

DEVELOPMENT MATTERS.

READY FOR AN OVERVIEW



Thesis FollowMe

Last revised: 04 June 2015

Author:

Gerard Heshusius

gerard.heshusius@nspyre.nl

Executive/student

Nspyre:

Maarten Fekkers

maarten.fekkers@nspyre.nl

Project Manager

Boudewijn Noordman

Boudewijn.noordman@nspyre.nl

Operational Manager

The Hague University:

Ben Kuiper

b.kuiper@hhs.nl

Teacher Electrical Engineering

T.J. Koreneef

tj.koreneef@hhs.nl

Teacher Mechatronics

Authorized representative:

R.M.E.M. Heijster

DETAILS

Student:	Gerard Heshusius 11007729 Sporbloem 2 2317 LJ Leiden gerard.heshusius@nspyre.nl +31(0)6- 216 782 44
School:	The Hague University Location Delft Rotterdamseweg 137 2628 AL Delft +31(0)15- 260 6200
Coach The Hague University:	Ben Kuiper Teacher Electrical Engineering Rotterdamseweg 137 2628 AL Delft b.kuiper@hhs.nl +31(0)15- 2606322 T.J. Koreneef Teacher Mechatronics 2628 AL Delft Tj.koreneef@hhs.nl +31(0)15- 2606304
Company:	Nspyre Herculesplein 24 3584 AA Utrecht +31(0)88- 8275 000
First coach Nspyre:	Boudewijn Noordman Operational Manager Herculesplein 24 3584 AA Utrecht boudewijn.noordman@nspyre.nl +31(0)88- 8275 000
Second coach Nspyre:	Maarten Fekkers Recruitment Manager Herculesplein 24 3584 AA Utrecht maarten.fekkers@nspyre.nl +31(0)88- 8275 000

SUMMARY

The concept originated from a brainstorm session with the project leader and the executive. Where the idea was to follow a professional cyclist with a drone. However soon thereafter it was realised that the project would not be limited to just following cyclist. Hence the name FollowMe.

The target (that will be followed) location is gotten from the GPS beacon is sent to a server. This server will broadcast all received data, and thus the location, to all connected devices. FollowMe will make a connection to the server and will thus receive the location of the target.

The FollowMe system, which is housed on a laptop, will direct a drone to the GPS beacon. Directing the drone is done with the help of the WI-FI connection with the drone. Through this medium data can be sent and received. The data can contain directions for the drone, position of the drone and the video stream.

By using both locations, an algorithm can determine the actions that must be made by the drone. The determined actions will then direct the drone to follow a target.

The results of this project are as follows:

- The location of the target is available for FollowMe.
- The location of the drone is available for FollowMe.
- The drone can be manually controlled through FollowMe.
- FollowMe can receive and display the video stream from the drone.

FollowMe does not yet have fully autonomous functionality. Achieving this feature will not take a very long time. The estimation lies around a week or two.

TABLE OF CONTENTS

1.	Details	2
	Summary	3
	Table of contents	4
2.	Introduction	7
3.	Company	8
3.1	Nspyre	8
3.2	Applications	9
4.	Project	10
5.	Drone flight laws	11
5.1	Autonomous drone flight	11
5.2	Remotely controlled drone flight	12
5.3	Conclusion	12
6.	AR.drone2.0	13
6.1	Movement	13
6.2	Communication	14
6.3	Navdata	14
6.4	Starting navdata	16
6.5	Video Streaming	16
6.6	AT commands	17
6.7	Control commands	19
6.8	Emergency mode	19
6.9	Hovering mode	19
6.10	Drones 4g capabilities	19
6.11	Conclusion	20
7.	The Wasmote	21
7.1	Wasmote in previous project	21
7.2	Wasmote for FollowMe	21
7.3	Conclusion	22
8.	Server	23
8.1	First concept	23
8.2	Second concept	24
9.	Other similiar products with the intended use	25
10.	FollowMe	26
10.1	FollowMe factory class	27
10.2	Commandport class	29

10.3	Navdataport class	30
10.4	Videoport class	32
10.5	Targetport class	33
10.6	Mainwindow class	33
10.7	UDP_Socket class.....	34
10.8	TCP_Client class.....	35
10.9	DroneController class.....	35
10.9.1	Determine orientation function.....	37
10.9.2	Calculate pitch function	37
10.9.3	Calculate throttle function.....	38
10.9.4	Calculate yaw function	38
10.10	Original FollowMe concept.....	38
11.	Testing.....	40
11.1	Test plan	40
11.2	Execution.....	40
11.2.1	Can the drone be controlled manually?.....	40
11.2.2	Does the drone orientate to the target?.....	40
11.2.3	Does the drone follow the target?.....	41
11.2.4	Does the drone remain on the intended altitude?.....	41
11.2.5	Does the system display the video stream?	41
11.2.6	What happens when one of the GPS signals are lost?.....	41
11.2.7	What happens when FollowMe loses connection to the drone?	41
11.2.8	What happens when the connection between FollowMe and the drone is bad?	42
11.3	Bug list	42
11.3.1	Setting new height and distance	42
11.3.2	Navdata variables.....	42
11.3.3	Closing FollowMe.....	42
11.3.4	Receiving navdata packets	43
11.3.5	Hover mode.....	43
11.3.6	Calculate yaw function	43
11.3.7	Hardcoded variables in VideoPort	43
11.4	Conclusion	43
12.	Conclusion.....	44
13.	Abbreviations	45
14.	Glossary.....	46
15.	Bibliography.....	47
16.	Attachment I flight exemptions	49
17.	Attachment II connected signals and slots.....	50

18.	Attachment III Data containers.....	51
18.1	Navdata.....	51
18.2	TargetData.....	52
18.3	Message.....	53
19.	Attachment IV FFMPEG and SDL code	54
20.	Attachment V Action Plan.....	56
20.1	Details	57
Table of contents		58
20.2	Background	59
20.3	Project	60
20.4	Activities	62
20.5	Boundaries	63
20.6	Products.....	64
20.7	Quality.....	65
20.8	Organisation	66
20.9	Plan	67
20.10	Costs and Benefits	68
20.11	Risks	69
20.12	Attachment I Detailed plan	70

1. INTRODUCTION

The biggest annual sports event in the world is coming to Utrecht. The Grand Depart is the festive start of the French cycling race, the Tour de France (1). In honour of this event Nspyre wanted a project representing themselves during this cycling festival. This together with Maarten Fekkers (he is the second coach and the project leader) undying love for professional cycling resulted in a unique idea: following a cyclist with a drone, while video streaming. Thus the concept FollowMe is born.

One could imagine a third person view of specific cyclists during this great event. Especially during the start of the Depart, where all cyclist are waiting, to prove themselves the best cyclist the world has seen. Or just before the start of the race, where all cyclist begin their journey. FollowMe could capture these kind of moments, in third person.

This idea followed from the following facts (excluding the Tour and the project leaders' passion):

- Nspyre recently purchased three drones where a third party controller could be developed.
- A previous graduation project: Project Cycling, Bicycle system. By Ollivier van den Akker 06-06-2014 (The thesis is not publicly available).

The previous graduation project concerned the Wasmote¹ and was used to create a cyclist measurement system. This device is equipped with a 3G+GPS module and can determine its location and send it through 3G. Since this project produced a working system, which could be used for FollowMe as a GPS beacon.

Although this concept originated for the professional cycling world it is not restricted to this implementation by any means. Just imagine how many sports could benefit from a third person view of an athlete.

This thesis is dedicated to the realisation of the FollowMe concept which was held during the period of February 9th 2015 up to and including June 5th 2015. It will entail the phases that were used to complete the FollowMe project. The phases are:

- Realising the action plan.
- Research.
- Realising FollowMe.
- Making the drone 4G capable.
- Configure/program the Wasmote.
- Testing.

Do note that this thesis assumes a basic understanding of C and C++ programming.

¹ See chapter 6 The Wasmote for more information.

2. COMPANY

This chapter describes the company and the unit I worked for during my graduation period. Do note that these text are copied from the official Nspyre website.

2.1 NSPYRE

Nspyre is the leading specialised IT service provider where technology matters. Through technology we aim to contribute to society and the success of our clients. These clients primarily come from the high-tech segment, industry and the public sector (transport, infrastructure, defence industry, aviation and the space industry). Our added value is most effectively utilised by companies for which operational reliability and innovation are essential. Nspyre is a specialist in developing software and applying technology in critical operational environments.

The name Nspyre actually communicates our main motive: "inspire". We are a high profile company, where technical specialists feel at home and inspire one another. With approximately 600 employees, we are active nationwide with ten specialized units. These units are competence based and an important contribution to our technical strength and development in favour of our customers and professionals. Nspyre also has its own nearshore software excellence centre in Romania. Our regional focus close to our customers and employees is an explicit choice based on our firm belief that entrepreneurial spirit should be stimulated throughout our company. Nspyre is fully independent and is one of the largest and most experienced service providers in its market segment.

Our services cover the entire development process, from consultancy & project management, development and engineering, to management based on a Service Level Agreement. We provide these services on a project basis or through secondment. Nspyre plans to grow in order to expand its services even further. Special areas of attention testing, consultancy, outsourcing and SLAs and application management.

Copied from: <http://nspyre.nl/profiel/profile.htm>

CUSTOMER	KEY MARKETS	SERVICES	SOLUTIONS	COMPETENCE UNITS	TECHNOLOGY GROUPS	STAFF
		▶ Consultancy	▶ Smart Manufacturing	Smart Manufacturing	▶ Manufacturing IT	
CUSTOMER RELATIONS	HIGH TECH	▶ Projects	▶ Big Data	Big Data	▶ Data & Information Management	RECRUITMENT
		▶ Secondment	▶ Model Based Testing	Smart Improvement	▶ Performance Improvement	
SALES	TRAFFIC & INFRA	▶ Maintenance	▶ Model Driven Engineering	Test & Integration	▶ Test & Integration	FINANCE
		▶ Nearshoring	▶ Simulation	Applied Modeling	▶ Scientific Computing	
MARKETING & COMMUNICATION	INDUSTRY	▶ Flex	▶ Systems Engineering	Engineering Services	▶ Model Driven Engineering	ICT
			▶ Mobile Solutions	Applications	▶ Simulation, Visualisation & Training Systems	
BUSINESS DEVELOPMENT	ENERGY & UTILITIES		▶ UX Design	Systems Engineering	▶ Agile	FACILITY
				Industrial Automation	▶ Adaptive! Systems Engineering	
				Embedded Software	▶ Industrial Automation	LEGAL & RISK
				Web & UX	▶ Java	
					▶ .NET	HR
					▶ Android	
					▶ User Experience	QUALITY ASSURANCE

Figure 1 The Nspyre services and organisation (2)

2.2 APPLICATIONS

The unit Applications is the integration specialist in the field of company critical software. With 140 specialists distributed over the locations Eindhoven, Zwolle en Utrecht. We provide the proper conversion of domain specific, technical and business requirements to intelligent solutions. This mostly applies to distributed (sub)systems where the interaction between these systems play an important role. Our software is often custom, scalable, real-time, 24x7 available and/or secure. Our people are multidisciplinary, technology driven software specialists with a “helicopterview” which can quickly adapt to complex domains.

Translated from: <http://nspyre.nl/technologie/applications-cu.htm>



Figure 2 Competence unit Applications symbol (3)

3. PROJECT

On 14th January 2015 Nspyre purchased three AR.Drone 2.0 elite editions from Parrot. So it was obvious to use these drones during the FollowMe project. Figure 3 Original concept FollowMe displays how FollowMe will achieve the one and only requirement of this project: follow a target autonomously while streaming video images.

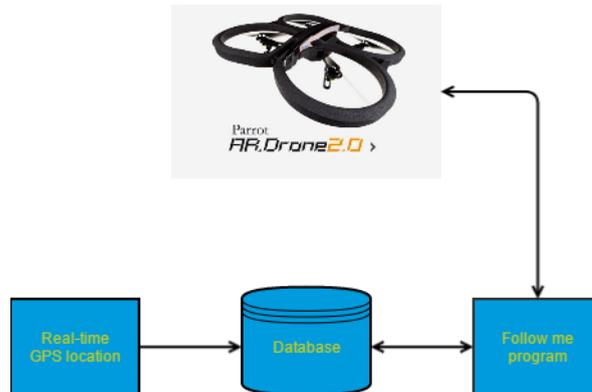


Figure 3 Original concept FollowMe

The real-time GPS location in the figure will be provided by the Wasmote and will communicate its location to the Database. This Database will be the database owned by Nspyre and will communicate the location from the Wasmote to the FollowMe program, which will be housed on a laptop. FollowMe will also request the location of the drone. The program will then, based on both locations, direct the drone autonomously, to the Wasmote. Do note that all arrows displayed in the figure represent IP communication.

Apart from sending a location, the drone will also send other basic information and the video stream. The basic information contains height, whether the drone is flying and other useful information.

However before realising FollowMe a couple of topics need to be researched. The following subjects were researched and documented in the research phase of the project:

- 1 Safety and regulations concerning drone flight.
- 2 The drone concerning its use and how to develop control applications.
- 3 The Wasmote, also concerning its use.

After researching all topics that could influence the design, the system can be realised. Realising the FollowMe system was done with Qt Creator¹ which included the Qt¹ library.

¹ See Glossary for more information.

4. DRONE FLIGHT LAWS

Before designing any kind of product, it was necessary to understand any laws and/or rules concerning drone autonomous flight. These laws and/or rules may result in design decisions to respect these laws/rules.

There are certain regulation that must be upheld while flying a drone in the Netherlands. These regulation can be found at “wet- en regelgeving Regeling modelvliegen” (4). Because drones are fairly new, drones fall under the same category as model planes which are RPAS (Remotely Piloted Aircraft Systems) and the complete category are listed as UAS (Unmanned Aerial Systems).

However no rules could be found concerning autonomous flight, after extensive research for safety regulations. This resulted in a quick e-mail to Inspectie Leefomgeving en Transport (ILT) (5) and resulted in the knowledge that autonomous flight is forbidden.

4.1 AUTONOMOUS DRONE FLIGHT

It was hard to find laws and rules concerning autonomous drone flight. This was because autonomous drone flight is not only forbidden in the Netherlands but in the rest of the world as well. At the moment, there is no official product that flies completely autonomous.

The law in its current form dictates that the pilot is responsible for the aircraft that he controls. Even though big passenger planes (like the Boeing 747) have autonomous flight capabilities, are always supervised by at least two pilots. This is for safety reasons, so that pilots can intervene when the autonomous system fails. But this is also to easily assign responsibility when errors are made (by machine or human alike).

It is hard to assign responsibility when an aircraft is fully autonomous. However the expected solution for the aforementioned problem is that the one that will be followed will be responsible for the drone. There is however, as of this time, no official statement concerning autonomous flights.

However to get a sense of possible rules concerning autonomous drone flight. Research was started about remotely controlled aircrafts.

4.2 REMOTELY CONTROLLED DRONE FLIGHT

There are two regulations that are in conflict with FollowMe, apart from the no flight rule of autonomous drones. The fact that a model plane is not allowed to fly over crowds of people and that professional flight is forbidden. Professional flight means that the flight and/or picture made during the flight are used for professional purpose (6).

To counteract these rules one must request a certain exemption. More information concerning exemptions can be found at 15 Attachment I flight exemptions. Although this information assumes that there is always a pilot, it can still be used to make an estimation for possible precautions in the near future.

4.3 CONCLUSION

The aforementioned attachment also contains the rules concerning drone flight and autonomous flight rules will probably at least contain the applicable rules. However it is expected that the person employing the autonomous drone would be considered the pilot for a legal stand point. Meaning that the one deploying will be responsible.

I also think that the code, like the code for medical purposes, will need to be checked by third party organizations specialized in testing/checking code. And thus will need to comply with not yet determined requirements.

5. AR.DRONE2.0

Before designing a system that can control the drone, it is a good idea to understand the drone. After understanding the rules concerning drone flight. It is important to understand the device that will be controlled. The following chapters covers all that was researched and is applicable to the FollowMe project.

The drone used during this FollowMe project is the AR.Drone 2.0 from Parrot (7). And is used for the very reason that the drone was already available within Nspyre. A lot of the information was gotten from the AR Drone 2.0 Developer guide (8).

5.1 MOVEMENT

All information concerning drone movement is obtained from the AR.Drone 2.0 developer guide (8) chapter 2 AR.Drone 2.0 Overview.

In Figure 4 are all possible movement and the means to achieve these movement are displayed.

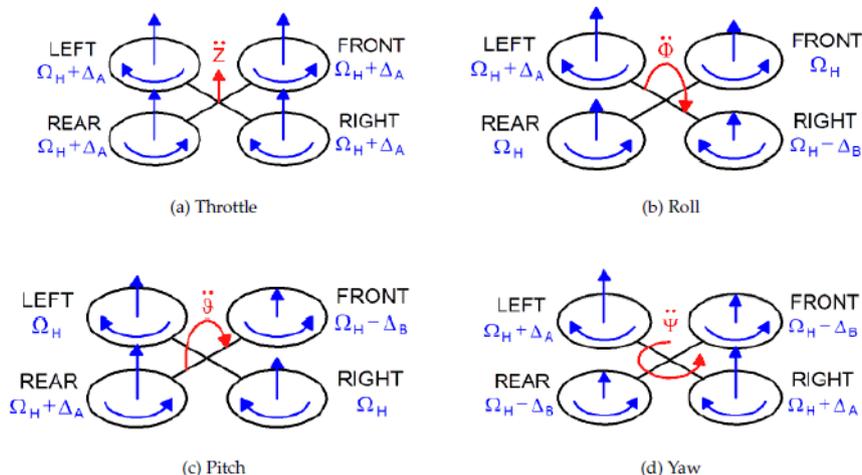


Figure 4 Drone movement (8)

Figure 4 has a lot of symbols and might be confusing. The symbols represent different engine speeds which in turn allow for all available movement of the drone:

- Ω_H represents nominal rotations per minute (RPM). When all motors function with Ω_H , the drone will theoretically hover in place.
- Δ_A is used to represent an increase in speed of the motors relative to nominal speed. The propellers will rotate faster and thus produce more downdraft.
- Δ_B is used to represent a decrease in speed of the motors relative to nominal speed. The propellers will rotate slower than nominal speed but will not stop. Else the drone will become imbalanced.

These three speed manipulations are used to control all of the available movement of the drone. And although Figure 4 displays 4 pictures, representing 4 different kinds of movement, there are actually only three.

- Throttle is the up- and downward movement of the drone. And is achieved by making the motors go an equivalent speed. Δ_A is up and Δ_B is down.
- Roll and Pitch are the same. They represent the movement of the drone in all directions while the drone remains at the same altitude. It is achieved by making the motor achieve Δ_B speed to the intended direction and make the opposite motor go Δ_A speed. The other two motors go with Ω_H speed to maintain altitude.

- Yaw represents the rotation around the drones own axis. This allows the drone to rotate to the left for example. The left rotation is achieved by making the front and rear motors go with Δ_B speed. Then the left and right motors will go with Δ_A speed. The inverse is true for right rotation.

The drone can fly in every direction using the above mentioned motor manipulations. However do note that these manipulations are done by the software in the AR.Drone itself. But the commands to direct the drone make use of the terminology.

5.2 COMMUNICATION

The drone functions as a Wi-Fi host and other application can connect using Wi-Fi. The drone has four available TCP and UDP ports and they all have a different function, see Table 1:

Table 1 List of ports (8)

Name:	Port nr:	Type:	Function:
Navdata port	5554	UDP	The navdata is received from this port.
Video Stream port	5555	TCP	The video stream is received from this port.
Command port	5556	UDP	AT commands are send to this port.
Control port	5559	TCP	Configurations are read from this port.

The drone in its current state communicates using Wi-Fi. A different medium should be chosen to enable long distance communication. A different medium should not impact the design of FollowMe because of the nature of IP communication.

Parrot encourages experimentation with the use of Wireshark. The IP packets send to and from the drone can be sniffed by Wireshark. Although this might not seem special, Wireshark can also interpreted and display the data in a readable fashion. This works out of the box and it is integrated in Wireshark itself.

Do note that there are more ports available on the drone for different functions. However, those weren't used during this FollowMe project and will thus not be documented.

5.3 NAVDATA

Navdata is how the drone communicates back to the controlling system. All information concerning navdata is obtained from the AR.Drone 2.0 developer guide (8) chapter 7.1 Navigation data.

Navdata will be received in the format displayed in Table 2. Every navdata packet will have this setup and the minimal size of this packet is 24 Byte (header + drone state + sequence number + vision flag + checksum block). Note that the navdata is send by little-endian although the positions of the blocks remain the same.

Table 2 Navdata format

Header 0x55667788	Drone state	Sequence number	Vision flag	Option 1			...	Checksum block		
				id	size	data		cks id	size	cks data
32-bit int.	32-bit int.	32-bit int.	32-bit int.	16-bit int.	16-bit int.	16-bit int.	16-bit int.	32-bit int.

The header is the same for every navdata block and can be used to identify the start of the data block. The drone state is displayed in Figure 5 and contain basic information about the drone.

5.4 STARTING NAVDATA

Certain steps must be taken to receive navdata. This process is displayed Figure 7. In this example the AT¹ call for navdata_demo is used. This allows the drone to send some of the most useful information, instead of enabling every option available.

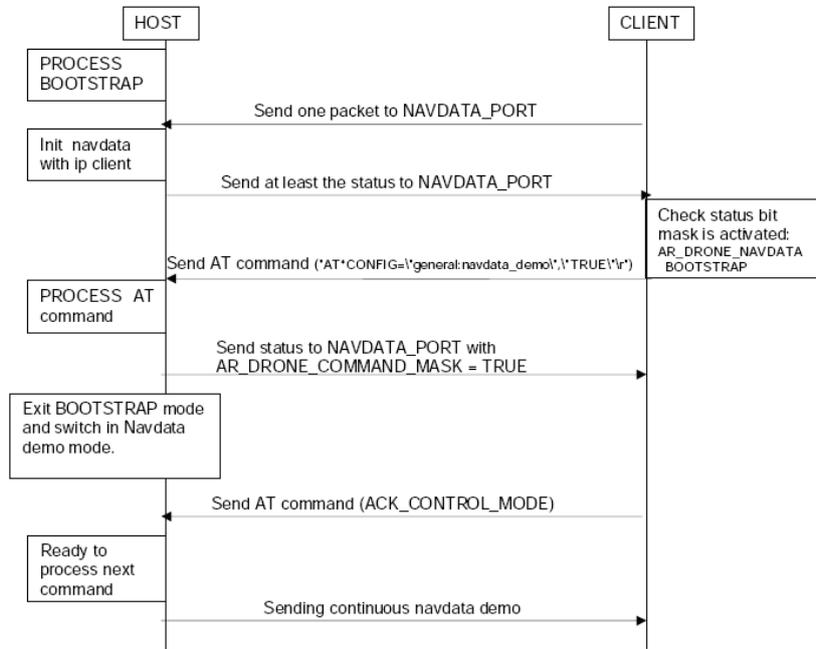


Figure 7 Navdata bootstrap process (8)

The “Send one packet to NAVDATA_PORT” in Figure 7 is not correctly documented in the developer guide. A forum post on a third party website answers the question of what this packet is: `char buffer[] = {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}` (10).

General navdata demo is a request of basic navdata information from the drone. When the drone receives this command it will start sending the most useful navdata options. The ACK_CONTROL_MODE is substituted for `AT+CTRL=sequence_number,5,sequence_number,5,0` (11). This command will acknowledge the configuration to the drone.

To activate specific options within the navdata packets, the programmer need to extend the navdata bootstrap process. After sending the ACK_CONTROL_MODE in Figure 7 Navdata bootstrap process, a new command should be sent. This command contains the specific options and is in the form of (a C string):

`“AT+CONFIG=\general:navdata_options!\,!\number!\r”`. Where the number contains the specific options.

5.5 VIDEO STREAMING

The stream from the drone is a TCP/IP connection. This connection is chosen by Parrot to ensure the frames arrive at the client, and that these arrive in the intended order.

The stream will start when a connection to the stream port is detected. However a very large latency will be experienced because of the nature of a TCP connection. Parrot advises a simple solution: only display the latest frame and discard all the frames before that.

¹ See chapter 5.6 AT commands for more information on these commands.

Although discarding frames seem simple, one should keep in mind that the drone sends I- and P-frames. Where I-frames do not require other video frames to decode and P-frames use data from the previous I-frames to decode (12). This technique saves bandwidth because of the fact that P-frames are smaller than I-frames. So only big changes in the video recorder need to be fully send in the form of I-frames, while small changes can be sent in the smaller version, P-frames.

This means that when a frame is received the receiving system should check if it is an I- or P-frame. In the case of an I-frame, the system should immediately decode and display the most recent I-frame. In the case of a P-frame, the system should decode and display the most recent P-frame. See chapter 7.3.4 Latency reduction mechanism of AR. Drone Developer Guide for more information (8).

The AR.Drone 2.0 uses codecs that are commonly used and there are open source codecs available for this stream. One used by Parrot is FFMPEG¹.

5.6 AT COMMANDS

All information concerning the AT commands (12) were obtained from the AR.Drone2.0 developer guide (8) chapter 6 AT Commands.

In Unix-like operating systems, which is housed in the drone, the AT command is used to schedule commands to be executed. These commands are executed once at a particular time in the near future. (12)

The AT commands have a specific syntax, for example in a C string: `"AT*PCMD=%d,%d,%d,%d,%d,%d<CR>"`. These commands are constructed as a strings and send to the drone. The first integer is always the sequence number and `<CR>` stands for carriage return. The carriage return command is used to set the cursor at the beginning of the line of text. And in this case it will set the reading cursor at the beginning of the command buffer, thus effectively overwriting every old command with a new one. (13)

The roll, pitch, throttle and yaw arguments are send as integers, which may seem weird since they are controlled by floating point variables [-1..1]. However Parrot advices not to convert floats to string, because of accuracy loss. However IEEE-754 floats are stored as a 32 bit word, which is the same size as an integer. So the programmer sends his float value while pretending it is an integer. The drone can translate these integers back to floats using a union. Thus Parrot recommends to use of a union to achieve this, see the AR.Drone2.0 developer guide (8) chapter 6.3 Floating-point parameters for more information.

A complete list of AT commands is listed in Table 3. This list contains the full name of the command, its arguments and a small description. Also note that every argument is a %d or %s like the afore mentioned example. Do note that all parameters are separated by commas. Please refer to the development guide of the AR.Drone 2.0, chapter 6 AT Commands for more information.

¹ See Glossary for more information.

Table 3 List of AT commands (8)

Command:	Arguments:	Description:
AT*REF	Sequence number, value.	Controls the basic behaviour of the drone. The value argument controls take-of/landing and emergency stop/reset.
AT*PCMD	Sequence number, flag, roll, pitch, throttle and yaw	The flag enables progressive commands ¹ and the selection of hover mode ² . The other four control the movement of the drone.
AT*PSMD_MAG	Sequence number, flag, roll, pitch, throttle, yaw, psi, psi accuracy.	The same as AT*PCMD except for the psi and psi accuracy, which allow for absolute control flight.
AT*FTRIM	Sequence number.	Sets the reference for the horizontal plane. The drone must be on the ground before using this command and may not be used during flight.
AT*CALIB	Sequence number.	Calibrates the magnetometer. This command may only be sent when the drone is flying.
AT*CONFIG	Sequence number, key, value.	Sets a configuration inside the drone. The key is the name of the configuration and the value is as the name implies. All arguments except for the sequence number are strings.
AT*CONFIG_IDS	Sequence number, session, user, application ids.	The identifiers for the next AT*CONFIG and must be sent before every AT*CONFIG when using multiple user settings. All arguments except for the sequence number are strings.
AT*COMWDG	Sequence number.	Resets the communication watchdog. Must be sent after 50ms inactivity of the client.

Do note that FollowMe does not use multiple configuration user settings. This means that when a configuration needed to be changed, a single AT*config command could be sent.

The sequence number should be taken care of by the client. The drone will not accept a command less than its current internal sequence number. A command with sequence number 1 can be sent to reset the internal command counter of the drone.

A single UDP packet can contain more than one AT command, by stringing the commands in a single string and wrapping this string in a UDP packet. However the drone has a buffer limit of 1024 characters internally. Which means that every UDP packets must not contain more than 1024 AT command characters. Although the drone should reject UDP packets that are too big, Parrot advises to not send packets that contain too much characters.

¹ Progressive commands allow the drone to fly roll+yaw based turns. Which can be used during racing games for example.

² When hover mode is activated, the drone will hover in place and disregard every move command until the mode is deactivated.

5.7 CONTROL COMMANDS

These commands are sent using the AT*CONFIG command and allows for writing specific configurations to the drone. The format for these commands are 'AT*CONFIG="name", "value"\r' as in Figure 7: 'AT*CONFIG="general:navdata_demo", "TRUE"\r'.

There are multiple elements when using the AT*CONFIG command. In Table 4 they are all listed:

Table 4 List of AT*CONFIG names (8)

Element:	Function:
GENERAL	Read/write general configurations
CONTROL	Read/write configurations concerning control of the drone. Think of maximums and minimums.
NETWORK	Read/write network configurations.
PIC	Read/write nav-board configurations like the ultrasound.
VIDEO	Read/write configurations for the captured video.
LEDS	Allow a pattern to be played by the LEDS. These patterns are determined by Parrot.
DETECT	Read/write detection configurations mostly used by games.
USERBOX	Instructs the drone to save flights and recordings locally.
GPS	This data is used for tagging and userbox recording.
CUSTOM	Multiconfig support.

AT*CONFIG_IDS is used when using multiple users on the drone. However FollowMe will not be using this functionality and will thus not be further explained.

The most important command for FollowMe is the GENERAL element. This one is used to start the navdata bootstrap process and request certain options within the navdata.

5.8 EMERGENCY MODE

When AT*REF is send with emergency stop or when the drone is upside down, it will go in to emergency mode. When this occurs all motors stop and the system goes into an emergency mode, to prevent unpredictable behaviour. This means that when the drone is flying, it will fall out of the sky. The drone recovers from this emergency mode after it is horizontal on the ground. And if not, an AT*REF should be sent with emergency bit on 0.

5.9 HOVERING MODE

When the drone has no commands to execute, the controlling system should send an "AT*PCMD" command with the hover mode selected. The drone will then enter hover mode and in this mode the drone will use its sonar and camera on the bottom to hover in place. In this mode it will even "fight" the wind to remain in place.

However when flying inside Nspyre in Utrecht, the drone drifts of to a seemingly random direction. This is because of the camera on the bottom. The carpet within Nspyre confuses it, probably because the carpet is too monotonous. The drone functions as advertised on different surfaces like hovering above the box it shipped in.

5.10 DRONES 4G CAPABILITIES

The dream for this FollowMe project was being able to direct the drone from great distances. To achieve this the drone would be equipped with a 4G dongle. The laptop however would be required to be equipped with a 4G antenna similar to a team from Alcatel Lucent (14). This antenna is necessary because the client, that controls the drone, needs to be directly connected to the drone.

After some research it was concluded that it is a nice idea but not as easily achievable as first thought. The YouTube pages description found in (14) suggest that the project was a cooperation between Alcatel-Lucent Bell Labs Acceleration Program and Parrot R&D. And further research have not resulted in any publicly available 4G add-ons for the drone. This would mean that the add-on needed to be requested from Parrot and make the FollowMe project dependable on a third party. Since the 4G communication is not very important for the main concept: Following a target, the choice was made, along with the project leader, to postpone any further research and realisation of 4G capabilities.

5.11 CONCLUSION

The drone is quite universal and can be used for prototyping the project. However the FollowMe system will be tailored to this drone specifically, because there are no universal rules that dictate how one should control a drone.

The drone also allows more advanced functions. This is done through the navdata options. These can contain all the data from almost all the sensors. Which could allow for more advanced and accurate localization. However these options will not be used because of the limited experience and knowledge of the executive.

6. THE WASPMOTE

The Wasmote (15) is developed by Libelium and is a low energy consuming, modular development board. And it is an Arduino like system, in the sense that it is a development board for easy development. It houses an ATmega1281 and multiple “plug and play” interfaces for Wi-Fi, GSM, GPS, Bluetooth, a wide range of sensors, etc. The IDE (integrated development environment) provide by Libelium houses classes to approach said hardware and more.

6.1 WASPMOTE IN PREVIOUS PROJECT

The Wasmote was used in the FollowMe project for a very simple reason; it was used in a project before this one. Which was the afore mentioned graduation project: Project Cycling, Bicycle system. It was used to read sensors and send the data to a server for later processing. The data was needed to be sent to a server owned by Nspyre. Those sensors were found within a bicycle of a professional cyclist. Thus the device was required to have all following mentioned capabilities and needed to be light as well.

- Determine the location.
- Measures speed.
- Measures hart rate.
- Measure cadence.
- Send the gathered data to the Nspyre server.

In theory the FollowMe system could be an extension of the measuring device, if the Wasmote was used.

6.2 WASPMOTE FOR FOLLOWME

The Wasmote will have a function within FollowMe as well. It will service as a GPS beacon and its functionality will be very simple.

- 1 Make a persistent TCP connection with the server.
- 2 Collect GPS data.
- 3 Send data to server.
- 4 Busy wait¹ for one second and repeat after that.

The previous project was modified to achieve these functions. Since most functionalities were already present within that project. Functions that were added:

- Setting APN settings through the function call setAPN. In the previous version the APN settings were set in the Wasp3G header file. They were defines hardcoded into the 3G library from the IDE.
- Automatic PIN login. In the previous version the system assumed a PIN disabled data card.
- Setting and creating a TCP/IP connection. In the previous version this connection was a UDP/IP.
- The ability to read the sensors were disabled.

A big disadvantage of the Wasmote is that it cannot sleep for short durations of time. When it is directed to sleep it will kill all connections with any clients/server, ISP (Internet Service Provider) network and GPS network. This means that when the Wasmote wakes up, it needs to re-establish connection to the 3G and GPS networks. Rendering the sleep function completely useless for the FollowMe project, since the sleep function is needed for one second.

¹ It is a technique in which a process repeatedly checks to see if a condition is true (55). In this case, it will constantly check when the second is over.

6.3 CONCLUSION

The Wasmote is a convenient and useful development board for prototyping. It is relatively easy to develop a program that would satisfy almost any need of a project. However the Wasmote will always lose from a dedicated system in the form of battery life and speed. Which is always the cost of Arduino like systems. For the FollowMe project however, the Wasmote will suffice.

7. SERVER

A server was used in the original idea of FollowMe. This server would function as a hatch, broadcasting all data received to connected devices. See Figure 3 Original concept FollowMe for more information.

7.1 FIRST CONCEPT

In order to prevent a lot of server programming and letting the program slip to much from an electrical engineering project (which was one of the concerns of the coach from The Hague University). A research was started concerning servers and how to set them up, approach them, manage them, etc.

After meeting with Erwin Neyt, who has some experience with servers and would help the research concerning server programming, we quickly concluded that the server is not necessary. At least not in this stage of the FollowMe project. In this stage the server would only function as a buffer, FollowMe (for now) is only interested in the last known location.

This functionality can easily be achieved by FollowMe itself. Which means that the Waspote will communicate to FollowMe directly. However during design it is accounted for that the location could be obtained through other means, like a server for instance. The basic concept is displayed in Figure 8.

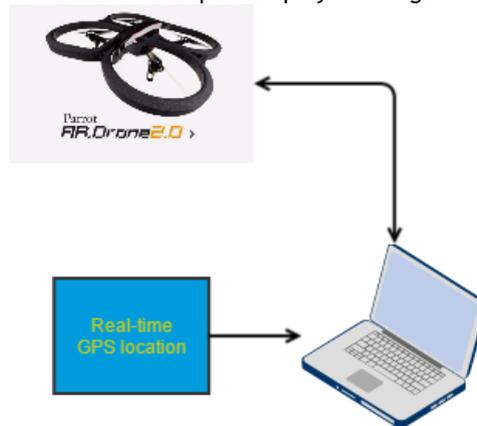


Figure 8 Basic concept FollowMe without server

7.2 SECOND CONCEPT

During the last couple of weeks, when FollowMe was ready for its first tests, a problem arose. The IP addresses assigned to the 3G data SIM cards are private i.e. they begin with a 10 number. This means that the addresses were part of a subnet from the ISP and is thus not publicly available. This resulted in the fact that the Wasmote cannot directly communicate with the laptop. This is in hindsight quite logical, as this prevents a lot of security issues for internet providers. This technique is also more efficient with the limited amount of IPv4 addresses.

So it was back to the original concept to counteract this issue, a server between the Wasmote and FollowMe as described in chapter 3 Project. A specialist in Romania programmed a simple program on the server. He also made the request to map the internal IP to the external one. Which basically means that the program can be approached from outside of the Nspyre network. The only service the server would provide for FollowMe is that there is a public IP address. The Wasmote and FollowMe could then connect to this address using TCP.

As stated in 7.1 First concept, the program functions as a hatch. It has a listening port, which is 10001. And when a connection is made to this port, a new port is assigned to the connection. When any data is received by one of the connections, it will be broadcasted to all other connections. This was done to keep the server program fast and simple. This does have the disadvantage of having bad scalability. But since this is a proof of concept project it has no real consequences for this specific project.

8. OTHER SIMILAR PRODUCTS WITH THE INTENDED USE

There are other products that are similar to the FollowMe concept. This means that the project is quite possible as it was already done. The four contenders of this idea are:



Figure 9 AirDog (16)



Figure 10 HEXO+ (17)



Figure 11 IRIS (18)



Figure 12 Lily camera (19)

However notable of these products (except for Lily) is that they are quite expensive and are not equipped with a camera. The camera is to be bought separately and can be the more popular GoPro camera. Which may be considered as a disadvantage compared to FollowMe.

The Lily camera mirrors this project the most. This drone does anything that is in the scope of the FollowMe project. Which means that this project is possible. Whether it is possible with the AR drone remains to be seen.

9. FOLLOWME

After researching all subjects that could influence and shape FollowMe, a design was made. This chapter describes the design and how it functions.

Figure 13 Schematic of FollowMe displays multiple things. It shows the class relations, internal communication, external communication, visual feedback to and the input from the user. In short, this figure displays the layout and all the communication channels.

Do note that the figure does not comply with the UML standard. The composition (black) arrows has the same function as one would expect. The other arrows however do not. These indicate in which direction the classes communicate. The form of the communication is indicated with the colour.

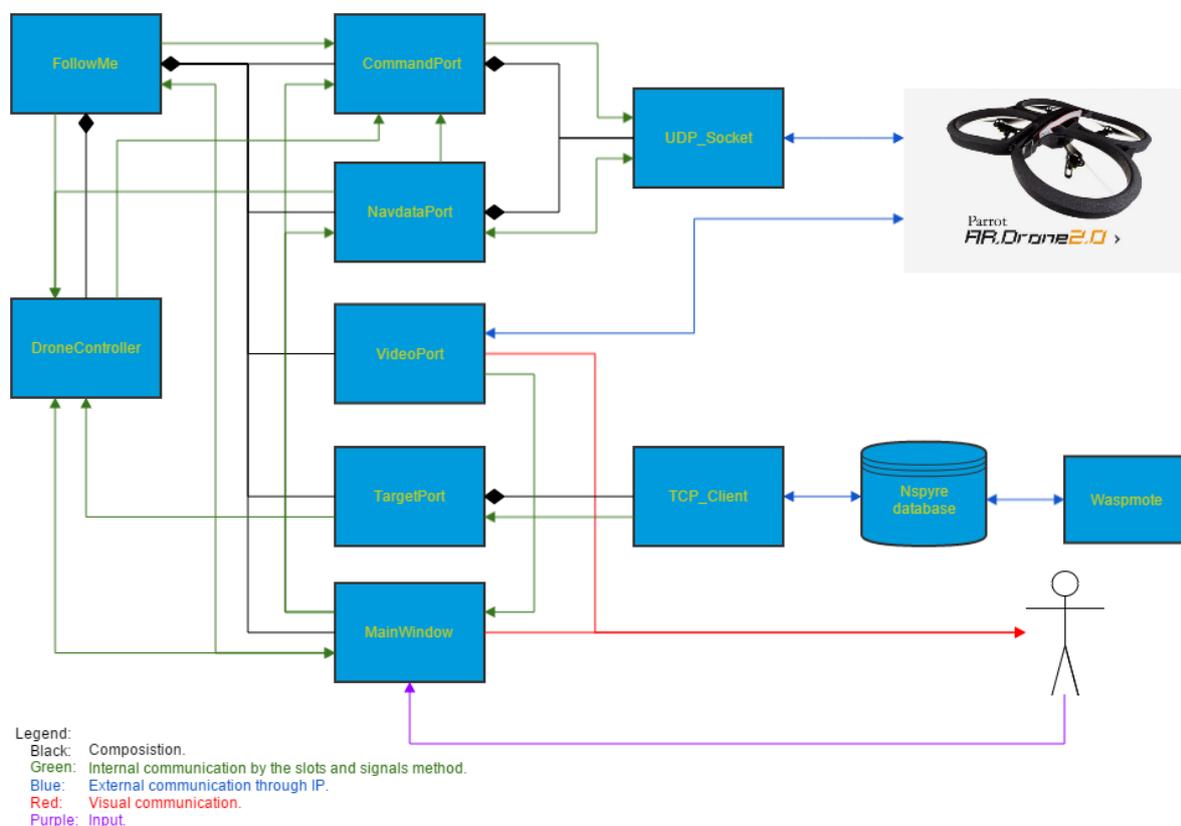


Figure 13 Schematic of FollowMe

The black lines in Figure 13 represent the structure of the program and shows that the program has a single factory class¹ called FollowMe. This class will connect all functionality of almost all the classes together. And as displayed in Figure 13 the DroneController, CommandPort, NavdataPort, VideoPort, TargetPort and MainWindow are part of the factory class. The UDP_Socket and TCP_Socket classes provide a service for specific classes and are thus not directly tied to the FollowMe factory class.

All of the classes displayed in Figure 13 are derived from QObject¹. This is for one simple reason, signals and slots¹. As displayed in Figure 13 by the green lines, a lot of interclass communication is necessary to achieve the best possible readability, maintainability and reusability. For this very reason the method of signals and slots were chosen for interclass communication. The QObject class out of the Qt library offers this functionality out

¹ See Glossary for more information.

of the box. And these signals and slots functionality are comparable to the signals and slots within the boost library.

The other colours in Figure 13 Schematic of FollowMe represent communication to the outside of the FollowMe program. Where the blue lines represent IP communication. The communication to and from the drone are achieved through WI-FI while to and from the database are achieved through 3G.

The red lines represent visual feedback to the user. Which is achieved through the monitor of the system that runs FollowMe. The feedback will entail the stream received through the VideoPort class and messages received from other classes. These messages can be errors or information about the current state of the program. The program will tell the user if autonomy is disabled and how to enable it for example.

The last line, which is the purple one, represents the input from the user. This is achieved through the keyboard of the system that runs FollowMe. See Table 5 Key shortcuts for the complete key list to control FollowMe and the drone.

Table 5 Key shortcuts

Key shortcut:	Description:
Up key	Will tell the drone to take off.
Down key	Will tell the drone to land.
W	Will tell the drone to fly forwards.
A	Will tell the drone to fly leftwards.
S	Will tell the drone to fly backwards.
D	Will tell the drone to fly rightwards.
I	Will tell the drone to fly upwards.
J	Will tell the drone to spin leftward.
K	Will tell the drone to fly downwards.
L	Will tell the drone to spin rightward.
Space	Will enable or disable autonomic functionality.
F1	Will start/restart the stream.
F2	Will start/restart the navdata initialization.

9.1 FOLLOWME FACTORY CLASS

Figure 14 FollowMe factory class displays the class with all its variables and functions. Do note that Figure 14 up to and including Figure 25 are similar to UML class diagrams, but differ from them in the form of using signals and slots. Also the variables given with functions are inverted from the UML standard. Meaning that the type is displayed first, followed by the name of the variable.

FollowMe : QObject
- commandPort : CommandPort - navdataPort : NavdataPort - videoPort : VideoPort - targetPort : TargetPort - droneController : DroneController - videoPortThread : QThread* - droneControllerThread : QThread* - window : MainWindow - timer : QTimer
+ FollowMe(const QString&droneIpAddress, quint16 droneCommandPortNr, quint16 localCommandPortNr, quint16 droneNavdataPortNr, quint16 localNavdataPortNr, const QString& serverIpAddress, quint16 serverPortNr, QObject* parent_FollowMe) : void + run() : void - Q_DISABLE_COPY(FollowMe)
signals: + writeConsole(const Message& msg) : void
slots: - enableStream() : void - stop() : void

Figure 14 FollowMe factory class

The service off this factory class is fairly simple. Keeping all other classes together while providing a simple interface to the “outside world”. All the classes’ signals and slots are connected when the constructor is called of this class. Which is why the programmer needs to give a lot of variables when calling the constructor. These variables will then be passed to the constructors of the CommandPort, NavdataPort and TargetPort. These will then set the given IP address and ports. This is displayed in Figure 15 sequence diagram for the constructors. Do note that this figure only displays the constructors. Also the VideoPort has its variables hardcoded in the header file. It is on the to do list to change that.

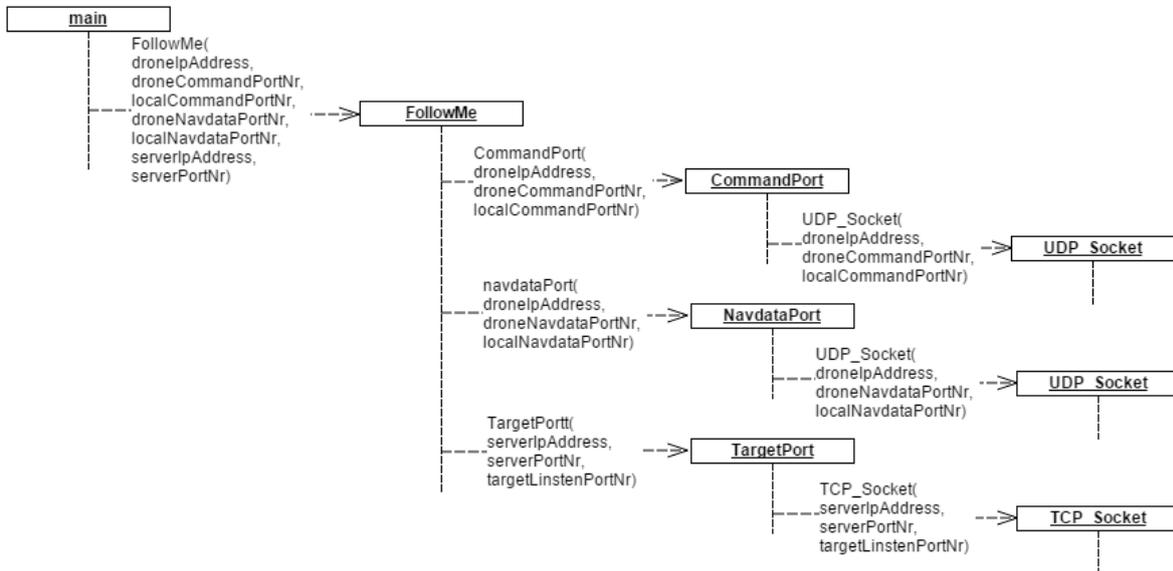


Figure 15 sequence diagram for the constructors

The run function will start all of the FollowMe program when called. Here two threads will be started: the MainWindow and the DroneController. The VideoPort will start when the F1 key is pressed¹. All other classes run in the main thread and are constructed in the FollowMe constructor.

Q_DISABLE_COPY(class name) is a function from Qt that will tell the compiler that there will be no copy constructor and no = operator overload. This function is used in all classes, within FollowMe, that derive from QObject. This will prevent multiple pointers pointing to a single socket or thread. The data containers however do have their copy constructor and = operator overload enabled² and are appropriately defined.

Timer hosts a QTimer³ class and this class provides a time out signal every 30 milliseconds. This signal is connected to the CommandPort and the DroneController. By doing this the program will send commands to the drone through the CommandPort, and calculate new commands through the DroneController every 30 milliseconds.

¹ See Attachment II connected signals and slots for all of the connections.

² See Attachment III Data containers for all the custom data containers used by FollowMe.

³ See Glossary for more information.

9.2 COMMANDPORT CLASS

The CommandPort class delivers one service to the rest of FollowMe: sending AT commands to the drone. In Figure 16 CommandPort class is displayed and in the variable section reside the AT_Command. This class is displayed in Figure 17 AT_Command class and explained below that.

CommandPort : QObject
- socket : UDP_Socket* - at_command : AT_Command - queue : QByteArray
+ CommandPort(const QString& droneIpAdress, quint16 dronePortNr, QObject *parent_CommandPort = 0) : void - addToQueue(const QByteArray& data) : void - Q_DISABLE_COPY(CommandPort)
signals: + writeQueue(const QByteArray& data) : void
slots: + takeOff() : void + land() : void + emergency() : void + fly(bool hoverMode_not, float roll, float pitch, float throttle, float yaw) : void + setFlatTrims() : void + calibrateMagnetometer(quint32 deviceId) : void + setConfig(const QString& optionName, const QString& optionValue) : void + resetCommunicationWatchdog() : void + ack_control_mode() : void + sendQueue() : void

Figure 16 CommandPort class

This class only functions when one of the slots are called. The slots takeOff up to and including ack_control_mode are called from other functions. These functions will construct an AT command string and add this string to the queue with the help of the addToQueue function. This function will check the length of the command to be added and the current length of the buffer. These two values will then be added to each other and when the sum does not exceed the maximum buffer size, the command will be added to the queue. Otherwise the command will be discarded.

The sendQueue function is the odd man out in among the slots. For this function is called solely every 30 milliseconds by the timer. This function will send the current queue to the drone, with the help of the writeQueue signal. Which is connected to the UDP_Socket class. After sending, the function will empty the queue.

The at_command variable is displayed in Figure 17 AT_Command class and this class is used to construct the AT commands in the form of a QByteArray¹.

AT_Command
- SequenceNumber : quint32 = 1 - float_int_union : _float_int_union
+ AT_Command(const AT_Command& other) : AT_Command + operator=(const AT_Command& other) : AT_Command& + takeOff() : QByteArray + land() : QByteArray + emergency() : QByteArray + fly(bool hoverMode_not, float roll, float pitch, float throttle, float yaw) : QByteArray + setFlatTrims() : QByteArray + calibrateMagnetometer(quint32 deviceId) : QByteArray + setConfig(const QString& optionName, const QString& optionValue) : QByteArray + resetCommunicationWatchdog() : QByteArray + ack_control_mode() : QByteArray - resetSequenceNumber() : void

Figure 17 AT_Command class

¹ See Glossary for more information.

The public functions in Figure 17 AT_Command class are called by the slots of the CommandPort with the corresponding name. And this class provides three services for the CommandPort:

- 1 The class keeps track of the sequence number. Every time a new command is constructed it will increment the sequence number.
- 2 The functions construct a QByteArray and fill it with the command and appropriate variables. This byte array will then be returned.
- 3 In case of the fly function call, it will convert the float variables to integers with the use of the float_int_union variable. It will makes new union variables when the function is called, to convert the variables. The float_int_union private is residual from an older version of FollowMe and needs to be removed.

Do note that copies of this class are allowed and the appropriate functions will only copy the SequenceNumber variable. This functionality is not used within FollowMe however.

9.3 NAVDATAPORT CLASS

The NavdataPort class has a single responsibility, managing the navdata stream. This includes starting, restarting and constructing the Navdata class¹ from the raw data. The class and all its functions and variables are displayed in Figure 18.

NavdataPort : QObject
- socket : UDP_Socket - initState : INITSTATE
+ NavdataPort(const QString &droneIpAddress, quint16 dronePortNr, quint16 localPortNr, QObject* parent_NavdataPort = 0) : void - Q_DISABLE_COPY(NavdataPort)
signals: + writeData(const QByteArray& data) : void + ack_control_mode() : void + setConfig(const QString& optionName, const QString& optionValue) : void + navdataUpdated(const Navdata& navdata) : void
slots: - responseNavdataPort(const QByteArray& response) : void + startNavdata() : void

Figure 18 NavdataPort class

When the class is constructed or the slot startNavdata receives a signal from the F2 button. A request will be sent to the navdata port of the drone and the navdata bootstrap process, described in Figure 7 in 5.4 Starting navdata will be started. This process is handled by the responseNavdataPort slot since this is the function that will be called when the UDP_Socket received a datagram. In Figure 19 NavdataPort::responseNavdataPort code the code used, and thus the process of the bootstrap is displayed.

¹ See Attachment III Data containers for all the custom data containers used by FollowMe.

```

void NavdataPort::responseNavdataPort(const QByteArray& response)
{
    const Navdata navdata(response);
    // Continues the navdata stream initiation as displayed in Figure 7.1: Navdata stream initiation found in the
    // ARDrone_Developer_Guide.pdf chapter 7.1.2 Initiating the reception of Navigation data.
    switch (initState) {
    case demo:
        // Send setConfig command.
        emit setConfig("general:navdata_demo", "TRUE");
        // Wait for a new navdata packet. Then check if the drone recieved the command.
        if (navdata.droneState().control_command_ack) {
            // If so, send ack and go to next stage.
            emit ack_control_mode();
            initState = options;
        }
        // If not, resend the setConfig command.
        break;
    case options:
        // Send setConfig command containing the options that needs to be recieved.
        emit setConfig("general:navdata_options", QString::number(1 << GPS_TAG | 1 << ZIMMU3000_TAG | 1 << DEMO_TAG));
        // Wait for a new navdata packet. Then check if the drone recieved the command.
        if (navdata.droneState().control_command_ack) {
            // If so, send ack and go to next stage.
            emit ack_control_mode();
            initState = done;
        }
        // If not, resend the setConfig command.
        break;
    case done:
        // Fall back for if the drone returns to the navdata bootstrap mode for some reason.
        if (navdata.droneState().navdata_bootstrap) {
            initState = demo;
            break;
        }
        // Send the navdata packet to the drone controller.
        emit navdataUpdated(navdata);
        break;
    }
}

```

Figure 19 NavdataPort::responseNavdataPort code

The navdata bootstrap process is handled using a switch case mechanism with an enumerate¹. This code will walk through the bootstrap as described in Figure 7 in chapter 5.4 Starting navdata. With the only exception of the fall back in the case done. This will check if the drone reinitiated the navdata bootstrap process and if so, the system will restart the navdata initialization for FollowMe.

The startNavdata slot is used to initiate the bootstrap and uses the writeData signal to send the char array described in 5.4 Starting navdata.

¹ See Glossary for more information.

9.4 VIDEOPORT CLASS

The VideoPort class displays the video stream received from the drone. This is achieved using two external libraries, FFMPEG¹ and SDL¹. FFMPEG is used to decode the received frames and SDL is used to display the decoded frames.

The class displayed in Figure 20 is largely based on two tutorials (where one is based on the other) found online (20) (21). To get a deeper understanding of the two libraries, a visit to these two sites is highly recommended.

```

class VideoPort : QObject
{
- pFromatCtx : AVFormatContext*
- pCodecCtx : AVCodecContext*
- pFrameRAW : AVFrame*
- pFrameBGR : AVFrame*
- packet : AVPacket
- pBuffer : uint8_t*
- pWindow : SDL_Window*
- pRenderer : SDL_Renderer*
- pTexture : SDL_Texture*
+ VideoPort(QObject* parent_VideoPort = 0) : void
- openStream(const QString& address, unsigned int port) : void
- prepareBuffer() : void
- readData() : void
- Q_DISABLE_COPY(VideoPort)

signals:
+ writeConsole(const Message& error) : void
+ finished() : void

slots:
+ run() : void
}

```

Figure 20 VideoPort class

The code used by FollowMe is displayed in Attachment IV FFMPEG and SDL code, so for a more complete explanation please refer to that chapter. But in short this is what the class does:

- 1 Open stream.
- 2 Get stream information.
- 3 Search for and open corresponding codec.
- 4 Prepare memory for accepting frames.
- 5 Prepare SDL for displaying frames.
- 6 Receive a frame.
- 7 Decode said frame.
- 8 Display said frame using SDL.
- 9 Empty windows message queue. To prevent windows from assuming that the system is in a hanging state.

These steps are made by the openStream, prepareBuffer and readData functions. And they are called by the run slot. This function is a slot because this class will be moved to a different thread by the FollowMe factory class. So when the tread is started, the run slot will be called to begin all functionality. The reason for changing the thread is that all functionality of this class is independent as compared to the rest of FollowMe. It is also a little pre-emptive, since the class does some (relative) heavy-duty processing.

This class makes use of exceptions¹ for catching errors from FFMPEG or SDL. When an exception is caught the class will stop and send the finished signal, which is connected to the quit slot of the thread it is on. That will put the thread in a “stopped but ready to start” state. By pressing the F1 key the thread will be restarted which allow for retrying in hope that the same error does not occur twice.

Every error caught by the class will generate a message using the Message class². This message will be sent to and then displayed to the screen through the MainWindow class.

¹ See Glossary for more information.

² See Attachment III Data containers for all the custom data containers used by FollowMe.

9.5 TARGETPORT CLASS

The TargetPort class may seem like a strange name but the idea behind it was that the data contained information about the target. While the class does not care where the data is coming from, i.e. from a server or the Waspote directly. That being said, Figure 21 TargetPort class may seem like a simple class compared to its peers, and that is because it is.

TargetPort : QObject
- client : TCP_Client*
+ TargetPort(const QString& hostIpAdress, quint16 hostPortNr, quint16 portNr, QObject* parent_TargetPort = 0) : void
- Q_DISABLE_COPY(TargetPort)
signals:
+ responseProcessed(const TargetData& targetData) : void
slots:
- responseUpdated(const QByteArray& response) : void

Figure 21 TargetPort class

The responseUpdated slot is connected to the TCP_Client and will be called when the TCP_Client has data available. This slot will receiving a string from and construct the TargetData¹ class using said string. This class will then be sent to the DroneController class, with the help of the responseProcessed signal.

And that is all what this class does. All complexity concerning string parsing is done within the TargetData constructor.

9.6 MAINWINDOW CLASS

As the name suggests, this class will manage and display a main window. This window provides two services for the FollowMe system.

- 1 Displaying messages from other classes.
- 2 Catching keyboard events and turn those into actions.

These two services require all functions displayed in Figure 22 MainWindow class and this is the only class that is derived from QWidget¹, i.e. QMainWindow¹ is derived from QWidget.

MainWindow : QMainWindow
- window : Ui::MainWindow*
+ MainWindow(QWidget* parent_MainWindow = 0);
+ ~MainWindow();
- keyPressEvent(QKeyEvent* keyEvent) : void
- keyReleaseEvent(QKeyEvent* keyEvent) : void
- Q_DISABLE_COPY(MainWindow)
signals:
+ takeOff() : void
+ land() : void
+ emergency() : void
+ fly(bool hoverMode_not, float roll, float pitch, float throttle, float yaw) : void
+ enableStream() : void
+ startNavdata() : void
+ toggleAutonomy() : void
+ append(const QString& text) : void
slots:
+ writeConsole(const Message& msg) : void

Figure 22 MainWindow class

¹ See Glossary for more information.

The first service is achieved by both the append signal and the writeConsole slot. The append signal is connected to a class called QTextBrowser¹. This is an out of the box text displayer complete with scrolling functionality. The only thing the programmer has to do is appending a string, hence the append signal.

The writeConsole slot receives signals from other classes and requires the Message² class as a variable. This class was chosen over the normal QString class simply for the fact that some messages come every 30 milliseconds. The Message class houses its message in the form of two integers, which is more efficient to compare. With this every Message will be compared with a local static “last Message” variable. This will ensure that a Message is only displayed once. The downside of this system is when there are two Messages sent every 30 milliseconds. Then it will flip between the two and fill the text browser in no time.

The second service is achieved with the derived keyPressEvent and keyReleaseEvent functions. These are functions directly from the QWidget class and can be used to catch keyboard press and release events respectively. The keyPressEvent function will check which key was pressed and will emit an appropriate signal. See Table 5 Key shortcuts for the appropriate keys.

Every key release will generate a signal that will direct the drone to hover mode and thus ignoring all other fly commands. The advantage of this system is that the drone will only fly in a direction in conjecture which key is held. Another advantage is when FollowMe is put in to or out of autonomy mode (by pressing space), it will instantly direct the drone to hover mode. Which will last until a key press event is detected or the autonomy takes over.

9.7 UDP_SOCKET CLASS

The UDP_Socket class is a composition of the CommandPort and NavdataPort class and functions as an interface to the QUdpSocket¹ class. It is displayed in Figure 23 UDP_Socket class and it is the final destination of three of the seven variables from the FollowMe constructor. Do note that there are 2 UDP_Sockets in the system and require thus twice the amount of variables. The destinationAddress is shared however, resulting in 5 variables required to construct both UDP_Sockets.

UDP_Socket : QObject
- socket : QUdpSocket*
- defaultAddress : QHostAddress
- defaultPort : quint16
+ UDP_Socket(const QHostAddress& destinationAddress, quint16 destinationPort, quint16 bindPort, QObject* parent_UDP_Socket = 0) : void
- Q_DISABLE_COPY(UDP_Socket)
signals:
+ responseUpdated(const QByteArray& response) : void
slots:
- readyRead() : void
+ writeData(const QByteArray& data) : void

Figure 23 UDP_Socket class

This is quite a simple class since the UDP/IP protocol is also simple. It will write data when a signal containing a QByteArray is received on the writeData slot. And all data received will call the readyRead slot. This will in turn emit the responseUpdated signal containing the response.

So in short, all it does is passing data through FollowMe and UDP/IP and vice versa.

¹ See Glossary for more information.

² See Attachment III Data containers for all the custom data containers used by FollowMe.

9.8 TCP_CLIENT CLASS

The TCP_Client class shares a lot similarities with the UDP_Socket class. It is even as simple as the UDP_Socket. This because of the fact that the interface of QUdpSocket is almost identical as the QTcpSocket¹. The similarities can be seen in Figure 24 TCP_Client class.

TCP_Client : QObject	
- socket :	QTcpSocket*
+ TCP_Client(const QString& ipAddress, quint16 portNr, QObject* parent_TCP_Client) : void	
- Q_DISABLE_COPY(TCP_Client)	
signals:	
+ responseUpdated(const QByteArray& response) : void	
slots:	
- readyRead() :	void
+ writeData(const char* data, quint64 maxSize) :	void

Figure 24 TCP_Client class

The big differences are that the TCP_Socket has no local variables containing the IP address and port numbers. This is for the reason that when the class is constructed, a socket is made and connected to the given IP address and port number. Which means that there is no need to save these values. This does mean though that it is a persistent TCP connection.

Then there is the final difference, the writeData slot. Where the difference lies in the variables given to the slot. The character pointer is used to iterate through the data and the maxSize variable is used to determine the end of the given data. These are necessary because the function will take parts of the data and transform them in TCP segments. These will then be sent in order to the recipient and the rules of TCP will be upheld. This functionality is done by the QTcpSocket which in turn makes this class rather simple.

The same could be said about the readyRead slot, which will be called when there is data available on the socket. However this slot will be called when the packet is complete and will not require any action on FollowMe part. Apart from passing data to FollowMe from a TCP/IP connection.

So when all is said and done, the class is almost the same as the UDP_Socket except for some function layout and that the data is sent/received through TCP/IP.

9.9 DRONECONTROLLER CLASS

The DroneController class is the heart of the FollowMe system. This is where every other class service will go to. This class will use the location in the TargetData and Navdata to calculate the four values in the fly command: roll, pitch, throttle, and yaw.

As the VideoPort class, this class is moved to a different thread. Though this is, as the VideoPort class, done pre-emptively for its (relatively) heavy-duty calculations. Although this class is more imbedded in the system than the VideoPort class. The signals and slot mechanism allow for seamless communication.

Figure 25 DroneController class and is the biggest class that has an active role in the FollowMe system.

¹ See Glossary for more information.

DroneController : QObject
<pre> - enable : bool - navdata : Navdata - targetData : TargetData - navdata_upToDate : bool - targetData_upToDate : bool - maxDistance : double - maxHeight : double - yawOffsetToNorth : double - navdataMu : QMutex - targetMu : QMutex - variableMu : QMutex </pre>
<pre> + DroneController(QObject* parent_DroneController = 0) : void - determineOrientation(const Navdata& navdataCopy) : bool - calculatePitch(const Navdata& navdataCopy, const TargetData& targetDataCopy, double maxDistanceCopy) : float - calculateThrottle(const Navdata& navdataCopy, double maxHeightCopy) : float - calculateYaw(const Navdata& navdataCopy, const TargetData& targetDataCopy, double yawOffsetToNorthCopy) : float - calculateLenghtOfVector(double lat1, double lon1, double lat2, double lon2) : double - calculateAngleRelativeToNorth(double lat1, double lon1, double lat2, double lon2, double northOffset = 0) : double - Q_DISABLE_COPY(DroneController) </pre>
<pre> signals: + takeOff() : void + land() : void + emergency() : void + fly(bool hoverMode_not, float roll, float pitch, float throttle, float yaw) : void + setFlatTrims() : void + calibrateMagnetometer(quint32 deviceId) : void + setConfig(const QString& optionName, const QString& optionValue) : void + resetCommunicationWatchdog() : void + ack_control_mode() : void + errorString(const Message& msg) : void </pre>
<pre> slots: + navdataUpdated(const Navdata& new_navdata) : void + targetUpdated(const TargetData& new_targetData) : void + controlDrone() : void + toggleAutonomy() : void + setDistance(double distance) : void + setHeight(double height) : void </pre>

Figure 25 DroneController class

The slot controlDrone is what actually, as the name states, controls the drone. This is the final slot connected to the 30 millisecond timeout signal from the QTimer. The other slots can be called asynchronously and is the reason why there are mutexes in this class.

The navdataUpdated and targetUpdated slots update the containers respectively. During the slot the navdataMu or targetMu will be locked to ensure proper copy actions. When reading the respective containers the appropriate mutex will be locked as well. Then there is the final mutex called variableMu. The mutex is locked when the other variables are read or written. For example the toggleAutonomy will invert the enable Boolean. Here the variableMu is used.

The other two slots, setDistance and setHeight, will set the respective variables and also use the variableMu. They have a safety system in place for not being able to set beyond a minimal height or distance. Which are in both cases 2 meters.

The signals are connected as you would expect¹ as they are the same as the other classes.

¹ See Attachment II connected signals and slots for all of the connections.

The controlDrone slot has a number of checks before it will actually control the drone. If one of the checks does not check out, the function will unlock all mutexes¹ and exit the function (with the exception of the TargetData check). The checks are:

- 1 Is enable true? If not, it will mean that autonomic functionality is not desired.
- 2 Is the Navdata up-to-date? If not, it will mean that FollowMe is unaware of the whereabouts of the drone.
- 3 Is the TargetData up-to-date? If not, the system will assume the last known position of the target. This is the only check that does not exit the slot.
- 4 Is the demo option available in the Navdata? If not, the height cannot be properly determined.
- 5 Is the GPS option available in the Navdata? If not, the location cannot be determined.
- 6 Is new GPS data available? If not, the new location cannot be determined.
- 7 Is the drone flying? If not, there is no need to send commands. And making the drone takeOff autonomously is dangerous.

After all checks check out the following 4 functions are called: determineOrientation, calculatePitch, calculateThrottle and calculateYaw. These functions have their own chapter for they require some more in-depth explanation.

Do note that currently the roll remains at 0 and is thus currently not used in autonomous flight. This makes it a three step autonomic function: calculating pitch, throttle and yaw.

9.9.1 Determine orientation function

During this project there was no variable found that would determine where North is or where the camera is facing. Yet these two variables are crucial for correct autonomous functionality. This function will determine the two variable in a 4 step initialization:

- 1 Determine location.
- 2 Fly forward for one second.
- 3 Determine second location.
- 4 Calculate angle between camera and North.

The first three steps will take 15 navdata packets. Since 15 navdata packet are sent every second, it will mean that those steps take 1 second. This is done to get a more accurate location, though this is done pre-emptive and may be unnecessary.

The two locations gotten from the first three initialization steps can be seen as a vector pointing to the facing of the camera in the drone. This vector will have an angle between itself and North. This angle is used in every future calculation for determining where North is.

9.9.2 Calculate pitch function

This function controls the forward speed and is relatively simple. All the system does is calculating the length of a vector between the drone and target. This length is then used to determine how far the drone is behind the maxDistance by dividing the length of the vector by the maxDistance. This will generate a percentage. And this percentage is used for the actual pitch value.

This function can calculate backwards movement as well. If the calculated vector is shorter than the maxDistance, it will direct the drone backward by generating a negative percentage.

¹ See Glossary for more information.

9.9.3 Calculate throttle function

This function controls the height of the drone and is relatively simple as well. It mirrors the calculatePitch function in the form of generating a percentage. Although this percentage is calculated from the height gotten from the navdata demo option and the maxHeight variable. Then again a number will be generated based on the percentage to be used in the fly command.

9.9.4 Calculate yaw function

The location of the drone (latitude and longitude) and the location of target make up a vector. This vector has an angle between itself and imaginary north i.e. 0 degrees. Then the offset, which was calculated with the determineOrientation function, is added.

After that the calculated angle is compared with the angle between the camera facing and North. These two angles will then be compared and the drone will be spinning until the two angles are roughly the same. Directing the drone to face the target with its camera.

Do note that in FollowMes current form this function probably does not work. And has to be changed to the theory described above.

9.10 ORIGINAL FOLLOWME CONCEPT

Figure 26 Original FollowMe concept displays the original idea of FollowMe. And with this and chapter 9 FollowMe, I try to show how different the two actually are and the grow in C++ knowledge that resulted of this. The only class that is mostly the same is the AT_Command, which sole purpose is constructing AT commands.

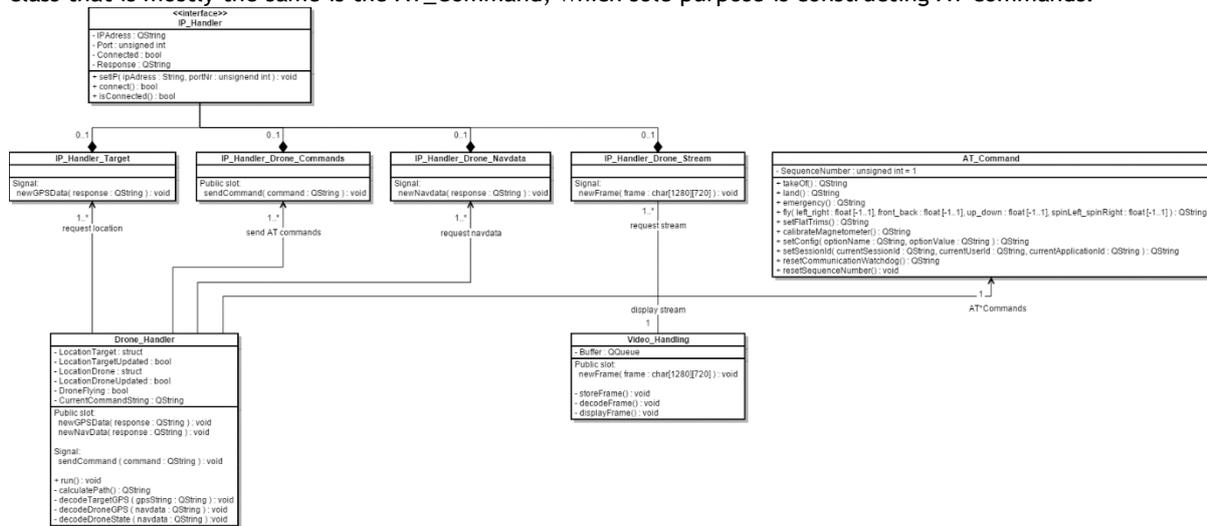


Figure 26 Original FollowMe concept

The first thing that may stand out is the fact that the classes in Figure 26 Original FollowMe concept are a lot smaller than the classes described in chapter 9 FollowMe. The reason for this is that a lot of functionality was not accounted for and most of its functionality were designed/thought out during coding of FollowMe:

- The four IP_Handlers:
These were designed with the idea: The only thing they do is receiving/sending data. But the truth of the matter is that they have more responsibilities now and taking functionality away from the drone controller. Resulting in a more structured system where classes have specific responsibilities.
- Video_Handling:
Because of the external libraries used (FFMPEG and SDL), the IP_Handler_Drone_Stream and Video_Handling were combined to one. Mainly for the fact that FFMPEG handles the TCP connection between FollowMe and the drone.
- AT_Command:
The biggest, if not the only, difference is that the functions now return a QByteArray instead of a QString. This was done because the QUdpSocket accepted QByteArrays as variables for sending.
- Drone_Handle:
The main idea remained the same with this class: Controlling the drone. However a lot of responsibility was taken away from this class so that it could focus on solely directing the drone. A couple of examples:
 - Initiating Navdata:
Initiating the navdata bootstrap process and setting the correct options are now handled by the NavdataPort.
 - Managing AT commands:
Constructing AT commands is not that exciting. Managing them however does require some more logic and was unnecessary for this class. This was a job very suitable for the CommandPort class. The biggest reason for this shift was because other classes needed to send AT commands. Thus it was a logical choice to shift the service.
 - Constructing/parsing received data:
Where the original concept required the Drone_handler to construct/parse Navdata and TargetData, it was a better job for their respected ports classes.

Although Figure 26 Original FollowMe concept houses signals and slots functionality from the Qt library. I was not yet aware of the true strength of this function. This is best illustrated by the shift of managing AT commands responsibility. Where the original concept required multiple AT_Command classes to construct AT commands. Take for instance the not displayed MainWindow class.

The class was necessary to catch keyboard keys and direct the drone accordingly, this was however not thought of in the original concept and needed to be added. This would result in two AT_Command classes and thus two separate sequence counters, which could only be a spell for disaster. Shifting the construction of the commands to one class: CommandPort would mean that there is only one class responsible for the AT commands. And thus having only one sequence counter. It would make respecting the buffer limit a lot easier as well.

So in conclusion: the original design is a little bit similar to the current FollowMe design. The similarity is in the composition of the classes. However all the classes are bigger and more advanced than the classes in the original design.

10. TESTING

There is only one way to determine if realising the FollowMe product was a success. That is by testing it, although FollowMe did not officially reached the testing phase yet. However there was still some time invested in determining a test plan and its expected outcome. See chapter 11. Conclusion for the reason of this delay.

10.1 TEST PLAN

To test this product, a simple question must be answered: does it work? Which followed from the requirement follow a target while streaming video. To answer this question one should look to the requirements of the project. However, since FollowMe was an internal project and has simple requirements, a few (safety) requirements were added during the testing phase. The original boundaries described in 19. Attachment V Action Plan were not exceeded.

The “does it work?” question, which is a system test, will be answered when the following sub questions in Table 6 are answered:

Table 6 Sub questions

Nr.:	Description:
1	Can the drone be controlled manually?
2	Does the drone orientate to the target? I.e. rotate the camera to the target?
3	Does the drone follow the target?
4	Does the drone remain on the intended altitude?
5	Does the system display the video stream?
6	What happens when one of the GPS signals are lost?
7	What happens when FollowMe loses connection to the drone?
8	What happens when the connection between FollowMe and the drone is bad? I.e. a lot of packet loss.

10.2 EXECUTION

Even if a lot of these questions cannot be officially answered yet. A prediction can still be made for the results of these questions.

Because the Wasmote sends its data every second the tester might experience a slight delay. Be sure to keep this into consideration when testing.

When testing the following questions the tester should go to a wide open and unpopulated space outside. This will allow the drone safe movement without the chance of hitting any object and/or person.

10.2.1 Can the drone be controlled manually?

To fully test this question the drone should be started normally. After initialization and connecting to the drone, the tester can then use the keyboard shortcuts displayed in Table 5 Key shortcuts to direct the drone.

Expected result:

The drone should follow the direction that are represented by the pressed key. When no keys are pressed the drone should be in hover mode.

10.2.2 Does the drone orientate to the target?

To fully test this question the drone should be started and finish its initialization. After the Wasmote is started up and has GPS signal. The tester should take the Wasmote and start walking around the drone.

Expected result:

The drone should follow the target by spinning around its own axis.

10.2.3 Does the drone follow the target?

To fully test this question the drone should be started and finish its initialization. After the Wasmote is started up and has a GPS signal. The tester should take the Wasmote and start walking toward and away from the drone.

Expected result:

The drone should move back- and forward respectively. It should maintain a distance of 2 meters.

10.2.4 Does the drone remain on the intended altitude?

To fully test this question the drone should be started and finish its initialization. After the Wasmote is started and has a GPS signal. The tester should find an elevation like a stairs. The tester should keep the drone higher than oneself and try to lower the target, while keeping the drone on its current position.

Expected result:

The drone should not go any lower than 2 meters relative to the platform it is hovering above.

10.2.5 Does the system display the video stream?

To fully test this question a connection must be made with the drone. To start the stream press the F1 key.

Expected result:

The stream should be displayed with a minimal delay. I.e. less than a second.

10.2.6 What happens when one of the GPS signals are lost?

To fully test this question the tester should start the system as normal. Then direct the drone and/or the Wasmote in an enclosed area where the tester expects to lose GPS signal.

Expected result:

- When the Wasmote loses GPS: the drone should fly to the last known location of the target and stay there.
- When the drone loses GPS: the drone should enter hover mode until the GPS signal is regained.

10.2.7 What happens when FollowMe loses connection to the drone?

To fully test this question the tester should start the system as normal. When all is working correctly, the tester should turn off the WI-FI module of the laptop. This will simulate the connection loss.

Expected result:

It should land at its current location. Although this is a functionality from the drone instead of FollowMe, it is still a good thing to be aware of.

10.2.8 What happens when the connection between FollowMe and the drone is bad?

To fully test this question the tester should start the system as normal but manually fly the drone just before its WI-FI boundary. After that the tester should test 3 things:

- 1 Write a configuration command like starting the navdata process.
- 2 Start the video stream.
- 3 Start autonomy.

Expected results:

- 1 FollowMe will send the AT command to activate the navdata process, until this command is acknowledged by the drone. FollowMe will also check the header and checksum in the navdata packet. However when the header or checksum check fails, FollowMe will no longer direct the drone. This means that the drone will keep moving if a move command was sent beforehand.
- 2 The video stream is a TCP protocol. This means that the FollowMe will request the frames until it receives or times out. This will probably result in a very slow video stream.
- 3 FollowMe in its current form stops directing the drone when a navdata packet was failed to construct. This means when the drone had received a move command and a navdata packet had failed to construct: drone will still use its previous move command.

10.3 BUG LIST

Except for the fact that all aforementioned questions need to be answered. There are also a few (yet) unexplainable behaviour for FollowMe.

10.3.1 Setting new height and distance

In its current state the minimal height and following distance are not yet adjustable. However this is easily achieved when the interface, in the MainWindow class, is made for it. The slots already exists in the DroneController class.

10.3.2 Navdata variables

This bug was encountered when FollowMe was ready to be tested. However the tests were stopped in its track when the “is the GPS plugged?” variable in the GPS option remains unchanged. This variable was used as a condition for the autonomous behaviour. If it remained a zero (which it did), the program would do nothing. There is another variable within the GPS option which is the “new data available?” variable. This variable will be a one when new GPS data is available and is now used to determine if the GPS module is plugged in to the drone.

Also at this time, there are 3 types of latitudes and longitudes within the GPS option. Three of them seem to be gibberish from the drone. The only ones that can be trusted are lat0 and lon0, and are thus used in FollowMe.

10.3.3 Closing FollowMe

At the moment there are problems when closing FollowMe. The video stream window cannot be closed by clicking the red cross in the top right corner. And when the main window is closed by the cross, Qt reports the error that pure virtual function is called.

A possible cause might be that the parent class is destroyed after closing the window. This is an educated guess however and still needs some research.

10.3.4 Receiving navdata packets

Sometimes the drone will stop sending navdata packets to FollowMe. Whether this is because the drone thinks the connection is lost or some other reason is still uncertain. This bug is inconsistent and hard to reproduce. However, this bug was only encountered during the start-up process of FollowMe.

An attempt to counteract this bug, the F2 shortcut was created. This shortcut will reinitiate the navdata bootstrap process described in Figure 7 Navdata bootstrap process in chapter 5.3 Navdata.

A possible solution may be a timer that will be used for a time-out mechanism. When the time-out is reached a new request will be sent to the drone to restart the navdata initialization process. Essentially automating the F2 shortcut.

10.3.5 Hover mode

When the controlDrone slot does not control the drone, because one of the checks failed for instance. The drone must be put in hover mode. This is not yet the case and the drone will, when it receives no new command, execute the last known order.

A solution could be to simply send a hover mode command to the drone just before exiting the controlDrone slot.

10.3.6 Calculate yaw function

This function is still all theory. I figured while making this thesis, that in its current form it will probably not work. The chapter 9.9.4 Calculate yaw function describes the theory behind the idea but it is not yet implemented.

10.3.7 Hardcoded variables in VideoPort

Although not a bug necessarily it is sloppy programming. Especially since the other ports have their variables set through their respected constructors.

10.4 CONCLUSION

Although the product is not yet fully tested there are some results. Which were the following questions:

- 1 Can the drone be controlled manually?
Answer: The drone follows the intended direction and hovers when no keys are pressed.
- 2 Does the system display the video stream?
Answer: The stream displays correctly.

Of all the remaining questions the biggest challenge lies with the following question: What happens when the connection between FollowMe and the drone is bad? I am unsure what will happen because the most important communication channels are UDP. This also cannot be changed since it is dictated by the drone.

11. CONCLUSION

The biggest conclusion was already made in the research phase. We are one of the first companies to do this project. This is based on the fact that it is still not condoned by the government and that all other products with the same intended use are new and expensive or even still a kick-starter project.

11.1 COMPLETED

The following things do work in FollowMe:

- All communications work. FollowMe can communicate with the drone in the form of sending commands, receiving navdata and receiving the stream. It can communicate with the Wasmote in the form of receiving data from the server.
- The Wasmote can determine its location and send it to the server. Which will in turn broadcast the data to all connected sockets.
- The drone can be manually controlled using the keyboard.

So in short the only thing that does not work is the autonomous part of the FollowMe project. However I do believe that after some more time (a week or two), the system can be made fully autonomous. I think that the bug list can also be addressed in this period. This is why I requested an extension of two and a half weeks (since that is the amount of time I have from original deadline to the date that I need to defend my thesis).

The reason for not being able to complete the product is that making the code took longer than expected. This is demonstrated in chapter Original FollowMe concept, where all the differences were designed and coded during realisation.

11.2 RECOMENDATIONS

My recommendation to Nspyre is simple: build your own drone. Or find a drone with more access to the hardware. This is not because I think the drone controls are bad, but because of the fact you want most of FollowMes functionality on the drone. This will also counter the question What happens when the connection between FollowMe and the drone is bad?

A drone equipped with some sensors for collision detection might become mandatory for autonomous drones. This is an educated guess however, since there are no official rules concerning autonomic flight. But a drone that is even slightly aware of its surroundings is a lot safer than one that is not. There already is a project concerning the drone and image recognition. If that project and FollowMe might get combined, it will warrant an external system. However this systems sole purpose will be the image recognition.

Another thing to consider though is the law of privacy. There is no automatic blur on faces, so any videos shots that are shot and will be published, need to be treated. The system has also no detection measurements, which means that it assumes a perfect working environment.

If Nspyre wants to compete with the other products the biggest chance for success can be had in collision detection. None of the researched drones has this functionality and will probably become a must for autonomous drones.

Also note that technically speaking a person could be followed by a drone. However this is at this moment not allowed for by law. However as a proof of concept the project will suffice.

12. ABBREVIATIONS

Short	Full	Description
AT	Attention	These commands are scheduled to be executed once, at a particular time in the future (12).
DOA	Design Organisation Approval	The approval of organisations that designed the aircraft, engines, propellers, or other related parts and appliances. These approvals are given by European Aviation Safety Agency (22).
EASA	European Aviation Safety Agency	A European agency that has regulatory and executive tasks in the field of civilian aviation safety (23).
ICAO annex 8	International Civil Aviation Organization	An international organisation concerned with safety in aeronautics and other related things to flight. Annex 8: specialises in Airworthiness of aircrafts (24) (25).
IDE	Integrated Development environment	An IDE is a software tool to develop software for a specific controller. IDEs sometimes houses a compiler, interpreter, or both, such as Eclipse (26).
ISP	Internet Service Provider	Is an organization that provides services for accessing, using, or participating in the internet. (28)
MOA	Maintenance Organisation Approval	The approval of organisations that maintain aircrafts. These approvals are given by European Aviation Safety Agency (22).
Navdata	Navigation data	This is a struct received from the AR.Drone2.0. This struct contains information like status, location, etc. (8)
POA	Production Organisation Approval	The approval that an organisation can produce an aircraft. These approvals can be given by Design Organisation Approved (22).
RPAS	Remotely Piloted Aircraft Systems	RPAS fall under the Unmanned Aerial Systems category (27).
RPM	Revolutions per Minute	It is a measure for the frequency of rotation around a fixed axis. It is used as a measure of rotational speed of a mechanical component (28).
TUG	Tijdelijk Uitzonderlijk Gebruik	A permit which can be requested of the province in the Netherlands. This permit is only given when a class 1 of class 2 permit is issued (29).
UAS	Unmanned Aerial Systems	Sophisticated systems comprising air vehicles, payloads and ground control stations (27).
VFR	Visual Flight Rules	They are a set of regulations under which a pilot operates an aircraft in weather conditions generally clear enough to allow the pilot to see where the aircraft is going (30).

13. GLOSSARY

Term	Description
Cadence	Synonym for pedalling rate (31).
Composition	A strong life cycle dependency between two classes. One class cannot exist without the other. (32)
Enumerate	Is a user-defined type that consists of a set of named integral constants that are known as enumerators. (33)
Euro USC	RPAS safety assurance for national aviation authorities, operators, manufacturers and pilots (24).
Exceptions	They provide a way to react to exceptional circumstances like errors. (34)
Factory class	In class-based programming, the factory method pattern is a creational pattern which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created (37). Within FollowMe however, it is used as a way to describe a class that governs all the other classes.
FFmpeg	A complete, cross-platform solution to record, convert and stream audio and video. (35)
QByteArray	Is a class in the Qt library and is an array of bytes comparable with a character array. However the QByteArray has some distinct advantages. The data is always followed by a '\0' terminator and uses implicit sharing (copy-on-write) to reduce memory usage and avoid copying of data. (36)
QMainWindow	Provides a main application window and can house different QWidgets. (37)
Qt	Is a cross-platform application framework that is widely used for developing application software. It can run on various software and hardware platforms with little or no change in the underlying codebase, while having the power and speed of native applications. (38)
QTextBrowser	Provides a rich text browser with hypertext navigation. (39)
QObject	Is the base class of all Qt objects and is the heart of the Qt object model. The central feature of this class is the signals and slots mechanism which allow for object communication. (40)
QTcpSocket	Is a class in the Qt library and provides a TCP socket. (41)
QTimer	Is a class in the Qt library which provides a repetitive and single-shot timers. (42)
QUdpSocket	Is a class in the Qt library and provides a UDP socket. (43)
QWidget	Is the base class of all user interface objects in the Qt library. (44)
SDL	Simple Direct Media Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. (45)
Signals & Slots	A method used for communication between objects. This mechanism is a central feature of Qt and highly used within FollowMe. See (49) for more information concerning this mechanism.
Verordening 216/2008	European rules for safety in aeronautics (47).

14. BIBLIOGRAPHY

1. **tourdefrancerecht.com**. The tour in Utrecht for a week. *tourdefrancerecht.com*. [Online] 2015. <http://www.tourdefrancerecht.com/en/grand-depart>.
2. **Nspyre**. About Nspyre. *nspyre.nl*. [Online] <http://nspyre.nl/profiel/profile.htm>.
3. —. Competentie unit Applications. *Nspyre.nl*. [Online] <http://nspyre.nl/technologie/applications-cu.htm>.
4. **Haegen, M.H. Schults van**. Regeling modelvliegen. *overheid.nl*. [Online] 16 January 2015. http://wetten.overheid.nl/BWBR0019147/geldigheidsdatum_16-01-2015.
5. **ILENT**. ILT en Luchtvaart. *ilent.nl*. [Online] Inspectie Leefomgeving en Transport Ministerie van Infrastructuur en Milieu. http://www.ilent.nl/onderwerpen/transport/luchtvaart/ilt_en_luchtvaart/.
6. **drones.nl**. Wetgeving in Holland for drones. *drones.nl*. [Online] 2014. <http://www.drones.nl/wetgeving/>.
7. **Parrot**. Home website of the AR.Drone 2.0. *Parrot AR.Drone 2.0*. [Online] 2015. <http://ardrone2.parrot.com/>.
8. —. AR.Drone SDK. *ARDRONE open API platform*. [Online] 11 December 2012. <https://projects.ardrone.org/>.
9. —. ARDrone_SDK_2_0_1\ARdroneLib\Soft\Common\config.h line 219 t/m 299. *projects.ardrone.org*. [Online] <https://projects.ardrone.org>.
10. **Herrendoerfer, Dirk**. Forum containing the question and answer concerning initiating navdata communication. *projects.ardrone.org*. [Online] 7 September 2012. <https://projects.ardrone.org/boards/1/topics/show/4859>.
11. **Brulez, Nicolas**. What does AT*CTRL=(sequence number),5,0 do? *projects.ardrone.org*. [Online] Parrot, 22 March 2011. <https://projects.ardrone.org/boards/1/topics/show/2364>.
12. **Wikipedia**. at (Unix). *Wikipedia*. [Online] 2015 February 2015. http://en.wikipedia.org/wiki/At_%28Unix%29.
13. —. Carriage return. *Wikipedia*. [Online] 14 November 2015. http://en.wikipedia.org/wiki/Carriage_return.
14. **Parrot AR.Drone**. Alcatel-Lucent Bell Labs & Parrot: AR.Drone with 4G Dongle over 1000 m / 3280 ft record! *Youtube.com*. [Online] 1 October 2012. <https://www.youtube.com/watch?v=HgWU8nXITKA>.
15. **Libelium**. Waspote product page. *Waspote*. [Online] <http://www.libelium.com/products/waspote/>.
16. **Airdog**. Airdog homepage. *Airdog*. [Online] 22 September 2014. <https://www.airdog.com/>.
17. **HEXO+**. HEXO+ homepage. *HEXO+*. [Online] 2014. <http://hexoplus.com/>.
18. **3DR**. IRIS+ product page. *IRIS+*. [Online] 2015. <https://store.3drobotics.com/products/iris>.
19. **Lily**. Lily homepage. *Lily*. [Online] Lily, 11 May 2015. <https://www.lily.camera/>.
20. **Ailab**. 2.1.2 AR.Drone 2.0 Video Decoding: FFMPEG + SDL2.0. *AR.Drone Development*. [Online] 25 July 2012. <http://ardrone-ailab-u-tokyo.blogspot.jp/2012/07/212-ardrone-20-video-decoding-ffmpeg.html>.
21. **Dranger**. How to Write a Video Player in Less Than 1000 Lines. *dranger.com*. [Online] February 2015. <http://dranger.com/ffmpeg/ffmpeg.html>.
22. **Wikipedia**. European Aviation Safety Agency. *Wikipedia*. [Online] 26 March 2015. http://en.wikipedia.org/wiki/European_Aviation_Safety_Agency.
23. **EuroUSC international**. Euro USC homepage. *eurousc*. [Online] eurousc.com.
24. **International Civil Aviation Organization**. ICAO homepage. *ICAO*. [Online] <http://www.icao.int/Pages/default.aspx>.
25. **ICAO**. ANNEX 8 - Airworthiness of Aircraft (Amdt 101). *icao.int*. [Online] 14 November 2013. <http://www.icao.int/safety/ism/ICAO%20Annexes/Annex%208.pdf>.
26. **Wikipedia**. Intergrated development environment. *Wikipedia*. [Online] 12 March 2015. http://en.wikipedia.org/wiki/Integrated_development_environment.
27. **AeroVironment**. Media FAQs, What's the difference between "UAS", "UAV", "RPAS" and "Drone"? *avinc.com*. [Online] AeroVironment, Inc, 30 April 2014. http://www.avinc.com/media_gallery/faqs.
28. **Wikipedia**. Revolutions per minute. *Wikipedia*. [Online] 28 April 2015. http://en.wikipedia.org/wiki/Revolutions_per_minute.
29. **Inspectie Leefomgeving en Transport Ministerie van Infrastructuur en Milieu**. Informatiebulletin lichte onbemande luchtvaartuigen. *www.ilent.nl*. [Online] 8 januari 2015.

- http://www.ilent.nl/Images/Informatiebulletin%20lichte%20onbemande%20luchtvaartuigen%20januari%202015_tcm334-362146.pdf.
30. **Wikipedia**. Visual flight rules. *Wikipedia*. [Online] 28 April 2015. http://en.wikipedia.org/wiki/Visual_flight_rules.
 31. —. Cadence (cycling). *Wikipedia*. [Online] 9 May 2015. http://en.wikipedia.org/wiki/Cadence_%28cycling%29.
 32. **aviade**. UML Class Diagram: Association, Aggregation and Composition. *aviadezra.blogspot.nl*. [Online] 28 May 2009. <http://aviadezra.blogspot.nl/2009/05/uml-association-aggregation-composition.html>.
 33. **Microsoft**. C++ Enumeration Declarations. *msdn.microsoft.com*. [Online] <https://msdn.microsoft.com/en-us/library/2dzy4k6e.aspx>.
 34. **cplusplus.com**. Exceptions. *cplusplus.com*. [Online] <http://www.cplusplus.com/doc/tutorial/exceptions/>.
 35. **FFMPEG**. FFMPEG home website. *Ffmpeg*. [Online] 16 March 2015. ffmpeg.org.
 36. **The Qt Company**. QByteArray Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qbytearray.html#details>.
 37. —. QMainWindow Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qmainwindow.html#details>.
 38. **Wikipedia**. Qt (software). *Wikipedia*. [Online] 8 May 2015. http://en.wikipedia.org/wiki/Qt_%28software%29.
 39. **The Qt Company**. QTextBrowser Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qtextbrowser.html#details>.
 40. —. QObject Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qobject.html#details>.
 41. —. QTcpSocket class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qtcpsocket.html#details>.
 42. —. QTimer class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qtimer.html#details>.
 43. —. QUdpSocket Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qudpsocket.html>.
 44. —. QWidget Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qwidget.html#details>.
 45. **Lantinga, Sam**. About SDL. *libsdl.org*. [Online] <https://www.libsdl.org/index.php>.
 46. **The Qt Company**. Signals & Slots. *doc.qt.io*. [Online] <http://doc.qt.io/qt-4.8/signalsandslots.html>.
 47. **Het Europees Parlement en de Raad**. VERORDENING (EG) Nr. 216/2008. *www.ilent.nl*. [Online] 24 November 2009. http://www.ilent.nl/Images/Basis%20Verordening%20EC%20216%20-%202008_tcm334-328908.pdf.
 48. **Puku0x**. <https://github.com/puku0x/cvdrone>. *github.com*. [Online] 2014. <https://github.com/puku0x/cvdrone/blob/master/src/ardrone/navdata.cpp>.
 49. **Zona Land Education**. Angle Measurement: Degrees, Minutes, Seconds. *zonalandeducation.com*. [Online] <http://zonalandeducation.com/mmts/trigonometryRealms/degMinSec/degMinSec.htm>.
 50. **The Qt Company**. QException Class. *doc.qt.io*. [Online] <http://doc.qt.io/qt-5/qexception.html#details>.
 51. **Nspyre**. Marketing. *nbase.nspyre.nl*. [Online] 12 2013. [nbase/teams/Marketing/Team tools/Bestanden/Templates Nspyre/Powerpoint/Introductieslides nspyre EN.ppx](nbase/teams/Marketing/Team%20tools/Bestanden/Templates%20Nspyre/Powerpoint/Introductieslides%20nspyre%20EN.ppx).

15. ATTACHMENT I FLIGHT EXEMPTIONS

In order to fly commercially one of the two types of flight exemptions must be requested. Those are a class 1 and 2 exemption. Class 1 is the standard exemption and consists of basic regulations that are also found in the flight regulations for model planes (4). A Class 2 exemption can be requested when an exemption is needed on class 1. In more detail (29):

- 1 Class 1, basic regulations:
 - Only fly in uncontrolled airspace.
 - Only fly within the visual line of sight of the controller.
 - Never fly higher than 120m.
 - Never fly further than 500m than the controller.
 - Fly with at least vertical distance off 150m between humans, railways and public roads. With exception of 30 km/h roads in urban areas and roads outside urban areas with a minimal speed of 60 km/h.
 - Always fly with visual flight rules (VFR).
- 2 Class 2, exemptions on class 1. To request this exemption you must have at least:
 - The type UAS has a type certificate. An ICAO annex 8 or verordening 216/2008 is required (25).
 - The designer of the craft is Design Organization Approval (DOA) qualified.
 - The producer of the craft is Production organization Approval (POA) qualified.
 - The system must be maintained by a Maintenance Organisation Approval (MOA) qualified organisation.

Before requesting an exemption, the requester must meet certain conditions:

- 1 Airworthiness of the request, which consist of:
 - Positive recommendation of the Euro USC concerning the airworthiness of the Remotely Piloted Aircraft Systems (RPAS) or;
 - Assessment of the RPAS by the national aviation Authority and;
 - A document containing all technical specifications of the RPAS and;
 - Acceptable evidence that the RPAS does not produce noise disturbance.

Estimated costs: €2500,-.

- 2 Proof of competence of the pilot, which consist of:
 - Proven experience with control over the RPAS type, and;
 - Proven theoretical knowledge, and;
 - Company book with the organisation structure and the overall responsibility, among others.

Estimated costs: €1250,-.

- 3 Insurance of the drone against risks of damage against others.
- 4 Safety management system of the organisation.

When the exemption is given by the government Tijdelijk Uitzonderlijk Gebruik (TUG) permit must be requested to the province.

16. ATTACHMENT II CONNECTED SIGNALS AND SLOTS

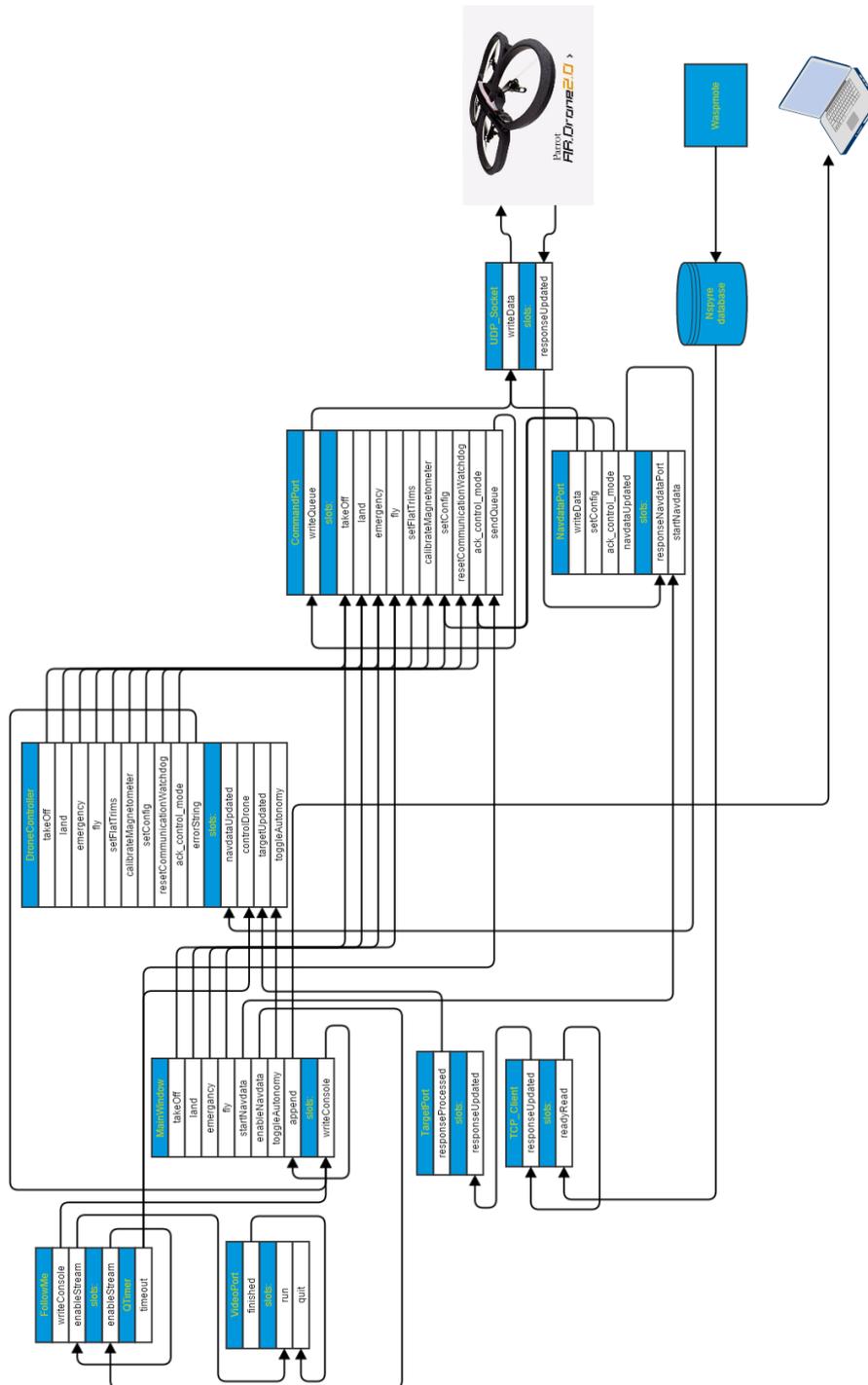


Figure 27 Connected signals and slots

17. ATTACHMENT III DATA CONTAINERS

17.1 NAVDATA

Navdata
<pre> -Updated: UPDATE_LIST -Header: quint32 -DroneState: DRONE_STATE -Sequence: quint32 -Vision_header: quint32 -Demo: NAVDATA_DEMO -Time: TIME -Raw_measures: RAW_MEASURES -Phys_measures: PHYS_MEASURES -Gyros_offsets: GYROS_OFFSETS -Euler_angles: EULER_ANGLES -References: REFERENCES -Trims: TRIMS -Rc_references: RC_REFERENCES -Pwm: PWM -Altitude: ALTITUDE -Vision_raw: VISION_RAW -Vision_of: VISION_OF -Vision: VISION -Vision_perf: VISION_PERF -Trackers_send: TRACKERS_SEND -Vision_detect: VISION_DETECT -Watchdog: WATCHDOG -Adc_data_frame: ADC_DATA_FRAME -Video_stream: VIDEO_STREAM -Games: GAMES -Pressure_raw: PRESSURE_RAW -Magneto: MAGNETO -Wind: WIND -Kalman_pressure: KALMAN_PRESSURE -Hdvideo_stream: HDVIDEO_STREAM -Wifi: WIFI -Zimmu_3000: ZIMMU_3000 -Gps: GPS -Cks: CKS +Navdata(const QByteArray& navdataRaw): void +Navdata(const Navdata& other): void +operator=(const Navdata& other): Navdata& +updated() const: const UPDATE_LIST& +header() const: quint32 +droneState() const: const DRONE_STATE& +sequence() const: quint32 +vision_header() const: quint32 +demo() const: const NAVDATA_DEMO& +time() const: const TIME& +raw_measures() const: const RAW_MEASURES& +phys_measures() const: const PHYS_MEASURES& +gyros_offsets() const: const GYROS_OFFSETS& +euler_angles() const: const EULER_ANGLES& +references() const: const REFERENCES& +trims() const: const TRIMS& +rc_references() const: const RC_REFERENCES& +pwm() const: const PWM& +altitude() const: const ALTITUDE& +vision_raw() const: const VISION_RAW& +vision_of() const: const VISION_OF& +vision() const: const VISION& +vision_perf() const: const VISION_PERF& +trackers_send() const: const TRACKERS_SEND& +vision_detect() const: const VISION_DETECT& +watchdog() const: const WATCHDOG& +adc_data_frame() const: const ADC_DATA_FRAME& +video_stream() const: const VIDEO_STREAM& +games() const: const GAMES& +pressure_raw() const: const PRESSURE_RAW& +magneto() const: const MAGNETO& +wind() const: const WIND& +kalman_pressure() const: const KALMAN_PRESSURE& +hdvideo_stream() const: const HDVIDEO_STREAM& +wifi() const: const WIFI& +zimmu_3000() const: const ZIMMU_3000& +gps() const: const GPS& +cks() const: const CKS& +rawData() const: const QByteArray& +printDroneState() const: void +printGps() const: void -computeCks(const QByteArray& navdataRaw): bool -bitArrayToDroneState(const QBitArray& array): void -toQBit(const quint32& val): QBitArray </pre>

This class is the largest of classes within FollowMe and is based on the code from (48). The class can contain all data that can be possibly received from the drone. Figure 28 Navdata class displays the class and it has a lot of variables and functions and may seem confusing, but every variable has a function to return a constant reference to the variable.

The variable Updated is the only variable not directly gotten from the navdata packet received from the drone. It is a variable that keeps track of which variable is up to data and is trustworthy.

The rest of the variables are custom structures from (48) and will be copied in the constructor. The constructor parses the raw data, which is in the form of a QByteArray, with the use of the tag and size¹ of the option.

The checksum is calculated in the computeCks function. It will also check in the raw data if there is a correct header. When this function returns a true, the programmer can be sure that the rest of the raw data is trustworthy.

The functions toQbit and bitArrayToDroneState are used to convert the drone state integer to a 32 Boolean wide structure. The first will convert the integer value to a QBitArray. The second will set the Boolean structure based on the QBitArray.

Do note that at this moment, FollowMe only makes use of the DEMO and GPS option.

Figure 28 Navdata class

¹ These are the tag and size found in every option. See Figure 6 Navdata option C structure for more information.

17.2 TARGETDATA

This class contains data concerning the target, or in this case the Wasmote. It is constructed from a string received from the server. The sole function of the functions in Figure 29 TargetData class is constructing or returning data.

TargetData	
- Data	: target_data
- RawData	: QByteArray
+ TargetData(const QByteArray& data) : void	
+ TargetData(const TargetData& other) : void	
+ operator=(const TargetData& other) : void	
+ data() const : const target_data&	
+ rawData() const : const QByteArray&	
- copyTag(const QString& tagName, void* tagPointer = nullptr) : bool	
- copyDegrees(const QString& tagName, float* tagPointer) : bool	
- copyTime() : void	
- toDecimalDegree(quint32 degrees = 0, quint32 minutes = 0, quint32 seconds = 0) : float	

Figure 29 TargetData class

The string that will be received will have tags for identifying the type of data and is distinguished with '#' symbols. The tags that are currently used are displayed in Table 7 List of TargetData tags.

Table 7 List of TargetData tags

Tag:	Type:
Bat:	Battery level in whole percentages.
Lat:	Latitude received in the form of: +/-ddmm.mmmmm ¹ .
Lon:	Longitude received in the form of: +/-dddmm.mmmmm ¹ .
Alt:	Altitude.
Spd:	Speed over ground.
Crs:	Course.
Clk:	Time and date received in the form of: ddmmyy.hhmmss ² .

The constructor will simply copy the data in the string to a variable in the target_data struct with the appropriate name. Exceptions of these are the lat, lon and clk tags. Main reason being that the data is formatted and needs extra operations.

Copying data from string to target_data struct is done with the copyTag, copyDegrees and copyTime functions. The copyTag functions tagName is used to determine if the tag is in simple format or in need of extra operations. When the tag is in need of extra operations copyDegrees or copyTime will be called accordingly. The tagName string is also used for searching within the received string.

The tagPointer variable is used to give an address to the destination variable. This makes the function more universal.

The toDecimalDegree function is necessary because of the "strange" format of the lat and lon tag. The format can be converted to a decimal degree value i.e. +50.89 degrees. The idea behind the format is instead of sending floats through the internet, it will send fractions. Degrees are whole degrees while the minutes are 1/60th of a degree. The seconds (while not used in FollowMe) is a 1/3600th of a degree (49).

Saving the raw data was originally an idea for testing purposes, mainly for easy printing. However it may be helpful when the data needs to be saved or sent to another system.

¹ d = degrees, m = minutes.

² d = days, m = months, y = years, h = hours, m = minutes, s = seconds.

17.3 MESSAGE

This class idea originates from the problem that some messages are generated every 30 milliseconds and comparing strings is very inefficient. Figure 30 Message class displays the (simple) solution.

Message : QException
- FirstElement : quint8
- SecondElement : quint8
+ Message(quint8 first_element, quint8 second_element) : void
+ Message(const Message& other) : void
+ operator=(const Message& other) : Message&
+ operator==(const Message& other) const : bool
+ operator!=(const Message& other) const : bool
+ raise() const : void
+ clone() const : Message*
+ what() const noexcept : const char*

Figure 30 Message class

Half of the class idea is not represented in the figure. It makes use of the two variables FirstElement and SecondElement. The second part of the idea is that there is a constant static vector containing other static vectors. Where the first vector (FirstElement) represents the message type and the second vector (SecondElement) the message itself. The second vector contains QStrings containing the message. Because these vectors are static they are shared through every instance of the object.

The raise and clone functions are derived from QException and have a standard implementations found in (50). The raise function will “re-throw” the exception and the clone function will make a new copy of the exception. These functions are used by Qt under the hood to navigate them through different instances.

The what function is also derived from the QException class but is the same as the what function from the normal exception class. In this Message class the what function will return a character pointer to one of the strings in the second vector, using the FirstElement and SecondElement variables.

18. ATTACHMENT IV FFMPEG AND SDL CODE

```

void VideoPort::openStream(const QString& address, unsigned int port)
{
    char errorBuff[1024];
    /// 1.0 Opening the file
    /// 1.2 Open video file.
    QByteArray fileName;
    fileName.append("tcp://" + address + ":" + QString::number(port));

    int openInputResult      = avformat_open_input(&pFormatCtx, fileName.data(), NULL, NULL);
    av_strerror(openInputResult, &errorBuff[0], 1024);
    if (openInputResult      < 0) throw Message(1, 0);

    // 1.3 Retrieve stream information.
    int findStreamInfoResult = avformat_find_stream_info(pFormatCtx, NULL);
    if (findStreamInfoResult < 0) throw Message(1, 1);
    // Dump information about file onto standard error.
    av_dump_format(pFormatCtx, 0, fileName, 0);

    // 1.4. Get a pointer to the codec context for the video stream.
    pCodecCtx      = pFormatCtx->streams[0]->codec;
    AVCodec *pCodec = avcodec_find_decoder(pCodecCtx->codec_id);
    if (pCodec      == NULL) throw Message(1, 2);

    // 1.5.3 Open codec
    int open2Result      = avcodec_open2(pCodecCtx, pCodec, NULL);
    if (open2Result      < 0) throw Message(1, 3);
}

```

Figure 31 VideoPort::openStream

```

void VideoPort::prepareBuffer()
{
    /// 2.0 Storing the Data
    /// 2.1 Allocate an AVFrame structure
    pFrameRAW      = av_frame_alloc();
    if (pFrameRAW  == NULL) throw Message(1, 4);
    pFrameBGR      = av_frame_alloc();
    if (pFrameBGR  == NULL) throw Message(1, 5);

    // 2.2 Determine required buffer size and allocate buffer
    pBuffer        = static_cast<uint8_t*>(av_malloc(avpicture_get_size(PIX_FMT_BGR24, pCodecCtx->width, pCodecCtx->height)));
    if (pBuffer    == NULL) throw Message(1, 6);

    // 2.3 Assign appropriate parts of buffer to image planes in pFrameBGR
    int pictureFillResult = avpicture_fill(reinterpret_cast<AVPicture*>(pFrameBGR), pBuffer, PIX_FMT_BGR24, pCodecCtx->width, pCodecCtx->height);
    if (pictureFillResult < 0) throw Message(1, 7);
}

```

Figure 32 VideoPort::prepareBuffer

```

void VideoPort::readData()
{
    // 3.0 Reading the Data
    // 3.2 Initialize SWS context for software scaling
    struct SwsContext *sws_ctx = sws_getContext(pCodecCtx->width,
                                                pCodecCtx->height,
                                                pCodecCtx->pix_fmt,
                                                pCodecCtx->width,
                                                pCodecCtx->height,
                                                PIX_FMT_BGR24,
                                                SWS_SPLINE,
                                                NULL,
                                                NULL,
                                                NULL);

    if (sws_ctx == NULL) throw Message(1, 8);

    // Initialize window, renderer and texture for SDL.
    pWindow = SDL_CreateWindow("FollowMe", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, pCodecCtx->width, pCodecCtx->height, SDL_WINDOW_SHOWN);
    if (pWindow == NULL) throw Message(1, 9);
    pRenderer = SDL_CreateRenderer(pWindow, -1, SDL_RENDERER_ACCELERATED);
    if (pRenderer == NULL) throw Message(1, 10);
    pTexture = SDL_CreateTexture(pRenderer, SDL_PIXELFORMAT_BGR24, SDL_TEXTUREACCESS_STREAMING, pCodecCtx->width, pCodecCtx->height);
    if (pTexture == NULL) throw Message(1, 11);

    uint8_t *pixels = 0;
    int frameDecoded= 0;
    int pitch = 0;
    int size = pCodecCtx->width * pCodecCtx->height * 3;

    SDL_Event sdlEvent;
    quint32 i = 0;
    bool terminate = false;
    while (!terminate) {
        // Read frame.
        av_read_frame(pFormatCtx, spacket);

        // Decode frame.
        avcodec_decode_video2(pCodecCtx, pFrameRAW, &frameDecoded, spacket);

        if (frameDecoded) {
            // 3.3.1 Decode the frame to YUV for Displaying
            sws_scale(sws_ctx, static_cast<const uint8_t * const *>(pFrameRAW->data), pFrameRAW->linesize, 0, pCodecCtx->height, pFrameBGR->data, pFrameBGR->linesize);

            // 3.3.2 Copy converted YUV to SDL 2.0 texture
            SDL_LockTexture(pTexture, NULL, reinterpret_cast<void*>(pixels), spitch);
            memcpy(pixels, pFrameBGR->data[0], size);

            SDL_UnlockTexture(pTexture);
            SDL_UpdateTexture(pTexture, NULL, pixels, pitch);

            // Refresh the screen
            SDL_RenderClear( pRenderer);
            SDL_RenderCopy( pRenderer, pTexture, NULL, NULL);
            SDL_RenderPresent( pRenderer);

            // saveFrame(pFrameBGR, pCodecCtx->width, pCodecCtx->height, i++);
        }

        // Prevents from windows thinking the app is in a hanging state.
        SDL_PollEvent(&sdlEvent);
        if (sdlEvent.type == SDL_KEYDOWN && sdlEvent.key.keysym.scancode == SDL_SCANCODE_ESCAPE) break;
    }
}

```

Figure 33 VideoPort::readData

19. ATTACHMENT V ACTION PLAN

DEVELOPMENT MATTERS.

READY FOR AN OVERVIEW

ACTION PLAN

FollowMe

Last revised: 15/05/2015

Author:

Gerard Heshusius

gerard.heshusius@nspyre.nl

Executive/student

Nspyre:

Maarten Fekkers

maarten.fekkers@nspyre.nl

Project Manager

Boudewijn Noordman

Boudewijn.noordman@nspyre.nl

Operational Manager

The Hague University:

B. Kuiper

b.kuiper@hhs.nl

Coach The Hague College

19.1 DETAILS

Student:	Gerard Heshusius 11007729 Sporbloem 2 2317 LJ Leiden gerard.heshusius@nspyre.nl +31(0)6- 216 782 44
School:	The Hague University Location Delft Rotterdamseweg 137 2628 AL Delft +31(0)15- 260 6200
Coach The Hague University:	B. Kuiper Teacher Electrical Engineering Rotterdamseweg 137 2628 AL Delft b.kuiper@hhs.nl +31(0)15- 2606322
Company:	Nspyre Herculesplein 24 3584 AA Utrecht +31(0)88- 8275 000
First coach Nspyre:	Boudewijn Noordman Operational Manager Herculesplein 24 3584 AA Utrecht boudewijn.noordman@nspyre.nl +31(0)88- 8275 000
Second coach Nspyre:	Maarten Fekkers Recruitment Manager Herculesplein 24 3584 AA Utrecht maarten.fekkers@nspyre.nl +31(0)88- 8275 000

TABLE OF CONTENTS

12.1	Details	57
12.2	Background	59
12.3	Project	60
12.4	Activities	62
12.5	Boundaries	63
12.6	Products	64
12.7	Quality	65
12.8	Organisation	66
12.9	Plan	67
12.10	Costs and Benefits	68
12.11	Risks	69
12.12	Attachment I Detailed plan	70

19.2 BACKGROUND

The Tour de France is an annual multiple stage bicycle race primarily held in France, while also occasionally making passes through nearby countries^{*1}. On July 4th 2015 the Grand Depart of the Tour de France will take place in Utrecht in front of the head office of Nspyre. Because of the enormous media exposure of the Tour, combined with the fact that within Nspyres core lies innovation, Nspyre will try to implement an innovative project called FollowMe, during the Tour.

Nspyre is primarily a software company. Seconding staff or taking on projects in subjects High Tech, Traffic & Infra, Industry and Energy & Utilities. There are about 630 employees and departments located in Utrecht, Eindhoven and Zwolle^{*2}. The department Applications mid-west 2 will take this project in the form of a graduation project.

This project is initiated by Nspyre and will remain an internal project during its development. The project manager is Maarten Fekkers and the executive is Gerard Heshusius.

The project was brainstormed by the aforementioned and there are currently no written details.

*1 source: http://en.wikipedia.org/wiki/Tour_de_France

*2 source: <http://Nspyre.nl>

19.3 PROJECT

An idea exist of following a cyclist real-time while recording said cyclist. Why not try and achieve this with an autonomous drone. And when all goes well, during the Tour de France.

This idea in its current state will be able to follow an intended target, for instance a cyclist. The drone will be able to identify said target using a real-time GPS location. This location will be generated by a device that is located on the target and will send this location to an Nspyre server. This location will be used to direct the drone to the target and will keep following it, until told otherwise. Captured images from the drone will be streamed to a client. All this combined will result in an autonomous flying recording device that will follow and record a target.

Although the primarily idea was for cyclist it will be no means be limited to cycling. It will be applicable for every sport where a target can carry the GPS signal. Or in a football for instance during soccer.

Figure 34 The FollowMe project shows the current design for this project.

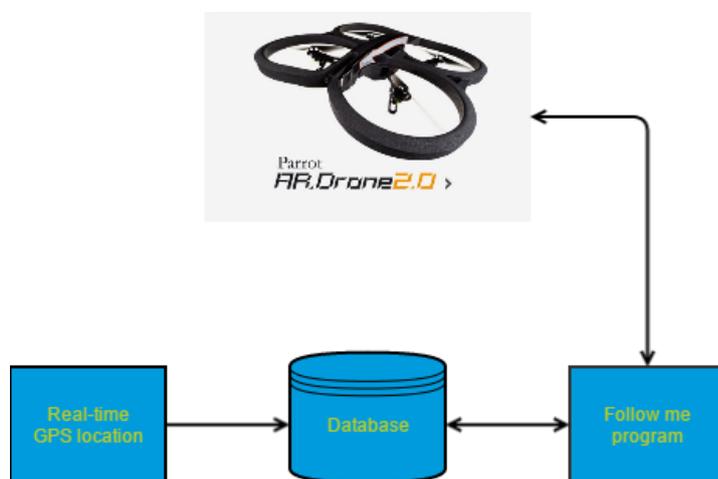


Figure 34 The FollowMe project

As seen in the figure, the FollowMe project will consist of multiple items:

- A device that will be able to determine a real-time GPS location and upload said location to the database. A Wasmote³ will be used to achieve this.
- A database containing the real-time location of the target.
- A device that will translate the real-time location to commands for the drone and receive captured images from the drone. Which is represented as the FollowMe program which will be run on a laptop.
- The Parrot AR.Drone.2.0⁴. This specific drone is used because of its high definition 720p camera and accessibility.

The Parrot drone can only communicate via Wi-Fi in its current state. This however will limit the project immensely. So one of the goals is to achieve 4G communication between the drone and FollowMe.

When all is successful the drone will be able to follow and record a specific target.

At the end of this project there will be a FollowMe prototype which will consist of the Parrot drone with 4G communication capabilities, FollowMe program, a database and a programmed Wasmote. With all this the FollowMe prototype will be able to follow a target and stream the captured images back to the FollowMe program. There will also be documentation in the form of a thesis.

This project will consist of 7 phases:

- 1 Creation of the action plan.
- 2 Research.
- 3 Realisation of the FollowMe program.
- 4 Set up of the drone for 4G communication.
- 5 Programming the Wasmote to relay real-time location to the server.
- 6 Testing the prototype.
- 7 Document the whole project in the form of a thesis.

There is a need for software that is able to:

- Get the real-time location on the Wasmote and send said location to the database.
- Use said location to direct the Parrot drone to specific locations.
- Receive and display the captured images from the drone.
- Communicate between FollowMe and the drone, using 4G.

Nspyre already owns at least one Parrot drone and one Wasmote. They also own a server and the executive already leased a laptop from Nspyre. Nspyre also houses multiple specialist that the executive can address. The software development tools concerning the Wasmote and Parrot drone are free software. Nspyre promised to cover any reasonable potential cost (if any).

This project will be run through the period of 09-02-2015 till 05-06-2015.

*3 for more information, please refer to: <http://www.libelium.com/products/wasmote/>

*4 for more information, please refer to: <http://ardrone2.parrot.com/>

19.4 ACTIVITIES

Certain steps must be taken to steer this project to a successful end:

- 1 An action plan must be formulated and be accepted by Nspyre and school of the executive known as The Hague University.
- 2 Research concerning:
 - a Safety regulations:
 - i Drone flight. To determine if the drone can be set up to fly in public places.
 - ii Wireless communications. To be able to take into account eventual safety regulations concerning 3G/4G, GPS and Wi-Fi.
 - b The Parrot drone must be researched for its capability to communicate with third party devices and other means of communication than Wi-Fi.
 - c The Waspote must be researched concerning:
 - i Determining real-time location using the 3G+GPS module.
 - ii Communicate said location to a server using the 3G+GPS module.
- 3 Creating FollowMe program to:
 - a Setting up a server.
 - b Get real-time location from the server.
 - c Translate the real-time location to commands for the drone.
 - d Communicate said commands to the drone.
 - e Receive and display the captured images from the drone.
- 4 Setting up Parrot drone to be able to communicate using 4G.
- 5 Programme the Waspote to get and send real-time location.
- 6 Extensive testing of the FollowMe prototype:
 - a Functionality of FollowMe program.
 - b Ability to send and receive data (IP packets) over 4G.
 - c Functionality of the programmed Waspote.
- 7 Documentation must be made in the form of a thesis.

19.5 BOUNDARIES

This project will be finished after the final document i.e. the thesis, is accepted by Nspyre and The Hague University. The thesis must be in the hands of The Hague University on the 5th of June 2015. When the thesis is accepted and is considered satisfactory by the school, a thesis defence will be held in The Hague University. At least one coach from Nspyre and at least two coaches from the school will be present during this defence. After the presentation the project will be fully closed.

There are certain actions that will not be done:

- The Parrot drone, the Wasmote and the server are already present within Nspyre and are thus not required to be purchased.
- Apart from the 4G modification, the Parrot drone will not be further modified. This project will not improve any potential shortcomings of the drone.
- There will not be a management program that will manage multiple drones or the safety of the drones. Which means the prototype assumes a clear airspace and only one drone connection to FollowMe.
- There will not be an application that will distribute images to different clients. The images will be streamed to the client that is paired with the drone.

The project must contain certain aspects, performed by the executive, for it to be successful for Nspyre:

- A working FollowMe prototype.
- An extensive documentation concerning:
 - Changes made of different devices.
 - Usage of the prototype.
 - Eventual recommendations for the future of the prototype.
- If there is no working prototype, then there needs to be an extensive report on:
 - Why the executive thinks the prototype does not work.
 - What the executive tried to resolve the issues of the prototype.
 - Recommendations to get the prototype working properly.

All of the aforementioned must be finished before the 5th of June 2015.

For this project to be successful for the executive (the student):

- The project was considered successful by Nspyre.
- Nspyre is satisfied with the behaviour of the executive.
- The thesis is considered satisfactory by Nspyre and The Hague University.
- The thesis defence held by the executive is considered satisfactory by The Hague University.

The aforementioned will be completed before 26th of June 2015.

19.6 PRODUCTS

After this project the 7 phases will have their results/products:

- 1 The action plan will be accepted by Nspyre and The Hague University.
- 2 The research will have results and be documented.
- 3 The FollowMe program will be realised.
- 4 The drone will have 4G communication capabilities.
- 5 The Waspote will be able to relay real-time location to the server.
- 6 The prototype will be tested and results will be documented.
- 7 The project will be documented in the form of a thesis.

The FollowMe prototype will essentially consist of the products of phases 3, 4 and 5. Test can be executed after finishing one phase. To ensure the phase is successful and can be used in the next phase. This will also result in a shorter testing phase because some of it is already done after realising the respective phase.

The thesis will have the results of phases 2 and 6, the research and test results respectively.

19.7 QUALITY

In order to ensure the quality of the prototype certain precautions are made. The prototype described in chapter 6 products, will be thoroughly tested for its intended use, eventual safety regulations and legal guidelines.

Because of the form that this project has the executive can perform test after one of the prototype pieces are finished. This will ensure the quality of the prototype. But also confers a high chance for this project to have at least basic functionality. Which is a program that can direct a drone to certain locations.

The quality of the thesis will be maintained by starting on time. When the thesis is finished the executive will request permission to send it to multiple unrelated people to check for grammar. Also a log will keep track of the activities that the executive will have. So when writing the thesis the executive can consult this log to accurately determine past activities.

To keep a pulse on the project the project manager has decided to organize weekly meetings between at least the executive and the project manager. These meetings are quite informal however. And when there is nothing to report the meetings can be rescheduled. In addition, there will be a meeting with the executive and operational manager every 3 weeks.

Specialists can be contacted within Nspyre in case of shortcomings on technical skills of the executive. The executive can contact these specialist through the project manager or operational manager. These specialist are specialized in the programming language required, the Wasmote, setting up servers, etc.

19.8 ORGANISATION

The following people are involved during this project:

Name:	Email:	Mobile number:	Function:
Maarten Fekkers	maarten.fekkers@nspyre.nl	+31(0)6- 215 119 23	Project Manager
Boudewijn Noordman	boudewijn.noordman@nspyre.nl	+31(0)6- 517 968 35	Operational Manager
Gerard Heshusius	gerard.heshusius@nspyre.nl	+31(0)6- 216 782 44	Executive

The project manager will be attending the aforementioned weekly meetings. And will invest in the project on request of the executive. His responsibilities are:

- Exerting light influence during the course of the project to ensure it goes in the intended direction.
- Providing communication between the executive and specialists.
- Providing communication between the executive and third parties.
- Providing the necessary equipment i.e. the Parrot, Waspote, etc.

The operational manager will attend once every 3 weeks, in the weekly meetings but is substantively not involved in the project. His responsibilities are:

- Being the interface between the executive and Nspyre.
- Providing the executive a workspace.
- Being the interface between The Hague University and Nspyre.

The executive will devote most if not all of his time in this project. His responsibilities are:

- Executing all actions mentioned in chapter 5 activities.
- When the executive cannot proceed because of challenges, he is required to seek council of the project manager, operational manager or (the later determined) professionals respectively.

19.9 PLAN

This is a rough plan, seen in Figure 35. A more detailed plan will be included to this document in chapter 19.12 Attachment I Detailed plan.

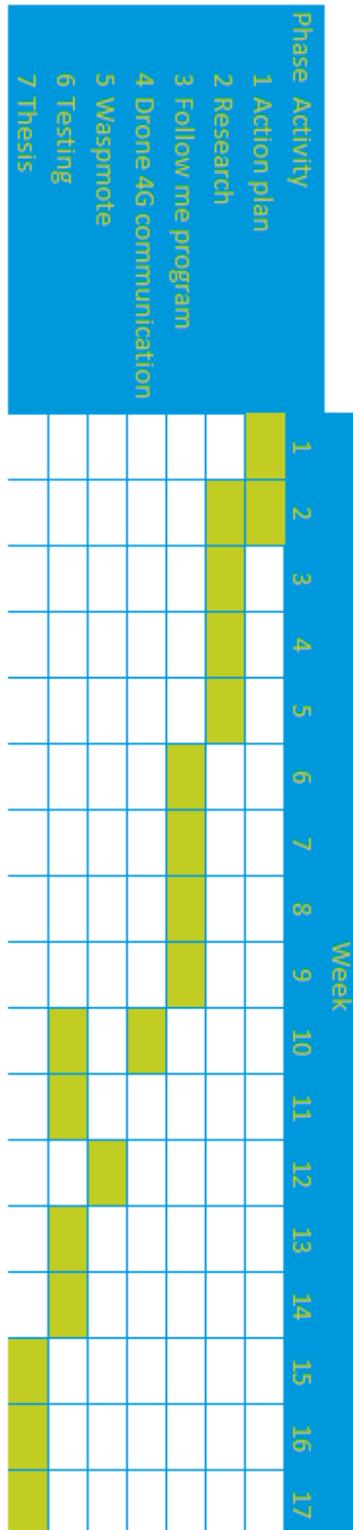


Figure 35 Rough plan

19.10 COSTS AND BENEFITS

This project will carry certain costs, as seen in Table 8:

Table 8 Costs

Name:	Costs:
Parrot AR.Drone2.0	€500,-
4G dongle	€50,-
Wasmote including accessories	€287,-
Space on the server	€5000,-

The aforementioned costs are already present within Nspyre and thus have no direct impact on Nspyre.

This project will carry benefits as well:

- The original intent of this project is for professional bicycling. However the applications of this solution will by no means be limited to bicycling. Giving it great flexibility for interested parties.
- When all is successful and be able to present during the Tour, will result in great media coverage of the project and Nspyre.
- For the Executive exclusive, when all is successful, he will be able to graduate from his university.

Certain assumptions are made before the project started:

- There already exists a functional drone and can be used or customized to meet the demands of the project.
- The Wasmote was already used in previous projects and these projects can be used to learn, customize and programme the device.
- The server is already present and will give no challenges to set up a test space for this project.
- The project manager has a lot of contacts within the professional cycling world. So when all is successful it will be easy to gain attention concerning the prototype.

19.11 RISKS

This project will carry certain risks:

- The executive is still a student and is not as proficient as a seasoned programmer. It will not break the project, but can result in some time loss. To counter this, the executive should try and make contact beforehand with the professionals.
- Both the project manager and the operational manager are not always on location. This again will not break the project, but can result in some time loss. Contact is possible via multiple mediums to counter this risk.
- The maximum speed of the Parrot drone is 40 km/h. There is a chance that the prototype cannot keep up with the target. To counter this a suitable target must be chosen or finding a new way to demonstrate the prototype.
- There is a chance that the prototype cannot be demonstrated during the Tour. As will be evidenced by research. A different way to demonstrate the prototype will be found with the help of the project managers many contacts within the professional cycling world.

After this project is complete using the prototype will carry certain risks:

- As mentioned in section 5 boundaries the drone will not be improved upon. Since the drone is intended for human control and not autonomous flight. Using the prototype carelessly could be dangerous for the prototype and surroundings.
- When using the prototype it is advised to council the manual within the thesis. It can result in unexpected behaviour and thus catastrophic results for the prototype or bystanders.
- All risks apply with normal quad copter usage.

