CRANFIELD UNIVERSITY

JORGE DURÁN ZAFRILLA

THE DEVELOPMENT AND GENERATION OF GNSS RF SCENARIOS FOR THE ANALYSIS AND OPTIMISATION OF DRONE PERFORMANCE IN A GEO-FENCED ENVIRONMENT.

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicles Dynamics & Control

MSc
Academic Year: 2017–2018

Supervisor: Dr Ivan Petrunin & Prof. Rafal Zbikowski
August 2018

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicles Dynamics & Control

MSc

Academic Year: 2017–2018

JORGE DURÁN ZAFRILLA

The development and generation of GNSS RF scenarios for the analysis and optimisation of drone performance in a Geo-Fenced environment.

Supervisor: Dr Ivan Petrunin & Prof. Rafal Zbikowski
August 2018

This thesis is submitted in partial fulfillment of the requirements for the degree of MSc Autonomous Vehicles Dynamics & Control.

Abstract

The aims of this Individual Research Project are the development of a Geo-Fencing software able to generate GNSS Geo-Fences with different shapes and conditions and the study of the UAV performance inside Geo-Fenced environments. In order to accomplish these two objectives, three different software modules have been developed, the main module is the Geo-Fencing tool and two support modules used to do the performance analysis.

Geo-Fencing is a key field that will help the development of Unmanned Traffic Management in the following years. The software developed in this project generates circular and polygonal Geo-Fences and adds to each GF an initial and final time, which is the main novelty and the reason why this project is interesting for the development of UTM. This software also receives a positioning of a UAV and displays in real time a message if the vehicle is inside the GF or not.

To study the drone performance in this type of environments, a motion planner and Scheduler able to deal with the UTM have been also developed within this project. The motion planner is a simple trajectory generator that can be used in the loop and also can follow a path plan. This path plan is generated by the Scheduler mentioned above. This software generates a path inside a chosen GF with a given mission (Boundary and interior surveillance mission). With this path plan two different performance analysis are done in this project.

The purpose of the first analysis is to study the minimum offset required with the boundary of a given GF in order to guarantee that the UAV is inside the GF at any moment taking into account the GNSS accuracy. This offset is calculated for different speeds. And from the analysis, it is proved that the offset required increases with the speed and the interior surveillance mission requires more offset because it performs more turns than the boundary surveillance mission.

In the second study, it is shown the influence of Multi-path in the GNSS accuracy within Geo-Fenced urban environments, concluding that only GNSS is not enough accurate to perform missions inside urban environments.

Keywords

Geo-Fencing; GNSS; GPS; Path Planning; UAV; UAS; Autonomous; UTM; Drone Performance; Sureillance Mission; Simulator; Path Following; GNSS simulator; Multi-path.

Contents

List of Figures

List of Tables

List of Abbreviations

ADS-B	Automatic dependent surveillancebroadcast
ATM	Air Traffic Management
COM	Communication port
CWAAS	Canada Wide Area Augmentation System
DGNSS	Differential GNSS
DOP	Dilution of Precision
ECEF	Earth-Centered, Earth-Fixed
EGNOS	European Geostationary Navigation Overlay Service
FAA	Federal Aviation Administration
GAGAN	GPS Aided Geo Augmented Navigation
GF	Geo-Fence
GLONASS	Global'naya Navigatsionnaya Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphic User Interface
HDOP	Horizontal Dilution of Precision
ICAO	International Civil Aviation Organization
IP	Internet Protocol
IRNSS	Indian Regional Navigation Satellite System
LLA	Latitude, Longitude, Altitude
MSAS	Multi-functional Satellite Augmentation System

MSL	Mean sea level
NED	Nort-East-Down
NMEA	National Marine Electronic Association)
PDOP	Position Dilution of Precision
PNT	Positioning, navigation and timing
PPS	Precise. Positioning Service
QZSS	Quasi-Zenith Satellite System
RF	Radio Frequency
SATM	School of Aerospace, Technology and Manufacturing
SBAS	Satellite Based Augmentation System
SNAS	Satellite Navigation Augmentation System
SPS	Standard Positioning Service
TCP	Transmission-Control-Protocol
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
USB	Universal Serial Bus
UTC	Coordinated Universal Time
UTM	Unmanned Traffic Management
VTP	Virtual Target Point
WAAS	Wide Area Augmentation System

Acknowledgements

The development and final achievement of this Individual Research Project would not have been possible without the invaluable help of all the people that supported me during this whole year in Cranfield University.

Firstly, I would like to start showing my entire gratitude to my supervisors Dr Ivan Petrunin and Prof. Rafal Żbikowski for their interest, patience and enthusiastic support during these almost 7 months. I am grateful for their help provided during the development of this thesis in terms of meetings, emails, advice, providing literature, explanations and tens of revisions for every presentation and report.

I want now to express my most sincere gratitude to Spirent, the company that has been supporting and sponsoring this project from the very beginning until its late end. They have been providing the specific (and expensive) hardware and software needed for the success of the project, but also the ideas and what is more important, answers and solutions when using their complex equipment. Especially I want to say thanks to Ricardo Verdaguer, who has been supervising the work from the company, spending his valuable time visiting Cranfield (more than 5 hours trip) more than twice to help and solve any issue we could have during our research. My gratitude for him is infinite.

In addition, I want to show my gratitude to my Spanish University UPV which gave me the opportunity to enjoy this experience studying abroad during the last academic year of my studies. Also, I need to thank the Erasmus+ program for the generous scholarship.

I do not want to forget to thank the people that made of me the person I am in this moment. Many thanks to my parents, my whole family, my friends and Cristina, who have been supporting and encouraging me through the years.

Finally, and leaving academic things aside, I want to sincerely thank all the people that made the Cranfield experience something unforgettable. Many thanks to the Cranfield Handball Wolves team, for teaching me this wonderful sport and helping me to release adrenaline during the development of the project. And, of Course, many thanks to Nil, Olatz, Itxaso, Dani and Marc, for their unconditional support during this long and unforgettable year.

Chapter 1

Introduction

During the last decade, the evolution of technology has improved the industry of unmanned vehicles. Moreover, the use of autonomous technologies is increasing considerably due to the large number of applications that it has. Autonomous vehicles include any type of platform such as UAVs, UGVs, etc. It is not difficult to imagine a future where every repetitive, boring or risky task is done by an autonomous system. Examples of this kind of applications are search and rescue, delivery process of packages, explosive deactivation, etc.

However, having hundreds or thousands of UAVs flying around autonomously needs the development and improvement of safety in order to guarantee the integrity of people and the success of the mission developed by each vehicle. In order to deal with this amount of traffic, the Unmanned Traffic Management (UTM) is being developed.

One of the most useful tools in UTM is Geo-Fencing. A Geo-Fence is a boundary region of interest in a geographical region and is used to increase the safety of UAS operations and the security of any particular ground area (Airfields, areas with a big number of people, Military areas). But Geo-Fencing cannot work with an accurate positioning. To guarantee this accuracy, GNSS can be used together with its different constellations, signals and augmentation systems.

Taking into account this background the aims of the project are:

- To develop a library of Geo-Fencing compatible with Spirent Simulator
- To develop a UAV Scheduler able to deal with UTM in a Geo-Fenced environment
- To study the performance of UAVs during missions inside a Geo-Fence

The scope of this project is to study the performance of UAVs in Geo-Fenced environments using GNSS positioning. To do so, the simulator of GNSS GSS 7000 provided by Spirent has been used to simulate the GNSS RF. All the test performed during this project have been done using only software and simulations. UAVs and Geo-Fences were simulated using different programs and Graphic User Interfaced developed for this project.

After the development of this software, several tests are will be performed to test the performance inside Geo-Fences. The first analysis is a study of the minimum offset with the boundary of the Geo-Fence required to guarantee that the UAVs are always inside the chosen area while performing a mission. The second analysis is the study of the influence of Multi-Path during UAV missions inside Geo-Fences and it was done using the software Sim3D provided by Spirent.

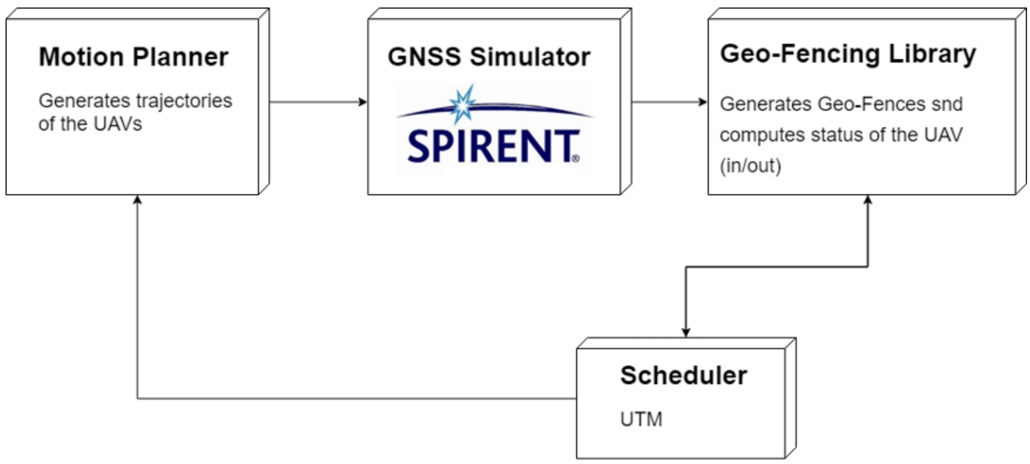


Figure 1.1: Flow diagram of the whole project

The main structure of the project is shown in 1.1. A motion planner deals with the generation of the trajectories of the UAVs, and send its motion commands to the GNSS simulator. The simulator generates the RF signal which is read by a GNSS receiver that computes the real position of the vehicle and sends this positioning to the GF library. In the GF library the GFs are created and then, once the positioning is being received, computes if the vehicle is inside or outside the GFs created.

Finally, the scheduler deals with the UTM, it knows the location of the GFs, and generates the UAVs, with a given mission inside a GF. Then computes the path planning of the selected mission in the chosen area and runs the motion planner to do the path following of the path created.

Chapter 2

Literature Review

2.1 Unmanned Aerial Systems (UAS) and Unmanned Traffic Management (UTM)

UAS of any type, (fixed-wing, multi-copters, etc), with different degrees of autonomy, are one of the most innovative industries these days (van Blyenburgh, 1999). The evolution of UAS in addition to the increase of the quality and the reduction of the cost of sensors of any type give to the UAS industry hundreds of possible applications for both civil and military areas.

Examples of this kind of operations are mapping (Samad et al., 2013), surveillance, aerial photography or filming, dull, dirty and dangerous missions, etc. However, new technologies need new regulations. Every aerial vehicle must follow some standard and requirements given by governments and agencies (ICAO, 2011). Furthermore, because of the increasing number of UAS, it is necessary to manage its traffic independently if they are using the same airspace of manned aircraft or not.

The idea of UTM comes from the well-known Air Traffic Management system that we have since the air collision over the Grand Canyon in 1956. Kopardekar (2014*b*) and Kopardekar (2014*a*) explain the current situation of UTM and its perspective for the short and long term. The objectives of UTM are: reduce the risk of accidents, avoid severe weather areas, increase the safety of missions and avoid unauthorized use of airspace.

The goal of UTM is to get UAS operations with safety and efficiency in a low-altitude airspace. The development of UTM is necessary to increase the number of operations of UAS and it is based on the rules of ATM. In the short-term (1-5 years) the goal of UTM is to manage with several UAVs in a low-altitude airspace and the steps to follow are clearly defined in [10]. The long-term goal is to increase the autonomy of UTM for a larger number of operations in a very dense air-space. The main goal of UTM is to achieve a completely unmanned management system.

One of the first steps of the UTM is to delimitate the airspace to guarantee safety operations of UAVs. Geo-fencing is the application that enables this delimitation.

2.2 Geo-Fencing

Geo-Fencing is defined as the boundaries of a region in a geographical area (Pratyusha, 2015). Geo-Fencing helps Control Stations to increase the safety of operations managed by UTM. When a vehicle enters or leaves a defined region, the user receives an alert. For example, in a completely autonomous UTM, the Control Station would take the control of the vehicle if it enters an unauthorized region until the vehicle is outside the region.

Regarding the shape of the regions, there exist two different types of Geo-Fencing. Polynomial Geo-Fencing is obtained from different selected points and is formed by the region inside these points. In order to determine whether a vehicle is inside a polynomial region or not, an algorithm is explained in (Pratyusha, 2015). Circular Geo-Fencing is given by a point and a radius. With this shape determining if a point is inside the circle is trivial.

Using different coordinate systems is possible to increase the accuracy of Geo-Fencing. For example, for small areas, the NED coordinate system can be more useful than ECEF or LLA. Therefore, conversions between systems are important taking into account that GNSS technology usually works with ECEF coordinates.

The use of GNSS is necessary to obtain accurate positioning for both UAVs and Geo-Fences (Stevens et al., 2015). Furthermore, GNSS technology does not only provide accurate positioning, but also a highly precise time which will help the development of 4D Geo-Fencing.

The papers of Cai et al. (2011) and Koks (n.d.) explain the different conversions between systems of references that are used in this project. The main systems used are NED (Local, North, East, Down) for defining the Geo-Fencing, ECEF (Global, Earth Centred Earth Fixed) and LLA (Global, Longitude, Latitude and Altitude) used by GNSS.

2.3 Positioning, Navigation and timing techniques (PNT) - GNSS, ADS-B

Satellite navigation has been used for aerial navigation for the last 50 years. However, in the last ten years, the development of differential satellite navigation has increased the accuracy of this technology. Despite this, aviation still uses ground-based navigation systems due to the slow and careful process that any innovation in air navigation must follow in order to guarantee enough safety in operations. The main four challenges of GNSS navigation for aircraft guidance are accuracy, integrity, continuity and availability (Blanch et al., 2012).

Global Navigation Satellite Systems existing currently are: NAVSTAR-GPS (USA, Military), GLONASS (Russia, Military), Galileo (Europe, Civil), Beidou (China), QZSS(Japan) and IRNSS (India).

As it was mentioned before, GNSS technology can increase its accuracy using augmentation systems or differential GNSS. Grewal et al. (n.d.) explains the different types of

DGNSS such as Local-Area DGNSS, Wide-Area DGNSS and Space-Based Augmentation Systems (SBAS). Typically, the LADGNSS uses GBAS (Ground-Base Augmentation Systems) and is used to increase vertical accuracy during approaches and landings.

SBAS uses Geostationary satellites to send GNSS corrections wide areas. Depending on agencies and territories, there are different SBAS systems working currently. Wide-Area Augmentation System (WAAS) gives corrections of the GPS area in the USA. Europe uses the European Global Navigation Overlay System (EGNOS). GAGAN(India), SNAS (China), MSAS (Japan) and CWAAS (Canada) are the other SBAS systems used in the world.

Different RF frequencies are used in each satellite positioning system as a carrier frequency to decrease errors. The errors from GNSS come from different sources: propagation errors (ionospheric and tropospheric), receiver errors (clock, noise, resolution), Ephemeris prediction errors, satellite dependant errors (clock offset and delays) and user dynamics errors (dynamics of the antenna, etc). From all of these errors can be obtained the accuracy of the navigation obtaining then the factors called DOP (Dilution of Precision)(Sabatini et al., 2017).

GNSS is not the only solution for positioning, navigation and timing (PNT). Despite GNSS is considered the first choice because of its high accuracy, wide coverage and low cost it has some disadvantages such as interferences (due to attacks or natural events) because of its large propagation distance (Han et al., 2016). There exist some other solutions for the PNT services.

One of these new technologies is called ADS-B (Automatic Dependent Surveillance Broadcast). The ADS-B was created as a system developed by the FAA that provided weather and traffic information in-flight (Podradchik, 2015). ADS-B messages were sent from ground stations.

Nowadays, this technology provides Air Traffic Control with more accurate position of aircraft in the enroute, terminal, approach and surface environments for surveillance. Aircraft broadcasts its information (ID, position, etc) and ground stations receive this signal to broadcast them (Corporations, 2016). The main benefits of ADS-B are: the reduction of cost of surveillance (secondary surveillance radar is limited by LoS and has a high maintenance cost), the improvement of the safety of operations and can reduce delays and fuel consumption (more efficient separation between aircraft).

In addition, space-based ADS-B is being developed in order to reach worldwide coverage for the ADS-B surveillance System using the IRIDIUM satellite constellation (voice and data coverage to satellite phones). This technology is expected to work by the end of 2018 (Aireon, 2018).

2.4 GNSS simulators and GNSS issues

Taking into account the objectives of the thesis to generate GNSS scenarios to evaluate the performance of UAS in Geo-Fenced environments is necessary to use a GNSS simulator. There exists different simulators available in the market. A comparative of most of the simulators available is shown in Staff (2018). These are some of them:

- CAST-5000 GPS WAVEFRONT GENERATOR (CAST NAVIGATION)
- CLAW 18-CHANNEL REAL-TIME GPS SIMULATOR and RSR TRANSCODER GPS SIMULATOR (JACKSON LABS TECHNOLOGIES INC)
- GSS9000, CRPA TEST SYSTEM, GSS6450, GSS200D, GSS7000 (SPIRENT FEDERAL SYSTEMS)
- NCS TITAN and NAVX-NCS ESSENTIAL SIMULATORS (IFEN GMBH)
- LABSAT 3 WIDEBAND (RACELOGIC)
- SDX: SOFTWARE-DEFINED GNSS SIMULATOR (SKYDEL)

- CONSTELLATOR, ECHO (SYNTONY GNSS)
- BROADSIM and PANACEA (TALEN-X)
- ALL CONSTELLATIONS, ALL FREQUENCIES (OROLIA/SPECTRACOM)

The choice of using the Spirent GSS7000 simulator for this project was made according to two main reasons: The systems provided by the University and the sponsor of this project (Spirent).

The advantages of the Spirent simulator with respect to the other solutions available are several: Multi-Frequency, Multi-Constellations and their signals(L1, L2, L5, etc) and the update rate of 1 ms which provides more accuracy and fidelity to the simulations. In addition, these simulators can test the resilience of GNSS scenarios letting the user change any desired parameter of the simulation (orbits, antenna patterns, the power level of each frequency, etc).

Another advantages of the Spirent solution is the possibility of simulating Multi-path. The influence of Multi-Path in GNSS positioning accuracy is great, and it can produce a high loss of accuracy in presence of urban environments, where signals can be reflected in a large number of surfaces. A study of this influence is done in this project.

To understand the influence of multi-path and obtain proper conclusions after the post-process of the data obtained from the tests done using Spirent software Sim3D, the following references were used: WANG (n.d.) and Peng Xie and Petovello (2015).

2.5 Path Planning and Path Following

In order to generate the desired scenarios with both different missions (Interior and Boundary surveillance), different methods of path planning have been studied. Taking into account the complexity of the desired paths inside Geo-Fences with different shapes, several steps have been followed.

Firstly, for both missions is necessary to obtain an interior polygon or circumference with an offset of the chosen Geo-Fence. in (Cacciola, n.d.) and (Palfrader and Held, 2015) different methods of offsetting polygons are explained. However, for the scope of this project, only non-convex polygons are considered and therefore a simpler algorithm was developed with this offsetting objective.

The new interior polygon is now the path plan of the boundary surveillance mission. To calculate the list of way-points of the path plan for the interior surveillance mission an algorithm was developed following (Aslund et al., 2011) and (Kariuki et al., 2014). In these two papers are explained different approaches of algorithms able to produce a path that covers the interior of any surface.

Different approaches of path planning were considered and discarded such as the one explained in (Pala et al., 2013), and others with no polygonal approach like Dubbins path planning or Pythagorean-Hodograph path planning in (Lazarus et al., 2010), (Farouki, n.d.) and (Shanmugavel et al., 2007).

Finally, the algorithm chosen to perform the path following is a Carrot-Chasing algorithm explained in (Seo, n.d.).

The most common positioning system is GNSS, therefore, this technology would be the one chosen in order to succeed with the aims of the project. In addition, Geo-Fencing is a field of knowledge that will become more important in the near future to develop UTM tasks. Therefore, the development of a Geo-Fencing software would be a very useful tool for the future of UTM.

After studying the current state of the art, the expected contributions to knowledge of this thesis are:

- The addition of GPS time to the Geo-Fences. (GFs managed by UTM can have initial and final time)

- Development of a Scheduler able to deal with several UAVs and Geo-Fences present in an airspace
- The study of the minimum offset required
- The influence of Multi-Path in Geo-Fencing

Therefore, taking into account the current State of the Art, the objectives of the project are defined as:

- To construct a motion planner in C++ to simulate UAV trajectories.
- To construct a GUI to set different types of Geo-Fences.
- To build an algorithm which compares relative position of UAV with Geo-Fences with the addition of GPS time.
- To develop a minimum required offset analysis for different scenarios
- To develop a study of the influence of Multi-Path for different scenarios

Chapter 3

Methodology

Now that the objectives and the scope of the project have been presented and the literature review has been explained, the development of the whole project is shown during this chapter. First of all, the hardware and software used will be explained, then the chapter will focus on the flow diagram of the project, to finish explaining all the developed software mentioned in the introduction.

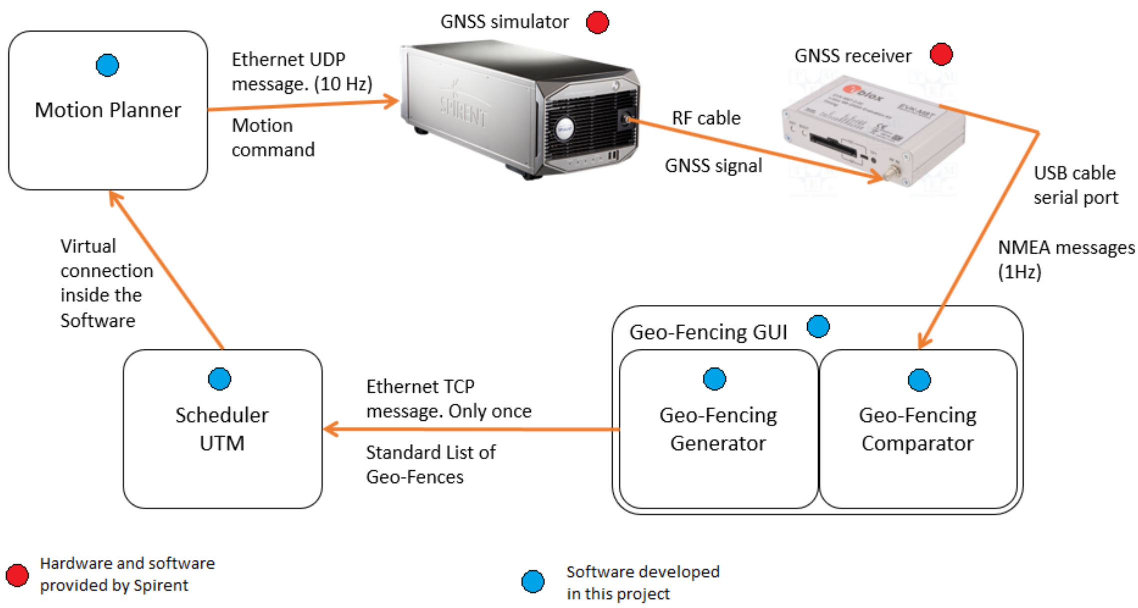


Figure 3.1: Diagram of the hardware and software used and connections between them

All the software involved during this project and the connections between them are shown in Figure 3.1. Then, every single part is explained individually, being more accurate and precise when explaining the software developed by me for this project.

Once all the software and hardware involved and all the developed software is explained, the experiments done to test all this software and the performance of the UAVs using this GF software will be explained and analyzed in the following chapter.

3.1 Hardware setup and software environment

3.1.1 GNSS simulator - GSS7000 (Spirent)

The GSS7000 series is a Multi-GNSS Constellation Simulator with multi-frequency (L2/B2, L5/E5) capability for R&D, verification testing. The simulator was provided by Spirent to Cranfield University with R&D purposes and is located in the Aerospace autonomy Laboratory in the AIRC building.

,

Figure 3.2: External aspect of the GSS 7000

In Figure 3.2 is shown the external aspect of the simulator. The remote commands are sent to the simulator with an Ethernet connection from the computer where the Motion Planner is being executed to the simulator port that is located in the rear part of the box.

Once the simulator generates the whole GNSS RF signal it is sent through the RF out port in the front side of the box to the GNSS receiver.

The simulator generates an RF GNSS signal of any scenario. The software environment used by the simulator is called SimGEN - PosApp and it is sown in Figure 3.3. As it is seen in the image, the amount of parameters that can be changed in the simulator is huge: Constellations and frequencies used, type of vehicle (car, ship, space shuttle, aircraft or remote vehicle), antenna pattern and antenna parameters in the vehicle, date of the simulation, power of the signal and a lot of the parameters each satellite (orbits, etc.).

The vehicle type used for this project is a remote vehicle. This means that the motion commands of the vehicle we are testing are generated outside this software and sent to them by standard UDP messages explained in the SimREMOTE Manual provided with the simulator.

,

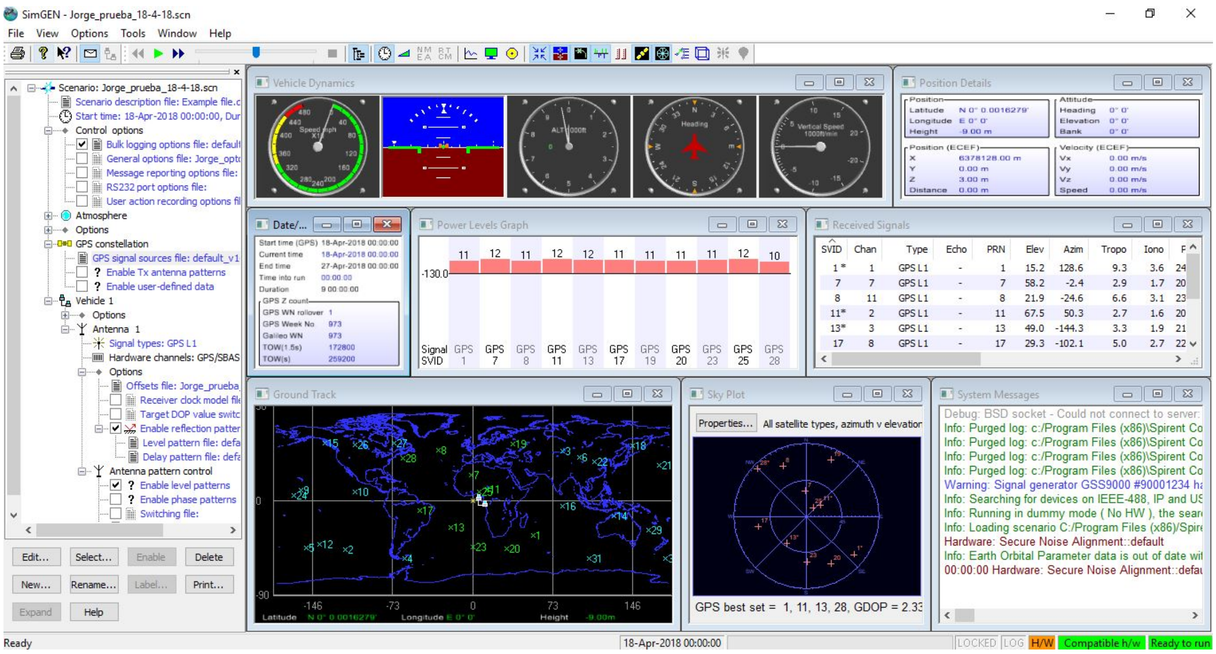


Figure 3.3: Screen-shot of SimGEN - PosApp

Most of the tools and parameters that can be selected and modify in SimGEN are explained in Chapter 5 where the parameters of the scenarios of the simulations are shown.

3.1.2 GNSS receiver - U-blox EVK-M8N

A GNSS receiver is an electronic device that receives RF signals from the constellations (in our case signals are generated by the simulator) and digitally process them to calculate position, velocity and time (time is the receiver time, so its accuracy depends on the accuracy of its clock).

U-blox EVK-M8N is an evaluation kit of the M8N GNSS receiver. It is used together with a PC to evaluate the performance of real vehicles or in laboratories using GNSS simulators. It is used with the software U-Center, which receives the NMEA messages calculated by the receiver and displays the desired parameters to test the performance of the simulations. The receiver is shown in Figure 3.4.

The connection between the receiver and the PC is done by a USB cable and a serial port as it is shown in Figure 3.1.

,

Figure 3.4: Receiver U-Blox EVK-M8N. (*EVK-8/EVK-M8*, 2018)

Chapter 4

Design of the Geo-Fencing software

All the software explained in this section has been developed in Managed C++ using Visual Studio 2017 and Matlab R2017b. C++ was chosen because the GNSS simulator and most of the Spirent software is developed in C++ and using it here makes this project more compatible with it. Matlab is used only for reading the GNSS receiver because it had some errors with C++ and to do the post process of all the data taken during the performance analysis.

The developed software for this project was explained in Figure 3.1. The main program is used to create GFs and give feedback to the user if a Vehicle invades a GF. The rest of the software was created to support the GF program in order to do the performance analyis.

In this section is presented the most important part of this project. The development of this Geo-Fencing library is one of the objectives of the thesis and was required by Spirent. The requirements for this software are:

- Generation of Geo-Fences
- Compatible with different shapes of GF:

    Polygonal

    Circular

- Possibility of adding initial and/or final time to the GFs (Geo-Fencing 4D)
- Compute if a vehicle is inside or outside the GF
- Develop a GUI to create GF and possibility of reading them from *.txt* and *.kml*

Once again the detailed explanation of the developed software will be based on the possible actions that the user can take in the Graphic User Interface (Buttons, text-boxes, etc). The design of the GUI is shown in Figure 4.1. The full GUI can be divided into two clearly separate parts: the **GF creator** and the **GF comparator**.

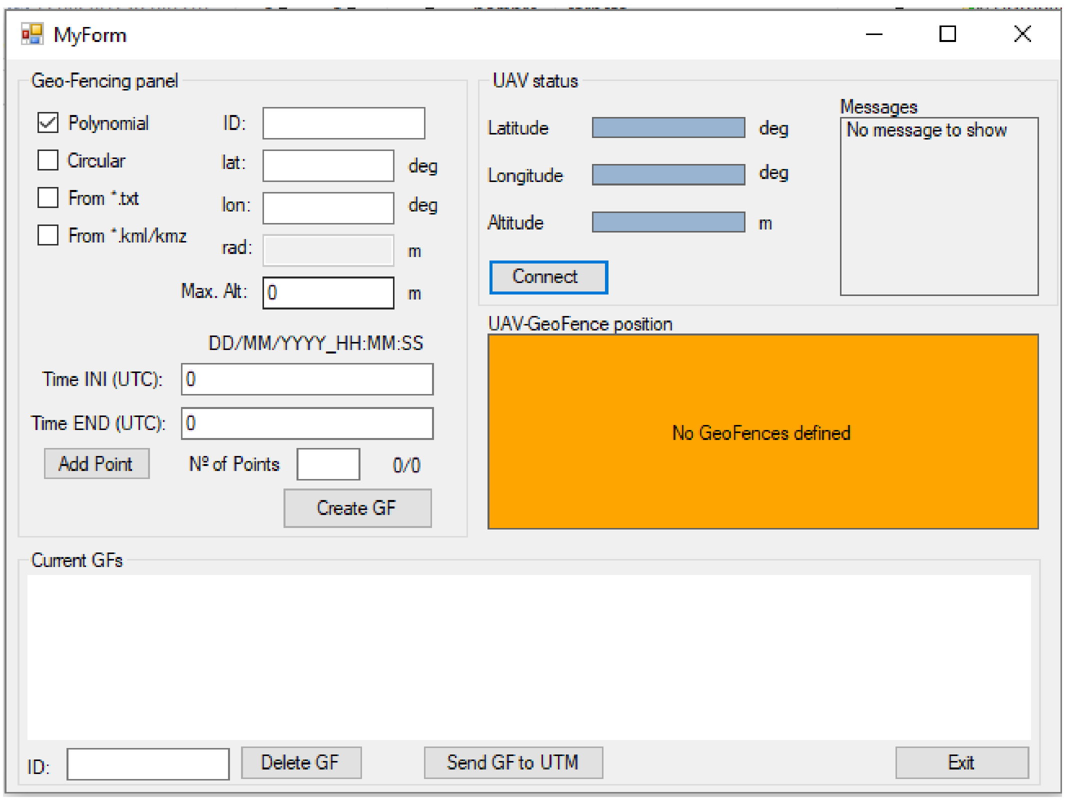
,


Figure 4.1: Graphic Using Interface of the Geo-Fencing software

4.1 GF creator

This part corresponds to the left-hand side of the GUI. The first action that the user has to take is the selection of the shape/input format of the Geo-Fence checking one of the four check boxes available. Depending on the checkbox that is checked, the text boxes available change according to the parameters that each shape needs as it is shown in Figure 4.2.

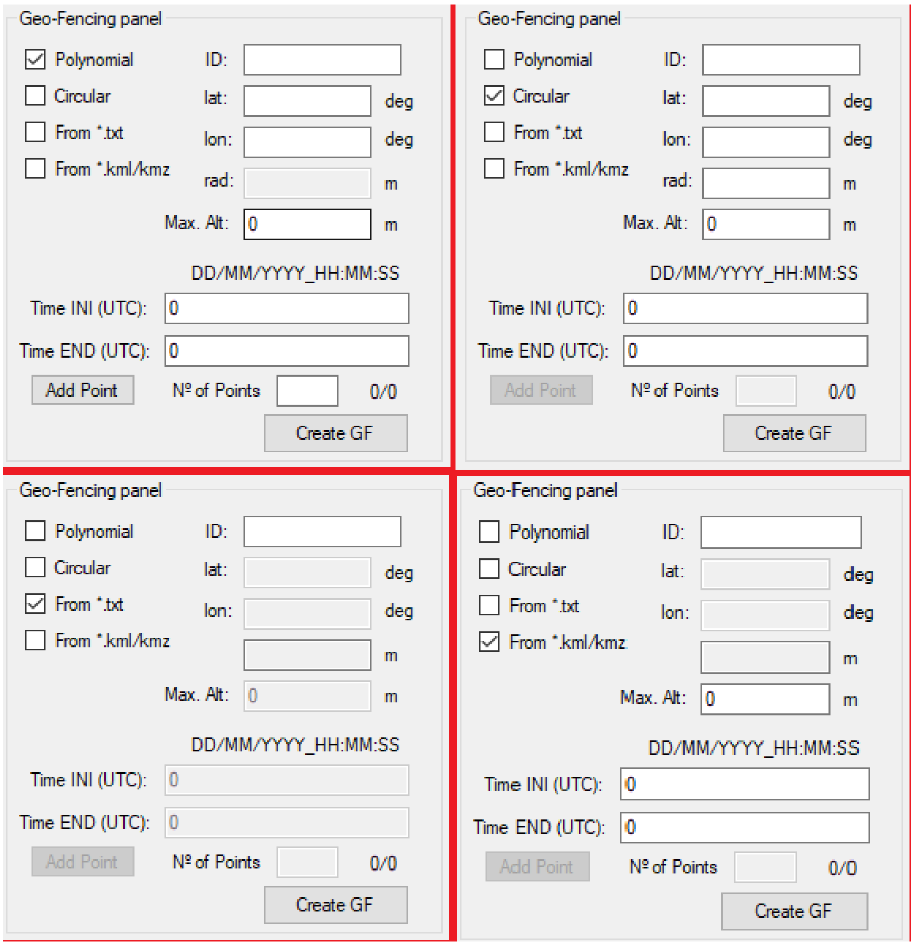


Figure 4.2: Parameters required for the different shapes/input formats

As mentioned before, the user can create manually the GFs selecting the shape polygonal or circular, or can select directly the input file with the GFs from a *.txt* or a *.kml*:

1. **Polygonal shape. Manual creation.**

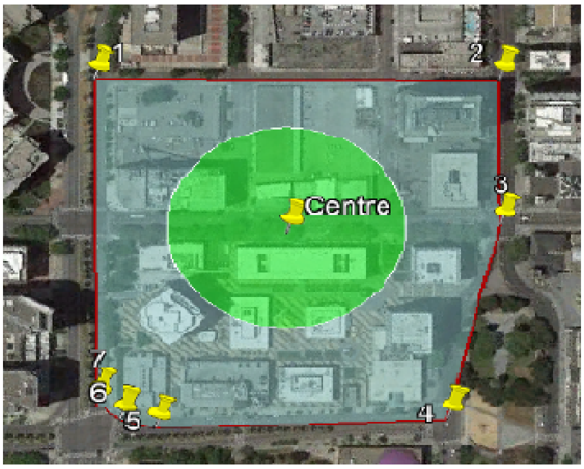
   When this check box is selected, the user can create a polygonal Geo-Fence from a list of points. The inputs that are introduced only once are: ID of the Geo-Fence, Maximum Altitude, Initial and Final time and the number of points. Then in the *lat:* and *lon:* text boxes is written the Latitude and Longitude of the first point and the button *Add Point* is pressed. Then you add the rest of the points following the same steps.

Every time that a point is added, the label next to the number of points text box displays the number of points added out of the total points. Once all the points are added, the button *Create GF* creates a string with a standard format that stores all the GFs created. The standard format for the polygonal GF is:

ID;shape;N of Points;Latitude 1; Longitude 1;...;...;Latitude N; Longitude N; Maximum Altitude; Initial Time; Final Time;end

For example, the standard format of the Geo-Fence shown in Figure 4.3 is:

,



Point	Latitude	Longitude
1	37°19'59.67"N	121°53'39.30"W
2	37°20'4.44"N	121°53'29.17"W
3	37°20'0.95"N	121°53'26.50"W
4	37°19'55.71"N	121°53'24.32"W
5	37°19'52.01"N	121°53'31.48"W
6	37°19'51.79"N	121°53'32.50"W
7	37°19'51.93"N	121°53'33.42"W

Centre Latitude	Centre Longitude	Radius
37°19'58.25"N	121°53'31.71"W	100 m

Figure 4.3: Example of Polygonal Geo-Fence

SanJose;poly;7;37.3332416666666;-121.8942472222222;37.334566666666667;-121.8914361111111;37.333597222222224;-121.8906944444444;37.332141666666665;-121.8900888888889;37.331113888888886;-121.8920777777778;37.331052777777778;-121.8923611111111;37.331091666666666;-121.8926166666667;0;03/08/2018_18:05:00;03/08/2018_18:45:00;end

2. **Circular shape. Manual creation.**

   When this check box is selected, the user can create a circular Geo-Fence from a centre and a radius. All the inputs are introduced only once: ID of the Geo-Fence, Latitude of the centre, Longitude of the centre, radius, Maximum Altitude, Initial and Final time and the number of points.

Then the button *Create GF* creates a string with a standard format that stores all the GFs created. The standard format for the circular GF is:

ID;shape;Latitude centre; Longitude centre;Radius; Maximum Altitude; Initial Time; Final Time;end

Once again, the standard format of the circular Geo-Fence shown in Figure 4.3 is:

***SanJoseCirc;circ;7;37.332847222222220;-121.8921416666667;100;500;03/08/20
18_18:05:00;03/08/2018_18:45:00;end***

3. **Input from *.txt*.**

   When this check box is activated, all the GFs are loaded from a *.txt* file. The name of the file has to be introduced in the ID text box without the extension *.txt*. The standard format of this file starts with a line containing the number of GFs. Every GF then is written in a new line with the same format explained before taking into account the shape of it. An example of a .txt file containing the 2 GFs shown before is:

   2 \n

   ***SanJose;poly;7;37.3332416666666666;-121.8942472222222;37.3345666666666667;-
121.8914361111111;37.333597222222224;-121.8906944444444;37.33214166666
6665;-121.8900888888889;37.331113888888886;-121.8920777777778;37.331052
777777778;-121.8923611111111;37.331091666666666;-121.8926166666667;0;03
/08/2018_18:05:00;03/08/2018_18:45:00;end\n***

   ***SanJoseCirc;circ;7;37.332847222222220;-121.8921416666667;100;500;03/08/20
18_18:05:00;03/08/2018_18:45:00;end\n***

4. **Input from *.kml*.**

   The *.kml* checkbox let the user create GFs from *kml.* files. The first input that the user introduces is the name of the file. The file has to contain a Google Earth polygon. Then the software extracts the list of points from the .kml and generates a string with the same format as the .txt file explained before. To build this string, the program reads from the interface the values introduced by the user in the text boxes: Maximum Altitude, Initial and Final time.

Once the standard string has been created, it can be sent to the Scheduler using the button Send GF to UTM. When the button is pressed, the software uses opens a socket and sends the information using the TCP/IP protocol.

In the bottom part of the GUI is shown a list with the current GFs. Typing any ID and pressing the button *Delete GF* the user can delete GFs that have been manually created (Using checkboxes Polygonal and Circular).

4.2 GF comparator

This part of the software corresponds to the right-hand side of the GUI. The program deals with two tasks: read the positioning from the receiver and compute the relative position between vehicle and GFs.

The receiver produces different NMEA sentences that the program reads. NMEA (National Marine Electronics Association) developed a standard list of messages. This standard lets marine electronics to send information to other equipment. GPS receiver communications use this specification. The idea of this standard is to send a line of data self-contained and independent from others. Each sentence begins with a $ and ends with a carriage return. The sentences cannot be longer than 80 characters and the data is separated by commas.

Depending on the constellation used, the 2 first characters of the sentence change being *GP* for GPS, *Gl* for GLONASS and *GA for GALILEO*. The information contained in the sentence depends on the following three characters. The standard NMEA messages used in GNSS can be found in Table 4.1.

Message	**Content**
DTM	Datum Reference
GBS	GNSS Satellite Fault Detection
GGA	Global positioning system fix data
GLL	Latitude and longitude, with time of position fix and status
GPQ	Poll message
GRS	GNSS Range Residuals
GSA	GNSS DOP and Active Satellites
GST	GNSS Pseudo Range Error Statistics
GSV	GNSS Satellites in View
RMC	Recommended Minimum data
THS	True Heading and Status
TXT	Text Transmission
VTG	Course over ground and Ground speed
ZDA	Time and Date

Table 4.1: List of Standard NMEA messages

From Table 4.1, the only messages we need to know the positioning are GGA, GLL and RMC and ZDA to know GNSS date and time. The information contained in these four messages is:

- **GGA** - Global positioning system fix data. See Table 4.2
- **GLL** - Latitude and longitude, with time of position fix and status. See Table 4.3
- **RMC** - Recommended Minimum data. Table 4.4
- **ZDA** - Global positioning system fix data. See Table 4.5

These NMEA messages are generated by the receiver explained in 3.1.2 and are sent using a USB cable as a serial COM port. Using managed C++ gives some problems when using serial COM ports. In order to solve these problems, a Matlab interface was created

Field No.	Example	Format	Unit	Description
0	$GPGGA	string	-	Message ID, GGA protocol header
1	92725	hhmmss.sss	-	UTC Time, Current time
2	4717.11399	ddmm.mmmm	-	Latitude, Degrees + minutes, see Format description
3	N	character	-	N/S Indicator, N=north or S=south
4	833.9159	dddmm.mmmm	-	Longitude, Degrees + minutes, see Format description
5	E	character	-	E/W indicator, E=east or W=west
6	1	digit	-	Position Fix Status Indicator, See Table below and Position Fix Flags description
7	8	numeric	-	Satellites Used, Range 0 to 12
8	1.01	numeric	-	HDOP, Horizontal Dilution of Precision
9	499.6	numeric	m	MSL Altitude
10	M	character	-	Units, Meters (fixed field)
11	48	numeric	m	Geoid Separation
12	M	character	-	Units, Meters (fixed field)
13	-	numeric	s	Age of Differential Corrections, Blank (Null) fields when DGPS is not used
14	0	numeric	-	Diff. Reference Station ID
15	*5B	hexadecimal	-	Checksum
16	-	character	-	Carriage Return and Line Feed

Table 4.2: Content of GGA nmea sentence

Field No.	Example	Format	Unit	Description
0	$GPGLL	string	-	Message ID, GLL protocol header
1	4717.11364	ddmm.mmmm	-	Latitude, Degrees + minutes, see Format description
2	N	character	-	N/S Indicator, hemisphere N=north or S=south
3	833.91565	dddmm.mmmm	-	Longitude, Degrees + minutes, see Format description
4	E	character	-	E/W indicator, E=east or W=west
5	92321	hhmmss.sss	-	UTC Time, Current time
6	A	character	-	V = Data invalid or receiver warning, A = Data valid.See Position Fix Flags description
7	A	character	-	Positioning Mode, see Position Fix Flags description. Optional block
7	*60	hexadecimal	-	Checksum
8	-	character	-	Carriage Return and Line Feed

Table 4.3: Content of GLL nmea sentence

Field No.	Example	Format	Unit	Description
0	$GPRMC	string	-	Message ID, RMC protocol header
1	083559.00	hhmmss.sss	-	UTC Time, Time of position fix
2	A	character	-	Status, V = Navigation receiver warning, A = Data valid, see Position Fix Flags description
3	4717.11437	ddmm.mmmm	-	Latitude, Degrees + minutes, see Format description
4	N	character	-	N/S Indicator, hemisphere N=north or S=south
5	00833.91522	dddmm.mmmm	-	Longitude, Degrees + minutes, see Format description
6	E	character	-	E/W indicator, E=east or W=west
7	0.004	numeric	knots	Speed over ground
8	77.52	numeric	degrees	Course over ground
9	091202	ddmmyy	-	Date in day, month, year format
10	-	numeric	degrees	Magnetic variation value, not being output byreceiver
11	-	character	-	Magnetic variation E/W indicator, not being output by receiver
12	-	character	-	Mode Indicator, see Position Fix Flags description
13	*57	hexadecimal	-	Checksum
14	-	character	-	Carriage Return and Line Feed

Table 4.4: Content of RMC nmea sentence

Field No.	Example	Format	Unit	Description
0	$GPZDA	string	-	Message ID, ZDA protocol header
1	082710.00	hhmmss.sss	-	UTC Time
2	16	dd	day	UTC time: day, 01..31
3	09	mm	month	UTC time: month, 01..12
4	2002	yyyy	year	UTC time: 4 digit year
5	00	-xx	-	Local zone hours, not supported (fixed to 00)
6	00	zz	-	Local zone minutes, not supported (fixed to 00)
7	*64	hexadecimal	-	Checksum
8	-	character	-	Carriage Return and Line Feed

Table 4.5: Content of ZDA nmea sentence

to guarantee the communication between the receiver and the Geo-Fencing comparator. This Matlab file is called *receive_reader.m* and can be found in Appendix 7.2.

Messages are read by the Matlab interface as fast as the receiver sends them, all the useful information is obtained and stored in a string with the format:

"Latitude;Longitude;Altitude;Speed;Heading;Day_UTC;Month_UTC;Year_UTC;Hour_UTC"

Then the program creates an opens a UDP socket and sends this string to the C++ Geo-Fencing comparator. The update rate of the messages is 1 s. Once the program has the positioning and time of the UAV, it compares them with the GFs created. For each GF this comparison follows three criteria in the next order:

1. **Time**

   The first thing to do is to compare the current time with the initial and final time of all the GFs. If the simulation time is not between them, the program return that the vehicle is outside.

2. **Maximum altitude**

   Then the program compares if the vehicle is above the maximum altitude of the GF. If so, the program returns that the vehicle is outside.

3. **Position**

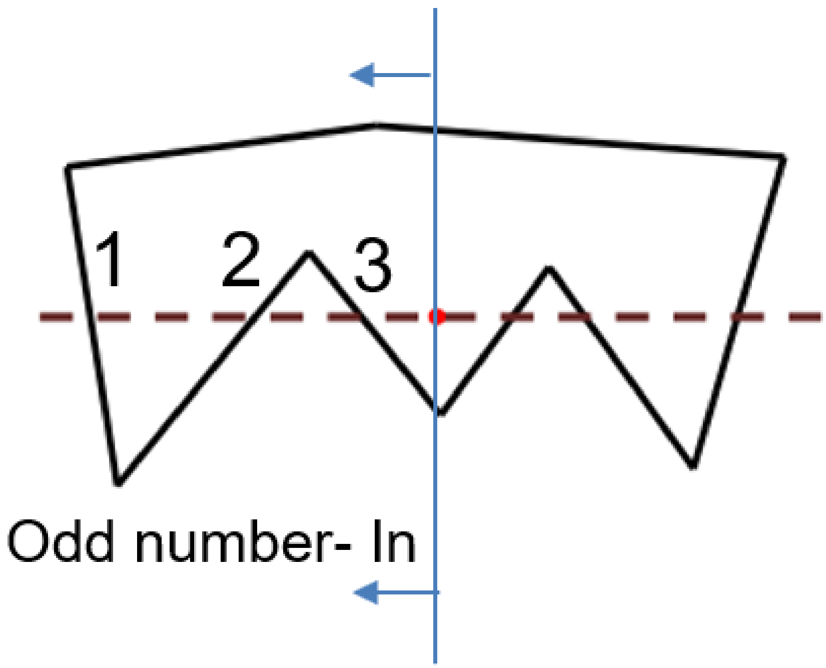
   Finally, if the previous two conditions are satisfied, the software computes if the vehicle is inside the boundary of the GF. The algorithms used to do that are explained below.

Depending on the shape of the region, the comparison is done using different algorithms. For circular GFs the way to compare is quite simple:

1. Calculate distance between UAV and centre of GF
2. If the distance is lower or equal than the radius of the GF, the vehicle is inside. Otherwise, the vehicle is outside.

However, computing the relative position between a point and a polygon is much more difficult. In order to know if a point is inside a polygon convex or non-convex the following algorithm is used:

1. The polygon is given by a list of points
2. Draw a horizontal line at the y-value of UAV as shown in Figure 4.4
3. Count the number of crosses between the horizontal line and the polygon only at one side of the UAV
4. If the number of crosses is even, the UAV is outside the Geo-Fence. Otherwise, the vehicle is inside.



,

Figure 4.4: Example of point inside non-convex polygon

As mentioned before, the receiver sends its positioning once per second and the program computes the relative position at the same rate time. The relative position is displayed in the orange box shown in Figure 4.1. When the UAV is outside the GF, the box shows *UAV outside* and the orange box changes to **green**. When the vehicle goes inside a GF, the message displayed is *UAV inside* and the background colour is **red**.

The main limitations of the Geo-Fencing software are:

- The program works properly when using small GFs, but for large GF is observed an error due to the use of the NED coordinate system (explained above).
- The GF comparator works with a rate of 1 s, which is the rate of the receiver sending NMEA messages. 1 s is a lot of time taking into account the speeds and distances of the tests done in this project, therefore, for a high precision performance inside GFs would be necessary a receiver with a lower update rate of the positioning

4.3 Supporting modules of the project

4.3.1 Motion Planner

As explained in Figures 1.1 and 3.1, the Motion Planner was developed in order to simulate the trajectories of the UAV to test the Geo-Fencing library. To test the Geo-Fences, we had two options: test them in a real environment with a real UAV which is the most complex and expensive option, and create virtual UAVs simulating their trajectories in the most simple way, which is much more simple, cheap and fast than using real drones.

Once the decision was made, we had two new options: use one of the complex and realistic software available in the market to simulate trajectories of vehicles such as AirSim or create a very simple generator less realistic, but faster and simpler, and enough accurate for our mission. Again this second option was the one we chose.

The motion command that the simulator needs uses the parameters shown in Table 4.6. In order to generate these parameters, two different versions of the motion planner have been developed. One version is used to use generate trajectories in real time with the user changing, speed, altitude and other parameters in the loop. The second version was developed to follow a path plan with a path following algorithm. The paths of this second version are created by the Scheduler taking into account the mission selected for the UAV. This will be explained in 4.3.2.

Symbol	**Parameter**	**Unit**
x	position ECEF, x-axis	$[m]$
y	position ECEF, y-axis	$[m]$
z	position ECEF, z-axis	$[m]$
vel_x	velocity ECEF, x-axis	$[m/s]$
vel_y	velocity ECEF, y-axis	$[m/s]$
vel_z	velocity ECEF, z-axis	$[m/s]$
acc_x	acceleration ECEF, x-axis	$[m/s^2]$
acc_y	acceleration ECEF, y-axis	$[m/s^2]$
acc_z	acceleration ECEF, z-axis	$[m/s^2]$
$jerk_x$	jerk ECEF, x-axis	$[m/s^3]$
$jerk_y$	jerk ECEF, y-axis	$[m/s^3]$
$jerk_z$	jerk ECEF, z-axis	$[m/s^3]$
h	heading - radians, range +/-	$[Rad]$
e	elevation - radians, range +/- /2	$[Rad]$
b	bank , radians - range +/-	$[Rad]$
$_x$, $_y$, $_z$	angular velocity about x,y,z body axes	$[Rad/s]$
$\dot{\omega}_x$, $\dot{\omega}_y$, $\dot{\omega}_z$	angular acceleration about x,y,z body axes	$[Rad/s^2]$
$\ddot{\omega}_x$, $\ddot{\omega}_y$, $\ddot{\omega}_z$	angular jerk about X,Y,Z body axes	$[Rad/s^3]$

Table 4.6: Parameters required by the motion commando MOT of the GSS 7000

Motion planner - Driver in the loop

As explained before, the purpose of this software is to let the user change the parameters of the flight as if it was the pilot of the UAV. It was developed in order to test the Geo-Fencing software, to be sure that it works computing if the vehicle is inside or outside the Geo-Fences.

The explanation of how it works is going to be based on the aspect of the Graphic User Interface, with its text boxes, buttons etc. The aspect of the GUI is shown in Figure

4.5. In the interface are found three different types of user controls: text boxes, check boxes and buttons, divided into three containers: Initial conditions, flight parameters and outputs.

,

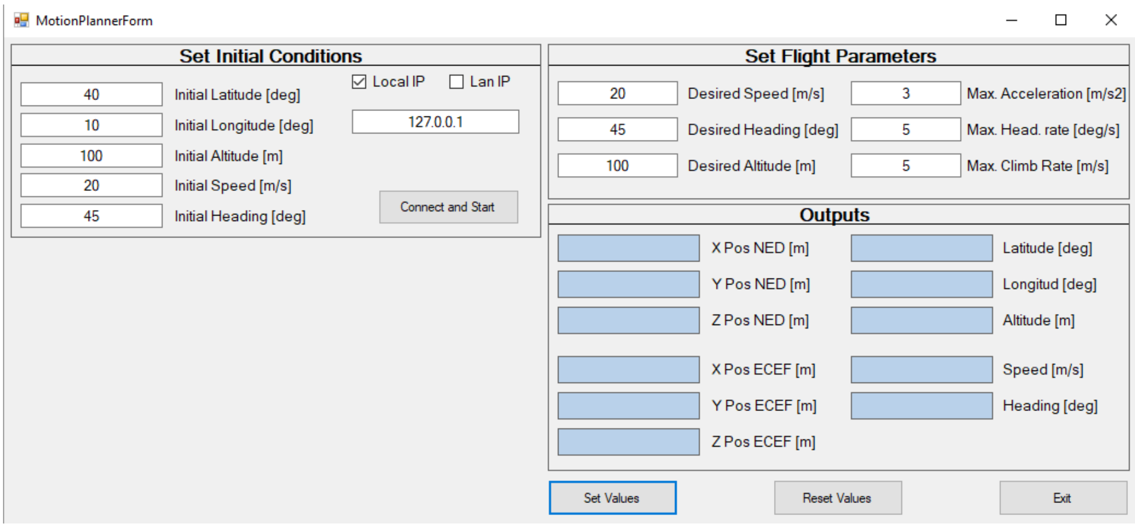


Figure 4.5: Graphic User Interface of the motion planner

Only when the buttons are pressed the data introduced in the text boxed is read by the software. In the *Set Initial Conditions* frame, the initial conditions of the UAV are selected. These conditions are:

- Initial Latitude [deg]
- Initial Longitude [deg]
- Initial Altitude [m]
- Initial Speed [m/s]
- Initial Heading [deg]

This frame is also used to select if the program is used in the same PC than the Simulator (Local IP check box - IP: 127.0.0.1) or remotely using Ethernet connection (Lan IP check box - Typical IP of the simulator: 192.1.1.1).

In the *Set Flight Parameters* frame, the flight conditions of the UAV are selected. These conditions are:

- Desired Speed [m/s]
- Desired Altitude [m]
- Desired Heading [deg]

This frame is used to select a very simple air-frame of the vehicle. This airframe is given by three different parameters:

- Acceleration [m/s^2]
- Heading Rate [deg/s]
- Climbing/Descending rate [m/s]

When the button *Set Values* is pressed, the selected values of both, flight conditions and airframe are stored in the program. If the button *Reset Values* is used, all the parameters of this frame are fixed with the default values shown in Figure 4.5.

Once all the conditions have been chosen, the button *Connect and Start* initializes the simulation of the UAV. To do that, the steps followed by the software when the button is pressed are:

1. A local frame is created using the initial latitude and longitude. This local frame is used to convert all the coordinates into a NED (North-East-Down) coordinate system with origin in this local frame.

2. An instance of the class UAV is created with the initial conditions from the GUI and the local frame.

3. A TCP/IP socket is created to send a command to the simulator that runs the scenario. This message **RUNOWAIT** is sent to the GSS 7000.

4. A UDP socket is created to send the motion command **MOT** to the simulator. And a Background Worker starts taking control of the UAV created

The Background Worker generates the parameters that the simulator needs which are shown in Table 4.6 and sends them using the UDP socket created before. A new message is sent every 0.1 seconds. To calculate the position, velocity, etc at each time step, very simple equations of motion are used divided into 8 different cases. The equations for each case are:

1. The current flight conditions are equal to the desired ones.

   The simplest case. The vehicle is moving with constant speed maintaining heading and altitude constant. The equations are:

$$x_t = x_{t-1} + \dot{x} \cdot t \ ,$$

   where x is a vector with the position in NED coordinates, $\dot{x}$ is the desired speed of the UAV and t is the time step. The velocity in the z-axis is 0 (Altitude constant).

2. The current speed is different of the desired speed.

$$\dot{x}_t = \dot{x}_{t-1} + \ddot{x} \cdot t \ ,$$

$$x_t = x_{t-1} + \dot{x}_t \cdot t + \frac{1}{2} \cdot \ddot{x} \cdot t^2$$

   where $\ddot{x}$ is a vector with the acceleration from the GUI in NED coordinates. Again the velocity and acceleration in the z-axis is 0.

3. The current altitude is different from the desired altitude.

   The equation of motion of this case is the same than before but changing the speed in the z-axis with the Climb rate from the GUI.

4. The current heading is different from the desired heading.

   The equations for this case are the same as the first case, but the heading is changing, so the conversion to NED takes into account the change in heading following the equation:

$$\theta_t = \theta_{t-1} + \dot{\theta}_t \cdot t \ ,$$

   where θ is the heading and $\dot{\theta}$ is the heading rate from the GUI.

The other 4 cases are combinations of cases 2, 3 and 3 in groups of 2 and 3.

Once position and speed are calculated, these are transformed into LLA and ECEF coordinates to be displayed in the GUI and to be sent to the simulator. In order to simplify, only the position, speed and heading are sent, the rest of the parameters expected in the motion command shown in Table 4.6 are sent as 0. With these three parameters, the simulator has enough inputs to generate the RF signal.

To increase the accuracy of the software, it calculates the execution time of each iteration in order to send a message exactly every 100 milliseconds.

Motion planner - Path Following algorithm

The purpose of this motion planner is to test the drone performance during missions inside Geo-Fences. This motion planner does not use a User Interface, it is used when the button *Run Simulation* of the GUI of the Scheduler is pressed (It is shown in Figure 4.7).

The path following algorithm starts from a path planning given by a list of points. The Carrot Chasing algorithm is the one selected for this purpose. This algorithm introduces a virtual target point (VTP) and makes the UAV chase the VTP. The VTP is called the carrot. The heading of the UAV is updated toward the VTP while it is moving as time progress.

The algorithm is divided in two different cases: straight-line following and loiter (arc of circumference following). The pseudo-code for the Straight-Line following case is:

1. Initialize parameters: $W_i = (x_i, y_i)$, $W_{i+1} = (x_{i+1}, y_{i+1})$, $p = (x, y)$, ψ, δ, v_a, κ

2. Calculate distance from current position to initial point of the straight segment and the angle between straight line and the current position:

$$R_u = ||W_i - p||$$

$$\theta = \tan^{-1} \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

3. Compute angle between the current position and the x axis, and difference between both angles:

$$\theta_u = \tan^{-1} \frac{y - y_i}{x - x_i}$$

$$\Delta\theta_u = \theta - \theta_u$$

4. Evaluate distance between initial point of the straight line and the projection of the current position in this line:

$$R = \sqrt{R_u^2 - (R_u \sin \Delta\theta)^2}$$

5. Compute VTP

$$s = (x_t, y_t) = (R + \delta)(\cos\theta, \sin\theta)$$

6. Calculate desired heading:

$$\psi_d = \tan^{-1} \frac{y_t - y}{x_t - x}$$

7. Extract lateral acceleration required:

$$u = max\_lim(\kappa(\psi_d - \psi)v_a)$$

where all the variables are shown in Figure 4.6 and *max_lim* is:

$$max\_lim(z) = \begin{cases} z & if|z| < |u|_{max} \\ sign(z)|u|_{max} & if|z| \geq |u|_{max} \end{cases}$$

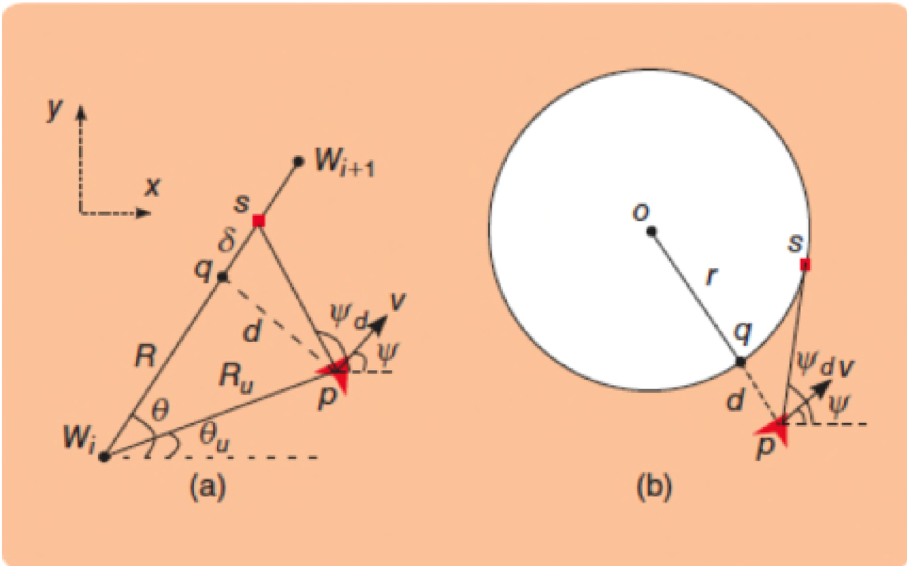
,


Figure 4.6: Carrot-Chasing straight and loiter algorithms

The value of δ indicates the distance between the closest point to the desired line and the VTP and it is fixed to 30 m. The line to follow changes when the vehicle is less than 5 m far from the final point of the straight line to follow.

The pseudo-code for the loiter algorithm is:

1. Initialize parameters: $O = (x_i, y_i)$, r, $p = (x, y)$, ψ, λ, v_a, κ

2. Compute minimum distance to the circumference:

$$d = ||O - p|| - r$$

3. Evaluate angle between current position and the center of the circumference:

$$\theta = \tan^{-1}\frac{y - y_i}{x - x_i}$$

4. Calculate VTP:

$$s = (x_t, y_t) = (x_i, y_i) + r(\cos\theta + \lambda, \sin\theta + \lambda)$$

5. Calculate desired heading:

$$\psi_d = \tan^{-1}\frac{y_t - y}{x_t - x}$$

6. Extract lateral acceleration required:

$$u = max\_lim(\kappa(\psi_d - \psi)v_a)$$

The value of λ indicates the angle between the closest point of the circumference to the UAV and the VTP as can be seen in Figure 4.6 and it is fixed to 0.1 rad.

During the development of this software, several limitations were found:

- Using the NED coordinate system makes calculations easier, and is accurate for trajectories that are not too large. However, if a very large Scenario is used, the NED coordinate system produces an error in altitude that increases with the distance from between the vehicle and the origin of the coordinate system.

- When using the Driver-in-the-Loop approach of the motion planner, the air-frame of the vehicle is very simple and is defined by the user, which can produce unrealistic trajectories. In addition, accelerations and jerks are not taken into account for the generation of trajectories because a very simple trajectory generator is enough for the scope of this project.

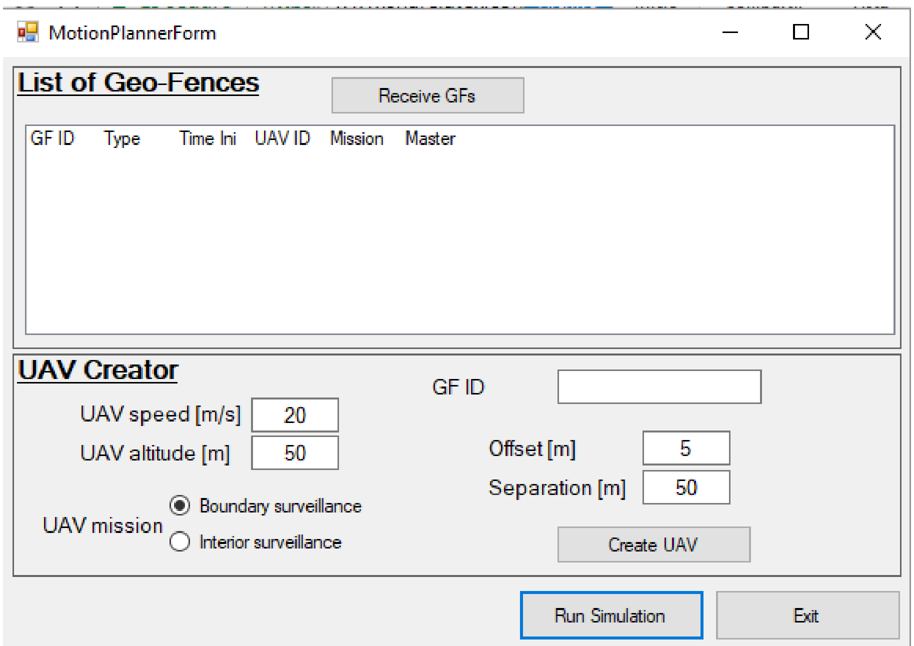
- The path following approach with the Carrot-chasing algorithm uses a very simple version of this algorithm which is designed to follow only straight and circular trajectories.

- The efficiency of the code is not high, therefore, it is only able to work with a rate of 100 ms, with a more efficient development of the software and/or more computational power, the software would be able to run with a shorter time step.

4.3.2 Scheduler

The last part of the software developed within this project is the Scheduler. The purpose of this software is to deal with the UTM. The tasks that the UTM designed in this project is supposed to deal with are several. First of all, it needs to know all the GFs in the scenario (The UTM is supposed to know where the restricted areas are, and for how long are they unavailable). Then it generates UAVs and selects a mission inside one of the GFs. It also creates the path planning of the vehicle depending on the mission selected.

The aspect of the GUI of the Scheduler is shown in Figure 4.7.

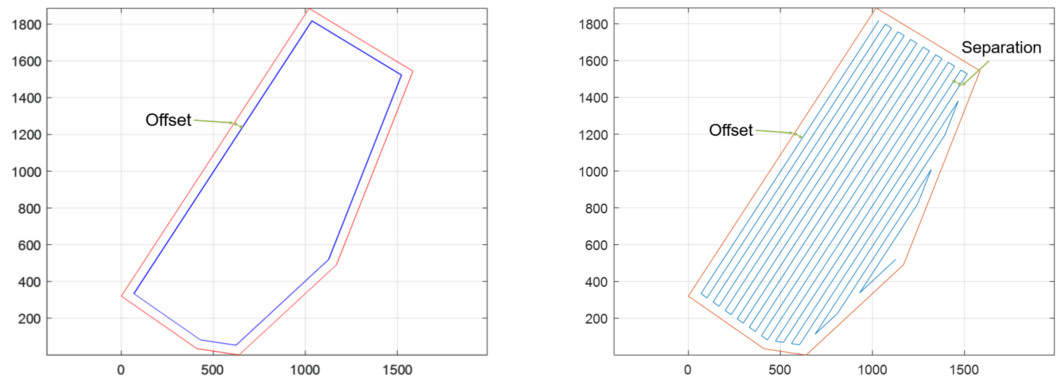


,

Figure 4.7: Example of point inside non-convex polygon

When the user presses the button *Receive GFs*, the list of GFs is sent by the Geo-Fencing creator with the format explained in 4.1. The list is displayed in the area of the GUI. Then, the user has to create the UAVs with a given mission and different parameters.

The two different missions available are boundary surveillance and interior surveillance and are shown in Figure 4.8. The other parameters to choose are altitude, speed, GF, offset and separation. GF means that each vehicle is assigned to a GF selected by its ID. The offset is the minimum between the path plan and the boundary of the GF. Finally, the separation is only used in the interior surveillance mission and gives us the distance between each long parallel line. These parameters are also shown in Figure 4.8.



(a) Boundary surveillance mission (b) Interior surveillance mission

Figure 4.8: Missions available in the Scheduler

Both missions are possible for circular and polygonal GFs. When the user presses the button *Create UAV*, an instance of UAV is created and the Path Planning for the chosen mission is computed. Below are explained the different algorithms used to do the path planning for each mission with each shape:

- **Boundary surveillance inside Circular GF**

  This is the only case where the path plan is not given by a list of points. Therefore, is the only mission that uses the circular carrot-chasing algorithm explained in 4.3.1.

The path plan is given by the position of the centre and the radius. This radius is calculated by subtracting the chosen offset to the radius of the GF.

- **Boundary surveillance inside a Polygonal GF**

  The path plan is given by a list of points. These points are the vertices of the interior polygon offset the chosen distance. The program generates a parallel line to each of the lines that form the boundary of the GF and computes the intersections between them.

  After trying to use different algorithms such as Straight-Skeleton and some others (Cacciola, n.d.; Palfrader and Held, 2015), it was decided to implement a simple algorithm using the bisectors of the interior angles of the polygon given by its vertices in clockwise order. This algorithm is very simple and has some limitations. For instance, it only works when the offset y lower than the length of every side of the polygon.

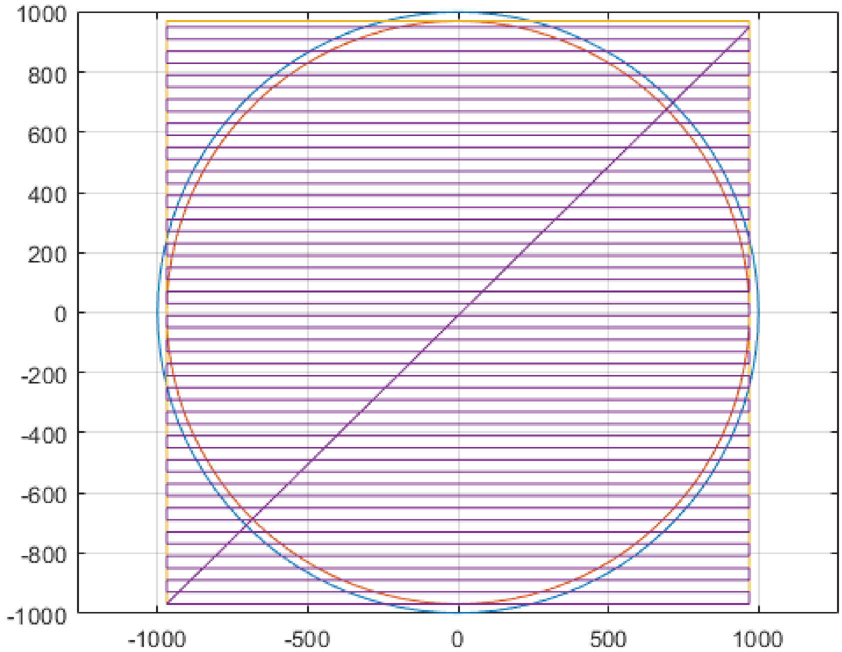
- **Interior surveillance inside a Circular GF**

  To calculate the list of points that form the path plan for this mission the steps explained below are shown:

  1. The offset interior circle is calculated like in the boundary surveillance mission.
  2. Calculate the square box that encloses the circumference.
  3. Calculate the number of parallel lines needed:

$$N = floor\left(\frac{2 \cdot Radius}{Separation}\right) \tag{4.1}$$

  4. Divide the big square in parallel lines as shown in Figure 4.9.



,

Figure 4.9: Bounding box and parallel lines for Interior surveillance mission inside circular GF

5. Compute the intersections between the lines and the circle in the right order to obtain the final path plan.

- **Interior surveillance inside a Polygonal GF**

  To calculate the list of points that form the path plan for this mission the steps explained below are shown:

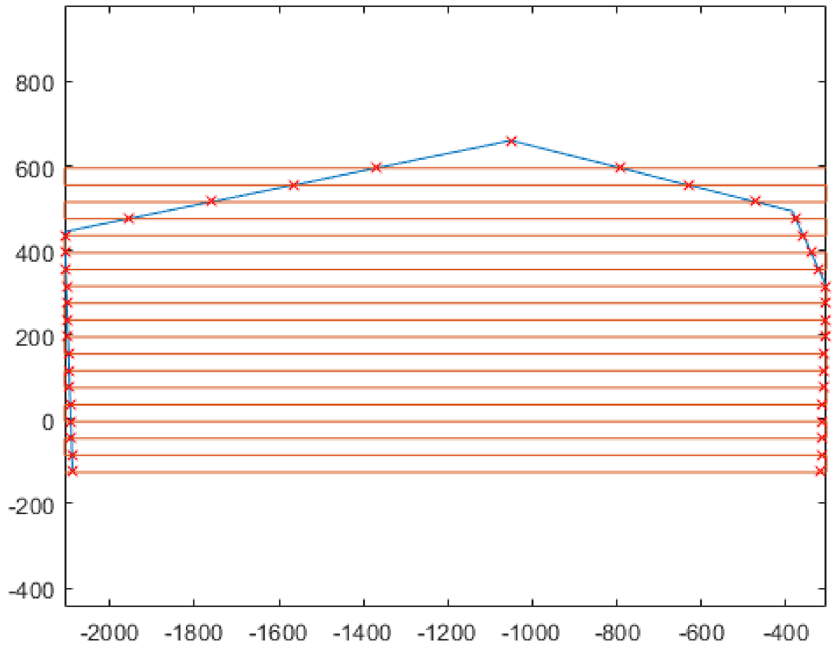
  1. Calculate the offset interior polygon like in the boundary surveillance mission.
  2. Calculate the longest side of the polygon in order to build the long lines parallel to it minimizing the number of turns.
  3. Rotate the polygon to have the longest side as a base
  4. Calculate the rectangular box that encloses the polygon.
  5. Calculate the height of the new rotated polygon:

$$h = max(DistPointToLine(Point_i, Base)) \tag{4.2}$$

6. Calculate the number of parallel lines needed:

$$N = floor\left(\frac{h}{Separation}\right) \tag{4.3}$$

7. Divide the big square in parallel lines as shown in Figure 4.10.



,

Figure 4.10: Bounding box and parallel lines for Interior surveillance mission inside Polygonal GF

8. Compute the intersections between the lines and the polygon in the right order.

9. Undo the rotation of the points done in 3 to obtain the final path plan.

Once the Path Planning is finished, it can be used by the motion planner explained in 4.3.1 when the button *Run simulation* is pressed.

The main limitation found in this software module is that, despite the GF software is able to generate and compute relative positioning using non-convex polygons, the path planning algorithms for both missions (Boundary and interior surveillance missions) are only able to work with convex polygons. That is the main reason why all the tests have been done with this type of polygon.

Chapter 5

Performance Analysis

The purpose of this chapter is to explain the two different performance analysis done in order to test the UAV performance inside Geo-fenced environments created using the Geo-Fencing software developed in this project. One is used to study the influence of Multi-Path in the GNSS positioning when a UAV is performing missions in Geo-Fenced environments. The second study is an analysis of the required distance margin needed to guarantee the safety of the missions done by UAVs within Geo-Fences.

For all these tests is used the configuration of the software where the Scheduler creates the UAVs with chosen parameters and mission and the motion planner uses the Carrot-Chasing path following approach explained in 4.3.1 instead of the Driver-in-the-Loop one.

Finally an extra experiment has been developed in order to test the boundary that any GF has to have, not only due to GNSS accuracy, but also taking into account the the reaction time and the manoeuvre time due to the air-frame of a given aircraft using Driver-in-the-Loop integrating the flight simulator Xplane with the GF software and the GNSS simulator.

5.1 Speed - Offset study

In these tests has been developed a parametric study to demonstrate the minimum offset required with the boundary off a GF to be completely sure that the vehicle is performing its mission always inside the GF. The test has been done several times for different missions, speeds and offsets.

The factors that can make the UAV be outside the GF when it is supposed to be inside are:

- Path following algorithm. Carrot-Chasing

  Due to the algorithm parameters, such as minimum turn ratio or maximum lateral acceleration, the trajectory of the vehicle does not follow the straight lines given by the path planning, so in the transition between 2 straight lines, the real trajectory followed by the UAV could cross the boundary of the GF.

- Receiver error

  The receiver used for the simulations has a Standard GNSS Precision, not a High GNSS Precision

- GNSS accuracy

  The GNSS accuracy depends on the number of constellations used and the signals. Which will be explained below.

The different test done are shown in Table 5.1. In order to have some accuracy in the results, every test shown in the table has been done 3 times for the same scenario and averages of the results have been taken and will be explained in 6.

The scenario chosen for this test can be separated into three different parts:

- **Location of the Scenario** The location of this test is shown in Figure 5.1. It is a polygonal GF with six irregular sides surrounding the airfield of Cranfield Univer-

	Mission	
	Interior Surveillance	Interior Surveillance
Speed [m/s]	Offset [m]	Offset [m]
5	1	1
	3	3
	5	5
10	1	1
	3	3
	5	5
	7.5	7.5

Table 5.1: List of tests done for the Speed-Offset analysis

sity. And the points are given in Table 5.2. The GF does not have initial time, final time and maximum altitude.

Point	**Latitude**				**Longitude**			
1	52 °	4 ’	2.930 ”	N	0 °	37 ’	44.760 ”	W
2	52 °	4 ’	53.580 ”	N	0 °	36 ’	51.190 ”	W
3	52 °	4 ’	42.510 ”	N	0 °	36 ’	21.550 ”	W
4	52 °	4 ’	8.440 ”	N	0 °	36 ’	43.390 ”	W
5	52 °	3 ’	52.560 ”	N	0 °	37 ’	11.180 ”	W
6	52 °	3 ’	53.680 ”	N	0 °	37 ’	23.200 ”	W

Table 5.2: List of points of the GF

- **Time of the Scenario**

  The date and time of the beginning of the simulation for each of the tests shown in Table 5.1 are always the same. However, the duration of the simulation and consequently the final time depends on the speed, the offset and the mission. From around 5 minutes for the boundary surveillance mission at 10 m/s up to almost 1 hour for the interior surveillance mission at 5 m/s.

  Therefore, and without taking into account the lost time due to simulations with errors and the time to load each test, the time used to do all the test for this performance analysis is around 20 hours.

  Then, the initial date and time of the Scenario used for these tests are:

,

Figure 5.1: GF of the scenario selected for the Speed-Offset analysis

4th July 2017 at 05:00:00 UTC

- **GNSS conditions of the Scenario**

  This is the most complex part of the Scenario due to the large number of different parameters which can be modified within the Spirent Simulator. The Configuration of the GSS7000 for in this for these simulations is shown in the list below:

  – GNSS constellations - GPS

    Only GPS constellation is Used in order to simplify simulations. All the available constellations are shown in Figure 5.2. GPS is theUnited States' Global Positioning System and consists of up to 32 medium Earth orbit satellites in six different orbital planes.

– Signal - L1

GPS has 3 different bands: L1, L2 and L5. The L1 frequency contains a coarse acquisition (C/A) code and a navigation data message. This means that the L1 is a positioning and timing service that transmits Standard Positioning Service (SPS). L2 and L5 are used to get a highly accurate military positioning called Precise Positioning Service (PPS). All the signals of each constellation are shown again in Figure 5.2.

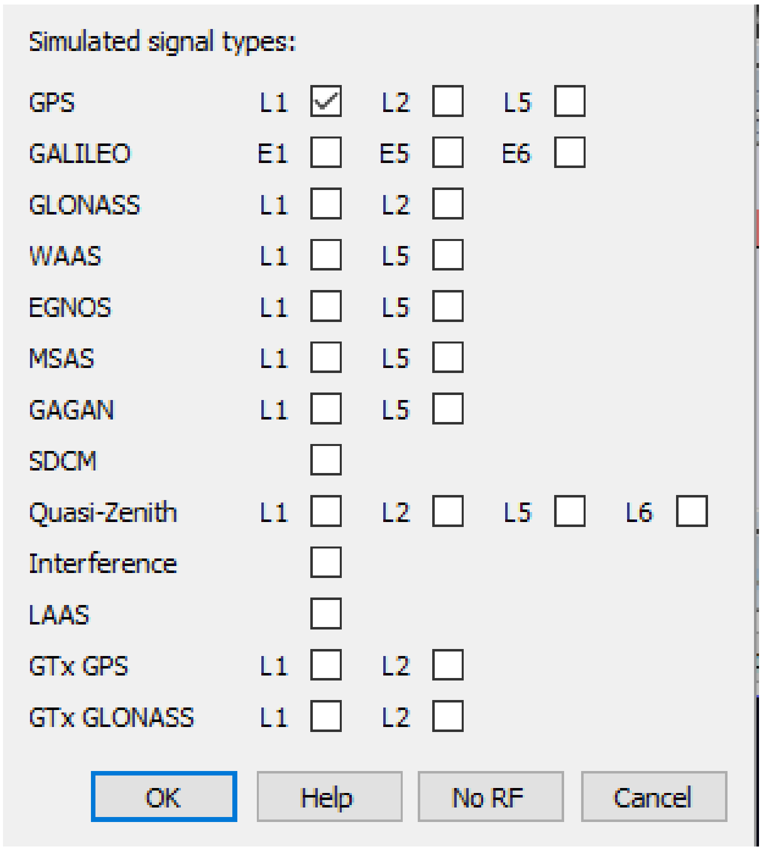


Figure 5.2: Constellations ans signals available

– Power of the signal of the satellites - between 10 and 12 dB (Default values of the simulator)

– Tropospheric model - STANAG

The tropospheric delay is caused by two different causes. The hydrostatic component delay is caused by the dry gases present at this layer of the atmosphere and is very predictable. The wet component delay which is caused by the water vapour and condensed water and depends on weather conditions and is very difficult to predict.

The tropospheric corrections currently used neglect the contribution due to the 'wet' component. There are different models available in the simulator such as BD2, RTCA (with different versions) and STANAG. The STANAG model is recommended in recently established standards for GPS time receiver software

– Ionospheric model - Klobuchar

  The ionospheric corrections are broadcast by the satellites. Each constellation uses a different model. The model used by GPS is the Klobuchar model. Galileo uses Nequick model. The Klobuchar model is estimated to reduce about 50% of the ionospheric error. It assumes that the electrons are concentrated in a thin layer of the atmosphere at 350 km altitude. Then the delay is computed taking into account the vertical delay corrected with an obliquity factor.

– Scintillation - Disabled

  Ionospheric scintillation is caused by rapid fluctuations in amplitude and phase of trans-ionospheric GNSS signals caused by the irregularities in the distribution of electrons during the propagation path. It is very complex to model, because it depends on the location and the time of the day and does not have a high impact in the final positioning, therefore for this simulations it is assumed negligible.

– Antenna of the Vehicle - Isotropic (Ideal)

  An isotropic antenna is an ideal antenna that radiates its power uniformly in all directions.

Those are the most relevant parameters that can be defined in the simulator, however, there exist many other variables that can be controlled and the scenario is available together with all the developed software attached to this report.

- **UAV parameters of the Scenario**

  Finally, the UAV uses the airframe given by the Carrot-Chasing algorithm explained in 4.3.1. And all the missions are done at an altitude of 50 m above the Geoid.

5.2 Multi-Path influence study

The purpose of this test is to study the influence of Multi-path in the GNSS RF in urban environments while a UAV is performing a mission in Geo-Fenced areas. In order to succeed with this analysis, it was necessary to use some of the equipment developed by Spirent and located in Paignton.

The software used to generate the Multi-path is called Sim3D. It uses a 3D model of the buildings of an urban environment, takes the GNSS RF signals and computes the different Multi-path signals, which are sent back to the GNSS simulator as it is shown in Figure 5.3.

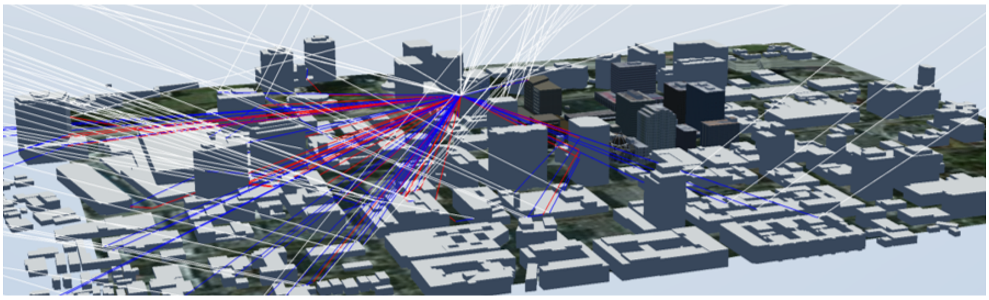

,

Figure 5.3: Sim3D computing Multi-path in San Jose(USA)

It is only developed for the city of San Jos (CA, US), therefore, the location of the tests was placed there. The different tests done are shown in Table 5.3. The first test is used as a reference and then some parameters are changed to see Multi-path influence when changing different parameters.

	Nominal Test	**1**	**2**
Scenario Size	S	S	S
PDOP	<4	<4	>4
Altitude	120 m	300 m	120 m
Mision	Interior S.	Interior S.	Interior S.
Offset	5 m	5 m	5 m
Separation	15 m	15 m	15 m
Speed	5 m/s	5 m/s	5 m/s
Initial date	4th July 2017	4th July 2017	4th July 2017
Initial time	05:00:00	05:00:00	00:00:00

Table 5.3: Selected parameters for each test

These tests have been done only once each due to the lack of time during the visit to Spirent. The two different GFs used for this test are shown in Figure 5.4. The area labelled with the S is the most critical part for the Multi-path influence because it takes the reflections of the signals on the surfaces of all the buildings that are inside the L area.

The tests with Multi-path were done using a different version of the GNSS simulator, the GSS 9000. However, the tests done whiteout Multi-path to study its influence were done back in Cranfield with the GSS 7000. The parameters chosen in the scenario for the simulation are the same than those of the previous section:

- GNSS constellations - GPS
- Signal - L1
- Power of the signal of the satellites - between 10 and 12 dB (Default values of the simulator)
- Tropospheric model - STANAG
- Ionospheric model - Klobuchar
- Scintillation - Disabled
- Antenna of the Vehicle - Isotropic (Ideal)

,


Figure 5.4: GFs of the scenario selected for the Multi-path analysis

In the following chapter will be explained the results obtained after the post-process done with all the data from the simulations.

Chapter 6

Results and Discussion

Once the detailed explanation of the development of this project and the methodology of the performance analysis has been done, this chapter is used to provide details of all the results obtained from the tests mentioned before. Firstly, the results of the Speed-Offset study are fully explained.

6.1 Speed - Offset study

The purpose of this test is to study the minimum offset that a GF has to have to guarantee that the vehicles working inside remain there at any moment of their mission. As it can be deduced intuitively and was explained in the previous chapter, this offset depends on the speed of the vehicle, being larger when the speed increases.

From each of the tests in Table 5.1, the programs store different parameters in *.csv* files. These variables are written at a rate of 1 second which is the rate of time used by the receiver to provide positioning information. The information of each simulation is stored in two different files, one containing the motion command generated by the *Motion Planner* which is the input of the simulator (real position of the vehicle) and a second file containing the positioning obtained from the GNSS receiver (where the vehicle thinks it is located). The information stored in each file is:

- Motion command:

  Time of simulation

  Positioning (ECEF and LLA)

  Velocity, acceleration and jerk in ECEF

  Heading, bank and elevation angles

  Angular velocities and accelerations

- GNSS positioning:

  UTC time

  Positioning LLA

  Satellites used

  PDOP

From all these values, in this test are only used the positioning in LLA and the time. In order to do this analysis, Matlab was used to post-process the large amount of data. The method that has been followed calculates the time that the vehicle is outside the GF of the scenario.

For each of the time steps from the files, the program compares its positioning with the GF of the scenario using the algorithm explained in 4.2. The total number of points where the vehicle is outside gives us the time that the boundary of the GF has been crossed.

All the numerical results of this analysis in Tables 6.1, 6.2, 6.3 and 6.4 are given in terms of percentage of the time that the UAV is outside with respect the total mission time of one full cycle of the mission, which corresponds with the point when the vehicle reaches again the first point of its path.

In addition, the tables containing the results show the numeric value for each of the individual tests and also the average for each offset at a given speed. The time outside the GF is computed for both the path following algorithm and the GNSS positioning.

This means, for a single test, the motion command created by the Scheduler and sent to the simulator (Path Following) and the GNSS positioning obtained from the receiver are compared separately with the boundary of the GF.

6.1.1 Boundary surveillance mission

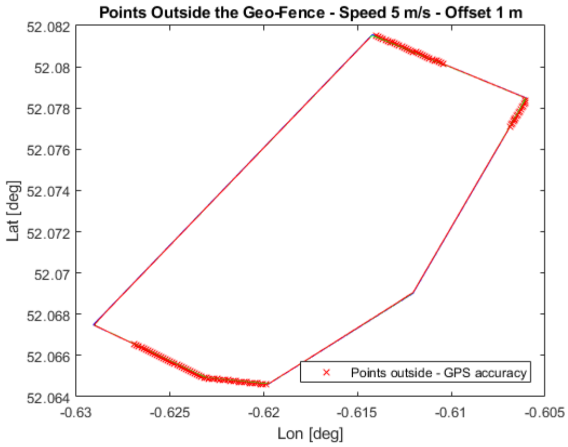
Table 6.1 shows the numerical results for the boundary surveillance mission at 5 m/s. At that speed can be observed that 1 meter of offset is enough to guarantee that the motion command due to path following is always inside the GF. The results in percentage represent the time that the UAV is outside the GF with respect the total mission time of one full cycle of the mission

Offset	**Path Following**			**PF Average**	**GPS accuracy**			**GPS average**
(m)	**Test 1**	**Test 2**	**Test 3**		**Test 1**	**Test 2**	**Test 3**	
1	0.00%	0.00%	0.00%	0.00%	1.97%	19.84%	0.00%	7.27%
3	0.00%	0.00%	0.00%	0.00%	2.99%	0.00%	2.15%	1.71%
5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

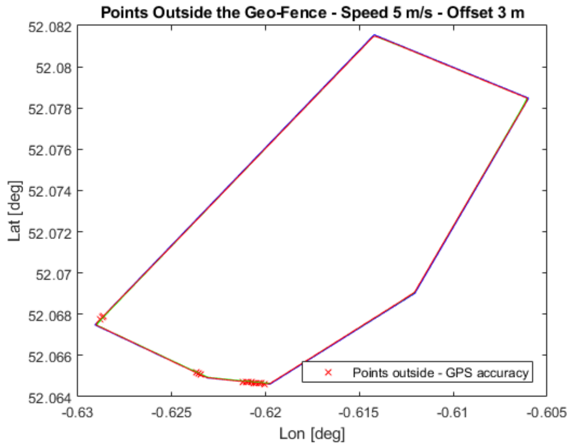
Table 6.1: Results for boundary surveillance mission at speed 5 m/s

Regarding the GNSS accuracy, from the table it is possible to deduce that increasing the offset, the possibilities of crossing the GF decrease, however, the variation between tests with the same conditions is very large as can bee seen in Table 6.1, the difference in percentage of time between tests 2 and 3 for 1 m offset due to GPS accuracy is almost 20 %. Therefore, more tests would be needed in order to verify these results.

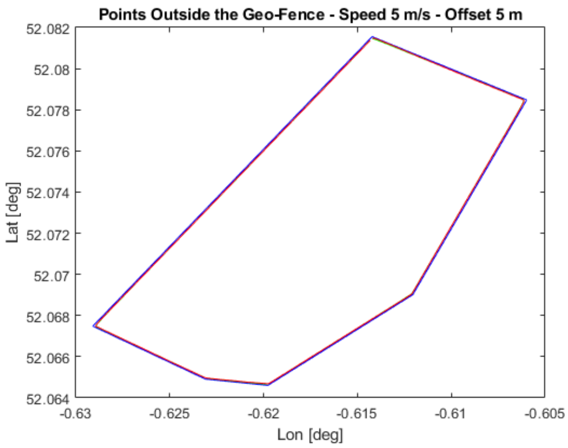
In the Figure 6.1 can be appreciated the locations where the GF is crossed. It is observer again how the number of crosses decreases when the offset increases and how the most of them occur after the turns in the Gf vertices.



(a) Offset 1 m



(b) Offset 3 m



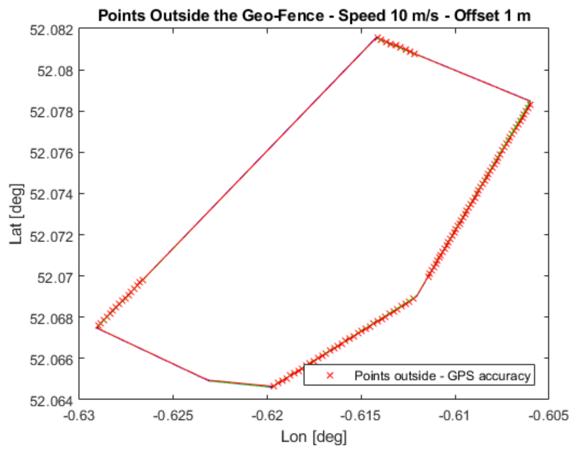
(c) Offset 5 m

Figure 6.1: Boundary surveillance at 5 m/s

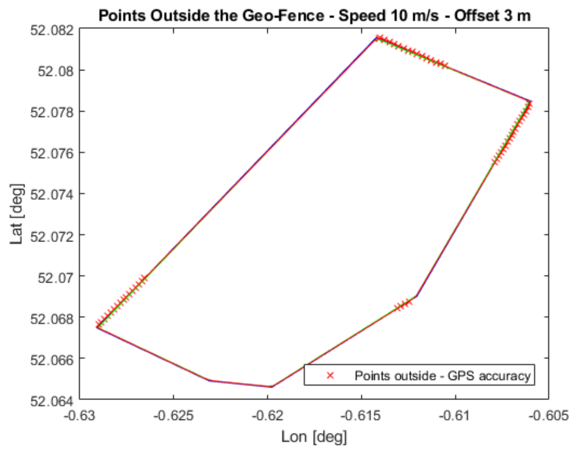
The same test has been done increasing the speed to 10 m/s. The results of is are shown in Table 6.2. One plot for each of the offsets are shown in Figure 6.2.

Offset	**Path Following**			**PF Average**	**GPS accuracy**			**GPS average**
(m)	**Test 1**	**Test 2**	**Test 3**		**Test 1**	**Test 2**	**Test 3**	
1	0.00%	0.00%	0.00%	0.00%	62.50%	44.57%	44.06%	50.38%
3	0.00%	0.00%	0.00%	0.00%	16.14%	21.01%	20.08%	19.08%
5	0.00%	0.00%	0.00%	0.00%	1.20%	7.22%	0.00%	2.81%
7.5	0.00%	0.00%	0.00%	0.00%	1.09%	0.00%	3.01%	1.36%

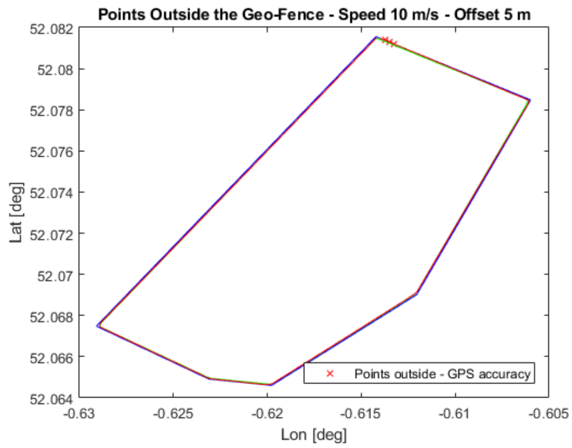
Table 6.2: Results for boundary surveillance mission at speed 10 m/s



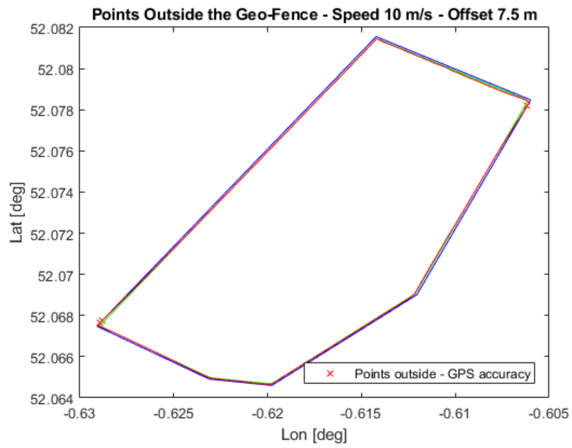
(a) Offset 1 m



(b) Offset 3 m



(c) Offset 5 m



(d) Offset 7.5 m

Figure 6.2: Boundary surveillance at 10 m/s

From the table and the figures can be stated that the path following algorithm with the current parameters does not produce enough deviation to the expected route to cross the boundary of the GF. However, due to GPS accuracy, it can be seen that increasing the offset, the probability of being outside the region decreases gradually as it is shown in Figures 6.3. But in order to be sure that the vehicle is always inside, a greater offset distance would be needed.

The reason why the path following algorithm has no effect in the boundary surveillance mission for both speeds is that the turns required to perform this type of mission are not very critical, as it will be shown when studying the interior surveillance mission.

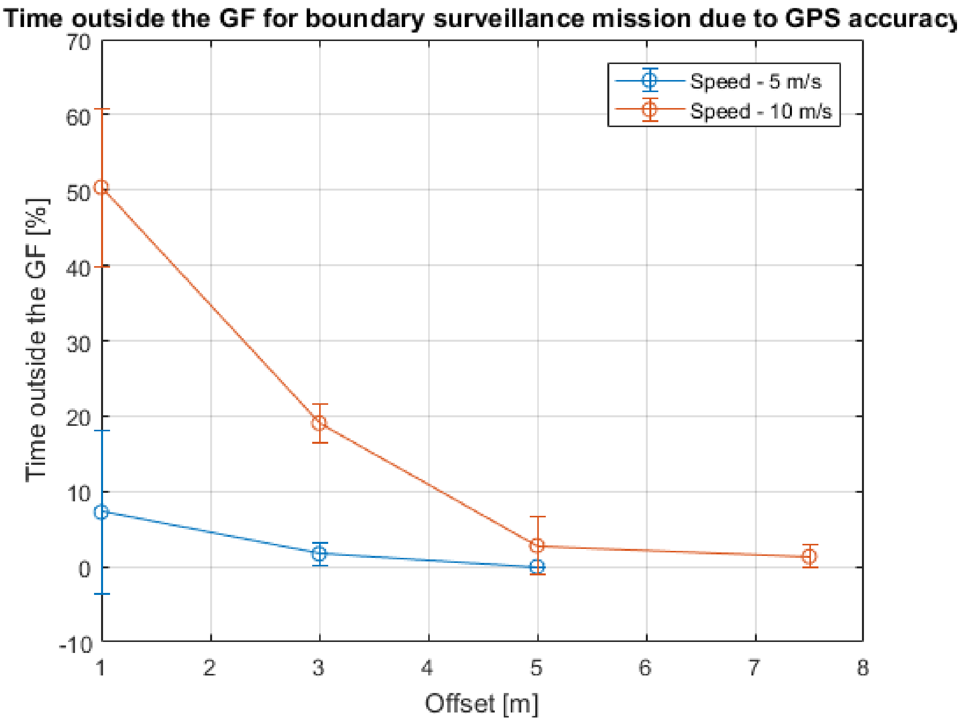


Figure 6.3: Time outside the GF with respect to the Offset due to GPS accuracy in Boundary surveillance mission

6.1.2 Interior surveillance mission

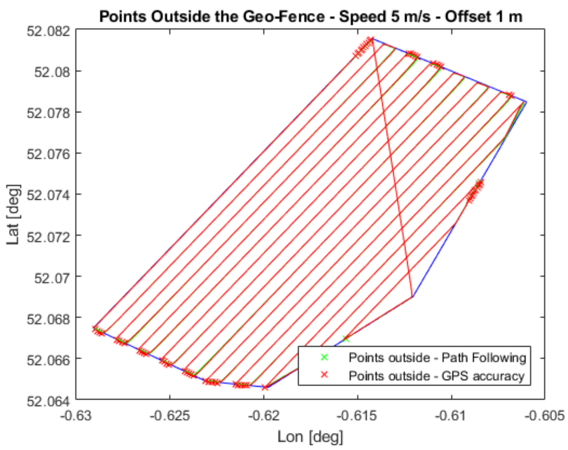
For this mission, the same tests have been done, for two different speeds (5 and 10 m/s) increasing progressively the offset in order to prove which is the minimum offset for this mission as a function of the velocity of the UAV.

The distance between each parallel line of the path followed by the vehicles is fixed at 50 m. This type of mission increases a lot the number of turns needed to perform it, therefore, the possibilities of crossing the boundary increase.

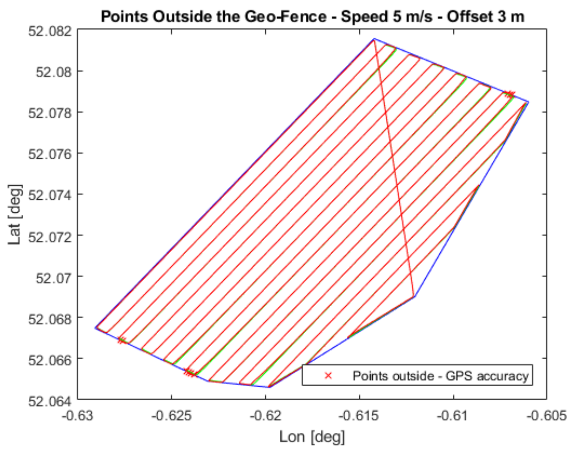
Offset (m)	Path Following Test 1	Path Following Test 2	Path Following Test 3	PF Average	GPS accuracy Test 1	GPS accuracy Test 2	GPS accuracy Test 3	GPS average
1	0.09%	0.13%	0.06%	0.09%	1.36%	0.90%	1.97%	1.41%
3	0.00%	0.00%	0.00%	0.00%	0.30%	0.43%	0.27%	0.33%
5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.06%	0.02%

Table 6.3: Results for interior surveillance mission at speed 5 m/s

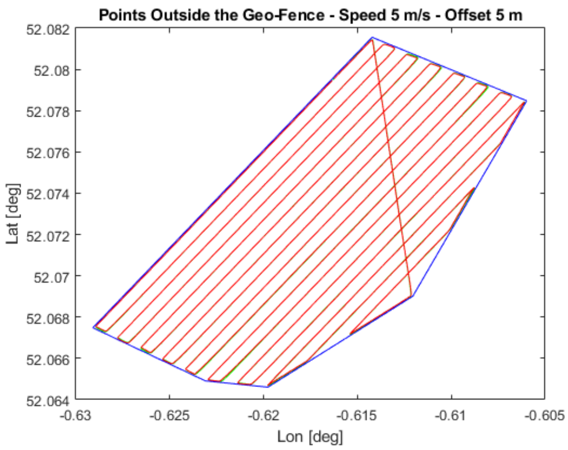
In Table 6.3 are shown the results for the interior surveillance mission at speed 5 m/s. Images of a significant case for each of the offsets tested are shown in Figure 6.4.



(a) Offset 1 m



(b) Offset 3 m



(c) Offset 5 m

Figure 6.4: Interior surveillance at 5 m/s

In the images can be seen that for this type of mission the angles of turn can reach angles of almost 180 degrees, which are very aggressive turns, but for this low speed, the turn can be performed using less space than the given by the offset. As it was explained in 4.3.1, the minimum turn radius depends on the maximum lateral acceleration, which depends on the speed, therefore, for low speeds, the vehicle is able to perform turns with a very low radius.

However, for an offset of 1 m, it can be observed that in some of the closest turns the trajectory of the vehicle crosses the GF.

Once again, it is proven that taking into account GPS accuracy, the probabilities of being outside the GF decrease when the length of the offset with the boundary increases as shown in Figure 6.5. However, more tests for this speed should be done to have an offset that gives more reliability.

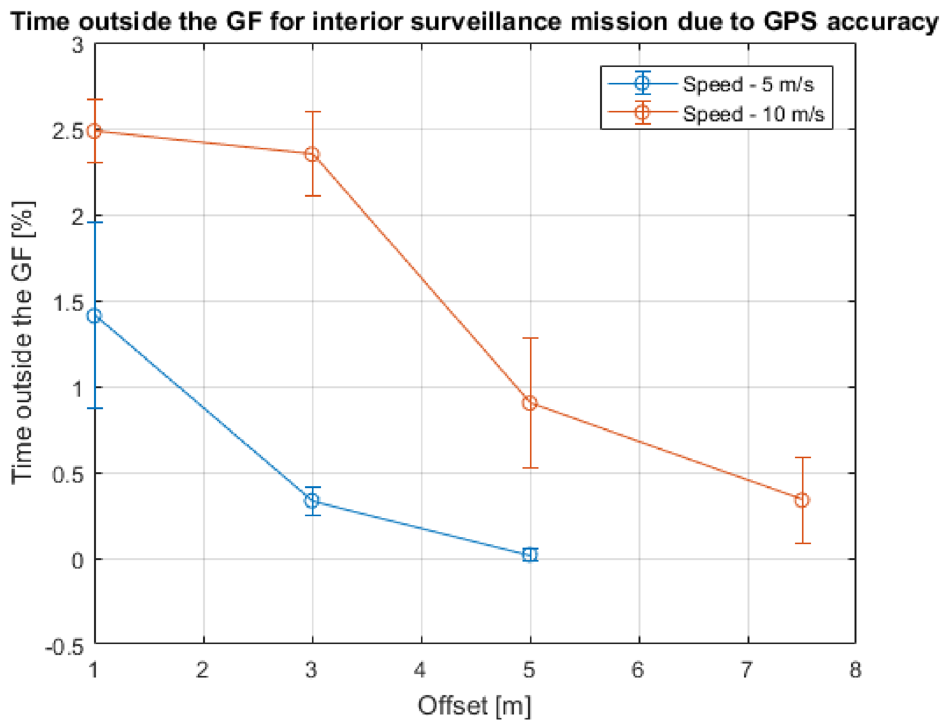
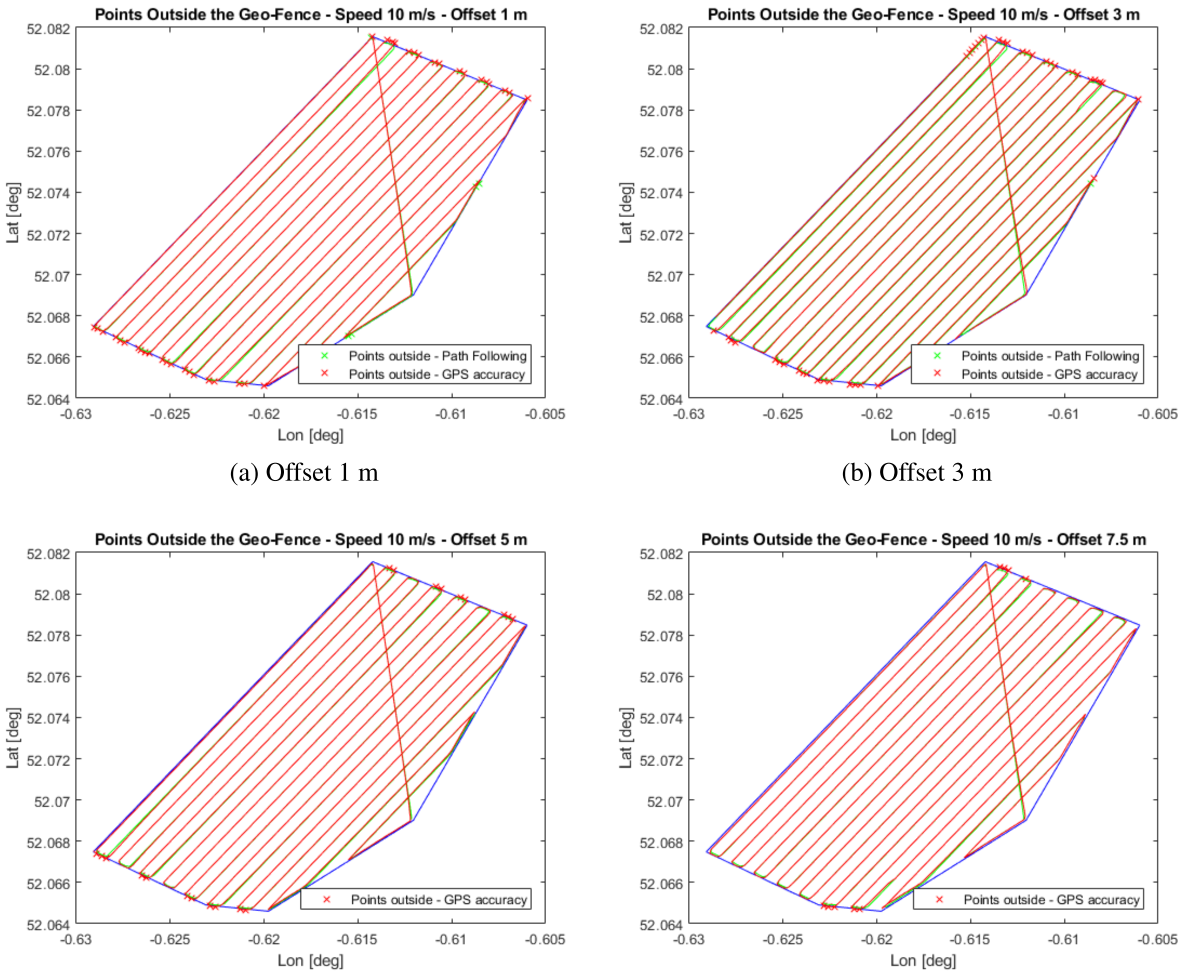


Figure 6.5: Time outside the GF with respect to the Offset due to GPS accuracy in Interior surveillance mission

The same test has been done increasing the speed to 10 m/s. The results of is are shown in Table 6.4. One plot for each of the offsets are shown in Figure 6.6.



(a) Offset 1 m

(b) Offset 3 m

(c) Offset 5 m

(d) Offset 7.5 m

Figure 6.6: Interior surveillance at 10 m/s

Offset	**Path Following**			**PF Average**	**GPS accuracy**			**GPS average**
(m)	**Test 1**	**Test 2**	**Test 3**		**Test 1**	**Test 2**	**Test 3**	
1	0.302%	0.303%	0.303%	0.303%	2.299%	2.486%	2.667%	2.484%
3	0.061%	0.061%	0.000%	0.041%	2.310%	2.614%	2.125%	2.350%
5	0.000%	0.000%	0.000%	0.000%	1.219%	0.999%	0.489%	0.902%
7.5	0.000%	0.000%	0.000%	0.000%	0.398%	0.060%	0.555%	0.338%

Table 6.4: Results for interior surveillance mission at speed 10 m/s

For this speed, it can be seen again a crossing with the GF due to Path Following. It is observed for low offset values (1 and 3 m). The influence of the offset in the time that the vehicle is outside due to Path Following algorithm is shown in Figure 6.7. This is due to the minimum turn radius explained before. Applying a zoom to Figure 6.6, we can see how the minimum turn radius is higher than the offset and adding to this the large turns typical of this type of mission, we get a real trajectory of the vehicle crossing the boundary of the region. See Figure 6.8.

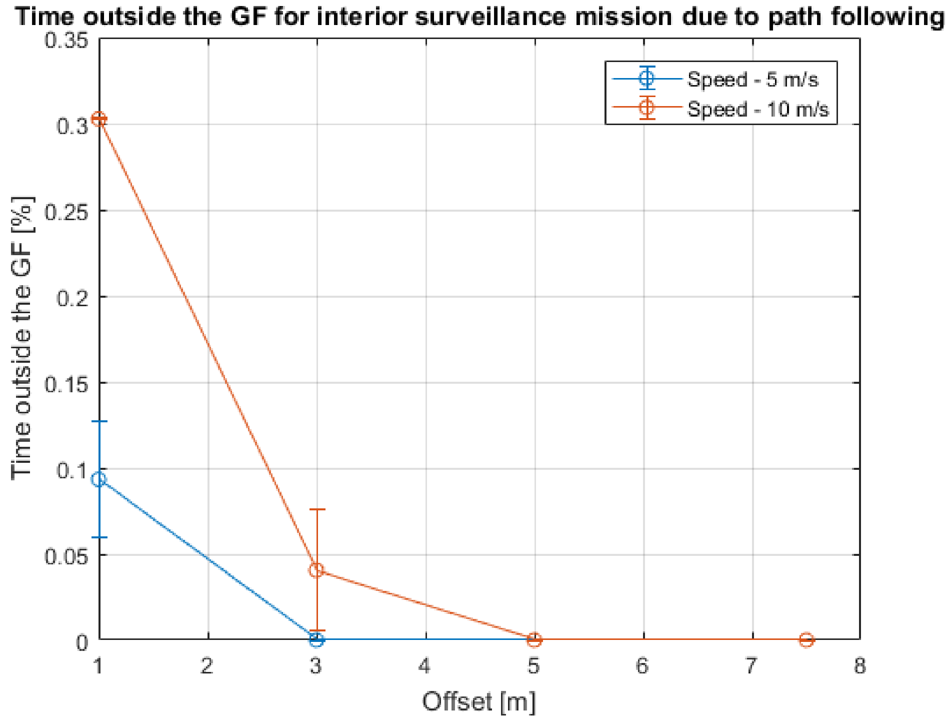


Figure 6.7: Time outside the GF with respect to the Offset due to Path Following in Interior surveillance mission

According to GPS accuracy, for 10 m/s the possibilities of being outside increase a lot. Being greater than 50 % on average for an offset of 1 m. Increasing the size of the offset, the probabilities decrease gradually.

From both boundary and interior surveillance missions can be observed how the points where the vehicle goes outside the GF are always during or after performing any turn. For small offsets, almost in every turn, the vehicle crosses the line. The interior surveillance

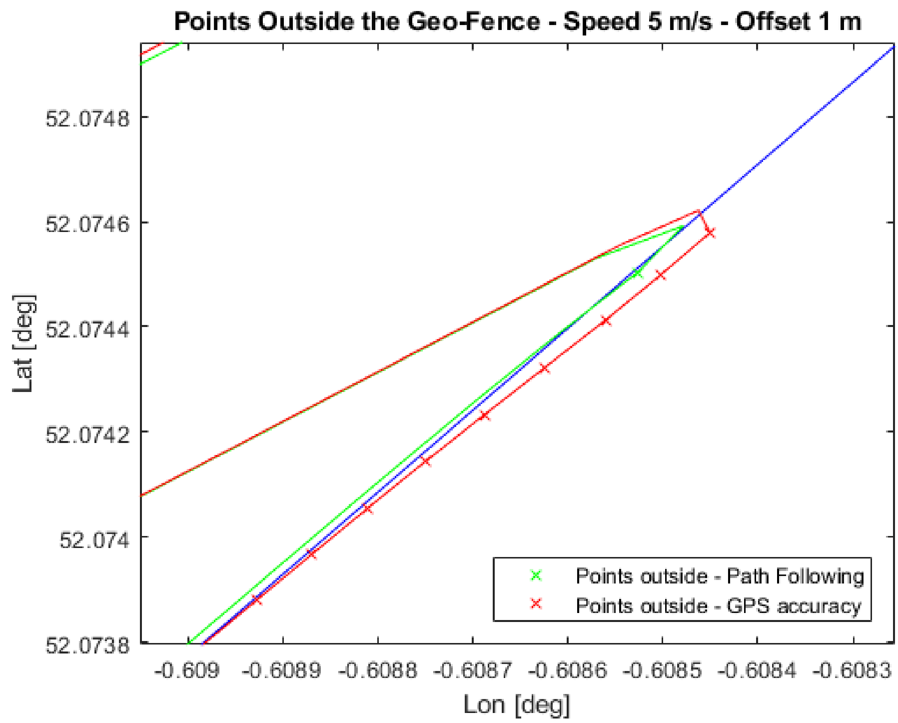


Figure 6.8: Zoom to show the cross with boundary due to Path Following

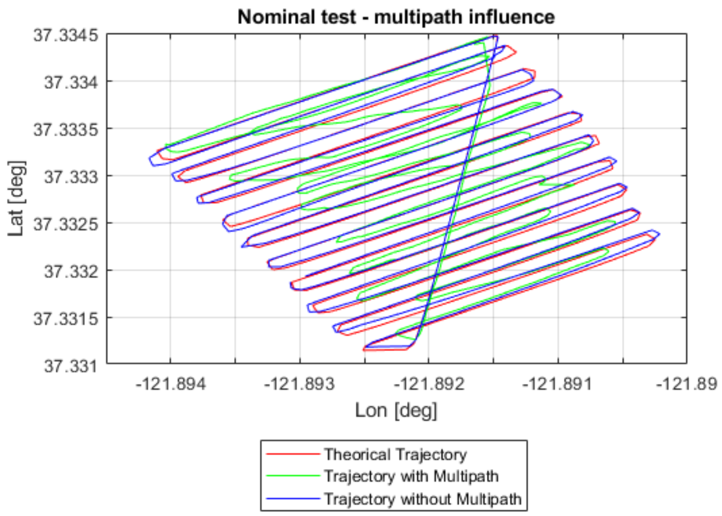
mission needs more offset distances in order to guarantee the safety and reliability of the mission due to the large number of turns that vehicles perform with this mission.

Finally, with this amount of tests it was possible to extract some conclusions and to observe several trends (Figures 6.3, 6.5 and 6.7) that have been explained in this chapter. However, it would be necessary to repeat several more times every test in order to have a larger and more significant sample, and also to add more offset distances and speeds to have an accurate view of the problem and extract reliable conclusions.

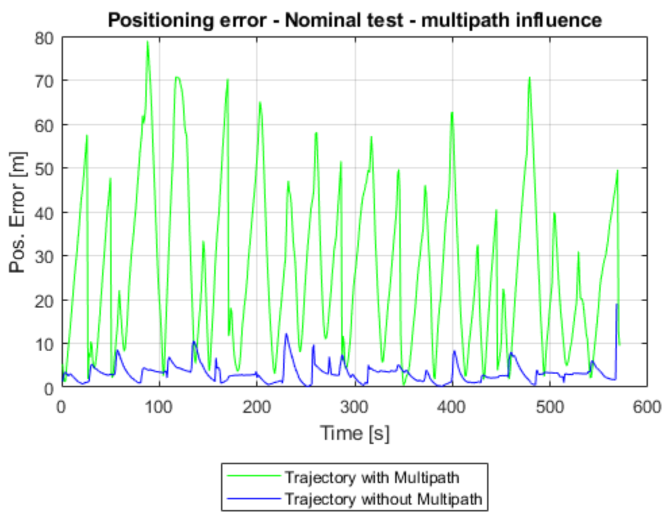
6.2 Multi-Path influence study

The purpose of this test is to study the influence of Multi-path in the GNSS RF in urban environments. To see the influence of Multi-path, the three tests explained in Table 5.3 have been done using the Multi-path software and without it.

The results for the nominal case are shown in Figure 6.9. In Figure 6.9(a) is shown the trajectory of a UAV flying in the conditions described above for both cases, with and without Multi-path. Its influence is very large, being greater near the turns. This can be explained because of the presence of high buildings.



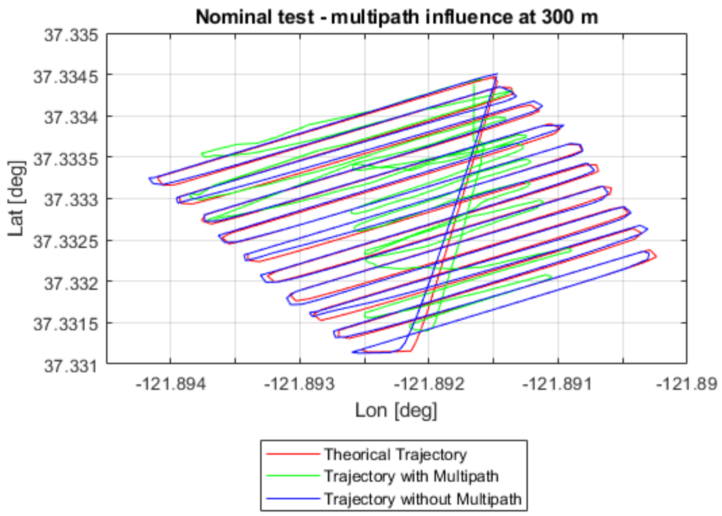
(a) Trajectory



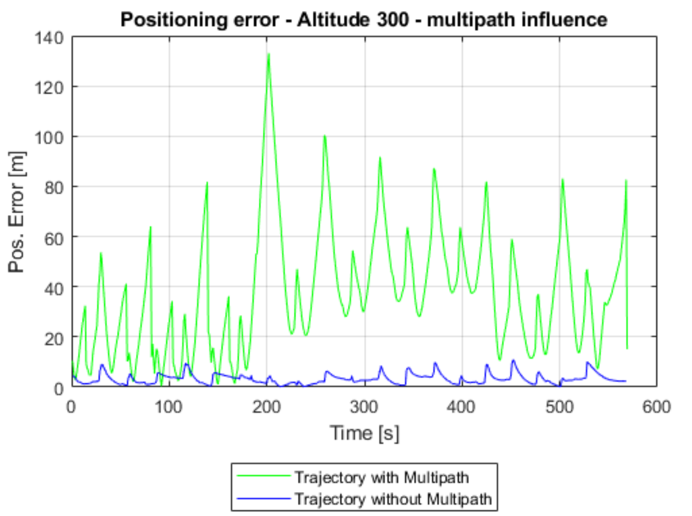
(b) Error

Figure 6.9: Trajectory and Positioning error in m with and without Multi-path for nominal case

In Figure 6.9 (b) is shown the positioning error calculated in meters with respect to the time of the mission. Here can be also observed how the greatest errors occur cyclically, corresponding with the turns after each straight parallel line. For this nominal case, the maximum positioning error is below 80 m, which is very high but is the lowest of all the tests done with Multi-path. The mean error is around 20 m.



(a) Trajectory



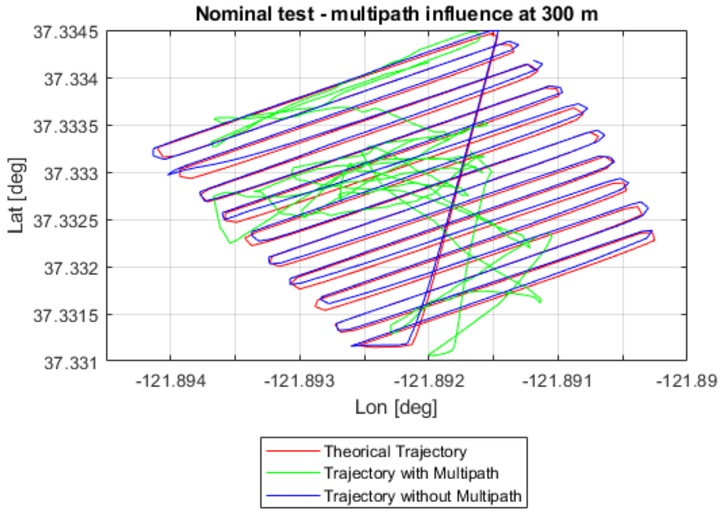
(b) Error

Figure 6.10: Trajectory and Positioning error in m with and without Multi-path for altitude 300 m

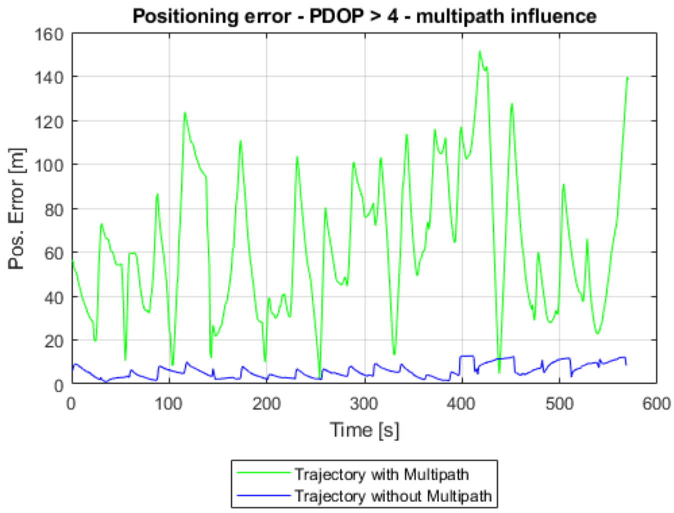
Increasing the altitude to 300 m, the results are quite similar but the error is bigger than before. This is causes because 300 m is not enough to have a decrease in the power level that makes the receiver refusing the signal. For higher altitudes, the loss of power would be enough and the receiver would refuse the signals, having then smaller positioning error.

Therefore, there exists an altitude with the highest Multi-path influence.

Results for the altitude set at 300 m are shown in Figure 6.10. Here the maximum positioning error reaches almost 140 m, which is quite large taking into account that the longest side of the GF is 300 m length. The mean error during the missions is 37 m.



(a) Trajectory



(b) Error

Figure 6.11: Trajectory and Positioning error in m with and without Multi-path for PDOP over 4

The influence of Multi-path in these two test is large taking into account that both are done at the same UTC time, which means that the PDOP caused by the relative position between satellites is low. The PDOP of both tests is lower than 4, which is Good or Excellent according to the standard that gives the confidence level. However, for the third test, with a PDOP always above 4, the confidence level is moderate. Therefore, the error comes not only from Multi-path but also due to the PDOP.

Results of this test are shown in Figure 6.11. It is shown in Figure 6.11(a) that the trajectory of the vehicle when it is influenced by Multi-path is impossible to follow and has a lot of points crossing the boundary of the GF. The error is extremely large, and not only is possible to see how it is larger than in the two previous tests when Multi-path is applied, but it is also greater for the simulation without it. Therefore this can be explained because of the increase in the PDOP.

In Table 6.5 is shown the mean error and the maximum error for each of the tests performed with and without the Multi-path influence. From the table is observed how both altitude and PDOP over 4 have influence, being the second one the highest having an average of the error of 63 m which is more than the 20% of the longest side of the GF.

	Multi-path		**No Multi-path**	
	Max. Error (m)	**Mean Error (m)**	**Max. Error (m)**	**Mean Error (m)**
Nominal	78.90	27.00	19.02	3.17
Altitude 300	133.12	37.60	10.88	3.26
PDOP > 4	151.62	63.17	12.75	5.90

Table 6.5: Mean and maximum errors of Multi-path tests

From these three tests is possible to see the influence of Multi-path when performing missions in Urban Environments. And can be stated that the biggest errors occur during the turns of the mission. However, the tests are not enough to extract accurate conclusions.

In order to have accurate and reliable results, it would be necessary to repeat each of the tests done several times and obtain the averages of these tests. However, this was not possible during the thesis due to the location of the Multi-path software and the time duration of each test.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This Individual Research Project is focused on the analysis of UAV performance inside Geo-Fenced environments. To do so, three different software modules were developed in order to obtain results from two different performance analysis. Firstly, a literature review was performed in order to understand the current state of the topic and to study the different points of view to choose the best scope for the project. After this, and taking into account the requirements of the sponsor company, the structure and the design of the project is presented. Then, two different performance analysis are developed to test the designed software and algorithms and to study the UAV performance within Geo-Fences. Finally, in the last chapter, the results of these two experiments are explained. The main contributions to knowledge of this project are the development of a 4D Geo-Fencing software, the study of the minimum offset required to guarantee the safety of operations and a qualitative study of the influence of Multi-path in urban environments.

The first aim of the project was the development of a GF library compatible with the GNSS simulator of Spirent. This software was successfully developed with the following features:

- The software is able to generate circular and polygonal (convex and non-convex)

Geo-Fences. It can generate them manually, and from .txt and .kml files. This option is very useful in order to use the software together with Google Earth or any other generator of .kml files

- The algorithm to compare relative position between a GF and a vehicle works properly for both circular and polygonal GFs. In addition, it also calculates the relative position taking into account the maximum altitude of the GF.

- Furthermore, a time comparator was added to compare the current time of simulation with the start and end time of each GF using UTC time, and it works properly. This extra feature is one of the novelties of this project.

The motion planner developed to simulate UAV trajectories has some limitations but it provides enough information for the purpose of the tests performed in this project. In order to use it for more complex simulations (accurate airframe of the vehicle or higher update rate) further development or the use of a commercial flight simulator would be needed.

The last part of the software developed is the Scheduler. This software has been developed to manage the UAVs flying inside each GF giving them missions. This software is one of the novelties of the IRP.

The developed software of this project has limited number of features. Because of this, it can be used only in scenarios with some characteristics: non-convex polygonal GFs, a low update rate and some other limited characteristics explained in Chapter 4. In order to use this software in a wider set of scenarios, further development would be needed. Some of this upgrades are explained in Future Work.

Related to the Speed-Offset analysis, it can be stated that the minimum offset required increases with the speed of the vehicle. This is due to two main reasons, the accuracy of GNSS and the update time of the positioning explained before. For a speed of 5 m/s and in a clear scenario, 5 meters of offset is enough to guarantee that the vehicle is always inside the region with the number of tests done. However, in order to have more reliable results, a larger number of test with different scenarios and different speeds should be done to ensure these results. For a speed of 10 m/s 7.5 m of offset are not enough to guarantee the security of the of any of the two mission.

It can also be stated that the maximum errors are always observed in the areas where the vehicle is performing turns, therefore, the interior surveillance mission has larger positioning errors than the boundary surveillance mission.

Finally, from the Multi-path analysis, only qualitative conclusions can be extracted. In order to obtain some quantitative conclusions, a larger number of tests should be performed. However, some findings can be extracted from this analysis:

- The influence of Multi-path in usban environments is huge (Maximum positioning errors greater than 150 m.
- The time to first fix (TTFF) is of the receiver very large with Multi-path due to the number of unexpected signals processed. The receiver needs to receive the Almanac (12.5 min) to get the positioning.
- The influence of Multi-path increases with the altitude if the receiver is not able to refuse the signal due to the loss of power. From the tests of this project, the average error increases 10 meters when increasing the altitude 150 m.

It can be observed the enormous influence of Multi-path in GNSS accuracy within urban environments.In addition, it was observed the time that the receiver need to get a fix when Multi-path is present is very large, needing to wait more than 10 minutes to get the Almanac from the satellites. It is also observed that the influence of Multi-path increases with the altitude if the receiver is not able to refuse the signal due to the loss of power.

Therefore, using only GNSS positioning while developing missions inside urban environments is not enough, and some extra positioning technology (Inertial, image-based, etc.) should be used together with GNSS to guarantee an accurate positioning.

7.2 Future Work

Taking into account the number of topics of this project, there are several fields where a future work could be developed:

- Firstly, to guarantee the reliability of the results of both performance analysis, a larger number of tests, with different scenarios and speeds should be done.
- Related to the software developed, a more efficient and accurate motion planner could be developed which have a time step lower than the one of this project. Also using accurate Air-Frames depending on the parameters of the UAVs.
- The Geo-Fencing library could be improved in different ways. One of the most interesting improvements of this software would be to have GFs able to change their shape with the time. There is not too much information in the literature referred to this topic and it would be a very useful tool for the industry and the UTM technologies.
- The Multi-path analysis of this project shows a general scope of its influence, but positioning in urban environments accurately is an issue very relevant for the industry and a deeper research in this area is needed.

References

Aireon (2018), 'Executive Reference Guide to Space-Based ADS-B'.

Aslund, S., Jensen, K. and Jrgensen, R. N. (2011), 'Complete coverage path planning of a random polygon - A FroboMind component'.

Blanch, J., Walter, T. and Enge, P. (2012), 'Satellite Navigation for Aviation in 2025', *Proceedings of the IEEE* **100**(Special Centennial Issue), 1821–1830.
URL: *http://ieeexplore.ieee.org/document/6184264/*

Cacciola, F. (n.d.), 'A CGAL implementation of the Straight Skeleton of a Simple 2d Polygon with Holes', p. 4.

Cai, G., Chen, B. M. and Lee, T. H. (2011), Coordinate Systems and Transformations, *in* 'Unmanned Rotorcraft Systems', Springer London, London, pp. 23–34.
URL: $http://link.springer.com/10.1007/978-0-85729-635-1_2$

Corporations, U. A. S. (2016), 'Understanding Compliance with Automatic Dependent Surveillance-Broadcast (ADS-B) Out', p. 8.

EVK-8/EVK-M8 (2018).
URL: *https://www.u-blox.com/en/product/evk-8evk-m8*

Farouki, R. T. (n.d.), 'Pythagorean-Hodograph Curves I', *Geometry Processing for Design and Manufacturing* p. 31.

Grewal, M. S., Weill, L. R. and Andrews, A. P. (n.d.), *Global Positioning Systems, Inertial Navigation, and Integration*, Wiley.

Han, S., Gong, Z., Meng, W., Li, C. and Gu, X. (2016), 'Future Alternative Positioning, Navigation, and Timing Techniques: A Survey', *IEEE Wireless Communications* **23**(6), 154–160.
URL: *http://ieeexplore.ieee.org/document/7593455/*

ICAO (2011), *Unmanned aircraft systems: UAS*, number 328 *in* 'ICAO circular', International Civil Aviation Organization, Montral. OCLC: 729908130.

Kariuki, L. W., Ikua, B. W. and Nyakoe, G. N. (2014), 'Generation and Optimization of Pocket Milling Tool Paths - A Review', **5**, 5.

Koks, D. (n.d.), 'Using Rotations to Build Aerospace Coordinate', p. 42.

Kopardekar, P. (2014*a*), 'Enabling Civilian Low-Altitude Airspace and Unmanned Aerial System (UAS) Operations By Unmanned Aerial System Traffic Management (UTM)'.

Kopardekar, P. (2014*b*), 'Unmanned Aerial System (UAS) Traffic Management (UTM): Enabling Low-Altitude Airspace and UAS Operations'.

Lazarus, S. B., Tsourdos, A., White, B. A., Silson, P. and bikowski, R. (2010), 'Cooperative unmanned aerial vehicle searching and mapping of complex obstacles using two-dimensional splinegon', *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* **224**(2), 149–170.
URL: *http://journals.sagepub.com/doi/10.1243/09544100JAERO585*

Pala, M., Eraghi, N., Lpez-Colino, F., Sanchez, A., de Castro, A. and Garrido, J. (2013), 'HCTNav: A Path Planning Algorithm for Low-Cost Autonomous Robot Navigation in Indoor Environments', *ISPRS International Journal of Geo-Information* **2**(3), 729–748.
URL: *http://www.mdpi.com/2220-9964/2/3/729*

Palfrader, P. and Held, M. (2015), 'Computing Mitered Offset Curves Based on Straight

Skeletons', *Computer-Aided Design and Applications* **12**(4), 414–424.
URL: *http://www.tandfonline.com/doi/abs/10.1080/16864360.2014.997637*

Peng Xie and Petovello, M. G. (2015), 'Measuring GNSS Multipath Distributions in Urban Canyon Environments', *IEEE Transactions on Instrumentation and Measurement* **64**(2), 366–377.
URL: *http://ieeexplore.ieee.org/document/6873332/*

Podradchik, S. (2015), 'FlyQ EFB from Seattle Avionics ADS-B Primer'.

Pratyusha, P. L. (2015), 'Geo-Fencing for Unmanned Aerial Vehicle', *International Journal of Computer Applications* p. 7.

Sabatini, R., Moore, T. and Ramasamy, S. (2017), 'Global navigation satellite systems performance analysis and augmentation strategies in aviation', *Progress in Aerospace Sciences* **95**, 45–98.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S037604211730163X*

Samad, A. M., Kamarulzaman, N., Hamdani, M. A., Mastor, T. A. and Hashim, K. A. (2013), The potential of Unmanned Aerial Vehicle (UAV) for civilian and mapping application, *in* 'Samad', IEEE, pp. 313–318.
URL: *http://ieeexplore.ieee.org/document/6650191/*

Seo, D. M.-G. (n.d.), 'Guidance &Navigation for Autonomous Systems (GNAS)', p. 65.

Shanmugavel, M., Tsourdos, A., White, B. A. and bikowski, R. (2007), 'Differential Geometric Path Planning of Multiple UAVs', *Journal of Dynamic Systems, Measurement, and Control* **129**(5), 620.
URL: *http://DynamicSystems.asmedigitalcollection.asme.org/article.aspx?articleid=1412601*

Staff, G. W. (2018), '2018 SIMULATOR BUYERS GUIDE.', *GPS World* **29**(3), 12–27.
URL: *https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=128622999&site=ehost-live*

Stevens, M. N., Coloe, B. and Atkins, E. M. (2015), Platform-Independent Geofencing for Low Altitude UAS Operations, *in* 'stevens', American Institute of Aeronautics and Astronautics.
URL: *http://arc.aiaa.org/doi/10.2514/6.2015-3329*

van Blyenburgh, P. (1999), 'UAVs: an overview', *Air & Space Europe* .

WANG, S. (n.d.), 'Detection and Analysis of GNSS Multipath', p. 53.

Appendices

Appendix A - C++ code

In this appendix are shown some examples of the managed C++ code used during the development of the software of this project. The software of the three different part is developed in three different projects of Windows forms. Therefore, each project is completely developed in only one file. On the following sections is shown the most significant parts of the code of each program.

Motion Planner Code - C++

The UAV class is defined as:

```
public ref class UAV {
public:
  UAV(double speed, double heading, double altitude,
     double time_iteration, double lat_ref, double
     lon_ref);
  ~UAV() {}
  void UpdateInitPos();
  void UpdateInitVel();
  void TrajGen();
  void disp_params();
  void Ned2Ecef(double pos_nedX, double pos_nedY, double
     pos_nedZ, double lat_ref, double lon_ref);
  void Ned2EcefVel(double vel_nedX, double vel_nedY,
     double vel_nedZ, double lat_ref, double lon_ref);
  double* GetVelNed(double speed, double climb_rate,
     double heading);
  void Ecef2Lla(double pos_ecef_X, double pos_ecef_Y,
     double pos_ecef_Z);
  double pos_init_nedX;
  double pos_init_nedY;
  double pos_init_nedZ;

  double vel_init_nedX;
  double vel_init_nedY;
```

```
  double vel_init_nedZ;

  double pos_final_nedX;
  double pos_final_nedY;
  double pos_final_nedZ;

  double vel_final_nedX;
  double vel_final_nedY;
  double vel_final_nedZ;

  double pos_ecefX;
  double pos_ecefY;
  double pos_ecefZ;

  double vel_ecefX;
  double vel_ecefY;
  double vel_ecefZ;

  double pos_llaY;
  double pos_llaZ;
  double pos_llaX;

  double timestep;

  //parametres given by the user
  double desired_speed; // m/s
  double desired_heading; // deg
  double desired_altitude; // m
  double max_acceleration; // m/s2
  double max_heading_rate; // deg/s
  double max_climb_rate; // m/s

        //calculated and displayed parameters
  double current_speed;
  double current_heading;
  double current_altitude;
};
```

The motion equations when all the parameters (Speed, heading and altitude) have been modified:

```
// If there is a desired change in speed, heading and
   altitude
else if (abs(current_altitude - desired_altitude) > 0.001
    && abs(current_speed - desired_speed) > 0.001
  && abs(current_heading - desired_heading) > 0.001) {

  double max_acc = max_acceleration;

  if (current_speed > desired_speed) {
    max_acc = -max_acceleration;
  }

  if (abs(current_speed - desired_speed) < 0.2) {
    max_acc = 0.1 * max_acc;
  }

  double max_acceleration_x = max_acc * cos(
     current_heading*PI / 180);
  double max_acceleration_y = max_acc * sin(
     current_heading*PI / 180);
```

```
  double climb_rate = max_climb_rate;

  if (current_altitude < desired_altitude) {
    climb_rate = -max_climb_rate;
  }

  if (abs(current_altitude - desired_altitude) < 0.1) {
    climb_rate = 0.1 * climb_rate;
  }

  double head_rate = max_heading_rate;

  if (current_heading < desired_heading) {
    head_rate = max_heading_rate;
    if ((-current_heading + desired_heading) > 180) {
      head_rate = -max_heading_rate;
    }
  }

  if (current_heading > desired_heading) {
    head_rate = -max_heading_rate;
    if ((current_heading - desired_heading) > 180) {
      head_rate = max_heading_rate;
    }
  }


  if (abs(current_heading - desired_heading) < 0.1) {
    head_rate = 0.1 * head_rate;
  }
  double* vel = GetVelNed(current_speed, climb_rate,
     current_heading);

  vel_init_nedX = *vel;
  vel_init_nedY = *(vel + 1);
  vel_init_nedZ = *(vel + 2);

  current_heading = current_heading + head_rate *
     timestep;
  current_heading = fmod(current_heading, 360);
  if (current_heading > 180) {
    current_heading = current_heading - 360;
  }

  pos_final_nedX = pos_init_nedX + vel_init_nedX *
     timestep + 0.5 * max_acceleration_x * pow(timestep,
     2);
  pos_final_nedY = pos_init_nedY + vel_init_nedY *
     timestep + 0.5 * max_acceleration_y * pow(timestep,
     2);
  pos_final_nedZ = pos_init_nedZ + vel_init_nedZ *
     timestep;

  vel_final_nedX = vel_init_nedX + max_acceleration_x *
     timestep;
  vel_final_nedY = vel_init_nedY + max_acceleration_y *
     timestep;
  vel_final_nedZ = vel_init_nedZ;


  current_altitude = -pos_final_nedZ;
```

```

  current_speed = sqrt(pow(vel_final_nedX, 2) + pow(
     vel_final_nedY, 2));

  if (abs(current_speed - desired_speed) <= 0.01) {
    current_speed = desired_speed;
  }

  if (abs(current_altitude - desired_altitude) <= 0.01) {
    current_altitude = desired_altitude;
  }


  if (abs(current_heading - desired_heading) <= 0.01) {
    current_heading = desired_heading;
  }
}
```

Conversions between coordinate systems are calculated as follow:

```
void UAV::Ned2Ecef(double pos_nedX, double pos_nedY,
     double pos_nedZ, double lat_ref, double lon_ref) {

  double lat = lat_ref * PI / 180;
  double lon = lon_ref * PI / 180;
  double h = 0;


  double ned[3];
  ned[0] = pos_nedX;
  ned[1] = pos_nedY;
  ned[2] = pos_nedZ;


  ////reference_point in ECEF
  double e = 8.1819190842622e-2;; //eccentricity
  double Rea = 6378137; //Radius of Earth [m]
  double Ne = Rea / sqrt(1 - pow(e, 2) * pow(sin(lat), 2)
     );

  double x_ref_ecef = (Ne + h)*cos(lat)*cos(lon);
  double y_ref_ecef = (Ne + h)*cos(lat)*sin(lon);
  double z_ref_ecef = (Ne*(1 - pow(e, 2)) + h)*sin(lat);


  // refLat y refLon est n dadas en radianes
  pos_ecefX = -ned[0] * sin(lat)*cos(lon) - ned[1] * sin(
     lon) -
    ned[2] * cos(lat)*cos(lon) + x_ref_ecef;
  pos_ecefY = -ned[0] * sin(lat)*sin(lon) + ned[1] * cos(
     lon) -
    ned[2] * cos(lat)*sin(lon) + y_ref_ecef;
  pos_ecefZ = ned[0] * cos(lat) - ned[2] * sin(lat) +
     z_ref_ecef;

}

void UAV::Ned2EcefVel(double vel_nedX, double vel_nedY,
     double vel_nedZ, double lat_ref, double lon_ref) {

  double lat = lat_ref * PI / 180;
```

```
  double lon = lon_ref * PI / 180;
  double h = 0;
  double uEast = vel_nedY;
  double vNorth = vel_nedX;
  double wUp = -vel_nedZ;

  double cosPhi = cos(lat);
  double sinPhi = sin(lat);
  double cosLambda = cos(lon);
  double sinLambda = sin(lon);

  double t = cosPhi*wUp - sinPhi*vNorth;
  vel_ecefZ = sinPhi*wUp + cosPhi*vNorth;

  vel_ecefX = cosLambda*t - sinLambda*uEast;
  vel_ecefY = sinLambda*t + cosLambda*uEast;

}

void UAV::Ecef2Lla(double pos_ecef_X, double pos_ecef_Y,
    double pos_ecef_Z) {
  double x = pos_ecef_X;
  double y = pos_ecef_Y;
  double z = pos_ecef_Z;
  const double a = 6378137.0;           // Equatorial
     Radius
  const double e = 8.1819190842622e-2;  // Eccentricity

  double Lat, N, NplusH, delta, esLat;
  uint16_t iter;

  pos_llaY = atan2(y, x) * 180 / PI;
  N = a;
  NplusH = N;
  delta = 1;
  Lat = 1;
  iter = 0;

  while (((delta > 1.0e-14) || (delta < -1.0e-14)) && (
     iter < 100))
  {
    delta = Lat - atan(z / (sqrt(x*x + y * y)*(1 - (N*e*e
        / NplusH))));
    Lat = Lat - delta;
    esLat = e * sin(Lat);
    N = a / sqrt(1 - esLat * esLat);
    NplusH = sqrt(x*x + y * y) / cos(Lat);
    iter += 1;
  }

  pos_llaZ = NplusH - N;
  double lat = Lat * 180 / PI;
  pos_llaX = lat;

}
```

Creation of UDP and UTC sockets to send the motion command to the simulator and first step of the infinite loop

```
//Creates a TCPClient using a local end point.

```

```
ipAddress = IPAddress::Parse(this->IPserver->Text);
ipAddress2 = IPAddress::Parse("192.168.1.1");


if (local == true) {

  ipLocalEndPoint = gcnew IPEndPoint(ipAddress, 56000);
  ipEndPoint = gcnew IPEndPoint(ipAddress, 15650);
  serverTCP = gcnew TcpClient(ipLocalEndPoint);
  serverTCP->Connect(ipAddress, 15650);
}

if (lan == true) {

  ipLocalEndPoint = gcnew IPEndPoint(ipAddress, 56000);
  ipEndPoint = gcnew IPEndPoint(ipAddress2, 15650);
  serverTCP = gcnew TcpClient(ipLocalEndPoint);
  serverTCP->Connect(ipAddress2, 15650);
}

//ipLocalEndPoint = gcnew IPEndPoint(ipAddress, 56000);
//ipEndPoint = gcnew IPEndPoint(ipAddress, 15650);
//serverTCP = gcnew TcpClient(ipLocalEndPoint);
//serverTCP->Connect(ipAddress, 15650);


//Send message to start simulation in SimGen.
message = "RU_NOWAIT";
data = System::Text::Encoding::ASCII->GetBytes(message);
NetworkStream^ stream = serverTCP->GetStream();
if (stream->CanWrite) {
  stream->Write(data, 0, data->Length);
}

//Create UDP socket
serverUDP = gcnew UdpClient();
//serverUDP->Connect(ipAddress, 15650);

// UDP Message
memset(&simgen, 0, sizeof(simgen)); // Poner ceros en
    todo el "vector"
simgen.mot_command_.time_action_ = UDP_rx_command::
    Command::action_immediately;
simgen.mot_command_.time_of_validity_ms_ = 0;
simgen.mot_command_.type_ = UDP_rx_command::Command::mot;
simgen.mot_command_.vehicle_id_ = 1;
start_s = clock();

UAV1->TrajGen();
UAV1->UpdateInitPos();
UAV1->UpdateInitVel();

UAV1->Ned2Ecef(UAV1->pos_final_nedX, UAV1->pos_final_nedY
    , UAV1->pos_final_nedZ, LocalFrame1->latitude,
    LocalFrame1->longitude);


UAV1->Ecef2Lla(UAV1->pos_ecefX, UAV1->pos_ecefY, UAV1->
    pos_ecefZ);


```

```
simgen.mot_command_.data.mot_.position_ecef_xyz_[0] =
   UAV1->pos_ecefX;
simgen.mot_command_.data.mot_.position_ecef_xyz_[1] =
   UAV1->pos_ecefY;
simgen.mot_command_.data.mot_.position_ecef_xyz_[2] =
   UAV1->pos_ecefZ;
size_t message_size = sizeof(UDP_rx_command::Command) +
   32;

//Send message UDP to Simgen
buf = gcnew IntPtr(&simgen.mot_command_);
      //Creater a pointer to the initial position of the
     mot command
sendBytes = gcnew array<byte>(message_size);
             //Initialises the array<byte> with the
   lenght of the message
System::Runtime::InteropServices::Marshal::Copy((IntPtr)
   buf, sendBytes, 0, message_size); //Copy the memory
   block of the mot command into the array of bytes
serverUDP->Send(sendBytes, sendBytes->Length, ipEndPoint)
   ;                //send the block of memory to simgen

...


stop_s = clock();
time_used = stop_s - start_s;
if (time_iteration_ms - time_used-time_archived < 0) {
  time_archived = abs(time_iteration_ms - time_used -
     time_archived);
}
else {
  Sleep(time_iteration_ms - time_used - time_archived);
  time_archived = 0;
}
start_s = stop_s;
backgroundWorker1->RunWorkerAsync(1);  //starting
   background worker
Sleep(time_iteration_ms);
```

Geo-Fencing Code - C++

The GeoFence class is given by:

```
public ref class GeoFence {
public:
  GeoFence(String^ ID_s, String^ type_s, int
     num_of_points, array<double>^ X_points, array<double
     >^  Y_points, double max_altitude, DateTime^ time_in
     , DateTime^ time_en);
  GeoFence(String^ ID_s, String^ type_s, double X_centre,
      double Y_centre, double radius, double max_altitude
     , DateTime^ time_in, DateTime^ time_en);
  ~GeoFence() {}
  bool UAVinside(Vehicle^ uav);


  String^ type;
```

```
  String^ ID;
  int npoints;
  array<double>^ X_values;
  array<double>^ Y_values;
  double max_alt;
  int count;
  double X_cent;
  double Y_cent;
  double rad;
  double distance;
  DateTime^ time_ini;
  DateTime^ time_end;

};
```

The UAV class of the Geo-Fencing software has the following parameters:

```
public ref class Vehicle {
public:
  Vehicle() {};
  ~Vehicle() {}
  void lla2ned();
  void Update_time();
  /*void Ecef2Lla();
  void Ecef2Ned();*/

  //double pos_ecefX;
  //double pos_ecefY;
  //double pos_ecefZ;

  double Latitude = 0;
  double Longitud = 0;
  double Altitud = 0;

  double Speed = 0;
  double Heading = 0;

  String^ DiaUTC = "0";
  String^ MesUTC = "0";
  String^ AnnoUTC = "0";
  String^ HoraUTC = "0";

  DateTime^ current_time;
  double pos_nedX = 0;
  double pos_nedY = 0;
  double pos_nedZ = 0;

  double ref_lat;
  double ref_lon;
  double ref_alt = 0;

};
```

The GF comparator containing the two algorithms explained for circular and polygonal shapes and taking into account the GF times and the maximum altitude is:

```
bool GeoFence::UAVinside(Vehicle^ uav) {
  double X_UAV = uav->pos_nedX;
  double Y_UAV = uav->pos_nedX;
  double Z_UAV = uav->pos_nedX;
  DateTime^ Current_Time = uav->current_time;
```

```
int compareValueIni = Current_Time->CompareTo(time_ini);
int compareValueEnd = Current_Time->CompareTo(time_end);

if (compareValueIni<0 || compareValueEnd>0) {
  return false;
}
else {

  if (max_alt != 0 && Z_UAV > max_alt) {
    return false;
  }
  else {
    if (type == "poly") {
      int   i, j = npoints - 1;
      //bool  oddNodes = false;
      int croses = 0;

      for (i = 0; i < npoints; i++) {
        if ((Y_values[i] < Y_UAV && Y_values[j] >= Y_UAV
          || Y_values[j] < Y_UAV && Y_values[i] >= Y_UAV)
          && (X_values[i] <= X_UAV || X_values[j] <= X_UAV)) {
          //oddNodes ^= (X_values[i] + (Y_UAV - Y_values[i]) / (Y_values[j] - Y_values[i])*(X_values[j] - X_values[i]) < X_UAV);

          if ((X_values[i] + (Y_UAV - Y_values[i]) / (Y_values[j] - Y_values[i])*(X_values[j] - X_values[i]) < X_UAV)) {
            croses++;
          }
        }
        j = i;
      }
      if (croses % 2 == 1) { return true; }
      if (croses % 2 == 0) { return false; }

      printf("%d \n", croses);
      //return oddNodes;

    }

    if (type == "circ") {
      distance = sqrt(pow(X_UAV - X_cent, 2) + pow(Y_UAV - Y_cent, 2));

      if (distance > rad) {
        return false;     //uav is outside
      }
      if (distance < rad) {
        return true;      //uav is inside
      }

    }
  }
}

```

```
}
```

Scheduler Code - C++

The part of the code used to create the path plan for the boundary surveillance mission is:

```
UAV::UAV(GeoFence^ GF, String^ mission, double speed, double altitude, double offset, double separation) {

  current_speed = speed;
  current_altitude = altitude;
  mis = mission;
  offs = offset;
  // polygonal and boundary
  if (mission == "bound" && GF->type == "poly") {
    //firstly a NED local frame is calculated as he bottom left corner of a rectangle that includes all the points
    straight = true;
    LocalFrameLat = GF->X_values[0];
    LocalFrameLon = GF->Y_values[0];
    for (int i = 1; i < GF->npoints; i++) {
      if (GF->X_values[i] < LocalFrameLat) {
        LocalFrameLat = GF->X_values[i];
      }
      if (GF->Y_values[i] < LocalFrameLon) {
        LocalFrameLon = GF->Y_values[i];
      }
    }

    //Convert GF to NED coordinates
    array<double>^ point_NED = gcnew array<double>(3);
    GF_X_NED = gcnew array<double>(GF->npoints);
    GF_Y_NED = gcnew array<double>(GF->npoints);
    GF_Z_NED = gcnew array<double>(GF->npoints);
    for (int i = 0; i < GF->npoints; i++) {
      point_NED = lla2ned(GF->X_values[i], GF->Y_values[i], GF->max_alt, LocalFrameLat, LocalFrameLon, 0);
      GF_X_NED[i] = point_NED[0];
      GF_Y_NED[i] = point_NED[1];
      GF_Z_NED[i] = point_NED[2];
    }


    //calculate internal points of the boundary
    path_nedX = gcnew array<double>(GF->npoints);
    path_nedY = gcnew array<double>(GF->npoints);


    //first point
    double angF, angB, angint;
    double difX, difY, diag_offset;
    double X_offset, Y_offset;

```

```
    //first point
    angF = atan2(GF_X_NED[1] - GF_X_NED[0], GF_Y_NED[1] - GF_Y_NED[0]);
    angB = atan2(GF_X_NED[GF->npoints - 1] - GF_X_NED[0], GF_Y_NED[GF->npoints - 1] - GF_Y_NED[0]);

    if (angF >= 0) {
      angint = angB + (angF - angB) / 2;
    }
    if (angF <= 0) {
      if (angB >= 0) {
        angint = angB + (2 * PI - angB + angF) / 2;
      }
      if (angB <= 0) {
        angint = angF + (angB - angF) / 2;
      }
    }

    diag_offset = offset / abs((sin(angF - angint)));

    difX = diag_offset * sin(angint);
    difY = diag_offset * cos(angint);

    path_nedX[0] = GF_X_NED[0] + difX;
    path_nedY[0] = GF_Y_NED[0] + difY;

    for (int i = 1; i < (GF->npoints - 1); i++) {
      angF = atan2(GF_X_NED[i + 1] - GF_X_NED[i], GF_Y_NED[i + 1] - GF_Y_NED[i]);
      angB = atan2(GF_X_NED[i - 1] - GF_X_NED[i], GF_Y_NED[i - 1] - GF_Y_NED[i]);

      if (angF >= 0) {
        angint = angB + (angF - angB) / 2;
      }
      if (angF <= 0) {
        if (angB >= 0) {
          angint = angB + (2 * PI - angB + angF) / 2;
        }
        if (angB <= 0) {
          angint = angF + (angB - angF) / 2;
        }
      }

      diag_offset = offset / abs((sin(angF - angint)));

      difX = diag_offset * sin(angint);
      difY = diag_offset * cos(angint);

      path_nedX[i] = GF_X_NED[i] + difX;
      path_nedY[i] = GF_Y_NED[i] + difY;

    }

    // last point
    angF = atan2(GF_X_NED[0] - GF_X_NED[GF->npoints - 1], GF_Y_NED[0] - GF_Y_NED[GF->npoints - 1]);
    angB = atan2(GF_X_NED[GF->npoints - 2] - GF_X_NED[GF->npoints - 1], GF_Y_NED[GF->npoints - 2] - GF_Y_NED[GF->npoints - 1]);
```

```

    if (angF >= 0) {
      angint = angB + (angF - angB) / 2;
    }
    if (angF <= 0) {
      if (angB >= 0) {
        angint = angB + (2 * PI - angB + angF) / 2;
      }
      if (angB <= 0) {
        angint = angF + (angB - angF) / 2;
      }
    }

    diag_offset = offset / abs((sin(angF - angint)));

    difX = diag_offset * sin(angint);
    difY = diag_offset * cos(angint);

    path_nedX[GF->npoints - 1] = GF_X_NED[GF->npoints -
       1] + difX;
    path_nedY[GF->npoints - 1] = GF_Y_NED[GF->npoints -
       1] + difY;
  }

    // circular and boundary
  else if (mission == "bound"  && GF->type == "circ") {
    straight = false;
    LocalFrameLat = GF->X_cent;
    LocalFrameLon = GF->Y_cent;

    array<double>^ point_NED = gcnew array<double>(3);
    GF_X_NED = gcnew array<double>(1);
    GF_Y_NED = gcnew array<double>(1);
    GF_Z_NED = gcnew array<double>(1);
    path_nedX = gcnew array<double>(1);
    path_nedY = gcnew array<double>(1);

    point_NED = lla2ned(GF->X_cent, GF->Y_cent, GF->
       max_alt, LocalFrameLat, LocalFrameLon, 0);
    GF_X_NED[0] = point_NED[0];
    GF_Y_NED[0] = point_NED[1];
    GF_Z_NED[0] = point_NED[2];
    radius = GF->rad - offset;
    path_nedX[0] = GF_X_NED[0] - radius;
    path_nedY[0] = GF_Y_NED[0];
  }
```

The part of the code used to create the path plan for the interior surveillance mission is:

```
// polygonal and interior
else if (mission == "interior"  && GF->type == "poly") {
  //firstly a NED local frame is calculated as he bottom
     left corner of a rectangle that includes all the
     points
  straight = true;
  LocalFrameLat = GF->X_values[0];
  LocalFrameLon = GF->Y_values[0];
  for (int i = 1; i < GF->npoints; i++) {
```

```
    if (GF->X_values[i] < LocalFrameLat) {
      LocalFrameLat = GF->X_values[i];
    }
    if (GF->Y_values[i] < LocalFrameLon) {
      LocalFrameLon = GF->Y_values[i];
    }
  }

  //Convert GF to NED coordinates
  array<double>^ point_NED = gcnew array<double>(3);
  GF_X_NED = gcnew array<double>(GF->npoints);
  GF_Y_NED = gcnew array<double>(GF->npoints);
  GF_Z_NED = gcnew array<double>(GF->npoints);
  for (int i = 0; i < GF->npoints; i++) {
    point_NED = lla2ned(GF->X_values[i], GF->Y_values[i],
        GF->max_alt, LocalFrameLat, LocalFrameLon, 0);
    GF_X_NED[i] = point_NED[0];
    GF_Y_NED[i] = point_NED[1];
    GF_Z_NED[i] = point_NED[2];
  }


  //calculate internal points of the boundary
  array<double>^ int_nedX = gcnew array<double>(GF->
     npoints);
  array<double>^ int_nedY = gcnew array<double>(GF->
     npoints);


  //first point
  double angF, angB, angint;
  double difX, difY, diag_offset;
  double X_offset, Y_offset;

  //first point
  angF = atan2(GF_X_NED[1] - GF_X_NED[0], GF_Y_NED[1] -
     GF_Y_NED[0]);
  angB = atan2(GF_X_NED[GF->npoints - 1] - GF_X_NED[0],
     GF_Y_NED[GF->npoints - 1] - GF_Y_NED[0]);

  if (angF >= 0) {
    angint = angB + (angF - angB) / 2;
  }
  if (angF <= 0) {
    if (angB >= 0) {
      angint = angB + (2 * PI - angB + angF) / 2;
    }
    if (angB <= 0) {
      angint = angF + (angB - angF) / 2;
    }
  }

  diag_offset = offset / abs((sin(angF - angint)));

  difX = diag_offset * sin(angint);
  difY = diag_offset * cos(angint);

  int_nedX[0] = GF_X_NED[0] + difX;
  int_nedY[0] = GF_Y_NED[0] + difY;

```

```
  for (int i = 1; i < (GF->npoints - 1); i++) {
    angF = atan2(GF_X_NED[i + 1] - GF_X_NED[i], GF_Y_NED[i + 1] - GF_Y_NED[i]);
    angB = atan2(GF_X_NED[i - 1] - GF_X_NED[i], GF_Y_NED[i - 1] - GF_Y_NED[i]);

    if (angF >= 0) {
      angint = angB + (angF - angB) / 2;
    }
    if (angF <= 0) {
      if (angB >= 0) {
        angint = angB + (2 * PI - angB + angF) / 2;
      }
      if (angB <= 0) {
        angint = angF + (angB - angF) / 2;
      }
    }

    diag_offset = offset / abs((sin(angF - angint)));

    difX = diag_offset * sin(angint);
    difY = diag_offset * cos(angint);

    int_nedX[i] = GF_X_NED[i] + difX;
    int_nedY[i] = GF_Y_NED[i] + difY;

  }

  // last point
  angF = atan2(GF_X_NED[0] - GF_X_NED[GF->npoints - 1], GF_Y_NED[0] - GF_Y_NED[GF->npoints - 1]);
  angB = atan2(GF_X_NED[GF->npoints - 2] - GF_X_NED[GF->npoints - 1], GF_Y_NED[GF->npoints - 2] - GF_Y_NED[GF->npoints - 1]);

  if (angF >= 0) {
    angint = angB + (angF - angB) / 2;
  }
  if (angF <= 0) {
    if (angB >= 0) {
      angint = angB + (2 * PI - angB + angF) / 2;
    }
    if (angB <= 0) {
      angint = angF + (angB - angF) / 2;
    }
  }

  diag_offset = offset / abs((sin(angF - angint)));

  difX = diag_offset * sin(angint);
  difY = diag_offset * cos(angint);

  int_nedX[GF->npoints - 1] = GF_X_NED[GF->npoints - 1] + difX;
  int_nedY[GF->npoints - 1] = GF_Y_NED[GF->npoints - 1] + difY;

  // now we have interior polygon, calculate rest of points
  //obtain longest side
```

```
double top_distance, dist, heading1;
int pos;
top_distance = sqrt(pow((int_nedX[int_nedX->Length - 1] - int_nedX[0]), 2) + pow((int_nedY[int_nedX->Length - 1] - int_nedY[0]), 2));
pos = int_nedX->Length - 1;

for (int i = 0; i <= int_nedX->Length - 2; i++) {
  dist = sqrt(pow((int_nedX[i + 1] - int_nedX[i]), 2) + pow((int_nedY[i + 1] - int_nedY[i]), 2));
  if (dist > top_distance) {
    top_distance = dist;
    pos = i;
  }
}

//re - order points
array<double>^ int_nedX2 = gcnew array<double>(GF->npoints);
array<double>^ int_nedY2 = gcnew array<double>(GF->npoints);
int_nedX2[0] = int_nedX[pos];
int_nedY2[0] = int_nedY[pos];

for (int i = 1; i <= int_nedX->Length - 1; i++) {
  pos = pos + 1;
  if (pos == int_nedX->Length) {
    pos = 0;
  }
  int_nedX2[i] = int_nedX[pos];
  int_nedY2[i] = int_nedY[pos];
}
heading1 = atan2(int_nedX2[1] - int_nedX2[0], int_nedY2[1] - int_nedY2[0]);

//rotate points to straight with the longest point as base
if (int_nedX2[0] > int_nedX2[2] || int_nedX2[1] > int_nedX2[2]) {
  heading1 = heading1 + PI;
}
double Rot11, Rot12, Rot21, Rot22;
Rot11 = cos(heading1);
Rot12 = sin(heading1);
Rot21 = -sin(heading1);
Rot22 = cos(heading1);

double Rotr11, Rotr12, Rotr21, Rotr22;
Rotr11 = cos(-heading1);
Rotr12 = sin(-heading1);
Rotr21 = -sin(-heading1);
Rotr22 = cos(-heading1);

array<double>^ int_nedX3 = gcnew array<double>(GF->npoints);
array<double>^ int_nedY3 = gcnew array<double>(GF->npoints);
for (int i = 0; i <= int_nedX2->Length - 1; i++) {
  int_nedX3[i] = int_nedX2[i] * Rot11 + int_nedY2[i] * Rot21;
```

```
    int_nedY3[i] = int_nedX2[i] * Rot12 + int_nedY2[i] *
       Rot22;
  }

  //calculate bounding box
  double minX, minY, maxX, maxY;
  minX = int_nedX3[0];
  maxX = int_nedX3[0];
  minY = int_nedY3[0];
  maxY = int_nedY3[0];
  for (int i = 1; i < int_nedX3->Length - 1; i++) {
    if (int_nedX3[i] < minX) {
      minX = int_nedX3[i];
    }
    else if (int_nedX3[i] < maxX) {
      maxX = int_nedX3[i];
    }
    else if (int_nedY3[i] < minY) {
      minY = int_nedY3[i];
    }
    else if (int_nedY3[i] < maxY) {
      maxY = int_nedY3[i];
    }

  }

  array<double>^ boundingX = gcnew array<double>{ minX,
     minX, maxX, maxX };
  array<double>^ boundingY = gcnew array<double>{ minY,
     maxY, maxY, minY };

  //number of lines
  array<double>^ d = gcnew array<double>(int_nedX->Length
     );
  array<double>^ v1 = gcnew array<double>{int_nedX2[0],
     int_nedY2[0]};
  array<double>^ v2 = gcnew array<double>{int_nedX2[1],
     int_nedY2[1]};
  array<double>^ pt = gcnew array<double>(2);
  for (int i = 0; i <= int_nedX2->Length - 1; i++) {
    pt[0] = int_nedX2[i];
    pt[1] = int_nedY2[i];
    d[i] = point_to_line(pt, v1, v2);
  }
  double max_dist;
  int index_max = 0;
  max_dist = d[0];
  for (int i = 0; i <= int_nedX->Length - 1; i++) {
    if (d[i] > max_dist) {
      max_dist = d[i];
      index_max = i;
    }
  }

  double n_of_lines;
  n_of_lines = floor(max_dist / separation);


  array<double>^ exterior_pointsX = gcnew array<double>(2
     * n_of_lines);
```

```
array<double>^ exterior_pointsY = gcnew array<double>(2 * n_of_lines);

exterior_pointsX[0] = boundingX[0];
exterior_pointsX[1] = boundingX[1];
exterior_pointsY[0] = boundingY[0];
exterior_pointsY[1] = boundingY[1];
int count = 2;
for (int i = 2; i <= n_of_lines; i = i + 2) {
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i - 1)*separation;
    exterior_pointsY[count] = boundingY[1];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i - 1)*separation;
    exterior_pointsY[count] = boundingY[0];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i)*separation;
    exterior_pointsY[count] = boundingY[0];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i)*separation;
    exterior_pointsY[count] = boundingY[1];
    count++;
  }
}


// crossing points with polygon
int lines = 0;
array<linea^>^ line_left = gcnew array<linea^>(index_max - 1);

for (int i = 2; i <= index_max; i++) {
  // lines at left of max point

  line_left[lines] = gcnew linea;
  line_left[lines]->pointsX[0] = int_nedX3[i - 1];
  line_left[lines]->pointsX[1] = int_nedX3[i];
  line_left[lines]->pointsY[0] = int_nedY3[i - 1];
  line_left[lines]->pointsY[1] = int_nedY3[i];
  line_left[lines]->angle = atan2(line_left[lines]->pointsX[1] - line_left[lines]->pointsX[0], line_left[lines]->pointsY[1] - line_left[lines]->pointsY[0]);


  lines = lines + 1;
}
lines = 0;


array<linea^>^ line_right = gcnew array<linea^>(
```

```
    int_nedX->Length - index_max);

  for (int i = int_nedX->Length - 1; i >= index_max; i--) {
    // lines at left of max point
    if ((i + 1) >= int_nedX->Length) {

      line_right[lines] = gcnew linea;
      line_right[lines]->pointsX[0] = int_nedX3[0];
      line_right[lines]->pointsX[1] = int_nedX3[i];
      line_right[lines]->pointsY[0] = int_nedY3[0];
      line_right[lines]->pointsY[1] = int_nedY3[i];
      line_right[lines]->angle = atan2(line_right[lines]->pointsX[1] - line_right[lines]->pointsX[0], line_right[lines]->pointsY[1] - line_right[lines]->pointsY[0]);

    }
    else {
      line_right[lines] = gcnew linea;
      line_right[lines]->pointsX[0] = int_nedX3[i + 1];
      line_right[lines]->pointsX[1] = int_nedX3[i];
      line_right[lines]->pointsY[0] = int_nedY3[i + 1];
      line_right[lines]->pointsY[1] = int_nedY3[i];
      line_right[lines]->angle = atan2(line_right[lines]->pointsX[1] - line_right[lines]->pointsX[0], line_right[lines]->pointsY[1] - line_right[lines]->pointsY[0]);


    }
    lines = lines + 1;


  }

  int points = 1;
  array<double>^ pathX = gcnew array<double>(2 * n_of_lines + 1);
  array<double>^ pathY = gcnew array<double>(2 * n_of_lines + 1);
  double x_value, y_value;
  int line = 0;

  for (int j = 0; j < exterior_pointsX->Length; j = j + 4) {
    // 1st point left
    if (points == 1 && j < exterior_pointsX->Length) {
      x_value = exterior_pointsX[j];

      for (int i = 0; i < line_left->Length; i++) {
        if ((line_left[i]->pointsX[0] <= x_value + 0.0001) && (x_value - 0.0001 <= line_left[i]->pointsX[1])) {
          line = i;
        }
      }
      pathX[j + points - 1] = x_value;
      if (line_left[line]->angle <= PI) {
        pathY[j + points - 1] = line_left[line]->pointsY
```

```
              [0] + (x_value - line_left[line]->pointsX[0])
              / tan(line_left[line]->angle);
        }
        else if (line_left[line]->angle > PI) {
          pathY[j + points - 1] = line_left[line]->pointsY
              [0] - (x_value - line_left[line]->pointsX[0])
              / tan(PI - line_left[line]->angle);
        }
        points = points + 1;
      }
      if (points == 2 && j + 1 < exterior_pointsX->Length)
         {
        x_value = exterior_pointsX[j + 1];

        for (int i = 0; i < line_right->Length; i++) {
          if ((line_right[i]->pointsX[0] <= x_value +
              0.0001) && (x_value - 0.0001 <= line_right[i
              ]->pointsX[1])) {
            line = i;
          }
        }
        pathX[j + points - 1] = x_value;
        if (line_right[line]->angle <= PI) {
          pathY[j + points - 1] = line_right[line]->pointsY
              [0] + (x_value - line_right[line]->pointsX[0])
               / tan(line_right[line]->angle);
        }
        else if (line_right[line]->angle > PI) {
          pathY[j + points - 1] = line_right[line]->pointsY
              [0] - (x_value - line_right[line]->pointsX[0])
               / tan(PI - line_right[line]->angle);
        }
        points = points + 1;
      }
      if (points == 3 && j + 2 < exterior_pointsX->Length)
         {
        x_value = exterior_pointsX[j + 2];

        for (int i = 0; i < line_right->Length; i++) {
          if ((line_right[i]->pointsX[0] <= x_value +
              0.0001) && (x_value - 0.0001 <= line_right[i
              ]->pointsX[1])) {
            line = i;
          }
        }
        pathX[j + points - 1] = x_value;
        if (line_right[line]->angle <= PI) {
          pathY[j + points - 1] = line_right[line]->pointsY
              [0] + (x_value - line_right[line]->pointsX[0])
               / tan(line_right[line]->angle);
        }
        else if (line_right[line]->angle > PI) {
          pathY[j + points - 1] = line_right[line]->pointsY
              [0] - (x_value - line_right[line]->pointsX[0])
               / tan(PI - line_right[line]->angle);
        }
        points = points + 1;
      }
      if (points == 4 && j + 3 < exterior_pointsX->Length)
```

```
        {
        x_value = exterior_pointsX[j + 3];

        for (int i = 0; i < line_left->Length; i++) {
          if ((line_left[i]->pointsX[0] <= x_value +
             0.0001) && (x_value - 0.0001 <= line_left[i]->
             pointsX[1])) {
            line = i;
          }
        }
        pathX[j + points - 1] = x_value;
        if (line_left[line]->angle <= PI) {
          pathY[j + points - 1] = line_left[line]->pointsY
             [0] + (x_value - line_left[line]->pointsX[0])
             / tan(line_left[line]->angle);
        }
        else if (line_left[line]->angle > PI) {
          pathY[j + points - 1] = line_left[line]->pointsY
             [0] - (x_value - line_left[line]->pointsX[0])
             / tan(PI - line_left[line]->angle);
        }
        points = 1;
      }
    }
    pathX[pathX->Length - 1] = int_nedX3[index_max];
    pathY[pathY->Length - 1] = int_nedY3[index_max];

    // reconversion to original ned
    path_nedX = gcnew array<double>(pathX->Length);
    path_nedY = gcnew array<double>(pathX->Length);
    for (int i = 0; i < pathX->Length; i++) {
      path_nedX[i] = pathX[i] * Rotr11 + pathY[i] * Rotr21;
      path_nedY[i] = pathX[i] * Rotr12 + pathY[i] * Rotr22;
    }


}
        // circular and interior
else if (mission == "interior" && GF->type == "circ") {
    straight = true;
    LocalFrameLat = GF->X_cent;
    LocalFrameLon = GF->Y_cent;

    array<double>^ point_NED = gcnew array<double>(3);
    GF_X_NED = gcnew array<double>(1);
    GF_Y_NED = gcnew array<double>(1);
    GF_Z_NED = gcnew array<double>(1);
    point_NED = lla2ned(GF->X_cent, GF->Y_cent, GF->max_alt
        , LocalFrameLat, LocalFrameLon, 0);
    GF_X_NED[0] = point_NED[0];
    GF_Y_NED[0] = point_NED[1];
    GF_Z_NED[0] = point_NED[2];
    radius = GF->rad - offset;


    //interior surveillance
      // calculate bounding box
    double minX, minY, maxX, maxY;
    minX = GF_X_NED[0] - radius;
    maxX = GF_X_NED[0] + radius;
```

```
minY = GF_Y_NED[0] - radius;
maxY = GF_Y_NED[0] + radius;
array<double>^ boundingX = gcnew array<double>{ minX, minX, maxX, maxX };
array<double>^ boundingY = gcnew array<double>{ minY, maxY, maxY, minY };

//num of lines
int n_of_lines;
n_of_lines = floor(2 * radius / separation);

array<double>^ exterior_pointsX = gcnew array<double>(2 * n_of_lines);
array<double>^ exterior_pointsY = gcnew array<double>(2 * n_of_lines);

exterior_pointsX[0] = boundingX[0];
exterior_pointsX[1] = boundingX[1];
exterior_pointsY[0] = boundingY[0];
exterior_pointsY[1] = boundingY[1];
int count = 2;
for (int i = 2; i <= n_of_lines; i = i + 2) {
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i - 1)*separation;
    exterior_pointsY[count] = boundingY[1];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i - 1)*separation;
    exterior_pointsY[count] = boundingY[0];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i)*separation;
    exterior_pointsY[count] = boundingY[0];
    count++;
  }
  if (count < exterior_pointsX->Length) {
    exterior_pointsX[count] = boundingX[0] + (i)*separation;
    exterior_pointsY[count] = boundingY[1];
    count++;
  }
}

int points = 1;
path_nedX = gcnew array<double>(2 * n_of_lines);
path_nedY = gcnew array<double>(2 * n_of_lines);
double x_value, y_value;
int line = 0;

path_nedX[0] = GF_X_NED[0] - radius;
path_nedY[0] = GF_Y_NED[0];
for (int j = 2; j < exterior_pointsX->Length; j = j + 4) {
  // 1st point left
  if (points == 1 && j < exterior_pointsX->Length) {
```

```
        x_value = exterior_pointsX[j];
        path_nedX[j + points - 2] = x_value;
        path_nedY[j + points - 2] = GF_Y_NED[0] + sqrt(pow(
           radius, 2) - pow(abs(x_value), 2));
        points = points + 1;
      }
      if (points == 2 && j + 1 < exterior_pointsX->Length)
         {
        x_value = exterior_pointsX[j + 1];
        path_nedX[j + points - 2] = x_value;
        path_nedY[j + points - 2] = GF_Y_NED[0] - sqrt(pow(
           radius, 2) - pow(abs(x_value), 2));
        points = points + 1;
      }
      if (points == 3 && j + 2 < exterior_pointsX->Length)
         {
        x_value = exterior_pointsX[j + 2];
        path_nedX[j + points - 2] = x_value;
        path_nedY[j + points - 2] = GF_Y_NED[0] - sqrt(pow(
           radius, 2) - pow(abs(x_value), 2));
        points = points + 1;
      }
      if (points == 4 && j + 3 < exterior_pointsX->Length)
         {
        x_value = exterior_pointsX[j + 3];
        path_nedX[j + points - 2] = x_value;
        path_nedY[j + points - 2] = GF_Y_NED[0] + sqrt(pow(
           radius, 2) - pow(abs(x_value), 2));
        points = 1;
      }
    }
    path_nedX[path_nedX->Length - 1] = GF_X_NED[0] + radius
       ;
    path_nedY[path_nedX->Length - 1] = GF_Y_NED[0];

}
```

The part of the code containing the path following algorithm is:

```
void UAV::TrajGen() {
  time_total = time_total + timestep;

  double Ru, theta, theta_u, delt_theta, R, s1, s2, u_max
     , delt_heading, u, heading_rate, d;
  if (straight == true) {

    //chasing carrot algorithm
    Ru = sqrt(pow((w1[0] - pos_init_nedX), 2) + pow((w1
       [1] - pos_init_nedY), 2));
    theta = atan2(w2[1] - w1[1], w2[0] - w1[0]);
    theta_u = atan2(pos_init_nedY - w1[1], pos_init_nedX
       - w1[0]);
    delt_theta = theta - theta_u;

    R = sqrt(pow(Ru, 2) - pow((Ru * sin(delt_theta)), 2))
       ;
    s1 = w1[0] + (R + delta)*cos(theta);
    s2 = w1[1] + (R + delta)*sin(theta);
    desired_heading = atan2(s2 - pos_init_nedY, s1 -
       pos_init_nedX);
```

```
  }

  if (straight == false) {
    d = sqrt(pow((pos_init_nedX), 2) + pow((pos_init_nedY
        ), 2)) - radius;

    theta = atan2(pos_init_nedY, pos_init_nedX);

    s1 = GF_X_NED[0] + radius * cos(theta + lambda);
    s2 = GF_Y_NED[0] + radius * sin(theta + lambda);
    desired_heading = atan2(s2 - pos_init_nedY, s1 -
        pos_init_nedX);


  }
  u_max = abs(pow(current_speed, 2) / Rmin);
  delt_heading = desired_heading - current_heading;
  if (abs(delt_heading) > PI) {
    delt_heading = -Math::Sign(delt_heading)*(2 * PI -
        abs(delt_heading));
  }
  if (abs(kappa*(delt_heading)*current_speed) < u_max) {
    u = (kappa*(delt_heading)*current_speed);
  }
  else if (abs(kappa*(delt_heading)*current_speed) >=
     u_max) {
    u = Math::Sign(kappa*(delt_heading)*current_speed) *
        u_max;
  }

  heading_rate = u / current_speed;
  current_heading = current_heading + heading_rate *
     timestep;
  vel_final_nedX = current_speed * cos(current_heading);
  vel_final_nedY = current_speed * sin(current_heading);
  vel_final_nedZ = 0;

  pos_final_nedX = pos_init_nedX + vel_final_nedX *
     timestep;
  pos_final_nedY = pos_init_nedY + vel_final_nedY *
     timestep;
  pos_final_nedZ = current_altitude;
  if (straight == true) {
    dist_to_change = sqrt(pow((pos_final_nedX - w2[0]),
        2) + pow((pos_final_nedY - w2[1]), 2));

  }
  if (current_heading >= PI) {
    current_heading = current_heading - 2 * PI;
  }
  if (current_heading <= -PI) {
    current_heading = current_heading + 2 * PI;
  }
}
```

Appendix B - Performance Analysis - Matlab Files

For each test done in in the Speed-Offset analysis, a block of code like the one show below is used to provide result in form of tables and figures:

```
% import GF data
% addpath('Z:\ProfileData\s278765\Desktop')
addpath('C:\Users\Jorge\Desktop\tesis\performance analysis\Scenarios');
GF = textread('cranfield.txt','%s','delimiter',';');

GeoFence.ID = GF(2);
GeoFence.shape = GF(3);
GeoFence.NP = str2double(GF(4));
j=1;
for i = 5:2:length(GF)-4
    GeoFence.X(j) = str2double(GF(i));
    GeoFence.Y(j) = str2double(GF(i+1));
    j=j+1;
end

%% speed 5 offset 1m 1st test
[~, ~, raw0_0] = xlsread('C:\Users\Jorge\Desktop\tesis\performance analysis\offset-speed_analysis\boundary_offset_analysis.xlsx','SP5_Off1','E2:F509');
[~, ~, raw0_1] = xlsread('C:\Users\Jorge\Desktop\tesis\performance analysis\offset-speed_analysis\boundary_offset_analysis.xlsx','SP5_Off1','I2:J509');
% [~, ~, raw0_0] = xlsread('Z:\ProfileData\s278765\Desktop\boundary_offset_analysis.xlsx','SP5_Off1','E2:F509');
% [~, ~, raw0_1] = xlsread('Z:\ProfileData\s278765\Desktop\boundary_offset_analysis.xlsx','SP5_Off1','I2:J509');


sp5off1.mot_comand = reshape([raw0_0{:}],size(raw0_0));
sp5off1.real_pos   = reshape([raw0_1{:}],size(raw0_1));
clearvars raw0_0 raw0_1;

% test offset 1m
sp5off1.time_out_path = 0;
sp5off1.time_out_GPS = 0;
bool1 = true;
bool2 = true;

figure(1)
plot ([GeoFence.Y GeoFence.Y(1)], [GeoFence.X GeoFence.X(1)],'color',[0,0,1])
hold on
for i = 1:length(sp5off1.mot_comand(:,1))
    bool1 = UAV_inside(sp5off1.mot_comand(i,1),sp5off1.mot_comand(i,2),0,GeoFence);
    bool2 = UAV_inside(sp5off1.real_pos(i,1),sp5off1.real_pos(i,2),0,GeoFence);
    if bool1 == false
        sp5off1.time_out_path = sp5off1.time_out_path + 1;
        h1 = plot(sp5off1.mot_comand(i,2),sp5off1.
```

```
                mot_comand(i,1),'x','color',[0,1,0]);
        end

        if bool2 == false
            sp5off1.time_out_GPS  = sp5off1.time_out_GPS  + 1;
            h2 = plot(sp5off1.real_pos(i,2),sp5off1.real_pos(i,1),'x','color',[1,0,0]);
        end
        bool1 = true;
        bool2 = true;
end

sp5off1.per_out_path = (sp5off1.time_out_path / length(sp5off1.mot_comand(:,1))) * 100;
sp5off1.per_out_GPS  = (sp5off1.time_out_GPS / length(sp5off1.real_pos(:,1))) * 100;


plot(sp5off1.mot_comand(:,2),sp5off1.mot_comand(:,1),'color',[0,1,0])
plot(sp5off1.real_pos(:,2),sp5off1.real_pos(:,1),'color',[1,0,0])
title('Points Outside the Geo-Fence - Speed 5 m/s - Offset 1 m')
xlabel('Lon [deg]')
ylabel('Lat [deg]')
if exist('h1','var') == 1
    legend([h1], {'Points outside - Path Following'}, 'Location', 'Southeast')
end
if exist('h2','var') == 1
    legend([h2], {'Points outside - GPS accuracy' }, 'Location', 'Southeast')
end
if exist('h2','var') == 1 && exist('h1','var') == 1
    legend([h1,h2], {'Points outside - Path Following','Points outside - GPS accuracy' }, 'Location', 'Southeast')
end
```

For each of the tests done in the Multi-path analysis, the code used to calculate errors and plots is given by:

```
% Date: 4th July 2018
% Authors: Duran Zafrilla, Jorge (Cranfield University) & Verdeguer Moreno, Ricardo (Spirent Communications)
% Description: Nominal Test with Sim3D (4 multipath) and the GSS9000 @ 100 Hz + Ublox M8T receiver
% ------
%
% Conditions:
%
% - Scenario Type: S
% - PDOP: <4
% - Altitude: 120 m
% - Separation: 15 m
% - Mission: Interior Surveillance
% - Geo-fence offset: 5 m
```

```
% - Speed: 5 m/s
%
% Scenario Start Date & Time: 4th July 2017 05:00:00 (
    Default SimGEN orbits)

clc;clear;

%%import data
[~, ~, raw0_0] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','B2:B572');
[~, ~, raw0_1] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','E2:E572');
[~, ~, raw0_2] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','H2:H572');
raw = [raw0_0,raw0_1,raw0_2];
pos_ideal_nav3d = reshape([raw{:}],size(raw));
clearvars raw raw0_0 raw0_1 raw0_2;

[~, ~, raw0_0] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','K2:K572');
[~, ~, raw0_1] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','M2:M572');
[~, ~, raw0_2] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\Multipath - Nav3D\Nominal Test\
    nominal_motion.xlsx','Hoja4','P2:P572');
raw = [raw0_0,raw0_1,raw0_2];
pos_real_nav3d = reshape([raw{:}],size(raw));
clearvars raw raw0_0 raw0_1 raw0_2;

[~, ~, raw] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\No Multipath\Nominal Test\
    nominal_motion2.xlsx','Hoja2','E2:F569');
pos_ideal_sinm = reshape([raw{:}],size(raw));
clearvars raw;

[~, ~, raw] = xlsread('C:\Users\Jorge\Desktop\tesis\
    performance analysis\No Multipath\Nominal Test\
    nominal_motion2.xlsx','Hoja2','I2:J569');
R = cellfun(@(x) (~isnumeric(x) && ~islogical(x)) ||
    isnan(x),raw); % Find non-numeric cells
raw(R) = {0.0}; % Replace non-numeric cells
pos_real_sinm = reshape([raw{:}],size(raw));
clearvars raw R;

%% distance between points
R = 6371*10^3; % metres


error_distance_nav3d = zeros(length(pos_real_nav3d(:,1))
    ,1);
error_distance_sinm = zeros(length(pos_real_sinm(:,1)),1)
    ;
for i = 1:length(pos_real_nav3d(:,1))
    lat1 = pos_ideal_nav3d(i,1)*pi/180;
    lat2 = pos_real_nav3d(i,1)*pi/180;
```

```
    lon1 = pos_ideal_nav3d(i,2)*pi/180;
    lon2 = pos_real_nav3d(i,2)*pi/180;
    delt_lat = (lat2-lat1);
    delt_lon = (lon2-lon1);

    a = sin(delt_lat/2) * sin(delt_lat/2) + cos(lat1) * cos(lat2) * sin(delt_lon/2) * sin(delt_lon/2);
    c = 2 * atan2(sqrt(a), sqrt(1-a));

    error_distance_nav3d(i) = R * c;
%     [error_distance_nav3d(i) , az] = distance (pos_ideal_nav3d(i,1),pos_ideal_nav3d(i,2),pos_real_nav3d(i,1),pos_real_nav3d(i,2));
end
for i = 1:length(pos_real_sinm(:,1))
    lat1 = pos_ideal_sinm(i,1)*pi/180;
    lat2 = pos_real_sinm(i,1)*pi/180;
    lon1 = pos_ideal_sinm(i,2)*pi/180;
    lon2 = pos_real_sinm(i,2)*pi/180;
    delt_lat = (lat2-lat1);
    delt_lon = (lon2-lon1);

    a = sin(delt_lat/2) * sin(delt_lat/2) + cos(lat1) * cos(lat2) * sin(delt_lon/2) * sin(delt_lon/2);
    c = 2 * atan2(sqrt(a), sqrt(1-a));

    error_distance_sinm(i) = R * c;

%     [error_distance_sinm(i) , az] = distance (pos_ideal_sinm(i,1),pos_ideal_sinm(i,2),pos_real_sinm(i,1),pos_real_sinm(i,2));
end



%% process data
f=figure(1);
subplot(6,1,1:5)
plot(pos_ideal_nav3d(:,2),pos_ideal_nav3d(:,1),'color',[1 0 0])
hold on
plot(pos_real_nav3d(:,2),pos_real_nav3d(:,1),'color',[0 1 0])
hold on
ax = f.CurrentAxes;
title('Nominal Case')
plot(pos_real_sinm(:,2),pos_real_sinm(:,1),'color',[0 0 1])
title('Nominal test - multipath influence')
legend('Theorical Trajectory','Trajectory with Multipath','Trajectory without Multipath','Location','SouthOutside')
xlabel('Lon [deg]')
ylabel('Lat [deg]')
grid on
subplot(6,1,6)
plot(1:1:length(error_distance_nav3d),error_distance_nav3d,'color',[0 1 0])
hold on
plot(1:1:length(error_distance_sinm),error_distance_sinm,
```

```
    'color',[0 0 1])
xlabel('Time [s]')
ylabel('Pos. Error [m]')
grid on
% %% dynamic plot

figure(3)
plot3(pos_ideal_nav3d(:,2),pos_ideal_nav3d(:,1),
    error_distance_nav3d,'color',[0 1 0])
hold on
plot(pos_ideal_nav3d(:,2),pos_ideal_nav3d(:,1),'color',[1
    0 0])
plot3(pos_ideal_sinm(:,2),pos_ideal_sinm(:,1),
    error_distance_sinm,'color',[0 0 1])
```

Appendix C - Performance Analysis - xlsx and csv Files

As explained during the development of this document, a .csv file was obtained from the GNSS simulator containing the motion command and another .csv file is stored from U-Center containing the positioning from the NMEA sentences of the receiver. In order to process this files, the relevant data for this project is stored in the same .xlsx file which is used by the Matlab files of the previous Appendix to calculate results for each of the tests done. Therefore, for each of the tests of both analysis performed the Excel files used are shown in Figures 7.1, 7.2 and 7.3.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Index	UTC	Lat	Lon	Alt (HAE)	Alt (MSL)	X	Y	Z	SVs Used	SVs Tracked	PDOP
2	109	07/04/2017 05:00:18	52.0810155	-0.61413717						0	29	4.2
3	110	07/04/2017 05:00:19	52.081091	-0.61405733						0	29	4.7
4	111	07/04/2017 05:00:20	52.0813358	-0.61442733						0	29	4.7
5	112	07/04/2017 05:00:21	52.0814092	-0.61435117						0	29	4.7
6	113	07/04/2017 05:00:22	52.081483	-0.61427233							58	
7	114	07/04/2017 05:00:23									29	
8	115	07/04/2017 05:00:24	52.081487	-0.614014							29	
9	116	07/04/2017 05:00:25	52.0814407	-0.61388233							58	
10	117	07/04/2017 05:00:26	52.0813958	-0.61374867	41.5	-4.8	3927619.84	-42074.018	5008406.91	0	29	4.7
11	118	07/04/2017 05:00:27	52.0813477	-0.61362167							29	
12	119	07/04/2017 05:00:28	52.0812982	-0.61349867						0	29	4.7
13	120	07/04/2017 05:00:29	52.0812492	-0.61337533							29	
14	121	07/04/2017 05:00:30	52.0812028	-0.61325183							29	
15	122	07/04/2017 05:00:31	52.0811543	-0.61312967						0	29	2.7
16	123	07/04/2017 05:00:32	52.0810958	-0.61301617						0	29	2.7
17	124	07/04/2017 05:00:33	52.0810475	-0.61289383							29	
18	125	07/04/2017 05:00:34	52.0810003	-0.61277033							29	
19	126	07/04/2017 05:00:35	52.0809537	-0.61264633						0	29	2.7
20	127	07/04/2017 05:00:36	52.0809068	-0.61252217						7	16	
21	128	07/04/2017 05:00:37	52.0808605	-0.61239783						7	16	
22	129	07/04/2017 05:00:38	52.0808138	-0.6122735	41.2	-5.1	3927671.82	-41973.439	5008366.88	0	32	2.7
23	130	07/04/2017 05:00:39	52.0807673	-0.61214917						0	32	2.7
24	131	07/04/2017 05:00:40	52.0807208	-0.61202467							32	
25	132	07/04/2017 05:00:41	52.0806742	-0.61190033						0	32	2.7
26	133	07/04/2017 05:00:42	52.0806315	-0.61178833						0	32	2.7
27	134	07/04/2017 05:00:43	52.0805867	-0.61166867						0	32	2.7

Figure 7.1: .csv file stored from U-Center with GNSS positioning data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	2	4	1000	1	GDOP	1183179600													
2	Time_ms	Pos_X	Pos_Y	Pos_Z	Vel_X	Vel_Y	Vel_Z	Acc_X	Acc_Y	Acc_Z	Jerk_X	Jerk_Y	Jerk_Z	Lat	Long	Height	Heading	Elevation	Bank
3	0	3927879.21	-42331.241	5008213.78	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894078	-0.0107767	50.2367	0.57748003	0	0
4	1000	3927866.11	-42320.176	5008224.09	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894341	-0.0107739	50.2421	0.57748014	1.58E-07	0
5	2000	3927859.91	-42314.935	5008228.97	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894466	-0.0107726	50.2447	0.57748019	2.32E-07	0
6	3000	3927853.38	-42309.423	5008234.1	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894597	-0.0107712	50.2475	0.57748019	2.26E-07	0
7	4000	3927846.85	-42303.905	5008239.24	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894728	-0.0107698	50.2502	0.57748018	2.23E-07	0
8	5000	3927840.32	-42298.39	5008244.37	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90894859	-0.0107685	50.253	0.57748018	2.18E-07	0
9	6000	3927833.8	-42292.876	5008249.51	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.9089499	-0.0107671	50.2558	0.57748018	2.13E-07	0
10	7000	3927827.27	-42287.361	5008254.64	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895121	-0.0107657	50.2587	0.57748017	2.08E-07	0
11	8000	3927820.75	-42281.857	5008259.77	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895252	-0.0107643	50.2615	0.57748017	2.01E-07	0
12	9000	3927814.23	-42276.351	5008264.9	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895382	-0.0107629	50.2643	0.57748016	1.94E-07	0
13	10000	3927807.71	-42270.836	5008270.03	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895513	-0.0107615	50.2672	0.57748016	1.89E-07	0
14	11000	3927801.18	-42265.321	5008275.17	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895644	-0.0107601	50.2701	0.57748016	1.84E-07	0
15	12000	3927794.65	-42259.806	5008280.3	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895775	-0.0107588	50.273	0.57748015	1.79E-07	0
16	13000	3927788.12	-42254.294	5008285.44	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90895906	-0.0107574	50.2759	0.57748015	1.74E-07	0
17	14000	3927781.6	-42248.78	5008290.57	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896037	-0.010756	50.2788	0.57748015	1.69E-07	0
18	15000	3927775.08	-42243.279	5008295.69	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896168	-0.0107546	50.2817	0.57748014	1.61E-07	0
19	16000	3927768.89	-42238.049	5008300.56	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896292	-0.0107533	50.2845	0.57748019	2.32E-07	0
20	17000	3927762.37	-42232.537	5008305.7	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896423	-0.0107519	50.2875	0.57748019	2.26E-07	0
21	18000	3927755.84	-42227.022	5008310.83	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896554	-0.0107505	50.2905	0.57748018	2.22E-07	0
22	19000	3927749.32	-42221.515	5008315.96	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896685	-0.0107491	50.2935	0.57748018	2.15E-07	0
23	20000	3927742.8	-42216.003	5008321.09	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896816	-0.0107477	50.2965	0.57748018	2.09E-07	0
24	21000	3927736.27	-42210.489	5008326.23	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90896947	-0.0107464	50.2996	0.57748017	2.05E-07	0
25	22000	3927729.74	-42204.974	5008331.36	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897078	-0.010745	50.3026	0.57748017	0.0000002	0
26	23000	3927723.21	-42199.456	5008336.5	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897209	-0.0107436	50.3057	0.57748017	1.96E-07	0
27	24000	3927716.69	-42193.95	5008341.63	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.9089734	-0.0107422	50.3088	0.57748016	1.89E-07	0
28	25000	3927710.16	-42188.432	5008346.77	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897471	-0.0107408	50.3119	0.57748016	1.85E-07	0
29	26000	3927703.64	-42182.92	5008351.9	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897602	-0.0107394	50.315	0.57748015	1.79E-07	0
30	27000	3927697.11	-42177.408	5008357.03	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897733	-0.010738	50.3181	0.57748015	1.74E-07	0
31	28000	3927690.58	-42171.894	5008362.17	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897864	-0.0107367	50.3212	0.57748015	1.69E-07	0
32	29000	3927684.05	-42166.379	5008367.3	-3.2739	2.7657	2.5754	0	0	0	0	0	0	0.90897995	-0.0107353	50.3244	0.57748014	1.65E-07	0

Figure 7.2: .csv file stored from GNSS simulator with motion data

	A	B	C	D	E	F	G	H	I	J
1	Motion Command from Simulator							Positioning from U-Center		
2	Time_ms	Lat	Long	Height				UTC	Lat	Lon
3	88000	0.90899187	-0.01071712	50.3573	52.0813976	-0.61404567		07/04/2017 05:01:11	52.0814362	-0.61396867
4	89000	0.90899034	-0.01071283	50.3572	52.0813103	-0.61379997		07/04/2017 05:01:12	52.0813385	-0.613733
5	90000	0.90898875	-0.01070847	50.3571	52.0812193	-0.61355032		07/04/2017 05:01:13	52.0812472	-0.6134875
6	91000	0.90898715	-0.01070413	50.357	52.0811273	-0.61330134		07/04/2017 05:01:14	52.0811537	-0.6132395
7	92000	0.90898554	-0.01069979	50.357	52.081035	-0.61305287		07/04/2017 05:01:15	52.0810598	-0.612992
8	93000	0.90898392	-0.01069546	50.357	52.0809425	-0.61280458		07/04/2017 05:01:16	52.0809657	-0.6127455
9	94000	0.90898235	-0.01069123	50.3571	52.0808523	-0.61256262		07/04/2017 05:01:17	52.0808732	-0.61250033
10	95000	0.90898073	-0.0106869	50.3572	52.0807596	-0.61231422		07/04/2017 05:01:18	52.0807798	-0.61225433
11	96000	0.90897911	-0.01068256	50.3574	52.0806668	-0.61206572		07/04/2017 05:01:19	52.0806858	-0.61200617
12	97000	0.90897749	-0.01067822	50.3576	52.0805738	-0.61181673		07/04/2017 05:01:20	52.0805815	-0.61176683
13	98000	0.90897587	-0.01067388	50.3579	52.080481	-0.61156824		07/04/2017 05:01:21	52.0804823	-0.61152367
14	99000	0.90897425	-0.01066954	50.3583	52.0803883	-0.61131988		07/04/2017 05:01:22	52.080386	-0.61127833
15	100000	0.90897263	-0.01066521	50.3588	52.0802954	-0.61107127		07/04/2017 05:01:23	52.0802912	-0.61103133
16	101000	0.90897101	-0.01066086	50.3593	52.0802024	-0.61082217		07/04/2017 05:01:24	52.0801965	-0.61078333
17	102000	0.90896942	-0.01065661	50.3598	52.0801115	-0.6105788		07/04/2017 05:01:25	52.0801025	-0.61053567
18	103000	0.9089678	-0.01065227	50.3604	52.0800186	-0.61032995		07/04/2017 05:01:26	52.0800088	-0.61028767
19	104000	0.90896618	-0.01064793	50.3611	52.0799258	-0.61008147		07/04/2017 05:01:27	52.0799155	-0.61003967
20	105000	0.90896456	-0.01064359	50.3619	52.0798329	-0.60983287		07/04/2017 05:01:28	52.0798222	-0.60979117
21	106000	0.90896293	-0.01063924	50.3627	52.0797399	-0.60958378		07/04/2017 05:01:29	52.0797288	-0.6095425
22	107000	0.90896131	-0.01063491	50.3635	52.0796471	-0.6093353		07/04/2017 05:01:30	52.0796358	-0.6092945
23	108000	0.9089597	-0.01063057	50.3645	52.0795543	-0.60908683		07/04/2017 05:01:31	52.0795428	-0.60904617
24	109000	0.90895807	-0.01062623	50.3655	52.0794614	-0.60883811		07/04/2017 05:01:32	52.0794497	-0.6087975
25	110000	0.9089565	-0.01062201	50.3665	52.079371	-0.60859625		07/04/2017 05:01:33	52.0793555	-0.60855917

Figure 7.3: .xlsx file used in the Matlab files to calculate results