MULTI-AGENT AERIAL MAPPING USING STRUCTURE FROM MOTION AND WIRELESS MESH NETWORKS

by

Raunak Mukhia

A thesis submitted in partial fulfillment of the requirements for the
degree of Master of Engineering in
Computer Science

Examination Committee:	Dr. Matthew N. Dailey (Chairperson) Dr. Adisorn Lertsinsrubtavee Dr. Mongkol Ekpanyapong
Nationality:	Indian
Previous Degree:	Bachelor of Technology in Computer Science and Engineering Sikkim Manipal Institute of Technology, India

Asian Institute of Technology
School of Engineering and Technology
Thailand
December 2020

Acknowledgments

Write your touching message here..

Abstract

Abstract here..

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The availability and financial accessibility of unmanned aerial vehicles (UAVs) have made them more widespread, with applications ranging over a wide range of civilian activities. Multi-rotors are used for recreational flying, research, cinematography, disaster observation, logistics, agriculture, public safety, construction, surveillance, and environmental protection, amongst other purposes.

A cluster of inexpensive autonomous multi-rotors connected through a wireless mesh network that can be deployed in a post-disaster situation could help avoid dangerous situations faced by ground-based human observers in such scenarios. The drones should be able to perform aerial mapping quickly, as they can coordinate so that one drone does not need to visit locations already visited by other drones in the cluster. The range of the cluster can also be large, since each drone would act as a wireless mesh access point. At the same time, each drone could act as an access point to provide connectivity in the disaster-stricken area.

Aerial mapping drones can be outfitted with an array of sensors such as ultrasonic sensors, infrared sensors, multi-array laser sensors, and RGB-D cameras. However, systems with many sensors will suffer from integration complexity, weight constraints, reduced battery lifetime, and high cost. Structure from motion (SfM) using monocular cameras attached to each drone provides a low cost, lightweight, simple alternative to sensor-intensive approaches.

This study proposes a system for command and control of a cluster of autonomous drones, each being a node in a wireless mesh network, controlled from a centralized control station that oversees coordinated exploratory path planning and aerial mapping using SfM.

1.2 Problem Statement

An inexpensive cluster of UAVs providing autonomous coordinated aerial mapping after a disaster can be a valuable asset for disaster observation and public safety that does not pose any risks for ground-based surveillance personnel. Multiple drones will reduce flight time and increase the spatial extent of the SfM result. A wireless mesh network can also be an inexpensive solution when communication infrastructure such as cellular data services are dysfunctional. Increasing the number of drones in the system can enable increased range of the mesh network, because each drone acts as a mesh node. Providing emergency network services using UAVs and mesh networks in disaster-stricken areas is an active research area, as discussed by Chand et al. (2018), but the best method to combine coverage

for mapping while maintaining mesh connectivity is an open problem. Sabino et al. (2018) discuss optimal placement of UAVs in a mesh network, and seperately, a survey of the multi-agent complete coverage problem (CCP) by Almadhoun et al. (2019) reviews many CCP methods proposed in the literature. However, the requirement for mesh network connectivity imposes severe constraint on the movement of agents, making the resulting "constrained CCP" an open research area.

Simulation environments exists that can be used for multi-drone planning and wireless mesh networks. Pixhawk PX4's software in the loop (SITL) along with Gazebo can simulate a physical fleet of UAVs with cameras. Wireless mesh networks can be simulated by Network Simulator 3 (NS3). The Robot Operating System (ROS), with its range of tools and available packages, provides a strong platform to develop this system. FlySimNet by Baidya et al. (2018) is a UAV network simulator created by combining NS3 and Ardupilot's UAV SITL simulator with lightweight publish/subscribe (ZMQ) based middleware. It uses a custom graphical ground control system (GCS) application that communicates via the ZMQ messaging service, which does not support ROS, and cameras are not supported by Ardupilot SITL. The CORNET middleware framework by Acharya RBCCPS et al. (2020) combines Gazebo, NS3 and PX4, but it is available only as a published paper and implementation of their method is not quite clear. Neither of these frameworks implement wireless mesh network simulation on NS3. Gazebo, NS3, PX4, and ROS together provide an open source platform, but integration of these tools is required.

This thesis aims to design and implement a multi-agent CCP method that satisfies wireless mesh network constraints, runs multi-agent SfM. It also aims to develop a simulation framework to support the CCP method.

1.3 Objectives

The main objective of this thesis is to improve on the state of the art in mapping disaster-stricken areas by designing and implementing a method for autonomous control of a multi-agent exploration and coordinated aerial mapping team performing SfM. The focus of this study is to map a disaster-stricken area with the possible damage to communication infrastructure; therefore, the system will use a wireless mesh network for internal communication, with each agent working as a mesh node. The objectives can be decomposed into the following specific tasks:

- Design and implement a user-friendly pipeline utilizing the Robot Operating System (ROS) that allows the operator to select an area to survey.
- Design and implement strategies to find the transformations between the global map and the local map of each drone continuously such that the control station can coordinate the position of each drone with respect to the global map.
- Design and implement a method for establishing a reliable communication channel between each drone and the control station over the mesh network.
- Design and implement an autonomous exploration and coordination system for the agents that operates within the restriction of the range of the wireless mesh network. The system should maximize the extent of the mappable area, by planning an appropriate arrangement of agents over time.

- Implement a multi-agent SfM method capable of generating an aerial map of the area under surveillance.
- Design and implement a simulation framework for the entire system using PixHawk PX4 as the drone flight controller, Gazebo and Network Simulator 3 as simulation tools, ROS for overall coordination, and useful robotics/vision algorithms.
- Design and implement a wireless mesh network with each drone as a mesh node. As the drones are fast-moving agents, this will require the study and implementation of solutions to maintain a reliable network with a usable quality of service, despite constant topology changes.
- Evaluate the system's ability to execute effectively in the real world.

1.4 Limitations and Scope

The study will not cover the following.

- Obstacle detection and avoidance: The control station will generate paths for the drones, such that they will not collide with each other. It is assumed that no unknown obstacle will be present at the drone's operational altitude. It is the operator's responsibility to indicate the minimum altitude during mapping.
- Dynamic path reconfiguration: The control station will generate complete paths for the drones before the start of the mission.
- Recovery mechanism: If communication is lost with any drone, it will return to the home position using the default behavior in PX4. The remaining drones will continue their operation.
- Global Positioning System failure: It is assumed that GPS will be continuously functional.

The scope of this study is to design and implement a system capable of exploratory coverage surveys using multiple drones using the wireless mesh network as a communication backbone. SfM is run as an application on top of this system to generate a map. The system developed for this thesis can potentially be given a wider scope by developing other applications that run on top of the system, such as an application that generates a heat map to detect survivors after a disaster, autonomous aerial seeding of crops, spraying pesticide in agricultural farms, and other tasks that require a group of drones to cover a geo-bounded area.

1.5 Thesis Outline

I organize the rest of this proposal as follows.

In Chapter 2, I provide a literature review.

In Chapter 3, I propose my methodology.

In Chapter 4, I present preliminary experimental results.

Finally, in Chapter 5, I provide a work plan for the rest of the thesis.

Chapter 2

Literature Review

This chapter begins with a review of the functional architecture of autonomous driving systems, coordinate systems, the PixHawk, and ROS coordinate conventions.

2.1 Software Architecture of Autonomous Driving Vehicles

Behere and Trngren (2016) splits the major components of the motion control part of autonomous driving systems into three main categories as shown in Figure 2.1. These categories are

- Perception of the external environment in which the vehicle operates
- Decision making and control of vehicle motion with respect to the external environment that is perceived
- Vehicle platform manipulation, which deals mostly with sensing, control, and actuation of the vehicle, with the intention of achieving the desired motion.

Each category can be further broken down into several components. I describe each component in more details in the following sections.

2.1.1 Perception

Sensing components sense the state of the vehicle and the state of the environment in which the vehicle operates. The sensor fusion component considers multiple sources of information to construct a hypothesis or belief about the state of the environment. The localization component determines the location of the vehicle with respect to a global map. The semantic understanding component processes the sensor input and derives meaningful information from it. The world model component holds the current estimate of the state of the external environment.

2.1.2 Decision and Control

The trajectory generation component repeatedly generates a set of obstacle-free trajectories in the world coordinate system and picks an optimal trajectory from the set. Energy management components deal with energy management of the vehicle. Diagnosis and fault management monitors the state of the overall system and its components. Reactive control components are used for immediate responses to unanticipated stimuli from the environment. The vehicle platform abstraction component refers to a minimal model of the vehicle platform.

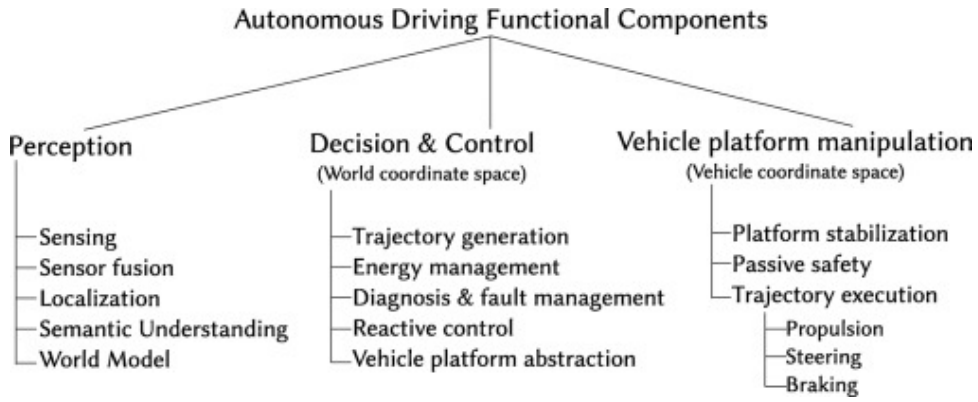

Figure 2.1: Functional architecture of an autonomous driving system. Reprinted from Behere (2016).

2.1.3 Vehicle Platform Manipulation

The platform stabilization component's task is to keep the vehicle platform in a controllable state during operation. Trajectory execution components are responsible for executing the trajectory generated by the decision and control component.

2.2 Coordinate Systems

In this section I describe the different coordinate system conventions in use in robotic applications.

2.2.1 Earth Centric, Earth Fixed

The earth-centered earth-fixed (ECEF) coordinate system, rotates with the Earth and has its origin at the center of the Earth. Refer to Figure 2.2 (a) for a visualization. ECEF follows the right handed coordinate system convention. Wikipedia (2019) describes ECEF as follows.

- The origin is at the center of mass of the Earth, a point close to the Earth's center.
- The Z axis is on the line between the magnetic north and south poles, with positive values increasing northward (but not exactly coinciding with the Earth's rotational axis).
- The X and Y axes lie in the plane of the equator.
- The X axis passes through the point on the equator from 180 degrees longitude (negative) to the prime meridian (0 degrees longitude, positive).
- The Y axis passes through the point at 90 degrees west longitude along the equator (negative) to 90 degrees east along the equator (positive).

2.2.2 World Geodetic System

The World Geodetic System is a standard system used in GPS, cartography and geodesy. The latest revision is WGS 1984 (WGS 84) that was established and is maintained by the United States National Geospatial-Intelligence Agency. It was last revised in 2004. The World Geodetic System 1984 (WGS 84) is one of the best global geodetic reference system for the Earth available presently. The datum (ellipsoid) defined by WGS 84 is generally used to convert the GPS latitutes and longitude to the UTM coordinate system.

2.2.3 Universal Transverse Mercator Geopraphic Coordinate System

The Universal Transverse Mercator (UTM) is a coordinate system that divided the earth into 60 zones. Each zone is a plane with its own x and y coordinates in meters. It ignores altitude and treats the earth as a perfect ellipsiod. A UTM coordinate has:

- A zone number.
- A hemisphere (N/S).
- An easting (X coordinate).
- A northing (Y coordinate)

A coordinate 14.08057584 °N, 100.61284553 °E in latitude and longitude is represented as 47 N 674132 1557234 in UTM.

2.2.4 Local tangent plane

In the local tangent plane coordinate system, a position on the earth is fixed about the origin. There are two conventions as shown in Figure 2.2 (b).

- X=East, Y=North, Z=Up (ENU).
- X=North, Y=East, Z=Down (NED), which is commonly used in aviation, as the objects of interest usually lie before an aircraft (down or positive Z).

2.2.5 PixHawk and ROS coordinate systems

PixHawk follows the NED convention, whereas ROS follows the ENU convention. The conversion between these different conventions is handled automatically by MAVROS. To translate air-frame

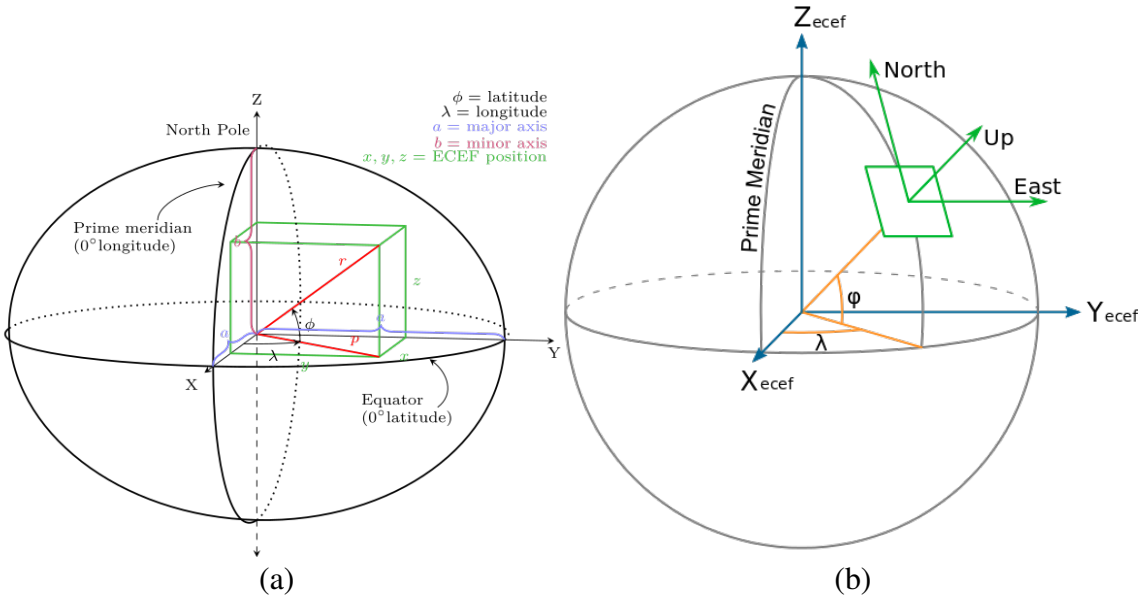

Figure 2.2: Coordinate systems. (a) Earth centric, earth fixed (ECEF) coordinate system. (b) Local tangent plane coordinate systems. Reprinted from Krishnavedala.

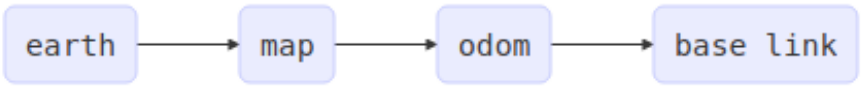

Figure 2.3: Relationship between ROS frames. Reprinted from Meeussen (2010).

related data, a rotation of 180° degrees is applied about the roll (X) axis. For local data, a 180° rotation is applied about the roll (X) and 90° rotation is applied about the yaw (Z) axes.

It is good practice in ROS to set up reference frames as described by Meeussen (2010), shown in Figure 2.3.

- The coordinate frame called base_link is rigidly attached to the mobile robot base. It can be attached in any arbitrary position or orientation.
- The coordinate frame called odom is a world-fixed frame. The pose of a robot is continuous in this frame, but it can drift over time.
- The coordinate frame called map is a world fixed frame, with Z-axis pointing upwards.
- The coordinate frame called earth is the origin of the ECEF frame.

2.3 Wireless Mesh Network

A wireless mesh network's (WMN) consists of two types of nodes, mesh routers and mesh clients. Mesh routers form the mesh backbone for mesh clients. Every node in the mesh has the capability to route packets forward for other nodes if the packet destination is not in direct wireless transmission range. Mesh routers provide functionality as bridge/gateway that enables the WMN to integrate with other networks such as cellular, ad-hoc and ethernet. In a wireless mesh network, all the participating nodes automatically establish and maintain mesh connectivity and are in effect dynamically self-organized and self-configured. WMN can be rapidly deployed where there is no network coverage and has the potential for application in disaster management.

2.3.1 WMN Network Architecture

A wireless mesh network has two types of nodes, mesh routers and mesh clients. Mesh routers can have multiple networking interfaces to provide bridge/gateway functionality to the mesh network and have additional routing functions to support mesh networking. Mesh routers can be built on similar hardware platform as a conventional wireless router. Mesh routers can be build based upon embedded systems and general-purpose computer systems. Mesh clients have capability for mesh networking and can also work as a mesh router, nevertheless they lack bridge/gateway function and usually have only one wireless interface.

The architecture of WMNs can be classified into three types based upon the composition of mesh routers and clients.

- *Infrastructure/Backbone WMNs.* In this architecture mesh routers form an infrastructure for clients that connect to them. The mesh routers automatically establish and maintain links between themselves and provide gateway functionality to connect the mesh network to other networks and the internet.
- *Client WMNs.* In this architecture, mesh routers are not required for maintaining a mesh network. Mesh clients provide peer-to-peer networking, performing routing and configuration management. A packet may reach the destination node through multiples hops through client nodes.
- *Hybrid WMNs.* In this architecture, mesh routers provide infrastructure backbone with access to other networks. Client nodes can connect to the mesh network through meshing with other clients, as well as through mesh routers. This in effect, is a combination of infrastructure and client WMNs, and provides increased connectivity and coverage.

2.3.2 Features of WMNs

WMNs have redundant nodes and can provide high level of fault tolerance and robustness. If the case of link failure, the mesh network can adapt and reroute the data. Using multi-hop strategies,

the coverage range can be increased, as intermediate router and clients can relay packets between hosts which do not have a direct line of sight. WMNs can be rapidly deployed with less maintenance because of self-forming, self-healing and self-organization capability. They are compatible with ad-hoc wireless clients.

2.3.3 Routing Protocols for WMNs

WMNs has redundant and mobile clients, hence, the routing protocols should find and maintain routes between the source and destination nodes by detecting and responding to changes in network topology. Ideally routing protocols should maximize the capacity of the network and minimize the packet delivery delays by creating and selecting efficient routes between nodes.

Wireless mesh routing protocols can be classified into three categories:

- *Proactive Routing Protocol.* Proactive routing protocols build and maintain the routing table by sharing routing data among nodes at periodic intervals. It selects the route from the routing table while forwarding packets from one node to the other. Optimized Link State Routing Protocol (OLSR) and Better Approach to Mobile Ad-hoc Networks (BATMAN) falls under this category.
- *Reactive Routing Protocol.* Reactive routing protocols searches the route from source to destination node on demand as it is required. Examples of this category are Ad Hoc On-Demand Distance Vector Protocol (AODV) and Dynamic Source Routing Protocol (DSR).
- *Hybrid Routing Protocol.* Hybrid Routing Protocol uses both reactive and proactive routing strategies depending upon the environment. As a reactive routing protocol, it will provide the path on demand, based on up-to-date information. Examples of hybrid routing protocols are ZRP (zone routing protocol), HSLS (hazy sighted link state routing protocol).

2.3.3.1 Comparision of Routing Protocols

Mbarushimana and Shahrabi (2007) does a comparative study between the performance of reactive (AODV and DSR) and proactive (OSLR) protocols. They run simulations for 1000 seconds, considering four different scenerios, namely, workload, number of flows, number of nodes in the network, and node movement speed. The performance of the three routing protocols is measured based on throughput, goodput, routing load and end-to-end delay. They come to the conclusion that OSLR shows the best performance in terms of data delivery ratio and end-to-end delay, and provide insight that network layer tends to drop more packets while reactive protocol is computing the route to the destination. In their study OSLR still outperforms the reactive routing protocols at higher speeds even though it has the highest routing overhead due to its exchange of periodic routing updates.

Kulla et al. (2012) does a performance comparison of OSLR and BATMAN routing protocols in an indoor stairs environment of 5 floor building. They come to the conclusion that both the protocols perform well for one-hop and two-hop communication. The delay and packet loss increased after

three hops for static node scenario and after two hops for moving node scenario. BATMAN performs better than OSLR for packet loss metrics because BATMAN buffers packets when routes are unstable. In general, OSLR protocol showed better performance than BATMAN protocol. OSLR shows better performance than BATMAN regarding the delay metrics, which is critical for realtime communication between a cluster of drones.

From these studies of performance comparisons OSLR seems to be the most suited to my use case, as:

- Proactive protocols (OSLR) performs better than reactive protocols in terms of data delivery ratio and end-to-end delay.
- Reactive protocols tends to drop packets while they are computing the route.
- OSLR shows better performance than BATMAN for delay metrics.
- Latest information about the drones in the mesh network is more critical than stale information, which means OSLR with less delay is more suited to the use case of my study.

2.3.4 Optimized Link State Routing Protocol

Optimized Link State Routing Protocol (OSLR) belongs to the class of link state routing protocol, with optimization for mobile ad hoc networks. It is a distributed protocol with no central agency that exchanges topology information with other node of the network periodically. Each node maintains a routing table, and is a proactive protocol. When data is to be sent, it refers to the routing table. The route to other nodes are maintained by each node proactively.

HELLO message is used for link sensing, neighbor detection, and multipoint relay (MPR) selection. HELLO messages are not forwarded by the nodes. The nodes in OSLR use only selected nodes called multipoint relay (MPR) to re-transmit control messages, reducing overhead from control messages flooding the network. Each node in the network selects a set of nodes in its symmetric 1-hop neighborhood called MPR set of that node. Neighbors which do not belong to MPR set of the node receive and process broadcast messages but do not re-transmit it. If an interface of a node has at least one verified bi-directional link with an interface of another node, it is considered as its symmetric 1-hop neighbor.

Topology Control (TC) message is used to disseminated topology information through the network, using optimized flooding through MPR mechanism. The TC messages distributed to each node is used to calculate routing table in each node.

If a node has multiple interfaces, it must announce, periodically, information describing its interface configuration to other nodes in the network. Multiple Interface Declaration (MID) message is flooded by such nodes using MPR mechanism in the network, and is used for routing table calculations.

Gateway/bridge nodes have non OSLR interfaces connected to external network such as the ethernet. Host and Network Association (HNA) message are periodically broadcasted by such nodes to register such routing information in the routing table.

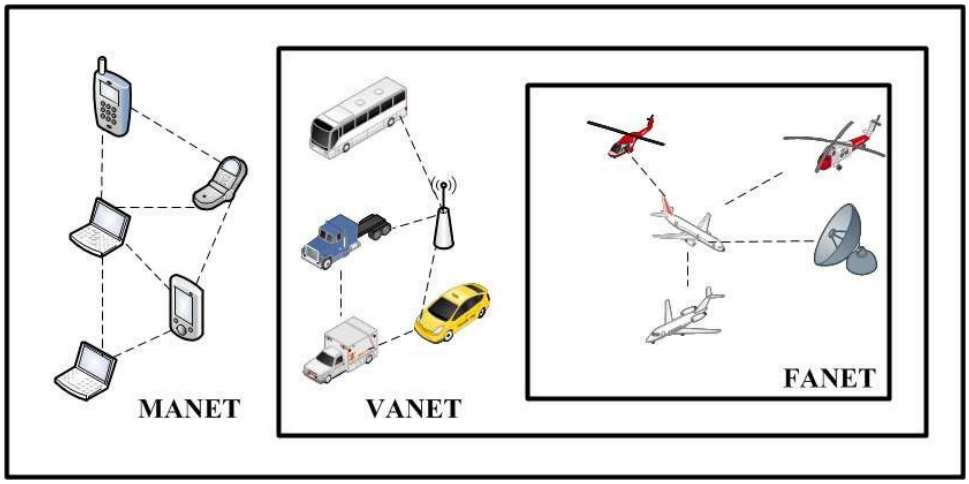

Figure 2.4: MANET, VANET,and FANET Reprinted from Tareque et al. (2015).

The role of OSLR is not to forward data packets. It maintains the routing table in each node which is used by the system to perform network layer forwarding function.

2.3.5 FANET

Flying ad hoc network (FANET) is a specialized subset of mobile ad hoc network(MANET). FANET takes into consideration the challanges faced by MANET with flying UAVs as the mesh nodes. FANET can also be classified as a subset of Vehicular ad hoc network (VANET), where the mesh nodes are vehicles on the ground. The relation between MANET, VANET and FANET is given in figure 2.4.

Surveys of FANET has been carried out by Chriki et al. (2019) and Tareque et al. (2015). The challenges of FANET described can be summarized as follows:

- *Network topology change.* In FANET, rapidly moving UAVs changes the network topology more frequently.
- *Node mobility.* UAVs have a higher degree of mobility than vehicles or people on ground. Node mobility issues can be considered as the most significant difference between FANET and other ad hoc networks.
- *Node density.* Node density in FANET is lower than other MANETs. FANET nodes are spread across the sky. There are less number of average nodes in a unit area in FANET.
- *Radio propagation model.* MANET and VANET nodes are close to the ground and because of the environment, they may not have direct line of sight. FANET nodes are in the sky and for most cases have direct line of sight between the nodes.

- *Power Consumption.* For large UAVs power consumption is not a significant factor when designing the FANET. However for small UAVs the power consumed by FANET must be taken into consideration.
- *Localization.* UAVs because of its higher mobility degree may have inaccurate GPS position information as compared to other MANETs.

Routing protocols for FANET follows similar classification as WMNs and are as follows:

- *Static protocols.* These protocols have fixed routing tables.
- *Proactive protocols.* Updates the routing tables periodically.
- *Reactive protocols.* Finds path to the destination node when data needs to be sent.
- *Hybrid protocols.* These use both reactive and proactive strategies when required.
- *Geographic based protocols.* These protocols use position information.

There are multiple optimization and extensions for OSLR specifically done to solve challenges for FANET. Some of them are:

- *DOLSR.* Alshabtat and Dong (2011) proposes Directional Optimized Link State Routing Protocol (DOLSR) which uses directional antenna and heuristic to minimize the number of MPRs. They show reduced end-to-end delay and enhanced overall throughout using DOLSR for FANET.
- *P-OSLR.* Rosati et al. (2013) proposes Predictive OSLR (P-OSLR) that uses GPS information to help the routing protocol. It take the relative speed between the nodes in account to calculated expected transmission count (ETX) metric. This metric is used by OSLR for link quality sensing. The show increased multi-hop reliability with minimum outage time as compared to OSLR in FANET environment.
- *ML-OSLR.* Zheng et al. (2014) proposes Mobility and Load aware OSLR (ML-OSLR) that introduces mobility and load aware algorithm in the routing protocol. ML-OLSR does not select high speed nodes as the MPR and avoid routing through high load, congested and high speed nodes to increase network stability. They show increased performance in packet delivery ratio and average end-to-end delay compared to OSLR with FANET based simulation.

Singh and Verma (2015) presents the application of OSLR to FANET under different mobility model of the nodes through simulation. He shows that the increase in speed of nodes decreases throughout, increases end to end delay and decreases packet delivery ratio for most mobility model. However for my study, the speed of the node are relatively low, and nodes spend most of their time waiting for other nodes to arrive to their waypoint and take picture, therefore I shall purse the use of generic OSLR for this thesis.

2.4 Simulators

An approximate imitation of a process or system's operation over time is known as simulation. A real-life or hypothetical situation modeled on a computer ro I how the system works is a computer simulation. Computer simulation can be a powerful tool to develop, investigate the behavior and save cost, by providing a means to virtually access the system under study.

Some of the classification of simulations are:

- *Continuous simulation.* Simulation is based on continuous time and uses numerical integration of differential equations.
- *Discrete-event simulation.* Simulation is based on discrete time intervals. The state of the components of the simulation change their value only at discrete time.
- *Stochastic simulation.* Simulation with same input will produce different results within some confidence interval. During simulation some variables or process is subject to random variations.
- *Deterministic simulation.* Simulation with same input will always produce the same deterministic results. The variables and process are regulated by deterministic algorithms.

2.4.1 Computer Network Simulators

Network simulators are those types of software that are capable of performing predictions on how a computer network will behave. The output of these simulations can be analyzed to derived metrics of the network under consideration. The use of network simulators provide a cost-effective method to validate and optimize the performance of the network before deployment. Network simulators may have the capability for emulation, that is, to pass real packets through the network simulation, and examine the effects of the network on the packets.

Siraj et al. (2012) has done survey on network simulators (ns-2, ns-3, OPNET, NetSim, OMNet++, REAL, J-Sim and QualNet), their availability, programming languages used and their architecture. Toor and Jain (2017) has done survey on network simulator which can support wireless infrastructure (ns-2, Ns-3, J-Sim, OMNeT++, OPNET, QUALNET and MATLAB) and listed their key features and limitations. Patel et al. (2018) has done survey on network simulator (ns-2, ns-3, OMNeT++, NetSim, REAL, OPNET and QualNet) and listed their features, advantages, disadvantages.

Network Simulator 3 (ns-3) is a discrete-event network simulator, and it's intended uses are for research and education. It is written from scratch using C++ and is not backward compatible with ns-2. NS-3 is a collection of core libraries, modules, and applications which are build using a python based build tool called waf. It can be linked with a user C++ or python application to create a simulation. It is possible to integrate ns-3 with other software through the user application. The architecture of ns-3 is illustrated in figure 2.5. In this paper I use ns-3 because:

- It is intended for scientific research.

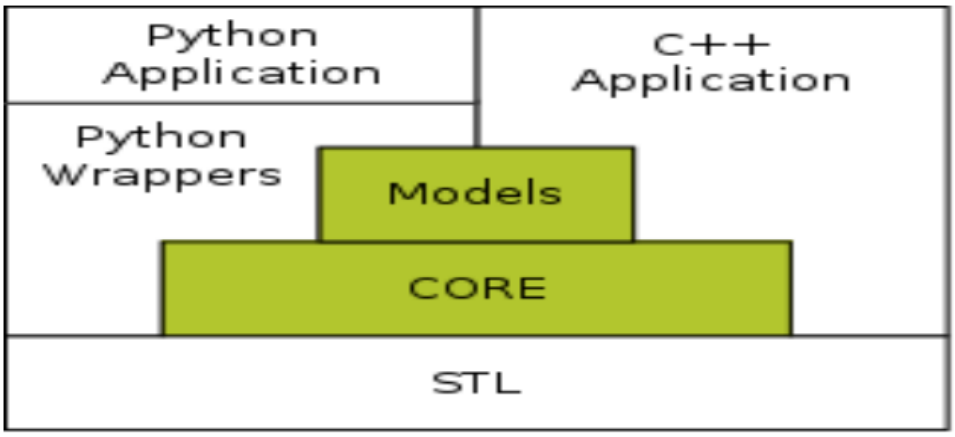

Figure 2.5: Ns-3 architecture. Reprinted from Siraj et al. (2012)

- It is more flexible than other simulators.
- It tries to simulate protocol genuinely.
- It is open source and free to use.
- It can be integrated with other software.
- It is actively developed and maintained. ns-2 development stopped around 2010.

2.4.2 Robotics Simulator and Robotic Software

Applications for physical robots can be created using robotics simulator without depending on the real hardware. This saves time and cost in the development of the application. Certain robotics simulator can render 3D model of robots and its environment, and can emulate robotic models, sensors, and control in virtual environment. They use physics engine to make the simulation more realistic.

Staranowicz and Mariottini (2011) has conducted a survey presenting comparison between the popular commercial and open-source robotic software for simulation and interfacing with real robots. Ivaldi et al. (2014) has conducted an online survey where participants present their feedback about the tools and use of dynamic simulation in robotics.

The most used robotics simulator are:

- *Gazebo.* Gazebo can simulate multiple robots in outdoor environments. It supports multiple physics engine. It can intergate with ROS using a set of ROS packages named gazebo_ros_pkgs. Gazebo is open source and free to use.
- *ODE.* Open Dynamics Engine is a open-source physics library for simulating rigid body dynamics. It is used in computer games and simulation tools. It is open source.

- *Bullet.* Bullet is an open-source physics library and is used in 3D animation software and game engines. It is open source.
- *V-Rep.* V-Rep is a robot simulator software developed by Coppelia Robotics. It is free for academic usage. Commercial license is not free.
- *Webots.* Webots is an open-source robot simulator developed by Cyberbotics. It was open sourced in 2018. Robots can be modeled, programmed and simulated with the provided development environment.

In this study, I intend to use Gazebo as the robotics simulator because:

- The UAV firmware PixHawk provides SITL simulation for Gazebo and is the recommended option.
- Gazebo supports PixHawk provided quad-copter (Iris).
- Gazebo supports multiple quad-copters.
- External world can be built in Gazebo using real world height maps.
- Gazebo supports sensors such as GPS and cameras.
- Gazebo can be integrated with ROS.
- Gazebo server provides API to integrate with other software. In my use case, I update the FANET nodes in ns-3 using the position provided by the robotics simulator.

2.5 Robot Operating System

Quigley et al. (2009) presents an overview of Robot Operating System (ROS). ROS is a framework for robotics software. It provides set of tools and communication layer on top of a host operating system, which helps in developing software for robotics. Availability of plethora of open source ROS packages avoids reinventing the wheel while developing the application.

ROS provides peer-to-peer connection to different processes of the robotics system. Each process is called a node and handles a specific aspect of the robotic system. The peer-to-peer topology makes use of a master node which enables the running nodes to find each other at run-time. Nodes exchange information with each other using messages. A message is a predefined strictly typed data structure. Nodes can exchange messages using either topic or service.

Nodes can publish message on a topic. Another node which uses the data generated by it can subscribe to the topic to receive messages. Nodes and topics have many-to-many association with each other. A single node can publish and subscribe to many topics and a topic may be published and subscribed by many nodes at once.

For synchronous communication, a node may call a service registered by another node. A node sends a request message to call a service and the node providing the service will reply with a response message. Unlike topics, service of a particular name can only be advertised by one node.

Nodes, topics and services can be grouped into namespaces. Topics and services follow URI reference naming convention. For example a topic that publishes GPS information for UAV 1 in a multi-UAV system may be named as */uav0/global_position/gps*.

ROS has tranformation system called tf/tf2. Tf2 is the new iteration of the transformation library. The transformation system helps the user keep track of multiple coordinate frames over time by using a tree structure to maintain relationship between coordinate frames. It allows for easy transformation of data between any two coordinate frames.

ROS has a wide array of tools for creating, debugging, inspecting and visualizing the data exchanged between the different nodes which speeds up the development.

2.6 Complete Coverage Problem

Chapter 3

Methodology

The different components of the proposed system and how they work together are described in this chapter.

3.1 System Overview

The system will use multiple drones with PixHawk flight controllers, each paired with an individual companion computer and a ground control station (GCS). The operator will set the origin of the global map and select a region of interest on the GCS using Mapviz, a ROS package. The system will then create a grid over the region of interest and calculate paths for the drones. After the expected paths are calculated, the system will calibrate the drones by finding an accurate transformation between each drone's local frame of reference and the global frame. To accomplish this, the drones will move in a predefined flight path while the transformations between each local map of and the global map is refined. After calibration is complete, the system coordinates the flights of the drones as they move along the path the system calculated ensuring coverage of the region of interest. While flying their paths, the drones will stream images from their cameras to the GCS. On the GCS, the SfM module will receive the images and construct an aerial map (a textured 3D mesh) with respect to the global map frame for the region of interest initially selected by the operator.

All the components will be run on a single computer while simulating the system. In simulation mode, PX4 will be run in software in the loop (SITL) mode. Gazebo will be used to visualize the physical world and render simulated video feeds. Network Simulator 3 (NS3) will be used to simulate the networking functions of the wireless mesh network.

3.2 System Design

The different compomenets comprising the system are described here.

3.2.1 Components

The system can be divided into two domains, one comprising the components in the drones, and the other comprising the components in the GCS. Components running on drones will not share information with other drones; they will only share with the GCS. These components will be involved with vehicle platform manipulation and forwarding the video stream. The components on the GCS will be involved with aerial mapping from the video stream received, getting the initial region of

interest from the operator, planning the path of each drone, and coordinating the movement of each drone during plan execution.

The flight controller will be installed with PX4 firmware execution. The companion computers and the GCS will run Ubuntu or Raspbian Linux with their Robot Operating System (ROS) installed and will be connected to a wireless mesh network through the IEEE 802.11 devices.

The components on the GCS are:

- Robot Operating System: All the components will be designed and implemented with ROS as the base infrastructure. This will give access to the plethora of ROS tools and packages.
- Structure from Motion: The aerial map from the video feed of the drones will be constructed by this component.
- mapviz: Mapviz is a ROS-based visualization tool with a plug-in system focused on visualization of 2D data. It shows tiled maps using the Microsoft Bing Maps API and has plugins to select polygons on the map, show markers, show NavSat data, among other features that fit the requirements of this system. Mapviz also allows us to fix the origin of the global map.
- pegasus_planner: A new multi-agent path planning ROS node designed for the project.
- pegasus_controller: A new ROS node that will calculate and maintain translations between different map frames, monitor links with the drones, transmit path planning information to the drones, and coordinate the drones during flight.

The components in the drones are:

- Flight Controller: The drones will use Pixhawk flight controllers with PX4 as their firmware.
- Companion Computer: The companion computer will control the flight controller in OFFBOARD mode.
- Robot Operating System: The software components will be designed and implemented as ROS nodes.
- mavros: The companion computer will use mavros, a ROS package for MavLink communication between companion computers, and flight controllers, to control the flight controller in OFFBOARD.
- pegasus_commander: A new ROS node designed for the project that receives paths and control messages from the GCS. It will also send local poses, GPS positions, flight controller status updates, and GCS link monitoring information.
- video streamer: The video feed from the onboard camera will be captured and streamed to the GCS by this component.

Figure 3.2.1 illustrates the different components of the system controlling three drones.

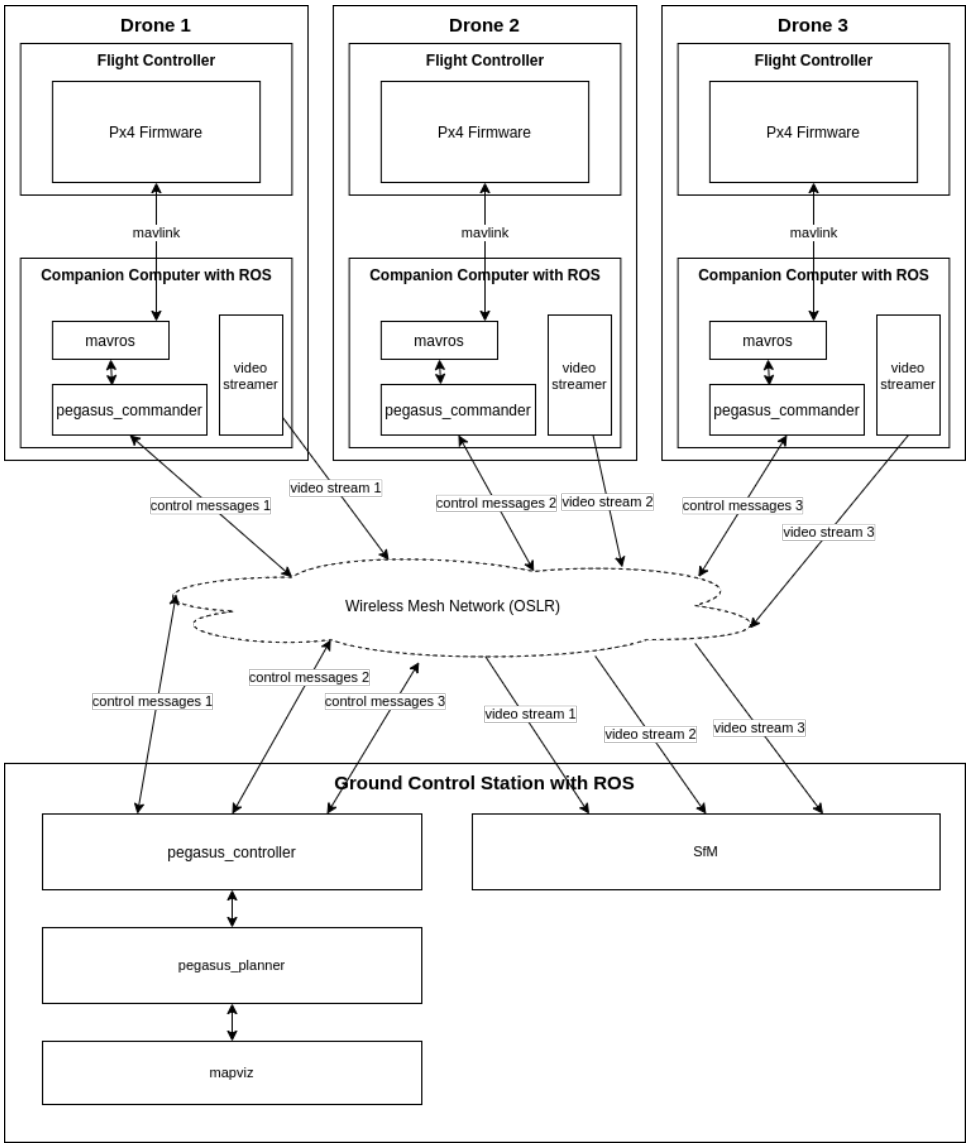

Figure 3.1: Pegasus system overview with three drones.

The operator will select the global map origin and region of interest through mapviz. The pegasus_planner will create a grid-based representation of the region of interest and plan the paths for the drones in the global map reference frame. The initial position of each drone will be provided by pegasus_controller to pegasus_planner. The path planned by pegasus_planner will be sent to pegasus_controller. Pegasus_controller will run a calibration routine on each of the drones to find the transformation between the local origin of the drone and the global map origin. After calibration completes, pegasus_controller will transform the paths for the drones from global map to the local map for each drone and send it to the drone. Pegasus_controller will monitor the link with each drone by sending a periodic heart-beat packet. If a drone does not receive a heart-beat packet for some interval, it will change to Return To Home (RTL) mode and abort its current path. To avoid collision, pegasus_controller will also maintain a path index counter. Paths are made up of a list of poses. Pegasus_controller will tell each drone to move to the next pose in its path when all the drones have reached their current goal pose.

The SfM module will receive image streams from the drones, and construct an aerial map with respect to the global map frame for the region of interest selected by the operator.

The drones will use mavros, which exposes MavLink protocol parameters and services as ROS topics and services. Pegasus_commander will receive commands and path information from the GCS and execute its plan while sending local poses, global GPS positions, and flight controller state information back to the GCS. The video streamer will capture images from the onboard camera, apply GPS tags and send them to the GCS.

3.2.2 Ground Control Station

The GCS will be a computer running Ubuntu, with ROS installed. In my particular case, the GCS has an i7 processor, 16 GB of RAM, and a wireless LAN interface.

3.2.2.1 Mapviz

Mapviz requires the operator to select the global map origin as a ROS parameter. ROS parameters can be set through the rosparams command-line tool, programmatically or in the launch file. In our case, we set the local map origin though the launch file using the ROS package swri_transform_util.

```
<?xml version="1.0"?>
<launch>
  <node pkg="mapviz" type="mapviz" name="mapviz"></node>
  <node pkg="swri_transform_util" type="initialize_origin.py"
    name="initialize_origin" >
    <param name="local_xy_frame" value="/map"/>
    <param name="local_xy_origin" value="control_station"/>
    <rosparam param="local_xy_origins">
      [{
```

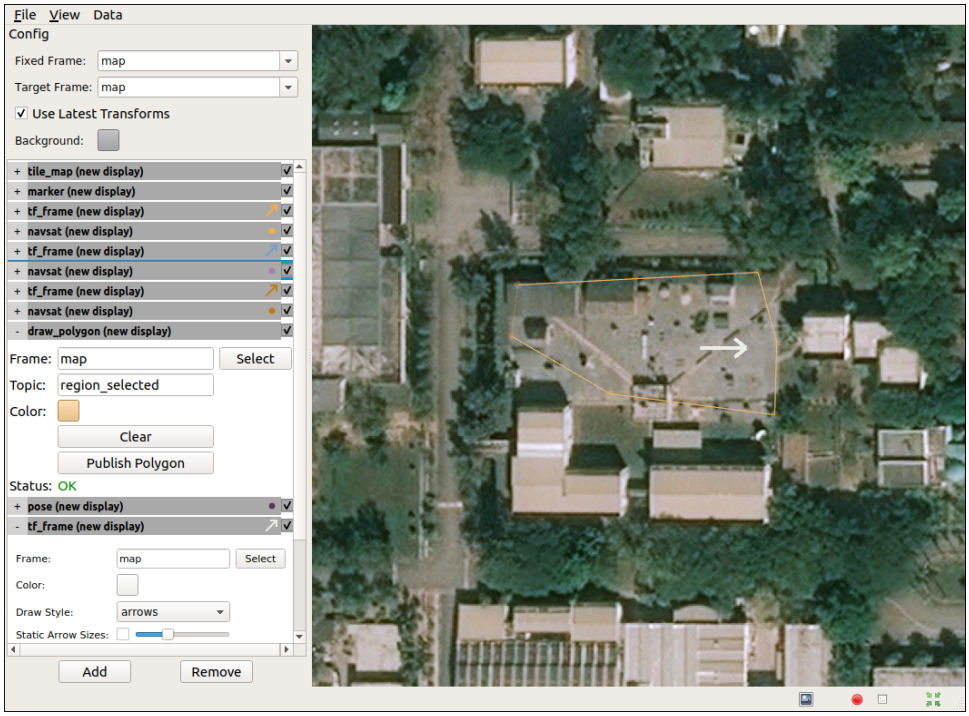

Figure 3.2: Mapviz visualizer with origin set near CSIM.

```
        name: control_station,
        latitude: 14.081104,
        longitude: 100.612743,
        altitude: 7,
        heading: 0.0
      }]
    </rosparam>
  </node>
</launch>
```

It will publish two ROS topics:

- /local_xy_origin: Origin of the global map.
- /mapviz/region_selected: The points of the polygon as selected by the user.

3.2.2.2 Pegasus_planner

Pegasus_planner will handle the following tasks:

- Generate grid cells in the region of interest.

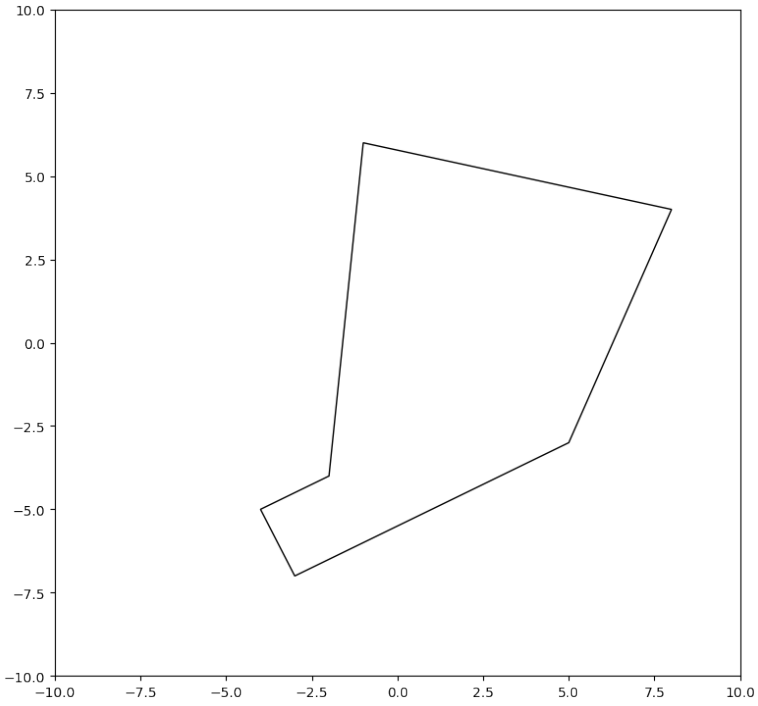

Figure 3.3: Polygon showing the area of interest.

- Try to calculate the optimal paths for each drone, such that they avoid collisions and stay within the constraints of the wireless mesh network in the global map frame.

To generate a valid grid inside the region of interest, consider a polygon with n vertices:

$$\text{polygon} = \begin{bmatrix} x^0 & y^0 \\ x^1 & y^1 \\ x^3 & y^2 \\ \vdots & \vdots \\ x^{n-1} & y^{n-1} \end{bmatrix}.$$

The vertices are arranged counter-clockwise/clockwise. To find the bounding box of the polygon, we need to find the vertices.

$$\text{boundingBox} = \begin{bmatrix} x_{min} & y_{min} \\ x_{max} & y_{min} \\ x_{max} & y_{max} \\ x_{min} & y_{max} \end{bmatrix},$$

where each row defines a vertex. Consider a grid with n cells either horizontally or vertically. Let c be the grid cell size.

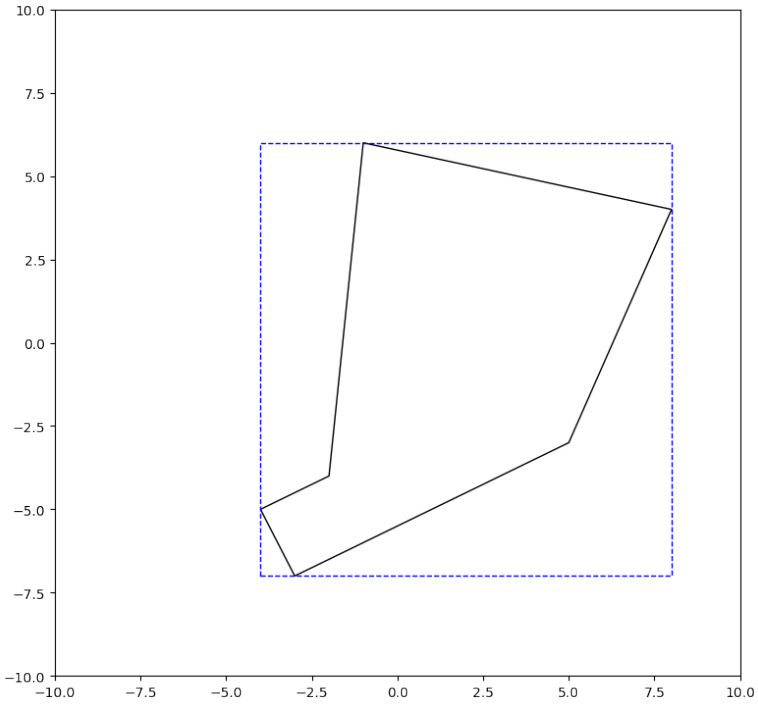

Figure 3.4: Bounding box on the Polygon showing the area of interest.

$$c = \text{BoundingBox}_{width|heigh} \div n$$

Alternatively, we can set a fixed size for each cell, or calculate the cell size from camera parameters and the altitude of the drone.

We can then calculate the coordinates $(b_x^{i,j}, b_y^{i,j})$ of the bottom left vertex of each cell.

$$\text{cells} = \begin{bmatrix} b_x^{0,0} & b_y^{0,0} \\ b_x^{1,0} & b_y^{1,0} \\ b_x^{2,0} & b_y^{2,0} \\ \vdots & \vdots \\ b_x^{i_{max}-1,0} & b_y^{i_{max}-1,0} \\ b_x^{0,1} & b_y^{0,1} \\ b_x^{1,1} & b_y^{1,1} \\ b_x^{2,1} & b_y^{2,1} \\ \vdots & \vdots \\ b_x^{i_{max}-1,0} & b_y^{i_{max}-1,0} \\ \vdots & \vdots \\ b_x^{i_{max}-1,j_{max}-1} & b_y^{i_{max}-1,j_{max}-1} \end{bmatrix}$$

$$b_x^{i,j} = x_{min} + i * c$$
$$b_y^{i,j} = y_{min} + j * c$$

i, j are the index in x and y axis respectively.

We can get the range of i and j as follows.

$$i_{max} = \left\lceil \frac{x_{max} - x_{min}}{c} \right\rceil$$
$$j_{max} = \left\lceil \frac{y_{max} - y_{min}}{c} \right\rceil$$

Let the number of vertices in $gridCells$ be k:

$$k = i_{max} \times j_{max}.$$

We can then calculate grid index (i, j) from $[0, k]$:

$$j = \left\lfloor \frac{k}{i_{max}} \right\rfloor$$
$$i = k - j \times i_{max}.$$

Since the vertex v of the polygon is arranged counter-clockwise/clockwise, we can easily get the vertices of line segment for each side of the polygon.

$$v_i = (x_i, y_i)$$
$$s < n$$

$$\text{polygonLineSegments} = \begin{bmatrix} v_0 & v_1 \\ v_1 & v_2 \\ \vdots & \vdots \\ v_s & v_{s+1} \\ \vdots & \vdots \\ v_{s_{max}-1} & s_0 \end{bmatrix}$$

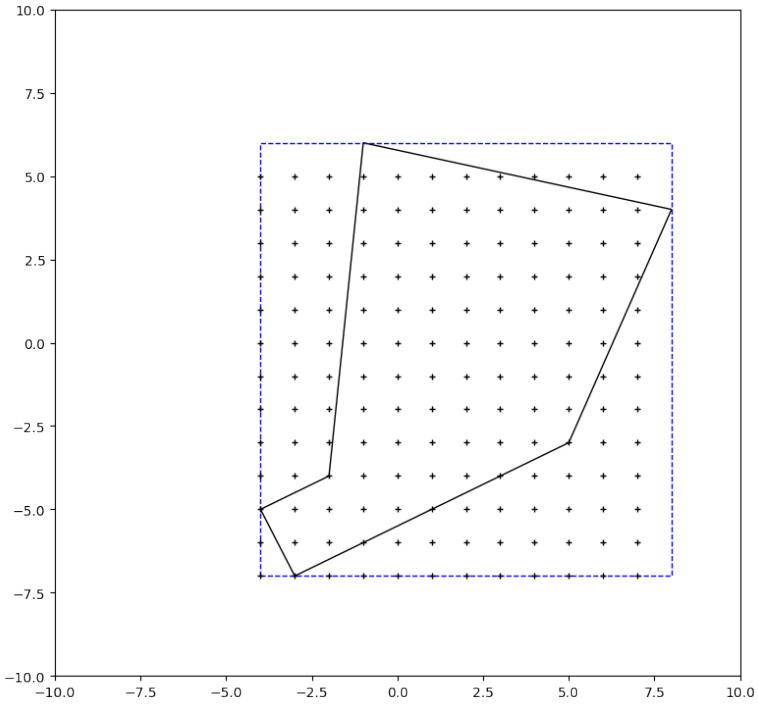

Figure 3.5: Bottom left position of the cells in the bounding box.

$$= \begin{bmatrix} x_0 & y_0 & x_1 & y_1 \\ x_1 & y_1 & x_2 & y_2 \\ \vdots & & \vdots & \\ x_s & y_s & x_{s+1} & y_{s+1} \\ \vdots & & \vdots & \\ x_{n-1} & y_{n-1} & x_0 & y_0 \end{bmatrix}$$

We can imagine that the center of each grid cell projects a line parallel to the x axis till $x_{max} + 1$.

$$\text{cellCenter} = [\text{cells}] + \frac{c}{2}$$

$$\text{rays} = \begin{bmatrix} \text{cellCenter}_x^{0,0} & \text{cellCenter}_y^{0,1} & x_{max} + 1 & \text{cellCenter}_y^{0,1} \\ \text{cellCenter}_x^{1,0} & \text{cellCenter}_y^{1,1} & x_{max} + 1 & \text{cellCenter}_y^{1,1} \\ \text{cellCenter}_x^{2,0} & \text{cellCenter}_y^{2,1} & x_{max} + 1 & \text{cellCenter}_y^{2,1} \\ \vdots & & \vdots & \\ \text{cellCenter}_x^{k-1,0} & \text{cellCenter}_y^{k-1,1} & x_{max} + 1 & \text{cellCenter}_y^{k-1,1} \end{bmatrix}$$

Now we apply ray tracing to check if $cell^{i,j}$ is inside our polygon or not. We count how many intersections each $rays^{i,j}$ makes with all the lines in $polygonLineSegments$. If the number of intersections are even, then the $cell$ is outside the polygon. If the number of intersections are odd,

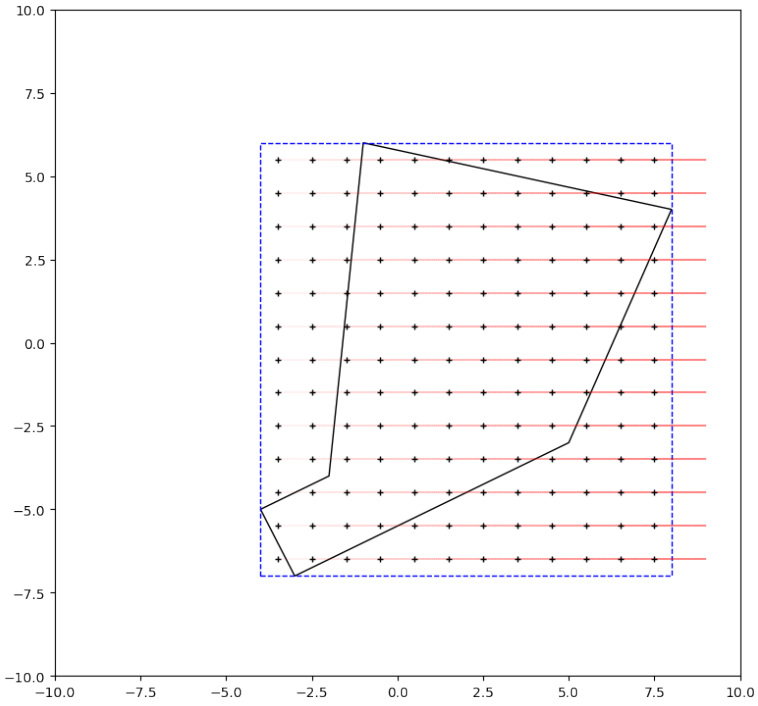

Figure 3.6: Rays projecting from cell centre to $x_{max} + 1$.

then the $cell$ is inside the polygon. We now have the valid cells that can the drones can traverse.

To plan the path for the drones.....

3.2.2.3 Pegasus_controller

Pegasus_controller will be responsible for:

- Calculating and maintaining the transformations between the local map frames of each drone and the global map frame.
- Coordinating and sending control messages to the drones. Refer to section 3.2.3.3 for the list of control messages available.
- Monitoring the link with the drones by sending heart beat control message.
- Publishing local pose, global GPS coordinates and state of each drone, received from pegasus_commander as ROS topics.
- Publishing the transposed pose and transposed GPS position of all the drones in global map frame.

It will have the following states:

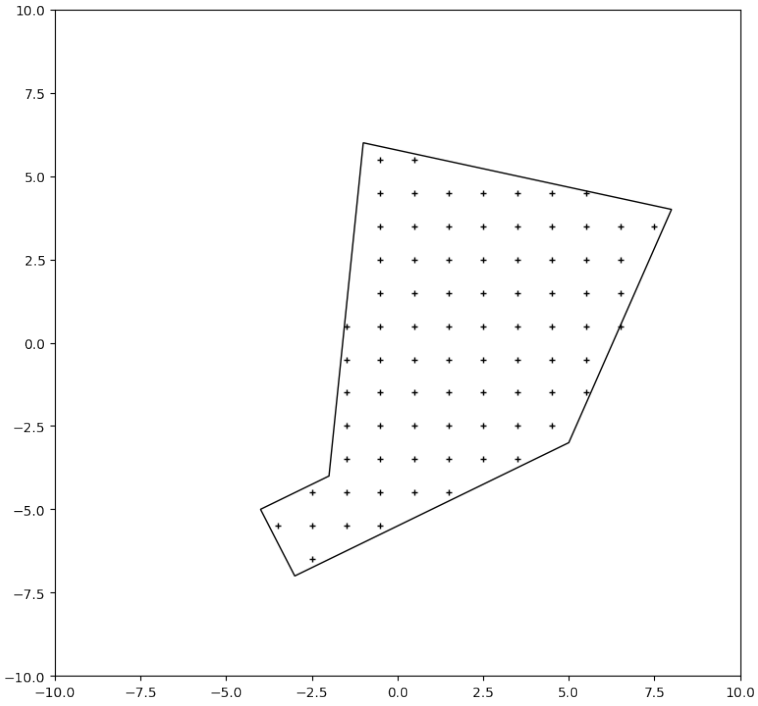

Figure 3.7: Grid cells inside the region of interest.

- IDLE: It will not do anything.
- OFFBOARD_MODE: It will send SET_OFFBOARD control message to the drones and wait for confirmation.
- ARMING: It will send SET_ARM control message to the drones and wait for confirmation.
- TAKE_OFF: It will send TAKEOFF control message with the altitude to the drones.
- CALIBRATE: It will send the calibration path to the drones and coordinate the flight of the drone along the calibration path. It will also capture the corresponding local poses and global GPS positions for calculation the map transformations.
- GENERATE_TRANSFORMS: It will generate the transforms between the local map frame of each drone and the global map frame using the corresponding points collected in CALIBRATE state.
- RUN: It will transform the path published by pegasus_planner on ROS topic /pegasus/[agent]/path to each drone local frame, send it to the drones using SET_PATH control message, and coordinate the drones using GOTO_PATH_INDEX control message.
- COMPLETE: It will reach this state when the system's execution is complete. Set the drone to Return mode by sending SET_RTH control message.

Algorithm 1 will be used to find transforms between local map of each drone and the global map.

Algorithm 1 Find Local To Global Transformation

Input: L: set of all corresponding local poses

Input: G: set of all corresponding global GPS positions

Input: O: global map origin GPS coordinate

Output: T: transformation between local and global map

$\widetilde{O} \leftarrow$ CONVERT-TO-UTM(O)

$\widetilde{G} \leftarrow$ CONVERT-TO-UTM(G)

$M \leftarrow \widetilde{O} - \widetilde{G}$

$\widehat{M} \leftarrow$ REMOVE-Z-AXIS$(\widetilde{M})$

$\widehat{L} \leftarrow$ REMOVE-Z-AXIS(L)

$H \leftarrow$ FIND-HOMOGRAPHY$(\widehat{L}, \widehat{M}, \text{RANSAC})$

$$T \leftarrow \begin{bmatrix} H_{1,1} & H_{2,1} & 0 & H_{3,1} \\ H_{1,2} & H_{2,2} & 0 & H_{3,2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2.3 Drone

3.2.3.1 PX4

The drone runs PX4 Firmware on the flight controller. The flight controller's movement and altitude can be controlled in Offboard mode through the MAVlink protocol. This mode supports only a very limited set of MAVLink commands.

MAVLink commands supported in Offboard modes are:

- SET_POSITION_TARGET_LOCAL_NED: Control vehicle position.
- SET_ATTITUDE_TARGET: Control vehicle altitude and orientation.

PX4 supports the coordinate frames (coordinate_frame field): MAV_FRAME_LOCAL_NED and MAV_FRAME_BODY_NED.

A stream of setpoint commands must be received by the vehicle prior to engaging the mode, and in order to remain in the mode (if the message rate falls below 2Hz the vehicle will stop). In order to hold position while in this mode, the vehicle must receive a stream of setpoints for the current position.

Offboard mode requires an active connection to a remote MAVLink system (e.g. companion computer or GCS). If the connection is lost, after a timeout (COM_OF_LOSS_T) the vehicle will attempt to land or perform some other failsafe action. The action is defined in the parameters COM_OBL_ACT and COM_OBL_RC_ACT which can be set through mavros' mavparam.

3.2.3.2 Mavros

Mavros is a ROS node that provides ROS interfaces for communication with various autopilots with MAVLink communication protocol. The ROS services provided by Mavros, of interest for this system are:

- mavros/set_mode: Required to set mode of the flight controller to Offboard mode.
- mavros/cmd/arming: Required to arm the flight controller.

The ROS topics published by Mavros, useful for this system are:

- mavros/state: Publishes the state of the flight controller.
- mavros/local_position/pose: Publishes pose, i.e. position and orientation of the flight controller in the local frame. Mavros handles the translation between NED and ENU conventions.

- mavros/global_position/global: Publishes the GPS coordinates, i.e. latitude, longitude and altitude of the flight controller.

The ROS topics subscribed by Mavros, that this system will use are:

- mavros/setpoint_position/local: Set the pose, i.e. position and orientation that the flight controller is desired to achieve in local frame.

3.2.3.3 Pegasus_commander

Pegasus_commander is a new ROS node, customized for this study that uses the topics and services provided by mavros to control the drone in Offboard mode. It receives control messages from the GCS. It also monitors the link between GCS and the drone, and a heart beat control message is not received for a particular duration, then the flight controller is set to Return mode.

The different types of control messages that will be received by pegasus_commander from GCS are:

- HEART_BEAT: GCS heart beat in regular intervals.
- SET_OFFBOARD: Put the flight controller in Offboard mode.
- SET_RTH: Put the flight controller in Return mode.
- SET_ARM: Arm the flight controller.
- TAKEOFF(float): Set the altitude of the drone.
- SET_PATH(pose): Set the path that the drone needs to follow in local map frame.
- GOTO_PATH_INDEX(int): Set the local position to the pose in the given index.

The different types of control messages that will be sent to the GCS are:

- HEART_BEAT_ACK: Acknowledgment to the GCS's heart beat.
- PATH_INDEX_REACHED(int, pose): Return the path index and the local pose when the drone reaches the set-point indicated with GOTO_PATH_INDEX message from GPS.
- SUCCESS(message): Return success confirmation for message from GCS.
- FAILED(message): Return fail confirmation for message from GCS.

Pegasus_commander also transmits the state, local pose and GPS coordinated of the drone to the GCS in regular intervals.

3.2.3.4 Video Streamer

The images from the onboard camera of the drone will be captured. Then each image frame will be geo-tagged with the current GPS coordinates from the flight controller and the geo-tagged image feed will be streamed to the GCS.

3.2.4 Simulation

Most of the components will remain the same regardless of whether runnng in simulation or in the real world. The simulation will run in a single computer. Three more components will be utilized to simulate the system:

- Gazebo: Gazebo simulator will be used because it supports fleet of drones and camera feed.
- Network Simulator 3: NS3 will be used to simulate the wireless mesh network functions.
- pegasus-net-sim: A custom NS3 module that will simulate the mobility of the drones and push and pop actual control messages and video feed between the drones and the GCS through the simulator. It will also publish the status of each device in the network.
- pegasus_networl_status_util_node: A custom ROS node that will receive noise and signal strength from pegasus-net-sim and publish the signal-to-noise ratio (SNR) value of each drone as ROS topic.

Figure 3.2.4 describes the simulation system schematically.

Gazebo will be used to simulate the world, the drone mechanics and the video feed. Pegasus-net-sim, a Network Simulator 3 custom module will be used to simulate the wireless mesh network. Pegasus-net-sim will receive the model info from Gazebo and update the position of its nodes. Each node in pegasus-net-sim will have NS3PegasusApp, a custom NS3 application installed, which will send and receive packets in the simulated network. Pegasus-net-sim will function as a proxy application that will receive data in one UDP port, simulate it in NS3 and transmit it through another UDP socket. Using the trace feature in NS3, pegasus-net-sim will publish the noise and signal value for each node of the wireless mesh network.

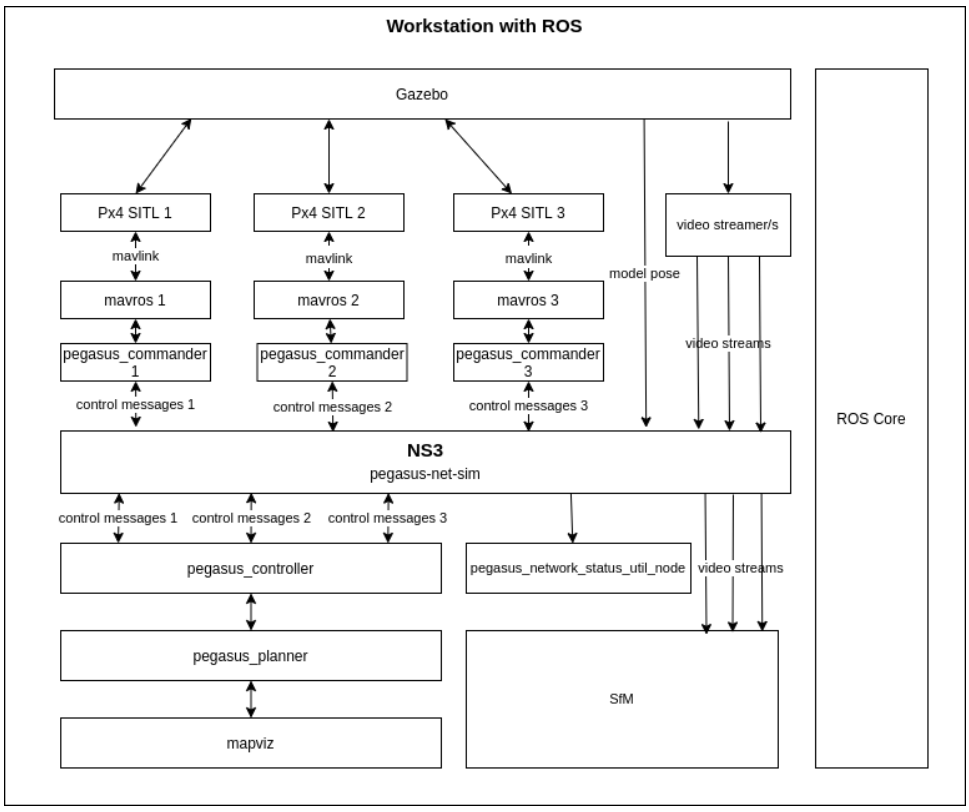

Figure 3.8: Pegasus Simulation System Overview

Chapter 4

Experimental Results

Some intro..

Chapter 5

Conclusion and Recommendations

Some text..

References

Acharya RBCCPS, S., Bharadwaj, A., Simmhan, Y., Gopalan, A., Parag, P., & Tyagi, H. (2020). *CORNET: A Co-Simulation Middleware for Robot Networks* (Tech. Rep.).

Almadhoun, R., Taha, T., Seneviratne, L., & Zweiri, Y. (2019, 08). A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, *1*.

Alshabtat, A. I., & Dong, L. (2011). Low latency routing algorithm for unmanned aerial vehicles ad-hoc networks. *World Academy of Science, Engineering and Technology*, *80*, 705–711.

Baidya, S., Shaikh, Z., & Levorato, M. (2018). FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot.

Behere, S., & Trngren, M. (2016). A functional reference architecture for autonomous driving. *Information and Software Technology*, *73*, 136 - 150.

Chand, G., Lee, M., & Shin, S. (2018, 01). Drone based wireless mesh network for disaster/military environment. *Journal of Computer and Communications*, *06*, 44-52.

Chriki, A., Touati, H., Snoussi, H., & Kamoun, F. (2019, nov). FANET: Communication, mobility models and security issues. *Computer Networks*, *163*, 106877.

Ivaldi, S., Padois, V., & Nori, F. (2014, feb). Tools for dynamics simulation of robots: a survey based on user feedback.

Kulla, E., Hiyama, M., Ikeda, M., & Barolli, L. (2012, jan). Performance comparison of OLSR and BATMAN routing protocols by a MANET testbed in stairs environment. In *Computers and mathematics with applications* (Vol. 63, pp. 339–349). Pergamon.

Mbarushimana, C., & Shahrabi, A. (2007). Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks. In *Proceedings - 21st international conference on advanced information networking and applications workshops/symposia, ainaw'07* (Vol. 1, pp. 679–684).

Meeussen, W. (2010). *REP 105 – Coordinate Frames for Mobile Platforms (ROS.org).*

Patel, R., Pathak, M., & Nayak, A. (2018). Survey on Network Simulators. *International Journal of Computer Applications*, *182*(21), 23–30.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., et al. (2009). ROS: An open-source robot operating system. In *Workshops at the IEEE international conference on robotics and automation.*

Rosati, S., Kruzelecki, K., Traynard, L., & Rimoldi, B. (2013). Speed-aware routing for UAV ad-hoc networks. In *2013 ieee globecom workshops, gc wkshps 2013* (pp. 1367–1373). IEEE Computer Society.

Sabino, S., Horta, N., & Grilo, A. (2018, Dec). Centralized unmanned aerial vehicle mesh network placement scheme: A multi-objective evolutionary algorithm approach. *Sensors*, *18*(12), 4387.

Singh, K., & Verma, A. K. (2015, jan). Applying OLSR routing in FANETs. In *Proceedings of 2014 ieee international conference on advanced communication, control and computing technologies, icaccct 2014* (pp. 1212–1215). Institute of Electrical and Electronics Engineers Inc.

Siraj, M. S., Ajay Kumar Gupta, M., & Rinku-Badgujar, M. (2012). *Network Simulation Tools Survey* (Vol. 1; Tech. Rep.).

Staranowicz, A., & Mariottini, G. L. (2011). A survey and comparison of commercial and open-source robotic simulator software. In *Acm international conference proceeding series.* Association for Computing Machinery.

Tareque, M. H., Hossain, M. S., & Atiquzzaman, M. (2015). On the routing in flying ad hoc networks. *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015*(October), 1–9.

Toor, A. S., & Jain, A. K. (2017). A survey on wireless network simulators. *Bulletin of Electrical Engineering and Informatics*, *6*(1), 62–69.

Wikipedia. (2019). *Geographic coordinate system.* `https://en.wikipedia.org/wiki/Geographic_coordinate_system`.

Zheng, Y., Wang, Y., Li, Z., Dong, L., Jiang, Y., & Zhang, H. (2014). A mobility and load aware OLSR routing protocol for UAV mobile AD-HOC networks. *2014 International Conference on Information and Communications Technologies, ICT 2014*.

Appendix A

.. TITLE HERE ..

Section Name

Figure A.1 shows something.

Some text ..

Figure A.1: CCTV monitoring room. Reprinted from the Twenty First Security Web site (`http://www.twentyfirstsecurity.com.au/`).