University POLITEHNICA of Bucharest

Automatic Control and Computers Faculty,
Computer Science and Engineering Department

BACHELOR THESIS

Formation Flight for Unmanned Aerial Vehicles

Scientific Adviser:
Prof. Adina Magda Florea
Ing. Mihai Trăscău
Ș.l. Cătălin Leordeanu

Author:
Alexandru George Burghelea

Bucharest, 2013

I want to thank all my colleagues that have made the Autonomous UAV project possible.
Special regards go to my mentors for guiding me in elaborating this thesis.

I also want to thank Nistor Moț for managing to combine my two favorite fields , aviation and computer science, by saying:

"Software that almost works is like airplanes that almost fly!"

Abstract

Here goes the abstract about UAV Formation Flight.

TODO:
Write abstract

Contents

Notations and Abbreviations

ACS – Faculty of Automatic Control and Computer Science
CAN – Controller Area Network
ECEF – Earth Center, Earth Fixed
FDM – Flight Dynamics Model
FG – Flight Gear Instance
FGFS – Flight Gear Flight Simulator
FGMS – FlightGear Multi-Player Server
GPS – Global Positioning System
GSR80 – Geodetic Reference System 1980
MAV – Micro Air Vehicles
ORCA – Optical Recognition Collision Avoidance
QGC – QGroundControl
RC – Remote Control
RPI – Raspberry PI
SO – Self Organized
TNI – Teamnet International
UAS – Unmanned Aerial System
UAV – Unmanned Aerial Vehicle
XML – eXtensible Markup Language

Chapter 1

Introduction

1.1 Domain Description

In the last 3 decades the aeronautic industry has focused on creating methods of flying that does not involve a human factor inside the airplanes, developing solutions for unmanned flight. The necessity for advancements in UAV domain is powered by the desire to keep human pilots out of harms way. UAV systems are useful in military missions, and high risk search and rescue missions. Along the military missions, UAVs can be used in civil context for missions like: traffic surveillance, cartography or animal tracking. An UAS or drone is a vehicle that doesn't have a human pilot on board and can be either controlled by a RC, a ground control system (Control Tower) or be fully autonomous. The concepts of unmanned vehicles emerged a couple of years after the first mechanized flight in 1903 by Orville and Wilbur Wright [17]. In 1915 Nikola Tesla had a vision about a fleet of unmanned military aircrafts and in 1919 was developed the first UAV by Elmer Sperry, that was used for sinking a captured German battle ship. The first two countries that saw the high potential of unmanned vehicles were U.S.A and Israel. In 1960 the U.S. Air Force started a research program for developing UAVs, and in 1964 is the first documented use of an UAV in a real war scenario, during the Vietnam War. Israel started using UAVs for reconnaissance and surveillance mission. As a result, Israel reported no downed pilot during the Lebanon War in 1982. In the present, drones are intensively used in the war theaters from Afghanistan and Iraq [6]. The development of the autopilot is strongly correlated to the with the developed of the UAV. The company of Elmer Sperry was the first to produce an autopilot that was able able to fly autonomous for three hours in a straight line without being supervised by a human. By 1933 Sperry's autopilot was able to flight on true heading and maintaining the altitude, compared to the gyroscopic heading of the first version.The current evolution of autopilots is in close relation with the development of reliable communication systems. Although the first UAVs were controlled remotely by a human operator, they are now able to receive a flight plan and based on that to calculate the flight path and follow it to complete the mission. In moder autopilots the human factor the secondary role of supervising the system and controlling the on board equipment, like cameras and sensors.

1.2 Motivation

When I was a child I received my first toy airplane and became fascinated by the idea of moving freely like a bird.A couple of years later I first stood near a MIG-21 Lancer at my fathers garrison. The passion with witch the pilots talked about being in the air close to the clouds inspired me the love for moving freely in 3 dimensions.

The high number of human casualties reported in war theaters and training missions determined me to explore the field of autonomous fling. The necessity for reducing the loss of human lives gives autonomous great potential for evolution.

My motivation is to help create a next generation of autopilots capable of accomplishing difficult missions where it is the risk for a human pilot would not be affordable.

In Romania the UAV fields is still unexplored. The main fields where a UAV platform would be useful are interest points detection and monitoring, border patrol, search and rescue team and imagery intelligence.

1.3 Objectives

Although a single UAV is already able do accomplishing various mission by it's own, an interesting, and in my opinion mandatory, field is the one of fling in formation. A mission where the objective is to track multiple targets becomes very hard for a single drone, thus emerging the necessity for a swarm of UAVs. There are situation where the risk of loosing an UAV due to hostile conditions is to high, being more affordable to deploy multiple, cheaper UAVs in contrast to an expensive drone. Another use case for a formation would be a search mission where we can't equip a single aircraft with all the sensors necessary for success and choosing to use multiple specialized drones.

Usually a human pilot is able to fly in formation using a combination of cognitive and reactive behavior, always making small adjustments to maintain a coherent formation.

There are two ways that formation flight could be achieved. One would be a centralized method, where all the drones report the telemetry data to a central authority like a ground control system and the later would make the necessary decisions for all the involved actors and then relay the data back to the aircrafts. Although in theory this approach could give an optimal flight path, problems like delay in communication and sensors reporting faulty data could jeopardize the success of the mission. Another approach would be a decentralized method, inspired by swarms of animals, like ants or bees. In the second approach each UAV would decide what actions to execute based on the actions of the others.

The main goal of this thesis is to design an decentralized algorithm responsible for maintaining a flight formation based on the leaders actions. The leader will not share the flight path or mission plan with the other drones, it will share only the current position, speed and direction. Based only on this data, the drones must be able to maintain a predefined flight formation. Thus each drone, except the leader, is modeled as an reactive agent that has the mission to approach the leader and mimic his actions.

The secondary goal of this paper is to design a management platform for a fleet of airplanes that are able to execute different missions. The platform has the role of programming the mission for each drone, manage the in flight performance for each UAV and if necessary to send commands, inserted by a human supervisor, to the drones and by this modifying the current state of the mission execution.

The platform developed is possible thanks to a collaboration between the TNI company and ACS, University Politehnica Bucharest.

This thesis present a possible solution for achieving UAV Formation Flight.

Chapter 2

Related Work

A large number of articles describe the work done to solve the problem of autopilots UAV swarms, communication and coordinate systems. In the following paragraphs I will describe the main ideas and trade-offs for each described solutions.

The autopilot problem is well covered in the literature and from real life practice it is considered that the best architecture is one that is stratified. Each layer is responsible for receiving commands from the superior layer, deciding if the the command would put the UAV in a state of imbalance, or danger in general, and forwarding the command to the layer below. If a possible imbalance state is detected the layer either discards the received command and does not send any command to the next layer, or it tweaks the command so that a incoherent state would not be induced.

The architecture as proposed by Borges de Sousa et. al [5] would have the following layers:

Platform
: The UAV vehicle with all the hardware

Maneuverer controller
: Controller that decides what hardware action is executed (roll, pitch, yaw)

Vehicle supervisor
: Basic autopilot capable to make simple decisions (setting the target speed, setting the heading)

Mission supervisor
: Artificial Intelligence System that plans the mission based on a template and the rest of the drones. It forwards the commands to the Vehicle Supervisor for validation

External controller
: Human factor that can interfere between the Mission Supervisor and Vehicle Supervisor to override or even deactivate the first one.

The architecture imagined by Borges de Souse can be seen in Figure 2.1.

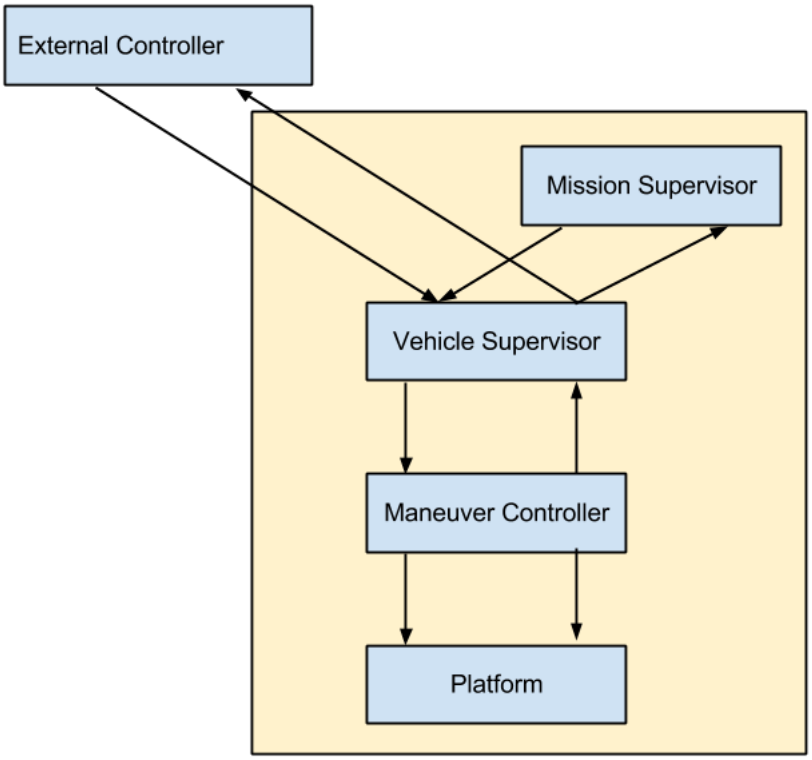

Figure 2.1: Vehicle Control Architecture.

The advantages provided by this kind of architecture is the separation of concerns for each level and the fact that the repair in case of failure can be easily detected and fixed. The only draw-back of this approach is the fact that in the case of poor implementation it could introduce a latency in communication loosing the possibility of having a real time system.

The autopilot system presented in this thesis is similar to the one described above because the Hirrus drone [18] provided by TNI already contains a maneuver controller,vehicle supervisor and an external controller. Thus the system described in these thesis would act as a mission supervisor.

In the terms of UAV teams, Mark D. Richards and his colleagues [16] identify two main groups of strategies. The first one, called **Behavior-based Control Systems**, uses a mesh of interacting high-level behaviors to perform a task. The second one, **Deliberative System**, acts by creating a specific flight path for each individual UAV to follow. This second behavior is a generalization of the search path optimization that Ablavsky proposes in [1]. Ablavsky approaches the problem by following the following steps:

1. Restrict the search area based on the mobility of the target.
2. Divide the search area in to the smallest number of sub-regions keeping in mind the constrains of the aircraft.
3. Determine a search pattern for each sub-region with the property of assuring full coverage with a minimized path length.
4. Combine the individual results into an optimal global path.

The deliberative approach used by Richards is based on dividing an area for each UAV and generating inside the designated zone a flight path to be followed. For achieving some degree of

flexibility, Richards opted not to use an adaptive replanning where a central controller computes a specific flight path for each agent and and then broadcasts it to the team. The drawbacks of this approach are represented by the fact that there would still be a single point of failure and the fact that by the time the plan is sent to the UAV it may be already be deprecated. The approach that was used was a reactive one. The initial path is computed and if an hostile condition is generated each UAV has to determine a way to exit that state. By these means Richards managed to sweep an area that is divided in sub-zones with different degrees of danger and even a no fly zone. The reactive behavior was useful for avoid collision with a friendly unit or for escaping a danger zone.

Although Richards proposal uses a long range unsynchronized flight team, in this thesis I used a similar approach for obtaining a synchronized formation.

Gaudiano and his team research the possibility to explore a zone based on the strategies used by schools of fish, flocks of birds and swarms of insects. The research they published in [7] and [8] is based on mimicking the pheromones left behind by the swarms of insects. To be able to replicate the insects behavior, Gaudiano assumed that each UAV is aware of the terrain geometry, is equipped with a sensor that provides a forward code of vision able to detect elevation and distance, has a sensor to detect the other UAVs in a given sphere, is positioned using GPS-like coordinates and has a sensor that mimics the pheromone detection. The later sensor detects, within a rectangular region centered on the UAV, the coverage of the current cell. In their simulation they have used a global communication system that was able to assure data flow between each two UAVs. The strategy to control the UAVs is implemented in a decentralized way where each UAV can decide what action to execute. Gaudiano tested 5 strategies to explore an area:

Baseline
: Each UAV starts with a different heading, flies in a straight line until the area limit is reached and then turns so that it doesn't leave the area.

Random
: Is similar to the baseline implementation but at each time step, the UAV will change it's heading by a small random angle.

Repulsion
: An UAV can detect the other UAVs on a given radius and it changes it's heading so that they do not intersect.

Pheromone
: Each UAV is able to detect the already explored cells and changes it's heading so that it heads in the direction of unexplored cells.

Global
: The area is divided in smaller regions ana an UAV knows how many UAVs are in one region and based on that it decides what region to explore.

Based on their experiments they have discovered that the most efficient way to obtain a higher coverage is the pheromone strategy. Compared to the repulsion strategy that obtained a 44% coverage the the pheromone strategy obtained a 60%. These coverages were obtained using 10 UAVs. The same experiments showed that scaling to larger UAV swarms presents a drawback represented by the fact that although the coverage efficiency increases with the number of UAVs, their relative efficiency decreases. While on a very large area 10 UAVs would cover 6%, 110 UAVs would cover only 33%.

In contrast with the approaches form above, Nowak centered his research on creating a self organized swarm. To achieve this he created a three-tired architecture for a SO System Model that separates the implementation details from the real world agents behavior. This architecture (as presented in [14]) can be seen in Figure 2.2.

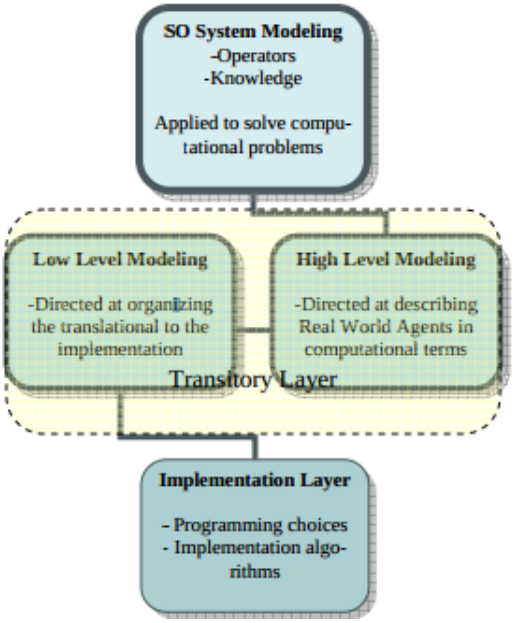

Figure 2.2: Three tier SO architecture.

The behavior of many cooperative agents in the same way is a behavior that has been studied in detail by many researchers. Reynolds proposed in 1987, three fundamental rules for swarming: [15]

Separation
The UAV must steer away from the nearest neighbors to avoid crowding.

Cohesion
The UAV must steer towards the average position of its neighbors.

Alignment
The UAV must steer towards an average heading of the neighbors.

A visual representation can be seen in Figure 2.3

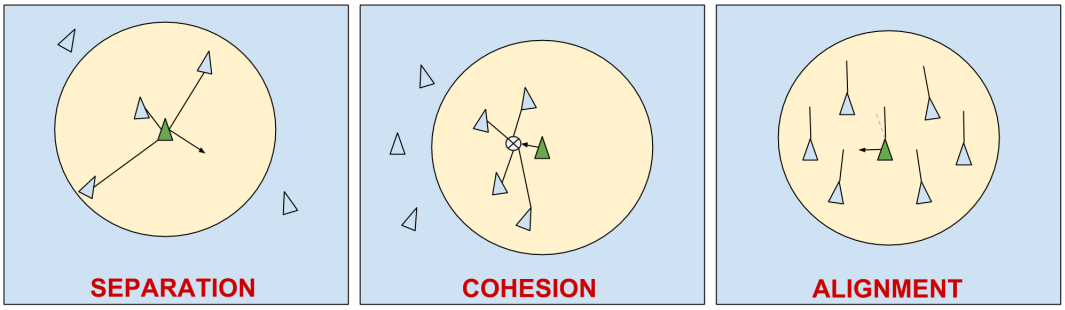

Figure 2.3: Fundamental Swarming Rules.

Based on Reynolds rules, Nowak implemented, in SWARMFARE 10, rules for a coherent formation flight [14]:

Flat Align
vector align neighbors

Target Orbit
orbit target at safe distance

Cluster Range Towards
cohesion

Cluster Range Away
separation

Attract
towards center of mass of all targets

Weighted Attract
towards closest target

Target Repel
repel if within 90% of UAV sensor range

Weighted Target Repel
repulsion based on proximity to target

Evade
Apriori collision detection and avoidance

Obstacle Avoidance
Real time obstacle avoidance

The difference between the previous approaches and Nowak's is that the later considers that at no point an agent will give commands such as "Slow down", "Turn" or "Dive". Each UAV is responsible for computing what action to execute based on a minimal set of date received from the other UAVs. A possible set of data is formed from:

- GPS coordinates
- altitude
- speed
- heading

To test the SWARMFARE framework, Nowak opted to paralyze the computation using a Master-Slave architecture. The resulting parallel time calculation results from the equation: f

$$T_p = k(t_s + t_w m)(p-1) + \frac{n^{ck}}{p} \qquad \begin{aligned} k &= \text{number of generations} \\ t_s &= \text{setup time} \\ t_w &= \text{transfer time} \\ p &= \text{number of processors} \\ n &= \text{number in population} \\ p &= \text{constant defining the variable} \\ n^c k &= \text{linear calculation time} \end{aligned} \tag{2.1}$$

The time obtained by using a initial population of 2 per node is 540 minutes for one node and 80 minutes for 16 nodes.

The SWARMFARE is a work in progress and the trade-offs that could appear only from the implementation of the 10 rules.

The Autonomous UAV project proposes to implement an autopilot that integrates the behaviors depicted by the rules proposed by Nowak. The formation flight module implement in this thesis will integrate with the obstacle avoidance module and surveillance modules developed by my colleagues.

Based on the communication mode, there are two main methods for implementing UAV formation flight: coordinated or cooperative. Bourgault et. al. depict the difference between the two

modes in [2]. According to them in coordinated approach each agent decides what actions to execute based on the current knowledge of the world but there is no possibility that the actions of an agent to influence the decisions of another. The cooperative mode has the advantage that agents negotiate what actions to execute, thus obtaining a global optimal solution. On the other side the coordinated mode obtains a suboptimal solution, but has the advantage that the individual computation time does not increase with the number of agents.

For a close range formation there are necessary both these modes. The coordinated mode is useful for computing the position in the formation for each aircraft and the cooperative mode is useful for a collision avoidance module so that the aircrafts do not collide. Similar to Bourgault, in this thesis is presented a coordinated mode with a one-step look-ahead.

For obtaining obstacle avoidance it's necessary to have a reactive system combined with a cooperative decision making algorithm. Call et. al tried to implement such a system by obtaining data from video cameras. The cameras are preferred because they are passive, lightweight, simple and inexpensive sensors. The drawback for these sensors is represented by the fact that under low-visibility, such as fog, smoke or night, have reduced efficiency. Also the processing of the information could have high computation time. In the research paper [3], they have managed to obtain three dimensional data from cameras either by using a stereo vision system or by using multiple frames from one camera. They have found out that the relative position of an object can be obtained with an enough amount of information by using a single camera and multiple frames. The main problem with detecting obstacles is that most of the existing algorithms are based on detecting a texture variation or the detection of corners. Problems in detecting obstacles appear when the objects are described by parallel lines with no visible endpoint or have no texture. Another case when the algorithm could fail is when the obstacles have reflective glass surfaces.

The results published in [3] can be seen Figure 2.4

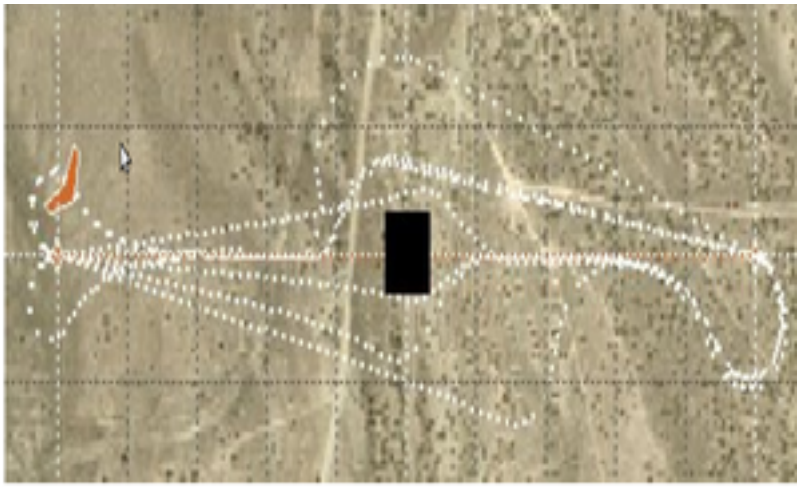

Figure 2.4: Overhead view of the flight path of the UAV as it avoids an obstacle indicated by the black box.

When using cameras to detect obstacles, the angular velocity of gyroscope often has a large amount of noise, mostly because of the yaw angle or wind. Thus the angular velocity has to be estimated using another type of on-board sensor.

He approaches this problem by following the following steps [9]:

1. Classify the image blocks in structural and non-structural blocks
2. Find multiple "best" motion vectors instead of a single one and chose the optimal one using a predefined metric
3. Using the data about structural motion, estimate the camera motion to reduce uncertainty in the motion for non-structural blocks.

In the same article [9] uses the following two formulas to compensate the motion noise of the camera:

$$m(O_B, r) = \frac{1}{|C(O_B, r|} \oint_{C(O_B,r)} I_t(x, y)dxdy, 0 < r \leq Rs \tag{2.2}$$

Where:

- O_B = (X_B, Y_B) center of block B
- C(O_B , r) = circle center in O_B with radius r
- R = maximum radius to search
- m(O_B , r) = intensity profile of pixel O_B or block B

$$d_1(A, B) = \min_{1-\delta \leq \lambda \leq 1+\delta} \max_{0 \leq r \leq \frac{R}{\lambda}} |m(O_A, \lambda r) - m(O_B, r)| \tag{2.3}$$

Where:

- λ = scaling factor
- $[1 - \delta, 1 + \delta]$ = search range for λ

From Equation 2.2 and Equation 2.3 the distance between blocks A and B can be computed with the following formula:

$$d(A, B) = wd_0(A, B) + (1 - w)d_1(A, B) \qquad w = \text{ weighted factor} \tag{2.4}$$

The formation flight module implemented for this thesis will integrate with a module of collision avoidance based on ORCA.

Millington proposes a set of algorithms for implemented moving in [11]. In his book, he divides these kind of algorithms based on variable of speed and acceleration. The categories are:

Stationary and Running where the output is represented simply by the direction of movement. The movement imposed by this type of algorithm is called **kinematic movement** and it is not influenced by acceleration or deceleration.

Dynamic where the output is represented by accelerations or forces that are meant to change the velocity of the agent.

If the worked environment is a 2D model, the output from this algorithms would be represented by one (or both) of the two following values: *a*) clockwise angle in radians from the positive z-axis (the vertical axis being the y-axis); and *b*) acceleration needed to change velocity . The mathematical model necessary used in 3D could be very complicated, thus Millington

proposes in [11], *Chapter 3. Movement* a hybrid coordinate system with 4 degrees of freedom called $\mathbf{2\frac{1}{2}}$ **Dimensions**. In most 3D simulations these system use sufficient because an agent is always pulled down by the gravity. Even in the cases where a character is looking up or down, the movement direction is not influenced by the viewing direction. For computational purposes it is usually more convenient to represent an orientation as a vector instead of a scalar. The transformation can be computed with the following formula (assuming a right-handed direction):

$$\overrightarrow{w_v} = \begin{bmatrix} \sin w_s \\ \cos w_s \end{bmatrix} \qquad \begin{matrix} w_s = \text{ orientation as scalar} \\ \overrightarrow{w_v} = \text{ orientation as vector} \end{matrix} \tag{2.5}$$

Millington proposes a set of algorithms for achieving kinematic movement by using only static data (position and orientation without velocity). The algorithms are described below [11]:

Seek
: Based on static data about the source and destination, the direction between them is computed and movement along that line is required. If the source is static, moving at full speed could cause the agent to pass the destination and move back and forward reaching a state of **Stable Equilibria**. To avoid this, it could be considered that the target has been reached if the agent is at a distance from the target that is smaller than a predefined radius.

Wandering
: The agent always moves at full speed in the direction of the current orientation and at each step the orientation could be altered by a small amount using a binomial random function (computes values between -1 and 1 with a higher probability for 0.

For these thesis a **Wandering behavior** would not be useful, but the **Seek behavior** is used for obtaining a *follow the leader* movement.

Millington also proposes a set of steering behavior based on outputting accelerations:

Variable matching One agent tries to mimic the kinematic of another agent.

Seek and Flee One agent tries to mimic the position of the target agent or move further from the target, being similar to the **kinematic Seek**

Arrive Is an improvement from **Seek**, but here the agent slows down when is close to the target.

Face Here the agent first calculates the target orientation and then tries to head in the direction of the target's next position.

For movement in 3 dimension, Millington states that the algorithms described by him have to be adapted by using quaternions.

In this thesis I used the *Variable Matching* behavior for fling in the same direction and *Seek and Flee* for entering formation.

Chapter 3

UAV Management Platform

3.1 Architecture

For the Autonomous UAV platform we have used an architecture based on multiple agents that communicate with the *Mission Supervisor* via a CAN. The CAN will also be able to route messages to other CAN buses.

The testing architecture can be seen in Figure 3.1.

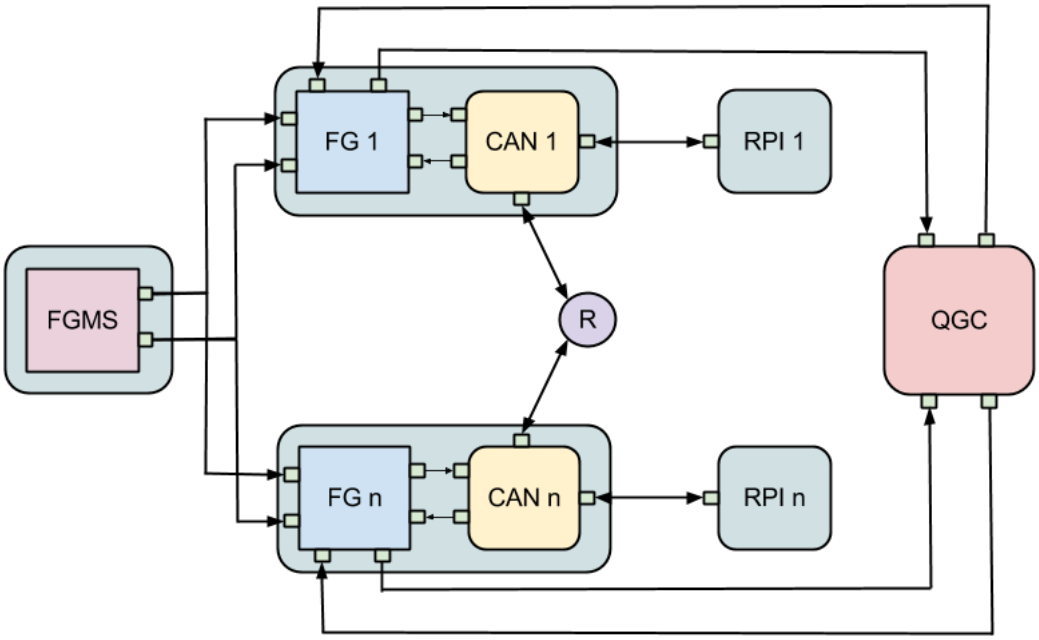

Figure 3.1: Autonomous UAV testing framework.

In Figure 3.1, the components are as follows:

$FG_{1..n}$ FlightGear Flight Simulator. Simulator responsible flying a single drone.

FGMS
FlightGear Multi-Player Server. It has the role of broadcasting information about UAVs between multiple instances of *FlightGear Flight Simulator*

$RPI_{1..n}$
Raspberry PI Computer. Runs the *Mission Supervisor*.

$CAN_{1..n}$
CAN Interface Simulator. Mimics the behavior of a CAN bus and sends messages between a *Flight Gear Flight Simulator* instance a *Raspberry PI Mission Supervisor*

R
CAN Interface Router. Broadcasts the telemetry position from one *Flight Gear Flight Simulator* instance to the rest of the instances.

QGC
QGroundControl. Ground Control Software responsible of collecting data from all the UAVs and plotting them on a map for visualizing the flight path.

In the Autonomous UAV project, QGC also has the role to prepare the mission plan that will be uploaded on each UAV.

The *Mission Supervisor* architecture is depicted in Figure 3.2.

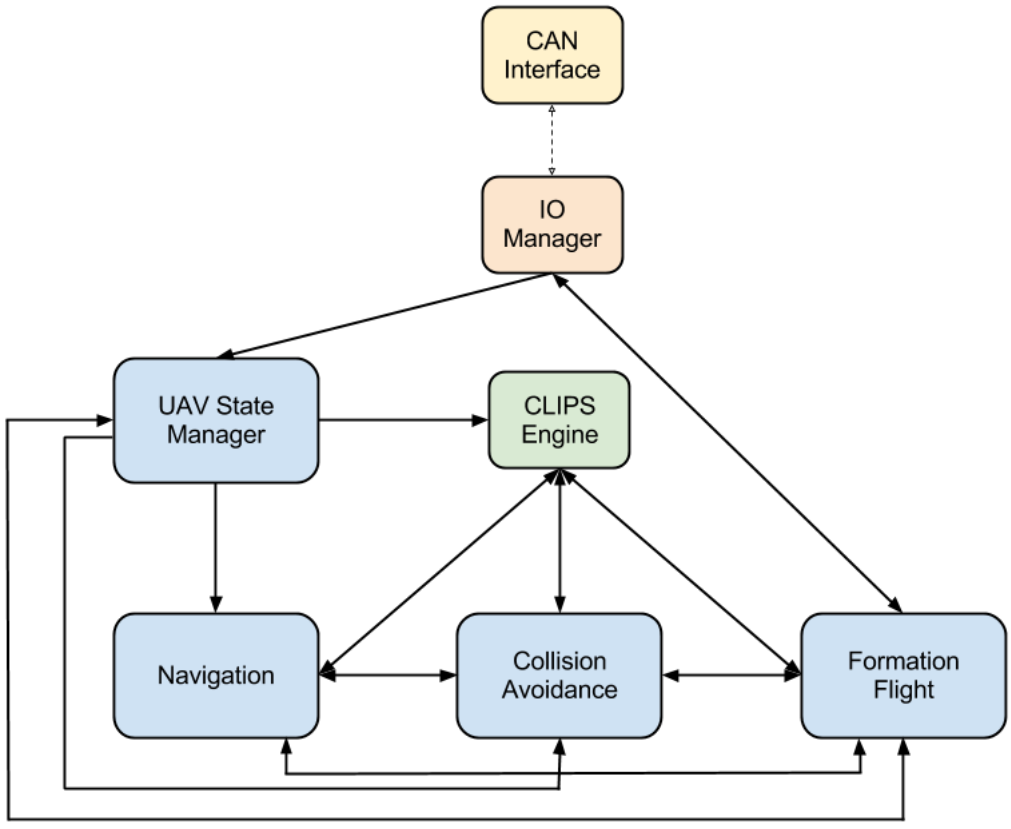

Figure 3.2: Mission Supervisor Architecture.

In Figure 3.2, the components are as follows:

IO Manger
Input-Output module responsible for communicating with the UAV via the CAN bus and passing the date to the flight modules.

Clips Engine
Rules based decision engine.

UAV State Manger Module responsible for monitoring the state of the UAV. The date that it monitors includes heading, speed, position etc.

Navigation
Module responsible for deciding the flight path

Collision Avoidance
Reactive module responsible for detecting and avoiding in flight collisions.

Formation Flight
Module responsible for coordinating a fleet of UAVs for maintaining a coherent flight formation.

This thesis handles the UAV Formation Flightmodule that will be integrated with the other modules developed in the Autonomous UAV project to obtain an automated pilot that will be used for controlling the Hirrus UAV [18].

3.2 Functionalities

The platform has the roles of:

1. Setting the flight mission objectives.
2. Selecting the flight area.
3. Selecting the aircraft types.
4. Configuring the sensors for each aircraft.
5. Based on the settings from above, generate and upload a configuration file to each aircraft.
6. After the airplanes are airborne, they will be supervised using the ground control module.
7. To control the airplanes in an autonomous way, where human intervention is needed at least as possible.
8. Override the *Mission Supervisor* or completely deactivate it, switching to a RC Controlled state.

For using the platform a will act as described below.

Using the custom widget built in QGroundControl will configure the flight objectives, selecting the number of aircrafts that will be launched, the type of sensor that will be equipped on each drone. Also the user has to create a flight path for each UAV if they are not fling in a tight formation. If the drone will fly in a close ranged formation, the user has to designate a leader and create a flight plan for him, the other UAVs will follow his path. In the case of close ranged formation, the user will specify the kind of formation that will be used (eg: V formation, Line formation). After the mission details have been set, the user will generate a configuration file that contains a rule based language that is close to the natural language and that will be interpreted by a CLIPS engine. The configuration file will be uploaded on the UAVs. When the drones are ready they are airborne and the autonomous pilot will guide to follow the mission.

From this point their mission can be overseen using QGroundControl where their flight path is displayed on the map and where their telemetry is also displayed. In case of necessity, the user will send additional commands to the drones, or it will will switch the drone to RC mode and return it to base.

Chapter 4

Formation Flight

For fling in formation a coordinate system is needed for both positioning each aircraft on the world and for positioning an aircraft relative to a leader. In aviation positioning the aircraft is usually done by GPS, but this is not the best best solution for close range formations.

4.1 Coordinates Systems

The Earth has a very complex and irregular shape. For mapping the position of the aircraft a simpler mathematical model is needed. In geodesy this simpler model is called *figure of the Earth*, where the surface is usually abstracted to the **Geoid** or **ellipsoids**. The Earth is a biaxial ellipsoid where the shorter axis almost matches the rotation axis [13].

Because the shape of the Earth doesn't fit perfectly on any ellipsoid, there are currently multiple ellipsoids that are in use. For example the GPS uses GRS80 [12] and in there United Kingdom the Airy 1830 ellipsoid is used [13], being designed to perfectly fit Britain only. The later ellipsoid is not useful in other parts of the world. For measuring heights, it's necessary to define an imaginary surface that represents the *zero height*. The Geoid is used to represent increasing heights uphill and decreasing height downhill. The zero height shape is commonly called **sea level**.

4.1.1 Latitude, Longitude, Altitude

Commonly the position of the aircraft is usually done by starting with two angles called latitude and longitude. These two angles define a point on the ellipsoid that fits the globe. Thus it is mandatory to know with witch ellipsoid we are working for having any degree of certainty. On the globe the North-South line are known as meridians, and East-West lines are called parallels. To accurately position an aircraft in air a third dimension is needed. Usually the third dimension used is altitude that is measured from *sea level*. Having a 3D Cartesian System with the z-axis oriented towards the North Pole and the x-axis and y-axis point towards the Equator, the **Zero on longitude** would be represented by the Z-X plane and **Zero of latitude** would be represented by the X-Y plane, depicted in [13] with the following Figure 4.1.

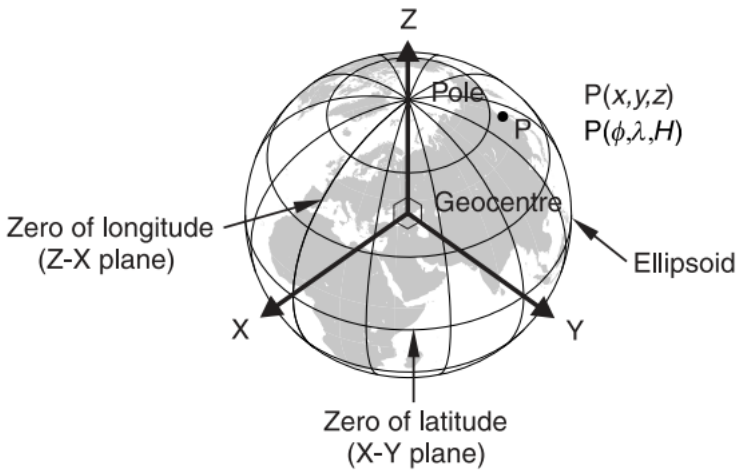

Figure 4.1: Latitude, longitude representation.

4.1.2 Earth-Centered, Earth-Fixed

An alternative to the angular coordinates system is the Cartesian system. Usually satellites use a 3D Cartesian System called ECEF Coordinates. For the Earth, the convention is to have the origin of the system placed at the center of the planet. The center is defined as the barycenter (center of mass of the earth) and respects the following mathematical formula:

$$\int \overrightarrow{x}\rho dx^3 = 0 \qquad \begin{array}{l}\overrightarrow{x} = \text{ position vector} \\ \rho = \text{ density over Earth}\end{array} \tag{4.1}$$

If Equation 4.1 is not satisfied, the center should be adjusted [4].

As seen in Figure 4.1, in ECEF the z-axis points to the North Pole, the x-axis points to the 0° meridian and leaving the y-axis to be set so that a right handed system is created.

4.1.3 Conversion

Conversion between an Ellipsoid System and a Cartesian System is possible as long as the ellipsoid parameters are known. Modern GPS systems use the WGS84 ellipsoid.Coincidently the ellipsoid has the same center as the ECEF system. The ellipsoid parameters fro WGS84 are [10]:

Semi-major axis
$a = 6378137$

Semi-minor axis
$b = a(1 - f) = 6356752.31424518$

Ellipsoid flattening
$f = \frac{1}{298.257223563}$

First Eccentricity
$e = \sqrt{\frac{a^2-b^2}{a^2}}$

Second Eccentricity
$e^{'} = \sqrt{\frac{a^2-b^2}{b^2}}$

LLA to ECEF

The conversion in meter from LLA to ECEF is described by the following equation system:

$$\begin{cases} X = (N+h)\cos\varphi\cos\lambda \\ Y = (N+h)\cos\varphi\sin\lambda \\ Z = (\frac{b_2}{a_2}N+h)\sin\varphi \end{cases} \tag{4.2}$$

where

φ
: = Latitude

λ
: = Longitude

h
: = Height in meters above ellipsoid

N
: = Radius in meters of Curvature, $= \frac{a}{\sqrt{1-e^2\sin^2\varphi}}$

ECEF to LLA

Converting from ECEF to LLA is more complicated but it can be achieved using one of the following methods [10]:

Iteration for φ and h
: This method converges quickly for $h \ll N$ starting at $h_0 = 0$

$$\begin{cases} \lambda = \arctan\frac{X}{Y} \\ h_0 = 0 \\ \varphi_0 = \arctan\frac{Z}{p(1-e^2)} \\ N_i = \frac{a}{\sqrt{1-e^2\sin^2\varphi_i}} \\ h_{i+1} = \frac{p}{\cos\varphi_i} - N_i \end{cases}$$

Closed set formula

$$\begin{cases} \lambda = \arctan\frac{X}{Y} \\ \varphi = \arctan\frac{Z+e'^2 b\sin^3\phi}{p(-e^2 a\cos^3\phi)} \\ h = \frac{p}{\cos\varphi} - N \\ p = \sqrt{X^2+Y^2} \\ phi = \arctan\frac{Za}{pb} \end{cases}$$

In this thesis the *Closed set formula* was used to convert from ECEF coordinates to GPS coordinates.

4.2 Formation types

Across the world human pilots are able to fly in formations that varying in both shape and difficulty. The most used formations are usually formed by 3 or 4 airplanes.

In teams of 3 airplanes the most common formations are *Aine Astern* and *V Formation* and can be seen in Figure 4.2.

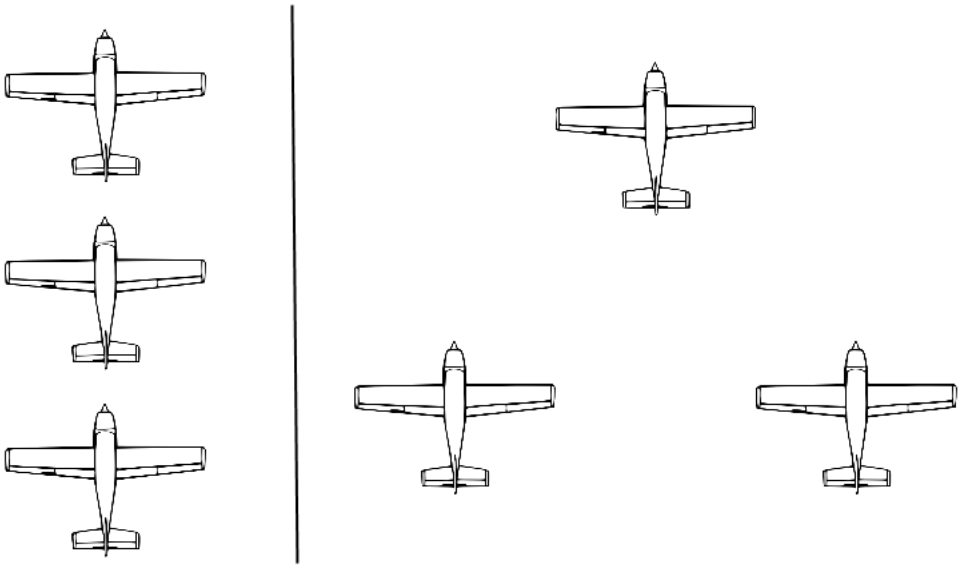

Figure 4.2: a) Line Astern Formation; b) V Formation

When having 4 airplanes the usual formations are: *Line astern, Box, Finger Right, Finger Left, Echelon Right, Echelon Left* and can be seen in

Figure 4.3, Figure 4.5 and Figure 4.4.

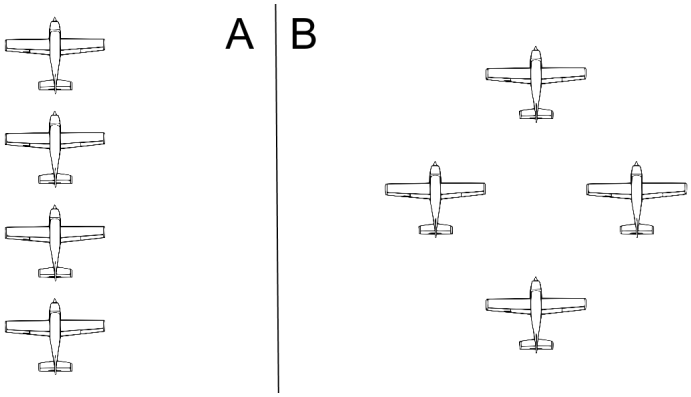

Figure 4.3: a)Line astern and b) Box

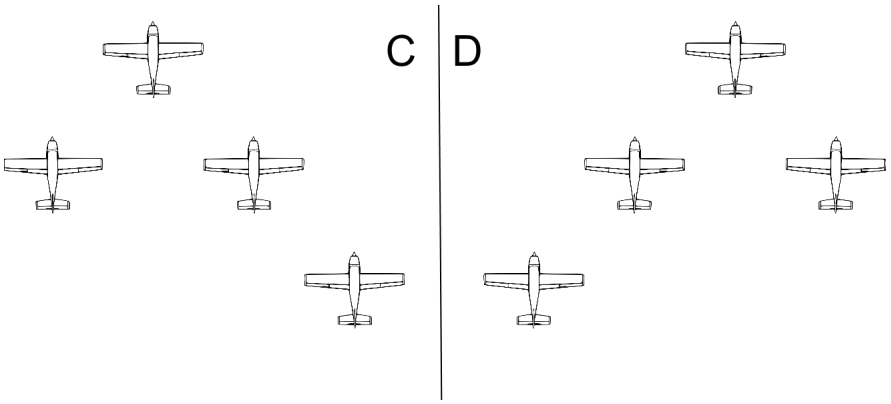

Figure 4.4: c) Finger Right and d) Finger Left

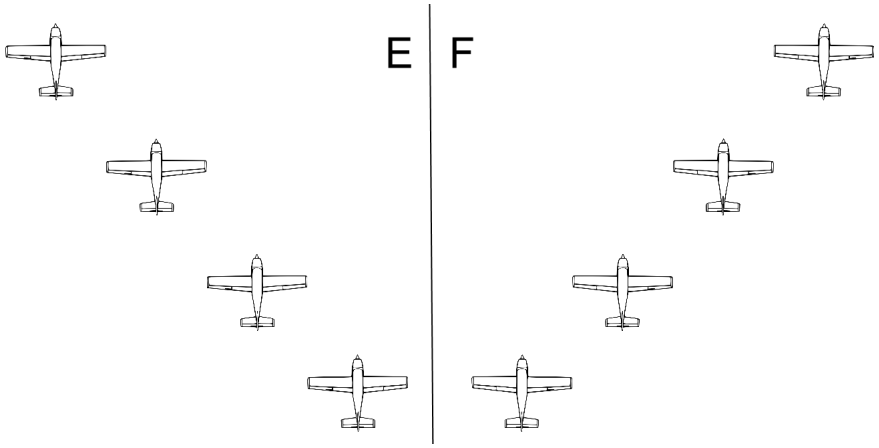

Figure 4.5: d) Finger Left and e) Echelon Right

In this thesis I studied the V Formation and *Line Astern* with different altitudes.

4.3 Entering the formation

TODO:
Describe Formation Entering

4.4 Maintaining the formation

TODO:
Describe entering the formation

Chapter 5

Implementation details

5.1 FlightGear

For the Autonomous project we have chosen an open-source, multi-platform flight simulator called **Flight Gear Flight Simulator**. It was first released in 1997 under *GNU General Public License* by David Murr. Being developed by a mature community in an academic environment, FGFS has reached in February 2013 version 2.10. Currently it is being used by various universities and companies for FAA flight simulators and research projects. Some of the universities that use FGFS are: University of Minnesota, Department of Aerospace Engineering at The Pennsylvania State University, University of Naples and University of Toulouse along with ATC Flight Simulator Company Aerospace Engineering Institute from RWTH Aachen.

Being an open-source product, it offers a high degree of freedom the code being easy to modify. Because it is written in C++ the code is cross-platform, running on various different operating systems (eg: Windows, Linux, MAC OS). For obtaining a higher degree of versatility, Flight Gear is able to use multiple FDMs. Independent implementations like JSBSim and YASim are built as libraries and integrated with FGFS. For these project we have used a remote model called Rascal110 bundled with JSBSim. The reason for using the Rascal110 model is that it has similar aerodynamics and dimensions as the Hirrus drone, for witch we are building this autopilot.

Another reason that led to the choice of this simulator is the fact that the central components can be configured by a component called **Property Tree**. Each node in the *Property Tree* represent one parameter of one component and the interaction between components can be assured by changing the values of the parameters. The *Property Tree* can be accessed in multiple ways. One of the modes is represented by a web interfaced where the tree can be navigated and using submit forms the values can be changed. Another is to specify the desired parameters as command line arguments. At startup FGFS also reads an external XML configuration files where any property can be set. This allows an external user to change the current status of the aircraft by modifying roll, speed, pitch, acceleration or even positions.

To connect to external components, FGFS uses a network communication module that uses either TCP or UDP sockets. In combination with the *Property Tree* this modules can offer full control for outside sources. The communication module requires an XML files that defines the file format. In the XML there are defined an **input** and **output** formats. The *output* format is usually used to notify the external controller of the aircraft's state, while the *input* format is used by the external controller to send commands to control the aircraft. The messages have a C-like format and each value can be configured to have the needed precision.

For these thesis the modules that were mostly used are the **Autopilot** and **Route Manager**. The *autopilot* has a simple interface and supports commands to specify heading, altitude or speed.For speed and heading changes, internally, FGFS implements PID controllers.

For ensuring a global environment for all the UAVs, we have used another component called Flight Gear Multi-player Server. FGMS has the role of representing in the same environment all the connected UAVs. FGMS acts as a global server where each UAV is a client and broadcasts all the positions and flight date to all the instances. This feature is necessary for visually observing the UAVs during the flight simulation.

In Figure 5.1 can be observed from diferent perspectives several UAVs fling in the same environment.

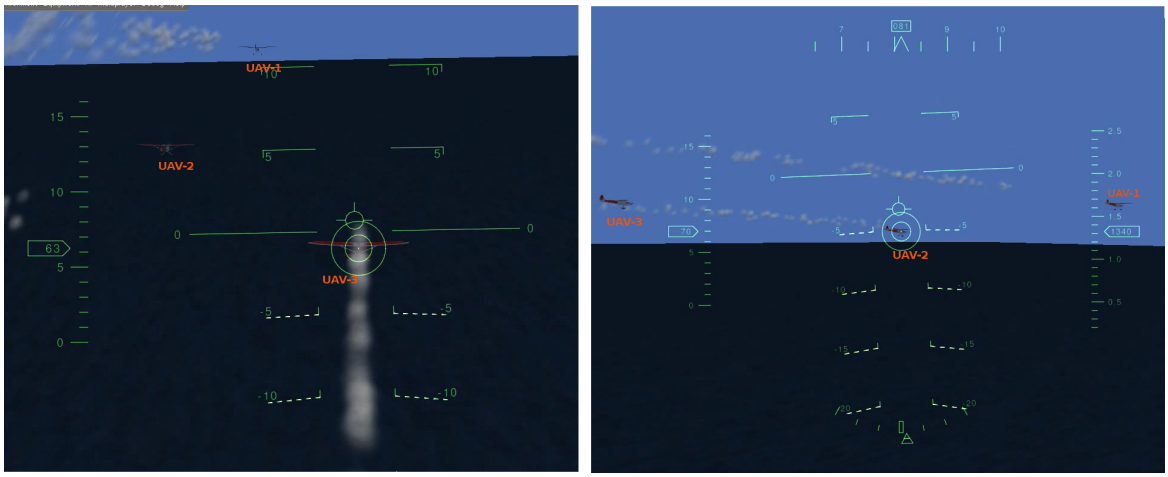

Figure 5.1: Multiple UAVs in the same environment from different perspectives

5.2 QGroundControl

Another mandatory component for controlling a fleet an UAVs is a ground control systems, that would allow th a graphical visualization of the mission and flight parameters. For this thesis the chosen ground control system is QGroundControl.

The base version that was used is v1.0.5. Originally QGC was developed for the PIXHawk quad-rotor drone by Lorent Meier at ETH Zurich. QGC offers supports for plotting the flight paths on a map, defining waypoints and event remote controlling the drones. It is developed in C++ using Qt, making it a cross-platform, modular architecture. QGC supports radio-copters, fixed-wing drones and even. RC cars. For communication with external controllers QGC accepts UDP connections, TCP sockets, USB connections and even radio links.

Being developed for MAVs, QGC uses an open-source protocol called MAVLink. MAVLink is based on a set of predefined messages for communication between MAVs.QGC offers various widgets for facilitating the use of MAVLink.

MAVLink uses a special message for tracking the integrity of the connected drones called a **heartbeat**. This message is expected to come at regular intervals of time. If this message is not received, the drone is considered incoherent and manual actions have to be taken. Even though other messages are received from the UAVs, without the heartbeat message, QGC sets them in a state of incoherence.

For this thesis the MAVLink protocol was bypassed and I implemented a generic protocol based on the messages sent by FGFS. Qt's signal mechanism permitted event generation every time a message was received from FGFS. The communication is based on two UDP sockets (one for input and one for output) for each UAV. Every time a message is received from FGFS it is parsed according to the XML configuration used at the startup of FGFS and the position, speed, orientation and other parameters are set and the MAV's position is rendered on the map.

An useful feature of QGC is the trail path rendered from the previous positions of the MAV. The trail can be set to leave a marker at fixed intervals of time or at fixed distances from one another.

QGC is used in different universities and research laboratories across the world like: University of Naples, French Aerospace Laboratory ONERA, University of Applied Science of Hamburg.

In Figure 5.2.

Figure 5.2: QGroundControl, monitoring the flight path of three UAVs.

5.3 Implementation

TODO:
talk about code, class-diagrams, describe methods, data-structures

The formation flight module was developed using the architecture describe in Section 3.1.

The connection between FGFS instances and QGC, the later was modified to establish UDP connection. For each monitored FGFS instance, QGC opens another UDP socket on witch it will send data back. For communication with FGFS, QGC uses a total number of $n+1$ sockets, where n is the number of FGFS instances and the last one is the UPD input socket. Also, I have modified the communication used by a MAV so that it does not use MAVLink. Instead it uses a generic protocol based on the FGFS protocol XML. Every time a message is received it is parsed and the position, orientation, speed and other telemetry data are set in QGC; After this data has been set a refresh signal is emitted and the position on the map is updated for each UAV.

Another QGC widgets was also developed in the *Autonomous UAV* project where the mission details can be configured. Using this widget a human user can select the number of UAVs used, their flight path, the equipment for every drone and the leader of the formation.

The formation flight module is written in C++ using the Boost library. The module connects to the CAN interface simulator via a TCP socket. When the module will be introduced on the Hirrus drone, on the Raspberry PI a character device will be crated that will read data from a CAN bus. Each drone will be equipped with a Raspberry PI that will run a Mission Supervisor.

At startup the module reads a JSON configuration file. The configuration file contains the number of drone and the geometry formation. Each UAV has a relative altitude to another UAV, a position (left or right) to the possible leader and at what distance it should be from the other UAVs. For extensibility reasons the JSON also contains data about the equipment.

After reading the JSON file, the data is store in a data structure called **uav_formation**. The *uav_formation* class contains the following fields and methods:

has_leader
: boolean flag that specifies if the formation has a UAV leader or a virtual leader

name
: the name of the formation; eg: V Formation

drones
: a vector of uav_drone structures that contain data about all the UAVs in formation.

getLeader
: returns the uav_drone if the formation has a UAV leader

getDrone
: returns the uav_drone with the desired name

An **uav_drone** class contains the configuration for each drone of how it should behave in formation and has the following fields:

name
the name of the UAV drone; eg: MAV 01

role
specifies the role based on the UAV equipment; eg: Infrared Scanner

personal_space
radius for a sphere were if another UAV is detected, mild evasive actions should be taken to avoid collision;

emergency_space
radius for a shpere where if another UAV is detected, the collision avoidance module will have maximum priority

is_leader
boolean marker that specifies if the current drone acts as the leader of the formation

geometry
array of uav_geometry specifying the relative position of the current UAV to other UAVs

The **uav_geometry** contains data about the relative position to another drone and it contains the following fields:

relative_drone
the UAV to witch the geometry is relative to

distance
array of three values containing the height difference between the UAVs, the heading multiply factor (used for determining how fast a UAV will turn to enter formation), the distance between the UAVs

The module starts to send commands to the CAN Interface as soon as initial position data is available about all the UAVs. Based on the number of drones needed in formation, the module waits until all the necessary data is received. As long as the initial data is not present, the UAV flight is not influenced by this module.

Data about each UAV is kept in a data structure called **position_container**. The *position_container* acts as a hash map, keeping as key the name of the UAV drone and as value a pair of two *uav_position* structures. The two *uav_position* instances represent the current and previous positions for each aircraft.

The **uav_position** class is the main class of the project and is responsible with creating commands for Flight Gear Instance. The class contains the following fields:

id
The id/name of the UAV

latitude
The GPS latitude

longitude
The GPS longitude

altitude
The altitude difference from sea level

heading
The true heading that the UAV drone is following

speed
The instantaneous speed of the UAV at the time of reporting

x
Value on ECEF x-axis

y
Value on ECEF y-axis

z
Value on ECEF z-axis

This class is also responsible for converting the GPS coordinates to ECEF coordinates and back. These methods are called: *lla2ecef* and *ecef2lla*. In this class the static method *parse_uav* creates a uav_position instance from a message received from the CAN Simulator. The received message is parsed using a regular expression and the position, GPS position, altitude, altitude heading and speed. After the ECEF data is set using the method *lla2ecef*.

Based on the formation configuration and current position, a command is created by setting the same altitude as the leader and determining the heading from the current position and and the desired position. For example, if the UAV is on the right of the leader and has to be on his left, the UAV receives a true heading that is smaller than the leaders heading. The UAV maintains this heading until the distance from the leader is in a desired range and is in the correct position. On the other hand if the UAV is on the left side of the leader and the formation specifies that it has to be on the left side, but the distance from the leader is to big, a true heading that is greater than the leader is reported, until the distance is in the desired range. To determine the side on witch the UAV resides, from the two known position of the leader the flight direction is calculated and if the point is on the left or right. In this calculations the altitude is ignored and considering that A is the leaders previous location and B is the leaders current location and C is the point for witch the side is computed, the formula used is the cross-product is Equation 5.1:

$$sign = \begin{vmatrix} B.x - A.x & B.y - A.y \\ C.x - A.x & C.y - A.y \end{vmatrix} \quad \begin{array}{l} sign > 0 \text{ point on left side of line} \\ sign = 0 \text{ point on same line} \\ sign < 0 \text{ point on right side of line} \end{array} \tag{5.1}$$

In the conversion between GPS and ECEF, an computational error is introduced in the mathematical model, mostly because the formulas consider the Earth Radius of constant size. In a *Line Astern* formation, this error determines the agents to slightly move from left to right. Also this error can be seen in calculating the distance between two UAVs. The distance computed in ECEF coordinates system using the Euclidean formal varies from the distance computed in GPS space using the Haversine formula with values ranging between 10 and 30 meters, for distance between 100 and 200 meters.

Chapter 6

Use cases

TODO:
describe use cases

TODO:
captures of 3 formations

TODO:
captures of flight-path

TODO:
possible missions

Chapter 7

Conclusions and future work

TODO:
describe conclusions and future work with low priority

TODO:
Add appendixes

Bibliography

[1] V. Ablavsky, D. Stouch, and M. Snorrason. Search path optimization for uavs using stochastic sampling with abstract pattern descriptors. AIAA Guidance Navigation and Control Conference, August 2003.

[2] Frédéric Bourgault, Tomonari Furukawa, and Hugh F. Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, October 2003.

[3] Brandon Call, Randy Beard, Clark Taylor, and Blake Barber. Obstacle avoidance for unmanned air vehicles using image feature tracking. American Institute of Aeronautics and Astronautics, 2006.

[4] James R. Clynch. Earth coordinates, February 2006.

[5] J. Borges de Sousa, G. Goncalves, A. Costa, and J. Morgado. Mixed initiative control of unmanned air vehicle systems: the pitvant rd uav programe. Portuguese Ministry of Defense under the project PITVANT, 2008.

[6] M. E. Dempsey. Eyes of the army - u.s. army roadmap for unmanned aircraft systems 2010-2035. U.S Army, Fort Rucker, 2010.

[7] Paolo Gaudiono, Benjamin Shargel, and Eric Bonabeau. Control of uav swarms: What bugs can teach us. Proceedings of the 2nd AIAA "Unmanned Unlimited" Systems, Technologies and operations - Aerospace, Land, And SEE Conference and Workshop, September 2003.

[8] Paolo Gaudiono, Benjamin Shargel, and Eric Bonabeau. Swarm intelligence: a new c2 paradigm with an application to control of swarms of uavs. ICCRTS Command and Control Research and Technology Symposium, 2003.

[9] Zhihai He. Vision-based uav flight control and obstacle avoidance. Proceedings of the 2006 AIAA Guidance, Navigation, and Control Conference, August 2006.

[10] micro-blox Agenty. Datum transformations of gps positions, July 1999.

[11] Ian Millington and John Funge. Artificial intelligence for games. Morgan Kaumann Publishers, 2009.

[12] H. Moritz. Geodetic reference system 1980. Bulletin géodésique, September 1984.

[13] The national mapping agency of Great Britain. A guide to coordinate systems in great britain. Crown Publishing, December 2010.

[14] Dustin J. Nowak, Ian Price, and Gary B. Lamont. Self organized uav swarms planning optimization for search and destroy using swarmfare simulation. Proceedings of the 2007 Winter Simulation Conference, 2007.

[15] C. Reynolds. Boids - background and update. http://www.red3d.com/cwr/boids/, September 2001.

[16] Marc D. Richards, Darell Whitley, and J. Ross Beveridge. Evolving cooperative strategies for uav teams. Department of Computer Science, Colorado State University, June 2005.

[17] R. Storm and T. Benson. Learning to fly: The wright brothers' adventure. National Aeronautics and Space Administration, 2002.

[18] Another Flight Team. Hirrus. http://www.aft.ro/bro.pdf, 2012.