* | The scroll offset along the y-axis. |

**See also**

glfwSetScrollCallback

### 5.3.4 Function Documentation

#### 5.3.4.1 GLFWAPI GLFWcursor ∗ glfwCreateCursor ( const GLFWimage ∗ *image*, int *xhot*, int *yhot* )

Creates a custom cursor.

Creates a new custom cursor image that can be set for a window with glfwSetCursor. The cursor can be destroyed with glfwDestroyCursor. Any remaining cursors are destroyed by glfwTerminate.

The pixels are 32-bit, little-endian, non-premultiplied RGBA, i.e. eight bits per channel. They are arranged canonically as packed sequential rows, starting from the top-left corner.

The cursor hotspot is specified in pixels, relative to the upper-left corner of the cursor image. Like all other coordinate systems in GLFW, the X-axis points to the right and the Y-axis points down.

**Parameters**

| | | |
|---|---|---|
| `in` | *image* | The desired cursor image. |
| `in` | *xhot* | The desired x-coordinate, in pixels, of the cursor hotspot. |
| `in` | *yhot* | The desired y-coordinate, in pixels, of the cursor hotspot. |

**Returns**

The handle of the created cursor, or `NULL` if an error occurred.

**Pointer Lifetime**

The specified image data is copied before this function returns.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_object
glfwDestroyCursor
glfwCreateStandardCursor

**Since**

Added in GLFW 3.1.

#### 5.3.4.2 GLFWAPI GLFWcursor ∗ glfwCreateStandardCursor ( int *shape* )

Creates a cursor with a standard shape.

Returns a cursor with a standard shape, that can be set for a window with glfwSetCursor.

**Parameters**

| | | |
|---|---|---|
| `in` | *shape* | One of the standard shapes. |

**Returns**

A new cursor ready to use or `NULL` if an error occurred.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_object
glfwCreateCursor

**Since**

Added in GLFW 3.1.

#### 5.3.4.3 GLFWAPI void glfwDestroyCursor ( GLFWcursor ∗ *cursor* )

Destroys a cursor.

This function destroys a cursor previously created with glfwCreateCursor. Any remaining cursors will be destroyed by glfwTerminate.

**Parameters**

| | | |
|---|---|---|
| `in` | *cursor* | The cursor object to destroy. |

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_object
glfwCreateCursor

**Since**

Added in GLFW 3.1.

#### 5.3.4.4 GLFWAPI const char ∗ glfwGetClipboardString ( GLFWwindow ∗ *window* )

Returns the contents of the clipboard as a string.

This function returns the contents of the system clipboard, if it contains or is convertible to a UTF-8 encoded string. If the clipboard is empty or if its contents cannot be converted, `NULL` is returned and a GLFW_FORMAT_UNAV↩ AILABLE error is generated.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that will request the clipboard contents. |

**Returns**

The contents of the clipboard as a UTF-8 encoded string, or `NULL` if an error occurred.

**Pointer Lifetime**

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the next call to glfwGetClipboardString or glfwSetClipboardString, or until the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

clipboard
glfwSetClipboardString

**Since**

Added in GLFW 3.0.

#### 5.3.4.5 GLFWAPI void glfwGetCursorPos ( GLFWwindow ∗ *window*, double ∗ *xpos*, double ∗ *ypos* )

Retrieves the position of the cursor relative to the client area of the window.

This function returns the position of the cursor, in screen coordinates, relative to the upper-left corner of the client area of the specified window.

If the cursor is disabled (with `GLFW_CURSOR_DISABLED`) then the cursor position is unbounded and limited only by the minimum and maximum values of a `double`.

The coordinate can be converted to their integer equivalents with the `floor` function. Casting directly to an integer type works for positive coordinates, but fails for negative ones.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The desired window. |
| out | *xpos* | Where to store the cursor x-coordinate, relative to the left edge of the client area, or `NULL`. |
| out | *ypos* | Where to store the cursor y-coordinate, relative to the to top edge of the client area, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_pos
glfwSetCursorPos

**Since**

Added in GLFW 3.0. Replaces `glfwGetMousePos`.

#### 5.3.4.6 GLFWAPI int glfwGetInputMode ( GLFWwindow ∗ *window,* int *mode* )

Returns the value of an input option for the specified window.

This function returns the value of an input option for the specified window. The mode must be one of `GLFW_CU↩ RSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window to query. |
| in | *mode* | One of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

glfwSetInputMode

**Since**

Added in GLFW 3.0.

#### 5.3.4.7 GLFWAPI const float ∗ glfwGetJoystickAxes ( int *joy,* int ∗ *count* )

Returns the values of all axes of the specified joystick.

This function returns the values of all axes of the specified joystick. Each element in the array is a value between -1.0 and 1.0.

**Parameters**

| | | |
|---|---|---|
| `in` | *joy* | The joystick to query. |
| `out` | *count* | Where to store the number of axis values in the returned array. This is set to zero if an error occurred. |

**Returns**

An array of axis values, or `NULL` if the joystick is not present.

**Pointer Lifetime**

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

joystick_axis

**Since**

Added in GLFW 3.0. Replaces `glfwGetJoystickPos`.

#### 5.3.4.8 GLFWAPI const unsigned char ∗ glfwGetJoystickButtons ( int *joy*, int ∗ *count* )

Returns the state of all buttons of the specified joystick.

This function returns the state of all buttons of the specified joystick. Each element in the array is either `GLFW_P←RESS` or `GLFW_RELEASE`.

**Parameters**

| | | |
|---|---|---|
| `in` | *joy* | The joystick to query. |
| `out` | *count* | Where to store the number of button states in the returned array. This is set to zero if an error occurred. |

**Returns**

An array of button states, or `NULL` if the joystick is not present.

**Pointer Lifetime**

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

joystick_button

**Since**

Added in GLFW 2.2.

**GLFW 3:** Changed to return a dynamic array.

#### 5.3.4.9 GLFWAPI const char ∗ glfwGetJoystickName ( int *joy* )

Returns the name of the specified joystick.

This function returns the name, encoded as UTF-8, of the specified joystick. The returned string is allocated and freed by GLFW. You should not free it yourself.

**Parameters**

| | | |
|---|---|---|
| `in` | *joy* | The joystick to query. |

**Returns**

The UTF-8 encoded name of the joystick, or `NULL` if the joystick is not present.

**Pointer Lifetime**

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified joystick is disconnected, this function is called again for that joystick or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

joystick_name

**Since**

Added in GLFW 3.0.

#### 5.3.4.10 GLFWAPI int glfwGetKey ( GLFWwindow ∗ *window,* int *key* )

Returns the last reported state of a keyboard key for the specified window.

This function returns the last state reported for the specified key to the specified window. The returned state is one of `GLFW_PRESS` or `GLFW_RELEASE`. The higher-level action `GLFW_REPEAT` is only reported to the key callback.

If the `GLFW_STICKY_KEYS` input mode is enabled, this function returns `GLFW_PRESS` the first time you call it for a key that was pressed, even if that key has already been released.

The key functions deal with physical keys, with key tokens named after their use on the standard US keyboard layout. If you want to input text, use the Unicode character callback instead.

The modifier key bit masks are not key tokens and cannot be used with this function.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The desired window. |
| `in` | *key* | The desired keyboard key. `GLFW_KEY_UNKNOWN` is not a valid key for this function. |

**Returns**

One of `GLFW_PRESS` or `GLFW_RELEASE`.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_key

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.3.4.11 GLFWAPI int glfwGetMouseButton ( GLFWwindow ∗ *window,* int *button* )

Returns the last reported state of a mouse button for the specified window.

This function returns the last state reported for the specified mouse button to the specified window. The returned state is one of `GLFW_PRESS` or `GLFW_RELEASE`.

If the `GLFW_STICKY_MOUSE_BUTTONS` input mode is enabled, this function `GLFW_PRESS` the first time you call it for a mouse button that was pressed, even if that mouse button has already been released.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The desired window. |
| `in` | *button* | The desired mouse button. |

**Returns**

One of `GLFW_PRESS` or `GLFW_RELEASE`.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_mouse_button

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.3.4.12 GLFWAPI double glfwGetTime ( void )

Returns the value of the GLFW timer.

This function returns the value of the GLFW timer. Unless the timer has been set using glfwSetTime, the timer measures time elapsed since GLFW was initialized.

The resolution of the timer is system dependent, but is usually on the order of a few micro- or nanoseconds. It uses the highest-resolution monotonic time source on each supported platform.

**Returns**

The current value, in seconds, or zero if an error occurred.

**Thread Safety**

This function may be called from any thread. Access is not synchronized.

**See also**

time

**Since**

Added in GLFW 1.0.

#### 5.3.4.13 GLFWAPI int glfwJoystickPresent ( int *joy* )

Returns whether the specified joystick is present.

This function returns whether the specified joystick is present.

**Parameters**

| | | |
|---|---|---|
| `in` | *joy* | The joystick to query. |

**Returns**

`GL_TRUE` if the joystick is present, or `GL_FALSE` otherwise.

**Thread Safety**

This function may only be called from the main thread.

**See also**

joystick

**Since**

Added in GLFW 3.0. Replaces `glfwGetJoystickParam`.

#### 5.3.4.14 GLFWAPI GLFWcharfun glfwSetCharCallback ( GLFWwindow $*$ *window,* GLFWcharfun *cbfun* )

Sets the Unicode character callback.

This function sets the character callback of the specified window, which is called when a Unicode character is input.

The character callback is intended for Unicode text input. As it deals with characters, it is keyboard layout dependent, whereas the key callback is not. Characters do not map 1:1 to physical keys, as a key may produce zero, one or more characters. If you want to know whether a specific physical key was pressed or released, see the key callback instead.

The character callback behaves as system text input normally does and will not be called if modifier keys are held down that would prevent normal text input on that platform, for example a Super (Command) key on OS X or Alt key on Windows. There is a character with modifiers callback that receives these events.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_char

**Since**

Added in GLFW 2.4.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.3.4.15 GLFWAPI **GLFWcharmodsfun** glfwSetCharModsCallback ( **GLFWwindow** ∗ *window,* **GLFWcharmodsfun** *cbfun* )

Sets the Unicode character with modifiers callback.

This function sets the character with modifiers callback of the specified window, which is called when a Unicode character is input regardless of what modifier keys are used.

The character with modifiers callback is intended for implementing custom Unicode character input. For regular Unicode text input, see the character callback. Like the character callback, the character with modifiers callback deals with characters and is keyboard layout dependent. Characters do not map 1:1 to physical keys, as a key may produce zero, one or more characters. If you want to know whether a specific physical key was pressed or released, see the key callback instead.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or an error occurred.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_char

**Since**

Added in GLFW 3.1.

#### 5.3.4.16 GLFWAPI void glfwSetClipboardString ( **GLFWwindow** ∗ *window,* const char ∗ *string* )

Sets the clipboard to the specified string.

This function sets the system clipboard to the specified, UTF-8 encoded string.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that will own the clipboard contents. |
| `in` | *string* | A UTF-8 encoded string. |

**Pointer Lifetime**

The specified string is copied before this function returns.

**Thread Safety**

This function may only be called from the main thread.

**See also**

clipboard
glfwGetClipboardString

**Since**

Added in GLFW 3.0.

#### 5.3.4.17 GLFWAPI void glfwSetCursor ( GLFWwindow ∗ *window,* GLFWcursor ∗ *cursor* )

Sets the cursor for the window.

This function sets the cursor image to be used when the cursor is over the client area of the specified window. The set cursor will only be visible when the cursor mode of the window is `GLFW_CURSOR_NORMAL`.

On some platforms, the set cursor may not be visible unless the window also has input focus.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to set the cursor for. |
| `in` | *cursor* | The cursor to set, or `NULL` to switch back to the default arrow cursor. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_object

**Since**

Added in GLFW 3.1.

#### 5.3.4.18 GLFWAPI GLFWcursorenterfun glfwSetCursorEnterCallback ( GLFWwindow ∗ *window,* GLFWcursorenterfun *cbfun* )

Sets the cursor enter/exit callback.

This function sets the cursor boundary crossing callback of the specified window, which is called when the cursor enters or leaves the client area of the window.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_enter

**Since**

Added in GLFW 3.0.

#### 5.3.4.19 GLFWAPI void glfwSetCursorPos ( GLFWwindow ∗ *window,* double *xpos,* double *ypos* )

Sets the position of the cursor, relative to the client area of the window.

This function sets the position, in screen coordinates, of the cursor relative to the upper-left corner of the client area of the specified window. The window must have input focus. If the window does not have input focus when this function is called, it fails silently.

**Do not use this function** to implement things like camera controls. GLFW already provides the `GLFW_CURS`←
`OR_DISABLED` cursor mode that hides the cursor, transparently re-centers it and provides unconstrained cursor motion. See glfwSetInputMode for more information.

If the cursor mode is `GLFW_CURSOR_DISABLED` then the cursor position is unconstrained and limited only by the minimum and maximum values of a `double`.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The desired window. |
| `in` | *xpos* | The desired x-coordinate, relative to the left edge of the client area. |
| `in` | *ypos* | The desired y-coordinate, relative to the top edge of the client area. |

**Remarks**

**X11:** Due to the asynchronous nature of X11, it may take a moment for the window focus event to arrive. This means you may not be able to set the cursor position directly after window creation.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_pos
glfwGetCursorPos

**Since**

Added in GLFW 3.0. Replaces `glfwSetMousePos`.

#### 5.3.4.20 GLFWAPI **GLFWcursorposfun** glfwSetCursorPosCallback ( **GLFWwindow** ∗ *window,* **GLFWcursorposfun** *cbfun* )

Sets the cursor position callback.

This function sets the cursor position callback of the specified window, which is called when the cursor is moved. The callback is provided with the position, in screen coordinates, relative to the upper-left corner of the client area of the window.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

cursor_pos

**Since**

Added in GLFW 3.0. Replaces `glfwSetMousePosCallback`.

#### 5.3.4.21 GLFWAPI **GLFWdropfun** glfwSetDropCallback ( **GLFWwindow** ∗ *window,* **GLFWdropfun** *cbfun* )

Sets the file drop callback.

This function sets the file drop callback of the specified window, which is called when one or more dragged files are dropped on the window.

Because the path array and its strings may have been generated specifically for that event, they are not guaranteed to be valid after the callback has returned. If you wish to use them after the callback returns, you need to make a deep copy.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new file drop callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

path_drop

**Since**

Added in GLFW 3.1.

#### 5.3.4.22 GLFWAPI void glfwSetInputMode ( GLFWwindow ∗ *window,* int *mode,* int *value* )

Sets an input option for the specified window.

This function sets an input mode option for the specified window. The mode must be one of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`.

If the mode is `GLFW_CURSOR`, the value must be one of the following cursor modes:

- `GLFW_CURSOR_NORMAL` makes the cursor visible and behaving normally.
- `GLFW_CURSOR_HIDDEN` makes the cursor invisible when it is over the client area of the window but does not restrict the cursor from leaving.
- `GLFW_CURSOR_DISABLED` hides and grabs the cursor, providing virtual and unlimited cursor movement. This is useful for implementing for example 3D camera controls.

If the mode is `GLFW_STICKY_KEYS`, the value must be either `GL_TRUE` to enable sticky keys, or `GL_FALSE` to disable it. If sticky keys are enabled, a key press will ensure that glfwGetKey returns `GLFW_PRESS` the next time it is called even if the key had been released before the call. This is useful when you are only interested in whether keys have been pressed but not when or in which order.

If the mode is `GLFW_STICKY_MOUSE_BUTTONS`, the value must be either `GL_TRUE` to enable sticky mouse buttons, or `GL_FALSE` to disable it. If sticky mouse buttons are enabled, a mouse button press will ensure that glfwGetMouseButton returns `GLFW_PRESS` the next time it is called even if the mouse button had been released before the call. This is useful when you are only interested in whether mouse buttons have been pressed but not when or in which order.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose input mode to set. |
| in | *mode* | One of `GLFW_CURSOR`, `GLFW_STICKY_KEYS` or `GLFW_STICKY_MOUSE_BUTTONS`. |
| in | *value* | The new value of the specified input mode. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

glfwGetInputMode

**Since**

Added in GLFW 3.0. Replaces `glfwEnable` and `glfwDisable`.

#### 5.3.4.23 GLFWAPI GLFWkeyfun glfwSetKeyCallback ( GLFWwindow ∗ *window*, GLFWkeyfun *cbfun* )

Sets the key callback.

This function sets the key callback of the specified window, which is called when a key is pressed, repeated or released.

The key functions deal with physical keys, with layout independent key tokens named after their values in the standard US keyboard layout. If you want to input text, use the character callback instead.

When a window loses input focus, it will generate synthetic key release events for all pressed keys. You can tell these events from user-generated events by the fact that the synthetic ones are generated after the focus loss event has been processed, i.e. after the window focus callback has been called.

The scancode of a key is specific to that platform or sometimes even to that machine. Scancodes are intended to allow users to bind keys that don't have a GLFW key token. Such keys have `key` set to `GLFW_KEY_UNKNOWN`, their state is not saved and so it cannot be queried with glfwGetKey.

Sometimes GLFW needs to generate synthetic key events, in which case the scancode may be zero.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose callback to set. |
| in | *cbfun* | The new key callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_key

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.3.4.24 GLFWAPI GLFWmousebuttonfun glfwSetMouseButtonCallback ( GLFWwindow ∗ *window,* GLFWmousebuttonfun *cbfun* )

Sets the mouse button callback.

This function sets the mouse button callback of the specified window, which is called when a mouse button is pressed or released.

When a window loses input focus, it will generate synthetic mouse button release events for all pressed mouse buttons. You can tell these events from user-generated events by the fact that the synthetic ones are generated after the focus loss event has been processed, i.e. after the window focus callback has been called.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

input_mouse_button

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.3.4.25 GLFWAPI GLFWscrollfun glfwSetScrollCallback ( GLFWwindow ∗ *window,* GLFWscrollfun *cbfun* )

Sets the scroll callback.

This function sets the scroll callback of the specified window, which is called when a scrolling device is used, such as a mouse wheel or scrolling area of a touchpad.

The scroll callback receives all scrolling input, like that from a mouse wheel or a touchpad scrolling area.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose callback to set. |
| in | *cbfun* | The new scroll callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

scrolling

**Since**

Added in GLFW 3.0. Replaces `glfwSetMouseWheelCallback`.

#### 5.3.4.26 GLFWAPI void glfwSetTime ( double *time* )

Sets the GLFW timer.

This function sets the value of the GLFW timer. It then continues to count up from that value. The value must be a positive finite number less than or equal to 18446744073.0, which is approximately 584.5 years.

**Parameters**

| | | |
|---|---|---|
| in | *time* | The new value, in seconds. |

**Remarks**

The upper limit of the timer is calculated as floor(($2^{64}$ - 1) / $10^9$) and is due to implementations storing nanoseconds in 64 bits. The limit may be increased in the future.

**Thread Safety**

This function may only be called from the main thread.

**See also**

time

**Since**

Added in GLFW 2.2.

## 5.4 Monitor handling

### Classes

- struct GLFWvidmode

  *Video mode type.*
- struct GLFWgammaramp

  *Gamma ramp.*

### Typedefs

- typedef struct GLFWmonitor GLFWmonitor

  *Opaque monitor object.*
- typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

  *The function signature for monitor configuration callbacks.*
- typedef struct GLFWvidmode GLFWvidmode

  *Video mode type.*
- typedef struct GLFWgammaramp GLFWgammaramp

  *Gamma ramp.*
- typedef struct GLFWmonitor GLFWmonitor

  *Opaque monitor object.*
- typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

  *The function signature for monitor configuration callbacks.*
- typedef struct GLFWvidmode GLFWvidmode

  *Video mode type.*
- typedef struct GLFWgammaramp GLFWgammaramp

  *Gamma ramp.*

### Functions

- GLFWAPI GLFWmonitor ∗∗ glfwGetMonitors (int ∗count)

  *Returns the currently connected monitors.*
- GLFWAPI GLFWmonitor ∗ glfwGetPrimaryMonitor (void)

  *Returns the primary monitor.*
- GLFWAPI void glfwGetMonitorPos (GLFWmonitor ∗monitor, int ∗xpos, int ∗ypos)

  *Returns the position of the monitor's viewport on the virtual screen.*
- GLFWAPI void glfwGetMonitorPhysicalSize (GLFWmonitor ∗monitor, int ∗widthMM, int ∗heightMM)

  *Returns the physical size of the monitor.*
- GLFWAPI const char ∗ glfwGetMonitorName (GLFWmonitor ∗monitor)

  *Returns the name of the specified monitor.*
- GLFWAPI GLFWmonitorfun glfwSetMonitorCallback (GLFWmonitorfun cbfun)

  *Sets the monitor configuration callback.*
- GLFWAPI const GLFWvidmode ∗ glfwGetVideoModes (GLFWmonitor ∗monitor, int ∗count)

  *Returns the available video modes for the specified monitor.*
- GLFWAPI const GLFWvidmode ∗ glfwGetVideoMode (GLFWmonitor ∗monitor)

  *Returns the current mode of the specified monitor.*
- GLFWAPI void glfwSetGamma (GLFWmonitor ∗monitor, float gamma)

  *Generates a gamma ramp and sets it for the specified monitor.*
- GLFWAPI const GLFWgammaramp ∗ glfwGetGammaRamp (GLFWmonitor ∗monitor)

  *Returns the current gamma ramp for the specified monitor.*
- GLFWAPI void glfwSetGammaRamp (GLFWmonitor ∗monitor, const GLFWgammaramp ∗ramp)

  *Sets the current gamma ramp for the specified monitor.*

### 5.4.1 Detailed Description

This is the reference documentation for monitor related functions and types. For more information, see the Monitor handling.

### 5.4.2 Typedef Documentation

#### 5.4.2.1 typedef struct GLFWgammaramp GLFWgammaramp

Gamma ramp.

This describes the gamma ramp for a monitor.

**See also**

glfwGetGammaRamp glfwSetGammaRamp

#### 5.4.2.2 typedef struct GLFWgammaramp GLFWgammaramp

Gamma ramp.

This describes the gamma ramp for a monitor.

**See also**

glfwGetGammaRamp glfwSetGammaRamp

#### 5.4.2.3 typedef struct GLFWmonitor GLFWmonitor

Opaque monitor object.

Opaque monitor object.

#### 5.4.2.4 typedef struct GLFWmonitor GLFWmonitor

Opaque monitor object.

Opaque monitor object.

#### 5.4.2.5 typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

The function signature for monitor configuration callbacks.

This is the function signature for monitor configuration callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *monitor* | The monitor that was connected or disconnected. |
| in | *event* | One of `GLFW_CONNECTED` or `GLFW_DISCONNECTED`. |

**See also**

glfwSetMonitorCallback

#### 5.4.2.6 typedef void(∗ GLFWmonitorfun) (GLFWmonitor ∗, int)

The function signature for monitor configuration callbacks.

This is the function signature for monitor configuration callback functions.

**Parameters**

| | | |
|---|---|---|
| in | *monitor* | The monitor that was connected or disconnected. |
| in | *event* | One of `GLFW_CONNECTED` or `GLFW_DISCONNECTED`. |

**See also**

glfwSetMonitorCallback

#### 5.4.2.7 typedef struct GLFWvidmode GLFWvidmode

Video mode type.

This describes a single video mode.

#### 5.4.2.8 typedef struct GLFWvidmode GLFWvidmode

Video mode type.

This describes a single video mode.

### 5.4.3 Function Documentation

#### 5.4.3.1 GLFWAPI const GLFWgammaramp ∗ glfwGetGammaRamp ( GLFWmonitor ∗ *monitor* )

Returns the current gamma ramp for the specified monitor.

This function returns the current gamma ramp of the specified monitor.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |

**Returns**

The current gamma ramp, or `NULL` if an error occurred.

**Pointer Lifetime**

The returned structure and its arrays are allocated and freed by GLFW. You should not free them yourself. They are valid until the specified monitor is disconnected, this function is called again for that monitor or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_gamma

**Since**

Added in GLFW 3.0.

#### 5.4.3.2 GLFWAPI const char ∗ glfwGetMonitorName ( GLFWmonitor ∗ *monitor* )

Returns the name of the specified monitor.

This function returns a human-readable name, encoded as UTF-8, of the specified monitor. The name typically reflects the make and model of the monitor and is not guaranteed to be unique among the connected monitors.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |

**Returns**

The UTF-8 encoded name of the monitor, or `NULL` if an error occurred.

**Pointer Lifetime**

The returned string is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_properties

**Since**

Added in GLFW 3.0.

#### 5.4.3.3 GLFWAPI void glfwGetMonitorPhysicalSize ( GLFWmonitor ∗ *monitor,* int ∗ *widthMM,* int ∗ *heightMM* )

Returns the physical size of the monitor.

This function returns the size, in millimetres, of the display area of the specified monitor.

Some systems do not provide accurate monitor size information, either because the monitor `EDID` data is incorrect or because the driver does not report it accurately.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |
| `out` | *widthMM* | Where to store the width, in millimetres, of the monitor's display area, or `NULL`. |
| `out` | *heightMM* | Where to store the height, in millimetres, of the monitor's display area, or `NULL`. |

**Remarks**

**Windows:** The OS calculates the returned physical size from the current resolution and system DPI instead of querying the monitor EDID data.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_properties

**Since**

Added in GLFW 3.0.

#### 5.4.3.4 GLFWAPI void glfwGetMonitorPos ( GLFWmonitor ∗ *monitor,* int ∗ *xpos,* int ∗ *ypos* )

Returns the position of the monitor's viewport on the virtual screen.

This function returns the position, in screen coordinates, of the upper-left corner of the specified monitor.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |
| `out` | *xpos* | Where to store the monitor x-coordinate, or `NULL`. |
| `out` | *ypos* | Where to store the monitor y-coordinate, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_properties

**Since**

Added in GLFW 3.0.

#### 5.4.3.5 GLFWAPI GLFWmonitor ** glfwGetMonitors ( int * *count* )

Returns the currently connected monitors.

This function returns an array of handles for all currently connected monitors. The primary monitor is always first in the returned array. If no monitors were found, this function returns `NULL`.

**Parameters**

| | | |
|---|---|---|
| `out` | *count* | Where to store the number of monitors in the returned array. This is set to zero if an error occurred. |

**Returns**

An array of monitor handles, or `NULL` if no monitors were found or if an error occurred.

**Pointer Lifetime**

The returned array is allocated and freed by GLFW. You should not free it yourself. It is guaranteed to be valid only until the monitor configuration changes or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_monitors
monitor_event
glfwGetPrimaryMonitor

**Since**

Added in GLFW 3.0.

#### 5.4.3.6 GLFWAPI **GLFWmonitor** ∗ glfwGetPrimaryMonitor ( void )

Returns the primary monitor.

This function returns the primary monitor. This is usually the monitor where elements like the task bar or global menu bar are located.

**Returns**

The primary monitor, or `NULL` if no monitors were found or if an error occurred.

**Thread Safety**

This function may only be called from the main thread.

**Remarks**

The primary monitor is always first in the array returned by glfwGetMonitors.

**See also**

monitor_monitors
glfwGetMonitors

**Since**

Added in GLFW 3.0.

#### 5.4.3.7 GLFWAPI const **GLFWvidmode** ∗ glfwGetVideoMode ( **GLFWmonitor** ∗ *monitor* )

Returns the current mode of the specified monitor.

This function returns the current video mode of the specified monitor. If you have created a full screen window for that monitor, the return value will depend on whether that window is iconified.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |

**Returns**

The current mode of the monitor, or `NULL` if an error occurred.

**Pointer Lifetime**

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_modes
glfwGetVideoModes

**Since**

Added in GLFW 3.0. Replaces `glfwGetDesktopMode`.

#### 5.4.3.8 GLFWAPI const GLFWvidmode ∗ glfwGetVideoModes ( GLFWmonitor ∗ *monitor*, int ∗ *count* )

Returns the available video modes for the specified monitor.

This function returns an array of all video modes supported by the specified monitor. The returned array is sorted in ascending order, first by color bit depth (the sum of all channel depths) and then by resolution area (the product of width and height).

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor to query. |
| `out` | *count* | Where to store the number of video modes in the returned array. This is set to zero if an error occurred. |

**Returns**

An array of video modes, or `NULL` if an error occurred.

**Pointer Lifetime**

The returned array is allocated and freed by GLFW. You should not free it yourself. It is valid until the specified monitor is disconnected, this function is called again for that monitor or the library is terminated.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_modes
glfwGetVideoMode

**Since**

Added in GLFW 1.0.

**GLFW 3:** Changed to return an array of modes for a specific monitor.

#### 5.4.3.9 GLFWAPI void glfwSetGamma ( GLFWmonitor ∗ *monitor*, float *gamma* )

Generates a gamma ramp and sets it for the specified monitor.

This function generates a 256-element gamma ramp from the specified exponent and then calls glfwSetGamma↩ Ramp with it. The value must be a finite number greater than zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor whose gamma ramp to set. |
| `in` | *gamma* | The desired exponent. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_gamma

**Since**

Added in GLFW 3.0.

#### 5.4.3.10 GLFWAPI void glfwSetGammaRamp ( **GLFWmonitor** ∗ *monitor,* const **GLFWgammaramp** ∗ *ramp* )

Sets the current gamma ramp for the specified monitor.

This function sets the current gamma ramp for the specified monitor. The original gamma ramp for that monitor is saved by GLFW the first time this function is called and is restored by glfwTerminate.

**Parameters**

| | | |
|---|---|---|
| `in` | *monitor* | The monitor whose gamma ramp to set. |
| `in` | *ramp* | The gamma ramp to use. |

**Remarks**

Gamma ramp sizes other than 256 are not supported by all platforms or graphics hardware.
**Windows:** The gamma ramp size must be 256.

**Pointer Lifetime**

The specified gamma ramp is copied before this function returns.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_gamma

**Since**

Added in GLFW 3.0.

#### 5.4.3.11 GLFWAPI **GLFWmonitorfun** glfwSetMonitorCallback ( **GLFWmonitorfun** *cbfun* )

Sets the monitor configuration callback.

This function sets the monitor configuration callback, or removes the currently set callback. This is called when a monitor is connected to or disconnected from the system.

**Parameters**

| | | |
|---|---|---|
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Bug** **X11:** This callback is not yet called on monitor configuration changes.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_event

**Since**

Added in GLFW 3.0.

This function sets the monitor configuration callback, or removes the currently set callback. This is called when a monitor is connected to or disconnected from the system.

**Parameters**

| | | |
|---|---|---|
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Bug** **X11:** This callback is not yet called on monitor configuration changes.

**Thread Safety**

This function may only be called from the main thread.

**See also**

monitor_event

**Since**

Added in GLFW 3.0.

## 5.5 Window handling

### Typedefs

- typedef struct GLFWwindow GLFWwindow

  *Opaque window object.*
- typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

  *The function signature for window position callbacks.*
- typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

  *The function signature for window resize callbacks.*
- typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

  *The function signature for window close callbacks.*
- typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

  *The function signature for window content refresh callbacks.*
- typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

  *The function signature for window focus/defocus callbacks.*
- typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

  *The function signature for window iconify/restore callbacks.*
- typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

  *The function signature for framebuffer resize callbacks.*
- typedef struct GLFWwindow GLFWwindow

  *Opaque window object.*
- typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

  *The function signature for window position callbacks.*
- typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

  *The function signature for window resize callbacks.*
- typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

  *The function signature for window close callbacks.*
- typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

  *The function signature for window content refresh callbacks.*
- typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

  *The function signature for window focus/defocus callbacks.*
- typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

  *The function signature for window iconify/restore callbacks.*
- typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

  *The function signature for framebuffer resize callbacks.*

### Functions

- GLFWAPI void glfwDefaultWindowHints (void)

  *Resets all window hints to their default values.*
- GLFWAPI void glfwWindowHint (int target, int hint)

  *Sets the specified window hint to the desired value.*
- GLFWAPI GLFWwindow ∗ glfwCreateWindow (int width, int height, const char ∗title, GLFWmonitor ∗monitor, GLFWwindow ∗share)

  *Creates a window and its associated context.*
- GLFWAPI void glfwDestroyWindow (GLFWwindow ∗window)

  *Destroys the specified window and its context.*
- GLFWAPI int glfwWindowShouldClose (GLFWwindow ∗window)

  *Checks the close flag of the specified window.*

- GLFWAPI void glfwSetWindowShouldClose (GLFWwindow ∗window, int value)

  *Sets the close flag of the specified window.*
- GLFWAPI void glfwSetWindowTitle (GLFWwindow ∗window, const char ∗title)

  *Sets the title of the specified window.*
- GLFWAPI void glfwGetWindowPos (GLFWwindow ∗window, int ∗xpos, int ∗ypos)

  *Retrieves the position of the client area of the specified window.*
- GLFWAPI void glfwSetWindowPos (GLFWwindow ∗window, int xpos, int ypos)

  *Sets the position of the client area of the specified window.*
- GLFWAPI void glfwGetWindowSize (GLFWwindow ∗window, int ∗width, int ∗height)

  *Retrieves the size of the client area of the specified window.*
- GLFWAPI void glfwSetWindowSize (GLFWwindow ∗window, int width, int height)

  *Sets the size of the client area of the specified window.*
- GLFWAPI void glfwGetFramebufferSize (GLFWwindow ∗window, int ∗width, int ∗height)

  *Retrieves the size of the framebuffer of the specified window.*
- GLFWAPI void glfwGetWindowFrameSize (GLFWwindow ∗window, int ∗left, int ∗top, int ∗right, int ∗bottom)

  *Retrieves the size of the frame of the window.*
- GLFWAPI void glfwIconifyWindow (GLFWwindow ∗window)

  *Iconifies the specified window.*
- GLFWAPI void glfwRestoreWindow (GLFWwindow ∗window)

  *Restores the specified window.*
- GLFWAPI void glfwShowWindow (GLFWwindow ∗window)

  *Makes the specified window visible.*
- GLFWAPI void glfwHideWindow (GLFWwindow ∗window)

  *Hides the specified window.*
- GLFWAPI GLFWmonitor ∗ glfwGetWindowMonitor (GLFWwindow ∗window)

  *Returns the monitor that the window uses for full screen mode.*
- GLFWAPI int glfwGetWindowAttrib (GLFWwindow ∗window, int attrib)

  *Returns an attribute of the specified window.*
- GLFWAPI void glfwSetWindowUserPointer (GLFWwindow ∗window, void ∗pointer)

  *Sets the user pointer of the specified window.*
- GLFWAPI void ∗ glfwGetWindowUserPointer (GLFWwindow ∗window)

  *Returns the user pointer of the specified window.*
- GLFWAPI GLFWwindowposfun glfwSetWindowPosCallback (GLFWwindow ∗window, GLFWwindowposfun cbfun)

  *Sets the position callback for the specified window.*
- GLFWAPI GLFWwindowsizefun glfwSetWindowSizeCallback (GLFWwindow ∗window, GLFWwindowsizefun cbfun)

  *Sets the size callback for the specified window.*
- GLFWAPI GLFWwindowclosefun glfwSetWindowCloseCallback (GLFWwindow ∗window, GLFWwindowclosefun cbfun)

  *Sets the close callback for the specified window.*
- GLFWAPI GLFWwindowrefreshfun glfwSetWindowRefreshCallback (GLFWwindow ∗window, GLF↩ Wwindowrefreshfun cbfun)

  *Sets the refresh callback for the specified window.*
- GLFWAPI GLFWwindowfocusfun glfwSetWindowFocusCallback (GLFWwindow ∗window, GLFWwindowfocusfun cbfun)

  *Sets the focus callback for the specified window.*
- GLFWAPI GLFWwindowiconifyfun glfwSetWindowIconifyCallback (GLFWwindow ∗window, GLF↩ Wwindowiconifyfun cbfun)

  *Sets the iconify callback for the specified window.*
- GLFWAPI GLFWframebuffersizefun glfwSetFramebufferSizeCallback (GLFWwindow ∗window, GLF↩ Wframebuffersizefun cbfun)

*Sets the framebuffer resize callback for the specified window.*

- GLFWAPI void glfwPollEvents (void)

    *Processes all pending events.*

- GLFWAPI void glfwWaitEvents (void)

    *Waits until events are queued and processes them.*

- GLFWAPI void glfwPostEmptyEvent (void)

    *Posts an empty event to the event queue.*

- GLFWAPI void glfwSwapBuffers (GLFWwindow ∗window)

    *Swaps the front and back buffers of the specified window.*

### 5.5.1 Detailed Description

This is the reference documentation for window related functions and types, including creation, deletion and event polling. For more information, see the Window handling.

### 5.5.2 Typedef Documentation

#### 5.5.2.1 typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

The function signature for framebuffer resize callbacks.

This is the function signature for framebuffer resize callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose framebuffer was resized. |
| `in` | *width* | The new width, in pixels, of the framebuffer. |
| `in` | *height* | The new height, in pixels, of the framebuffer. |

**See also**

glfwSetFramebufferSizeCallback

#### 5.5.2.2 typedef void(∗ GLFWframebuffersizefun) (GLFWwindow ∗, int, int)

The function signature for framebuffer resize callbacks.

This is the function signature for framebuffer resize callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose framebuffer was resized. |
| `in` | *width* | The new width, in pixels, of the framebuffer. |
| `in` | *height* | The new height, in pixels, of the framebuffer. |

**See also**

glfwSetFramebufferSizeCallback

#### 5.5.2.3 typedef struct GLFWwindow GLFWwindow

Opaque window object.

Opaque window object.

#### 5.5.2.4 typedef struct GLFWwindow GLFWwindow

Opaque window object.

Opaque window object.

#### 5.5.2.5 typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

The function signature for window close callbacks.

This is the function signature for window close callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that the user attempted to close. |

**See also**

glfwSetWindowCloseCallback

#### 5.5.2.6 typedef void(∗ GLFWwindowclosefun) (GLFWwindow ∗)

The function signature for window close callbacks.

This is the function signature for window close callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that the user attempted to close. |

**See also**

glfwSetWindowCloseCallback

#### 5.5.2.7 typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

The function signature for window focus/defocus callbacks.

This is the function signature for window focus callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that gained or lost input focus. |
| `in` | *focused* | `GL_TRUE` if the window was given input focus, or `GL_FALSE` if it lost it. |

**See also**

glfwSetWindowFocusCallback

#### 5.5.2.8 typedef void(∗ GLFWwindowfocusfun) (GLFWwindow ∗, int)

The function signature for window focus/defocus callbacks.

This is the function signature for window focus callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that gained or lost input focus. |
| `in` | *focused* | `GL_TRUE` if the window was given input focus, or `GL_FALSE` if it lost it. |

**See also**

glfwSetWindowFocusCallback

#### 5.5.2.9 typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

The function signature for window iconify/restore callbacks.

This is the function signature for window iconify/restore callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was iconified or restored. |
| `in` | *iconified* | `GL_TRUE` if the window was iconified, or `GL_FALSE` if it was restored. |

**See also**

glfwSetWindowIconifyCallback

#### 5.5.2.10 typedef void(∗ GLFWwindowiconifyfun) (GLFWwindow ∗, int)

The function signature for window iconify/restore callbacks.

This is the function signature for window iconify/restore callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was iconified or restored. |
| `in` | *iconified* | `GL_TRUE` if the window was iconified, or `GL_FALSE` if it was restored. |

**See also**

glfwSetWindowIconifyCallback

#### 5.5.2.11 typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

The function signature for window position callbacks.

This is the function signature for window position callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was moved. |
| `in` | *xpos* | The new x-coordinate, in screen coordinates, of the upper-left corner of the client area of the window. |
| `in` | *ypos* | The new y-coordinate, in screen coordinates, of the upper-left corner of the client area of the window. |

**See also**

glfwSetWindowPosCallback

#### 5.5.2.12 typedef void(∗ GLFWwindowposfun) (GLFWwindow ∗, int, int)

The function signature for window position callbacks.

This is the function signature for window position callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was moved. |
| `in` | *xpos* | The new x-coordinate, in screen coordinates, of the upper-left corner of the client area of the window. |
| `in` | *ypos* | The new y-coordinate, in screen coordinates, of the upper-left corner of the client area of the window. |

**See also**

glfwSetWindowPosCallback

### 5.5.2.13 typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

The function signature for window content refresh callbacks.

This is the function signature for window refresh callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose content needs to be refreshed. |

**See also**

glfwSetWindowRefreshCallback

### 5.5.2.14 typedef void(∗ GLFWwindowrefreshfun) (GLFWwindow ∗)

The function signature for window content refresh callbacks.

This is the function signature for window refresh callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose content needs to be refreshed. |

**See also**

glfwSetWindowRefreshCallback

### 5.5.2.15 typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

The function signature for window resize callbacks.

This is the function signature for window size callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was resized. |
| `in` | *width* | The new width, in screen coordinates, of the window. |
| `in` | *height* | The new height, in screen coordinates, of the window. |

**See also**

glfwSetWindowSizeCallback

#### 5.5.2.16 typedef void(∗ GLFWwindowsizefun) (GLFWwindow ∗, int, int)

The function signature for window resize callbacks.

This is the function signature for window size callback functions.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window that was resized. |
| `in` | *width* | The new width, in screen coordinates, of the window. |
| `in` | *height* | The new height, in screen coordinates, of the window. |

**See also**

glfwSetWindowSizeCallback

### 5.5.3 Function Documentation

#### 5.5.3.1 GLFWAPI GLFWwindow ∗ glfwCreateWindow ( int *width,* int *height,* const char ∗ *title,* GLFWmonitor ∗ *monitor,* GLFWwindow ∗ *share* )

Creates a window and its associated context.

This function creates a window and its associated OpenGL or OpenGL ES context. Most of the options controlling how the window and its context should be created are specified with window hints.

Successful creation does not change which context is current. Before you can use the newly created context, you need to make it current. For information about the `share` parameter, see context_sharing.

The created window, framebuffer and context may differ from what you requested, as not all parameters and hints are hard constraints. This includes the size of the window, especially for full screen windows. To query the actual attributes of the created window, framebuffer and context, see glfwGetWindowAttrib, glfwGetWindowSize and glfw←GetFramebufferSize.

To create a full screen window, you need to specify the monitor the window will cover. If no monitor is specified, windowed mode will be used. Unless you have a way for the user to choose a specific monitor, it is recommended that you pick the primary monitor. For more information on how to query connected monitors, see monitor_monitors.

For full screen windows, the specified size becomes the resolution of the window's *desired video mode*. As long as a full screen window has input focus, the supported video mode most closely matching the desired video mode is set for the specified monitor. For more information about full screen windows, including the creation of so called *windowed full screen* or *borderless full screen* windows, see window_windowed_full_screen.

By default, newly created windows use the placement recommended by the window system. To create the window at a specific position, make it initially invisible using the GLFW_VISIBLE window hint, set its position and then show it.

If a full screen window has input focus, the screensaver is prohibited from starting.

Window systems put limits on window sizes. Very large or very small window dimensions may be overridden by the window system on creation. Check the actual size after creation.

The swap interval is not set during window creation and the initial value may vary depending on driver settings and defaults.

**Parameters**

| | | |
|---|---|---|
| in | *width* | The desired width, in screen coordinates, of the window. This must be greater than zero. |
| in | *height* | The desired height, in screen coordinates, of the window. This must be greater than zero. |
| in | *title* | The initial, UTF-8 encoded window title. |
| in | *monitor* | The monitor to use for full screen mode, or `NULL` to use windowed mode. |
| in | *share* | The window whose context to share resources with, or `NULL` to not share resources. |

**Returns**

The handle of the created window, or `NULL` if an error occurred.

**Remarks**

**Windows:** Window creation will fail if the Microsoft GDI software OpenGL implementation is the only one available.
**Windows:** If the executable has an icon resource named `GLFW_ICON,` it will be set as the icon for the window. If no such icon is present, the `IDI_WINLOGO` icon will be used instead.
**Windows:** The context to share resources with may not be current on any other thread.
**OS X:** The GLFW window has no icon, as it is not a document window, but the dock icon will be the same as the application bundle's icon. For more information on bundles, see the `Bundle Programming Guide` in the Mac Developer Library.
**OS X:** The first time a window is created the menu bar is populated with common commands like Hide, Quit and About. The About entry opens a minimal about dialog with information from the application's bundle. The menu bar can be disabled with a compile-time option.
**OS X:** On OS X 10.10 and later the window frame will not be rendered at full resolution on Retina displays unless the `NSHighResolutionCapable` key is enabled in the application bundle's `Info.plist`. For more information, see `High Resolution Guidelines for OS X` in the Mac Developer Library. The GLFW test and example programs use a custom `Info.plist` template for this, which can be found as `C↩ Make/MacOSXBundleInfo.plist.in` in the source tree.
**X11:** There is no mechanism for setting the window icon yet.
**X11:** Some window managers will not respect the placement of initially hidden windows.
**X11:** Due to the asynchronous nature of X11, it may take a moment for a window to reach its requested state. This means you may not be able to query the final size, position or other attributes directly after window creation.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_creation
glfwDestroyWindow

**Since**

Added in GLFW 3.0. Replaces `glfwOpenWindow`.

#### 5.5.3.2 GLFWAPI void glfwDefaultWindowHints ( void )

Resets all window hints to their default values.

This function resets all window hints to their default values.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_hints
glfwWindowHint

**Since**

Added in GLFW 3.0.

#### 5.5.3.3 GLFWAPI void glfwDestroyWindow ( GLFWwindow * *window* )

Destroys the specified window and its context.

This function destroys the specified window and its context. On calling this function, no further callbacks will be called for that window.

If the context of the specified window is current on the main thread, it is detached before being destroyed.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to destroy. |

**Note**

The context of the specified window must not be current on any other thread when this function is called.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_creation
glfwCreateWindow

**Since**

Added in GLFW 3.0. Replaces `glfwCloseWindow`.

#### 5.5.3.4 GLFWAPI void glfwGetFramebufferSize ( GLFWwindow ∗ *window,* int ∗ *width,* int ∗ *height* )

Retrieves the size of the framebuffer of the specified window.

This function retrieves the size, in pixels, of the framebuffer of the specified window. If you wish to retrieve the size of the window in screen coordinates, see glfwGetWindowSize.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose framebuffer to query. |
| `out` | *width* | Where to store the width, in pixels, of the framebuffer, or `NULL`. |
| `out` | *height* | Where to store the height, in pixels, of the framebuffer, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_fbsize
glfwSetFramebufferSizeCallback

**Since**

Added in GLFW 3.0.

#### 5.5.3.5 GLFWAPI int glfwGetWindowAttrib ( GLFWwindow ∗ *window,* int *attrib* )

Returns an attribute of the specified window.

This function returns the value of an attribute of the specified window or its OpenGL or OpenGL ES context.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to query. |
| `in` | *attrib* | The window attribute whose value to return. |

**Returns**

The value of the attribute, or zero if an error occurred.

**Remarks**

Framebuffer related hints are not window attributes. See window_attribs_fb for more information.
Zero is a valid value for many window and context related attributes so you cannot use a return value of zero as an indication of errors. However, this function should not fail as long as it is passed valid arguments and the library has been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_attribs

**Since**

Added in GLFW 3.0. Replaces `glfwGetWindowParam` and `glfwGetGLVersion`.

#### 5.5.3.6 GLFWAPI void glfwGetWindowFrameSize ( GLFWwindow ∗ *window*, int ∗ *left*, int ∗ *top*, int ∗ *right*, int ∗ *bottom* )

Retrieves the size of the frame of the window.

This function retrieves the size, in screen coordinates, of each edge of the frame of the specified window. This size includes the title bar, if the window has one. The size of the frame may vary depending on the window-related hints used to create it.

Because this function retrieves the size of each window frame edge and not the offset along a particular coordinate axis, the retrieved values will always be zero or positive.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose frame size to query. |
| `out` | *left* | Where to store the size, in screen coordinates, of the left edge of the window frame, or `NULL`. |
| `out` | *top* | Where to store the size, in screen coordinates, of the top edge of the window frame, or `NULL`. |
| `out` | *right* | Where to store the size, in screen coordinates, of the right edge of the window frame, or `NULL`. |
| `out` | *bottom* | Where to store the size, in screen coordinates, of the bottom edge of the window frame, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_size

**Since**

Added in GLFW 3.1.

#### 5.5.3.7 GLFWAPI GLFWmonitor ∗ glfwGetWindowMonitor ( GLFWwindow ∗ *window* )

Returns the monitor that the window uses for full screen mode.

This function returns the handle of the monitor that the specified window is in full screen on.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to query. |

**Returns**

The monitor, or `NULL` if the window is in windowed mode or an error occurred.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_monitor

**Since**

Added in GLFW 3.0.

#### 5.5.3.8 GLFWAPI void glfwGetWindowPos ( GLFWwindow ∗ *window*, int ∗ *xpos*, int ∗ *ypos* )

Retrieves the position of the client area of the specified window.

This function retrieves the position, in screen coordinates, of the upper-left corner of the client area of the specified window.

Any or all of the position arguments may be `NULL`. If an error occurs, all non-`NULL` position arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to query. |
| `out` | *xpos* | Where to store the x-coordinate of the upper-left corner of the client area, or `NULL`. |
| `out` | *ypos* | Where to store the y-coordinate of the upper-left corner of the client area, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_pos
glfwSetWindowPos

**Since**

Added in GLFW 3.0.

#### 5.5.3.9 GLFWAPI void glfwGetWindowSize ( GLFWwindow ∗ *window,* int ∗ *width,* int ∗ *height* )

Retrieves the size of the client area of the specified window.

This function retrieves the size, in screen coordinates, of the client area of the specified window. If you wish to retrieve the size of the framebuffer of the window in pixels, see glfwGetFramebufferSize.

Any or all of the size arguments may be `NULL`. If an error occurs, all non-`NULL` size arguments will be set to zero.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose size to retrieve. |
| `out` | *width* | Where to store the width, in screen coordinates, of the client area, or `NULL`. |
| `out` | *height* | Where to store the height, in screen coordinates, of the client area, or `NULL`. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_size
glfwSetWindowSize

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.10 GLFWAPI void ∗ glfwGetWindowUserPointer ( GLFWwindow ∗ *window* )

Returns the user pointer of the specified window.

This function returns the current value of the user-defined pointer of the specified window. The initial value is `NULL`.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose pointer to return. |

**Thread Safety**

This function may be called from any thread. Access is not synchronized.

**See also**

window_userptr
glfwSetWindowUserPointer

**Since**

Added in GLFW 3.0.

#### 5.5.3.11 GLFWAPI void glfwHideWindow ( GLFWwindow ∗ *window* )

Hides the specified window.

This function hides the specified window if it was previously visible. If the window is already hidden or is in full screen mode, this function does nothing.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window to hide. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_hide
glfwShowWindow

**Since**

Added in GLFW 3.0.

#### 5.5.3.12 GLFWAPI void glfwIconifyWindow ( GLFWwindow ∗ *window* )

Iconifies the specified window.

This function iconifies (minimizes) the specified window if it was previously restored. If the window is already iconified, this function does nothing.

If the specified window is a full screen window, the original monitor resolution is restored until the window is restored.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window to iconify. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_iconify
glfwRestoreWindow

**Since**

Added in GLFW 2.1.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.13 GLFWAPI void glfwPollEvents ( void )

Processes all pending events.

This function processes only those events that are already in the event queue and then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.

On some platforms, a window move, resize or menu operation will cause event processing to block. This is due to how event processing is designed on those platforms. You can use the window refresh callback to redraw the contents of your window when necessary during such operations.

On some platforms, certain events are sent directly to the application without going through the event queue, causing callbacks to be called outside of a call to one of the event processing functions.

Event processing is not required for joystick input to work.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

events
glfwWaitEvents

**Since**

Added in GLFW 1.0.

#### 5.5.3.14 GLFWAPI void glfwPostEmptyEvent ( void )

Posts an empty event to the event queue.

This function posts an empty event from the current thread to the event queue, causing glfwWaitEvents to return.

If no windows exist, this function returns immediately. For synchronization of threads in applications that do not create windows, use your threading library of choice.

**Thread Safety**

This function may be called from any thread.

**See also**

events
glfwWaitEvents

**Since**

Added in GLFW 3.1.

#### 5.5.3.15 GLFWAPI void glfwRestoreWindow ( GLFWwindow ∗ *window* )

Restores the specified window.

This function restores the specified window if it was previously iconified (minimized). If the window is already restored, this function does nothing.

If the specified window is a full screen window, the resolution chosen for the window is restored on the selected monitor.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to restore. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_iconify
glfwIconifyWindow

**Since**

Added in GLFW 2.1.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.16 GLFWAPI GLFWframebuffersizefun glfwSetFramebufferSizeCallback ( GLFWwindow ∗ *window,* GLFWframebuffersizefun *cbfun* )

Sets the framebuffer resize callback for the specified window.

This function sets the framebuffer resize callback of the specified window, which is called when the framebuffer of the specified window is resized.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_fbsize

**Since**

Added in GLFW 3.0.

#### 5.5.3.17 GLFWAPI GLFWwindowclosefun glfwSetWindowCloseCallback ( GLFWwindow ∗ *window,* GLFWwindowclosefun *cbfun* )

Sets the close callback for the specified window.

This function sets the close callback of the specified window, which is called when the user attempts to close the window, for example by clicking the close widget in the title bar.

The close flag is set before this callback is called, but you can modify it at any time with glfwSetWindowShouldClose.

The close callback is not triggered by glfwDestroyWindow.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Remarks**

**OS X:** Selecting Quit from the application menu will trigger the close callback for all windows.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_close

**Since**

Added in GLFW 2.5.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.5.3.18 GLFWAPI GLFWwindowfocusfun glfwSetWindowFocusCallback ( GLFWwindow ∗ *window*, GLFWwindowfocusfun *cbfun* )

Sets the focus callback for the specified window.

This function sets the focus callback of the specified window, which is called when the window gains or loses input focus.

After the focus callback is called for a window that lost input focus, synthetic key and mouse button release events will be generated for all such that had been pressed. For more information, see glfwSetKeyCallback and glfwSet↩ MouseButtonCallback.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_focus

**Since**

Added in GLFW 3.0.

#### 5.5.3.19 GLFWAPI GLFWwindowiconifyfun glfwSetWindowIconifyCallback ( GLFWwindow ∗ *window,* GLFWwindowiconifyfun *cbfun* )

Sets the iconify callback for the specified window.

This function sets the iconification callback of the specified window, which is called when the window is iconified or restored.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose callback to set. |
| in | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_iconify

**Since**

Added in GLFW 3.0.

#### 5.5.3.20 GLFWAPI void glfwSetWindowPos ( GLFWwindow ∗ *window,* int *xpos,* int *ypos* )

Sets the position of the client area of the specified window.

This function sets the position, in screen coordinates, of the upper-left corner of the client area of the specified windowed mode window. If the window is a full screen window, this function does nothing.

**Do not use this function** to move an already visible window unless you have very good reasons for doing so, as it will confuse and annoy the user.

The window manager may put limits on what positions are allowed. GLFW cannot and should not override these limits.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window to query. |
| in | *xpos* | The x-coordinate of the upper-left corner of the client area. |
| in | *ypos* | The y-coordinate of the upper-left corner of the client area. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_pos
glfwGetWindowPos

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.21 GLFWAPI GLFWwindowposfun glfwSetWindowPosCallback ( GLFWwindow ∗ *window*, GLFWwindowposfun *cbfun* )

Sets the position callback for the specified window.

This function sets the position callback of the specified window, which is called when the window is moved. The callback is provided with the screen position of the upper-left corner of the client area of the window.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose callback to set. |
| in | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_pos

**Since**

Added in GLFW 3.0.

#### 5.5.3.22 GLFWAPI GLFWwindowrefreshfun glfwSetWindowRefreshCallback ( GLFWwindow ∗ *window*, GLFWwindowrefreshfun *cbfun* )

Sets the refresh callback for the specified window.

This function sets the refresh callback of the specified window, which is called when the client area of the window needs to be redrawn, for example if the window has been exposed after having been covered by another window.

On compositing window systems such as Aero, Compiz or Aqua, where the window contents are saved off-screen, this callback may be called only very infrequently or never at all.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_refresh

**Since**

Added in GLFW 2.5.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.5.3.23 GLFWAPI void glfwSetWindowShouldClose ( GLFWwindow ∗ *window,* int *value* )

Sets the close flag of the specified window.

This function sets the value of the close flag of the specified window. This can be used to override the user's attempt to close the window, or to signal that it should be closed.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose flag to change. |
| `in` | *value* | The new value. |

**Thread Safety**

This function may be called from any thread. Access is not synchronized.

**See also**

window_close

**Since**

Added in GLFW 3.0.

#### 5.5.3.24 GLFWAPI void glfwSetWindowSize ( GLFWwindow ∗ *window,* int *width,* int *height* )

Sets the size of the client area of the specified window.

This function sets the size, in screen coordinates, of the client area of the specified window.

For full screen windows, this function selects and switches to the resolution closest to the specified size, without affecting the window's context. As the context is unaffected, the bit depths of the framebuffer remain unchanged.

The window manager may put limits on what sizes are allowed. GLFW cannot and should not override these limits.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to resize. |
| `in` | *width* | The desired width of the specified window. |
| `in` | *height* | The desired height of the specified window. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_size
glfwGetWindowSize

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.25 GLFWAPI GLFWwindowsizefun glfwSetWindowSizeCallback ( GLFWwindow ∗ *window,* GLFWwindowsizefun *cbfun* )

Sets the size callback for the specified window.

This function sets the size callback of the specified window, which is called when the window is resized. The callback is provided with the size, in screen coordinates, of the client area of the window.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose callback to set. |
| `in` | *cbfun* | The new callback, or `NULL` to remove the currently set callback. |

**Returns**

The previously set callback, or `NULL` if no callback was set or the library had not been initialized.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_size

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter. Updated callback signature.

#### 5.5.3.26 GLFWAPI void glfwSetWindowTitle ( GLFWwindow ∗ *window,* const char ∗ *title* )

Sets the title of the specified window.

This function sets the window title, encoded as UTF-8, of the specified window.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window whose title to change. |
| `in` | *title* | The UTF-8 encoded window title. |

**Remarks**

**OS X:** The window title will not be updated until the next time you process events.

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_title

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.27 GLFWAPI void glfwSetWindowUserPointer ( GLFWwindow ∗ *window,* void ∗ *pointer* )

Sets the user pointer of the specified window.

This function sets the user-defined pointer of the specified window. The current value is retained until the window is destroyed. The initial value is `NULL`.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose pointer to set. |
| in | *pointer* | The new value. |

**Thread Safety**

This function may be called from any thread. Access is not synchronized.

**See also**

window_userptr
glfwGetWindowUserPointer

**Since**

Added in GLFW 3.0.

#### 5.5.3.28 GLFWAPI void glfwShowWindow ( GLFWwindow ∗ *window* )

Makes the specified window visible.

This function makes the specified window visible if it was previously hidden. If the window is already visible or is in full screen mode, this function does nothing.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window to make visible. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_hide
glfwHideWindow

**Since**

Added in GLFW 3.0.

#### 5.5.3.29 GLFWAPI void glfwSwapBuffers ( GLFWwindow ∗ *window* )

Swaps the front and back buffers of the specified window.

This function swaps the front and back buffers of the specified window. If the swap interval is greater than zero, the GPU driver waits the specified number of screen updates before swapping the buffers.

**Parameters**

| | | |
|---|---|---|
| in | *window* | The window whose buffers to swap. |

**Thread Safety**

This function may be called from any thread.

**See also**

buffer_swap
glfwSwapInterval

**Since**

Added in GLFW 1.0.

**GLFW 3:** Added window handle parameter.

#### 5.5.3.30 GLFWAPI void glfwWaitEvents ( void )

Waits until events are queued and processes them.

This function puts the calling thread to sleep until at least one event is available in the event queue. Once one or more events are available, it behaves exactly like glfwPollEvents, i.e. the events in the queue are processed and the function then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.

Since not all events are associated with callbacks, this function may return without a callback having been called even if you are monitoring all callbacks.

On some platforms, a window move, resize or menu operation will cause event processing to block. This is due to how event processing is designed on those platforms. You can use the window refresh callback to redraw the contents of your window when necessary during such operations.

On some platforms, certain callbacks may be called outside of a call to one of the event processing functions.

If no windows exist, this function returns immediately. For synchronization of threads in applications that do not create windows, use your threading library of choice.

Event processing is not required for joystick input to work.

**Reentrancy**

This function may not be called from a callback.

**Thread Safety**

This function may only be called from the main thread.

**See also**

events
glfwPollEvents

**Since**

Added in GLFW 2.5.

#### 5.5.3.31 GLFWAPI void glfwWindowHint ( int *target,* int *hint* )

Sets the specified window hint to the desired value.

This function sets hints for the next call to glfwCreateWindow. The hints, once set, retain their values until changed by a call to glfwWindowHint or glfwDefaultWindowHints, or until the library is terminated.

**Parameters**

| | | |
|---|---|---|
| `in` | *target* | The window hint to set. |
| `in` | *hint* | The new value of the window hint. |

**Thread Safety**

This function may only be called from the main thread.

**See also**

window_hints
glfwDefaultWindowHints

**Since**

Added in GLFW 3.0. Replaces `glfwOpenWindowHint`.

#### 5.5.3.32 GLFWAPI int glfwWindowShouldClose ( GLFWwindow * *window* )

Checks the close flag of the specified window.

This function returns the value of the close flag of the specified window.

**Parameters**

| | | |
|---|---|---|
| `in` | *window* | The window to query. |

**Returns**

The value of the close flag.

**Thread Safety**

This function may be called from any thread. Access is not synchronized.

**See also**

window_close

**Since**

Added in GLFW 3.0.

## 5.6 Keyboard keys

### Macros

- #define **GLFW_KEY_UNKNOWN** -1
- #define **GLFW_KEY_SPACE** 32
- #define **GLFW_KEY_APOSTROPHE** 39 /∗ ' ∗/
- #define **GLFW_KEY_COMMA** 44 /∗ , ∗/
- #define **GLFW_KEY_MINUS** 45 /∗ - ∗/
- #define **GLFW_KEY_PERIOD** 46 /∗ . ∗/
- #define **GLFW_KEY_SLASH** 47 /∗ / ∗/
- #define **GLFW_KEY_0** 48
- #define **GLFW_KEY_1** 49
- #define **GLFW_KEY_2** 50
- #define **GLFW_KEY_3** 51
- #define **GLFW_KEY_4** 52
- #define **GLFW_KEY_5** 53
- #define **GLFW_KEY_6** 54
- #define **GLFW_KEY_7** 55
- #define **GLFW_KEY_8** 56
- #define **GLFW_KEY_9** 57
- #define **GLFW_KEY_SEMICOLON** 59 /∗ ; ∗/
- #define **GLFW_KEY_EQUAL** 61 /∗ = ∗/
- #define **GLFW_KEY_A** 65
- #define **GLFW_KEY_B** 66
- #define **GLFW_KEY_C** 67
- #define **GLFW_KEY_D** 68
- #define **GLFW_KEY_E** 69
- #define **GLFW_KEY_F** 70
- #define **GLFW_KEY_G** 71
- #define **GLFW_KEY_H** 72
- #define **GLFW_KEY_I** 73
- #define **GLFW_KEY_J** 74
- #define **GLFW_KEY_K** 75
- #define **GLFW_KEY_L** 76
- #define **GLFW_KEY_M** 77
- #define **GLFW_KEY_N** 78
- #define **GLFW_KEY_O** 79
- #define **GLFW_KEY_P** 80
- #define **GLFW_KEY_Q** 81
- #define **GLFW_KEY_R** 82
- #define **GLFW_KEY_S** 83
- #define **GLFW_KEY_T** 84
- #define **GLFW_KEY_U** 85
- #define **GLFW_KEY_V** 86
- #define **GLFW_KEY_W** 87
- #define **GLFW_KEY_X** 88
- #define **GLFW_KEY_Y** 89
- #define **GLFW_KEY_Z** 90
- #define **GLFW_KEY_LEFT_BRACKET** 91 /∗ [ ∗/
- #define **GLFW_KEY_BACKSLASH** 92 /∗ \ ∗/
- #define **GLFW_KEY_RIGHT_BRACKET** 93 /∗ ] ∗/
- #define **GLFW_KEY_GRAVE_ACCENT** 96 /∗ ‘ ∗/
- #define **GLFW_KEY_WORLD_1** 161 /∗ non-US #1 ∗/

- #define **GLFW_KEY_WORLD_2** 162 /∗ non-US #2 ∗/
- #define **GLFW_KEY_ESCAPE** 256
- #define **GLFW_KEY_ENTER** 257
- #define **GLFW_KEY_TAB** 258
- #define **GLFW_KEY_BACKSPACE** 259
- #define **GLFW_KEY_INSERT** 260
- #define **GLFW_KEY_DELETE** 261
- #define **GLFW_KEY_RIGHT** 262
- #define **GLFW_KEY_LEFT** 263
- #define **GLFW_KEY_DOWN** 264
- #define **GLFW_KEY_UP** 265
- #define **GLFW_KEY_PAGE_UP** 266
- #define **GLFW_KEY_PAGE_DOWN** 267
- #define **GLFW_KEY_HOME** 268
- #define **GLFW_KEY_END** 269
- #define **GLFW_KEY_CAPS_LOCK** 280
- #define **GLFW_KEY_SCROLL_LOCK** 281
- #define **GLFW_KEY_NUM_LOCK** 282
- #define **GLFW_KEY_PRINT_SCREEN** 283
- #define **GLFW_KEY_PAUSE** 284
- #define **GLFW_KEY_F1** 290
- #define **GLFW_KEY_F2** 291
- #define **GLFW_KEY_F3** 292
- #define **GLFW_KEY_F4** 293
- #define **GLFW_KEY_F5** 294
- #define **GLFW_KEY_F6** 295
- #define **GLFW_KEY_F7** 296
- #define **GLFW_KEY_F8** 297
- #define **GLFW_KEY_F9** 298
- #define **GLFW_KEY_F10** 299
- #define **GLFW_KEY_F11** 300
- #define **GLFW_KEY_F12** 301
- #define **GLFW_KEY_F13** 302
- #define **GLFW_KEY_F14** 303
- #define **GLFW_KEY_F15** 304
- #define **GLFW_KEY_F16** 305
- #define **GLFW_KEY_F17** 306
- #define **GLFW_KEY_F18** 307
- #define **GLFW_KEY_F19** 308
- #define **GLFW_KEY_F20** 309
- #define **GLFW_KEY_F21** 310
- #define **GLFW_KEY_F22** 311
- #define **GLFW_KEY_F23** 312
- #define **GLFW_KEY_F24** 313
- #define **GLFW_KEY_F25** 314
- #define **GLFW_KEY_KP_0** 320
- #define **GLFW_KEY_KP_1** 321
- #define **GLFW_KEY_KP_2** 322
- #define **GLFW_KEY_KP_3** 323
- #define **GLFW_KEY_KP_4** 324
- #define **GLFW_KEY_KP_5** 325
- #define **GLFW_KEY_KP_6** 326
- #define **GLFW_KEY_KP_7** 327
- #define **GLFW_KEY_KP_8** 328
- #define **GLFW_KEY_KP_9** 329

- #define **GLFW_KEY_KP_DECIMAL** 330
- #define **GLFW_KEY_KP_DIVIDE** 331
- #define **GLFW_KEY_KP_MULTIPLY** 332
- #define **GLFW_KEY_KP_SUBTRACT** 333
- #define **GLFW_KEY_KP_ADD** 334
- #define **GLFW_KEY_KP_ENTER** 335
- #define **GLFW_KEY_KP_EQUAL** 336
- #define **GLFW_KEY_LEFT_SHIFT** 340
- #define **GLFW_KEY_LEFT_CONTROL** 341
- #define **GLFW_KEY_LEFT_ALT** 342
- #define **GLFW_KEY_LEFT_SUPER** 343
- #define **GLFW_KEY_RIGHT_SHIFT** 344
- #define **GLFW_KEY_RIGHT_CONTROL** 345
- #define **GLFW_KEY_RIGHT_ALT** 346
- #define **GLFW_KEY_RIGHT_SUPER** 347
- #define **GLFW_KEY_MENU** 348
- #define **GLFW_KEY_LAST** GLFW_KEY_MENU
- #define **GLFW_KEY_UNKNOWN** -1
- #define **GLFW_KEY_SPACE** 32
- #define **GLFW_KEY_APOSTROPHE** 39 /∗ ' ∗/
- #define **GLFW_KEY_COMMA** 44 /∗ , ∗/
- #define **GLFW_KEY_MINUS** 45 /∗ - ∗/
- #define **GLFW_KEY_PERIOD** 46 /∗ . ∗/
- #define **GLFW_KEY_SLASH** 47 /∗ / ∗/
- #define **GLFW_KEY_0** 48
- #define **GLFW_KEY_1** 49
- #define **GLFW_KEY_2** 50
- #define **GLFW_KEY_3** 51
- #define **GLFW_KEY_4** 52
- #define **GLFW_KEY_5** 53
- #define **GLFW_KEY_6** 54
- #define **GLFW_KEY_7** 55
- #define **GLFW_KEY_8** 56
- #define **GLFW_KEY_9** 57
- #define **GLFW_KEY_SEMICOLON** 59 /∗ ; ∗/
- #define **GLFW_KEY_EQUAL** 61 /∗ = ∗/
- #define **GLFW_KEY_A** 65
- #define **GLFW_KEY_B** 66
- #define **GLFW_KEY_C** 67
- #define **GLFW_KEY_D** 68
- #define **GLFW_KEY_E** 69
- #define **GLFW_KEY_F** 70
- #define **GLFW_KEY_G** 71
- #define **GLFW_KEY_H** 72
- #define **GLFW_KEY_I** 73
- #define **GLFW_KEY_J** 74
- #define **GLFW_KEY_K** 75
- #define **GLFW_KEY_L** 76
- #define **GLFW_KEY_M** 77
- #define **GLFW_KEY_N** 78
- #define **GLFW_KEY_O** 79
- #define **GLFW_KEY_P** 80
- #define **GLFW_KEY_Q** 81
- #define **GLFW_KEY_R** 82
- #define **GLFW_KEY_S** 83

- #define **GLFW_KEY_T** 84
- #define **GLFW_KEY_U** 85
- #define **GLFW_KEY_V** 86
- #define **GLFW_KEY_W** 87
- #define **GLFW_KEY_X** 88
- #define **GLFW_KEY_Y** 89
- #define **GLFW_KEY_Z** 90
- #define **GLFW_KEY_LEFT_BRACKET** 91 /∗ [ ∗/
- #define **GLFW_KEY_BACKSLASH** 92 /∗ \ ∗/
- #define **GLFW_KEY_RIGHT_BRACKET** 93 /∗ ] ∗/
- #define **GLFW_KEY_GRAVE_ACCENT** 96 /∗ ‘ ∗/
- #define **GLFW_KEY_WORLD_1** 161 /∗ non-US #1 ∗/
- #define **GLFW_KEY_WORLD_2** 162 /∗ non-US #2 ∗/
- #define **GLFW_KEY_ESCAPE** 256
- #define **GLFW_KEY_ENTER** 257
- #define **GLFW_KEY_TAB** 258
- #define **GLFW_KEY_BACKSPACE** 259
- #define **GLFW_KEY_INSERT** 260
- #define **GLFW_KEY_DELETE** 261
- #define **GLFW_KEY_RIGHT** 262
- #define **GLFW_KEY_LEFT** 263
- #define **GLFW_KEY_DOWN** 264
- #define **GLFW_KEY_UP** 265
- #define **GLFW_KEY_PAGE_UP** 266
- #define **GLFW_KEY_PAGE_DOWN** 267
- #define **GLFW_KEY_HOME** 268
- #define **GLFW_KEY_END** 269
- #define **GLFW_KEY_CAPS_LOCK** 280
- #define **GLFW_KEY_SCROLL_LOCK** 281
- #define **GLFW_KEY_NUM_LOCK** 282
- #define **GLFW_KEY_PRINT_SCREEN** 283
- #define **GLFW_KEY_PAUSE** 284
- #define **GLFW_KEY_F1** 290
- #define **GLFW_KEY_F2** 291
- #define **GLFW_KEY_F3** 292
- #define **GLFW_KEY_F4** 293
- #define **GLFW_KEY_F5** 294
- #define **GLFW_KEY_F6** 295
- #define **GLFW_KEY_F7** 296
- #define **GLFW_KEY_F8** 297
- #define **GLFW_KEY_F9** 298
- #define **GLFW_KEY_F10** 299
- #define **GLFW_KEY_F11** 300
- #define **GLFW_KEY_F12** 301
- #define **GLFW_KEY_F13** 302
- #define **GLFW_KEY_F14** 303
- #define **GLFW_KEY_F15** 304
- #define **GLFW_KEY_F16** 305
- #define **GLFW_KEY_F17** 306
- #define **GLFW_KEY_F18** 307
- #define **GLFW_KEY_F19** 308
- #define **GLFW_KEY_F20** 309
- #define **GLFW_KEY_F21** 310
- #define **GLFW_KEY_F22** 311
- #define **GLFW_KEY_F23** 312

- #define **GLFW_KEY_F24** 313
- #define **GLFW_KEY_F25** 314
- #define **GLFW_KEY_KP_0** 320
- #define **GLFW_KEY_KP_1** 321
- #define **GLFW_KEY_KP_2** 322
- #define **GLFW_KEY_KP_3** 323
- #define **GLFW_KEY_KP_4** 324
- #define **GLFW_KEY_KP_5** 325
- #define **GLFW_KEY_KP_6** 326
- #define **GLFW_KEY_KP_7** 327
- #define **GLFW_KEY_KP_8** 328
- #define **GLFW_KEY_KP_9** 329
- #define **GLFW_KEY_KP_DECIMAL** 330
- #define **GLFW_KEY_KP_DIVIDE** 331
- #define **GLFW_KEY_KP_MULTIPLY** 332
- #define **GLFW_KEY_KP_SUBTRACT** 333
- #define **GLFW_KEY_KP_ADD** 334
- #define **GLFW_KEY_KP_ENTER** 335
- #define **GLFW_KEY_KP_EQUAL** 336
- #define **GLFW_KEY_LEFT_SHIFT** 340
- #define **GLFW_KEY_LEFT_CONTROL** 341
- #define **GLFW_KEY_LEFT_ALT** 342
- #define **GLFW_KEY_LEFT_SUPER** 343
- #define **GLFW_KEY_RIGHT_SHIFT** 344
- #define **GLFW_KEY_RIGHT_CONTROL** 345
- #define **GLFW_KEY_RIGHT_ALT** 346
- #define **GLFW_KEY_RIGHT_SUPER** 347
- #define **GLFW_KEY_MENU** 348
- #define **GLFW_KEY_LAST** GLFW_KEY_MENU

### 5.6.1 Detailed Description

See key input for how these are used.

These key codes are inspired by the *USB HID Usage Tables v1.12* (p. 53-60), but re-arranged to map to 7-bit ASCII for printable keys (function keys are put in the 256+ range).

The naming of the key codes follow these rules:

- The US keyboard layout is used
- Names of printable alpha-numeric characters are used (e.g. "A", "R", "3", etc.)
- For non-alphanumeric characters, Unicode:ish names are used (e.g. "COMMA", "LEFT_SQUARE_BRAC↩ KET", etc.). Note that some names do not correspond to the Unicode standard (usually for brevity)
- Keys that lack a clear US mapping are named "WORLD_x"
- For non-printable keys, custom names are used (e.g. "F4", "BACKSPACE", etc.)

## 5.7 Modifier key flags

### Macros

- #define GLFW_MOD_SHIFT 0x0001

  *If this bit is set one or more Shift keys were held down.*
- #define GLFW_MOD_CONTROL 0x0002

  *If this bit is set one or more Control keys were held down.*
- #define GLFW_MOD_ALT 0x0004

  *If this bit is set one or more Alt keys were held down.*
- #define GLFW_MOD_SUPER 0x0008

  *If this bit is set one or more Super keys were held down.*

- #define GLFW_MOD_SHIFT 0x0001

  *If this bit is set one or more Shift keys were held down.*
- #define GLFW_MOD_CONTROL 0x0002

  *If this bit is set one or more Control keys were held down.*
- #define GLFW_MOD_ALT 0x0004

  *If this bit is set one or more Alt keys were held down.*
- #define GLFW_MOD_SUPER 0x0008

  *If this bit is set one or more Super keys were held down.*

### 5.7.1 Detailed Description

See key input for how these are used.

## 5.8 Mouse buttons

### Macros

- #define **GLFW_MOUSE_BUTTON_1** 0
- #define **GLFW_MOUSE_BUTTON_2** 1
- #define **GLFW_MOUSE_BUTTON_3** 2
- #define **GLFW_MOUSE_BUTTON_4** 3
- #define **GLFW_MOUSE_BUTTON_5** 4
- #define **GLFW_MOUSE_BUTTON_6** 5
- #define **GLFW_MOUSE_BUTTON_7** 6
- #define **GLFW_MOUSE_BUTTON_8** 7
- #define **GLFW_MOUSE_BUTTON_LAST** GLFW_MOUSE_BUTTON_8
- #define **GLFW_MOUSE_BUTTON_LEFT** GLFW_MOUSE_BUTTON_1
- #define **GLFW_MOUSE_BUTTON_RIGHT** GLFW_MOUSE_BUTTON_2
- #define **GLFW_MOUSE_BUTTON_MIDDLE** GLFW_MOUSE_BUTTON_3

- #define **GLFW_MOUSE_BUTTON_1** 0
- #define **GLFW_MOUSE_BUTTON_2** 1
- #define **GLFW_MOUSE_BUTTON_3** 2
- #define **GLFW_MOUSE_BUTTON_4** 3
- #define **GLFW_MOUSE_BUTTON_5** 4
- #define **GLFW_MOUSE_BUTTON_6** 5
- #define **GLFW_MOUSE_BUTTON_7** 6
- #define **GLFW_MOUSE_BUTTON_8** 7
- #define **GLFW_MOUSE_BUTTON_LAST** GLFW_MOUSE_BUTTON_8
- #define **GLFW_MOUSE_BUTTON_LEFT** GLFW_MOUSE_BUTTON_1
- #define **GLFW_MOUSE_BUTTON_RIGHT** GLFW_MOUSE_BUTTON_2
- #define **GLFW_MOUSE_BUTTON_MIDDLE** GLFW_MOUSE_BUTTON_3

### 5.8.1 Detailed Description

See mouse button input for how these are used.

## 5.9 Joysticks

### Macros

- #define **GLFW_JOYSTICK_1** 0
- #define **GLFW_JOYSTICK_2** 1
- #define **GLFW_JOYSTICK_3** 2
- #define **GLFW_JOYSTICK_4** 3
- #define **GLFW_JOYSTICK_5** 4
- #define **GLFW_JOYSTICK_6** 5
- #define **GLFW_JOYSTICK_7** 6
- #define **GLFW_JOYSTICK_8** 7
- #define **GLFW_JOYSTICK_9** 8
- #define **GLFW_JOYSTICK_10** 9
- #define **GLFW_JOYSTICK_11** 10
- #define **GLFW_JOYSTICK_12** 11
- #define **GLFW_JOYSTICK_13** 12
- #define **GLFW_JOYSTICK_14** 13
- #define **GLFW_JOYSTICK_15** 14
- #define **GLFW_JOYSTICK_16** 15
- #define **GLFW_JOYSTICK_LAST** GLFW_JOYSTICK_16

- #define **GLFW_JOYSTICK_1** 0
- #define **GLFW_JOYSTICK_2** 1
- #define **GLFW_JOYSTICK_3** 2
- #define **GLFW_JOYSTICK_4** 3
- #define **GLFW_JOYSTICK_5** 4
- #define **GLFW_JOYSTICK_6** 5
- #define **GLFW_JOYSTICK_7** 6
- #define **GLFW_JOYSTICK_8** 7
- #define **GLFW_JOYSTICK_9** 8
- #define **GLFW_JOYSTICK_10** 9
- #define **GLFW_JOYSTICK_11** 10
- #define **GLFW_JOYSTICK_12** 11
- #define **GLFW_JOYSTICK_13** 12
- #define **GLFW_JOYSTICK_14** 13
- #define **GLFW_JOYSTICK_15** 14
- #define **GLFW_JOYSTICK_16** 15
- #define **GLFW_JOYSTICK_LAST** GLFW_JOYSTICK_16

### 5.9.1 Detailed Description

See joystick input for how these are used.

## 5.10 Error codes

### Macros

- #define GLFW_NOT_INITIALIZED 0x00010001

  *GLFW has not been initialized.*
- #define GLFW_NO_CURRENT_CONTEXT 0x00010002

  *No context is current for this thread.*
- #define GLFW_INVALID_ENUM 0x00010003

  *One of the arguments to the function was an invalid enum value.*
- #define GLFW_INVALID_VALUE 0x00010004

  *One of the arguments to the function was an invalid value.*
- #define GLFW_OUT_OF_MEMORY 0x00010005

  *A memory allocation failed.*
- #define GLFW_API_UNAVAILABLE 0x00010006

  *GLFW could not find support for the requested client API on the system.*
- #define GLFW_VERSION_UNAVAILABLE 0x00010007

  *The requested OpenGL or OpenGL ES version is not available.*
- #define GLFW_PLATFORM_ERROR 0x00010008

  *A platform-specific error occurred that does not match any of the more specific categories.*
- #define GLFW_FORMAT_UNAVAILABLE 0x00010009

  *The requested format is not supported or available.*

- #define GLFW_NOT_INITIALIZED 0x00010001

  *GLFW has not been initialized.*
- #define GLFW_NO_CURRENT_CONTEXT 0x00010002

  *No context is current for this thread.*
- #define GLFW_INVALID_ENUM 0x00010003

  *One of the arguments to the function was an invalid enum value.*
- #define GLFW_INVALID_VALUE 0x00010004

  *One of the arguments to the function was an invalid value.*
- #define GLFW_OUT_OF_MEMORY 0x00010005

  *A memory allocation failed.*
- #define GLFW_API_UNAVAILABLE 0x00010006

  *GLFW could not find support for the requested client API on the system.*
- #define GLFW_VERSION_UNAVAILABLE 0x00010007

  *The requested OpenGL or OpenGL ES version is not available.*
- #define GLFW_PLATFORM_ERROR 0x00010008

  *A platform-specific error occurred that does not match any of the more specific categories.*
- #define GLFW_FORMAT_UNAVAILABLE 0x00010009

  *The requested format is not supported or available.*

### 5.10.1 Detailed Description

See error handling for how these are used.

## 5.10.2 Macro Definition Documentation

#### 5.10.2.1 #define GLFW_API_UNAVAILABLE 0x00010006

GLFW could not find support for the requested client API on the system.

GLFW could not find support for the requested client API on the system. If emitted by functions other than glfw↩
CreateWindow, no supported client API was found.

**Analysis**

The installed graphics driver does not support the requested client API, or does not support it via the chosen context creation backend. Below are a few examples.

Some pre-installed Windows graphics drivers do not support OpenGL. AMD only supports OpenGL ES via EGL, while Nvidia and Intel only support it via a WGL or GLX extension. OS X does not provide OpenGL ES at all. The Mesa EGL, OpenGL and OpenGL ES libraries do not interface with the Nvidia binary driver.

#### 5.10.2.2 #define GLFW_API_UNAVAILABLE 0x00010006

GLFW could not find support for the requested client API on the system.

GLFW could not find support for the requested client API on the system. If emitted by functions other than glfw↩
CreateWindow, no supported client API was found.

**Analysis**

The installed graphics driver does not support the requested client API, or does not support it via the chosen context creation backend. Below are a few examples.

Some pre-installed Windows graphics drivers do not support OpenGL. AMD only supports OpenGL ES via EGL, while Nvidia and Intel only support it via a WGL or GLX extension. OS X does not provide OpenGL ES at all. The Mesa EGL, OpenGL and OpenGL ES libraries do not interface with the Nvidia binary driver.

#### 5.10.2.3 #define GLFW_FORMAT_UNAVAILABLE 0x00010009

The requested format is not supported or available.

If emitted during window creation, the requested pixel format is not supported.

If emitted when querying the clipboard, the contents of the clipboard could not be converted to the requested format.

**Analysis**

If emitted during window creation, one or more hard constraints did not match any of the available pixel formats. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

If emitted when querying the clipboard, ignore the error or report it to the user, as appropriate.

#### 5.10.2.4 #define GLFW_FORMAT_UNAVAILABLE 0x00010009

The requested format is not supported or available.

If emitted during window creation, the requested pixel format is not supported.

If emitted when querying the clipboard, the contents of the clipboard could not be converted to the requested format.

**Analysis**

If emitted during window creation, one or more hard constraints did not match any of the available pixel formats. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

If emitted when querying the clipboard, ignore the error or report it to the user, as appropriate.

#### 5.10.2.5 #define GLFW_INVALID_ENUM 0x00010003

One of the arguments to the function was an invalid enum value.

One of the arguments to the function was an invalid enum value, for example requesting GLFW_RED_BITS with glfwGetWindowAttrib.

**Analysis**

Application programmer error. Fix the offending call.

#### 5.10.2.6 #define GLFW_INVALID_ENUM 0x00010003

One of the arguments to the function was an invalid enum value.

One of the arguments to the function was an invalid enum value, for example requesting GLFW_RED_BITS with glfwGetWindowAttrib.

**Analysis**

Application programmer error. Fix the offending call.

#### 5.10.2.7 #define GLFW_INVALID_VALUE 0x00010004

One of the arguments to the function was an invalid value.

One of the arguments to the function was an invalid value, for example requesting a non-existent OpenGL or OpenGL ES version like 2.7.

Requesting a valid but unavailable OpenGL or OpenGL ES version will instead result in a GLFW_VERSION_UN↩
AVAILABLE error.

**Analysis**

Application programmer error. Fix the offending call.

#### 5.10.2.8 #define GLFW_INVALID_VALUE 0x00010004

One of the arguments to the function was an invalid value.

One of the arguments to the function was an invalid value, for example requesting a non-existent OpenGL or OpenGL ES version like 2.7.

Requesting a valid but unavailable OpenGL or OpenGL ES version will instead result in a GLFW_VERSION_UN←AVAILABLE error.

**Analysis**

Application programmer error. Fix the offending call.

#### 5.10.2.9 #define GLFW_NO_CURRENT_CONTEXT 0x00010002

No context is current for this thread.

This occurs if a GLFW function was called that needs and operates on the current OpenGL or OpenGL ES context but no context is current on the calling thread. One such function is glfwSwapInterval.

**Analysis**

Application programmer error. Ensure a context is current before calling functions that require a current context.

#### 5.10.2.10 #define GLFW_NO_CURRENT_CONTEXT 0x00010002

No context is current for this thread.

This occurs if a GLFW function was called that needs and operates on the current OpenGL or OpenGL ES context but no context is current on the calling thread. One such function is glfwSwapInterval.

**Analysis**

Application programmer error. Ensure a context is current before calling functions that require a current context.

#### 5.10.2.11 #define GLFW_NOT_INITIALIZED 0x00010001

GLFW has not been initialized.

This occurs if a GLFW function was called that may not be called unless the library is initialized.

**Analysis**

Application programmer error. Initialize GLFW before calling any function that requires initialization.

#### 5.10.2.12 #define GLFW_NOT_INITIALIZED 0x00010001

GLFW has not been initialized.

This occurs if a GLFW function was called that may not be called unless the library is initialized.

**Analysis**

Application programmer error. Initialize GLFW before calling any function that requires initialization.

#### 5.10.2.13 #define GLFW_OUT_OF_MEMORY 0x00010005

A memory allocation failed.

A memory allocation failed.

**Analysis**

A bug in GLFW or the underlying operating system. Report the bug to our `issue tracker`.

#### 5.10.2.14 #define GLFW_OUT_OF_MEMORY 0x00010005

A memory allocation failed.

A memory allocation failed.

**Analysis**

A bug in GLFW or the underlying operating system. Report the bug to our `issue tracker`.

#### 5.10.2.15 #define GLFW_PLATFORM_ERROR 0x00010008

A platform-specific error occurred that does not match any of the more specific categories.

A platform-specific error occurred that does not match any of the more specific categories.

**Analysis**

A bug or configuration error in GLFW, the underlying operating system or its drivers, or a lack of required resources. Report the issue to our `issue tracker`.

#### 5.10.2.16 #define GLFW_PLATFORM_ERROR 0x00010008

A platform-specific error occurred that does not match any of the more specific categories.

A platform-specific error occurred that does not match any of the more specific categories.

**Analysis**

A bug or configuration error in GLFW, the underlying operating system or its drivers, or a lack of required resources. Report the issue to our `issue tracker`.

### 5.10.2.17 #define GLFW_VERSION_UNAVAILABLE 0x00010007

The requested OpenGL or OpenGL ES version is not available.

The requested OpenGL or OpenGL ES version (including any requested context or framebuffer hints) is not available on this machine.

**Analysis**

The machine does not support your requirements. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

Future invalid OpenGL and OpenGL ES versions, for example OpenGL 4.8 if 5.0 comes out before the 4.↩
x series gets that far, also fail with this error and not GLFW_INVALID_VALUE, because GLFW cannot know what future versions will exist.

### 5.10.2.18 #define GLFW_VERSION_UNAVAILABLE 0x00010007

The requested OpenGL or OpenGL ES version is not available.

The requested OpenGL or OpenGL ES version (including any requested context or framebuffer hints) is not available on this machine.

**Analysis**

The machine does not support your requirements. If your application is sufficiently flexible, downgrade your requirements and try again. Otherwise, inform the user that their machine does not match your requirements.

Future invalid OpenGL and OpenGL ES versions, for example OpenGL 4.8 if 5.0 comes out before the 4.↩
x series gets that far, also fail with this error and not GLFW_INVALID_VALUE, because GLFW cannot know what future versions will exist.

## 5.11 Standard cursor shapes

### Macros

- #define GLFW_ARROW_CURSOR 0x00036001

  *The regular arrow cursor shape.*
- #define GLFW_IBEAM_CURSOR 0x00036002

  *The text input I-beam cursor shape.*
- #define GLFW_CROSSHAIR_CURSOR 0x00036003

  *The crosshair shape.*
- #define GLFW_HAND_CURSOR 0x00036004

  *The hand shape.*
- #define GLFW_HRESIZE_CURSOR 0x00036005

  *The horizontal resize arrow shape.*
- #define GLFW_VRESIZE_CURSOR 0x00036006

  *The vertical resize arrow shape.*
- #define GLFW_ARROW_CURSOR 0x00036001

  *The regular arrow cursor shape.*
- #define GLFW_IBEAM_CURSOR 0x00036002

  *The text input I-beam cursor shape.*
- #define GLFW_CROSSHAIR_CURSOR 0x00036003

  *The crosshair shape.*
- #define GLFW_HAND_CURSOR 0x00036004

  *The hand shape.*
- #define GLFW_HRESIZE_CURSOR 0x00036005

  *The horizontal resize arrow shape.*
- #define GLFW_VRESIZE_CURSOR 0x00036006

  *The vertical resize arrow shape.*

### 5.11.1 Detailed Description

See standard cursor creation for how these are used.

### 5.11.2 Macro Definition Documentation

#### 5.11.2.1 #define GLFW_ARROW_CURSOR 0x00036001

The regular arrow cursor shape.

The regular arrow cursor.

#### 5.11.2.2 #define GLFW_ARROW_CURSOR 0x00036001

The regular arrow cursor shape.

The regular arrow cursor.

#### 5.11.2.3 #define GLFW_CROSSHAIR_CURSOR 0x00036003

The crosshair shape.

The crosshair shape.

#### 5.11.2.4 #define GLFW_CROSSHAIR_CURSOR 0x00036003

The crosshair shape.

The crosshair shape.

#### 5.11.2.5 #define GLFW_HAND_CURSOR 0x00036004

The hand shape.

The hand shape.

#### 5.11.2.6 #define GLFW_HAND_CURSOR 0x00036004

The hand shape.

The hand shape.

#### 5.11.2.7 #define GLFW_HRESIZE_CURSOR 0x00036005

The horizontal resize arrow shape.

The horizontal resize arrow shape.

#### 5.11.2.8 #define GLFW_HRESIZE_CURSOR 0x00036005

The horizontal resize arrow shape.

The horizontal resize arrow shape.

#### 5.11.2.9 #define GLFW_IBEAM_CURSOR 0x00036002

The text input I-beam cursor shape.

The text input I-beam cursor shape.

#### 5.11.2.10 #define GLFW_IBEAM_CURSOR 0x00036002

The text input I-beam cursor shape.

The text input I-beam cursor shape.

#### 5.11.2.11 #define GLFW_VRESIZE_CURSOR 0x00036006

The vertical resize arrow shape.

The vertical resize arrow shape.

#### 5.11.2.12 #define GLFW_VRESIZE_CURSOR 0x00036006

The vertical resize arrow shape.

The vertical resize arrow shape.

## 5.12 Native access

**By using the native access functions you assert that you know what you're doing and how to fix problems caused by using them. If you don't, you shouldn't be using them.**

Before the inclusion of glfw3native.h, you must define exactly one window system API macro and exactly one context creation API macro. Failure to do this will cause a compile-time error.

The available window API macros are:

- `GLFW_EXPOSE_NATIVE_WIN32`
- `GLFW_EXPOSE_NATIVE_COCOA`
- `GLFW_EXPOSE_NATIVE_X11`

The available context API macros are:

- `GLFW_EXPOSE_NATIVE_WGL`
- `GLFW_EXPOSE_NATIVE_NSGL`
- `GLFW_EXPOSE_NATIVE_GLX`
- `GLFW_EXPOSE_NATIVE_EGL`

These macros select which of the native access functions that are declared and which platform-specific headers to include. It is then up your (by definition platform-specific) code to handle which of these should be defined.

# Chapter 6

# Class Documentation

## 6.1 _GPU_DEVICE Struct Reference

### Public Attributes

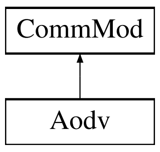
- DWORD **cb**
- CHAR **DeviceName** [32]
- CHAR **DeviceString** [128]
- DWORD **Flags**
- RECT **rcVirtualScreen**

The documentation for this struct was generated from the following file:

- code/liboctodrone/include/GL/wglew.h

## 6.2 Aodv Class Reference

Inheritance diagram for Aodv:



### Public Member Functions

- Aodv (Environment ∗, std::string, std::atomic_flag ∗, bool)

  *Implementation of the AODV routing protocol.*

### Protected Member Functions

- void comm_function ()

  *The main communications loop which handles incomming and outgoing messages.*

## Private Member Functions

- Aodv_rreq ∗ create_hello ()
- Aodv_rreq ∗ create_rreq (std::string, std::string, int)

    *Helper method for creating route requests.*
- Aodv_rrep ∗ create_rrep (std::string, std::string, int)
- Aodv_rerr ∗ create_rerr (std::string, int)

    *Helper method for creating route errors.*
- void process_rreq (Aodv_rreq ∗)

    *Handle an RREQ that was received.*
- void process_rrep (Aodv_rrep ∗)

    *Handle an RREP that was received.*
- void process_rerr (Aodv_rerr ∗)

    *Handle an RERR that was received.*
- void process_data (std::string)

    *Handle an data packet that was received.*
- std::string get_attribute (std::string)

    *Attribute extraction for AODV messages.*
- bool have_route (std::string)

    *Checks if a route to the destination is known.*
- void add_route (std::string, int, int, std::string)

    *Creates a route in the routing table.*
- Aodv_rreq ∗ deserialize_rreq (std::string)

    *Deserialization for route requests.*
- Aodv_rrep ∗ deserialize_rrep (std::string)

    *Deserialization for route replies.*
- Aodv_rerr ∗ deserialize_rerr (std::string)

    *Deserialization for route errors.*
- void log (std::string)

    *Helper function to log internal information.*
- void broadcast (std::string)

    *Helper function to broadcast a message through the environment.*

## Private Attributes

- std::map< std::string, Aodv_route ∗ > route_table

    *Routing table for the communication module.*
- std::string ip_address

    *The IP address of the communication module.*
- double HELLO_INTERVAL

    *The interval at which hello messages are sent to discover nearby nodes.*
- int SEQUENCE_NUMBER

    *The AODV sequence number of the communication module.*
- double ACTIVE_ROUTE_TIMEOUT

    *The AODV active route timeout used to determine how long routes stay fresh for.*
- double PATH_DISCOVERY_TIME

    *How long to wait for relies to route requests.*
- int BROADCAST_ID

    *AODV broadcast ID used to prevent loops and ensure fresh information.*
- int RANGE

*The amount of power used to broadcast messages.*
- int TTL

  *Default time to live for messages, dependent on network size.*
- double last_hello

  *The time at which the last hello message was sent.*
- std::pair< std::string, std::string > **current_message**
- int state

  *The internal state of the AODV implementation.*
- std::atomic_flag ∗ lock

  *An atomic lock to regulate access to stdout.*
- bool logging

  *Switch to enable or disable logging.*

## Additional Inherited Members

### 6.2.1 Member Function Documentation

#### 6.2.1.1 void Aodv::add_route ( std::string *ip,* int *dest_seq,* int *hop_count,* std::string *next_hop* ) [private]

Creates a route in the routing table.

Adds in the standard route timeout and ensures that all variables are set to prevent memory issues later

#### 6.2.1.2 void Aodv::broadcast ( std::string *message* ) [private]

Helper function to broadcast a message through the environment.

Broadcast requires the coordinates of the broadcasting unit which can be unwieldy to do every time This process is simplified by wrapping it in a helper function

#### 6.2.1.3 void Aodv::comm_function ( ) [protected], [virtual]

The main communications loop which handles incomming and outgoing messages.

Every loop we check to see if we should send a hello, based on the hello interval Next, any incomming messages are deserialized and handled appropriately If our messageable told us to shut down, the communications module exits Next, any outgoing messages are either immediately sent (if we have a route to the destination already) or a route request is distributed (if we do not have a route)

Implements CommMod.

#### 6.2.1.4 Aodv_rreq ∗ Aodv::create_hello ( ) [private]

Creates a special route request for a route to this node, with TTL 1

#### 6.2.1.5 Aodv_rerr ∗ Aodv::create_rerr ( std::string *dst_ip,* int *ttl* ) [private]

Helper method for creating route errors.

Abstracts away the destination sequence number lookup for convienence

#### 6.2.1.6 Aodv_rrep ∗ Aodv::create_rrep ( std::string *dst_ip,* std::string *src_ip,* int *ttl* ) [private]

Abstracts away next hop lookup for convienence

#### 6.2.1.7 Aodv_rreq ∗ Aodv::create_rreq ( std::string *dst_ip,* std::string *src_ip,* int *ttl* ) [private]

Helper method for creating route requests.

Abstracts away the destination sequence number lookup for convienence

#### 6.2.1.8 Aodv_rerr ∗ Aodv::deserialize_rerr ( std::string *message* ) [private]

Deserialization for route errors.

Takes a raw message string of a route error message and returns an AODV object representing that message

#### 6.2.1.9 Aodv_rrep ∗ Aodv::deserialize_rrep ( std::string *message* ) [private]

Deserialization for route replies.

Takes a raw message string of a route reply message and returns an AODV object representing that message

#### 6.2.1.10 Aodv_rreq ∗ Aodv::deserialize_rreq ( std::string *message* ) [private]

Deserialization for route requests.

Takes a raw message string of a route reuest message and returns an AODV object representing that message

#### 6.2.1.11 std::string Aodv::get_attribute ( std::string *message* ) [private]

Attribute extraction for AODV messages.

Takes in a message string and returns the first field of that message (semicolon delimited)

#### 6.2.1.12 bool Aodv::have_route ( std::string *ip* ) [private]

Checks if a route to the destination is known.

Determines if a route to the destination is known, and if that route is fresh

#### 6.2.1.13 void Aodv::log ( std::string *log_message* ) [private]

Helper function to log internal information.

Makes use of the stdout lock we were passed on creation to make sure that only one thread is printing at any one time Also prints the ip of the communications module and the current environment time to help with debugging

#### 6.2.1.14 void Aodv::process_data ( std::string *message* ) [private]

Handle an data packet that was received.

First we check if the message was for us, and if so we deliver the contents to our messageable If the message is destined for another node and we are specified as the next hop, then we forward that packet to the next hop according to our own routing table Note that messages destined for other nodes that we are not specified as a next hop for will be dropped

#### 6.2.1.15 void Aodv::process_rerr ( Aodv_rerr * *message* ) [private]

Handle an RERR that was received.

This functionality os not yet implemented

#### 6.2.1.16 void Aodv::process_rrep ( Aodv_rrep * *message* ) [private]

Handle an RREP that was received.

First we check if the message contains new information about other nodes, and if so we always update our routing table If the message was a response to a request we initiated, then we should send our data packet. If the message was a response to a request we forwarded for another node, then we should pass it on Note that nodes will only act as a middle hop for one message at a time (other reponses will be dropped)

#### 6.2.1.17 void Aodv::process_rreq ( Aodv_rreq * *message* ) [private]

Handle an RREQ that was received.

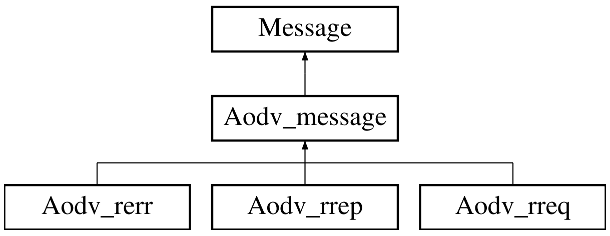
First we check if the message was a hello message, and if so we always respond to it If the message was a normal request, then we either answer it (if we have the info) or forward it if we do not Note that non-hello RREQs will be dropped if we are currently trying to send a message

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv.hpp
- code/liboctodronecomms/src/Aodv.cpp

## 6.3 Aodv_message Class Reference

Inheritance diagram for Aodv_message:



### Public Member Functions

- Aodv_message (std::string, int, int)

  *Base class for all AODV message types.*
- std::string get_dest_ip ()

  *Getter method for the IP address of the destination node.*
- std::string serialize ()

  *Returns a string representation of the message.*
- int get_dest_seq ()

  *Getter method for the sequence number of the destination node.*
- int get_ttl ()

  *Getter method for the time to live.*

### Private Attributes

- std::string dest_ip

  *The IP address of the detination node.*
- int dest_seq

  *The sequence number of the destination node.*
- int ttl

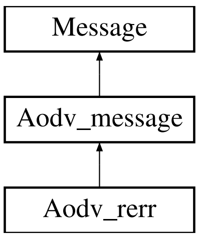
  *The time to live of the message.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv_message.hpp
- code/liboctodronecomms/src/Aodv_message.cpp

## 6.4 Aodv_rerr Class Reference

Inheritance diagram for Aodv_rerr:



### Public Member Functions

- Aodv_rerr (std::string dst_ip, int dst_seq, int ttl)

  *AODV route error message.*
- std::string to_string ()

  *Returns a string representation of the message.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv_rerr.hpp
- code/liboctodronecomms/src/Aodv_rerr.cpp

## 6.5 Aodv_route Class Reference

### Public Member Functions

- Aodv_route (int, int, std::string, int)

  *Entry in ann AODV routing table.*
- int get_seq ()

  *Getter method for the sequence number of the route destination.*
- int get_hop ()

  *Getter method for the route hop count.*
- std::string get_next_hop ()

  *Getter method for the next hop on this route.*
- int get_life ()

  *Getter method for the route life time.*

### Private Attributes

- int dst_seq

  *Sequence number of the route destination.*
- int hop_count

  *Number of hops required to get to the destination node.*
- std::string next_hop

  *Next hop on this route.*
- int life_time

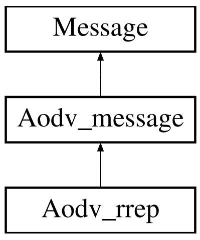
  *Lifetime of this route.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv_route.hpp
- code/liboctodronecomms/src/Aodv_route.cpp

## 6.6 Aodv_rrep Class Reference

Inheritance diagram for Aodv_rrep:



### Public Member Functions

- Aodv_rrep (int, std::string, std::string, std::string, int, int, int, std::string)

  *Aodv route reply message.*
- int get_hop_count ()

  *Getter method for the hop count of the message.*
- std::string get_source_ip ()

  *Getter method for the IP address of the sending node.*
- std::string to_string ()

  *Returns a string representation of the message.*
- int get_life_time ()

  *Getter method for the life time of the route.*
- std::string get_last_hop ()

  *Getter method for te last hop on this route.*
- std::string get_next_hop ()

  *Getter method for the next hop on this route.*

### Private Attributes

- int m_hop_count

  *Number of hops required to get from the source to the destination of the route.*
- std::string m_source_ip

  *IP address of the node at which the route terminates.*
- int m_life_time

  *Time for which this route is valid.*
- std::string m_last_hop

  *The last node which forwarded this reply.*
- std::string m_next_hop

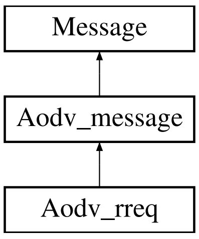
  *The next node to which this reply will be forwarded.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv_rrep.hpp
- code/liboctodronecomms/src/Aodv_rrep.cpp

## 6.7 Aodv_rreq Class Reference

Inheritance diagram for Aodv_rreq:



### Public Member Functions

- Aodv_rreq (int hop, std::string src_ip, std::string dst_ip, int src_seq, int dst_seq, int ttl)

  *AODV route request message.*
- int get_hop_count ()

  *Getter method for the hop count of the message.*
- std::string get_source_ip ()

  *Getter method for the IP address of the sending node.*
- std::string to_string ()

  *Returns a string representation of the message.*
- int get_source_seq ()

  *Getter method for the sequence number of the sending node.*

### Private Attributes

- int hop_count

  *The number of hops taken towards the destination.*
- std::string source_ip

  *The IP address of the sending node.*
- int source_seq

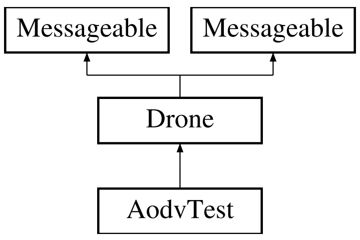
  *The sequence number of the sending node.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Aodv_rreq.hpp
- code/liboctodronecomms/src/Aodv_rreq.cpp

## 6.8 AodvTest Class Reference

Inheritance diagram for AodvTest:



### Public Member Functions

- **AodvTest** (CommMod ∗, double, double, double, double, Environment ∗, int, int ∗, std::atomic_flag ∗)
- bool **message_callback** (Message ∗)
- void **run** ()

### Private Attributes

- int **m_task**
- int ∗ **m_flag**
- std::atomic_flag ∗ **m_lock**
- Environment ∗ **m_env**

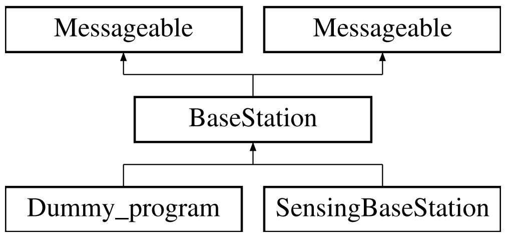
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/AodvTest.hpp
- code/programs/src/AodvTest.cpp

## 6.9 BaseStation Class Reference

Inheritance diagram for BaseStation:



### Public Member Functions

- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp, bool vs)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp)
- **BaseStation** (CommMod ∗cm, double xp, double yp, double zp, bool vs)

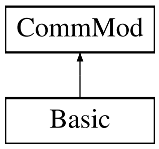
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/liboctodrone/include/BaseStation.hpp
- code/liboctodrone/src/BaseStation.cpp

## 6.10 Basic Class Reference

Inheritance diagram for Basic:



### Public Member Functions

- Basic (Environment ∗, std::atomic_flag ∗)

    *Basic messaging protocol for testing where nodes receive all messages sent if they are within range.*

### Protected Member Functions

- void comm_function ()

    *The main communications loop which handles incomming and outgoing messages.*

### Private Member Functions

- void log (std::string)

    *Helper function to log internal information.*

### Private Attributes

- double RANGE

    *The amount of power used to broadcast messages.*
- std::atomic_flag ∗ lock

    *An atomic lock to regulate access to stdout.*

### Additional Inherited Members

### 6.10.1 Member Function Documentation

#### 6.10.1.1 void Basic::comm_function ( ) `[protected]`, `[virtual]`

The main communications loop which handles incomming and outgoing messages.

Every loop we send any messages that are waiting to be sent Then we deliver any messages that we have received

Implements CommMod.

#### 6.10.1.2 void Basic::log ( std::string *log_message* ) [private]

Helper function to log internal information.

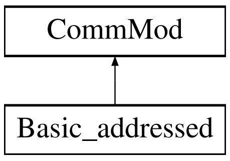
Makes use of the stdout lock we were passed on creation to make sure that only one thread is printing at any one time

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Basic.hpp
- code/liboctodronecomms/src/Basic.cpp

## 6.11 Basic_addressed Class Reference

Inheritance diagram for Basic_addressed:



### Public Member Functions

- Basic_addressed (Environment ∗, std::atomic_flag ∗, std::string ip)

  *Basic messaging protocol with addressing for testing, where nodes receive all messages sent if they are within range and addressed to them.*

### Protected Member Functions

- void comm_function ()

  *The main communications loop which handles incomming and outgoing messages.*

### Private Member Functions

- void **log** (std::string)
- std::string **get_attribute** (std::string)

### Private Attributes

- double RANGE

  *The amount of power used to boradcast messages.*
- std::atomic_flag ∗ lock

  *An atomic lock to regulate access to stdout.*
- std::string ip_address

  *The IP address of the communication module.*

### Additional Inherited Members

### 6.11.1 Member Function Documentation

#### 6.11.1.1 void Basic_addressed::comm_function ( ) [protected], [virtual]

The main communications loop which handles incomming and outgoing messages.

Every loop any incomming messages are deserialized and delivered if they match our IP address Next, any outgoing messages are sent Note that received messages not addressed to this IP address will be dropped

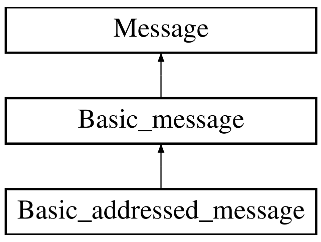
Implements CommMod.

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Basic_addressed.hpp
- code/liboctodronecomms/src/Basic_addressed.cpp

## 6.12 Basic_addressed_message Class Reference

Inheritance diagram for Basic_addressed_message:



### Public Member Functions

- Basic_addressed_message (std::string, std::string, std::string)

  *Basic message with addressing information.*
- std::string to_string ()

  *Returns a string representation of the message.*
- std::string get_message ()

  *Getter method for the message content.*
- std::string get_destination ()

  *Getter method for the IP address of the node this message is destined for.*
- std::string get_source ()

  *Getter method for the IP address of the node which sent this message.*

### Private Attributes

- std::string message

  *The contents of the message.*
- std::string destination

  *The IP address of the messages destination.*
- std::string source

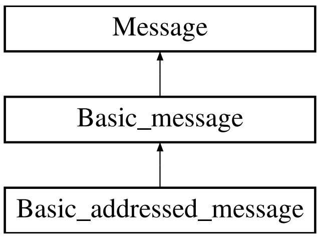
  *The IP address of the node which sent the message.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Basic_addressed_message.hpp
- code/liboctodronecomms/src/Basic_addressed_message.cpp

## 6.13 Basic_message Class Reference

Inheritance diagram for Basic_message:



### Public Member Functions

- Basic_message (std::string)

  *A basic message containing a payload wth no addressing information.*
- std::string to_string ()

  *Returns a string representation of the message.*

### Private Attributes

- std::string message

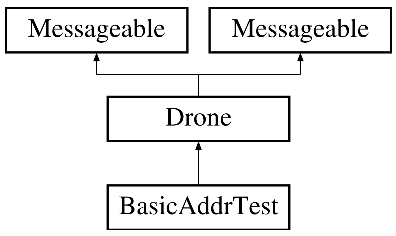
  *The contents of the message.*

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Basic_message.hpp
- code/liboctodronecomms/src/Basic_message.cpp

## 6.14 BasicAddrTest Class Reference

Inheritance diagram for BasicAddrTest:



### Public Member Functions

- **BasicAddrTest** (CommMod ∗, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

### Private Attributes

- bool **sink_node**

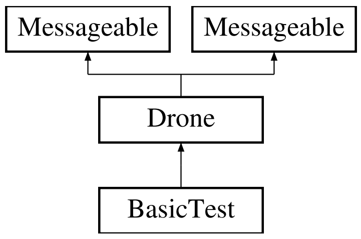
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/BasicAddrTest.hpp
- code/programs/src/BasicAddrTest.cpp

## 6.15 BasicTest Class Reference

Inheritance diagram for BasicTest:



### Public Member Functions

- **BasicTest** (CommMod ∗, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

### Private Attributes

- bool **sink_node**

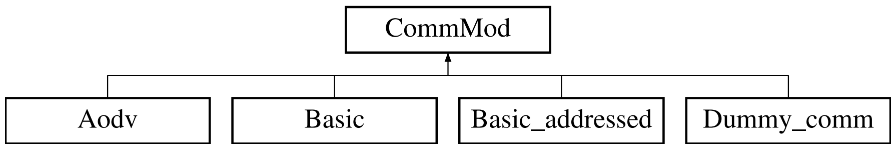
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/BasicTest.hpp
- code/programs/src/BasicTest.cpp

## 6.16 CommMod Class Reference

Inheritance diagram for CommMod:



### Public Member Functions

- CommMod (Environment ∗env)
  *Base class for communication algorithm implementations.*
- void set_messageable (Messageable ∗msg)
  *Set the messageble associated with this communications module.*
- void broadcast (std::string message, double xPos, double yPos, double zPos, double range)
  *Passes a message to the environment for transmission.*
- void broadcast (Message ∗message, double xPos, double yPos, double zPos, double range)
  *Passes a message to the environment for transmission.*
- void push_out_message (Message ∗message)
  *Called by a messagable to send a message.*
- void push_in_message (std::string message)
  *Called by a messagable to receive a message.*
- void **pass_message** (Message ∗message)
- void **comm_function_wrapper** ()
- virtual void comm_function ()=0
  *Main loop which must be defined by communications implementations.*
- bool **getAlive** ()
- double getTime ()
  *Get the time from the environment.*
- **CommMod** (Environment ∗env)
- void **set_messageable** (Messageable ∗msg)
- void **broadcast** (std::string message, double xPos, double yPos, double zPos, double range)
- void **broadcast** (Message ∗message, double xPos, double yPos, double zPos, double range)
- void **push_out_message** (Message ∗message)
- void **push_in_message** (std::string message)
- void **pass_message** (Message ∗message)
- void **comm_function_wrapper** ()
- virtual void comm_function ()=0
  *Main loop which must be defined by communications implementations.*
- bool **getAlive** ()
- double **getTime** ()

### Protected Attributes

- std::queue< Message ∗ > outQueue

  *the queue of messages to be sent*
- std::queue< std::string > inQueue

  *the queue of received messages waiting for collection by the messageable*
- Environment ∗ environment

  *Reference to the simulation environment.*
- Messageable ∗ messageable

  *Reference to the associated messageable.*

### Private Attributes

- bool **alive**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/CommMod.hpp
- code/liboctodrone/src/CommMod.cpp

## 6.17 Coord Struct Reference

### Public Attributes

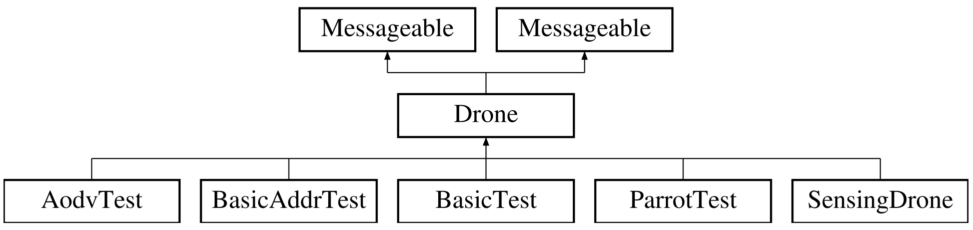
- double **x**
- double **y**
- double **z**

The documentation for this struct was generated from the following file:

- code/liboctodrone/include/Messageable.hpp

## 6.18 Drone Class Reference

Inheritance diagram for Drone:



## Public Member Functions

- Drone (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e)

  *Modified to start flight when the program runs.*
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e, bool vis)
- bool **isAlive** ()
- void upkeep (bool)

  *Modified to make the drone hover when it has finished a movement instruction.*
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e)
- **Drone** (CommMod ∗cm, double iX, double iY, double iZ, double maxSpeed, Environment ∗e, bool vis)
- bool **isAlive** ()
- void **upkeep** (bool)
- void execute (std::string, double)

  *Takes a command and sends it to the nodeServer which is connected to the drone.*
- void **execute** (std::string)

## Protected Member Functions

- void kill ()

  *Modified to end flight when the program terminates.*
- void turn (double dAngle)

  *Modified to create estimates based on real time and to send instructions to nodeServer.*
- void move (Direction direction, double speed, double distance)

  *Modified to calculate duration based on real time and to send instructions to nodeServer.*
- double **getMaxSpeed** ()
- double **getSpeed** ()
- double **getAngle** ()
- bool hasFinishedMoving ()

  *Modified to evaluate estimated move finishing times.*
- double **sense** (std::string type)
- void **kill** ()
- void **turn** (double dAngle)
- void **move** (Direction direction, double speed, double distance)
- double **getMaxSpeed** ()
- double **getAngle** ()
- bool **hasFinishedMoving** ()
- double **sense** (std::string type)

## Private Attributes

- double **oTime**
- bool **alive** = true
- Direction **dir**
- double **maxSpeed**
- double **ang** = 0
- Environment ∗ **env**
- double **moveDR**
- double **moveSpd**
- bool **visualise**

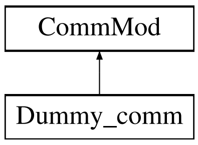
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/liboctodrone/include/Drone.hpp
- code/liboctodrone/src/Drone.cpp

## 6.19 Dummy_comm Class Reference

Inheritance diagram for Dummy_comm:



### Public Member Functions

- **Dummy_comm** (Environment ∗)

### Protected Member Functions

- void comm_function ()
  *Main loop which must be defined by communications implementations.*

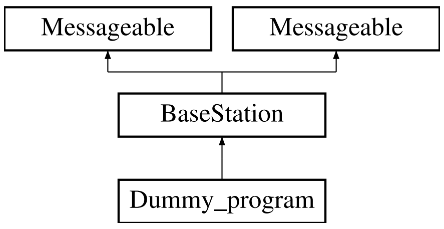
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/liboctodronecomms/include/Dummy_comm.hpp
- code/liboctodronecomms/src/Dummy_comm.cpp

## 6.20 Dummy_program Class Reference

Inheritance diagram for Dummy_program:



### Public Member Functions

- **Dummy_program** (CommMod $*$, double, double, double)
- bool **message_callback** (Message $*$)
- void **run** ()

### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/Dummy_program.hpp
- code/programs/src/Dummy_program.cpp

## 6.21 Element Class Reference

### Public Member Functions

- **Element** (IMG img, int x, int y, int size, int steps)
- void **draw** ()
- bool **step** ()

### Private Attributes

- int **stepsLeft**
- int **x**
- int **y**
- int **size**
- IMG **image**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/Visualisation.hpp
- code/liboctodrone/src/Visualisation.cpp

## 6.22 Environment Class Reference

### Public Types

- typedef std::vector$<$ std::vector$<$ std::vector$<$ double $>$ $>$ $>$ **data_type**

## Public Member Functions

- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, double timestep)
- **Environment** (std::map< std::string, data_type >, double timestep)
- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, double timestep, bool visualise)
- **Environment** (std::map< std::string, data_type >, double timestep, bool visualise)
- void **broadcast** (std::string message, double xOrigin, double yOrigin, double zOrigin, double range, Comm↩ Mod ∗)
- void **addData** (std::string type, data_type d)
- void **addDrone** (Drone ∗m)
- void **setBaseStation** (BaseStation ∗m)
- double **getData** (std::string type, double x, double y, double z)
- double getTime ()

    *Modified to use real time.*
- void **run** ()
- **Environment** (std::map< std::string, data_type >, std::function< std::string(std::string)>, std::string)
- **Environment** (std::map< std::string, data_type >, std::string)
- void **broadcast** (std::string message, double xOrigin, double yOrigin, double zOrigin, double range, Comm↩ Mod ∗)
- void **addData** (std::string type, data_type d)
- void **addDrone** (Drone ∗m)
- void **setBaseStation** (BaseStation ∗m)
- double **getData** (std::string type, double x, double y, double z)
- Drone ∗ getDrone ()

    *Returns the only drone in the sharded environment.*
- double **getTime** ()
- void **run** ()

## Private Types

- typedef std::vector< std::vector< std::vector< double > > > **data_type**

## Private Attributes

- double **timeElapsed**
- double **timeStep**
- bool **visualise**
- BaseStation ∗ **baseStation**
- std::vector< Drone ∗ > **drones**
- std::map< std::string, data_type > **data**
- std::function< std::string(std::string)> **noiseFun**
- std::string **if_addr**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/Environment.hpp
- code/liboctodrone/src/Environment.cpp

## 6.23 GLFWgammaramp Struct Reference

Gamma ramp.

`#include <glfw3.h>`

### Public Attributes

- unsigned short ∗ red
- unsigned short ∗ green
- unsigned short ∗ blue
- unsigned int size

### 6.23.1 Detailed Description

Gamma ramp.

This describes the gamma ramp for a monitor.

**See also**

glfwGetGammaRamp glfwSetGammaRamp

### 6.23.2 Member Data Documentation

#### 6.23.2.1 unsigned short ∗ GLFWgammaramp::blue

An array of value describing the response of the blue channel.

#### 6.23.2.2 unsigned short ∗ GLFWgammaramp::green

An array of value describing the response of the green channel.

#### 6.23.2.3 unsigned short ∗ GLFWgammaramp::red

An array of value describing the response of the red channel.

#### 6.23.2.4 unsigned int GLFWgammaramp::size

The number of elements in each array.

The documentation for this struct was generated from the following files:

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

## 6.24 GLFWimage Struct Reference

Image data.

```
#include <glfw3.h>
```

### Public Attributes

- int width
- int height
- unsigned char ∗ pixels

### 6.24.1 Detailed Description

Image data.

### 6.24.2 Member Data Documentation

#### 6.24.2.1 int GLFWimage::height

The height, in pixels, of this image.

#### 6.24.2.2 unsigned char ∗ GLFWimage::pixels

The pixel data of this image, arranged left-to-right, top-to-bottom.

#### 6.24.2.3 int GLFWimage::width

The width, in pixels, of this image.

The documentation for this struct was generated from the following files:

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

## 6.25 GLFWvidmode Struct Reference

Video mode type.

```
#include <glfw3.h>
```

## Public Attributes

- int width
- int height
- int redBits
- int greenBits
- int blueBits
- int refreshRate

### 6.25.1 Detailed Description

Video mode type.

This describes a single video mode.

### 6.25.2 Member Data Documentation

#### 6.25.2.1 int GLFWvidmode::blueBits

The bit depth of the blue channel of the video mode.

#### 6.25.2.2 int GLFWvidmode::greenBits

The bit depth of the green channel of the video mode.

#### 6.25.2.3 int GLFWvidmode::height

The height, in screen coordinates, of the video mode.

#### 6.25.2.4 int GLFWvidmode::redBits

The bit depth of the red channel of the video mode.

#### 6.25.2.5 int GLFWvidmode::refreshRate

The refresh rate, in Hz, of the video mode.

#### 6.25.2.6 int GLFWvidmode::width

The width, in screen coordinates, of the video mode.

The documentation for this struct was generated from the following files:

- code/liboctodrone/include/GLFW/glfw3.h
- code/liboctodrone/include/GLFW/glfw3_32.h

## 6.26 IpAllocator Class Reference

### Public Member Functions

- **IpAllocator** (int, int, int, int)
- std::string **next** ()
- **IpAllocator** (int, int, int, int)
- std::string **next** ()

### Private Attributes

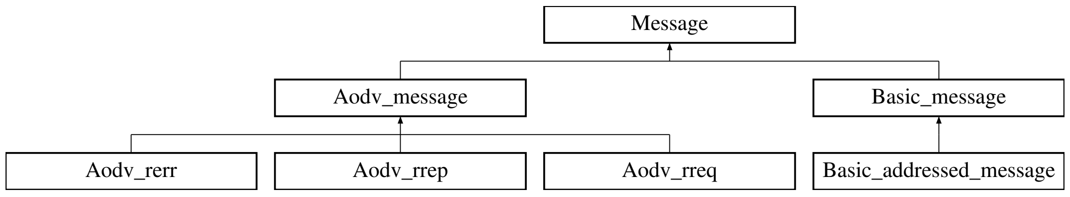
- int **m_first**
- int **m_second**
- int **m_third**
- int **m_fourth**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/IpAllocator.hpp
- code/liboctodrone/src/IpAllocator.cpp

## 6.27 Message Class Reference

Inheritance diagram for Message:



### Public Member Functions

- **Message** (std::string type)
- time_t **get_current_time** ()
- virtual std::string **to_string** ()=0
- std::string **get_type** ()
- **Message** (std::string type)
- time_t **get_current_time** ()
- virtual std::string **to_string** ()=0
- std::string **get_type** ()

### Private Attributes

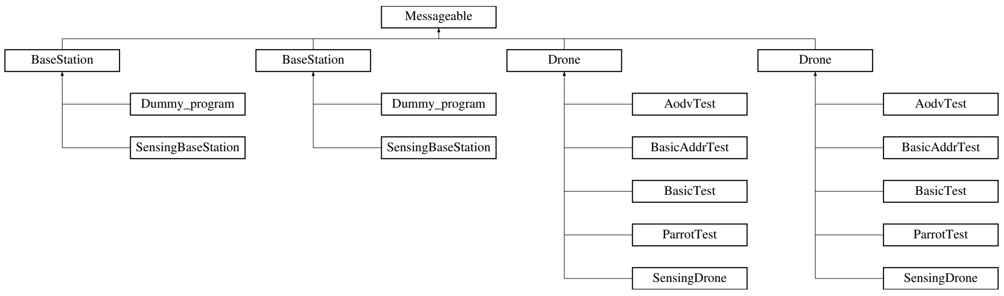
- std::string **type**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/Message.hpp
- code/liboctodrone/src/Message.cpp

## 6.28 Messageable Class Reference

Inheritance diagram for Messageable:



### Public Member Functions

- **Messageable** (CommMod ∗cm, double xp, double yp, double zp)
- void **send_message** (Message ∗contents)
- Message ∗ **wait_for_message** ()
- void **push_message** (Message ∗contents)
- void **receive_message** (std::string contents)
- CommMod ∗ **get_comm_mod** ()
- double **getX** ()
- double **getY** ()
- double **getZ** ()
- Coord **getPosition** ()
- double **getTime** ()
- virtual bool **message_callback** (Message ∗message)=0
- bool **getAlive** ()
- void **run_wrapper** ()
- virtual void **run** ()=0
- void **runCommMod** ()
- **Messageable** (CommMod ∗cm, double xp, double yp, double zp)
- void **send_message** (Message ∗contents)
- Message ∗ **wait_for_message** ()
- void **push_message** (Message ∗contents)
- void **receive_message** (std::string contents)
- CommMod ∗ **get_comm_mod** ()
- double **getX** ()
- double **getY** ()
- double **getZ** ()
- Coord **getPosition** ()
- double **getTime** ()
- virtual bool **message_callback** (Message ∗message)=0
- bool **getAlive** ()
- void **run_wrapper** ()
- virtual void **run** ()=0
- void **runCommMod** ()

### Protected Attributes

- std::queue< Message ∗ > **inQueue**
- CommMod ∗ **communicationsModule**
- Coord **position**

### Private Attributes

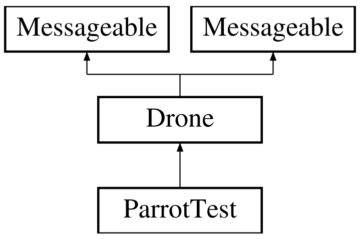
- bool **alive**

The documentation for this class was generated from the following files:

- code/liboctodrone/include/Messageable.hpp
- code/liboctodrone/src/Messageable.cpp

## 6.29 ParrotTest Class Reference

Inheritance diagram for ParrotTest:



### Public Member Functions

- **ParrotTest** (CommMod ∗, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

### Private Attributes

- bool **m_sink_node**

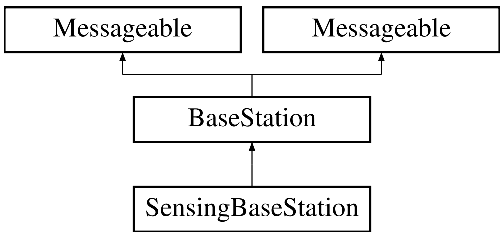
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/ParrotTest.hpp
- code/programs/src/ParrotTest.cpp

## 6.30 SensingBaseStation Class Reference

Inheritance diagram for SensingBaseStation:



### Public Member Functions

- **SensingBaseStation** (CommMod ∗cm, double xp, double yp, double zp, double areaX1, double areaY1, double areaX2, double areaY2)
- void **run** ()
- bool **message_callback** (Message ∗message)

### Private Member Functions

- void **interpretMessage** (Message ∗message)

### Private Attributes

- std::vector< std::string > **droneIPs**
- double **areaX1**
- double **areaX2**
- double **areaY1**
- double **areaY2**

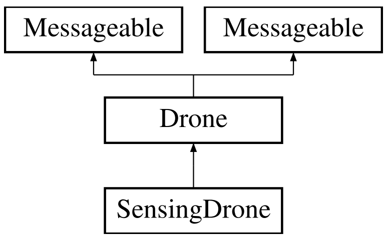
### Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/SensingBaseStation.hpp
- code/programs/src/SensingBaseStation.cpp

## 6.31 SensingDrone Class Reference

Inheritance diagram for SensingDrone:



### Public Member Functions

- **SensingDrone** (CommMod ∗, double, double, double, double, double, Environment ∗, bool)
- bool **message_callback** (Message ∗)
- void **run** ()

## Private Member Functions

- void **continueJob** ()
- void **interpretMessage** (Message ∗message)
- void **sendDataPoint** (double, double, double, double)
- int **atLoc** (Coord location)
- void **newArea** (double x1, double y1, double x2, double y2, double height)
- void **quit** ()

## Private Attributes

- int **m_task**
- int ∗ **m_flag**
- double **sensorRadius**
- int **routingPriority**
- double **lastTime**
- double **waitTimer**
- bool **waiting**
- std::queue< Coord > **remainingPoints**
- bool **sink_node**
- std::string **baseStationIP**
- std::vector< std::string > **droneIPs**

## Additional Inherited Members

The documentation for this class was generated from the following files:

- code/programs/include/SensingDrone.hpp
- code/programs/src/SensingDrone.cpp

# Index

APPENDIX B

# Original Project Specification

# CS407 Specification

Ben De Ivey, Will Seymour, Jon Gibson, Alex Henson

December 10, 2015

## 1 Introduction

Airborne drones, as referred to in this project, are unmanned aerial vehicles (UAV) capable of sensing their surroundings and performing some processing or computation in an automated fashion. They are used for a variety of purposes, including filming[3], exploration[4], and even selfie taking[1]. Along with the numerous uses of drones in the military setting[7], the applications for a network of monitoring drones are broad. For this project, the type of airborne drones that will be used and talked about are smaller and more compact than those, for example, that are military UAVs.

Over the last couple of decades, research and development has been done looking into replacing humans with drones in potentially dangerous situations[2][6]. Unstable mines, flooded tunnels, or areas with high levels of radiation of harmful gas are examples of places where it is unsafe for a human to go, but fine for a drone. With suitable sensors and algorithms, it is possible for drones to perform the same tasks as humans in these situations but in a much safer way.

While the main focus of this project is create a framework for a drone network to undertake a specified task, an application of this project is firefighting in situations too dangerous, expensive or impractical or human to perform. For example, California frequently has issues with forest fires. It has been shown that 80%-96% of area burned by forest fires in California were caused by 1% of the largest fires[5]. Unmanned flying drones in a network, capable of detecting initial outbreaks of a fire, could potentially suppress the fire before the situation were to get worse, either entirely dealing with the fire, or giving the fire response teams longer to reach the site and less danger when they arrive.

In this project, we propose to tackle the problem with a drone network, where if any one drone detects a fire, then that information is immediately broadcast to others, and appropriate action taken.

## 2 Overview and Intended Functionality

The intended outcome of the project is to accurately model airborne sensor networks (drone networks) operating over a well defined area of physical space. While the fire fighting case study presented will target a specific problem, the software produced as part of the project will operate on more general problems. At a high level, our simulated networks will be able to monitor a range of different properties using any continuous linear sensors, returning data to the fixed base station from which they were tasked. Properties might be tasked as compiling a map of a property over an area or finding only the maximum value of a property within an area (such as temperature). It will also be possible to specify actions to perform when certain conditions are met (such as when a fire has got out of control).

The network model will solve the problem of work division, with individual nodes electing a leader and having that leader distribute the tasked workload between the swarm. The other main problems associated with modelling mobile sensor networks are the routing of individual nodes and the efficient transmission of communications data over long distances. In terms of directing individual nodes, drones will be assigned airspace and maintain a model of which units are operational in each area, negotiating passing space when required. When a node needs to communicate data back to the base station it will choose an appropriate method of doing so. This may involve connecting directly to the base station or by passing the message to a nearby drone which is closer to the base station.

The network simulation should also have a concept of fault tolerance. If an individual node fails then the system should be able to detect this and reallocate work without compromising the integrity of the data collected. A node may fail in a number of ways: becoming unresponsive, providing incorrect responses, providing responses outside of the specified time intervals, and in other ways as arbitrary (or byzantine) failure.

If time and resources permit, we intend to deploy the developed software to physical drones in order to better demonstrate the network. This will show how the network reacts to real world conditions which is especially relevant in terms of fault tolerance.

## 3 Interfaces

In order to simulate a drone network and formulate the interaction between drones and the base station, it will be necessary to use various different interfaces to achieve the intended functionality. The main program which will be used is NS-3, free software which provides discrete event network simulation, which can construct a theoretical simulation of the drone network and the tasks that it will perform. NS-3 will be useful for a number of reasons. Firstly, using NS-3 will allow for the testing of how the physical drones will/should perform in practice, simultaneously giving a framework of the code with which they will be programmed. NS-3 uses Python or C++, providing full support for each language, and therefore the creation of a theoretical simulation in NS-3 would also equate to a skeleton program for the coding of the physical drones, in the event that these languages are used for drone functionality. Additionally, if it is the case that the intended outcome of the project is not fully realised on a physical level, it will still be possible to create a scalable model solution for airborne sensor networks for a given problem set.

However, NS-3 is merely a library which provides a set of network simulation models that users can interact with to set up a simulation and provide an output. In other words, because these events are simply function calls, it is also necessary to use a graphical front end for NS-3, so that the simulation can be readily visualised. For this task, NetAnim will be used. NetAnim is an offline animator which animates a simulation using an XML trace file collected during simulation, creating a GUI that will give information and statistics about the simulation. The advantage of using NetAnim is that it will not only provide a visual display for the simulation, but it will be possible to step through a simulation one event at a time and pause the simulation at a given time, as well as printing routing tables and nodes at various points in time. These features will allow for much more flexibility than standard output alone.

The drones which will be used in the project are Parrot AR 2.0 Power Edition quadcopter drones. Given that these drones are not in and of themselves programmable and autonomous, a Raspberry Pi or Arduino (small, single-board computers) will be mounted onto the drone itself and programmed in order to give instructions to the drone for basic autonomous functionality, including any sensor manipulation which may be required for the project. Finally, in order to perform the task of interaction between the drone and its base station, a custom-made language will be required such that problem detection and correction can be performed autonomously. This language will be responsible for processing any information passed by the drain to the base station, as well as handling the subsequent commands which will be passed back to the drone.

## 4 Scheduling

The project can be broken down into N major components: Network communications, directing drones in physical space, specifying and executing input, deployment to real drones. These sections are estimated to be of roughly equal size. Networking and physical routing will be developed in parallel, with the intention of having them complete by the start of term two. During term two, the tasking logic will be constructed, along with the method by which the user inputs instructions for the network to complete.

## 5 Management and Development Strategies

Throughout the course of the project, the team will partake in weekly meetings with the project supervisor. Progress tracked throughout the project against the scheduling plan in section 4 will ensure that the project's time is managed appropriately.

At the midway point through the project, a progress report will be written with details for the interested parties of how the project is progressing, any revised development plans, and a revised and detailed specification.

Due to the small size of the project team and the fluctuating nature of time each team member has available, a development methodology inspired by Agile methods will be used. Two week sprints ending in a meeting with the project supervisor, with another meeting half way through the sprint, enable the progress of the project in a flexible but structured way. Sprint goals can be decided upon at the start of the fortnightly period based on the current state of the project, as well as the estimated available time of the project members during the period.

To manage the project's material, a git repository will be created, ensuring consistency amongst the project members' work, and safety for the code base. Not only does this reduce duplication of work, but it also prevents data loss in the event of the unexpected. Additionally it allows code to be reverted if updates break previous functionality.

## References

[1] Lily Robotics Inc. Lily - camera. reinvented, 2015.

[2] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *Pervasive Computing, IEEE*, 3(4):24–33, 2004.

[3] Skyvantage LTD. Stunning aerial filming  photography flying uav drones, london uk, 2015.

[4] Gem Systems Advanced Magnetometers. Uavs - pathway to the future - gem systems, 2015.

[5] Stephen J Pyne et al. *Fire in America. A cultural history of wildland and rural fire.* Princeton University Press, 1982.

[6] Niramon Ruangpayoongsak, Hubert Roth, and Jan Chudoba. Mobile robots for search and rescue. In *Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International*, pages 212–217. IEEE, 2005.

[7] Noel Sharkey. The automation and proliferation of military drones and the protection of civilians. *Law, Innovation and Technology*, 3(2):229–240, 2011.

APPENDIX C

# Bibliography

[1] Stuart M Adams and Carol J Friedland. A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management, 2011.

[2] Arduino. What is arduino? https://www.arduino.cc/en/Guide/Introduction.

[3] Arduino. What is degrees of freedom, 6dof, 9dof, 10dof, 11dof, may 2012.

[4] Civil Aviation Authority. Small unmanned aircraft, dec 2015. https://www.caa.co.uk/Commercial-Industry/Aircraft/Unmanned-aircraft/Small-unmanned-aircraft/.

[5] He Bin and Amahah Justice. The design of an unmanned aerial vehicle based on the ardupilot. *Indian Journal of Science and Technology*, 2(4):12–15, 2009.

[6] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *Computer*, (5):59–67, 2000.

[7] Shelby Carpenter. Fighting forest fires before they get big - with drones, jun 2015. http://www.wired.com/2015/06/fighting-forest-fires-get-big-drones/.

[8] Ian D Chakeres and Elizabeth M Belding-Royer. Aodv routing protocol implementation design. In *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on*, pages 698–703. IEEE, 2004.

[9] Jim Wright Chris Cole. What are drones?, jan 2010.

[10] Mani B. Srivastava Curt Schurgers. Energy efficient routing in wireless sensor networks. Technical report, University of California at Los Angeles (UCLA), 2010.

[11] Nick Dijkshoorn and Arnoud Visser. Integrating sensor and motion models to localize an autonomous ar. drone. *International Journal of Micro Air Vehicles*, 3(4):183–200, 2011.

[12] Joseph Dussault. 7 commercial uses for drones, mar 2014.

[13] Marcus Faires. Domino's launches world's first driverless pizza delivery vehicles, apr 2015.

[14] Jon Fingas. Us army hopes to outfit soldiers with tiny drones by 2018, apr 2016. http://www.engadget.com/2016/04/04/army-mini-drones/.

[15] U.S. Air Force. Mq-9 reaper, sep 2015.

[16] Raspberry Pi Foundation. Raspberry pi hardware. https://www.raspberrypi.org/documentation/hardware/raspberrypi/.

[17] Fuego. Fire urgency estimation from geosynchronous orbit, may 2015. https://fuego.ssl.berkeley.edu/.

[18] Yash Garg. Knowledge base: What are rtf, bnf and arf drone kits?, feb 2015.

[19] John Paulin Hansen, Alexandre Alapetite, I. Scott MacKenzie, and Emilie Møllenbach. The use of gaze to control drones. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14, pages 27–34, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2578153.2578156.

[20] National Instruments. What is a wireless sensor network?, may 2012.

[21] David Johnson, Y Hu, and David Maltz. Rfc for the dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4. Technical report, 2007.

[22] David B. Johnson, David A. Maltz, and Josh Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pages 139–172. Addison-Wesley, 2001.

[23] Mitch Johnson. Components for creating an unmanned aerial vehicle. Online Manual, mar 2015.

[24] Brad Karp and Hsiang-Tsung Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM, 2000.

[25] Taieb Znati Kazem Sohraby, Daniel Minoli. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley, 2007.

[26] Frederic Guinand Luc Hogie, Pascal Bouvry. An overview of manets simulation. *Electronic Notes in Theoretical Computer Science*, 150(1):81–101, 2006.

[27] Kamin Whitehouse Luca Mottola, Mattia Moretta and Carlo Ghezzi. Team-level programming of drone sensor networks. Technical report, SICS Swedish ICT, 2014.

[28] Yajie Ma, Yike Guo, Xiangchuan Tian, and Moustafa Ghanem. Distributed clustering-based aggregation algorithm for spatial correlated sensor networks. *Sensors Journal, IEEE*, 11(3):641–648, 2011.

[29] Robert Cecil Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[30] Luca Mottola, Mattia Moretta, Kamin Whitehouse, and Carlo Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190. ACM, 2014.

[31] Colin Neagle. You can't shoot a drone, so what can you do if it invades your privacy?, jun 2015. http://www.networkworld.com/article/2941952/opensource-subnet/you-cant-shoot-a-drone-so-what-can-you-do-if-it-invades-your-privacy.html.

[32] ISO Norm. Iec 7498-1: 1994 (e). *Information technology–Open Systems Interconnection–Basic Reference Model: The Basic Model*.

[33] ns. ns-2 & ns-3, oct 2006.

[34] ns. Netanim, jan 2016.

[35] OMNet++. Omnet++ home page, apr 2016.

[36] Paparazzi. Paparazzi overview, jan 2016.

[37] Tommaso Pecorella. Introduction - tutorial - ns-3, apr 2016.

[38] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Rfc for ad hoc on-demand distance vector (aodv) routing. Technical report, 2003.

[39] Charles E Perkins, Elizabeth M Royer, Samir R Das, and Mahesh K Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. *Personal Communications, IEEE*, 8(1):16–28, 2001.

[40] Stephane Piskorski, Nicolas Brulez, Pierre Eline, and Frederic DâĂŹHaeyer. Ar. drone developer guide. *Parrot, sdk*, 1, 2012.

[41] Markus Quaritsch, Karin Kruggl, Daniel Wischounig-Strucl, Subhabrata Bhattacharya, Mubarak Shah, and Bernhard Rinner. Networked uavs as aerial sensor network for disaster management applications. *e & i Elektrotechnik und Informationstechnik*, 127(3):56–63, 2010.

[42] Jason Redi Ram Ramathan. A brief overview of ad hoc networks: Challenges and directions. *IEEE Communications Magazine*, pages 98–102, 2012.

[43] Bart Remes, Dino Hensen, Freek van Tienen, Christophe De Wagter, Erik van der Horst, and Guido de Croon. Paparazzi: how to make a swarm of parrot ar drones fly autonomously based on gps. *International Micro Air Vehicle Conference and Flight Competition (IMAV2013)*, pages 17–20, sep 2013.

[44] Khanh Pham Robert Sivilli, Yunjun Xu. Pathfinding for mobile sensor networks on the fly. *SPIE Newsroom. DOI: 10.1117/2.1201209.004486*, oct 2012.

[45] RT. Pentagon admits using drones to spy on americans, mar 2016. https://www.rt.com/usa/335068-pentagon-drones-spy-americans/.

[46] Mary-Ann Russon. Dark side of the drone: Police reveal uavs being used for theft, smuggling and spying on children, oct 2015. http://www.ibtimes.co.uk/dark-side-drone-police-reveal-uavs-being-used-theft-smuggling-spying-children-1523662.

[47] Parrot SA. Ar drone 2 techincal specifications, 2012. http://ardrone2.parrot.com/ardrone-2/specifications/.

[48] Mina Vajed Khiavi Sajjad Jahanbakhsh Gudakahriz1, Shahram Jamali. Energy efficient routing in mobile ad hoc networks by using honey bee mating optimization. *Journal of Advances in Computer Research*, 3(4):77–87, nov 2012.

[49] Victor Sanchez. Sensor networks and mobile data communications. Lecture Series, 2015.

[50] Jaydip Sen. A robust and efficient node authentication protocol for mobile ad hoc network. In *Proceedings of the 2nd International Conference on Computational Intelligence, Modelling and Simulation*, pages 476–481, Bali, Indonesia, sep 2010.

[51] VA Amala Shiny and V Nagarajan. Energy efficient routing protocol for mobile wireless sensor networks. *International Journal of Computer Applications (0975 âĂŞ 8887)*, 43(21), 2012.

[52] Ian Sommerville. *Software Engineering.* Pearson, Boston, ninth edition, 2010.

[53] Michael Colagrosso Stuart Kurkowski, Tracey Camp. Manet simulation studies: The incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50–61, 2005.

[54] Quest UAV. Uav sensors, jun 2015.

[55] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[56] Arnoud Visser, Nick Dijkshoorn, M Veen, Robrecht Jurriaans, et al. Closing the gap between simulation and reality in the sensor and motion models of an autonomous ar. drone. *International Micro Air Vehicle conference and competitions 2011 (IMAV 2011)*, 2011.

[57] Wikipedia. Unmanned aerial vehicle, 2016.

[58] Sonia McNeil William Marra. Understanding 'the loop': Regulating the next generation of war machines. Technical report, Harvard law school, 2012.

[59] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pages 154–161. IEEE, 1998.