University of Southern Queensland
Faculty of Health, Engineering & Sciences

Eye of the Swarm: Real-Time Analysis of Factors Affecting Dynamic UAV Pursuit of a Moving Target

A dissertation submitted by

C. Arnold

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Electrical & Electronic Engineering

Submitted: October, 2020

Abstract

The increased use of unmanned aerial vehicles (UAVs) both of a commercial or consumer level has presented a problem of protecting nation- or company-critical sites from intelligence gathering, surveillance, or reconnaissance activities. Recent attacks and intrusions of restricted airspace by UAVs raise questions about how to tackle the problem of tracking autonomous malicious UAVs of increasing abilities. The use of consumer or commercial grade UAVs may provide an answer to the problem through implementing swarm formations and tactics to pursue a malicious UAV to its landing point, and thereby its operator. Swarms of UAVs can provide redundancy and group agility greater than an individual drone, as well as a larger tracking radius than fixed ground-based radars or expensive military-grade UAVs. The use of UAV swarms consisting of differing sizes and formations were examined to determine their effectiveness in pursuing a malicious UAV breaching restricted airspace. Based within a simulated environment, the modelling involved an analysis of the distance between the swarm and the malicious UAVs landing site at the end of the simulation. The effects of increasing the swarm size, the formation that the collective swarm takes, and the fight characteristics (speed, acceleration, flight path, etc.) of the malicious UAV were varied to test the relative strengths and weaknesses of the a swarm compared to the same number of UAV pursuers working independently.

The results show the use of collaborative formations decrease the final distance from the target, especially in swarms containing five or more UAVs. The cone formation proved to be the overall better choice of the two collaborative formations developed and tested. This formation provided the greatest resilience in adapting to increases in malicious UAV flight abilities, though in several cases the less processor-intensive surround method performed sufficiently better than the baseline to be considered useful in certain applications. The results from this project were utilised in a submitted, and accepted, peer-reviewed paper presented at the IEEE UEMCON 2020 conference.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

C. Arnold

[redacted]

Acknowledgments

My partner, Jules, had the unenviable task of being the sounding board for untold hours of technical issues and half-baked ideas throughout this project, knowing she could not help.

My supervisor, Dr Jason Brown, managed to keep me from getting stuck in the technical weeds of this project every week for over a year.

Finishing this project would not be possible without them. Thank you both.

C. Arnold

Contents

List of Figures

List of Tables

Acronyms & Nomenclature

UAV	Unmanned Aerial Vehicle
UCAV	Unmanned Combat Aerial Vehicle
SUAV	Small Unmanned Aerial Vehicle
OMNeT	Objective Modular Network Testbed
INET	Product name (not acronym) for OMNeT add-on framework
GPS	Global Positioning Satellite
Swarm	Collection of two or more UAVs
MPC	Model Predictive Control
PN	Proportional Navigation
MANET	Mobile Ad-hoc Network
VANET	Vehicle Ad-hoc Network
WMN	Wireless Mesh Network
NED	Network Descriptor file extension
INI	Initialisation file extension
GUI	Graphical User Interface
Wi-Fi	Wireless Fidelity - Wireless network connection
C++	Object-Oriented C computer language
.cc	C++ source file
.h	C++ header file
Kernel	Simulation operating system
PNID	Position Node ID

Chapter 1

Introduction

1.1 Background

From missile-capable unmanned combat aerial vehicles (UCAVs) to object-tracking consumer "small UAVs" (SUAVs), airborne UAVs have become cheaper and more widely used in the last decade (Yanmaz, Yahyanejad, Rinner, Hellwagner & Bettstetter 2018). Over 30,000 UAVs are expected to be used in the consumer and commercial area within the USA alone with global spending on UAVs expected to double over the decade to 2023 (West & Bowman 2016, Boussios 2014). This is attributed mainly to advances in the material and electronic areas leading to reduced weight with increased functionality, run time, and lower cost (West & Bowman 2016, Boussios 2014, Akram, Markantonakis, Mayes, Habachi, Sauveron, Steyven & Chaumette 2017, George & Ghose 2009). With these advances come a widening of mission profiles that are possible without direct human intervention.

The recent UAV attack on a Saudi oilfield and the resulting 15% surge in oil price raised questions regarding the vulnerability of non-military targets to improvised UAV-based threats (NPR 2019). Just the deployment of unknown UAVs near military positions has prompted concerns over possible incursions (Rossiter 2018). Though typically only thought of in terms of offensive actions, unmanned vehicles originally filled a surveillance and reconnaissance role (Cevik, Kocaman, Akgul & Akca 2013).

Today, the use of UAVs for these purposes against national infrastructure or for corpo-

rate espionage can have a large negative effect both economically and strategically. The small physical size typically renders SUAVs less susceptible to site defence such as surface-to-air missiles, electronic countermeasures, or occasionally even visual detection (Cevik et al. 2013, BBC 2017). Several recent examples show the vulnerabilities nation-critical infrastructure faces. In December of 2018 two UAVs caused chaos at the UK's second-busiest international airport (Gatwick International) by flying within restricted airspace several times over multiple days. The airport was locked down for a total of 30 hours, resulting in an estimated £1.4 million in lost revenue and an additional £4 million required for anti-UAV technology (BBC 2019, Guardian 2019*b*). These were direct costs to the airport, with customer compensation costs of just one airline operating from Gatwick totalling £15 million (Guardian 2019*a*). Other incidents involving consumer UAVs landing on aircraft carriers and flying within restricted airspace of nuclear power plants are becoming increasingly common (BBC 2017, Forbes 2019)

SUAVs – typically just referred to as UAVs – are classified as weighing between 1 – 13.5kg and operate within "close range" (Kakar 2015). There has been increasing research and development regarding SUAVs' abilities to process information and act autonomously (Boussios 2014, Yanmaz et al. 2018, George & Ghose 2009). The inclusion of GPS and optical (landmark) way finding presents difficulties in determining the origin of malicious UAVs by traditional means. This stems from an inability to track the radio control signal. The only way to find the origin, therefore, is to physically follow the UAV to its landing point.

Adding to the complexity of the problem is the large number of UAVs available to the consumer market. Each brand and model can offer vastly different flight characteristics. Defending corporate or commercial sites against every type of threat would require a system that could handle the worst-case scenario. This would require expensive (possibly military-grade) tracking capabilities which would be neither technically nor financially feasible for all but the most critical sites.

1.2 Problem Specification & Idea development

Conventional thought would lead to the use of a single vehicle to track a malicious UAV to its final destination. The thesis of this project stipulates the use of a fleet of UAVs

rather than relying on one single platform. The use of multiple (lower specification and cost) adaptive UAVs working in concert could reliably track a malicious UAV, regardless of agility or geographical area covered (Yanmaz et al. 2018). UAVs of this nature – that is to say clusters of UAVs employed for similar purposes and having similar flight patterns - are typically defined as a UAV swarm (Cevik et al. 2013).

A key advantage in utilising a swarm of UAVs for pursuit is the redundancy case if any one UAV develops a fault. A fault may lead to the cancellation of the mission if just one UAV were utilised. Therefore, having multiple UAVs increases the mission robustness and likelihood of mission success (Cevik et al 2013; Akram et al 2017). Swarms of UAVs can also provide group agility greater than an individual drone and a larger tracking radius than fixed ground-based radars or expensive military-grade UAVs. The trade-off, however, is that the controlling algorithm needs to be robust enough to automatically allow for fault events (Dedousis & Kalogeraki 2018, Yanmaz et al. 2018).

1.3 Aim and objectives

The purpose of this research project is to evaluate the effectiveness of different formation algorithms controlling a UAV swarm pursuing a malicious UAV of unknown flight characteristics. As shown in the literature review, there are several broad gaps associated with the topic of defensive UAV swarms. In order to keep within the time constraints of a final-year dissertation the project must be constrained to a specific area of interest. As such, this project aims to evaluate which coordination algorithm is able to pursue a malicious target to its final landing position for increasing levels of malicious UAV agility.

To address this question, the following objectives are set out:

- Successfully simulate a swarm of UAVs working in concert to pursue a target
- Develop specific metrics that allow formation algorithms to be compared
- Measure different formation algorithms' ability to utilise a UAV swarm to outmanoeuvre a malicious UAV of unknown agility
- Measure any decrease in effectiveness of different formation algorithms against increasingly evasive malicious UAV flight paths

1.4 Limitations

The general nature of the topic combined with the multitude uses of swarm UAVs and the simulated nature of the project presents a danger of creating a project with unrealistic or unattainable scope. As there is a firm deadline on completion, limitations to the scope of the project will be implemented. Limitations to the initial scope will include:

- Malicious UAV will follow pre-set flight paths, regardless of swarm position
- Environmental obstacles will not be added
- Swarm UAVs will have fixed velocity and lateral acceleration abilities

As the project is aimed at full simulation, it will start with C++ coding to simulate both the malicious and tracking UAVs. The code will be developed for the UAV swarm in individual modules defined as:

- Basic flight & movement
- Individual pursuit algorithms

The malicious UAV will have some modules in common, such as the basic flight and movement module. As it has a different mission profile to the swarm (simulating reconnaissance by moving along a pre-set path) it will also have distinct modules coded that are necessary only for its use. From there, coding of the swarm UAV inter-communication network and tracking algorithm will be carried out, with the communication network latency and range set to unlimited. This will allow analysis of the tracking algorithm without the added complexity of different communication methods, effectively setting a base standard from which to set context of further datasets.

1.5 Articles Published

As a result of the work completed for this project a paper titled “Performance Evaluation for Tracking a Malicious UAV using an Autonomous UAV Swarm” was submitted for presentation at the IEEE Ubiquitous Computing, Electronics, and Mobile Communication

(UEMCON) 2020 conference. This article was peer reviewed and accepted for presentation at the conference.

1.6 Dissertation Overview

Chapter 1 contains a brief introduction with regards to the project context, problem specifications, and the project aim.

Chapter 2 provides a literature review on research that has been conducted into related subjects. This includes target encirclement, object detection and pursuit, UAV path finding, inter-UAV co-ordination, swarm tactical co-ordination, and mobile node communication methods.

Chapter 3 discusses the simulation design for the project. This includes the details of the base program selected for simulation, the input and output information utilised, and the bespoke algorithms and modules designed for this project.

Chapter 4 contains the project methodology including the data types required and resulting analysis style.

Chapter 5 contains the results of the simulations. This includes detailed analysis of both specific and overall results with comments on unexpected outcomes.

Chapter 6 details the project conclusion with an outline of the project outcomes, whilst providing discussions on further work achievable in this field.

Chapter 2

Literature Review

2.1 Knowledge gap

A literature review into utilising a UAV swarm for pursuing objects was conducted for this dissertation. As UAVs have been around for many years and their popularity has increased, research on the general topic of UAVs has become extensive. Despite the growing rate of research into the area of robot swarms, however, information regarding utilising a UAV swarm for pursuit of a target is not readily available. Research into the use of swarms for tracking of targets within environmental occlusion (Jung & Sukhatme 2002), localised targets (Ma'Sum et al 2013), and even multiple moving targets (Lee, Chong & Christensen 2010) have been conducted. These papers, whilst providing several methods for achieving their end goal, are primarily constrained either by a set area of operations (Jung & Sukhatme 2002, Ma'Sum, Jati, Arrofi, Wibowo, Mursanto & Jatmiko 2013), or within the 2D plane (Lee et al. 2010). Concentrating instead on the use of a single UAV for pursuit yields several articles. Kakar (2015) proposes a pursuit algorithm to perform path planning for a fixed wing UAV. The algorithm would be computed by the onboard UAV processor. As such it has limitations based on the processing power and energy supply available to the UAV. Though the algorithm was successfully implemented, it was only useful in tracking ground-based targets. Lee et al. (2010) aimed to utilise a swarm of UAVs in tracking and converging on multiple targets. The implementation also did not require that all members of the swarm have sight of the target in order to converge on it. The experiment saw successful coordination of the UAVs but the target UAVs were of the same agility as those tracking it whilst just moving in a random fashion. As such the

swarm did not require special algorithms for anticipating the tracked UAVs' movements for fear of losing them.

Further analysis of the available literature found that research regarding swarm UAVs near the topic of interest is usually specialised into three areas: encirclement, coordination & control and communication methods.

2.2 Target Encirclement

Methods into restricting the movements of malicious UAVs are required in order to combat the threat posed. Research has been conducted in both simulated and real models that aim to find efficient predictive models to encircle and therefore restrict the movements of objects, in particular UAVs. This research, whilst ongoing, typically concentrates on restriction of stationary objects. Work by Hafez, Marasco, Givigi, Iskandarani, Yousefi & Rabbath (2015) aimed to use model predictive control (MPC) to control a number of UAVs in real-time in order to encircle a target whilst taking into account non-linearities and external disturbances experienced by UAVs. The proposed MPC model was simulated in a horizontal 2D plane and achieved sufficient encirclement of the target, and was even implemented on real quad rotor UAVs within a laboratory environment. The results showed successful encirclement within acceptable margins for two variations of the MPC implementation. The research also clearly showed the issue of increased algorithmic complexity returning diminishing returns as, though the non-linear MPC (NMPC) showed more precise movement of the swarm UAVs, the computational effort required over linear MPC (LMPC) was marked and ultimately only the LMPC algorithm was implementable for real-time control.

Another form of MPC control takes away from the centralised computation and control to leverage the natural decentralised nature of multiple UAVs working together (Marasco, Givigi & Rabbath 2012). Decentralized MPC (DMPC) is shown to be effective in simulations involving several UAV/target types. These include simulations of a single UAV encircling both a static and moving target, and of multiple UAVs encircling a stationary target. Just as LMPC and NMPC, DMPC was shown as effective in encirclement of the given targets, though the ability to encircle a moving target was limited to slow movements. Hafez, Givigi, Schwartz, Yousefi & Iskandarani (2015) conducted further research

on LMPC implementation combined with feedback linearisation (FL) and decentralised control for multiple UAVs. This research resulted in an overall stable team of cooperative UAVs that can avoid collision whilst maintaining encirclement velocity and proximity.

2.3 Coordination & Control

2.3.1 Detection and pursuit

Pursuit of objects presents many obstacles, especially in real-time processing of outdoor environments. The complexity is increased by the need to track targets not bounded by two dimensions such as cars but instead in three. Added to this is that individual UAV not only tracks the target but also other UAVs which are themselves moving within the 3 dimensions. Research has shown many methods to track objects through the use of UAVs, though they are typically used to enhance the tracking ability of stationary sensors within a bounded area (Jung & Sukhatme 2002).

For a mission area of known size and estimated number of targets, a region-based approach can be utilised in object detection. This method distributes the swarm over the area for coarse detection, with individual UAVs within each region executing various methods of search or track functions (Jung & Sukhatme 2002). This method requires previous knowledge of the area to be searched (i.e. geographic landmarks) and does not account for targets that move outside the bounds of the search area. Fu, Feng & Gao (2012) utilises an error-correction path finding algorithm with feedback to direct the UAV towards its target. The targets future location is estimated through a least-squares filter applied to its current motion. Though this algorithm was only used in the tracking of ground-based objects, it was found to be computationally economical and reliable. This presents a better choice for active tracking for individual UAVs as it is not geography based.

2.3.2 Path finding

Though the methods mentioned are generally suitable for advancing on a target, their main objective is to get to the target position directly with no thought for the final orientation of the UAV. Huang, Tung & Ciou (2009) aimed to solve this problem in the

context of a motorised serving robot that is required to navigate an environment with obstacle and arrive at the correct location with the correct orientation. Fuzzy controllers were utilised due to their popularity in the robotic research space – typically for their ease of design with no requirement for coding mathematical models. Two fuzzy controllers were used to guide the robot to its destination - one for navigation and the other for obstacle avoidance. The experiment was carried out in both simulation and on hardware which resulted in adequate performance in target intercept, though correct final orientation was a side-effect of the intercept algorithm and not directly accounted for in the fuzzy controller logic.

This experiment is a good basis on navigation and path finding for environments with obstacles, but would require further work in order to produce results for simulating UAVs. First, UAVs must contend with three dimensions. Though only one extra axis, this represents and increase in complexity not just in navigation but also when calculating avoidance of both stationary and mobile obstacles. Secondly, the swarm UAVs will be operating in a real-time environment against a target that is moving. This will require constant calculation of the most efficient method of interception coupled with heading adjustment. Finally, unlike the robot used in the experiment, the swarm UAVs will need to contend with increased movement abilities regarding slew, pitch and roll attributes on manoeuvrability.

Moving targets have been considered by Belkhouche, Belkhouche & Rastgoufard (2006). Here the final heading is a moving target and obstacles are introduced, though the path is still constrained to the horizontal plane and, again, no direct efforts to match target heading is introduced.

2.3.3 Inter-UAV (swarm) coordination

UAVs working within a swarm, especially SUAVs with fast and fragile moving parts, must be coordinated around each other and the environment they operate in with care. Coordination between UAVs will be necessary in order to outmanoeuvre the malicious drone, but also prevent collisions with each other when working in close proximity.

There are many different types of UAV coordination currently advocated, each with their own spin on their advantages over other types. Tang, Yang & Li (2001) theorised that

avoidance of dynamic objects in a real-time environment benefited from the use of fuzzy logic. Fuzzy logic is based on degrees of truth rather than the typical true and false options of "crisp" logic (Hooda & Raich 2015). In the case of UAV collision avoidance, fuzzy inference would allow decisions to be made even if the input vector was not completely accurate. This would allow for better reaction to real-time events. Fuzzy inference would not allow for uncertain obstacles, however. Rathbun, Kragelund, Pongpunwattana & Capozzi (2002) presents a solution in the form of an Evolutionary Algorithm (EA) for autonomous path planning. This model, whilst being able to deal with fixed objects, aims to factor in the uncertainty of motion of moving obstacles. Objects are treated by their uncertainty rating. A simple object trajectory (e.g. a ball arcing through the air) will having a low uncertainty rating, whilst a military UAV set to evade hostiles would have a high uncertainty rating (Rathbun et al. 2002). This method would be highly effective at not only avoiding collision with UAVs of the swarm and other environmental objects, but also useful in tracking the malicious UAV. The cost, however, to energy consumption from computing the algorithm necessary would be overwhelming, especially if each UAV needed to compute their own avoidance vectors.

A less processor-intensive method of avoidance is presented by Tomlin, Pappas & Sastry (1998). This hybrid system would sense objects around the UAV within two spheres; a small "protected" and larger "alert" sphere. The radius of the spheres would change depending on the speed of the UAV to account for proportional avoidance reaction time. For this model the sensors initially only need to detect the distance of objects and relate them to the sphere radius. Only upon entering within the radius of the alert sphere would avoidance trajectory information be computed and acted upon. This method allows low-consumption processing until higher consumption is required. However, computation priority issues arise in the case of multiple objects entering the alert sphere. Essentially if two objects enter the alert sphere the UAV would not know which should take priority in trajectory processing. The calculation may be made first on an object that never would have collided (e.g. a stationary car in front of the drone), leaving little time to compute and react to the real threat. The alert sphere could be increased to allow more time, but this would allow more false positives. The increased number of threats would need to be computed, raising energy consumption. This reduces its efficiency compared to the previous algorithms.

Though the previously mentioned algorithms are useful in their own ways, the most

promising guidance method is the use of Proportional Navigation-based Collision Avoid Guidance (PNCAG). PNCAG aims to maintain a predefined safe distance between each drone. Though Proportional Navigation (PN) is typically used in missile guidance, the algorithm is modified to provide a collision avoidance vector that the UAV is directed to. A slight variant to PNCAG is a Reactive Inverse PN algorithm. This differs to the PNCAG model in that it is designed for highly-concentrated space use (i.e. a lot of UAVs in a small area). The inverse algorithm posits that avoiding near-misses in such an environment is not realistic so instead near misses are accepted but "minimised" (George & Ghose 2009). Vector information (position, direction, and velocity) for both the host and nearby UAVs is all that is needed for both PNCAG and Reactive Inverse PN algorithms to work (Han & Bang 2004, George & Ghose 2009).

2.3.4 Tactical coordination

Once the UAV swarm is able to avoid each other and obstacles they will require pursuit guidance as a whole to achieve mission success. The tactical control of individual UAVs within swarms can be divided into centralised and decentralised topologies. Centralised topologies (Figure 2.1 and Figure 2.2) allow for a central processing point to do the heavy lifting for computation of pursuit data and swarm formation. However, they require the nodes being controlled to be within the communication radius of the central processing point (Akram et al. 2017) . Decentralised topologies are more robust overall as they do not require a central authority to operate. Instead, the individual UAVs collect data from each other in order to make their own decisions (Akram et al. 2017).

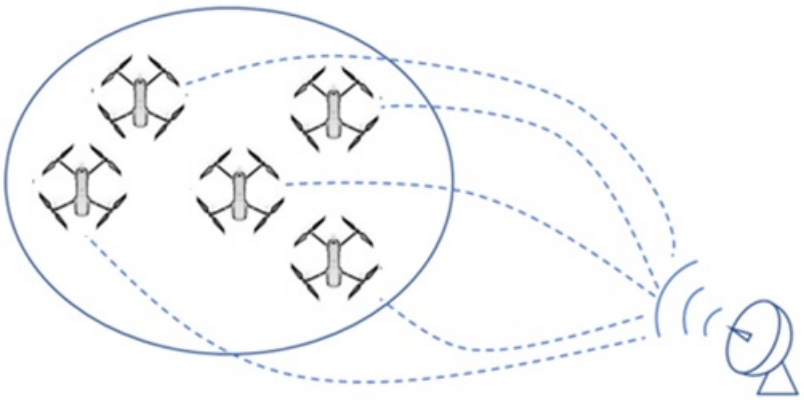


Figure 2.1: Centralised topology.

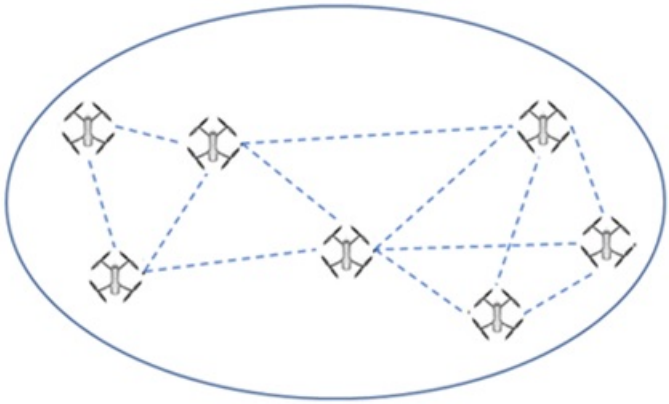


Figure 2.2: Decentralised topology.

2.4 Communication methods

Recent research into communication between UAVs has been likened to ad-hoc mobile and vehicle networks (MANETs and VANETs respectively). These networks, however, do not directly compare to the overly dynamic nature of UAVs operating cooperatively in three dimensions (Cui, Liu, Wang & Yu 2017). Wireless mesh networks (WMN) are the most reliable network configuration for UAV swarms as it presents a robust communication infrastructure (Cui et al 2017). This method also constitutes an ad-hoc network where each node is both a client and router. Two main types of WMN are available; routed and flooded. The routed mesh type (Figure 2.3) requires added power consumption for routing data not required by that specific drone. This increases battery consumption (McCune & Madey 2013). Further network resilience may be achieved through the use of "flooding" type message routing (Figure 2.4). This method, rather than sending through nodes on a preselected route, sends to all nodes within its transmission range. This method removes the need for communication overhead due to maintaining routing paths. The increase in processing power of redundant messages, however, usually outweighs the benefits (Cui et al. 2017).

As shown, there resides large amounts of research targeted towards highly specific areas of both individual and swarm UAV topics. The use of swarm UAVs purely for sustained pursuit rather than direct engagement, however, has yet to be simulated and as such is not well understood. An amalgamation of the above concepts is required to produce data that can be analysed. This will produce evidence related to the benefits of cost-effective swarm UAVs in counter-espionage or defence of nation-critical sites.

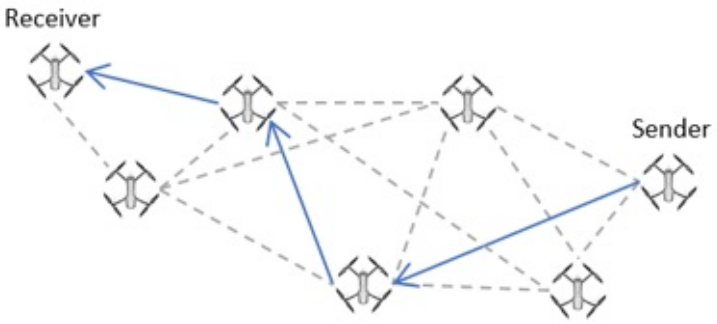


Figure 2.3: Routed mesh network.

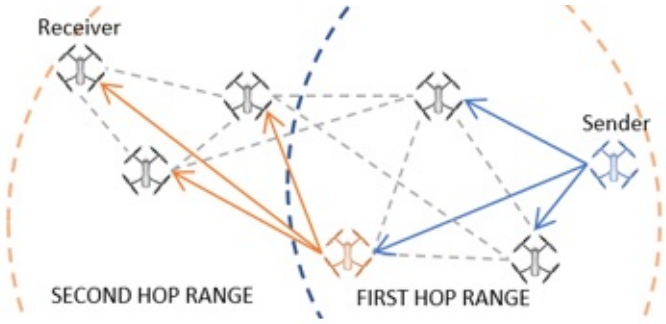


Figure 2.4: Flooded mesh network.

Chapter 3

Simulation Design

3.1 Chapter Overview

This chapter aims to detail all relevant information of the design work that pertains to the input, and subsequent output, of simulation data to be analysed. This includes sections outlining the simulation environment used, the formation algorithms developed specifically for this research project, and the implementation of the algorithms and UAV physics within the simulation environment.

3.2 Simulation Environment

3.2.1 OMNeT++

In order to reduce the number of outside variables and ensure completion of the project within the required time frame, the use of a simulated environment is necessary. OMNeT++ is a discrete event-based network simulator primarily based on the C++ coding language. OMNeT is open source and has been designed to be modular, allowing users to both modify the base simulation environment whilst also creating custom extensions that build on existing modules and templates.

Many extensions for OMNeT have been created and are available that fill niche roles within the simulated network environment. SimuLTE builds on the OMNeT framework to

further simulate mobile wireless transmission methods such as 3GPP LTE (3rd Generation Partnership Project Long Term Evolution) to perform complex systems-level analysis of performance. VeINS (Vehicle In Network Simulation) builds on both OMNeT and the road-traffic simulator SUMO (Simulation of Urban Mobility) to provide tools for evaluating inter-vehicle communications. These examples show how OMNeT can be built upon to provide more specific tools for modelling communication networks in the real world.

3.2.2 INET Framework

Like SimuLTE and VeINS, INET is an open-source model library for the OMNeT++ simulation environment. It provides protocols, agents and other models for researchers and students working with communication networks. INET contains many models that can be used for further evaluating and validating communication and network protocols, such as HTTP, physical, and link-layer tools. Though largely used in validating new protocols, INET has modules aimed at simulating and visualising node movement within the environment. This can include mobile phone nodes moving within a building and communication nodes moving in 3D space. These modules are especially useful when creating moving simulations such as vehicular networks, overlay/peer-to-peer networks, or LTE. Several other simulation frameworks take INET as a base, and extend it into specific directions.

3.2.3 Network Descriptor Files

As OMNeT is primarily designed as a communication simulator it uses specialised files adjacent to the C++ logic to represent the creation of network models and simulation initialisation variables. Network descriptor (NED) files (.ned extensions) allow networks to be built in a modular, hierarchical fashion similar to that of classes in Object Oriented Programming languages. Networks range from simple to compound modules with customisable interconnections. Multiple networks for a simulated workspace can be created that either extend on existing modules, or are stand-alone modules that act on nodes specific to its configuration setup. NED files contain information such as the layout of networks (including node connections), node physical parameters, and initial placement vectors. NED files rely on modular implementation as it is impractical to code new net-

works for every use-case, especially niche or exotic network designs. Instead, compound networks can be developed which build from pre-existing network designs, with the impetus that any module being too complex as a single entity should be broken down into multiple simple modules that are combined into a compound module. Breaking down complex networks into simple modules allows them to become reusable outside of the specific use-case for which they are first written.

3.2.4 Initialisation Files

Upon simulation start, the program reads the appropriate NED files, then the initialisation files. Initialisation files (.ini extension) are used by OMNeT to setup the overall simulation variables. Just like network descriptor modules, initialisation configurations can be set up in a modular, hierarchical fashion or as stand-alone configurations. This file lays out the configurations of the simulation including network(s) utilised, node parameters, and communication configurations. For complex systems, configuration settings are best setup in a extendable modular fashion to allow for several configurations that have similar parameters except for a few key changes. These parameters are assigned in the appropriate network descriptor file of each module that is called. Not every parameter must be assigned a value if it is assigned a "default" value within the network descriptor file. If no value is assigned to the parameter in the initialisation file, the default is used. If a value is not assigned, nor a default given, the program will ask for a value upon start-up of the simulation. Figure 3.1 shows an example of code used in the initialisation file.

```
10 *.visualizer.osgVisualizer.typename = "IntegratedOsgVisualizer"
11 *.visualizer.osgVisualizer.sceneVisualizer.typename = "SceneOsgEarthVisualizer"
12 *.visualizer.osgVisualizer.sceneVisualizer.mapFile = "boston.earth"
13 # Coordinates of the scene origin on the map
14 #*.coordinateSystem.sceneLongitude = 150.8deg richmond
15 #*.coordinateSystem.sceneLatitude = -33.6deg
16 *.coordinateSystem.sceneLongitude = 146.766402deg #townsville
17 *.coordinateSystem.sceneLatitude = -19.248006deg
18 **.networkConfiguratorModule = ""
19 *.visualizer.*.mobilityVisualizer.displayMobility = true # master switch
20 #*.visualizer.*.mobilityVisualizer.displayPositions = true
21 *.visualizer.*.mobilityVisualizer.displayOrientations = true
22 *.visualizer.*.mobilityVisualizer.displayVelocities = true
23 *.visualizer.*.mobilityVisualizer.displayMovementTrails = true
24 #*.visualizer.*.mobilityVisualizer.positionCircleRadius = 4
```

Figure 3.1: Initialisation configuration code.

3.2.5 Nodes Representing UAVs

Though shown in the figures throughout this document, and the videos of the associated presentation, at the OMNeT kernel level the swarm and malicious UAVs are not registered as UAVs but instead communication nodes. OMNeT is designed so that each node can be set a range of pre-defined or even custom attributes based on what it is meant to represent within the network. Examples include radio transmitters with propagation characteristics, routers with associated switching and packet processing lag, or application software that processes packet data. The INET framework further modifies and augments the nodes through the ability to move in real-time for pre-defined or dynamic ways, as shown with the malicious UAV and swarm UAVs respectively.

3.2.6 Batch Processing

The final simulations required the testing of multiple different independent variables, alongside many iterations of the same simulation parameters for randomisation (see Chapter 4 for details). This combination of variables results in the need to perform 2430 simulation runs. Though OMNeT allows for individual simulation runs within its Graphical User Interface (GUI), it is also configured to carry out batch runs through the Command Line without the need to see each run. This reduces the time and effort taken to run the simulations considerably. Utilising the GUI results in each simulation taking approximately 7 seconds to run, with the need to physically change the parameters each time. This would result in over 5 hours of hands-on effort. After configuring the appropriate loops, the batch command can carry out the same number of simulations in the background in approximately 40 minutes with no human interaction required. Figure 3.2 shows the hierarchy of loops that the batch process runs through for this project.

```
for(Paths A:C)
    for(Speed Levels 1:3)
        for(No. UAVs 2:10)
            for(Formation Type Follow, Surround, Cone)
                for(Random Iteration 1:10)
                    Output results to text file
                end
            end
        end
    end
end
```

Figure 3.2: Summary of batch loops.

3.2.7 Inter-UAV Communication

Though OMNeT is configured as a communication simulator, the functions to configure communications exist within the NED files. From here, modules for any type of normal or exotic communication network can be defined through pre-made packages that are imported to the NED file. Data packet propagation delays, radio communication latency, Wi-Fi standards - all of these communication specifications can be implemented. However the requirement for this information to be processed within the C++ files presents the issue of routing the relevant data into the custom DroneMobility module. This was initially undertaken but not implemented as the time required to develop the appropriate code became burdensome, especially considering communication effects were not part of the core specifications of this project. As such, a work around in the form of a static array was implemented that stood in for a flood-type broadcast network between the swarm UAVs. Due to the nature of implementation, this array exists within "heap" memory which can exist similar to a global variable in that it is accessible outside of the object that created it. Unlike global variables, static variables created within an object (such as the DroneMobility class) are only accessible to other objects of the same class type, preventing functions from other classes within the program from manipulating the values (Malik 2014). This means that a static array created by the swarm UAVs would be accessible to any swarm UAV, but not to the malicious UAV which is of a different class.

Table 3.1: Static communications array index data information.

Column Index No.	Information Stored
0	Target acquisition: -1 if no target, 1 if target within range
1 - 3	Targets' percieved X/Y/Z coordinates (post-accuracy function)
4 - 6	Actual target X/Y/Z coordinates (for debugging)
7	UAV Surround formation Position Node ID

The static array `s_swarmInfo` is used to store the information for communication between swarm UAVs. This array is initialised with a fixed number of rows and columns (30 x 30), with the rows representing each UAV ID and the columns storing specific information for transmission. Table 3.1 lists the specific information stored in each array index.

3.3 Simulation Implementation

Though powerful in their ability to simulate communication and moving nodes, OMNeT++ and INET allow nodes to behave in ways that would not be possible in real life. Acceleration, maximum speed, maximum turning rate, and sensor range are just a few of the major factors that need to be modelled in order to evaluate the effectiveness of UAVs. As such, custom code is required to both input simulation commands to the OMNeT/INET framework as well as pull relevant data for analysis. The modules expanded upon below are coded in C++, and though they ultimately are used by OMNeT as mobile nodes, they are created as objects within the C++ environment. This allows the use of built-in C++ tools like object-specific variables, global-level array access, and the ability to create a dynamic number of nodes without the need for pre-programming. The ability to expand the number of swarm UAVs is especially useful for this project as 2,340 simulations are run with varying numbers of objects (in this case pseudo-coded UAVs) for each simulation run. Figure 3.3 shows how the OMNeT++ GUI (Graphical User Interface) represents a simulation run involving 6 swarm UAVs.

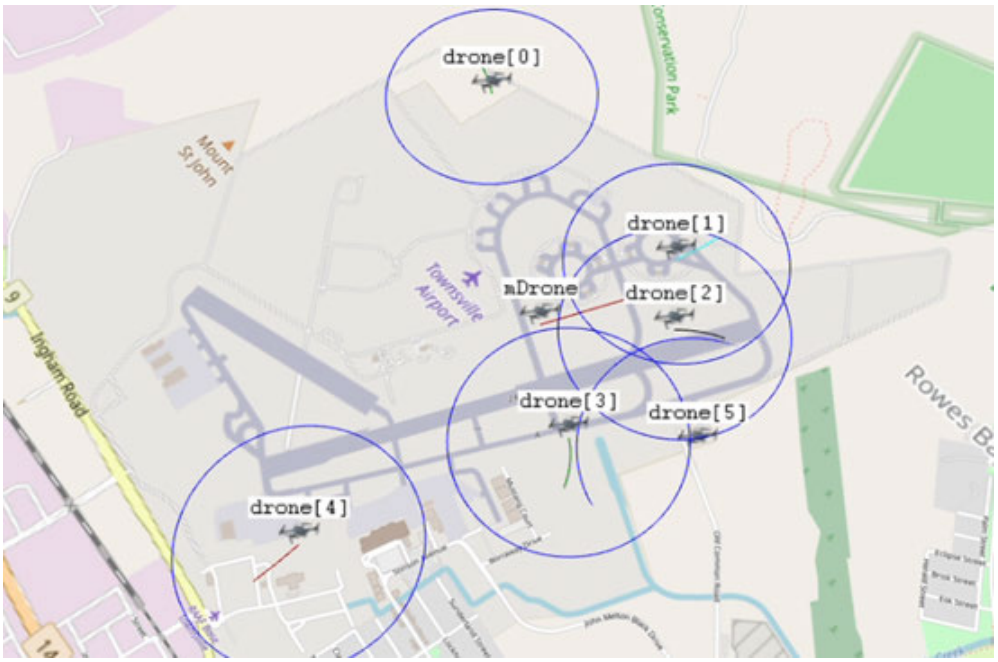


Figure 3.3: Example of simulation through GUI.

3.3.1 `DroneMobility` module

The `DroneMobility` module is comprised of the base .cc and .h C++ files with its required OMNeT NED (Network Descriptor) file. These files represent the majority of the work for this dissertation. `DroneMobility.cc` acts as the logic for all nodes created by OMNeT++ that are classed as a swarm UAV. Initialisation of each UAV upon creation is carried out upon simulation start, with any dynamic variables being assigned at this point for use throughout that simulation iteration. For each simulation time period the OMNeT kernel runs all libraries and code in the background before calling each nodes' `move()` function. Outside of access to the objects constructor and destructor this is the entry point of the kernel to the logic that directly influences the actions of the UAV nodes, both swarm and malicious. It is from this function that all subsequent UAV functions are called. The swarm UAVs go through specific functions before and after running the `setTargetPosition()` function that sets the point the swarm UAV will move to next. Figure 3.4 shows an overview of the function flow within the `DroneMobility` module.

Initialisation Functions

Two functions are called by the OMNeT kernel during initialisation. Simulation initialisation has up to 12 stages, with `initialize()` called during the first (initial) stage. This function allows for the initialisation of any variables needing to be created and populated with information from the initialisation file or simulation input. `setInitialPosition()` takes appropriate data from the `omnetpp.ini` or `droneNetwork.ned` files (in that order of precedence) to place the UAVs according to their ID. This function can initialise the location of each node based on either latitude & longitude or X & Y coordinates, but, if both are present, will prioritise latitude and longitude inputs over X/Y coordinates.

Run-time Functions

`move()` is the main entry and exit point that the OMNeT kernel uses to access the module. During the simulation run all functions are run from within this function. Upon exiting the `move()` function the OMNeT kernel moves to either the next UAV (in order of ID number). Once all swarm and malicious UAVs' code has been run, the kernel increments the simulation time by the specified time increment (i.e. 0.1 seconds) and begins again.

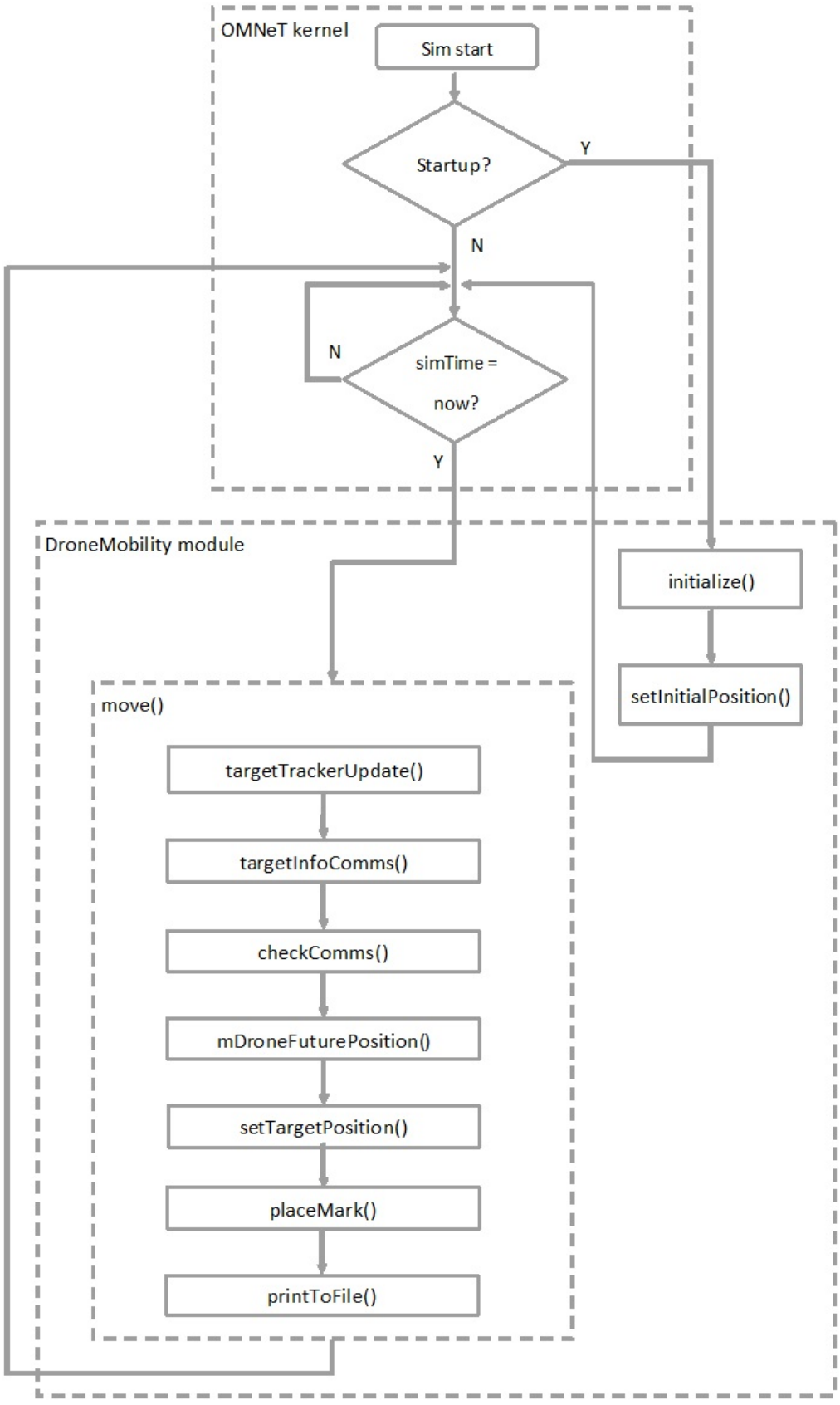


Figure 3.4: `DroneMobility` function flow.

Sub functions were created within the `move()` function to separate the code that controls different aspects of the swarm UAVs. These functions and their primary roles are:

`targetTrackerUpdate()` - simulates acquiring information from an image processing unit on board the UAV. Implementation of a randomised accuracy modifier were to be added in this function

`targetInfoComms()` - simulates broadcasting relevant data specific to the current swarm UAV such as perceived target position, current UAV position, and its current swarm target node number

`checkComms()` - simulates receiving and storing relevant data from other swarm UAVs

`mDroneFuturePosition()` - performs simple linear analysis of the malicious UAVs movements based on previous malicious UAV data.

`setTargetPosition()` - the main function called by move() and contains the state machine that defines which movement mode the swarm UAV is operating in. Except when in the stationary state, each other state performs similar base functions of target modification (if in a swarm mode), turn rate restriction and setting velocity.

`printToFile()` - used to store the information from the current time period and specific UAV for extraction after the simulation has completed. Data such as formation mode selected, the current iteration number (for randomised results), and the distance from the UAV to its target are stored in a static array.

`placeMark()` - places an "x" on the simulation to show the target position of each swarm UAV. This helps visualise the swarm formation algorithm at work, whilst also allowing for tracing bugs during development

Within the `setTargetPosition()` function are four UAV flight states that the swarm UAVs may be in at any time (discussed later). Several of these states utilise sub-functions common between them. These include:

`speedModify()` - takes a value input and modifies the current speed to match, within the acceleration abilities of the UAV

`circleFunction()` - determines if the target position is within or outside of the maximum turn radius. If outside, modifies target vector to conform with turn radius restrictions

`swarmTarget()` - depending on the mode selected, this function either modifies the target position to form around the malicious UAV in a cone or circle pattern, or trailing behind for "follow"

State-based implementation

To simplify the code a state-based implementation of flight modes was applied to the swarm UAVs within the `setTargetPosition()` function. Four modes were created; stationary, chase, approach, and formation.

STATIONARY - the initial state of the swarm UAVs upon initialisation. This state checks for any objects (malicious UAVs) within sensor range. If within range, the UAV changes to CHASE state.

CHASE - accelerates the swarm UAV to its maximum speed whilst checking the relative velocity between the malicious UAV and itself to ensure it has enough time to slow and not overshoot its target. If the distance to the target is less than the distance to slow, APPROACH mode is activated. If the target moves out of sensor range, STATIONARY mode is activated.

APPROACH - an intermediate state that slows the UAV to match speed and orientation with the malicious UAV. Once the UAV has entered within a 15-meter radius of the target, FORMATION state is activated.

FORMATION - operating within a 15-meter radius of the target, this state ensures the UAV can adapt to sudden changes in target position (such as moving behind the UAV) without causing it to loop around. This is achieved by matching the heading of the malicious UAV and regulating its speed based on the relative velocity between itself and the target position.

Specific source code listings for the `MaliciousDroneMobility` module can be found in Appendix D.

3.3.2 `MaliciousDroneMobility` module

As with the `DroneMobility` collection of files, `MaliciousDroneMobility` incorporates a main C++ file, as well as a header and NED file. The .cc file contains methods (class functions) that form the basis for all movement of the malicious UAV. Figure 3.4 shows an overview of the function flow within the `DroneMobility` module. The module functions fulfil the following roles:

`setNewWaypoint()` - upon reaching the set waypoint, this function accesses the `.movements` file to determine the next waypoint

`checkWaypoint()` - checks current position to see if the UAV is at the waypoint targetted. A margin of 2 meters from the specified X/Y waypoint location is allowed to ensure location errors do not result in the UAV not proceeding to the next waypoint

`setTargetPosition()` - modifies the target position (the next waypoint) to account for restrictions of speed, acceleration, and maximum turn rate

Missing from the list are functions that have been adapted from `DroneMobility` that relate to UAV flight characteristics. These include `speedModify()`, `setInitialPosition()` , and `circleFunction()` and are identitical in their code and implementation. As such, their descriptions have not been repeated.

At the initialisation of the simulation, the module accesses the specific `.movements` file required that is located with the UAV project simulation folder (where the main project `omnetpp.ini` configuration file is located). These files contains groups of four numbers that represent waypoint information for the malicious UAV. The first group of four numbers takes the format:

```
[Acceleration, Initial X Position, Initial Y Position, Initial Z Position]
```

This is read upon simulation start by the `MaliciousDroneMobility` module to place the malicious UAV at the specified coordinates, as well as set the maximum acceleration of the UAV during that simulation run. The proceeding number-groups follow a similar format, but are read as:

```
[Maximum Speed, Next Waypoint X Pos, Next Waypoint Y Pos, Next Waypoint Z Position]
```

In both cases the Z position is set at 20 meters, though this does not affect the results as the simulation is only conducted within the horizontal 2D plane for this project. The "maximum speed" value represents the speed the malicious UAV will fly at until the next waypoint, not including time taken to accelerate/decelerate to that speed. This allows the ability to change the speed between waypoint legs. To reduce complexity and the number of independent variables in this project, the speed is set for the entirety of the simulation run, with the exception of slowing to 0 m/s at the end of the simulation.

In order to allow batch runs to occur, a `.movements` file exists for each of the 9 combinations of 3 speed Levels and 3 Paths defined in Chapter 4. The OMNeT simulation program access each in turn as is required to perform the specific simulation.

Specific source code listings for the `MaliciousDroneMobility` module can be found in Appendix E.

3.3.3 `DroneNetwork.ned`

`DroneNetwork.ned` is the main network layout file for this project. As mentioned, the NED file is where the overall network structure and the nodes within are configured. Though nodes are usually specifically created and placed by the user, OMNeT allows nodes to be created in an array-style at start-up. Only one network is configured for this project - "DroneNetwork" - that is utilised by all configurations within the initialisation file. Typically communication paths would be set up within this file but, as a static array is used for passing inter-UAV information, this is not required. Other details such as the background image, node representations images, and common visualisation information is set up in this file. The `.ned` source file can be found in Appendix F.

3.3.4 `omnetpp.ini`

`omnetpp.ini` is the main initialisation file that the simulation runs upon start-up after any associated NED files. For this project there are three configuration setups utilised that are extensions of one main (base) configuration setup that contains cross-config data. The layout of the configurations created follow those shown in Figure 3.5. Figure 3.6 shows

the changes made in the `omnetpp.ini` file that allows for easy change of formation modes within the simulation environment. Further lower-level and generic parameters can be found in the full omnetpp.ini listing in Appendix G.

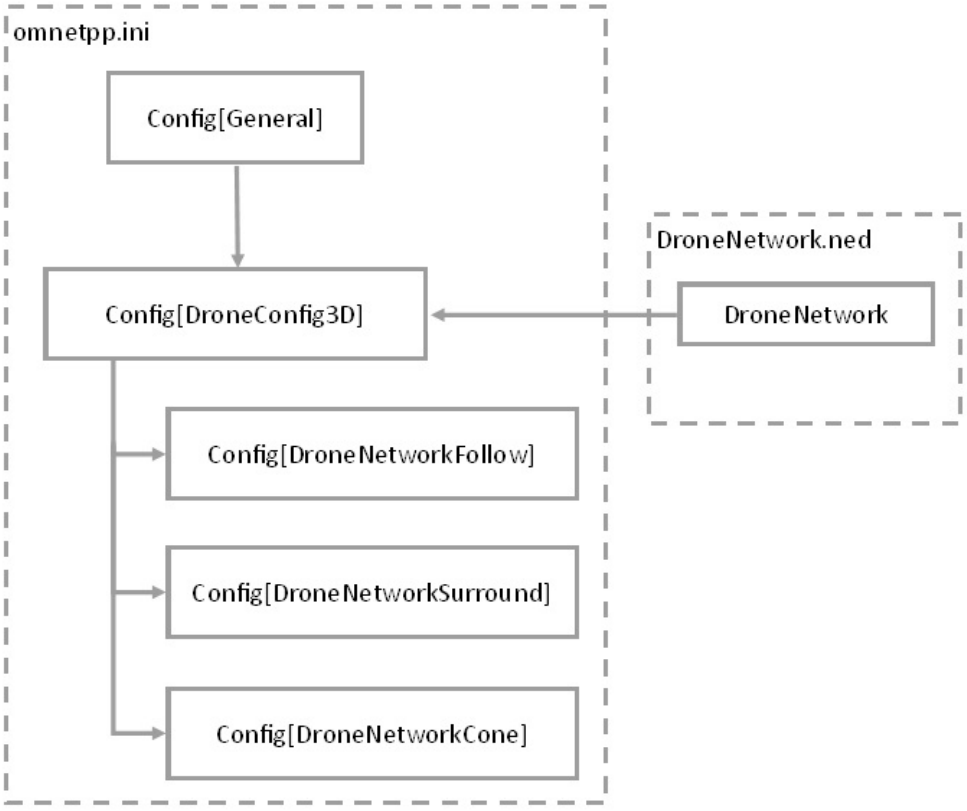


Figure 3.5: Initialisation file configuration layout.

```
185 [Config DroneNetworkFollow]
186 extends = DroneConfig3D
187 *.drone[*].mobility.droneMode = "direct" #direct or swarm
188 *.drone[*].mobility.swarmPosMode = "follow"
189
190 [Config DroneNetworkCone]
191 extends = DroneConfig3D
192 *.drone[*].mobility.droneMode = "swarm" #direct or swarm
193 *.drone[*].mobility.swarmPosMode = "cone"
194
195 [Config DroneNetworkSurround]
196 extends = DroneConfig3D
197 *.drone[*].mobility.droneMode = "swarm" #direct or swarm
198 *.drone[*].mobility.swarmPosMode = "surround"
```

Figure 3.6: Derived configuration classes.

3.4 Physical Constraints

3.4.1 Speed & Acceleration

OMNeT nodes (through the INET framework) are not restricted to any form of physics in their ability to move within the simulation. Nodes may have any speed, and may achieve that speed in the discrete time period that the simulation is running (0.1 seconds by default). Maximum speed and acceleration needs to be applied to nodes operating within these simulations as UAVs in order to recreate real-world effects and maintain validity of the output data. For the swarm UAVs, the maximum speed they may travel and how fast they may accelerate or decelerate is set within the `omnetpp.ini` file and is used for all simulations. The malicious UAV takes its acceleration/deceleration and maximum speed data from the `.movements` files, each of which has specific information regarding the physical limitations for the given simulation parameters.

3.4.2 Turn Radius

Due to the discrete nature of the simulation environment mentioned above, the arc that the UAV would traverse during the 0.1 second jump is translated through an application of the law of cosines to a vector that points to the location along the arc that the UAV would travel in the given time. This is calculated within the `circleFunction()` function.

Fotouhi, Ding & Hassan (2017*b*) detailed the equations that define the instantaneous manoeuvrability of a UAV for a given velocity and lateral acceleration. Specific UAV characteristics such as lateral acceleration are not readily available, but due to the symmetrical nature typically found in commercial UAVs it can be taken that the forward acceleration would be a close approximation of the overall acceleration abilities of the UAV. The CSIRO research team detailed that the radius (r) of the circle that described the turning arc and the angle (θ) between the current UAV position and the point on the arc it will travel in the time period can be found thus:

$$r = \frac{v^2}{a}; \theta = \frac{ta}{v} \tag{3.1}$$

where v is the current UAV velocity (m/s);

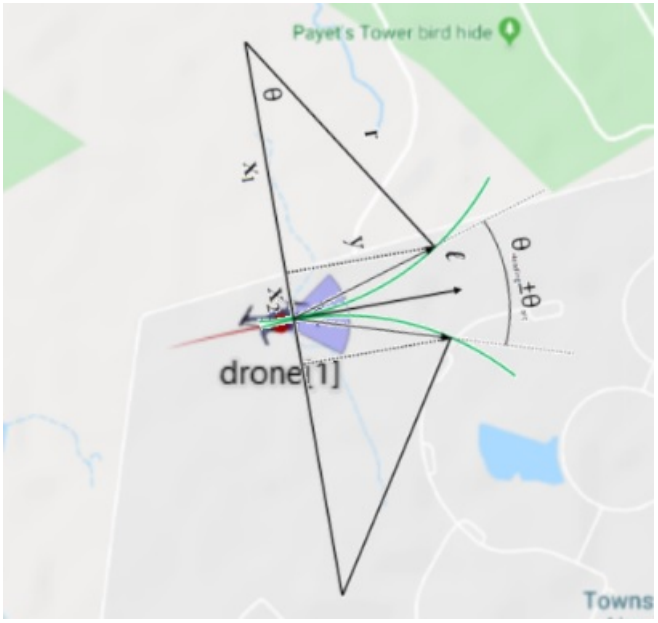


Figure 3.7: Trigonometric calculation of a discrete vector representing a turning arc.

a is the lateral acceleration ability of the UAV (m/s^2); and
t is the time period in which the UAV turns (0.1 seconds).

Obtaining the movement vector that represents the point the UAV would travel to along the turning arc in the given time period is simply a matter of applying the Law of Cosines to obtain the vector from the UAV to point P in Figure 3.8. Utilising the general equation for the Law of Cosines and using the side definitions found in Figure 3.8 results in Equations 3.2 and 3.3:

$$c^2 = a^2 + b^2 - 2ab\cos(\theta_C) \tag{3.2}$$

$$a^2 = b^2 + c^2 - 2bc\cos(\theta_A) \tag{3.3}$$

The sides b and c in Equations 3.2 and 3.3 are of the same length r. The angle θ_A in Equation 3.3 is equivalent to θ as represented in Equations 3.1. Substituting r and θ into Equations 3.2 and 3.3 and solving Equation 3.2 for θ_C and results in:

$$\theta_C = \cos^{-1}(\frac{a^2}{2ar}) \tag{3.4}$$

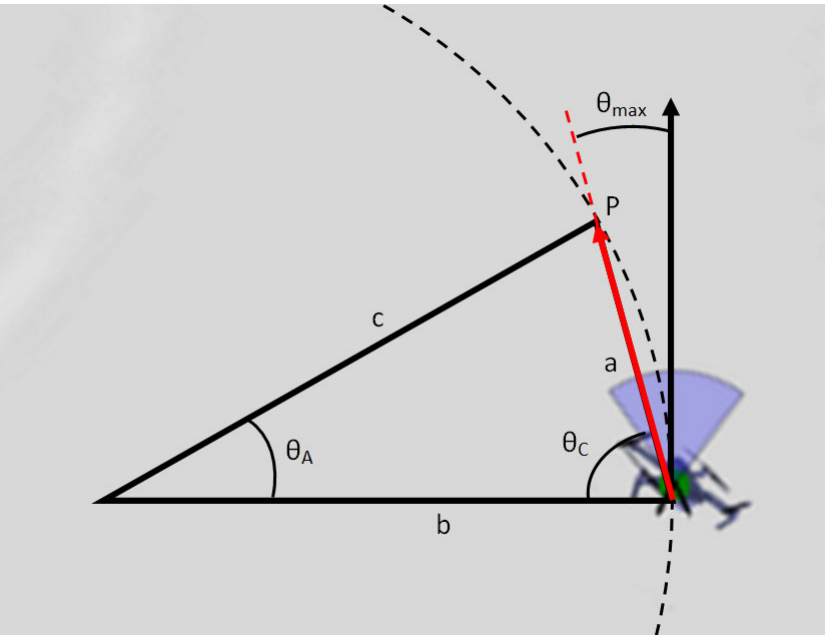


Figure 3.8: Visual representation for derivation of $\vec{SP}$ and θ_{max}.

$$a^2 = 2r^2(1 - \cos(\theta)) \tag{3.5}$$

Combining Equations 3.4 and 3.5 and subtracting from $\pi/2$ results in the maximum turning angle from the current heading θ_{max}:

$$\theta_{max} = \frac{\pi}{2} - \cos^{-1}\left(\sqrt{\frac{1 - \cos(\theta)}{2}}\right) \tag{3.6}$$

This results in a vector $\overrightarrow{SP}$ of magnitude a denoted in Figure 3.8. Deriving a from Equation 3.5 equates to:

$$a = \sqrt{2r^2(1 - \cos(\theta))} \tag{3.7}$$

and completes the formula `circleFunction()` uses to determine the point the swarm UAV will be after the 0.1 second time interval. The full form vector is:

$$\overrightarrow{SP} = |a|\angle\theta_{max}$$

$$= |\sqrt{2r^2(1 - \cos(\theta))}|\angle\frac{\pi}{2} - \cos^{-1}\left(\sqrt{\frac{1 - \cos(\theta)}{2}}\right) \tag{3.8}$$

3.5 Project-Specific Algorithms

The core of this research project is to evaluate the effectiveness of multiple UAVs (i.e. swarms) working together to pursue a target. Formation algorithms were required to be developed in order to achieve this objective. As the project developed several other algorithms were conceived that would likely increase the efficiency or effect of the formation algorithms. Swarm target efficiency coding and a novel path finding theory was developed, though due to time constraints path finding was not implemented.

3.5.1 Swarm & Baseline Formations

Follow Mode

When set to "follow", the swarm UAVs target the direct rear of the malicious UAVs' position (Figure 3.9). This mode is used as the baseline for which surround and cone formations are compared. As such, the swarm UAVs also do not communicate with each other as to the known position of the malicious UAV, nor do the swarm UAVs make any attempt to coordinate into a formation. As noted in Section 4.2.1, collisions are not enabled for this project as the complexity of coding collision avoidance path finding was outside the time-frame of the project. This means that the UAVs typically are in close proximity by the end of the simulation as they are all targetting the same position.

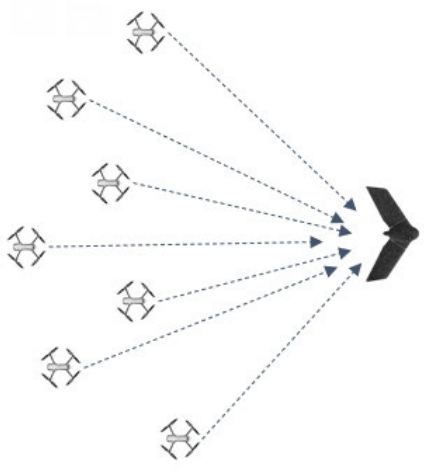


Figure 3.9: Follow-type formation.

Surround Mode

This a coordinated formation mode that transcribes a circle around the malicious UAV with the swarm UAVs being positioned evenly around the circle perimeter. This formation algorithm adapts to the number of UAVs within the swarm by increasing the circle radius to maintain a constant perimeter distance between each UAV. This is achieved through the allocation of a fixed ratio of area within the circle proportional to the number of UAVs making up the swarm. Based on an initial radius of 150 meters for one UAV equates to 70,685 m^2 per UAV. Figures 3.11 - 3.11 shows the differences in radius and perimeter distance between swarms of 2, 3, and 4 UAVs.

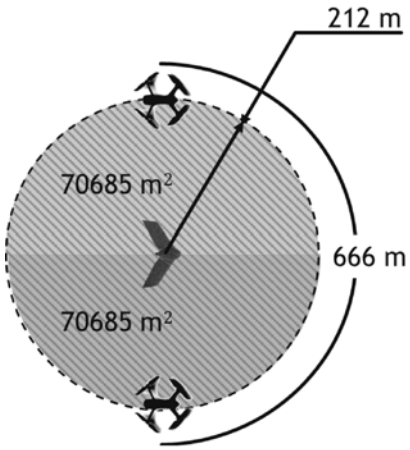


Figure 3.10: Surround profile of a 2 UAV swarm.

This method of spacing and allocation allowed for a processor-friendly method of ensuring that the space around the malicious UAV did not become crowded due to a fixed radius resulting in decreasing perimeter separation as the swarm grew. This also meant that an increase in swarm size meant a larger coverage of area and an increased separation from the malicious UAV, allowing for better reaction to changes in the malicious UAVs heading. The initial radius of 150 meters was determined as a sufficient distance based on experience from several simulation runs, and could be adjusted to allow for a tighter or looser formation. Table 3.2 lists the specific distances that are used for each size of swarm in surround formation, including the individual perimeter distance and the minimum direct separation of the UAVs. As shown in the table, this algorithms method of ensuring relative consistency of separation for each swarm size results in an area coverage an order of magnitude larger for 10 UAVs than for a single UAV which increases the swarms ability to continue pursuit.

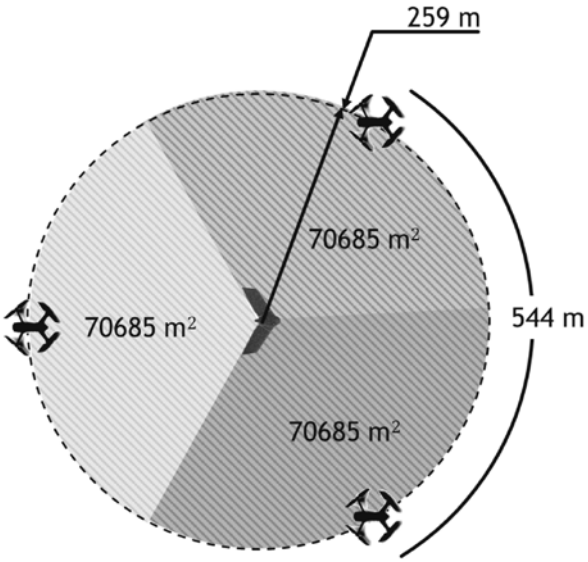


Figure 3.11: Surround profile of a 3 UAV swarm.

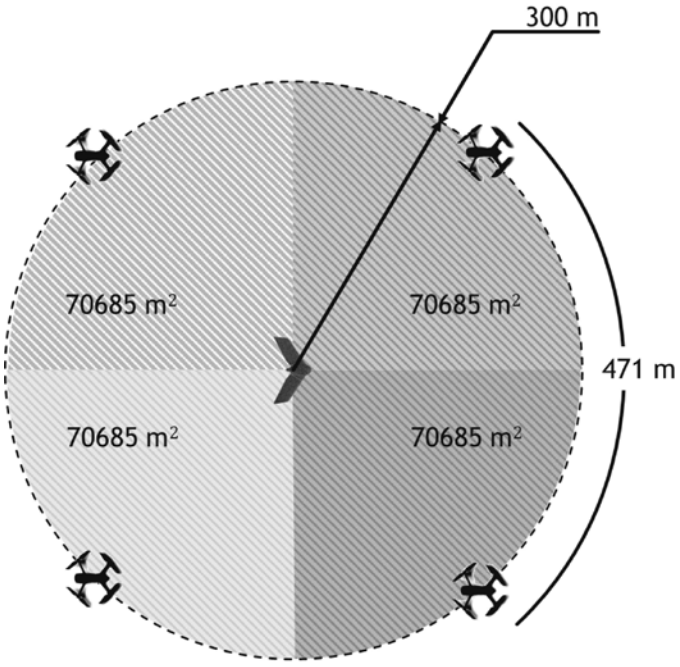


Figure 3.12: Surround profile of a 4 UAV swarm.

Table 3.2: Separation metrics for the surround formation.

No. UAVs	Total Area (m^2)	Radius (m)	Perimeter UAV Separation (m)	Direct UAV Separation (m)
1	70,685	150.00	N/A	N/A
2	141,370	212.13	666.43	424.26
3	212,055	259.81	544.14	450.00
4	282,740	300.00	471.24	424.26
5	353,425	335.41	421.49	394.30
6	424,110	367.42	384.76	367.42
7	494,795	396.86	356.22	344.38
8	565,480	424.26	333.21	324.72
9	636,165	450.00	314.16	307.82
10	706,850	474.34	298.04	293.16

Cone Mode

This is the second coordinated formation that forms a triangle from an overhead view with the top 'point' just behind and the 'base' of the triangle ahead of the malicious UAV (Figure 3.13). the placement algorithm distributes each extra UAV added to the swarm at a position to ensure priorities are placed on filling the points of the triangle, before placing UAVs equally along each side. This formation was created with the intent to determine if placing UAVs further forward and to the sides of the malicious UAV would increase the combined abilities of the swarm to keep within pursuit range compared to the surround method.

The restriction of working within the horizontal 2D plane means that only a triangle form is necessary for this project. However, this formation is named 'cone' as it is envisaged that expansion of this research into the 3D space will revert this formation to its 3D version which would be represented as the swarm taking up positions on the perimeter of a cone form around the malicious UAV.

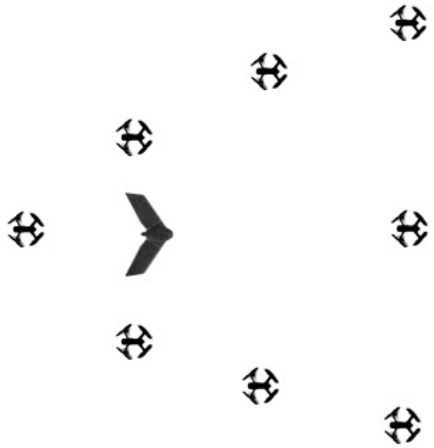


Figure 3.13: Cone formation example.

3.5.2 Swarm Target Efficiency

The main issue with the two swarm UAV formations developed were that the positions that the swarm UAVs would take were based solely on their UAV ID (i.e. 1 - 10). The initial positioning algorithm did not make any effort to determine the most efficient allocation of position nodes based on relevant information such as which UAV was closest to the node. Towards the end of the project, time was spent developing such an algorithm based on each individual UAVs distance to the available target nodes.

`surroundEfficiency` is a subfunction developed for the surround formation as an initial test-bed to determine if there was any significant increase in swarm pursuit ability caused by more intelligent selection of node positioning. Figure 3.14 shows the flow diagram of the logic used within this function.

The algorithm contained within `surroundEfficiency` stores the distance from the UAVs current location to each node location available based on the number of UAVs within the swarm, and the associated node ID number. A bubble sort is used to reorder the values in ascending order. The function then checks the static function of each swarm UAV whose ID number is less than its to see if they have already reserved that node position. This is required due to the way that OMNeT runs the simulation - in particular the fact that each UAVs' code is not run concurrently but consecutively. This means that the UAV that holds the '0' ID number will run before any others, and will not be able to react to any updated information from subsequent UAVs within the same time period until the

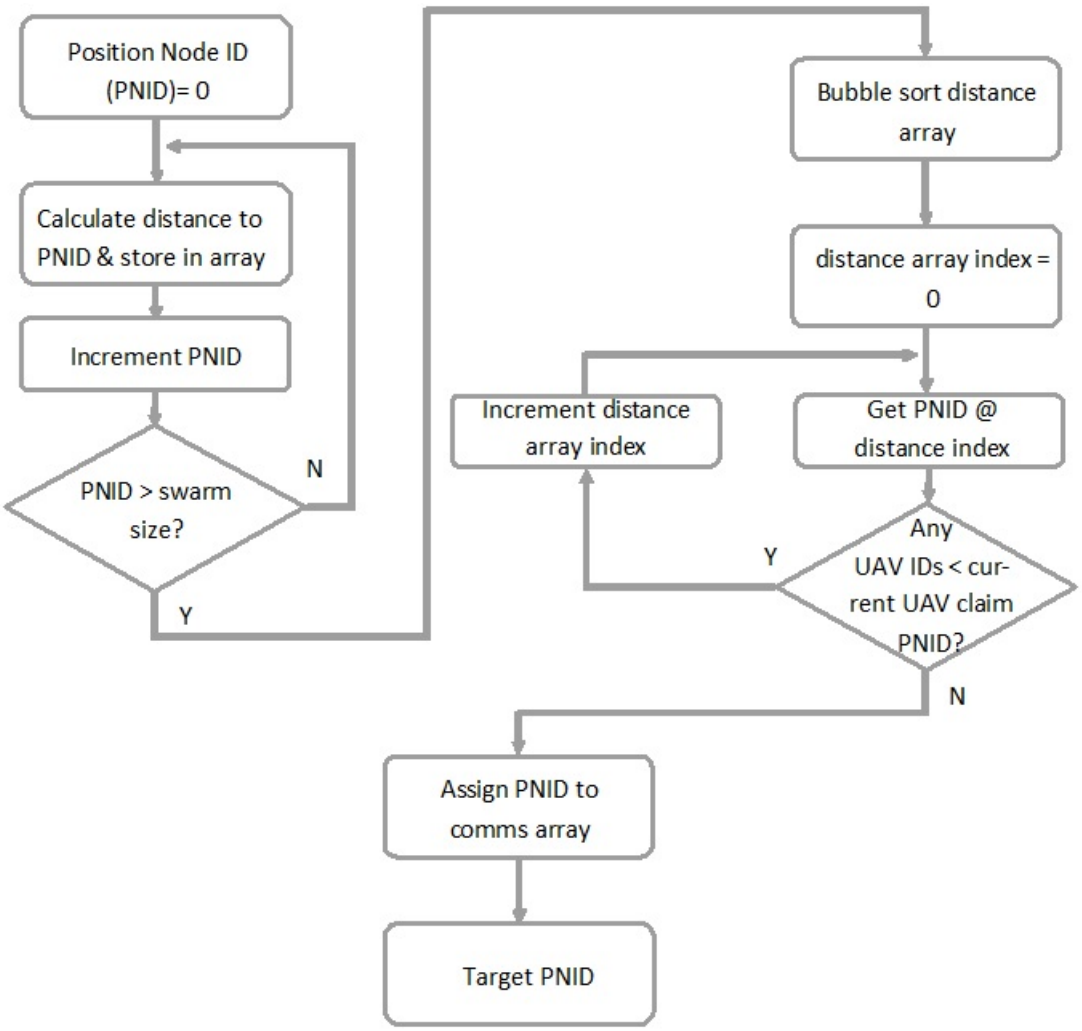


Figure 3.14: `swarmEfficiency` function flow diagram.

next time period. In order to work with this, the algorithm has to allow priority to the first UAV to run the code in that time period. This means that UAV 'drone[0]' will always have priority of choice over all others, even if the sum total distance is not optimal.

This function was implemented in time for final simulation runs and the results are commented on within Chapter 5

3.6 Unimplemented Algorithms

Initially the project was to encompass many features that are possible through simulation. These included:

- Efficient real-time path finding with correct final position orientation

- Enrolment of swarm UAVs based on the predicted path of the malicious UAV
- Enhanced tracking methods such as Proportional Navigation
- Multiple malicious UAVs

The following subsections will detail the issues encountered with implementing these features.

3.6.1 Path finding & Orientation

One of the main issues not adequately researched within the reviewed literature whose aim was to successfully have a device (UAV, robot, etc.) track another object was the requirement that the robot be of the correct orientation upon reaching the target position. Even when not considering a swarm of UAVs it is important that for long-term pursuit the pursuing device be in the correct orientation if and when it catches the target. Initial thoughts of using navigation techniques such as proportional navigation allows for a more efficient path to the target but, as PN techniques (and its variants) were designed for missiles to intercept and destroy a target, these techniques do not account for final orientation after interception.

A proposed method of path finding couples the limiting physics of a moving object having some finite turning ability (i.e. not being able to instantaneously change their velocity vector) with the need to reduce the total distance that the device must travel to be at the correct position and the correct heading. This is done by determining the turning arc of radius r as found in Equation 3.1 of the swarm UAV and apply it to the current UAV position and the malicious UAV (specifically the formation target point). Determining the internal or external tangent between the two circles would give the shortest path between the swarm UAV and its target taking into account the current heading of both the swarm and malicious UAV (Figure 3.15). This would also have the effect that the swarm UAV would arrive at the target position at the correct heading, negating the possibility of overshooting and having to turn around if the swarm UAV was heading in any direction not equal to the malicious UAV.

This path finding algorithm would be processor-friendly as it requires simple trigonometric formulae relying on only a few known variables. As the path would not change drastically

in a short period calculations could be skipped in a scaled manner proportional to the recent deviation of the malicious UAV from its previous course.

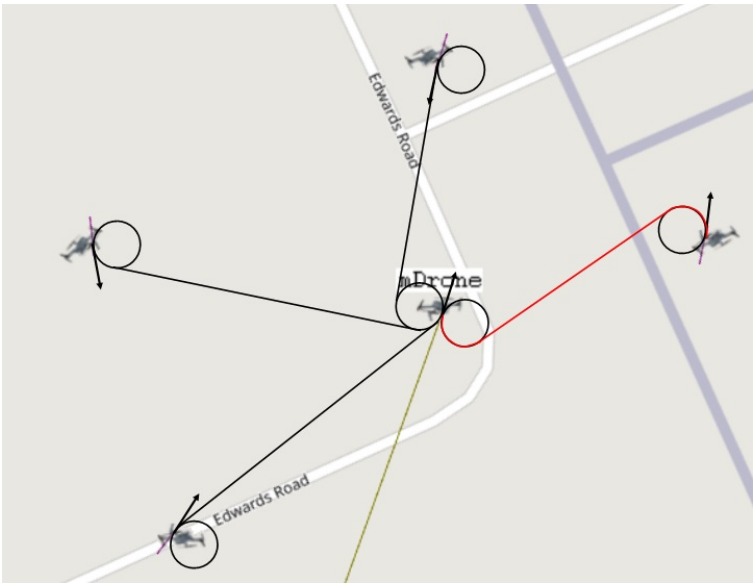


Figure 3.15: Proposed method of efficient path finding.

3.6.2 Swarm Enrolment & Malicious Path Prediction

The current method of swarm enrolment is that each swarm UAV checks for any targets within its sensor range and, if none found, checks the communications array (simulated by a static array) to see if any other swarm UAV has a target within its sensor range. Efforts to create code that predicted the path of the malicious UAV was carried out with some success and the immediate future movements (i.e. within 0.5 seconds) was found to be accurate enough that it could be developed into further algorithms. The development of an algorithm that would intelligently decide on the likely path of the malicious UAV proved to be outside of the main scope with regards to the amount of time required to implement. The advantages of a predictive path algorithm lies in the ability to leave swarm UAVs in position that would not greatly increase the ability of the swarm to pursue the target due to the distance that swarm UAVs may have to travel to get into formation.

Though not implemented, some thought did go into how such an algorithm would perform. Figures 3.16 and 3.17 show two examples of how different predicted flight paths may cause the enrolment of different swarm UAVs. Levels of certainty surround the dashed predicted flight path which would factor into which swarm UAVs would be enrolled to pursue that target, based on current swarm size and number of remaining un-enrolled UAVs.

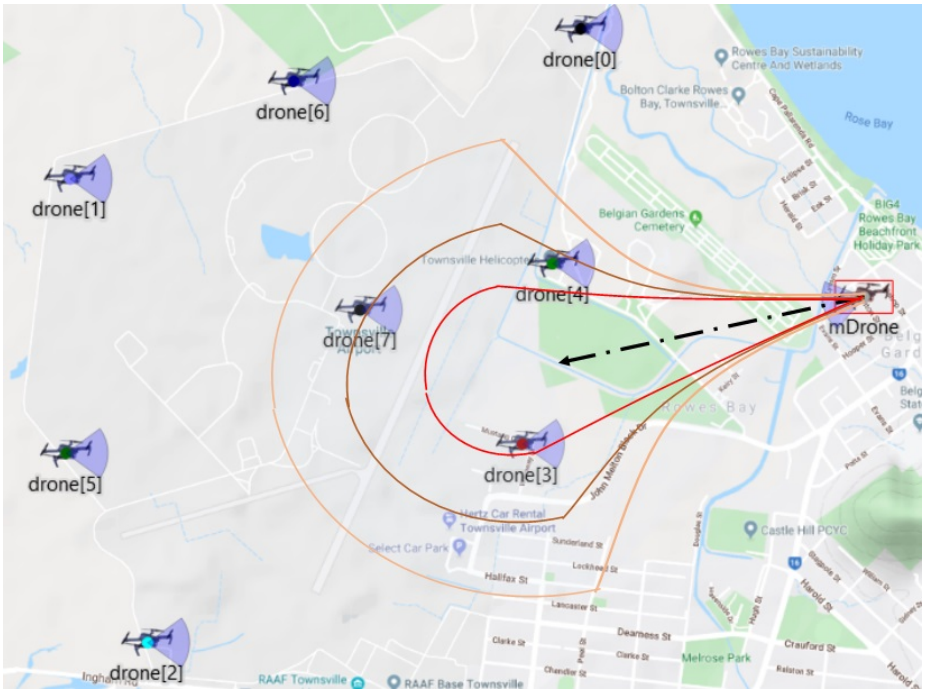


Figure 3.16: Example 1 of predicted malicious UAV path.

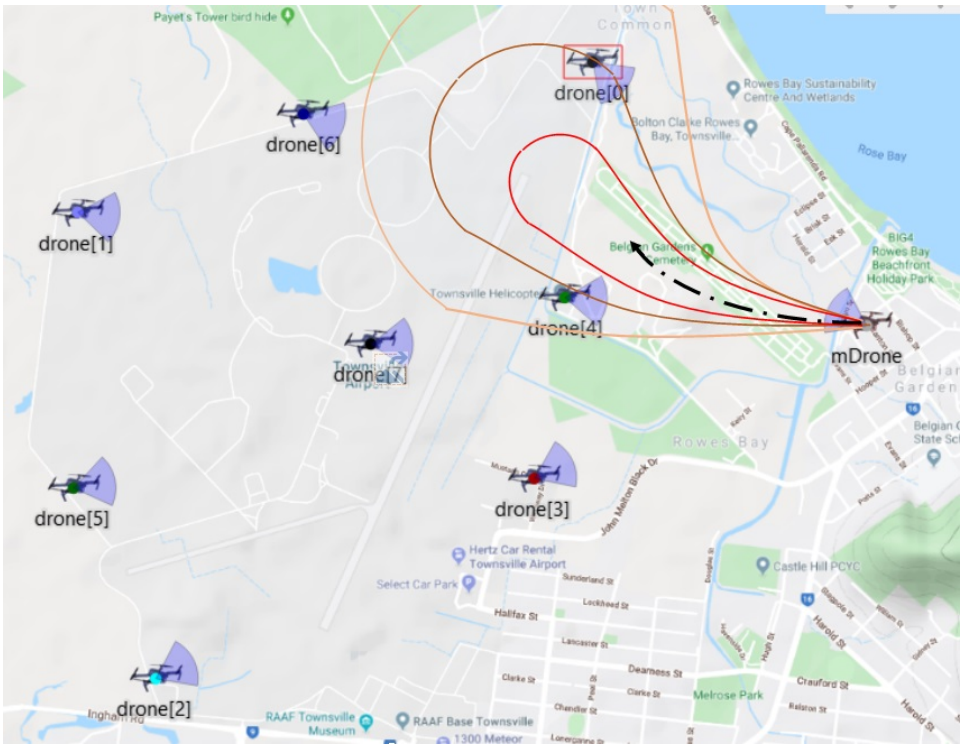


Figure 3.17: Example 2 of predicted malicious UAV path.

3.6.3 Proportional Navigation

Initial project goals included utilising existing navigation methods to increase the abilities of the swarm in pursuing targets. As discussed in Chapter 2, Proportional Navigation (PN) is a method of dynamic object path prediction used primarily in defensive missiles to reduce the chance of non-collision. PN and its variants are also used in novel cases as efficient path finding collision avoidance techniques. Efforts were made to implement a variant of PN into the coordination code of the swarm UAVs, however it was never successfully implemented as excessive time was used in debugging.

3.6.4 Multiple Malicious UAVs

Outside of processing power there are no limits to the number of simulated UAVs at any given time. With this in mind it would be useful to simulate more than one malicious UAV that is operating within the restricted area at any given time. However this would require either the swarm enrolling less than the total amount of available UAVs for pursuit during initial contact of the first malicious UAV or the ability to de-enrol UAVs from the first swarm to track subsequent targets. This level of coding presents two separate issues; the coding required for implementation would be complex and the time required to implement and debug excessive.

Chapter 4

Methodology

4.1 Chapter Overview

This chapter aims to lay out the project methodology for this thesis. Specific sections introduce the overall methodological approach, assumptions underpinning the methodology used and their rationale, tools utilised, methods of data collection, methods of analysis, and justification of the methodology chosen.

4.2 Methodological Approach

This dissertation aims to determine the effectiveness of UAVs operating as a swarm when compared to individual control. In order to achieve this, it is necessary to have quantitative data that can be compared between scenarios that represent a change in independent variables. The ability to precisely control the independent variables, as well as remove outside influences that would tarnish the data accuracy, is crucial. For these reasons this project is completed within a simulated environment through which credible data may be input to, and taken from, the simulation for further analysis.

Though not able to fully account for all real-life events, the use of simulations for this research project is appropriate as it removes any technical aspects that may delay gathering of data for analysis. These aspects include UAV faults, freak weather events, and unintentional interference or jamming. The use of software simulation also allows for testing of

various UAV platforms and situations, without the time and financial cost that would be associated with the increased scope. Use of a simulated environment eliminates the need for ethical considerations with regards to flying experimental (i.e. modified/non-OEM flight code) UAVs where there is potential to lose control of the UAVs resulting in injury to personnel or damage to property. The costs of performing research involving up to 11 UAVs would also be infeasible as even consumer-level UAVs considered retail from $600 to $2500 each (DJI 2020). This would mean even a small swarm of 5 UAVs would require $3000, without accounting for spare parts.

The data collected from the simulation constitutes primary experimental data. The data is gathered through the manipulation of several variables that are expected to have an effect on the overall performance of the swarm as a whole. These variables include:

- Number of UAVs that make up the swarm
- Formations the swarm UAVs take
- Flight characteristics of the malicious UAV, including maximum speed and acceleration
- Path the malicious UAV takes that constitutes differing levels of evasiveness

These variables are easily manipulated within a simulated environment, and many more may be coded to be changed to suit other research questions.

4.2.1 Assumptions & Rationale

In order to reduce the complexity of code required to be completed, a set of assumptions were made. These assumptions allow for a reduction in coding complexity whilst providing consistency of results between the three separate malicious UAV path models. Consistency of results is required to be able to compare the effects that each of separate flight paths the malicious UAV takes has on pursuit ability. The assumptions applicable to this project include:

- Discrete time instances set at 0.1 seconds
- Limit of 10 UAVs in the swarm

- Constant turn speed for both malicious and swarm UAVs
- All UAVs operate at a constant altitude
- No terrain obstacles will be included, nor collision avoidance between UAVs
- No range limitations due to flight time, transmission distance, or battery capacity
- Sensor accuracy for targeting of the malicious UAV is 100%

Simulation Timing & Swarm Size

All computer-based simulators operate on discrete simulation time instances. For this project, the simulation time is incremented in 0.1 second intervals. This time period allows for sufficient granularity of movement whilst not imposing an undue burden on the processor when simulating the maximum number of UAVs for the swarm. The number of UAVs within the simulation also effects the ability of the processor to run the simulation. For this reason a limit of 10 swarm UAVs is imposed which represents acceptable performance of the available equipment. An offshoot effect of the discrete nature of the simulation requires that movement be calculated as an vector residing on the UAVs' current location and ending at the calculated UAV position at the end of the 0.1 second time period.

Turn Speed

In order to minimise the complexity of the movement code, the turn speed of the UAVs (both malicious and swarm) maintain the speed at which they entered the turn for the duration of the turn. (Fotouhi et al. 2017*b*) outlines the formulae that describes the arc of a UAV for a given maximum acceleration ability for a constant speed. This only applies in the instances that the UAV is making a hard turn left or right. Further information regarding the implementation of a hard turn within the simulation can be found in Chapter 3.

Constant Altitude, Terrain & Collisions

To further reduce the large scope of the problem and therefore coding complexity, the UAVs will operate at a consistent altitude of 20 meters. This effectively constrains the UAV movements to the horizontal 2D plane. Though the UAVs will not move outside of this plane for this project, the custom coding has been designed to be transferred to 3D for further research. In line with reducing overall complexity, thought the INET framework has modules available to simulate terrain features (both natural or man-made), these were not introduced to the simulation for the UAVs to avoid due to the time required to implement. Collision avoidance code was introduced but, due to the work required to implement a suitable navigation algorithm to deal with path finding, it is disabled.

Range Limitations & Sensor Accuracy

Range limitations regarding either battery life or transmission range are key factors to the ability of a swarm to function over a longer period of time than that used within this projects' simulations. With battery capacity constantly evolving and better construction techniques resulting in more efficient flight profiles, utilising current range data as limitations would render the results of this project antiquated relatively quickly. Range limitations, therefore, were not coded into this project.

Code was initially created to simulate sensor inaccuracy. This was achieved through the use of a random walk algorithm. Unfortunately, due to the length of time required to run sufficient simulations that would give a frequency distribution of the sensor accuracy, it was left disabled. This will reduce the validity of the results though it will not be detrimental to the overall research conclusion due to all simulations having the same sensor accuracy. Every swarm UAV will have the exact X/Y coordinate of the malicious UAV through its sensor function.

4.2.2 Simulated UAV Characteristics

Swarm UAVs

In keeping with the aim of this dissertation, readily-available consumer-grade UAV characteristics need to be compared. Table 4.1 outlines the differences between market-leading UAV models currently available (September 2020). DJI makes up the majority of the list as they represent a large part of the global UAV market share, as well as offering far more models than comparable companies.

Table 4.1: Comparison of consumer-grade UAVs (DJI 2020, Autel 2020, Parrot 2020)

UAV Brand	Model	Max Speed	Max Flight Time	Max Range
DJI	Mavic 2 Pro	72 km/h / 20 m/s	31 min	6 km
	Inspire 2	94 km/h / 26.1 m/s	27 min	3.5 km
	Mavic Air 2	68.4 km/h / 19 m/s	34 min	6 km
	Mavic 2 Zoom	72 km/h / 20 m/s	31 min	6 km
	Mavic Spark	50 km/h / 13.8 m/s	16 min	0.5 km
Autel Robotics	EVO	72 km/h / 20 m/s	30 min	7 km
Parrot	Anafi	55 km/h / 15.3 m/s	25 min	4 km

From this data, utilising median speed as the swarm UAV speed results in a maximum simulated swarm UAV speed of 20 m/s. Acceleration is set to 4 m/s^2. These values are appropriate as the time taken for the malicious UAV to complete the longest path is 1,524 seconds (as noted in Table 4.3). Adding 10% for the swarm UAVs to be able to settle into position is just under 28 minutes flight time. This rules out the faster Inspire 2 as an appropriate simulated UAV as its run-time is 27 minutes, with a much shorter maximum range than the UAVs that average 20 m/s.

Swarm Initial Positions

Randomisation of the placement of the swarm UAVs is conducted upon each simulation start to ensure that the placement of the swarm by the algorithm created does not effect the results. This is done through a pseudo-random number generator that receives its seed number from the current time, allowing for a sufficient level of randomness between each simulation run. The placement algorithm pulls a new pseudo-random number for

each variable requiring randomisation thereby ensuring no linearities exist between them. The placement algorithm changes the X and Y location from a preset position to one within ±15 meters as well as the starting radial angle to any angle within the circle.

Swarm Communications

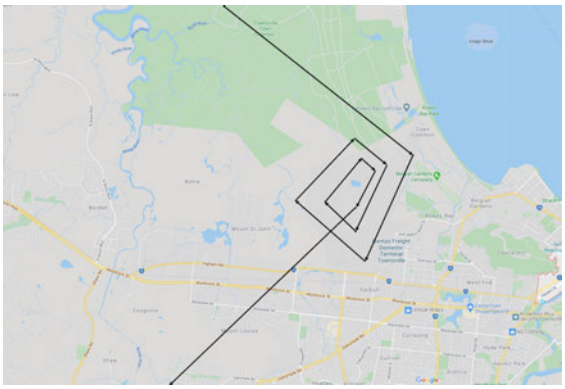
The cone and surround formations follow specific code that adapts to the varying number of UAVs in a swarm. As the swarm shares information through the communications array, each UAV is not restricted to engagement only when the malicious UAV is within its sensor range.

Malicious UAV

The malicious UAV operates at three different levels of flight characteristics that relate directly to its speed and acceleration abilities. The base level (Level 1) runs at a slightly faster speed (21 m/s) than the swarm UAVs (20 m/s) and the same acceleration of 4 m/s^2. Level 2 increases the speed to 23 m/s and acceleration to 4.5 m/s^2, with Level 3 the highest at 26 m/s and 5 m/s^2. These increases in flight characteristics are used in conjunction with differing flight paths. The first flight path (Path A - Figure 4.1a) represents a simple spiral search pattern as might be used to ensure coverage of a site for intelligence, surveillance, and reconnaissance (ISR) reasons. Path B (Figure 4.1b) aims to replicate site-specific waypoint targetting, and results in a more random pattern with sharper turns. The "Path C" flight path (Figure 4.1c) is designed solely to test the effects that an extremely erratic malicious UAV would have on the formation abilities of the swarm.

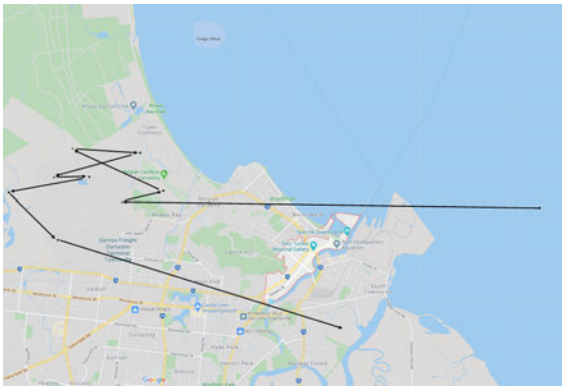
Table 4.2: Velocity and acceleration details for each Level.

Level	Velocity	Acceleration
1	21 m/s	4 m/s^2
2	23 m/s	4.5 m/s^2
3	26 m/s	5 m/s^2



(a) Flight path “A” layout.

(b) Flight path “B” layout.



(c) Flight path “C” layout.

Figure 4.1: Malicious UAV flight paths A-C.

4.3 Data Collection

The data collected from the simulations is numerical in nature, mostly being floating point values representing the distance from the swarm UAV to its formation target. This data was analysed in Microsoft Excel. Excel can import data in a tabulated format from a text (.txt) file. Writing the data to a text file in C++ is relatively straight-forward but this is complicated by the expandable nature of the swarm UAV numbers in the simulation. Upon simulation start, a text file “output.txt” is created. The code originally had each swarm UAV open this file at each time period, then write the distance data before closing the file. This presents a problem as, with increasing numbers of UAVs making up the swarm, the overhead per time period would increase creating considerable lag on the system and simulation. A work-around was implemented that involved the creation of a static array similar to the array used to simulate communications between the swarm UAVs. The computational effort for storing the data in the array is considerably less than

Table 4.3: Time for malicious UAV to complete each path type.

Speed/Acceleration	Path A	Path B	Path C
21m/s / 5m/s^2	1,331 sec	1,344 sec	1,524 sec
23m/s / 5.5m/s^2	1,196 sec	1,230 sec	1,395 sec
26m/s / 6.5m/s^2	1,060 sec	1,093 sec	1,240 sec

opening, writing, and closing the text file. In order to store the data in the static array, the text write code is included in the destructor for the swarm class. This code loops through each index of the 2D array, and writes the data into the text file in a format appropriate for Excel to read. With batch processing the new data is amended to the end of the file, allowing it to only need to be opened once the simulation batch has finished.

Given the malicious UAV does not react to the swarm UAVs and moves to specific coordinates at fixed velocities and accelerations, it is possible to know the length of time that each path will take to simulate. Table 4.3 shows the simulation times to final "landing" point. The simulation was set to 1,800 seconds duration to ensure enough time elapsed to allow all UAVs to finalise their positions.

The median of the ten randomisation iterations are used rather than average in the final results. This is to allow for any instances where the swarm UAVs are unable to follow as close resulting in a variation difference. An example would be the simulation returning a distance of less than 1 meter for nine of the ten runs, but have one run end with a result of over 1000 meters, which would greatly skew the overall results. This did not happen often but was the cause of several inconsistencies within the initial dataset. Examples can be found in Section 5.2.

4.4 Methods Of Analysis

This project conducts its analysis through quantitative means based on original data collected from the simulation runs. Several runs of the same simulation parameters were conducted in order to verify the results of each battery of tests. The project involves three factors:

- Evaluation of swarm performance with increasing number of UAVs making up the swarm
- Performance of separate coordination algorithms
- Malicious UAVs pre-programmed flight path level of evasiveness

The first factor is tested from two to ten UAVs. Three coordination algorithms (follow, surround, and cone) are utilised. Of the pre-programmed flight path there are three different paths represented differing styles of evasiveness. These paths are pre-set and did not rely or react to the positioning of the swarm UAVs. These combined factors (illustrated in Figure 4.2) repeated through ten iterations to allow randomisation of swarm UAV starting positions will result in 2430 treatments, allowing for substantial analysis of the differences each independent factor has on the other dependent settings.

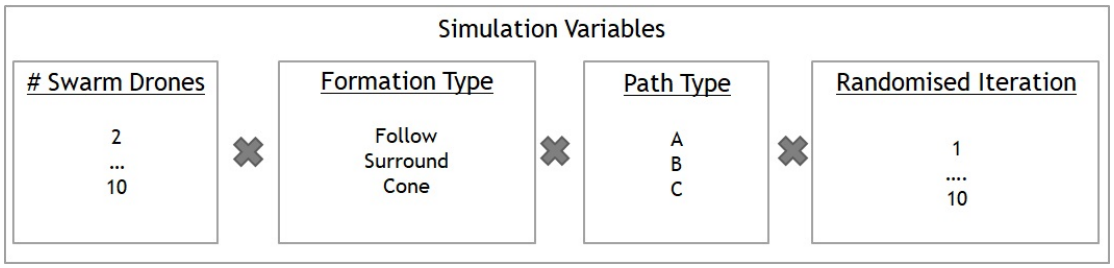


Figure 4.2: Layout of independent variables cross-analysed.

In order to achieve the objectives, this project sets a baseline. This standard will be either the successful pursuit of the malicious UAV to its landing site or the minimum distance of the swarm UAVs from the landing site. As the swarm UAVs will be aiming for target positions around the malicious UAVs when in swarm mode, the distance measured will be from the swarm UAVs to its swarm position around the malicious UAVs at the landing site. Figure 4.3 shows an example of a swarm of surround formation type that has fallen out of sensor range of the malicious UAV before it has landed. The final distance is the shortest distance of the 5 vectors (A- E) representing the swarm UAVs final position and their intended formation positions around the malicious UAV if they had successfully followed it to its final destination.

The original methodology proposed for determining the effectiveness of the swarms was to measure the length of time that the swarm was able to stay within sensor range of the malicious drone. As the project neared completion, this method presented problems regarding its ability to be used to compare the effectiveness of each differing variable change.

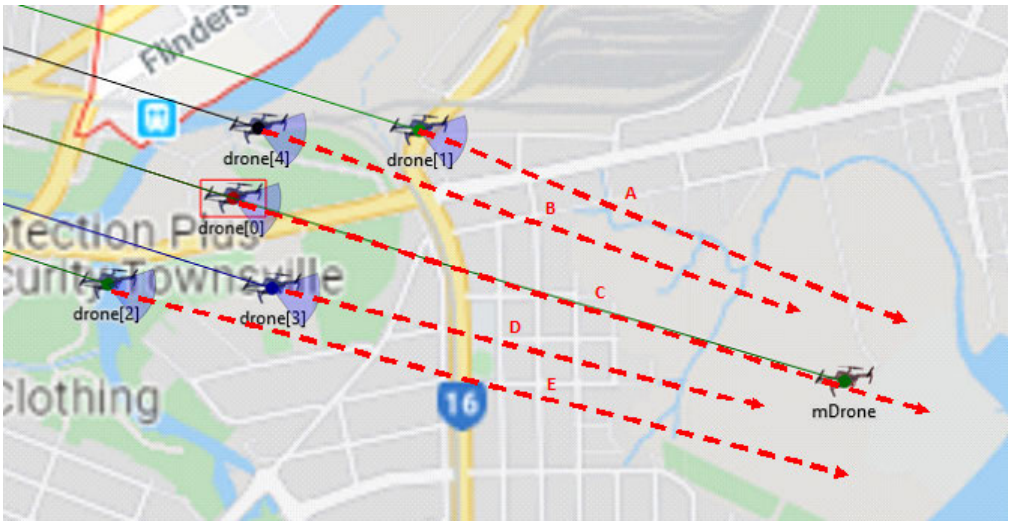


Figure 4.3: Example of dependent variable measurement.

The main issue with this method of measurement was that the automated placement of swarm UAVs changed with each increase of swarm drone numbers. Upon reflection it was noted that the main goal is to measure the effectiveness of the swarm in following the malicious drone back to its landing site. This meant that a measure of the distance from the malicious drones' final position to the closest swarm drone would better show the effectiveness of the swarm as a whole.

The resulting data is collated into graphs which plot the minimum distance of the swarm from their final targets (Y axis) against the number of UAVs in the swarm (X axis). As each iteration of swarm with increasing numbers will complete three different paths, the plot will contain a composite overlay of the results of each flight path simulation. Three composite plots are produced for each of the three different speed/acceleration levels (Levels 1-3). Each composite plot will be converted into an separate plot that takes the average of each flight path result for each swarm size. Final analysis is completed through the comparison of average plots that represent follow, cone, and surround data for each of the three flight characteristics levels (i.e. Levels 1, 2, & 3 as listed above). Figure 4.4 illustrates the process of information collation. Collation of data in this manner allows direct visual and numerical comparison of formation effects on both differing UAV flight paths and flight characteristics.

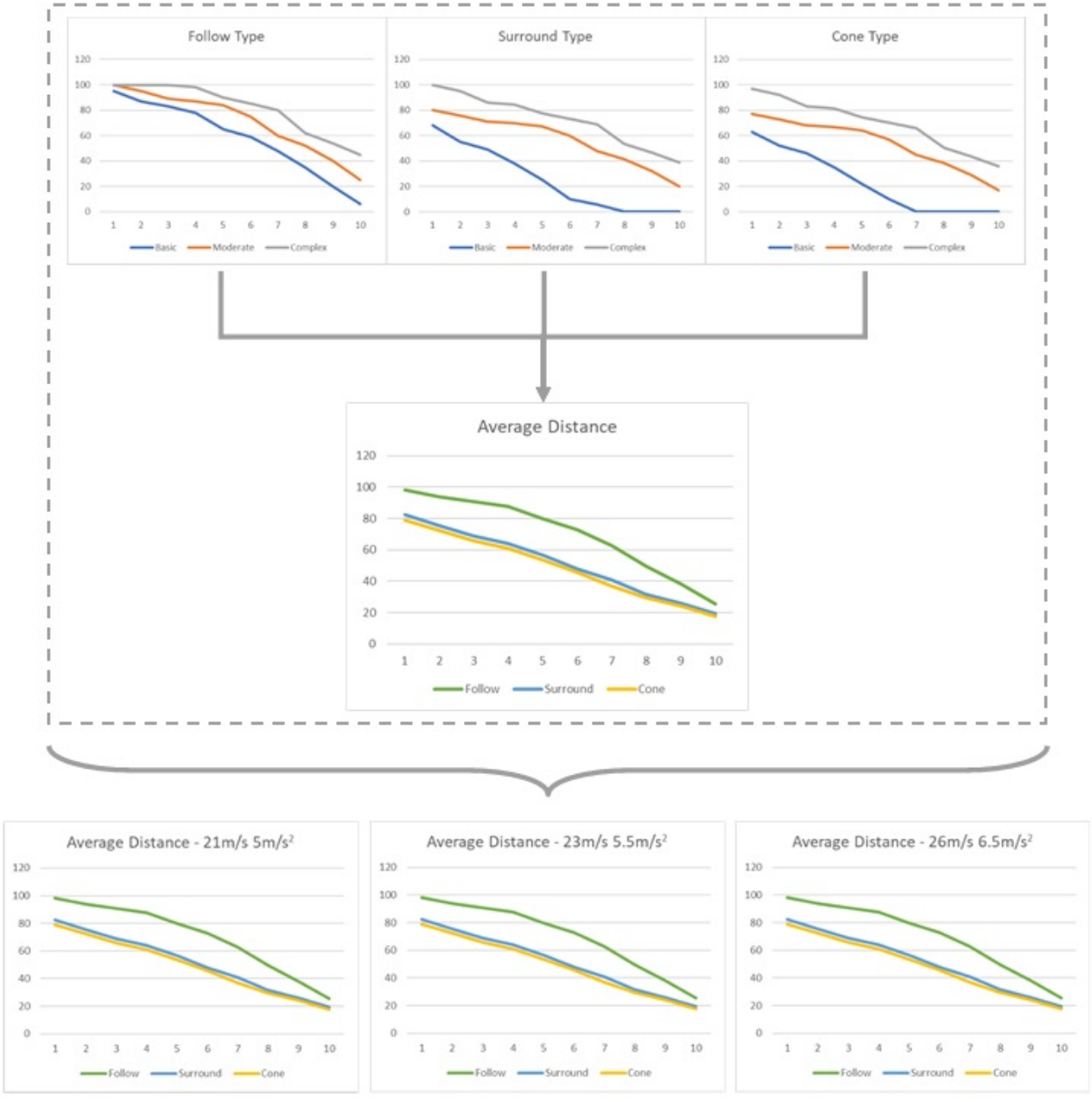


Figure 4.4: Flow of results graphed for visual analysis.

4.5 Methodology Justification

Though usually conducted with real-world experiments, similar research into individual or swarm UAVs analysing tracking or path finding methods have relied on numerical analysis to determine the effectiveness of their research. Research into similar areas of UAV pursuit and encirclement has been conducted as shown in the Literature Review chapter. Though some performed their experimental tests on real-world equipment, the typical numerical analysis used distance as the primary measure of success. The research in Hafez, Iskandarani, Givigi, Yousefi, Rabbath & Beaulieu (2014), Hafez, Givigi, Schwartz, Yousefi & Iskandarani (2015) and Hafez, Marasco, Givigi, Iskandarani, Yousefi & Rabbath (2015) all use radial distance from the target as a measure of their ability to achieve encirclement of the target.

Chapter 5

Results & Analysis

5.1 Research Variables

This dissertation aimed to determine the effectiveness of multiple UAVs working in concert to pursue a moving target. The independent variables within the simulation were the swarm formation, the number of UAVs within the swarm, and the malicious UAV flight characteristics. The malicious UAV flight characteristics are made up of its velocity, lateral acceleration, and path type. The monitored dependent variable is the distance from the closest swarm UAV to its formation target point around the malicious UAV at the final landing point of the simulation. All of the described variables are of the continuous type. The swarm UAV flight characteristics (speed and lateral acceleration) are also factors that would influence the outcome of the simulations. These are controlled for by fixing the speed and acceleration to be the same value for all simulations. Another independent variable requiring control is the initial placement of the swarm UAVs, which are controlled through the use of a randomisation algorithm and multiple simulation runs to average out the effect of UAV placement.

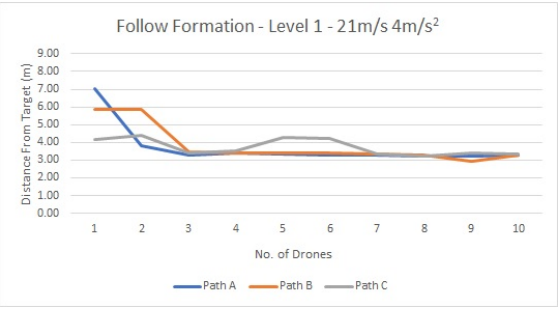
5.2 Individual Path Comparison Results

This section aims to provide in-depth analysis of the values from the median result of the ten randomised iterations, comparing the effects of the different malicious UAV paths on each formation type. As noted in Section 4.3, the median results are used rather than the

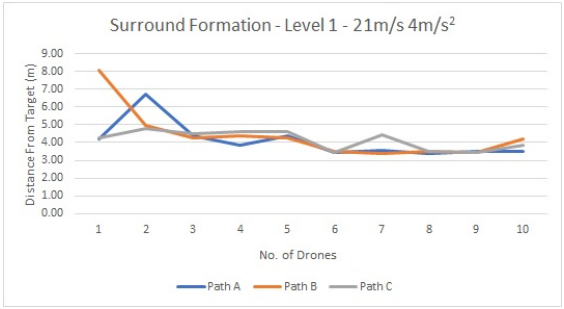
mean as any iteration of a swarm that loses sight of the malicious UAV at (for example) 1005 meters would give an output final distance several orders of magnitude larger than a swarm that lasted another 6 meters which, due to the malicious UAVs landing position being within the 1000 meter sensor range, is then able to track the UAV to sub-10 meter distance. This is represented in Figure 5.2 where the results for swarms utilising the cone formation and following a malicious UAV in Path B configuration had a sudden step from 1129.2 meters to 4.7 meters final distance between swarm sizes of 2 and 3 UAVs respectively.

5.2.1 Level 1 - 21m/s 4m/s

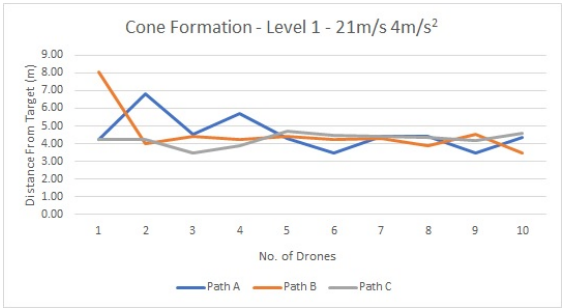
Though the results for malicious UAV travelling at 21 m/s had a downwards trend of final distance, all formation types were within 10 meters of the final target. Results this close to the target can be classed as having reached the final target position as it is within 100 meters of the final distance. As all formations and swarm compositions reached their destination, there is no clear data at this flight characteristic level to show any advantages of swarm formations when pursuing a malicious UAV.



(a) Follow formation results.



(b) Surround formation results.



(c) Cone formation results.

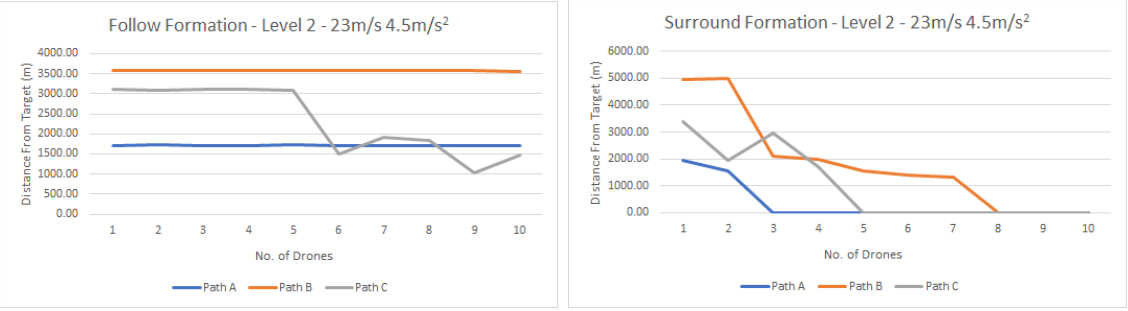
Figure 5.1: Comparison of individual formation results for Level 1.

Table 5.1: Level 1 results for Paths A, B, and C, and the average for each formation type.

	Follow				Surround				Cone			
	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)
1	7.03	5.88	4.19	5.70	4.20	8.07	4.23	5.50	4.22	8.06	4.24	5.51
2	3.81	5.87	4.38	4.68	6.69	4.96	4.79	5.48	6.82	4.00	4.22	5.01
3	3.30	3.49	3.38	3.39	4.37	4.24	4.50	4.37	4.52	4.39	3.49	4.13
4	3.41	3.38	3.54	3.44	3.88	4.36	4.61	4.28	5.70	4.22	3.86	4.59
5	3.32	3.43	4.29	3.68	4.38	4.24	4.64	4.42	4.29	4.39	4.69	4.46
6	3.27	3.40	4.20	3.62	3.45	3.48	3.43	3.45	3.45	4.26	4.45	4.05
7	3.26	3.32	3.35	3.31	3.55	3.41	4.42	3.79	4.40	4.27	4.43	4.37
8	3.23	3.31	3.20	3.25	3.38	3.48	3.52	3.46	4.40	3.86	4.34	4.20
9	3.24	2.92	3.37	3.18	3.50	3.47	3.42	3.46	3.47	4.52	4.16	4.05
10	3.27	3.28	3.36	3.30	3.48	4.21	3.85	3.84	4.34	3.45	4.60	4.13

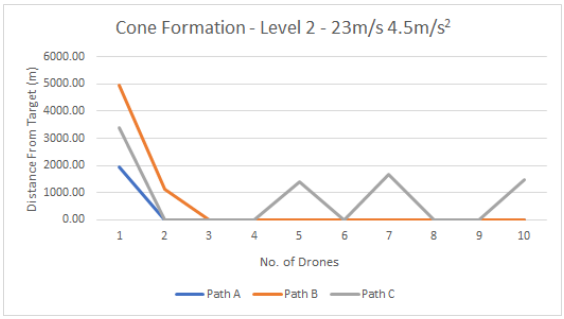
5.2.2 Level 2 - 23m/s 4.5m/s

The baseline follow-type formation had some interesting artefacts present over the span of swarm UAV composition numbers. The resulting drop-off points for both A and B path types stay relatively consistent at 1725 m and 3591 m respectively for all 10 iterations of swarm size. Path C results in an erratic decreasing of final distance. This seems to be due to the path having several long sections which allowed for the malicious UAV to outrun smaller swarm sizes. The cone formation had artefacts present for swarm of sizes 5,7, and 10 (Figure 5.2c). This was due to inconsistencies in the data that came from the results of Path C, as shown in Table 5.2. These inconsistencies were found to be a result of the cone formation placement algorithm which assigns the swarm UAVs placements based on the number of UAVs in the swarm. Uneven numbers are placed with a preference to the far sides of the cone, whilst even numbers place UAVs with a preference to the direct front. This, alongside the fact the swarm UAVs have a sensor radius of 1000 meters, means that the swarm UAVs of larger odd swarm sizes were placed in a position that put them just out of range of 1000 meters from the final landing site of the malicious UAV, hence coming to a stop at 1390 - 1660 meters.



(a) Follow formation results.

(b) Surround formation results.



(c) Cone formation results.

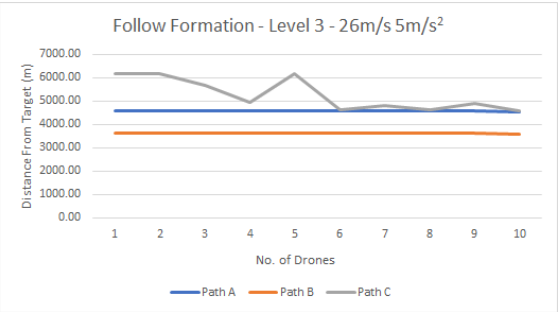
Figure 5.2: Comparison of individual formation results for Level 2.

Table 5.2: Level 2 results for Paths A, B, and C, and the average for each formation type.

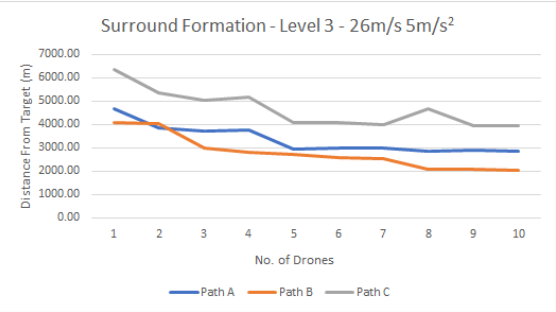
	Follow				Surround				Cone			
	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)
1	1725.5	3591.1	3100.3	2805.6	1956.8	4966.1	3404.5	3442.5	1956.8	4965.9	3404.5	3442.4
2	1725.5	3591.1	3083.2	2799.9	1539.9	4982.7	1930.4	2817.7	4.3	1129.2	4.2	379.2
3	1725.5	3591.1	3100.4	2805.6	3.5	2119.9	2976.2	1699.8	4.1	4.7	4.4	4.4
4	1725.5	3592.1	3100.4	2806.0	4.4	1980.5	1712.1	1232.4	4.2	4.2	3.5	4.0
5	1725.5	3591.1	3092.4	2803.0	3.4	1564.7	4.3	524.1	4.6	3.9	1391.9	466.8
6	1725.5	3590.8	1496.4	2270.9	4.4	1393.7	3.4	467.2	4.2	4.3	4.3	4.3
7	1725.5	3591.1	1909.1	2408.5	4.3	1311.0	3.5	439.6	3.8	3.9	1657.6	555.1
8	1725.5	3588.0	1835.4	2383.0	3.6	1.7	4.2	3.2	3.5	4.2	0.0	2.6
9	1725.5	3591.0	1042.2	2119.6	3.5	3.5	4.2	3.7	3.5	4.3	5.9	4.6
10	1725.5	3581.1	1461.6	2256.1	3.6	3.9	4.3	3.9	6.0	5.7	1487.7	499.8

5.2.3 Level 3 - 26m/s 5m/s

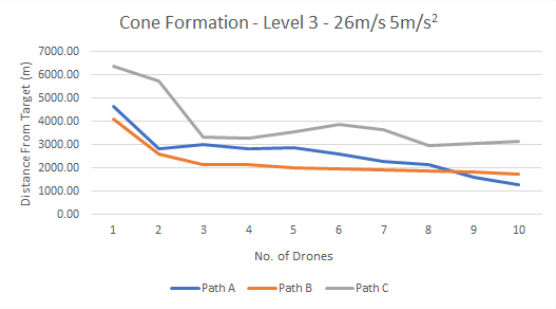
Table 5.3 outlines the individual results for each swarm size for Level 3 simulations. For all formation types Path B was the easiest path to track (Figure 5.3). The follow-type formation had steady results for Paths A and B, with a general trend downwards for Path C for increasing swarm size. The surround formation (Figure 5.3b) showed a preference for Path A over C, with a small downwards trend for all three paths. The cone formation had the greatest trend downwards and final distance for increasing swarm size, showing that it benefits more from increased number of UAVs in forward positions over the surround formation.



(a) Follow formation results.



(b) Surround formation results.



(c) Cone formation results.

Figure 5.3: Comparison of individual formation results for Level 3.

Table 5.3: Level 3 results for Paths A, B, and C, and the average for each formation type.

	Follow				Surround				Cone			
No. UAVs	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)	A (m)	B (m)	C (m)	Avg (m)
1	4578.2	3628.9	6174.5	4793.9	4665.9	4088.5	6375.6	5043.3	4665.9	4087.4	6375.6	5043.0
2	4578.2	3628.9	6174.3	4793.8	3838.3	4014.3	5371.2	4407.9	2824.1	2582.0	5728.4	3711.5
3	4578.2	3628.1	5671.3	4625.9	3732.9	2994.1	5023.4	3916.8	2998.9	2160.0	3344.5	2834.5
4	4578.2	3627.8	4947.2	4384.4	3779.8	2812.8	5154.1	3915.6	2830.7	2140.4	3274.4	2748.5
5	4578.2	3628.1	6167.8	4791.4	2939.8	2702.1	4094.2	3245.4	2862.3	2010.7	3553.2	2808.7
6	4578.1	3627.2	4618.2	4274.5	2966.4	2593.8	4059.3	3206.5	2578.8	1979.2	3880.7	2812.9
7	4578.0	3627.0	4810.6	4338.5	2981.5	2533.9	3980.0	3165.1	2293.1	1893.7	3636.2	2607.7
8	4578.1	3625.6	4606.8	4270.2	2832.1	2062.8	4648.5	3181.1	2149.4	1851.9	2959.9	2320.4
9	4578.1	3626.6	4896.5	4367.1	2893.7	2064.7	3947.9	2968.8	1592.0	1806.7	3056.5	2151.8
10	4575.9	3622.2	4592.2	4263.5	2869.6	2036.9	3932.1	2946.2	1265.1	1737.9	3157.8	2053.6

5.3 Amalgamated Results

Figure 5.4 - 5.6 shows the amalgamated results that compare the different swarm formations to each other for Levels 1, 2, and 3. These plots are based on the UAV with the minimum final distance from the malicious UAV at the end of the simulation. As there was a complete set of simulations carried out before randomisation code was introduced, both graphs from single- and multi-iteration (left and right graphs respectively) are added to compare and validate the final results.

5.3.1 Statistical Analysis

Table 5.4 outlines the mean final distances and their standard deviation (SD) for each formation and flight characteristic level. As noted previously, all formation types successfully pursued the malicious UAV for Level 1. Level 2 resulted in follow formation mean distance of 2545 meters with a small SD of 282 meters, surround formation mean distance of 1063 meters with a large 1228 meter SD, and cone formation mean distance of 536 meters with a SD of 1048 meters. The high SD present in the surround and cone formations for Level 2 is derived from the smaller swarm sizes having a significantly larger distance to the target point than swarm sizes over 5 UAVs. Level 3 presented follow formation mean

distance as 4490 meters with a 233 meter SD, surround formation 3600 meters and SD of 700 meters, and cone formation presenting 2909 meters mean distance with 881 meters SD. Level 3 had smaller SD than Level 2 due to a more consistent downwards trend per increase in swarm size. Both Level 2 and Level 3 results show a clear improvement in results over the follow formation type, though the standard deviation of both coordinated formations are much higher than the follow-type formation.

The higher standard deviation indicates that, though the coordinated swarm types are better than uncoordinated on average, the results have a wider range than the follow type. This was initially seen as being a result of the nature of the final distance being used as the dependent variable. As noted in Section 5.2, the results for a swarm that was less than 1000 meters (the sensor distance) from the malicious UAV when the malicious UAV reached its final position would be sub-10s of meters. The results for anything over 1000 meters would mean the swarm would lose sight before the malicious UAV stopped and therefore not be able to cover the final distance to the formation points. This is why the median of the iteration results were used to ensure outliers would not cause large shifts in the overall results.

In an attempt to account for this anomaly the data was put through a simple `if-then-else` filter that reduced any value over 1000 meters by 1000 meters, and the statistical analysis run again (Table 5.5). As expected, no change occurred for the Level 1 formations as all simulations successfully pursued the target to the final position. There was no change in SD for the follow-type formation against Level 2 flight characteristics as the 1000 meter reduction is applied across all iterations and swarm sizes. The main effect was with surround and cone formations, where the reduction of SD was slightly larger than the reduction in mean final distance for surround type, and significantly larger for cone type formation. This shows that for the cone formation with Level 2 parameters the sensor cut-off had a larger effect than for the surround formation method. All three formations were reduced by the 1000 meters as no simulations successfully pursued the malicious UAV to the final point, meaning there were no sensor cut-off issues.

Table 5.4: Mean and standard deviation statistics.

FC Level	Formation Type	Mean Final Distance	SD	% SD
1	Follow	3.76 m	0.809	22%
	Surround	4.21 m	0.771	11%
	Cone	4.45 m	0.477	5%
2	Follow	2545.82 m	282.840	18%
	Surround	1063.41 m	1228.184	115%
	Cone	536.32 m	1048.469	19%
3	Follow	4490.30 m	233.150	11%
	Surround	3599.67 m	699.765	195%
	Cone	2909.25 m	881.334	30%

Table 5.5: Mean and standard deviation statistics for anomaly-cancelled data.

FC Level	Formation Type	Mean Final Distance	SD	%SD	Δ Mean Final Distance	Δ SD	Δ %SD
1	Follow	3.76 m	0.81	22%	0.00 m	0.00	0%
	Surround	4.21 m	0.77	18%	0.00 m	0.00	0%
	Cone	4.45 m	0.48	11%	0.00 m	0.00	0%
2	Follow	1545.82 m	282.84	18%	-1000.00 m	0.00	7%
	Surround	663.41 m	823.17	124%	-400.00 m	-405.01	9%
	Cone	436.32 m	743.97	171%	-100.00 m	-304.50	-25%
3	Follow	3490.30 m	233.15	7%	-1000.00 m	0.00	1%
	Surround	2599.67 m	699.77	27%	-1000.00 m	0.00	7%
	Cone	1909.25 m	881.33	46%	-1000.00 m	0.00	16%

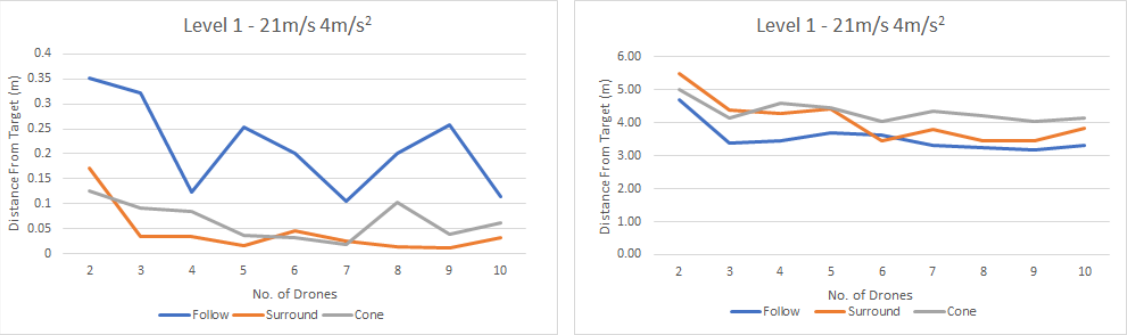
5.3.2 Formation Comparison

This sections outlines specific comparison of each formation type through analysis of the mean results for each flight characteristic level.

Level 1

Figure 5.4 (a) and (b) shows the results of Level 1 simulations for both pre- and post-randomisation implementation. Due to the method by which the simulator determines the closest point at the end of the simulation, the results of the randomisation are larger than pre-randomisation. This is to be expected as the pre-randomisation code took the minimum distance at the end of the simulation such that the UAVs crossing over the target point would produce sub-meter distances. However the post-randomisation code slowed the swarm UAVs to a stop upon hitting the stationary targets, resulting in an overrun of up to 6 meters. Nevertheless, considering the results typical of Level 2 and 3 and the sensor range of the UAVs (1000 meters), it can be concluded that any distance sub- 100 meters is a successful pursuit of the malicious UAV to its landing point.

All types of formations (including the base 'follow' case) successfully pursued the target to the landing point. As the results are an amalgamation of the three different Path types, this shows that the swarm UAVs in any configuration can pursue a malicious UAV of slightly better flight characteristics.



(a) Single-run simulation results.

(b) Multi-run randomised simulation results.

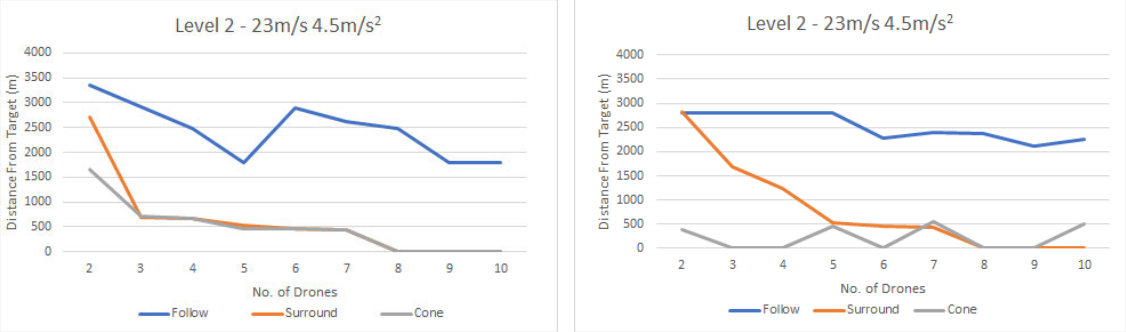
Figure 5.4: Level 1 formation result comparison.

Level 2

Figure 5.5 (a) and (b) details the final results pre- and post-randomisation for Level 2. The post-randomisation data sets a more consistent level for the base follow case, though not without artefacts present with the cone formation. Regardless, both the surround and cone formations show significant pursuit abilities over the base follow formation. For the surround formation the number of UAVs within the swarm has a direct effect on

the ability pursue the malicious UAV, with significant gains found with 5 UAVs or more within the swarm significantly reducing the final distance.

The cone formation had better results as, due to the layout of the formation, more UAVs were placed ahead and to the far left and right of the malicious UAV. This resulted in an increase in reaction time for the swarm as a whole, with the passing of target information allowing swarm UAVs that fall out of sensor range to still be able to catch up and get back into position.



(a) Single-run simulation results.

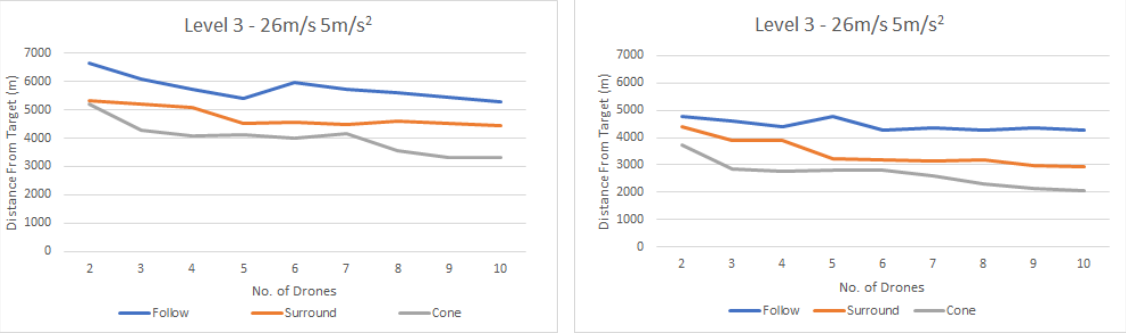
(b) Multi-run randomised simulation results.

Figure 5.5: Level 2 formation result comparison.

Level 3

Application of Level 3 flight characteristics to the malicious UAV represented a significant superiority of speed and acceleration over the swarm UAVs. Level 3 had consistent trends between the pre- and post-randomisation results, though the averaging of the ten random iterations led to the distances being lower than the single-run simulations. Though no formation was able to successfully pursue the malicious UAV to the final resting point, both the surround and cone formations performed consistently better than the baseline follow formation for all swarm sizes. The cone formation was able to track the malicious UAV to within 2000 meters of the final target with the maximum swarm size of 10 UAVs, with the surround formation being able to track within 3000 meters. Increases in swarm size had diminishing returns after 5 UAVs for surround formation, and was showing a slope returning to level values for the cone formation, indicating that an increase in swarm size would likely not increase the overall ability of the swarm to track a malicious UAV of such superior flight characteristics. Despite this, both cone and surround formations showed significant increases in tracking ability over both the non-collaborative follow

formation (51.8% and 30.9% decrease in distance respectively) as well as swarms of the same formation type but minimum size (44.7% and 33.2% distance decrease respectively). Even though neither cooperative formations were successful in tracing the malicious UAV to the final point, the ability to reduce the range to the final location by up to 52% shows significant reductions in the total distance that would be required to search for the UAV by other means. This extra time in pursuit may also allow time for long-distance and better-equipped equipment (e.g. fixed wing UAVs) to be deployed from further afield which would take over before the swarms battery capacity or communication range is reached.



(a) Single-run simulation results.

(b) Multi-run randomised simulation results.

Figure 5.6: Level 3 formation result comparison.

5.3.3 Target Optimisation Effects

A noticeable effect of the coding of the formations was the lack of flexibility for the swarm to optimise their targetted positions. As detailed in Chapter 3, a simple efficiency algorithm was implemented that allowed swarm UAVs to better choose the target position best for the individual UAV, though not without compromises. This algorithm was implemented and showed promise in how the swarm UAVs, within the confines of the algorithm given, optimised their target preference to reduce the distance they were required to travel to reach their target. Time constraints resulted in this algorithm only being implemented for the surround formation.

Subsequent simulation runs were conducted with the randomisation algorithm in place. Though presenting in the GUI simulations as visually more efficient, the datasets represented in Figures 5.7 - 5.9 show that there were no significant and consistent gains made over the non-modified surround formation. This is likely due to the compromise

discussed in Chapter 3 where position node assignment priority is based on swarm UAV ID. The algorithm could be made more efficient within the confines of the coding restrictions through the use of a priority system that has the first UAV determine the smallest combined distance and assigns position node IDs to each swarm UAV at the start of each time period. This would represent a shift from a decentralised swarm (decisions are made individually) to a centralised swarm (some decisions are made by a central node).

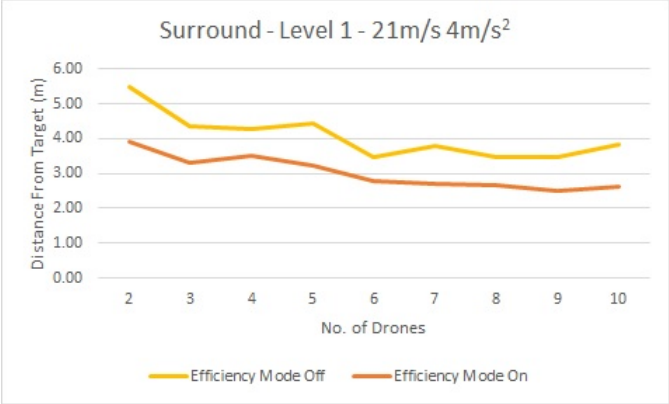


Figure 5.7: Level 1 surround efficiency results.

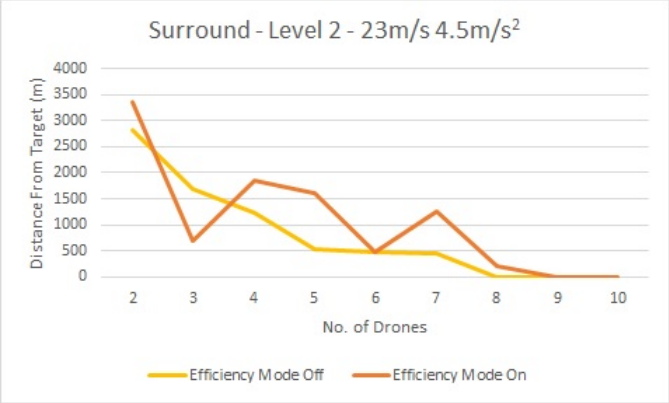


Figure 5.8: Level 2 surround efficiency results.

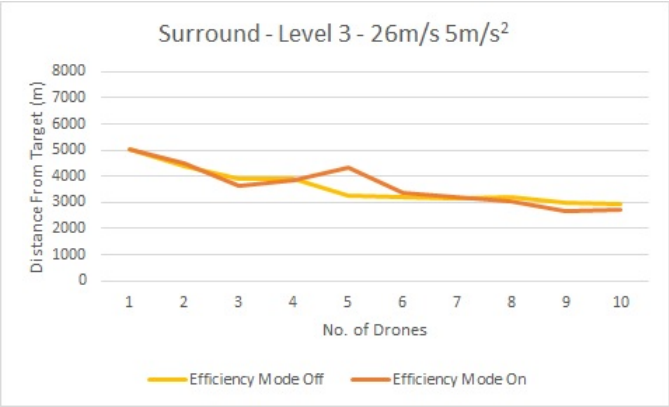


Figure 5.9: Level 3 surround efficiency results.

5.3.4 Interesting Observations

An overall observation of the three paths defined for the malicious UAV to follow (Paths A, B, & C) showed that, for the "search pattern" path (Path A), the swarm fell out of sensor range when the malicious UAV was travelling at 26 m/s but, due to the nature of the path, was picked back up when the malicious UAV came back around. Another interesting observation common to all three paths were that increasing the speed of the malicious UAV did not specifically mean the swarm UAVs would lose the malicious UAV earlier as, because the swarm UAVs had fallen behind (but not out of range) they were in a more advantageous position when the malicious UAV changed directions than if they were closer to it.

Chapter 6

Conclusions and Further Work

6.1 Conclusions

This project aimed to distinguish between different methods of swarm UAV coordination and their effectiveness in pursuing a target UAV. These results would verify if UAV swarms can provide redundancy and greater group agility than traditional ground-based radar systems and expensive individual military-grade UAV solutions. In order to have sufficient data to distinguish between the different swarm formations and compositions, multiple simulations were conducted against a malicious UAV of increasing flight characteristics. Three different levels of flight agility, alongside three differing flight paths, were used to test the overall effectiveness of the swarm.

The original methodology proposed for determining the effectiveness of the swarms was to measure the length of time that the swarm was able to stay within sensor range of the malicious UAV. As discussed, this proved untenable as a method of comparison between the different independent variables. Instead, the distance from the final position of the malicious UAV and the closest swarm UAV was chosen as a more appropriate method of determining swarm effectiveness. Factors that would have an affect the simulation results but which were not part of this projects scope or were expected to have a linear effect through all types were set to a fixed value for all simulations. This included the velocity and lateral acceleration of the swarm UAVs, which were set to 20 m/s and 4 m/s^2. The use of randomisation allowed the effect of initial swarm placement to be averaged out through ten iterations of each simulation run. The use of random iterations with the

independent variables resulted in 2430 simulation runs being performed for this project.

The quantitative analysis of the final distances from the malicious UAV having completed differing flight paths with increasing levels of agility (both speed and handling) showed that both surround and cone formations outperformed a follow-type formation when pursuing malicious UAVs operating with flight characteristics far greater than the individual swarm UAVs ability. Swarm sizes over 5 UAVs were able to successfully pursue the malicious UAV for Level 2 conditions which was operating at a 7.2 km/h velocity and 0.5 m/s^2 lateral acceleration advantage over the swarm UAVs. Though not able to successfully pursue the malicious UAV to the final point, Level 3 results still showed significant decreases in the final distance for both formation types over the stock follow-type. The cone formation showed a 51.8% decrease in distance over the baseline for the maximum swarm size, and a 44.7% decrease in distance for a swarm of 2 UAVs in cone formation mode. The surround formation saw a smaller distance decrease of 30.9% over baseline, and 33.2% when compared to a 2-UAV swarm in the same surround mode. These decreases in distance are due to the formations overall increased ability to stay within sensor range, effectively increasing the time spent in pursuit and maximising the swarms' ability to follow the malicious UAV to its final landing place. Even when not able to track the malicious UAV back to the target point, the ability to stay in pursuit longer reduces the final distance - hence allowing for a reduction in the search radius required to locate the landing point of the UAV by other means - whilst also allowing more time for other methods of tracking to be deployed. It can be concluded from the results presented that use of either a surround or cone method as described increases the overall flight characteristics of the swarm, resulting in a group ability that is greater than the sum of its parts. The use of cone formation presented the best overall results, though the final distance results in combination with the lower computing resources necessary show there is a place for the surround method in real-world applications.

This research has provided insight into cost-effective methods of asset protection through the use of swarm UAVs, whilst developing a foundation from which other related swarm research can be conducted. This project has enhanced the practical applications of UAV swarms through the development of two formation algorithms that can be easily adapted within the base code of user-built UAVs. Applications within the research community have been created through the development of bespoke modular code within an open-source simulation environment that allows for further development and testing of swarm and

individual UAV-related projects and practical work. As noted in Chapter 1, the design work, data, and results from this project has been used in a paper that was subsequently accepted through the peer-review process for presentation at the IEEE UEMCON 2020 conference.

6.2 Further Work

From the outset of this project there have been many topics and lines of inquiry that lend themselves to further beneficial research for which this research can be used as a base. Many areas relating to more efficient methods of tracking a malicious UAV can be applied to determine if an increase in pursuit ability is achieved. As shown in the literature review, these methods include (but are not limited by) Proportional Navigation, centralised and decentralised networks, and advanced path finding.

Efforts to adapt the swarm formation algorithm to be more efficient in the allocation of swarm UAV positions would be expected to result in significant overall gains to the abilities of the swarm. Different methods of approach for this problem (and resulting analysis) could range from a straight-coded function native to each UAV to application of machine learning algorithms. As this would require the use of communication within the swarm (to broadcast each UAVs current and target positions) analysis of the effects of different communication methods (e.g. radio, LI-FI, etc.) on the proposed algorithms could also be conducted.

As noted in the assumptions and rationale section of the methodology chapter, terrain and path finding were not enabled for these simulations. This was done to reduce overall complexity of the project, though the ability to add these features are already supported by the INET framework. This could be coupled with transmission characteristics supported by the overall OMNeT++ framework to analyse the effects of different terrain scenarios on the ability for the swarm to pursue a target. Possible scenarios could include the effects of operating in an area containing hills, buildings, etc. These test-bed scenarios could also provide research focuses on determining what effects different swarm compositions (e.g. centralised versus decentralised) would have on the swarms ability to operate within those environments.

Other types of formations could be checked with the results from this project used as

a base of comparison. Modifying the surround formation to bias the circle forward into an ellipse as in Figure 6.1 or utilising mini-swarms shown in Figure 6.2 may increase the overall abilities of the swarm.

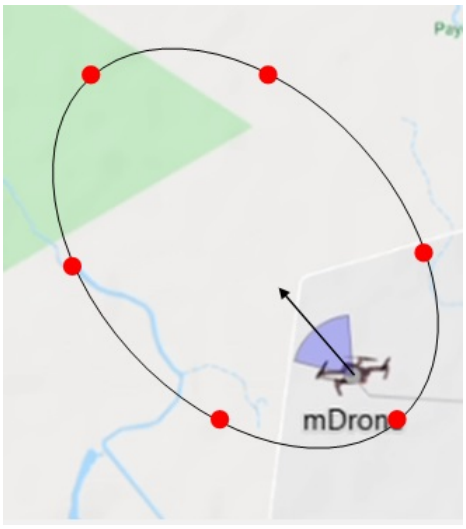


Figure 6.1: Modified surround formation.

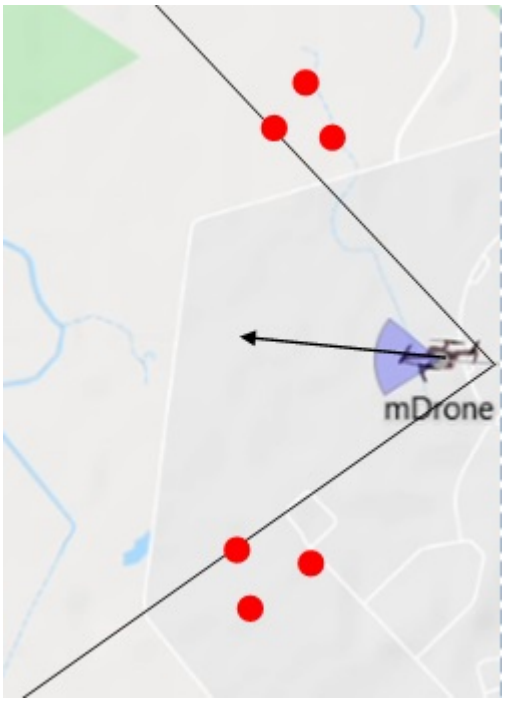


Figure 6.2: Mini-swarm formation.

Other limitations of this project could be lifted and researched, such as moving the research focus of this project from the horizontal 2D plane to a full 3D simulation. This is supported within the INET framework, with the ability to work with the INET or OMNeT OSG (Open Scene Graph) modules (each are used slightly differently) would provide a realistic environment to simulate, especially if coupled with the previously mentioned

terrain code. Analysis for this type of simulation could be made for the effects of novel malicious UAV flight paths including operating at relatively high altitudes.

As UAVs are heavily dependent on their battery capacity in determining the time they can stay in pursuit, research into whether increases in complexity of formation (i.e. processor time required to calculate positions & therefore increased battery usage) negates the advantages of specific swarm formations such as the cone formation recommended above can be conducted. Further research on this line would include the use of a centralised topology for the swarm where one UAV conducts the calculations and broadcasts to the swarm.

Beyond the simulation realm, the formations within this research and that proposed as further research needs to be applied in the real world to judge their effectiveness when presented with outside forces. Application within real UAVs will necessitate more processor-efficient code to ensure that the benefits of the algorithms developed are not outweighed by the subsequent shortening of flight time through increased battery drain.

References

Akram, R. N., Markantonakis, K., Mayes, K., Habachi, O., Sauveron, D., Steyven, A. & Chaumette, S. (2017), Security, privacy and safety evaluation of dynamic and static fleets of drones, *in* '2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)', IEEE, pp. 1–12.

Autel (2020), 'Evo'. Viewed: 5 September 2020.
URL: *https://auteldrones.com/products/evo*

BBC (2017), 'Tiny drone lands on queen elizabeth aircraft carrier'. Viewed: 10 October 2019.
URL: *https://www.bbc.com/news/uk-scotland-highlands-islands-40910087*

BBC (2019), 'Gatwick airport police "not prepared for two drones"'. Viewed: 9 October 2020.
URL: *https://www.bbc.com/news/uk-england-sussex-48929442*

Belkhouche, F., Belkhouche, B. & Rastgoufard, P. (2006), 'Line of sight robot navigation toward a moving goal', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **36**(2), 255–267.

Boussios, E. G. (2014), 'The proliferation of drones: A new and deadly arms race', *Journal of Applied Security Research* **9**(4), 387–392.

Cevik, P., Kocaman, I., Akgul, A. S. & Akca, B. (2013), 'The small and silent force multiplier: a swarm uav—electronic attack', *Journal of Intelligent & Robotic Systems* **70**(1-4), 595–608.

Cui, Q., Liu, P., Wang, J. & Yu, J. (2017), Brief analysis of drone swarms communication, *in* '2017 IEEE International Conference on Unmanned Systems (ICUS)', IEEE, pp. 463–466.

Dedousis, D. & Kalogeraki, V. (2018), A framework for programming a swarm of uavs, *in* 'Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference', pp. 5–12.

DJI (2020), 'Dji - the world leader in camera drones/quadcopters for aerial photography'. Viewed: 5 September 2020.
URL: *https://www.dji.com/au*

Ehrhard, T. P. (2010), Air force uavs: The secret history, Technical report, Mitchell Inst for Airpower Studies Arlington VA.

Forbes (2019), "drone swarm' invaded palo verde nuclear power plant last september — twice'. Viewed: 10 August 2019.
URL: *https://www.forbes.com/sites/davidhambling/2020/07/30/drone-swarm-invaded-palo-verde-nuclear-power-plant/3940307a43de*

Fotouhi, A., Ding, M. & Hassan, M. (2017*a*), 'Dronecells: Improving 5g spectral efficiency using drone-mounted flying base stations', *arXiv preprint arXiv:1707.02041* .

Fotouhi, A., Ding, M. & Hassan, M. (2017*b*), Understanding autonomous drone maneuverability for internet of things applications, *in* '2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)', IEEE, pp. 1–6.

Fu, X., Feng, H. & Gao, X. (2012), 'Uav mobile ground target pursuit algorithm', *Journal of Intelligent & Robotic Systems* **68**(3-4), 359–371.

George, J. & Ghose, D. (2009), A reactive inverse pn algorithm for collision avoidance among multiple unmanned aerial vehicles, *in* '2009 American Control Conference', IEEE, pp. 3890–3895.

Guardian (2019*a*), 'Easyjet says gatwick drone chaos cost it £15m'. Viewed: 20 September 2020.
URL: *https://www.theguardian.com/business/2019/jan/22/easyjet-gatwick-drone-cost-brexit-flights*

Guardian (2019*b*), 'Gatwick drone disruption cost airport just £1.4m'. Viewed: 20 September 2020.
URL: *https://www.theguardian.com/uk-news/2019/jun/18/gatwick-drone-disruption-cost-airport-just-14m*

Hafez, A. T., Givigi, S. N., Schwartz, H. M., Yousefi, S. & Iskandarani, M. (2015), Real time tactic switching for multiple cooperative uavs via model predictive control, *in* '2015 Annual IEEE Systems Conference (SysCon) Proceedings', IEEE, pp. 432–438.

Hafez, A. T., Iskandarani, M., Givigi, S. N., Yousefi, S., Rabbath, C. A. & Beaulieu, A. (2014), Using linear model predictive control via feedback linearization for dynamic encirclement, *in* '2014 American Control Conference', IEEE, pp. 3868–3873.

Hafez, A. T., Marasco, A. J., Givigi, S. N., Iskandarani, M., Yousefi, S. & Rabbath, C. A. (2015), 'Solving multi-uav dynamic encirclement via model predictive control', *IEEE Transactions on control systems technology* **23**(6), 2251–2265.

Han, S.-C. & Bang, H. (2004), Proportional navigation-based optimal collision avoidance for uavs, *in* '2nd International Conference on Autonomous Robots and Agents', pp. 13–15.

Hooda, D. & Raich, V. (2015), *Fuzzy Set Theory and Fuzzy Controller*, Alpha Science International.

Huang, G.-S., Tung, C.-K. & Ciou, J.-C. (2009), To achieve the path planning of mobile robot for a correct destination and direction using fuzzy theory, *in* '2009 IEEE International Symposium on Industrial Electronics', IEEE, pp. 1737–1742.

Jung, B. & Sukhatme, G. S. (2002), 'Tracking targets using multiple robots: The effect of environment occlusion', *Autonomous robots* **13**(3), 191–205.

Kakar, J. A. (2015), UAV communications: Spectral requirements, MAV and SUAV channel modeling, OFDM waveform parameters, performance and spectrum management, PhD thesis, Virginia Tech.

Lee, G., Chong, N. Y. & Christensen, H. (2010), 'Tracking multiple moving targets with swarms of mobile robots', *Intelligent Service Robotics* **3**(2), 61–72.

Malik, D. S. (2014), *C++ programming: Program design including data structures*, Nelson Education.

Marasco, A. J., Givigi, S. N. & Rabbath, C. A. (2012), Model predictive control for the dynamic encirclement of a target, *in* '2012 American Control Conference (ACC)', IEEE, pp. 2004–2009.

Ma'Sum, M. A., Jati, G., Arrofi, M. K., Wibowo, A., Mursanto, P. & Jatmiko, W. (2013), Autonomous quadcopter swarm robots for object localization and tracking, *in* 'MHS2013', IEEE, pp. 1–6.

McCune, R. R. & Madey, G. R. (2013), 'Swarm control of uavs for cooperative hunting with dddas', *Procedia Computer Science* **18**, 2537–2544.

NPR (2019), 'Attack on saudi oil facilities makes oil prices spike'. Viewed: 20 September 2020.
URL: *https://www.npr.org/2019/09/16/761118726/oil-prices-jump-following-drone-attack-on-saudi-oil-facility*

Parrot (2020), 'Parrot - european leader in professional and consumer drones'. Viewed: 5 September 2020.
URL: *https://www.parrot.com/en/shop/buy-anafi*

Rathbun, D., Kragelund, S., Pongpunwattana, A. & Capozzi, B. (2002), An evolution based path planning algorithm for autonomous motion of a uav through uncertain environments, *in* 'Proceedings. The 21st Digital Avionics Systems Conference', Vol. 2, IEEE, pp. 8D2–8D2.

Rossiter, A. (2018), 'Drone usage by militant groups: exploring variation in adoption', *Defense & Security Analysis* **34**(2), 113–126.

Tang, P., Yang, Y. & Li, X. (2001), Dynamic obstacle avoidance based on fuzzy inference and transposition principle for soccer robots, *in* '10th IEEE International Conference on Fuzzy Systems.(Cat. No. 01CH37297)', Vol. 3, IEEE, pp. 1062–1064.

Tomlin, C., Pappas, G. J. & Sastry, S. (1998), 'Conflict resolution for air traffic management: A study in multiagent hybrid systems', *IEEE Transactions on automatic control* **43**(4), 509–521.

West, J. P. & Bowman, J. S. (2016), 'The domestic use of drones: An ethical analysis of surveillance issues', *Public Administration Review* **76**(4), 649–659.

Yanmaz, E., Yahyanejad, S., Rinner, B., Hellwagner, H. & Bettstetter, C. (2018), 'Drone networks: Communications, coordination, and sensing', *Ad Hoc Networks* **68**, 1–15.

Appendix A

Project Specification

ENG4111/4112 Research Project

Project Specification

For: Chris Arnold

Title: Drone swarm for tracking malicious drone

Major: Electrical & Electronics Engineering

Supervisor: Dr. Jason Brown

Enrolment: ENG4111 – ONL S1, 2020 ENG4112 – ONL S2, 2020

Project Aim: To determine the effectiveness of different drone swarm algorithms when pursuing a malicious drone of unknown agility or performance.

Programme: Version 1, 16th March 2020

1. Gather information relating to currently used methods for controlling and organising swarms of UAV's or other semi-autonomous machines.

2. Create a custom program module that simulates individual UAV drones taking into account real-world issues such as turn rate, acceleration, target tracking accuracy, etc.

3. Integrate inter-swarm communications methods for passing data between swarm UAV's.

4. Create a custom program module to simulate malicious UAV drone(s) that fly along a pre-set path.

5. Design algorithms for pathfinding, collision avoidance, and swarm positioning.
6. Test code for software bugs and suitability to perform within the simulation framework.
7. Design appropriate simulation parameters to test effectiveness of swarm algorithm such as the malicious drone flight path and manoeuvring properties or swarm size and initial positions.
8. Run the simulations and gather appropriate data.
9. Analyse simulation data and adjust simulation parameters if further data is required.

If time permits:

10. Determine if simple machine learning code will increase effectiveness of swarm tracking abilities.
11. Convert from 2 axis flight paths (horizontal) to 3 axis simulations.
12. Add environmental obstacles for collision avoidance
13. Source data from real-world UAV's to use in the simulations.
14. Utilise OMNET++ built-in communication functions to simulate real-world communication issues (latency, range attenuation, packet loss etc)

Appendix B

Project Timeline

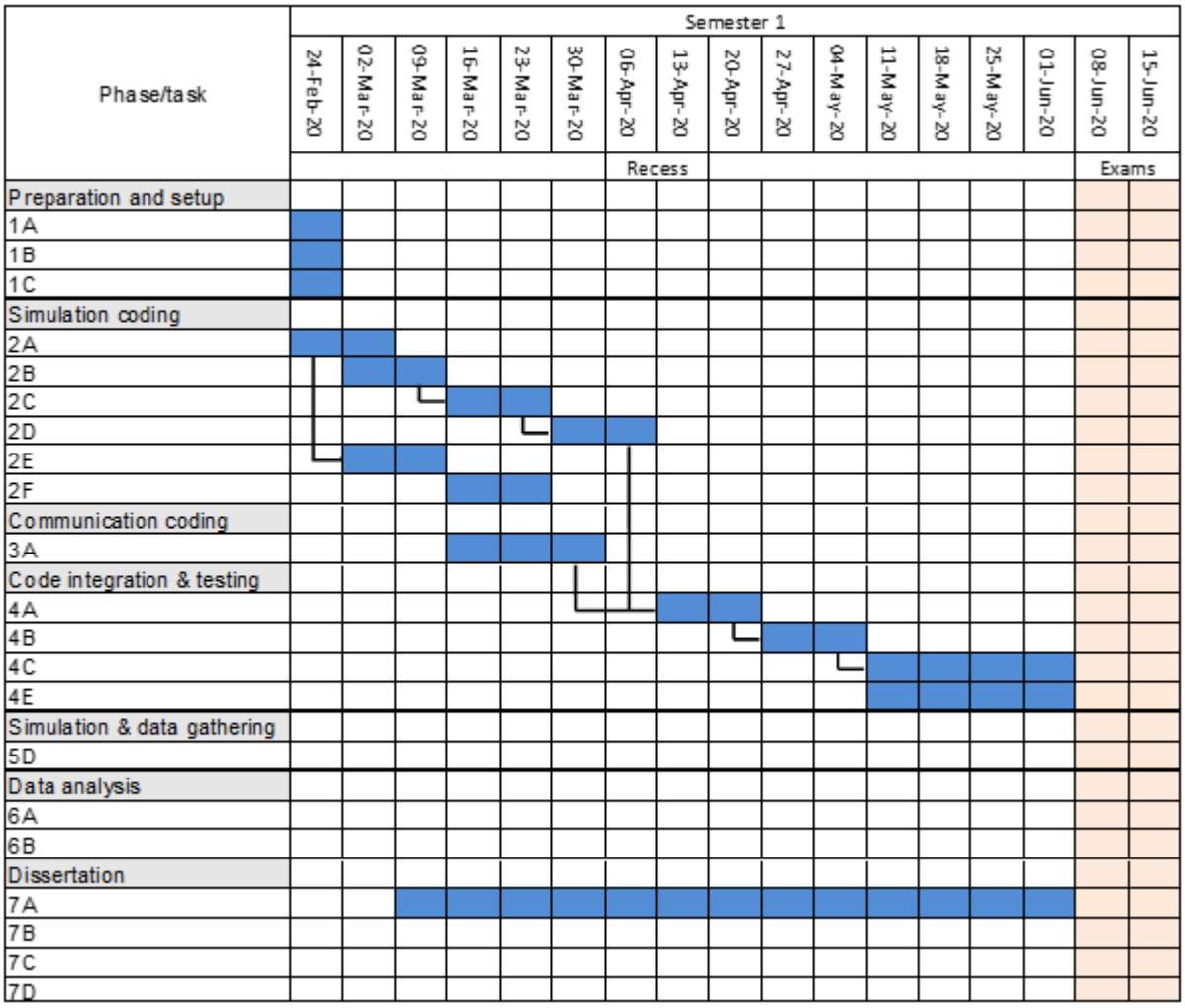


Figure B.1: Semester 1 timeline.

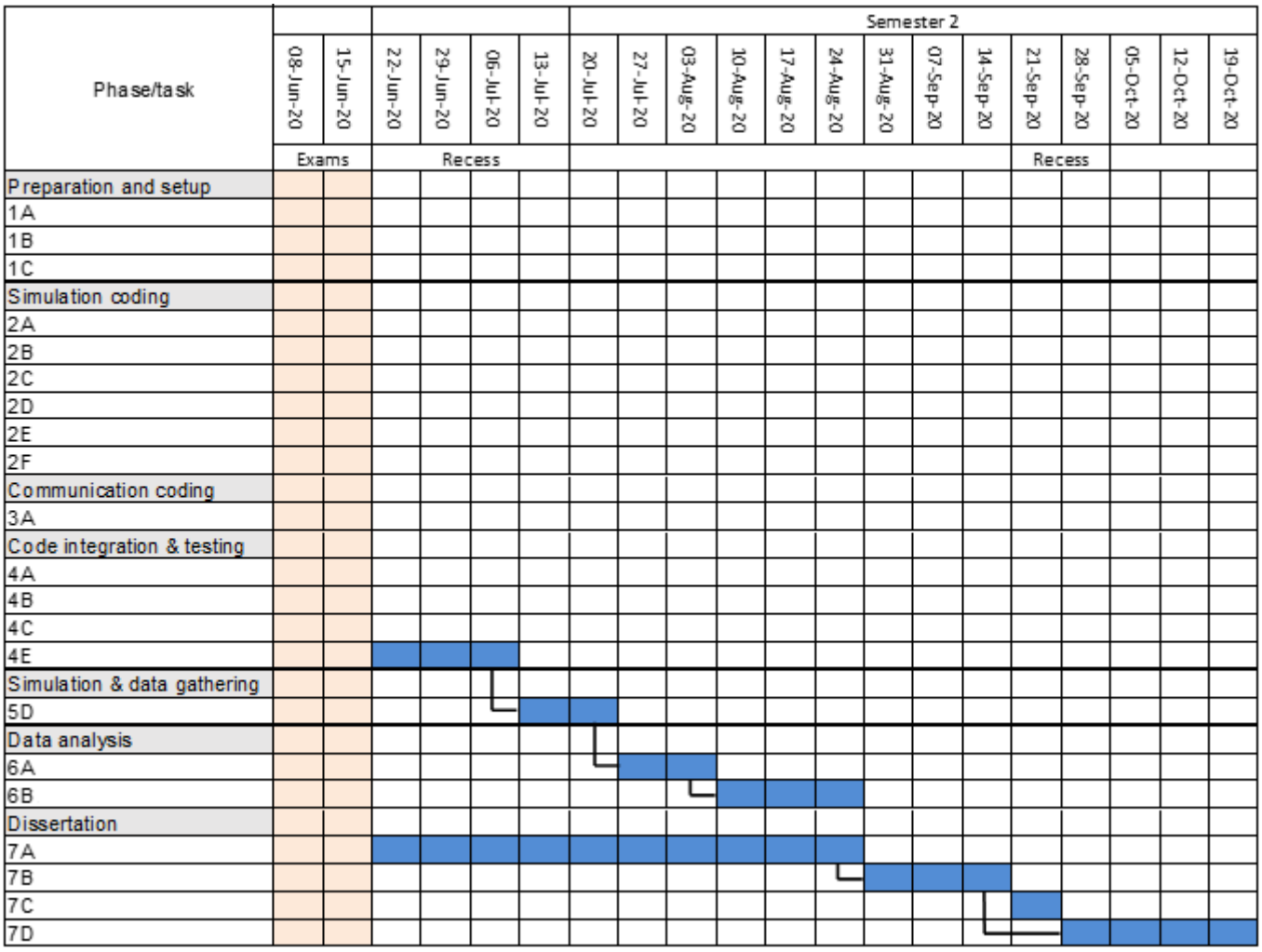


Figure B.2: Semester 2 timeline.

Phase 1	Preparation and setup
1A	Submit Project approval to USQ ENP2020 by deadline
1B	Obtain Omnet++ IDE
1C	Gather Omnet++ and C++ libraries relevant to Project
Phase 2	Simulation coding
2A	Basic simulated flight code implemented
2B	Inter-drone communication nodes code implemented
2C	Collision avoidance code implemented
2D	Pursuit code implemented
2E	Modify swarm basic flight code for malicious drone
2F	Malicious drone itinerary (flight path) planning code
Phase 3	Communication coding
3A	Integrate communication coding between nodes
Phase 4	Code integration & testing
4A	Individual swarm drone processes integrated and tested
4B	Malicious drone process integrated and tested
4C	Testing of overall system configuration
4D	Debug of code
Phase 5	Simulation & data gathering
5A	Run official simulations & record results
Phase 6	Data analysis
6A	Process data and collate
6B	Analyse results
Phase 7	Dissertation
7A	Draft dissertation. Submit to supervisor
7B	Review feedback. Construct presentation for PP2
7C	PP2 conference presentation
7D	Final submission
7E	Submit for marking

Figure B.3: Timeline phase descriptions.

Appendix C

Risk Assessment

This project was expected to be fully completed in simulation as, without the foundation that this project laid down, the ability to implement in real-world within the time frame was not feasible. As such, there were no identifiable physical risks to record. Risks to the project from not meeting deadlines were considered due to the implications of unforeseen coding complications on the timeline. These were mitigated by utilising a modular approach to the research project; there were three overall aims to be achieved depending on time available. The three aims listed in order of priority are:

- Simulation of drones working in a swarm to pursue a target
- Simulation of different drone coordination algorithms
- Simulation of communication network limitations applied to drone swarm

This ensured a base level of research output in the worst-case scenario that outside influences caused delays.

Appendix D

DroneModule Code

D.1 DroneMobility.h

```
#ifndef __INET_DRONEMOBILITY_H
#define __INET_DRONEMOBILITY_H


#include "inet/common/INETDefs.h"
#include "inet/mobility/base/LineSegmentsMobilityBase.h"
#include "inet/common/geometry/common/GeographicCoordinateSystem.h"
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <ctime>
#include <string>
#include <chrono>

//#include <string.h>
namespace inet {

/*
 * @brief Base model for swarm drone decision and control
 *
 * @author Chris Arnold
 */
class INET_API DroneMobility : public LineSegmentsMobilityBase
```

```
{
  protected:
    double maxSpeed = 1.0;     ///< speed of the host
    double speed = 0.0;     ///< speed mean
    double speedStdDev = 0.0;     ///< speed standard deviation
    rad angle = rad(0.0);     ///< angle of linear motion
    rad angleMean = rad(0.0);     ///< angle mean
    rad angleStdDev = rad(0.0);     ///< angle standard deviation
    double alpha = 0.0;     ///< alpha parameter in [0;1] interval
    double margin = 0.0;     ///< margin at which the host gets repelled from
        the border
    IMobility *sourceMobility = nullptr;
    IMobility *targetMobility = nullptr;
    double sensorRange;
    int droneNum = 0;
    Coord target, source, targetActual, swarmTarget;
    double range;
    double maxAcceleration;
    double acceleration, deceleration;
    double sensorAccuracy;
    Coord mDronePrevPos;
    Coord mDroneHeading;
    Coord mDroneFuturePos;
    std::string outputFileName;
    std::ofstream dataOut;
    int finishCount = 0;
    int seed;
    double initDist = -1;
    Coord swarmAvTarget;
    double turnRate;
    static std::string s_comms[];
    int swarmSize, swarmPositionNum, totalsDrones;
    static double s_swarmInfo[30][30];
    static double s_dataOut[30000][30];
    std::string outputFile = "outputDrones.txt";
    double indSwarmInfo[30][30];
    Coord prevPosition;
    Coord droneHeading;
    Coord direction;
    double droneElev = 10;
    double randNum;
    Coord accuracyTrack;
    double accuracyModifier;
```

```
    Coord targetWalkModifier;
    std::string targetTrackMode, droneMode, swarmPosMode;
    double surroundRadius;
    double vClosing, prevLOSmag, N;
    int initStage;
    int iteration;
    Coord targetPrev, sourcePrev;
    double formationRadius = 0;
    double mDroneAvgSpeed = 0, mDroneMaxSpeed = 0, mDroneAgility = 0,
        mDroneAcceleration = 1, mDroneCurrentSpeed = 0;
    double mDronePrevSpeed = 0;
    int averageCount = 0;
    double approachDistance;
    bool overshootMode = false;
    enum movement{STATIONARY, CHASE, APPROACH, FORMATION, END} movementMode;
    double relVelocity;
    bool initCheck;
    cTextFigure *textFigure;
  protected:
    virtual int numInitStages() const override { return NUM_INIT_STAGES; }

    /** @brief Initializes mobility model parameters.*/
    virtual void initialize(int stage) override;

    /** @brief If the host is too close to the border it is repelled */
    void preventBorderHugging();

    /** @brief Move the host*/
    virtual void move() override;

    /** @brief Calculate a new target position to move to. */
    virtual void setTargetPosition() override;

    /** @brief Initializes the position according to the mobility model. */
    virtual void setInitialPosition() override;

    //virtual void refreshDisplay() override;

    double quadrantHeading();

    void mDroneFuturePosition();

    void targetAccuracyAdjust();
```

```
void targetTrackerUpdate();

void checkComms();

void targetInfoComms();

void circleFunction();

void circleFunction(double);

void circleFunction(bool);

bool collisionCheck();

void collisionAvoid();

void setAccuracyTrack();

void targetTrackModeModify();

void swarmPositioning();

void targetApproach();

void speedModify(std::string);

void speedModify(double);

void convergeDiverge();

void tangentCoord();

void updateVariables();

void placeMark();

void surroundCheck();

int leftRightCheck();

void printToFile();
```

```
    void surroundEfficiency();

    unsigned long xorshf96(void);
  public:
    virtual double getMaxSpeed() const override { return speed; }
    DroneMobility();
    ~DroneMobility();
};

} // namespace inet

#endif // ifndef _INET_GAUSSMARKOVMOBILITY_H
```

D.2 DroneMobility.cc

```
#include "inet/mobility/single/DroneMobility.h"


#define MAX_DRONE_LIMIT 30
#define COMMS_ARRAY_DATA 20
#define SWARM_NUM 0
#define TARGET_X 1
#define TARGET_Y 2
#define TARGET_Z 3
#define SOURCE_X 4
#define SOURCE_Y 5
#define SOURCE_Z 6
#define SWARM_POS 7
#define COLLISION_RADIUS 10
#define PI osg::PI
#define TARGET_TRACK "direct"
#define FASTER true
#define SLOWER false
#define ENABLED true
#define PICONV *180/PI
#define DEG *180/PI

namespace inet {

enum side{LEFT,RIGHT,STRAIGHT};

Define_Module(DroneMobility);
```

```
std::string DroneMobility::s_comms[MAX_DRONE_LIMIT];
double DroneMobility::s_swarmInfo[30][30];
double DroneMobility::s_dataOut[30000][30];
DroneMobility::DroneMobility()
{
    //std::remove("outputDrones.txt");

}
DroneMobility::~DroneMobility()
{
    if(droneNum == 0){
    std::string outputFileName = swarmPosMode + ".txt";
    dataOut.open("output.txt", std::ios_base::app);
        for(int j = 0;j < 10+2;j++){
            if(j < swarmSize+2){
            dataOut << std::to_string(s_dataOut[swarmSize][j]) << "\t";
            }else{
                dataOut << "\t";
            }
        }
    dataOut << endl;
    }
    dataOut.close();

}

void DroneMobility::initialize(int stage)
{
    LineSegmentsMobilityBase::initialize(stage);

    EV_INFO << "initializing DroneMobility stage " << stage << endl;
    initStage = stage;
    if (stage == INITSTAGE_LOCAL) {
        droneNum = par("droneNumber");
        EV_INFO << " should be only here on init stage 0. stage:" << stage
            << " drone: " << droneNum<<endl;
        maxSpeed = par("maxSpeed");
        speed = par("speed");
        totalsDrones = par("swarmSize");
        stationary = false;
        movementMode = CHASE;
        EV_INFO << "movementMode: " <<  movementMode << endl;
        averageCount = 0;
```

```
sourceMobility = getModuleFromPar<IMobility>(par("sourceMobility"),
    this);
targetMobility = getModuleFromPar<IMobility>(par("targetMobility"),
    this);
sensorRange = par("sensorRange");
Coord targetWalkModifier;
iteration = par("iteration");
swarmSize = par("swarmSize");
seed = iteration + swarmSize*10 + time(NULL);
srand (seed);
acceleration = deceleration = par("acceleration");
sensorAccuracy = par("sensorAccuracy");
surroundRadius = par("surroundRadius");
initCheck = true;
targetTrackMode = par("targetTrackMode").stdstringValue();
droneMode = par("droneMode").stdstringValue();
swarmPosMode = par("swarmPosMode").stdstringValue();
mDronePrevPos = mDroneFuturePos = Coord();//initialise prev pos as
    0,0,0
turnRate = par("turnRate");//turn rate should be inversely
    proportional to speed (in degrees per second
s_comms[droneNum] = "test";
overshootMode = par("overshootMode");
double animSpeed = getEnvir()->getAnimationSpeed();
EV_INFO << s_comms[droneNum]<< droneNum << "ani_speed:" << animSpeed
    << endl;
s_swarmInfo[MAX_DRONE_LIMIT][29] = {};
s_dataOut[30000][29] = {};
cCanvas *canvas = getSystemModule()->getCanvas(); // toplevel canvas
canvas->setAnimationSpeed(1.0, this); //smooth animation
cFigure *figure;
switch(droneNum){
case 0:{
    figure = canvas->getFigureByPath("drone0.label");
    break;
}
case 1:{
    figure = canvas->getFigureByPath("drone1.label");
    break;
}
case 2:{
    figure = canvas->getFigureByPath("drone2.label");
    break;
```

```
        }
        case 3:{
            figure = canvas->getFigureByPath("drone3.label");
            break;
        }
        case 4:{
            figure = canvas->getFigureByPath("drone4.label");
            break;
        }
        case 5:{
            figure = canvas->getFigureByPath("drone5.label");
            break;
        }
        case 6:{
            figure = canvas->getFigureByPath("drone6.label");
            break;
        }
        case 7:{
            figure = canvas->getFigureByPath("drone7.label");
            break;
        }
        case 8:{
            figure = canvas->getFigureByPath("drone8.label");
            break;
        }
        case 9:{
            figure = canvas->getFigureByPath("drone9.label");
            break;
        }
        default:
            break;
        }

        textFigure = check_and_cast<cTextFigure *>(figure);
        textFigure->setText("x");
        textFigure->setColor("red");

        }
}

void DroneMobility::preventBorderHugging()
{
    bool left = (lastPosition.x < constraintAreaMin.x + margin);
```

```
    bool right = (lastPosition.x >= constraintAreaMax.x - margin);
    bool top = (lastPosition.y < constraintAreaMin.y + margin);
    bool bottom = (lastPosition.y >= constraintAreaMax.y - margin);
    if (top || bottom) {
        angleMean = bottom ? deg(270.0) : deg(90.0);
        if (right)
            angleMean -= deg(45.0);
        else if (left)
            angleMean += deg(45.0);
    }
    else if (left)
        angleMean = deg(0.0);
    else if (right)
        angleMean = deg(180.0);

}

void DroneMobility::move()
{

    simtime_t now = simTime();
    if (now == nextChange) {
        targetTrackerUpdate(); // simulates pulls the info from the image
            processing
        targetInfoComms();
        checkComms();   // simulates pulling info from static comms array
        mDroneFuturePosition(); // predict mDrone heading - mainly for
            sDrone enrollment
        prevPosition = lastPosition;
        lastPosition = targetPosition;
        setTargetPosition();
        lastVelocity = (targetPosition - lastPosition) / (nextChange -
            simTime()).dbl();
        EV_INFO << "lastVelocity: " << lastVelocity << " speed : " << speed
            << " simtime:"<<simTime().dbl()<<endl;
        placeMark();
        printToFile();

    }
    handleIfOutside(REFLECT, lastPosition, lastVelocity, angle);
}

void DroneMobility::printToFile(){
```

```
    int timeStamp = ceil(simTime().dbl()*10);
    int modeOut = -1;
    if(swarmPosMode == "follow"){
        modeOut = 1;
    }else if(swarmPosMode == "surround"){
        modeOut = 2;
    }else if(swarmPosMode == "cone"){
        modeOut = 3;
    }

    if(droneNum == 0){
        s_dataOut[swarmSize][0] = modeOut;
        s_dataOut[swarmSize][1] = iteration;
    }
    s_dataOut[swarmSize][droneNum+2] = source.distance(swarmTarget);
}

void DroneMobility::placeMark(){
    textFigure->setPosition(cFigure::Point(target.x,target.y));
}

void DroneMobility::checkComms(){
    int tempCheck = 0;
    for(int i = 0;i<MAX_DRONE_LIMIT;i++){
        for(int j = 0;j<COMMS_ARRAY_DATA;j++){
            indSwarmInfo[i][j] = s_swarmInfo[i][j];
        }
        if(indSwarmInfo[i][0]!=-1 && indSwarmInfo[i][0]!=0 ){//if that drone
            has a target
            target.x = s_swarmInfo[i][TARGET_X];
            target.y = s_swarmInfo[i][TARGET_Y];
            target.z = s_swarmInfo[i][TARGET_Z];//use their info
            if((swarmPosMode == "surround" || swarmPosMode == "cone") &&
                movementMode == STATIONARY){
                movementMode = CHASE;
            }
        }
        tempCheck += s_swarmInfo[i][SWARM_NUM];
    }
    EV_INFO << "swarmSize:" << swarmSize;
}

void DroneMobility::targetInfoComms(){
```

```
    //sends information to static array
    if(movementMode != STATIONARY){
        s_swarmInfo[droneNum][SWARM_NUM] = 1;
        s_swarmInfo[droneNum][TARGET_X]= target.x;
        s_swarmInfo[droneNum][TARGET_Y]= target.y;
        s_swarmInfo[droneNum][TARGET_Z]= target.z;
        s_swarmInfo[droneNum][TARGET_X+3]= targetActual.x;
        s_swarmInfo[droneNum][TARGET_Y+3]= targetActual.y;
        s_swarmInfo[droneNum][TARGET_Z+3]= targetActual.z;
    }else{ //if drone doesn't have a target or is at end (so is stationary)
        s_swarmInfo[droneNum][SWARM_NUM] = -1;//unenrol from swarm
        s_swarmInfo[droneNum][TARGET_X]= -1;
        s_swarmInfo[droneNum][TARGET_Y]= -1;
        s_swarmInfo[droneNum][TARGET_Z]= -1;
        s_swarmInfo[droneNum][TARGET_X+3]= -1;
        s_swarmInfo[droneNum][TARGET_Y+3]= -1;
        s_swarmInfo[droneNum][TARGET_Z+3]= -1;
    }
    s_swarmInfo[droneNum][SOURCE_X]= source.x;
    s_swarmInfo[droneNum][SOURCE_Y]= source.y;
    s_swarmInfo[droneNum][SOURCE_Z]= source.z;

}

void DroneMobility::updateVariables(){
    EV_INFO << "mdroneCurrentSpeed: " << mDroneCurrentSpeed << " " <<
        mDroneHeading;
    mDroneCurrentSpeed = mDronePrevPos.distance(target)/updateInterval.dbl()
        ;
    EV_INFO << "mdroneCurrentSpeed: " << mDroneCurrentSpeed << " " <<
        mDronePrevPos.distance(target) <<endl;
}


void DroneMobility::setTargetPosition()
//mainly used to update targetPosition through speed (speedModify) &
    direction (circleFunction) variables
{
    EV_INFO << "here:" << (mDroneCurrentSpeed < 0.5 && speed < 0.5) << endl;
    if(mDroneCurrentSpeed < 0.5 && speed < 0.5){
        if(finishCount++ > 10){
            EV_INFO << "end here" << endl;
            movementMode = END;
```

```
        }

    }
    nextChange = simTime() + updateInterval;
    source = sourceMobility->getCurrentPosition();
    droneHeading = (source - prevPosition);
    EV_INFO << " target: " << target << " targetPrev: " << targetPrev <<
        endl;
    switch(movementMode){
    case STATIONARY:{
        swarmPositioning();
        if(speed>0){
            speedModify(0.0);//slow down to stop
            targetPosition = lastPosition + direction * (speed *
                updateInterval.dbl());
        }
        //check for change in state
        EV_INFO << " dist: " << source.distance(target)<< " range: " <<
            sensorRange<<endl;
        if(source.distance(target)<= sensorRange){
            movementMode = CHASE;
        EV_INFO << " here" << endl;
        }
        break;
    }
    case CHASE:{
        EV_INFO << "chase mode" << endl;
        EV_INFO << "sDroneHeading: " << droneHeading<< " prev pos: " <<
            prevPosition<< " source: " << source<< " diff: " <<(source -
            prevPosition)<<endl;
        targetTrackModeModify();//changes where the target point is +
            targetApproach
        if(collisionCheck()){
            collisionAvoid(); //modify targetPosition to avoid obstacle
            circleFunction(true); //check within bounds of turning circle &
                modify speed based on mDrone pos in turn circle
            EV_INFO << "near collision"<<endl;
            targetPosition = lastPosition + direction * (speed *
                updateInterval.dbl());
        }else{//if not going to collide with something
            circleFunction(true); //check within bounds of turning circle &
                modify speed based on mDrone pos in turn circle
            speedModify(maxSpeed);
```

```
            targetPosition = lastPosition + direction * (speed *
                updateInterval.dbl()); //need to change for targeting method
        }

        EV_INFO << "current heading(degrees): " << (atan2(droneHeading.y,
            droneHeading.x)*180/PI) << "target heading: " << (atan2(
            direction.y,direction.x)*180/PI) <<endl;
        EV_INFO << "my location: " << source << " :: next location: " <<
            targetPosition << " speed: "<< speed<< endl;
        double angle = atan2(target.y-source.y, target.x-source.x);
        EV_INFO << "angle:" << angle*180/PI<<" target:"<<target<<" source:"
            << source<< endl;
        Coord temp = droneHeading - mDroneHeading;
        relVelocity = sqrt(pow(temp.x,2) + pow(temp.y,2))/updateInterval.dbl
            ();
        double distanceToSlow = (pow(speed-relVelocity,2) - pow(speed,2))
            /(2*deceleration);
        EV_INFO << "rel vel: " << relVelocity << " distanceToSlow: " << abs(
            distanceToSlow) << " act distance: "<<source.distance(
            swarmTarget)<< endl;
        if(initCheck) {
            initCheck = false;
            break;
        }
        if(source.distance(targetActual) > sensorRange){
            movementMode = STATIONARY;
        }else if(overshootMode == ENABLED && source.distance(swarmTarget) <=
             abs(distanceToSlow)){
            movementMode = APPROACH;
            EV_INFO << "approach mode activated" << endl;
        }
        break;

    }
    case APPROACH:{
        EV_INFO << "approach mode" << endl;
        targetTrackModeModify();
        Coord temp = droneHeading - mDroneHeading;
        relVelocity = sqrt(pow(temp.x,2) + pow(temp.y,2))/updateInterval.dbl
            ();
        double distanceToSlow = (pow(speed-relVelocity,2) - pow(speed,2))
            /(2*deceleration);
        EV_INFO << "rel vel: " << relVelocity << " distanceToSlow: " << abs(
```

```
            distanceToSlow) << " act distance: "<<source.distance(
            swarmTarget)<< endl;
        speedModify(mDroneCurrentSpeed);
        circleFunction(true);
        EV_INFO << "rel vel: " << relVelocity << " distanceToSlowmod: " <<
            abs(distanceToSlow*1.2) << " act distance: "<<source.distance(
            swarmTarget)<< endl;

        targetPosition = lastPosition + direction * (speed * updateInterval.
            dbl());
        EV_INFO << "a:" <<(source.distance(swarmTarget) > distanceToSlow
            +1.2) << " b:" <<(relVelocity > acceleration) << endl;
        if(source.distance(swarmTarget) > abs(distanceToSlow+1.2)){// ||
            relVelocity > acceleration){
            movementMode = CHASE;
        }
        if(source.distance(swarmTarget) < 15 || relVelocity <= acceleration)
            {
            formationRadius = 10;
            movementMode = FORMATION;
        }
        break;
    }
    case FORMATION:{
        EV_INFO << "formation mode" << endl;
        targetTrackModeModify();
        speedModify(mDroneCurrentSpeed<0.5 ? 0 : mDroneCurrentSpeed);
        Coord temp = droneHeading - mDroneHeading;
        relVelocity = sqrt(pow(temp.x,2) + pow(temp.y,2))/updateInterval.dbl
            ();
        double distanceToSlow = (pow(speed-relVelocity,2) - pow(speed,2))
            /(2*deceleration);
        EV_INFO << "rel vel: " << relVelocity << " distanceToSlow: " << abs(
            distanceToSlow) << " act distance: "<<source.distance(
            swarmTarget)<< " target: " << swarmTarget << endl;
        double mDroneHeadingAngle = atan2(-mDroneHeading.y, mDroneHeading.x)
            ;
        circleFunction(true);
        targetPosition = lastPosition + direction * (speed * updateInterval.
            dbl());
        if(source.distance(swarmTarget) > formationRadius){
            movementMode = CHASE;
        }
```

```
            break;
        }
        case END:{}
        default:{
            break;
        }
        }
}

void DroneMobility::targetApproach(){
    speedModify(mDroneCurrentSpeed);
    EV_INFO << "approach mode. setting speed to " << mDroneCurrentSpeed <<
        endl;
}

void DroneMobility::circleFunction(double headingInput){//takes reference
    input. not stopping sudden turns

    double radius = speed*speed/acceleration;//meters
    double theta = (speed*updateInterval.dbl()*acceleration/speed);
    double maxTurnAngle = PI/2 - acos(sqrt(1-cos(theta)/2));
    //determine the relative angle of the mDrone
    double heading = (atan2(-droneHeading.y,droneHeading.x));//rads
    Coord mHeadingCoord = target - source;
    //EV_INFO << "target: " <<target<< " source: " <<source<< "
        mHeadingCoord: " <<mHeadingCoord<<endl;
    double angleToTarget = (atan2(-mHeadingCoord.y,mHeadingCoord.x));
    double difference = headingInput - heading;
    EV_INFO << " headingInput: " << headingInput DEG<< " heading: " <<
        heading DEG;
    EV_INFO << " difference: " << difference DEG;
    if(difference > PI){
        difference =  difference-(2*PI);
        EV_INFO << "difference changed to : " << difference DEG;
    }else if(difference < -PI){
        difference = difference + (2*PI);
        EV_INFO << "difference changed to : " << difference DEG;
    }
    EV_INFO << endl;
    if(abs(difference) > maxTurnAngle){ //target outside turn arc
        double turnAngle = 0;
        if(speed == 0){
            speedModify("FASTER");
```

```
        }else if(difference > 0){ //target to left of drone
            turnAngle = heading + maxTurnAngle;
            EV_INFO << "turn left" << endl;
        }else{
            turnAngle = heading - maxTurnAngle;
            EV_INFO << "turn right" << endl;
        }
        double xMove = cos(turnAngle);
        double yMove = -sin(turnAngle);
        direction = Coord(xMove, yMove,0.0);
    }else{ //within turn arc
        if(speed == 0)
            speedModify("FASTER");
        direction = (target - source).normalize();
    }
}

void DroneMobility::circleFunction(bool check){
    double radius = speed*speed/acceleration;//meters
    double theta = (speed*updateInterval.dbl()*acceleration/speed);
    double maxTurnAngle = PI/2 - acos(sqrt(1-cos(theta)/2));
    double heading = (atan2(-droneHeading.y,droneHeading.x));//rads
    Coord mHeadingCoord = target - source;
    double angleToTarget = (atan2(-mHeadingCoord.y,mHeadingCoord.x));
    double difference = angleToTarget - heading;
    if(difference > PI){
        difference =  difference-(2*PI);
        EV_INFO << "difference changed to : " << difference DEG;
    }else if(difference < -PI){
        difference = difference + (2*PI);
        EV_INFO << "difference changed to : " << difference DEG;
    }
    if(abs(difference) > maxTurnAngle){ //target outside turn arc
        double turnAngle = 0;
        if(speed == 0 && movementMode != FORMATION){
            speedModify("FASTER");
        }else if(difference > 0){ //target to left of drone
            turnAngle = heading + maxTurnAngle;
            EV_INFO << "turn left" << endl;
        }else{
            turnAngle = heading - maxTurnAngle;
            EV_INFO << "turn right" << endl;
        }
```

```
        double xMove = cos(turnAngle);
        double yMove = -sin(turnAngle);
        direction = Coord(xMove, yMove,0.0).normalize();
    }else{ //within turn arc
        if(speed == 0)
            speedModify("FASTER");
        double xMove = cos(difference);
        double yMove = -sin(difference);
        direction = (target - source).normalize();
    }
}


int DroneMobility::leftRightCheck(){
        double heading = (atan2(-droneHeading.y,droneHeading.x));//rads
        Coord mHeadingCoord = target - source;
        double angleToTarget = (atan2(-mHeadingCoord.y,mHeadingCoord.x));
        double difference = angleToTarget - heading;
        if(difference > PI){
            difference =  difference-(2*PI);
            EV_INFO << "difference changed to : " << difference DEG;
        }else if(difference < -PI){
            difference = difference + (2*PI);
            EV_INFO << "difference changed to : " << difference DEG;
        }
            if(difference > 0){ //target to left of drone
                return 1;
                EV_INFO << "target left" << endl;
            }else{
                return -1;
                EV_INFO << "target right" << endl;
            }
}

void DroneMobility::tangentCoord(){

    double radiusSDrone = speed*speed/acceleration;//meters
    double radiusMDrone = mDroneCurrentSpeed*mDroneCurrentSpeed/
        mDroneAcceleration;
    double droneDistance = source.distance(target);
    double tangentAngle, tangentA2;
    double angleToMDrone = atan2(target.y-source.y, target.x-source.x);
    double angleToSDrone = atan2(source.y-target.y, source.x-target.x);
```

```
Coord tangentA1Coord,tangentA2Coord, sTurnCenterPoint, mTurnCenterPoint;
    //=centrepoint of turning circle
double sDroneHeading = atan2(droneHeading.y,droneHeading.x);
side sDroneSide = STRAIGHT;
side mDroneSide = STRAIGHT;
if(angleToMDrone-sDroneHeading < 0){// target to drones right
    sDroneSide = RIGHT;
    sTurnCenterPoint = source + Coord(radiusSDrone*cos(sDroneHeading-PI
        /2),radiusSDrone*sin(sDroneHeading-PI/2));
    EV_INFO << "turn right : " <<Coord(radiusSDrone*cos(sDroneHeading-PI
        /2),radiusSDrone*sin(sDroneHeading-PI/2));
}else{
    sDroneSide = LEFT;
    sTurnCenterPoint = source + Coord(radiusSDrone*cos(sDroneHeading+PI
        /2),radiusSDrone*sin(sDroneHeading+PI/2));
    EV_INFO << "turn left : " <<Coord(radiusSDrone*cos(sDroneHeading+PI
        /2),radiusSDrone*sin(sDroneHeading+PI/2));
}
double mDroneHeadingAngle = atan2(mDroneHeading.y, mDroneHeading.x);
if(angleToSDrone - mDroneHeadingAngle < 0){
    mDroneSide = RIGHT;
    EV_INFO << " mDrone right side "<<angleToSDrone*180/PI<<" "<<
        mDroneHeadingAngle*180/PI;
    mTurnCenterPoint = target + Coord(radiusMDrone*cos(
        mDroneHeadingAngle-PI/2),radiusMDrone*sin(mDroneHeadingAngle-PI
        /2));
}else{
    mDroneSide = LEFT;
    EV_INFO << " mDrone left side "<<angleToSDrone*180/PI<<" "<<
        mDroneHeadingAngle*180/PI;
    mTurnCenterPoint = target + Coord(radiusMDrone*cos(
        mDroneHeadingAngle+PI/2),radiusMDrone*sin(mDroneHeadingAngle+PI
        /2));
}
EV_INFO << " turnCenterPoint:" << sTurnCenterPoint<< endl;
if(sDroneSide == STRAIGHT){



}else if(sDroneSide != mDroneSide ){//internal tangent
    EV_INFO << "internal tangent ";
    double radiusAdd = radiusSDrone + radiusMDrone;
    double hypA1 = 0;
    double ratio = radiusAdd/radiusMDrone;
```

```
        double x = droneDistance/(ratio+1);
        tangentAngle = acos(radiusAdd/(droneDistance - x));
        tangentAngle += angleToSDrone;
        tangentA2Coord = mTurnCenterPoint + Coord(radiusAdd*cos(tangentAngle
            ),radiusAdd*sin(tangentAngle));
        //tangentA2Coord =
    }else{//external tangent
        EV_INFO << "external tangent ";
        double radiusSubtract = radiusSDrone - radiusMDrone;
        tangentAngle = acos(radiusSubtract/droneDistance);
    }


}

void DroneMobility::convergeDiverge(){

}

void DroneMobility::speedModify(std::string input){
    EV_INFO << "speed: " << speed;
    if(input == "FASTER"){
        EV_INFO << "speed up ";
        if(speed < maxSpeed){
            if(speed + (acceleration*updateInterval.dbl()) > maxSpeed){
                speed = maxSpeed;
            }else{
                speed += (acceleration*updateInterval.dbl());
            }
        }
    }else if(input == "SLOWER"){
        EV_INFO << "slow down ";
        if(speed > 0){                  //
            if(speed - (acceleration*updateInterval.dbl()) < 0){
                speed = 0;
            }else{
                speed -= (acceleration*updateInterval.dbl());
            }
        }
    }
    EV_INFO << " new speed: " << speed << endl;
}
```

```
void DroneMobility::speedModify(double newSpeed){ //mainly used to adjust
    speed for use in setTargetPosition
    EV_INFO << "speed: " << speed << " set speed: " << newSpeed;
    double difference = newSpeed - speed;
    double maxAcceleration = acceleration*updateInterval.dbl();
        if(newSpeed > speed){
            EV_INFO << " accelerating to " << newSpeed;

            if(speed < maxSpeed && difference >= maxAcceleration){
                if(speed + (maxAcceleration) > maxSpeed){
                    speed = maxSpeed;
                }else{
                    speed += (maxAcceleration);
                }
            }else if(speed < maxSpeed && difference < maxAcceleration){
                speed = newSpeed;
            }
        }else if(newSpeed < speed){
            if(speed > 0){                  //
                EV_INFO << " decelerating to " << newSpeed;
                if(speed - (maxAcceleration) < 0){
                    speed = 0;
                }else if(speed > maxSpeed && difference > maxAcceleration){
                }else{
                    speed -= (maxAcceleration);//same here
                }
            }
        }
        EV_INFO << " new speed: " << speed << endl;
}

void DroneMobility::setInitialPosition()
{
    auto coordinateSystem = getModuleFromPar<IGeographicCoordinateSystem>(
        par("coordinateSystemModule"), this, false);
    if (coordinateSystem != nullptr && hasPar("initialLatitude") && hasPar("
        initialLongitude") && hasPar("initialAltitude")) {
        auto initialLatitude = deg(par("initialLatitude"));
        auto initialLongitude = deg(par("initialLongitude"));
        auto initialAltitude = m(par("initialAltitude"));
        EV_INFO << "input type:"<<(typeid(initialLatitude).name())<<endl;
        lastPosition = coordinateSystem->computeSceneCoordinate(GeoCoord(
            initialLatitude, initialLongitude, initialAltitude));
```

```
        EV_INFO << "position initialized from initialLatitude/Longitude/
            Altitude parameters: " << lastPosition << endl;
    }else if (coordinateSystem == nullptr && hasPar("initialX") && hasPar("
        initialY") && hasPar("initialZ")) {
        double radsPerDrone = 2*PI/totalsDrones;
        double radius = 1145;
        double xRand = rand()%10-5;
        double yRand = rand()%10 - 5;
        double rotationRand = rand()%(360)-(180);
        EV_INFO << "xRand: "<< xRand << "yRand: "<< yRand << "rotRand: "<<
            rotationRand<<endl;
        double x = par("initialX");
        double y = par("initialY");
        double droneRads = droneNum * radsPerDrone+rotationRand*PI/180 +
            rotationRand;
        lastPosition.x = x+radius*cos(droneRads)+xRand;
        lastPosition.y = y+radius*sin(droneRads)+yRand;
        lastPosition.z = par("initialZ");
        EV_INFO << "position initialized from initialX/Y/Z parameters: " <<
            lastPosition << endl;
    }
}

void DroneMobility::mDroneFuturePosition(){
    //set
    if(mDronePrevPos != Coord()){ //if set to initial (0,0,0)
        if(target - mDronePrevPos != Coord()){
            mDroneHeading = (target - mDronePrevPos);//.normalize();
            mDroneFuturePos = target + mDroneHeading;
            mDroneCurrentSpeed = mDronePrevPos.distance(target)/
                updateInterval.dbl();
            if(mDroneCurrentSpeed > mDroneMaxSpeed){
                mDroneMaxSpeed = mDroneCurrentSpeed;
            }
            mDroneAvgSpeed = (mDroneAvgSpeed * averageCount +
                mDroneCurrentSpeed)/(averageCount+1);
            averageCount++;
            double currentAccel = (mDronePrevSpeed-mDroneCurrentSpeed)/
                updateInterval.dbl();
            if(currentAccel>mDroneAcceleration){
                mDroneAcceleration = currentAccel;
                EV_INFO << "max mDroneAccel:"<<mDroneAcceleration<<endl;
            }
```

```
            mDronePrevSpeed = mDroneCurrentSpeed;

        }
    }
        mDronePrevPos = target;
}

void DroneMobility::targetTrackerUpdate(){
    target = targetMobility->getCurrentPosition();
    targetActual = target;
    targetInfoComms();
}

void DroneMobility::targetAccuracyAdjust(){
    //random walk to go here
/*
    accuracyModifier = ((100-sensorAccuracy)/100);
    int temp = rand()%3-1;
    EV_INFO << "temp: " << temp;
    double x = (rand()%3-1)*accuracyModifier;
    double y = (rand()%3-1)*accuracyModifier;
    double z = (rand()%3-1)*accuracyModifier;//
    Coord randomWalkModifier(x,y); //z set to zero for 2d implementation
    targetWalkModifier = targetWalkModifier+randomWalkModifier;
    EV_INFO << "rand coord:"<< targetWalkModifier << endl;
    //accuracyTrack+= Coord(uniform(-), uniform())
        EV_INFO << "target before: " << target;
        target += targetWalkModifier;
        EV_INFO << " fuzzy target: " << target <<endl;*/
}

bool DroneMobility::collisionCheck(){
    //perform distance calc on each drone in swarm or not
    int closestDroneID = -1;
    double closestDrone = INFINITY;
    for(int i=0;i<swarmSize;i++){
        if(i != droneNum){
            Coord distCoord(indSwarmInfo[i][SOURCE_X],indSwarmInfo[i][
                SOURCE_Y],indSwarmInfo[i][SOURCE_Z]);
            double otherDroneDistance = source.distance(distCoord);
            if(otherDroneDistance > 10000){otherDroneDistance = 0;}//if
                drones are exact same position will return inf for distance
            if(otherDroneDistance < COLLISION_RADIUS && otherDroneDistance <
```

```
                closestDrone){
                closestDrone = otherDroneDistance;
                closestDroneID = i;
            }
        }
    }
    if(closestDroneID != -1){
        return true;
    }
    return false;
}

void DroneMobility::collisionAvoid(){//modify targetPosition to avoid
    obstacle
//need to include circleFunction as it is the delimiter of movement,
    regardless of collision



}

void DroneMobility::targetTrackModeModify(){
    //in PN mode - modifies the position of the target for more efficient
        tracking. This needs to be called
    //before the swarm positioning algorithm
    if(initStage < 12 && initStage >0){
                //EV_INFO << "sTP stage " << initStage << endl;
                return;
            }
    if(swarmPosMode == "surround" || swarmPosMode == "cone" ){ // not
        tracking to the mDrone specifically
            swarmPositioning(); //modify the target Coord to instead target
                drones swarm position
        }else{
            swarmTarget = target;
        }

    Coord temptargetprev = target;
    if((swarmPosMode == "follow" && movementMode == CHASE) || (speed < 1 &&
        movementMode != FORMATION)){
        //dont modify target pos
        EV_INFO <<"jere " << movementMode << endl;
        speedModify(maxSpeed);
    }else if(targetTrackMode == "PN" && (movementMode == CHASE ||
```

```
        movementMode == FORMATION)){
        //lat accel(vector add to target pos)= N(3to5) * LOS change rate *
            closing velocity (relative vel)
        Coord temptargetprev = target;
        N=3;
        Coord RTMold = (targetPrev - sourcePrev);
        Coord RTMnew = (target - source);
        Coord LOSdelta = RTMnew - RTMold;
        double LOSrate = LOSdelta.length();
        double Vc = (RTMold.length() - RTMnew.length());
        double currentLOSAngle = atan2(-RTMnew.y,RTMnew.x);
        vClosing = source.distance(targetPrev) - source.distance(target);
        double latAccelMag = (N * LOSrate * -Vc)*RTMnew.length();
        //left right?
        int test = leftRightCheck();
        EV_INFO << "targetPrev: " << targetPrev << " target: " << target <<
            " sourcePrev: " << sourcePrev<< " source: " << source <<  endl;
        EV_INFO << "RTMold: " << RTMold << " RTMnew: " << RTMnew << "
            LOSrate: " << LOSrate << " Vc: " << Vc  << " leftright:" << test
             << " currentLOSAngle" << currentLOSAngle DEG<<endl;
        Coord Acmd = Coord(latAccelMag*cos(currentLOSAngle+PI/2*test),
            latAccelMag*sin(currentLOSAngle+PI/2*test));
        EV_INFO << " targetb4: " <<target << "Acmd:" << Acmd ;
        target += Acmd;
        EV_INFO << " targetafter: " <<target <<endl;

    }
    targetPrev = swarmTarget;
    sourcePrev = source;
}

//sets the positions of drones based on positioning mode and number of
    drones
void DroneMobility::swarmPositioning(){
    enum{follow, surround, modifiedSurround};
    double distanceModifier = (swarmSize/(1+swarmSize/20));
    double mDroneHeadAngle = atan2(mDroneHeading.y,mDroneHeading.x);
    if(swarmPosMode == "surround"){
        if(droneMode == "efficient"){
            surroundEfficiency();
        }else{
            double radsPerDrone = 2*PI/swarmSize;
            double thisDroneRads = radsPerDrone*droneNum;
```

```
            double startAngle = 0;
            if(swarmSize%2==0){//if swarm size is even
                        startAngle = mDroneHeadAngle - PI/2;
            }else{
                        startAngle = mDroneHeadAngle - PI;
            }
            double positionAngle = startAngle + thisDroneRads;
            double tempRadius = sqrt(70685*swarmSize/PI);
            target += Coord(cos(positionAngle),sin(positionAngle)) *
                tempRadius;
        }
    }else if(swarmPosMode == "cone"){
        double startAngle = 0;
        double ahead = 300;
        double sides = 150;
        Coord mDroneCoord(cos(mDroneHeadAngle),sin(mDroneHeadAngle));
        if(swarmSize%2!=0){
            if(droneNum == (swarmSize-1)) target -= mDroneCoord*(ahead/2) *
                distanceModifier;
        }
        if(swarmSize == 10){
            if(droneNum == 9) target += (Coord(cos(mDroneHeadAngle),sin(
                mDroneHeadAngle))* ahead  - mDroneCoord*(ahead/2))*
                distanceModifier;
            if(droneNum == 8) target -= Coord(cos(mDroneHeadAngle),sin(
                mDroneHeadAngle))*(ahead/2) * distanceModifier;
        }
        switch(swarmSize){
        case 10:
        case 9:
        case 8:
            if(droneNum == 7 || droneNum == 6){
                double positionAngle = atan((sides/2)/ahead);
                double hyp = sqrt(pow(ahead,2) + pow(sides/2,2));
                positionAngle = positionAngle*(((droneNum-6)*2)-1) +
                    mDroneHeadAngle;
                target +=  (Coord(cos(positionAngle),sin(positionAngle)) *
                    hyp - mDroneCoord*(ahead/2))* distanceModifier ;
            }
        case 7:
        case 6:
            if(droneNum == 5 || droneNum == 4){
                double positionAngle = atan((sides)/(ahead));
```

```
                double hyp = sqrt(pow(ahead,2) + pow(sides,2))/3;
                positionAngle = positionAngle*(((droneNum-4)*2)-1) +
                    mDroneHeadAngle;
                target +=  (Coord(cos(positionAngle),sin(positionAngle)) *
                    hyp  - mDroneCoord*(ahead/2))* distanceModifier;
            }
        case 5:
        case 4:
            if(droneNum == 3 || droneNum == 2){//top middle
                double positionAngle = atan((sides)/(ahead));
                double hyp = sqrt(pow(ahead,2) + pow(sides,2))*2/3;
                positionAngle = positionAngle*(((droneNum-2)*2)-1) +
                    mDroneHeadAngle;
                target +=  (Coord(cos(positionAngle),sin(positionAngle)) *
                    hyp  - mDroneCoord*(ahead/2))* distanceModifier;
            }
        case 3:
        case 2:
            if(droneNum == 0 || droneNum == 1){
                double positionAngle = atan(sides/ahead);
                double hyp = sqrt(pow(ahead,2) + pow(sides,2));
                positionAngle = positionAngle*((droneNum*2)-1) +
                    mDroneHeadAngle;
                target +=  (Coord(cos(positionAngle),sin(positionAngle)) *
                    hyp  - mDroneCoord*(ahead/2))* distanceModifier;//place
                    left or right
            }
            break;
        default :
            break;
        }
    }
    swarmTarget = target;
}

void DroneMobility::surroundEfficiency(){
        double surroundDistance[10][2] = {};
        double startAngle = 0;
        double mDroneHeadAngle = atan2(mDroneHeading.y,mDroneHeading.x);
        if(swarmSize%2==0){//if swarm size is even
            startAngle = mDroneHeadAngle - PI/2;
        }else{
            startAngle = mDroneHeadAngle - PI;
```

```
}
for(int i = 0; i < swarmSize; i++){
    double radsPerDrone = 2*PI/swarmSize;
    double thisIterationRads = radsPerDrone*i;
    double positionAngle = startAngle + thisIterationRads;
    double tempRadius = sqrt(70685*swarmSize/PI);
    Coord itTarget = target + Coord(cos(positionAngle),sin(
        positionAngle)) * tempRadius;
    surroundDistance[i][0] = i;
    surroundDistance[i][1] = abs(itTarget.distance(source));
     EV_INFO << surroundDistance[i][1] << " : " ;
}
EV_INFO << endl;
{
int i, j;
for (i = 0; i < swarmSize-1; i++){
 // Last i elements are already in place
    for (j = 0; j < swarmSize-i-1; j++){
        if (surroundDistance[j][1] > surroundDistance[j+1][1]){
            double temp1 = surroundDistance[j][1];
            int temp2 = surroundDistance[j][0];
            surroundDistance[j][1] = surroundDistance[j+1][1];
            surroundDistance[j+1][1] = temp1;
            surroundDistance[j][0] = surroundDistance[j+1][0];
            surroundDistance[j+1][0] = temp2;
       }
    }
}
}
for(int i = 0; i < swarmSize; i++){
    EV_INFO << surroundDistance[i][0] << " : "<< surroundDistance[i
        ][1] << " : " ;
}
EV_INFO << endl;

int i =0, flag = 0, sent = 0;
do{
    sent = 0;
    flag++;
    for(int j =0;j<droneNum;j++){
        EV_INFO <<"s " <<s_swarmInfo[j][SWARM_POS] <<" a "<<
            surroundDistance[i][0]<<endl;
        if(s_swarmInfo[j][SWARM_POS] == surroundDistance[i][0]){
```

```
                    sent = 1;
                }
            }
            i++;
        }while(sent == 1 && flag < 10);
        s_swarmInfo[droneNum][SWARM_POS] = surroundDistance[i-1][0];
        EV_INFO <<"aiming for node " << s_swarmInfo[droneNum][SWARM_POS] <<
            endl;
        double radsPerDrone = 2*PI/swarmSize;
        double thisIterationRads = radsPerDrone*surroundDistance[i-1][0];
        double positionAngle = startAngle + thisIterationRads;
        double tempRadius = sqrt(70685*swarmSize/PI);
        target += Coord(cos(positionAngle),sin(positionAngle)) * tempRadius;
}

void DroneMobility::setAccuracyTrack(){
    if(sensorAccuracy > 100){
        sensorAccuracy = 100;
    }else{//implement random walk
        double distance = source.distance(target);
        accuracyModifier = ((100-sensorAccuracy)/100); //gives heading
            limits
        Coord direction = target - source;
        Coord limits = direction * accuracyModifier;
        double randX = uniform((-limits).x, limits.x);  //random number
            between range of sized determined by accuracy
        double randY = uniform((-limits).y, limits.y);
        double randZ = uniform((-limits).z, limits.z);
        accuracyTrack = Coord(randX, randY, randZ);
    }
}

} // namespace inet
```

D.3 DroneMobility.ned

```
package inet.mobility.single;

import inet.mobility.base.MovingMobilityBase;

simple DroneMobility extends MovingMobilityBase
{
```

```
    parameters:
        @class(DroneMobility);
        double maxSpeed @unit(mps) = default(10mps);
        double speed @unit(mps) = default(16mps);
        string sourceMobility = default("."); // the default source mobility
            is this
        string targetMobility = default(".");
        double sensorRange @unit(m) = default(100m);
        int droneNumber;
        double initialX = default(0);
        double initialY = default(0);
        double initialZ @unit(m) = default(0m);
        double range @unit(m) = default(500m);
        double acceleration @unit(mpss) = default(10mpss);
        double sensorAccuracy @unit(pc) = default(100pc);
        double turnRate @unit(radpersec) = default(10radpersec);
        int swarmSize;
        string targetTrackMode; //= default("direct");
        double initialLatitude @unit(deg) = default(nan deg);
        double initialLongitude @unit(deg) = default(nan deg);
        double initialAltitude @unit(m) = default(20m);
        double initialHeading @unit(deg) = default(0deg);
        string droneMode;
        string swarmPosMode; // = default("follow");
        double surroundRadius @unit(m) = default(10m);
        double approachDistance @unit(m) = default(0m);
        bool overshootMode = default(false);
        int iteration;
}
```

Appendix E

MaliciousMobility Code

E.1 MaliciousDroneMobility.h

```
#ifndef __INET_MALICIOUSDRONEMOBILITY_H
#define __INET_MALICIOUSDRONEMOBILITY_H

#include "inet/common/INETDefs.h"
#include "inet/mobility/base/LineSegmentsMobilityBase.h"
#include "inet/mobility/single/BonnMotionFileCache.h"

namespace inet {

/**
 * @brief Uses the BonnMotion native file format. See NED file for more info
     .
 *
 * @ingroup mobility
 * @author Chris Arnold
 */
class INET_API MaliciousDroneMobility : public LineSegmentsMobilityBase
{
  protected:
    // state
    bool is3D;
    const BonnMotionFile::Line *lines;
    int currentLine;
    double maxSpeed, speed, acceleration;
    Coord direction;
```

```
    Coord lastWaypoint;
    Coord nextWaypoint;
    Coord prevPos;
    IMobility *sourceMobility = nullptr;
    Coord source;
    int initStage;


  protected:
    virtual int numInitStages() const override { return NUM_INIT_STAGES; }

    /** @brief Initializes mobility model parameters. */
    virtual void initialize(int stage) override;

    /** @brief Initializes the position according to the mobility model. */
    virtual void setInitialPosition() override;

    /** @brief Overridden from LineSegmentsMobilityBase. */
    virtual void setTargetPosition() override;

    /** @brief Overridden from LineSegmentsMobilityBase. */
    virtual void move() override;

    virtual void computeMaxSpeed();

    void setNewWaypoint();

    bool checkWaypoint();

    void circleFunction();

    void speedModify();

  public:
    MaliciousDroneMobility();

    virtual ~MaliciousDroneMobility();

    virtual double getMaxSpeed() const override { return maxSpeed; }
};

} // namespace inet
```

```
#endif // ifndef __INET_MALICIOUSDRONEMOBILITY_H
```

E.2 MaliciousDroneMobility.cc

```
//
// 2020 Chris Arnold
//
//

#include "inet/common/INETMath.h"
#include "inet/mobility/single/BonnMotionFileCache.h"
#include "inet/mobility/single/MaliciousDroneMobility.h"
#define TOLERANCE 5
#define DEG *180/PI

namespace inet {

Define_Module(MaliciousDroneMobility);

MaliciousDroneMobility::MaliciousDroneMobility()
{
    is3D = false;
    lines = nullptr;
    currentLine = -1;
    maxSpeed = 0;
    //waypoint = Coord();
}

void MaliciousDroneMobility::computeMaxSpeed()
{
    const BonnMotionFile::Line& vec = *lines;

    double lastTime = 0;
    acceleration = vec[0];
    Coord lastPos(vec[1],vec[2],(is3D ? vec[3] : 0));
    unsigned int step = (is3D ? 4: 3);
    for (unsigned int i = step; i < vec.size(); i += step)
    {
        double elapsedTime = vec[i] - lastTime;
        Coord currPos(vec[i+1], vec[i+2], (is3D ? vec[i+3] : 0));
        double distance = currPos.distance(lastPos);
        double speed = distance / elapsedTime;
```

```
        if (speed > maxSpeed)
            maxSpeed = speed;
        lastPos.x = currPos.x;
        lastPos.y = currPos.y;
        lastPos.z = currPos.z;
        lastTime = vec[i];
    }
}

MaliciousDroneMobility::~MaliciousDroneMobility()
{
    BonnMotionFileCache::deleteInstance();
}

void MaliciousDroneMobility::initialize(int stage)
{
    LineSegmentsMobilityBase::initialize(stage);

    //EV_INFO << "initializing MaliciousDroneMobility stage " << stage <<
        endl;
    initStage = stage;
    if (stage == INITSTAGE_LOCAL) {

        //sourceMobility = getModuleFromPar<IMobility>(par("sourceMobility")
            , this);
        is3D = par("is3D");
        int nodeId = par("nodeId");
        if (nodeId == -1)
            nodeId = getContainingNode(this)->getIndex();
        const char *fname = par("traceFile");
        const BonnMotionFile *bmFile = BonnMotionFileCache::getInstance()->
            getFile(fname);
        //acceleration = 4.755;
        lines = bmFile->getLine(nodeId);
        if (!lines)
            throw cRuntimeError("Invalid nodeId %d -- no such line in file "
                "%s'", nodeId, fname);
        currentLine = 0;
        //acceleration = par("mAccel");
        //load first waypoint
        sourceMobility = getModuleFromPar<IMobility>(par("sourceMobility"),
            this);
        computeMaxSpeed();
```

```
        cCanvas *canvas = getSystemModule()->getCanvas(); // toplevel canvas
        canvas->setAnimationSpeed(1.0, this); //smooth animation
    }
}

void MaliciousDroneMobility::setInitialPosition()
{
    const BonnMotionFile::Line& vec = *lines;

        lastPosition.x = vec[1];//x
        lastPosition.y = vec[2];//y
        lastPosition.z = vec[3];//z
        //currentPos = lastPosition;
        targetPosition = lastPosition;
        speed = vec[0];
        currentLine += 4;
        maxSpeed = vec[currentLine];
        lastWaypoint = nextWaypoint;
        nextWaypoint.x = vec[currentLine + 1];
        nextWaypoint.y = vec[currentLine + 2];
        nextWaypoint.z = vec[currentLine + 3];
        //EV_INFO <<"initialise x:" << lastPosition.x << " y:" <<
            lastPosition.y <<" z:" << lastPosition.z <<" speed:" << speed <<
            endl;
        //EV_INFO <<"init-nextWaypoint:" <<nextWaypoint << endl;

    //lastWaypoint = lastPosition;
    //need to set first waypoint
}

void MaliciousDroneMobility::setTargetPosition()
{
    if(initStage < 12 && initStage >2){
        //EV_INFO << "sTP stage " << initStage << endl;
        return;
    }
    nextChange = simTime() + updateInterval;
    source = sourceMobility->getCurrentPosition();
    //EV_INFO << "next interval: " << nextChange << endl;
    const BonnMotionFile::Line& vec = *lines;
    //check if at waypoint target- get next waypoint if yes
    if(checkWaypoint()){
        setNewWaypoint();
```

```
    }
    // check if waypoint target is outside turning circle
    circleFunction();
    //adjust speed if not at correct speed
    if(speed != maxSpeed){
        speedModify();
    }


}

void MaliciousDroneMobility::speedModify(){ //mainly used to adjust speed
    for use in setTargetPosition
    //EV_INFO << "speed: " << speed << " set speed: " << maxSpeed;
    double difference = maxSpeed - speed;
    double maxAcceleration = acceleration*updateInterval.dbl();
        if(maxSpeed > speed){
            //EV_INFO << " accelerating to " << maxSpeed;

            if(speed < maxSpeed && difference >= maxAcceleration){
                if(speed + (maxAcceleration) > maxSpeed){
                    speed = maxSpeed;
                }else{
                    speed += (maxAcceleration);
                }
            }else if(speed < maxSpeed && difference < maxAcceleration){
                speed = maxSpeed;
            }
        }else if(maxSpeed < speed){
            if(speed > 0){                  //
                //EV_INFO << " decelerating to " << maxSpeed;
                if(speed - (maxAcceleration) < 0){
                    speed = 0;
                }else if(speed > maxSpeed && difference > maxAcceleration){
                }else{
                    speed -= (maxAcceleration);//same here
                }
            }
        }
        //EV_INFO << " new speed: " << speed << endl;

}
```

```
void MaliciousDroneMobility::circleFunction(){
    double radius = speed*speed/acceleration;//meters
    double theta = (speed*updateInterval.dbl()*acceleration/speed);
    double maxTurnAngle = PI/2 - acos(sqrt(1-cos(theta)/2));
    //determine the relative angle of the mDrone
    Coord droneHeading = (source - prevPos);
    double heading = (atan2(-droneHeading.y,droneHeading.x));//rads
    Coord mHeadingCoord = nextWaypoint - source;
    //EV_INFO << "target: " <<nextWaypoint<< " source: " <<source<< "
        mHeadingCoord: " <<mHeadingCoord<<" prevPos: " <<  prevPos << endl;
    double angleToTarget = (atan2(-mHeadingCoord.y,mHeadingCoord.x));
    double difference = angleToTarget - heading;
    //EV_INFO << " heading: " <<heading DEG<< " angleToTarget: " <<
        angleToTarget DEG<< " difference: " <<difference DEG<< endl;
    //EV_INFO << "difference: " << difference DEG;
    if(difference > PI){
        difference =  difference-(2*PI);
        //EV_INFO << "difference changed to : " << difference DEG;
    }else if(difference < -PI){
        difference = difference + (2*PI);
        //EV_INFO << "difference changed to : " << difference DEG;
    }
    //EV_INFO << endl;
    //EV_INFO << " angleToTarget: " << angleToTarget DEG << " heading: " <<
        heading DEG << " difference: " << difference DEG<< " maxTurnAngle: "
         << maxTurnAngle DEG << endl;
    if(abs(difference) > maxTurnAngle){ //target outside turn arc
        double turnAngle = 0;
        if(speed == 0){
            //speedModify("FASTER");
        }else if(difference > 0){ //target to left of drone
            turnAngle = heading + maxTurnAngle;
            //EV_INFO << "turn left" << endl;
        }else{
            turnAngle = heading - maxTurnAngle;
            //EV_INFO << "turn right" << endl;
        }
        double xMove = cos(turnAngle);
        double yMove = -sin(turnAngle);
        direction = Coord(xMove, yMove,0.0).normalize();
    }else{ //within turn arc
        if(speed == 0)
            //speedModify("FASTER");
```

```
        double xMove = cos(difference);
        double yMove = -sin(difference);
        //direction = Coord(xMove, yMove,0.0).normalize();
        direction = (nextWaypoint - source).normalize();
    }

}

void MaliciousDroneMobility::setNewWaypoint(){
    const BonnMotionFile::Line& vec = *lines;
    currentLine += 4;
    lastWaypoint = nextWaypoint;
    maxSpeed = vec[currentLine];
    nextWaypoint.x = vec[currentLine + 1];
    nextWaypoint.y = vec[currentLine + 2];
    nextWaypoint.z = vec[currentLine + 3];
    //EV_INFO << "setWaypoint:"<< nextWaypoint << " maxSpeed: "<< maxSpeed
        << endl;

}

bool MaliciousDroneMobility::checkWaypoint(){
    bool xTest = (source.x > nextWaypoint.x-TOLERANCE && source.x <
        nextWaypoint.x+TOLERANCE);
    bool yTest = (source.y > nextWaypoint.y-TOLERANCE && source.y <
        nextWaypoint.y+TOLERANCE);
    bool zTest = (source.z > nextWaypoint.z-TOLERANCE && source.z <
        nextWaypoint.z+TOLERANCE);
    //EV_INFO << " xTest: " << xTest<< " yTest: " << yTest<< " zTest: " <<
        zTest << endl;
    if(xTest && yTest && zTest){
        return true;
    }
    return false;
}

void MaliciousDroneMobility::move()
{
    simtime_t now = simTime();
    if (now == nextChange) {
        lastPosition = targetPosition;
        //currentPos = source;
        //EV_INFO << "reached current target position = " << lastPosition <<
```

```
            endl;

        setTargetPosition();

        //EV_INFO << "source test: " << source << endl;
        lastVelocity = (targetPosition - lastPosition).normalize() *(speed *
            updateInterval.dbl());
        //lastVelocity = (targetPosition - lastPosition) / (nextChange -
            simTime()).dbl();
        //EV_INFO << "lastPoistion: " <<lastPosition<< " direction: " <<
            direction << " speed: " << speed<< endl;
        targetPosition = lastPosition + direction * (speed * updateInterval.
            dbl());
        //EV_INFO << "new target position = " << targetPosition << ", next
            change = " << nextChange << endl;
        prevPos = source;
    }
    //LineSegmentsMobilityBase::move();
    //raiseErrorIfOutside();
}

} // namespace inet
```

E.3 MaliciousDroneMobility.ned

```
package inet.mobility.single;

import inet.mobility.base.MovingMobilityBase;

simple MaliciousDroneMobility extends MovingMobilityBase
{
    parameters:
        bool is3D = default(false); // whether the trace file contains
            triplets or quadruples
        string traceFile; // the BonnMotion trace file
        int nodeId; // selects line in trace file; -1 gets substituted to
            parent module's index
        string sourceMobility = default("."); // the default source mobility
            is this
        double mAccel = default(6.5);
        @class(MaliciousDroneMobility);
}
```

Appendix F

DroneNetwork.ned Code

```
package inet.examples.testInet;


import inet.node.inet.StandardHost;
import inet.visualizer.integrated.IntegratedVisualizer;
import inet.node.inet.WirelessHost;
import inet.node.inet.AdhocHost;
import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
import inet.node.inet.AdhocHost;
import inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
//import inet.visualizer.contract.IIntegratedVisualizer;
//import inet.common.geometry.common.OsgGeographicCoordinateSystem;
//import inet.environment.common.PhysicalEnvironment;

network DroneNetwork
{
    parameters:
        int numDrones;
        @figure[drone0](type=rectangle; pos=10,50; size=50,50; );
        @figure[drone0.label](type=text; pos=20,80; text=placeholder);
        @figure[drone1](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone1.label](type=text; pos=20,80; text=placeholder);
        @figure[drone2](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone2.label](type=text; pos=20,80; text=placeholder);
        @figure[drone3](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone3.label](type=text; pos=20,80; text=placeholder);
```

```
        @figure[drone4](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone4.label](type=text; pos=20,80; text=placeholder);
        @figure[drone5](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone5.label](type=text; pos=20,80; text=placeholder);
        @figure[drone6](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone6.label](type=text; pos=20,80; text=placeholder);
        @figure[drone7](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone7.label](type=text; pos=20,80; text=placeholder);
        @figure[drone8](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone8.label](type=text; pos=20,80; text=placeholder);
        @figure[drone9](type=rectangle; pos=10,50; size=1,1; );
        @figure[drone9.label](type=text; pos=20,80; text=placeholder);
        @display("bgb=20800,9550;bgi=background/townsvilleScale2");
    submodules:
        visualizer: IntegratedVisualizer {
            parameters:
                @display("p=0,50");
        }
//          coordinateSystem: OsgGeographicCoordinateSystem {
//              parameters:
//                  @display("p=12868.674,3622.9373");
//          }
        drone[numDrones]: StandardHost {
            parameters:
                @display("p=382,50;i=misc/drone,blue");



        }
        radioMedium: Ieee80211ScalarRadioMedium {
            @display("p=12868.674,5217.03");
        }
        mDrone: StandardHost {
            @display("i=misc/drone,red");
        }
//          physicalEnvironment: PhysicalEnvironment {
//              @display("p=13129.525,2115.7954");
//          }
        configurator: Ipv4NetworkConfigurator {
            @display("p=13129.525,550.68646");
        }

}
```

Appendix G

omnetpp.ini Code

```
[General]

image-path = "/home/user/Integration/inet/showcases/general/simpleMobility"
user-interface = Qtenv  # Tkenv does not support 3D visualization

**.arp.typename = "GlobalArp"
# Visualizer settings
#*.visualizer.osgVisualizer.typename = "IntegratedOsgVisualizer"
#*.visualizer.osgVisualizer.sceneVisualizer.typename = "
    SceneOsgEarthVisualizer"
#*.visualizer.osgVisualizer.sceneVisualizer.mapFile = "boston.earth"
# Coordinates of the scene origin on the map
#*.coordinateSystem.sceneLongitude = 150.8deg richmond
##*.coordinateSystem.sceneLatitude = -33.6deg
#*.coordinateSystem.sceneLongitude = 146.766402deg #townsville
#*.coordinateSystem.sceneLatitude = -19.248006deg
**.networkConfiguratorModule = ""
*.visualizer.*.mobilityVisualizer.displayMobility = true # master switch
*.visualizer.*.mobilityVisualizer.displayPositions = true
*.visualizer.*.mobilityVisualizer.displayOrientations = true
*.visualizer.*.mobilityVisualizer.displayVelocities = true
*.visualizer.*.mobilityVisualizer.displayMovementTrails = true
*.visualizer.*.mobilityVisualizer.positionCircleRadius = 4

#*.visualizer.*.mediumVisualizer.displayCommunicationRanges = true
```

```
#[Config Mtest]
#network = EarthVisualizationShowcase
## Visualizer settings
#*.visualizer.osgVisualizer.typename = "IntegratedOsgVisualizer"
#*.visualizer.osgVisualizer.sceneVisualizer.typename = "
    SceneOsgEarthVisualizer"
#*.visualizer.osgVisualizer.sceneVisualizer.mapFile = "boston.earth"
#
## Coordinates of the scene origin on the map
#*.coordinateSystem.sceneLongitude = 150.8deg
#*.coordinateSystem.sceneLatitude = -33.6deg


[Config DroneConfig3D]
#network = EarthVisualizationShowcase
network = DroneNetwork

#*.mDrone.osgModel = "3d/drone.ive.100.scale.0,0,90.rot"
#*.drone[*].numApps = 1
#*.drone[*].app[0].typename = "UdpBasicApp"
#*.visualizer.*.mediumVisualizer.displayCommunicationRanges = true
#*.drone[*].wlan[*].radio.transmitter.power = 10mW
#*.drone[*].forwarding = true
#*.drone[*].wlan[*].mgmt.typename = "Ieee80211MgmtAdhoc"
#*.drone[*].wlan[*].agent.typename = ""

#*.numDrones = 2 #max 10 drones
#*.drone[*].mobility.swarmSize = 2 #must match numDrones

*.drone[*].mobility.typename = "DroneMobility"


*.drone[*].mobility.surroundRadius = 150m

*.mDrone.mobility.typename = "MaliciousDroneMobility"
#**.mDrone.mobility.traceFile = "maliciousDrone.movements"
**.mDrone.mobility.is3D = true
**.mDrone.mobility.nodeId = -1
*.drone[*].mobility.maxSpeed = 20mps
*.drone[*].mobility.speed = 0mps
*.drone[*].mobility.sensorRange = 1000m#680m
*.drone[*].mobility.acceleration = 4mpss
#*.drone[0].mobility.sensorAccuracy = 50pc
```

```
#*.drone[2].mobility.sensorAccuracy = 90pc
#*.drone[3].mobility.sensorAccuracy = 90pc
*.drone[*].mobility.overshootMode = true

#vertical acceleration = 3mpss
**.mobility.margin = 0m
**.mobility.speedStdDev = 0.5mps
**.mobility.angleStdDev = 0.5deg
**.mobility.alpha = 0.5
*.drone[0].mobility.droneNumber = 0
*.drone[1].mobility.droneNumber = 1
*.drone[2].mobility.droneNumber = 2
*.drone[3].mobility.droneNumber = 3
*.drone[4].mobility.droneNumber = 4
*.drone[5].mobility.droneNumber = 5
*.drone[6].mobility.droneNumber = 6
*.drone[7].mobility.droneNumber = 7
*.drone[8].mobility.droneNumber = 8
*.drone[9].mobility.droneNumber = 9

*.drone[*].mobility.initialZ = 20m

#*.drone[*].app[0].destPort = 0
#*.drone[*].app[0].messageLength = 0B
#*.drone[*].app[0].sendInterval = 0s

*.drone[*].mobility.targetMobility = "^.^.mDrone.mobility"




# Physical environment settings
#*.physicalEnvironment.coordinateSystemModule = "coordinateSystem"
#*.physicalEnvironment.config = xmldoc("obstacle.xml")

# Visualizer settings
#*.visualizer.osgVisualizer.sceneVisualizer.sceneShading = true
#*.visualizer.osgVisualizer.sceneVisualizer.sceneColor = "#000000"
#*.visualizer.osgVisualizer.sceneVisualizer.sceneOpacity = 1

# Coordinate system settings
#*.coordinateSystem.sceneAltitude = 0m
#*.coordinateSystem.sceneHeading = 68.3deg
```

```
#*.drone[*].mobility.coordinateSystemModule = "coordinateSystem"


# Node position settings
#*.drone[0].mobility.initialLatitude = -19.24809deg #-19.239009deg
#*.drone[0].mobility.initialLongitude = 146.76649deg #146.752424deg
#*.drone[*].mobility.initialAltitude = 20m

#
#*.drone[1].mobility.initialLatitude = -19.234457deg
#*.drone[1].mobility.initialLongitude = 146.769012deg
#
#*.drone[2].mobility.initialLatitude = -19.240155deg
#*.drone[2].mobility.initialLongitude = 146.776072deg
#
#*.drone[3].mobility.initialLatitude = -19.251766deg
#*.drone[3].mobility.initialLongitude = 146.774744deg
#
#*.drone[4].mobility.initialLatitude = -19.263054deg
#*.drone[4].mobility.initialLongitude = 146.769320deg
#
#*.drone[5].mobility.initialLatitude = -19.250118deg
#*.drone[5].mobility.initialLongitude = 146.750906deg
#
#*.drone[6].mobility.initialLatitude = -19.250118deg
#*.drone[6].mobility.initialLongitude = 146.750906deg
#
#*.drone[*].mobility.initialLatitude = -19.250130deg
#*.drone[*].mobility.initialLongitude = 146.750906deg

#*.drone[0].mobility.initialX = 5169 #-19.239009deg
#*.drone[0].mobility.initialY = 1473 #146.752424deg
#*.drone[*].mobility.initialAltitude = 20m

#
#*.drone[1].mobility.initialX = 9724
#*.drone[1].mobility.initialY = 3893
#
#*.drone[2].mobility.initialX = 8152
#*.drone[2].mobility.initialY = 5661
#
#*.drone[3].mobility.initialX = 10028
```

```
#*.drone[3].mobility.initialY = 5161
#
#*.drone[4].mobility.initialX = 7780
#*.drone[4].mobility.initialY = 4393
#
#*.drone[5].mobility.initialX = 8860
#*.drone[5].mobility.initialY = 5941
#
#*.drone[6].mobility.initialX = 9004
#*.drone[6].mobility.initialY = 3581

*.drone[*].mobility.initialX = 8948
*.drone[*].mobility.initialY = 4769
record-eventlog = true
sim-time-limit = 1800s
*.mDrone.mobility.typename = "MaliciousDroneMobility"
#**.mDrone.mobility.traceFile = "M2.movements"
**.mDrone.mobility.is3D = true
**.mDrone.mobility.nodeId = -1
*.drone[*].mobility.targetTrackMode = "direct" #direct or PN

[Config DroneNetworkFollow]
extends = DroneConfig3D
#*.drone[*].mobility.droneMode = "direct" #direct or swarm
*.drone[*].mobility.swarmPosMode = "follow"
#repeat = 10

*.numDrones = ${N=1..10} #${N_drones=1..10} #max 10 drones
*.drone[*].mobility.swarmSize = ${N} #must match numDrones
*.drone[*].mobility.iteration = ${N2=1..10}


[Config DroneNetworkCone]
extends = DroneConfig3D
*.drone[*].mobility.droneMode = "swarm" #direct or swarm
*.drone[*].mobility.swarmPosMode = "cone"

*.numDrones = ${N=1..10} #${N_drones=1..10} #max 10 drones
*.drone[*].mobility.swarmSize = ${N} #must match numDrones
*.drone[*].mobility.iteration = ${N2=1..10}


[Config DroneNetworkSurround]
```

```
extends = DroneConfig3D
*.drone[*].mobility.droneMode = "efficient" #direct or swarm
*.drone[*].mobility.swarmPosMode = "surround"
**.mDrone.mobility.traceFile ="C2.movements"
*.numDrones = ${N=1..10} #${N_drones=1..10} #max 10 drones
*.drone[*].mobility.swarmSize = ${N} #must match numDrones
*.drone[*].mobility.iteration = ${N2=1..10}
#
#[Config DroneNetwork3DlinearPN]
#extends = DroneConfig3D
#*.drone[*].mobility.targetTrackMode = "PN" #direct or PN
#
[Config DroneNetworkAll]
extends = DroneConfig3D
*.drone[*].mobility.droneMode = "direct" #direct or swarm

#repeat = 10

**.mDrone.mobility.traceFile =  ${N4="M1.movements","M2.movements","M3.
    movements","C1.movements","C2.movements","C3.movements"}
*.numDrones = ${N=1..10}#${N=1..10} #${N_drones=1..10} #max 10 drones
*.drone[*].mobility.swarmSize = ${N} #must match numDrones

*.drone[*].mobility.swarmPosMode =${N3="follow","surround","cone"}
*.drone[*].mobility.iteration = ${N2=1..10}
```