2015

Path planning for UAVs

Distributed Computing

ED5-1-E15
ROLAN ABDULRAZZAK OSSI
MIKE CASTRO LUNDIN
RADU NICOLAE MARICA
DRAGOS CRISTIAN DANILA
ARTURS GUMENUKS

Abstract

In project of semester 5, the group aimed to make a drone which would be able to avoid obstacles and travel through the shortest path in a map. The map and its obstacles are known and provided to the drone in advance.

The purpose of the project is to develop an autonomous aerial vehicle which does not require any human control. The group worked with drone technology which is widely used nowadays for both research and hobby.

The drone was assembled and used, including the necessary firmware and remote control. An Arduino microcontroller was used for testing and performing modifications to the behaviour of the drone.

Both theoretical and practical aspects of work were covered. Path-finding algorithm theories were discussed and analysed. The algorithm itself was successfully implemented in C++. Java programming language was used to create a simulation program for the drone. Radio telemetry as well as X-Bee modules allow different parts of the system to communicate.

The project reached a level of development where both advantages and disadvantages could be defined regarding the implementation. Further improvements are also discussed.

Rolan Abdulrazzak Ossi

Radu Nicolae Marica

Mike Castro Lundin

Dragos Cristian Danila

Arturs Gumenuks

Preface

The project has the theme “Distributed Computing” and wasprepared by the group ED5-1-E15 from Aalborg University Esbjerg, under the coordination of the supervisor Daniel Ortiz-Arroyo.

The parts of the project were assembled over 3 months. All of the members worked on the prototype, programming, and report equally.

Several sources have been used such as books in digital format as well as in physical format, educational websites and personal experience from past semesters.

In this report, the Vancouver referencing style was used to organize references and it can be said all references were taken from reliable sources.

With respect to this project, special thanks must be addressed to our main supervisor, Daniel Ortiz Arroyo for his support and enlightening advice with respect to algorithm selection and software implementation.

Further thanks must be addressed to Henry Enevoldsen who, as always, provided us with the necessary hardware when in need.

Contents

1. Problem Analysis

1.1 Introduction

In a research paper in applied mathematics, the author, Teppo Lukkonen defines a quadcopter as being a "a helicopter with four rotors".[1] In the past couple of years, the drone industry has encountered a development boom, now drones being assembled, improved, and researched by both companies or enthusiasts.

The idea of a quadcopter originates since the 20' and was an US Air Force project.[2] Almost a century later, quadcopters have become lighter, easier to manoeuvre, designed for more purposes and more accessible to people, both financially and by the comprehension of this continuously developing technology.

Some of the purposes that proved to be highly efficient for quadcopters are reconnaissance, aerial photography, military and the list can go on. Due to the continuous expansion of this technology, there is a possibility that society will be highly dependent on drones in the years to come.

In this project, the group starts from the idea that this technology can be improved further with even more creative and efficient ways than before. The purpose of this project is to make a quadcopter able to avoid obstacles and go from one place to another being provided with a map of the terrain. An algorithm will also be implemented in order to optimize the path the quadcopter will follow in order to reach a desired point in the provided map, knowing where the obstacles are located in the terrain. This concept can be further worked on to include more complex scenarios such as moving obstacles, further algorithmic optimization and machine learning.

In order to assess the efficiency and the error the quadcopter might have with respect to obstacle avoidance, the prototype was tested indoors, in a place where the error can be traced and determined (if there is any). In order to obtain this desired outcome, some important questions must be answered. One of these questions is simply which algorithm is more efficient in the case of this project with respect to complexity (in terms of memory, instructions and time), how it will behave in a variety of maps. How can the terrain and obstacles be discretized? How can deriving the

position of the drone can be done (with and without GPS)? Will the prototype movement be different from the expected path? If yes, how will it be traced, reduced and how will it affect the further behaviour of the quadcopter?

These questions as well as any problems that might be further encountered will be analysed in this report.

This project is also a distributed system for control of the quadcopter. The notion of distributed system can be simply defined as a group of computers or different processing units which act as a single entity. Our project implements this concept by having pathfinding algorithm on one computer entity, simulation on another one and control of drone on the prototype itself.

1.2 Problem analysis

One of the project tasks is to implement algorithms in software and assess the efficiency of it in avoiding obstacles and reaching a certain set point being provided with a map of the terrain beforehand.

In order to achieve the desired outcome, some points of interest must be analysed before proceeding with the actual implementation and testing sessions. One point of interest is the size of the terrain and the obstacles that the quadcopter will have to avoid. In order to be able to track with precision whether the prototype has any deviation error from its optimal path, the testing session will be held indoors in such a room that is big enough for the drone to have a secure flight. Due to this fact, the drone size must be carefully selected, with the maximum performance capability (flight time, drone weight, manoeuvrability) for the size and components. An overall analysis of the drone and the parts it is comprised of will be presented in a further subchapter.

Another important aspect which needs to be taken into consideration is the size and the amount of obstacles in the room (obstacle density). The obstacle density in the room must be such that drone is able to manoeuvre to the finish. Moreover, the fact that the exact position of the drone is hard to predict should not cause a collision. If the accuracy in drone positioning prediction is admissible, the density of objects in the room can be further increased to check whether it still performs the task correctly. The size of the obstacles is another factor that needs to be taken into consideration. Using UART communication and X-Bee transmitter and receiver, on processing unit will send information

to the quadcopter being able to control the PWM input and move it any direction without any human input from the remote control. The processing unit should determine the most efficient way by time and obstacle avoidance to reach its desired target, using an algorithm (the algorithms that are taken into consideration and the differences between them will be discussed in another chapter).

Furthermore, the choice of algorithm is an important step. There are many algorithms available for achieving this behaviour of the quadcopter (knowing the location of the obstacles in the terrain it should be able to follow the most effective route towards the desired position). Some of these algorithms are A*, Dijkstra, AA and the list can go on and on.[3] In assessing the efficiency of an algorithm, the time and memory efficiency of each algorithm will be determined as well as its complexity.[4] This subject will be tackled in later chapters.

Another important aspect to take into consideration is the difference the prototype might have between the expected path and its actual path. One issue might be the error stacking. There is this much error the prototype can handle before going off track. For example, if there is a small error off the track (unnoticeable at first sight), at every step the same error will be repeated and the error will stack until the prototype will be off path visible. The goal is not to make an error-free prototype, but error tolerant, in order for the prototype to move as far as possible relatively keeping track of its position without going off track.

Last but not least, the problem of discretization of the terrain must be taken into consideration. By discretization of the terrain, it must be understood that it simply means a way of dividing the terrain in longitudinal and cross sections in order to always be able to keep track of the position of the quadcopter in the terrain and determine an optimized path to a certain location based on the discretization of the terrain. Based on this discretization and the position of the obstacles in the terrain, the algorithm should determine the shortest path for the prototype to reach a point in the terrain. There are several more advanced ways of discretizing a terrain. Some of these are roadmap technique, cell-decomposition technique and potential field approach [5]. The roadmap technique is a way to discretize a certain terrain using graph theory, whereas the cell-decomposition technique is as the name says, is a discretization of the space in cells.

1.3 Alternative obstacle avoidance techniques

When speaking about obstacle avoidance techniques, there is no doubt that another method to consider when speaking about obstacle avoidance is image recognition. This method implies a camera (as visual information is the key input in this case) and processing software. This method has a lot of appliances in surveillance as it can be used for face recognition, license plate recognition, pattern matching etc. In other words, this technology is revolutionary and it has a perfect fit in a police departments and industry in general, but this technology comes along with a certain difficulty and price. These are the reasons why this approach in obstacle avoidance have not been taken into consideration, but rather a more simplistic approach using a predefined map. Ultrasonic sensors are another option which is easier to use and acquire (strictly speaking in a financial way), but there is also a trade-off between ease of use and the error they might have.

Most approaches done in different research papers require a visual input of the object and an image processing algorithm. However, this is not the only possible approach to this problem. Other sensing devices might be as good as any camera used for this specific purpose. These other sensing devices might be radars, sonars or different types of sensors such as ultrasonic.

1.4 Prototype Building

In this subchapter, the parts acquired for the prototype will be presented. Furthermore, in order for one to understand certain specifications, which these components might have, their working principle and general functionality of the quadcopter must be discussed. It can be said that the quadcopter has seven components which need to be acquired (some of them can even be built by oneself such as the power distribution board, frame and propellers). These components are

- frame
- propellers
- battery
- electronic speed controllers (four of them for quadcopter)
- motors (four of them for quadcopter)
- power distribution board
- flight controller which will translate the user input into actual motion.

These parts comprise the bare minimum necessary for a quadcopter to fly.

Of course, one might not wish to stop only at the point where one makes a drone able to fly. It can be even further improved, being almost no limitation in which this technology can be expanded and used. There is yet to discover and apply most of the possible uses this technology might have. One can attach a DSLR camera or a GoPro camera and use the drone in order to obtain aerial photography. Adding up a high quality flight controller ensured better hardware and software quality and now the drone can be used with image processing technology and with access to a database, and this technology can be used in a very broad range of applications such as obstacle avoidance, licence plate identification and so on. The obstacle avoidance outcome is attempted in this project, but instead of image processing software and pattern recognition, the prototype will rely on in advance known map and pathfinding algorithm, which will be implemented in software.

With respect to this project, a smaller quadcopter would be more suitable than a bigger quadcopter because it is easier to manoeuvre, it allows more flight time and it can achieve the same results as a bigger Quadcopter for less money.

Motor	High Kv(over 1000 Kv)	Small Kv(under 1000 Kv)	Medium Kv(around 1000 Kv)
Propeller	Small propeller(around 8 inches in length)	Long propeller(over 12 inches in length)	Average sized propeller(around 10 inches length)
Effect	-high torque -smaller thrust -used mostly for acrobatic drones -about same flight time as the other option	-low torque -high thrust -average flight time -used mostly in drones which are prone to lift payloads	-combined effect of both -average flight time

Figure 1. Quadcopter specification comparison

2. Theory/Hardware

2.1 Flight Controller

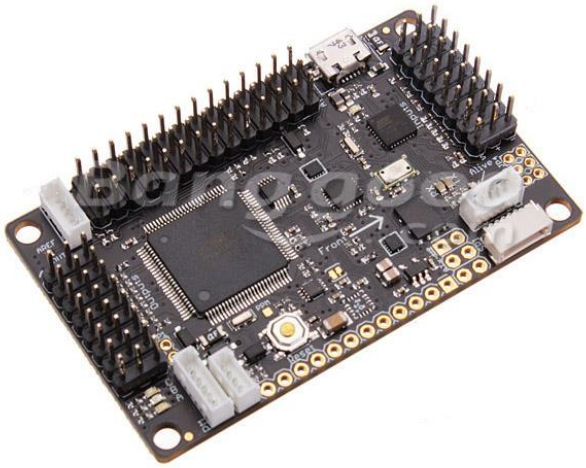

Figure 2.Flight controller [6]

It is a complex small circuit board, which works to direct the RPM of each motor in response to input. The commands are coming from the users into the flight controller, which define the how to spin the motors. The pinout configuration should be carefully implemented, as it will ensure perfect behavior of the craft. The flight controller is programmable and configurable and can adjust based on the motor configurations. Flight controller is using PID and gains to tune the controller.

We should know that there are three styles of flying styles, but most flight controllers are good for only one style.

The first style is called cinema flying. This type is using to produce smooth video. The flight controller in this style has dampened flight characteristics and small control stick rates for slow maneuvers.

The second style is called autonomous flying, and that means fly a prototype without touching the controls. The flight controller in this style should contain some flight characteristics like landing and flight, in addition to have open-source code, in which you can be able to add or remove some characteristics.

The third style is called sport flying, in which the users be able to fly fast and do maneuvers. [7]

There are different types of flight controllers and each type has different features. For example:

- **DJI NAZA M V2** this type can be used for the first flying style (cinema flying), in which the flight features are smooth and is easy to fly. In addition it has some smart features like smart orientation control. This type is easy to use by a cinematographer who has no experience in flying drones. The bad thing about the NAZA flight controller is it cannot be modified so the user cannot add any extra features**.**
- **3DR Pixhawk** this type using for the second flying style (Autonomous flying). This type has more characteristics then NAZA type. It is open-source, so it is possible to add or remove features.
- **CC3D/Naze32** like we mentioned before this type has a good flight features, which using for high speed maneuvers. [7]
- APM (Ardupilot) is our choice in this project. This type has normal prices, is reliable and flexible. This flight controller has open-source and both piloted and unpiloted (autonomous) modes. APM has many different applications like search and rescue, scientific research, and entertainment.

APM flight controller has best features than other types. It has low price compare to other types. APM has full autonomy open-source tools and code and open-source protocol communication (MAVLINK). This type can be used also for Tricopters, Quadcopters, Hexacopters and Octocopters.

2.1.1 APM2.5 and APM2.6

In this paragraph we will compare between the last APM versions, which is APM 2.5 and APM 2.6.

Some properties of APM 2.5 that is similar to APM 2.0 version. But it also has some improvements. It is ready to use and does not need to be assembled. APM 2.6 uses an external compass, so it has no on board compass, but has a connector to use the external compass. The both APM 2.5 and 2.6 have a barometric pressure sensor, so it is important to cover the sensor to make it waterproof and protect it from wind and turbulence. In some cases the APM is covered by an enclosure with foam protecting the sensor.

To understand the APM we first have to take a look to at its main parts:

1) I/O ports

2) GPS port
3) External I2C port
4) Expansion port
5) USB port
6) New wireless telemetry port (3DR radio): this port is used to the new telemetry adapter cable for APM 2.5
7) Data flash
8) Compass
9) Reset button

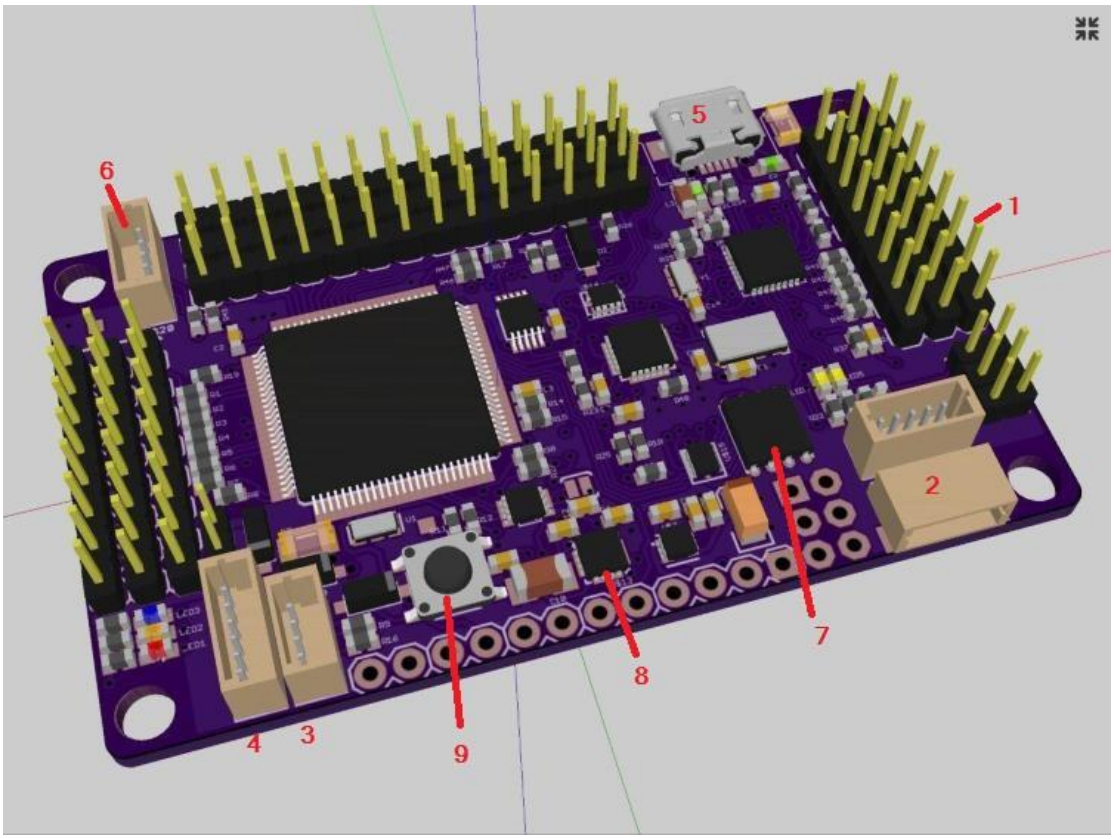

Figure 3. APM board[8]

APM has analog input pins and digital output pins. Analog pins are from pin number A0 to A8. All pins can use 5v. Pin 12 is the power management connector current pin and pin 13 is the connector

voltage pin, both accept up to 5v. Digital pins have also 9 pins. For digital pins we are using pins from 54 to 62. [8]

APM 2.5 is the local brain of the project, since all the components connect to APM2.5 flight controller. All the components that we use in the prototype are connecting to APM 2.5 Figure 4 where 3DR radio connects to APM via 3DR radio connector, 3DR radio has 6 wires (5v, TX, RX, CTS, RTS, GND). GPS is connecting to APM 2.5 via GPS connector, where has 4 wires (TX, RX, 5v, GND), output pins (1-8) are using to connect motors to APM flight controller.

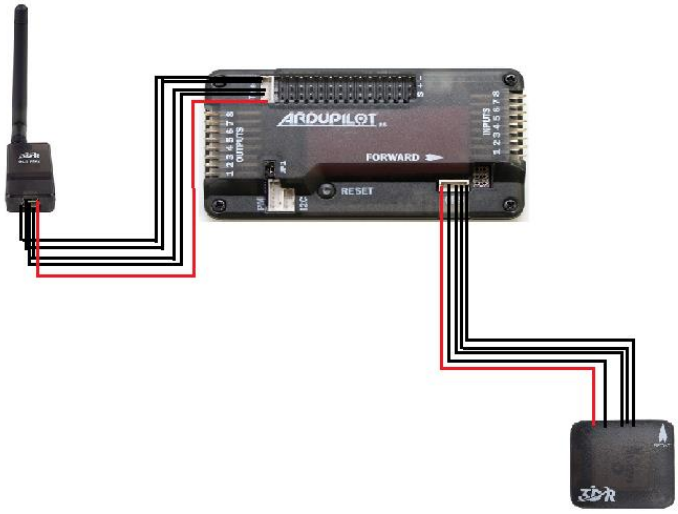

Figure 4.Connection

2.2 Brushless Motors

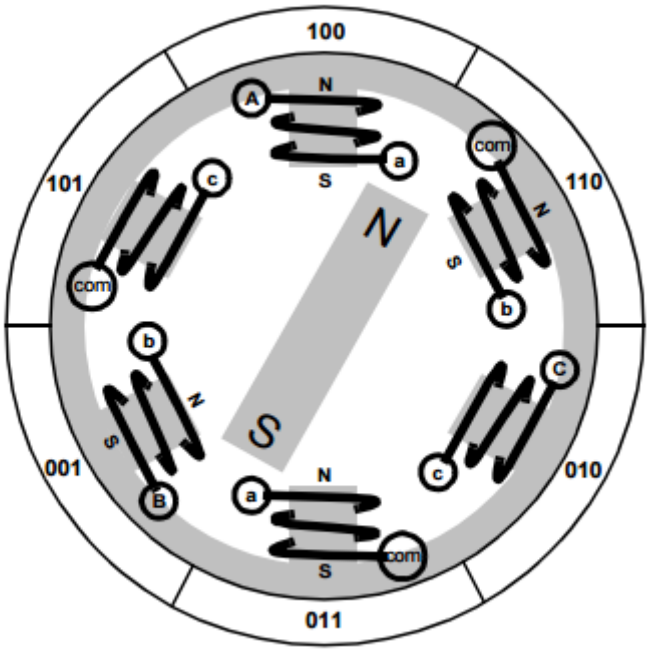

Figure 5. Brushless motor [9]

Brushless motor is one of the simplest types of motors and is powered by a DC electric source. It produces AC electrical signal to drive the motor. The motor is DC because the coil is powered by DC power.

Brushless motor has fewer parts that can be replaced, and because there are no brushes to replace, so a brushless motor can potentially work for more than 10,000 hours.

Brushless motors can be used in many applications. Low power brushless motors can be used in radio controlled model airplanes. High power brushless motor can use in electrical cars, and it is commonly used in some computer parts such as PC cooling fans.

The properties of motor are determine by material that used in the motor, the number of coils and the density of coils.

Brushless motor consists of four magnets. The stator of motor is composed by the electromagnets and placed as a cross with 90 degrees angle between them. Brushless motor rotors have permanent

magnets, so it does not need power all the time, brushless motors are silent and more effective in terms of power consumption. [10]

Hall sensor is using to find the rotor's pole (N or S), for a millisecond the motor will try to rotate the wrong way, but after a few degrees of rotation the Hall sensor will change the coils, then the motor will turn ON the correct rotation, where the sensor can sense if it is North or South Pole. Then Hall sensor transmits the signals to the controller of the motor, which the controller turn ON or OFF the coils that needed in order to provide the torque. [11]

In real life the brushless motor consist of four coils and four magnets around the rotor N-S-N-S, which Hall sensor is working not only to sense the magnetic field but also to sense North and South Poles. [12] [13]

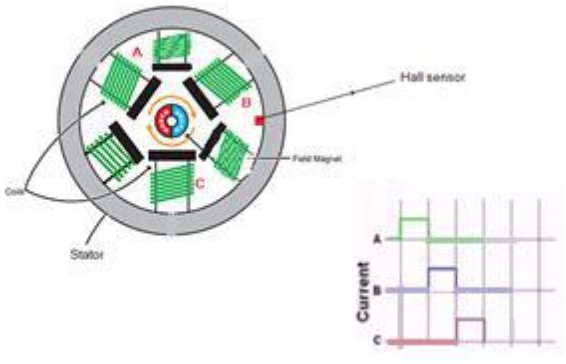

Figure 6. Motor's poles

Figure 6 showing the poles of the brushless motor, where it consists of three poles A, B and C, these poles represent the current of the motor. When the rotor heading to A phase that cause the current passes through coil of A pole and show as an pulse, when the rotor move to next pole, the coil will pass through that pole. [13] [14]

2.3 Speed controller (ESC)

Electric speed controller is a device that converts signals to electric supply, which can control the amount of power/speed of the electric motor. ESC analyses the signals from the microcontroller and regulates the difference of the direction and the speed of the motor, which indirectly regulates the power.

Electronic Speed Controllers are used in many R/C applications. One of the ESC applications can be in R/C cars, having the advantage of dynamic breaking.

ESC is of two different types, Brushed ESC and Brushless ESC. Brushless electric speed controller is a newer type of ESC which is more expensive compared to its older counterpart. The Brushless ESC can deliver more power to brushless motor and can continue for a longer time compared to Brushed ESC.[15]

There are different types of ESC, which each one has different current, size, and weight as shown in the table [16]

Type	Current A	Voltage v	Size mm	Weight g
ESC-12ALBEC	12 A	5 v	25x24x7	13.1 g
ESC-20A LBEC	20 A	5 v	35x24x8	19.9 g
ESC-20A SBEC	20 A	5.5v	35x24x8	22.8 g
ESC-30A SBEC	30 A	5.5v	41x24x8	30.4 g
ESC-45A SBEC	45 A	5.5v	50x32x13	55.4 g
ESC-60A SBEC	60 A	5.5v	50x32x13	58.1 g

Figure 7. Speed Controller specification

The ESC is usually connected to the flight controller/microcontroller, battery and motor. The wirings of the ESC are the following:

- Anode(+) and Cathode (-) wires, which are used to connect to the power supply
- JR connector, which is used to connect to the microcontroller/flight controller and is comprised of anode, cathode and signal wires
- 3 wires (3 phase power input), which are connected to the Brushless motor

The way to connect ESC will be as following steps:

First step is to connect JR connector to the throttle channel of the receiver and switch the transmitter ON, thus move the stick throttle to the lowest position. Next step is to connect the red (+) and black (-) wires to the power supply and the three Brushless motor cables to the wires of the ESC.

Next, flight controller sends a signal by the white wire to make sure it that has the correct connection to ESC. If the ESC sends a single 'beep' that means the brake is ON and the double 'beep' means the brake is OFF. The 'beep' is a signal to make sure that you have a correct connection to the other parts. The transmitter by this way is able to set one parameter of the ESC.

In this project we used 4 speed controller of Plush 30 Brushless Speed Controller type, one for each brushless motor, which has a very good quality and a wide range of programming. The voltage is 5v, has 25g weight and 45 x 24 x 11 mm size. [17]

2.4 LiPo battery

In order to power the motor we used a type Lipo battery (Lithium-polymer batteries). The battery has voltage 11.1V, 3.7V for each cell, and the capacity is 6000 mA, which it represents the energy stored within the cells of the battery.

Lithium batteries usually have a fully charge value, which is 4.2V for each cell, that means if the cell charged above that limit the battery will damage. The discharge value for each cell is 3.0V, and the discharging under this limit can damage the battery too. [18]

The nominal value of each cell is 3.7V, which means the average of the charge and discharge value. Lithium battery consists 2 sets of cables, one pair is represented by the discharge anode (red +) and cathode (black -), and the balance wires, which are used to charge all the cells equally in the battery. [19]

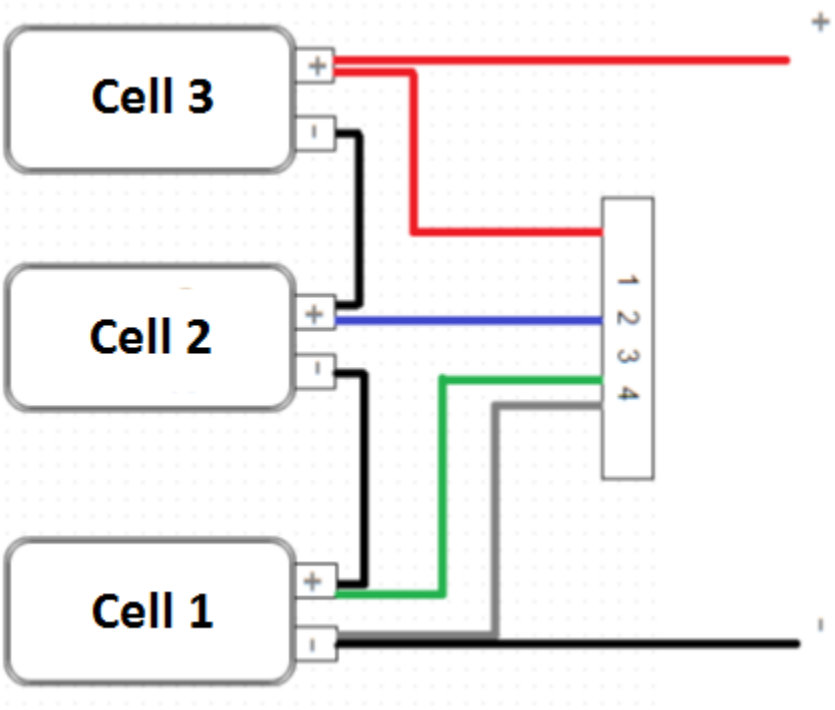

Figure 8 - LiPo Battery

To power our project we connect the power supply (LiPo battery) to the speed controller via two wires anode (+) and cathode (-).

There are some characteristics of Lithium battery.

- Power input: The battery needs to be charged to a larger voltage than the minimum required voltage by the engines because of inefficiency.
- Power output: we know that the wattage is the product of voltage and current, so the current of battery depends on the voltage. However, the max charge current on a 50W charger will be 3.9A as the following equation (50W / 12,6V = 3.9A).
- Balancing: is the most important part of charging Lithium battery. The cells can discharge unequal and become unbalanced, so the balancing plug leads to balance the cell.

Lithium batteries must be charged with a compatible charger, which has an equal amount of cells as the battery. Additionally they must be charged under supervision, since fires are not uncommon.

2.5 Error analysis

At the start of the project two possibilities where considered regarding tracking the position of the drone. The first option was using a GPS module and tracking the position of the drone using the data received. This has the disadvantage of having to buy a module, which can be very costly if a good precision is desired and furthermore, the discretization must be minimum the size of each GPS coordinate, which in affordable modules can be too high to involve obstacle avoidance, unless there are large areas without obstacles. Furthermore, scaling the discretization makes testing more difficult, since it will require a very large outdoors area and the drone has a battery time limitation.

Assuming GPS coordinated have a size of 2m from the centre to the next coordinates; it will mean that we can have a random error within this range as shown in the following figure.

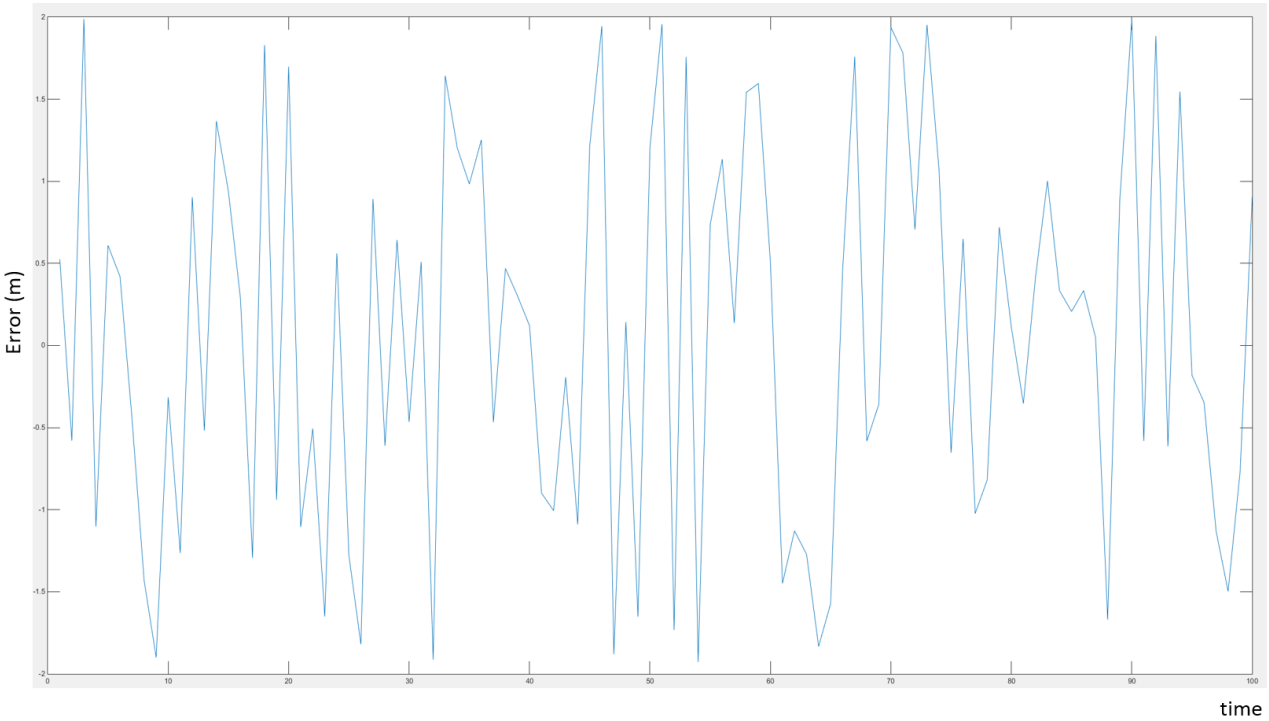

Figure 9: Example of random error in GPS

This has a large advantage that the error will always stay within the range, and will escalate to a large number. However as discussed before, this would mean that having a small obstacle would mean blocking the whole GPS block, which is a large area. Because of this, and our GPS module was not compatible with the flight controller, we decided to use the second option.

The second option was to implement a similar system to what in navigation systems is known as dead reckoning. This means that knowing the initial position, we calculate the next position using the direction of movement, duration and velocity to get an estimated position. Since the drone will only move within discretization's we can reverse this concept, and estimate a duration and velocity in which to move, in order to end in an adjacent discretization node. Additionally, if the discretization is shaped in squares, then the direction of the movement will simply mean changing which 2 engines spin faster in the drone. Also, once the duration that is needed is calculated, then it will be the same for all sides.

If the concept is to be extended to diagonal movement, then we simply need to spin three engines instead of two, and ideally increase the duration by a factor of $\sqrt{2}$ since it has a Pythagorean proportion. In practice of course, this will not be completely accurate though, since the velocity will be a curve and not a straight line as shown in the following picture:

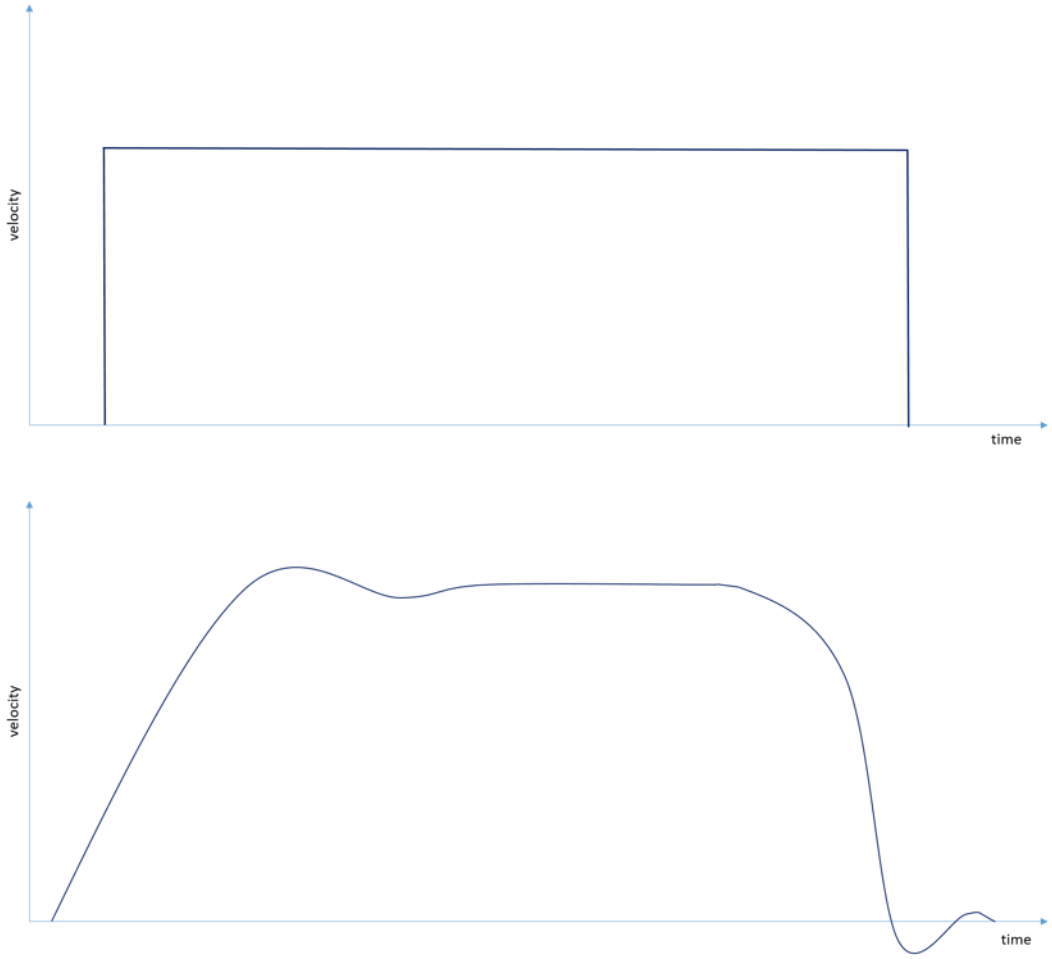

Figure 10: Ideal velocity and realistic velocity

This is one of the causes that there will be an error at each step. Other reasons that will cause errors are the possibility that the battery charge level will affect the speed range, disparity in the engines/propellers and wind or other interference.

This concept relies in having a small error, since if for example each step has a 50% error, then by two steps you might be in the wrong discretization, and potentially collide with an obstacle. If the error for example is ±0.01%, then depending on the discretization size, it will take a larger amount of movements for the estimated location to be critically inaccurate. The error has a potential of escalating, since if the errors on one direction are more often, it will result in potentially a very large error, as the figure shows:

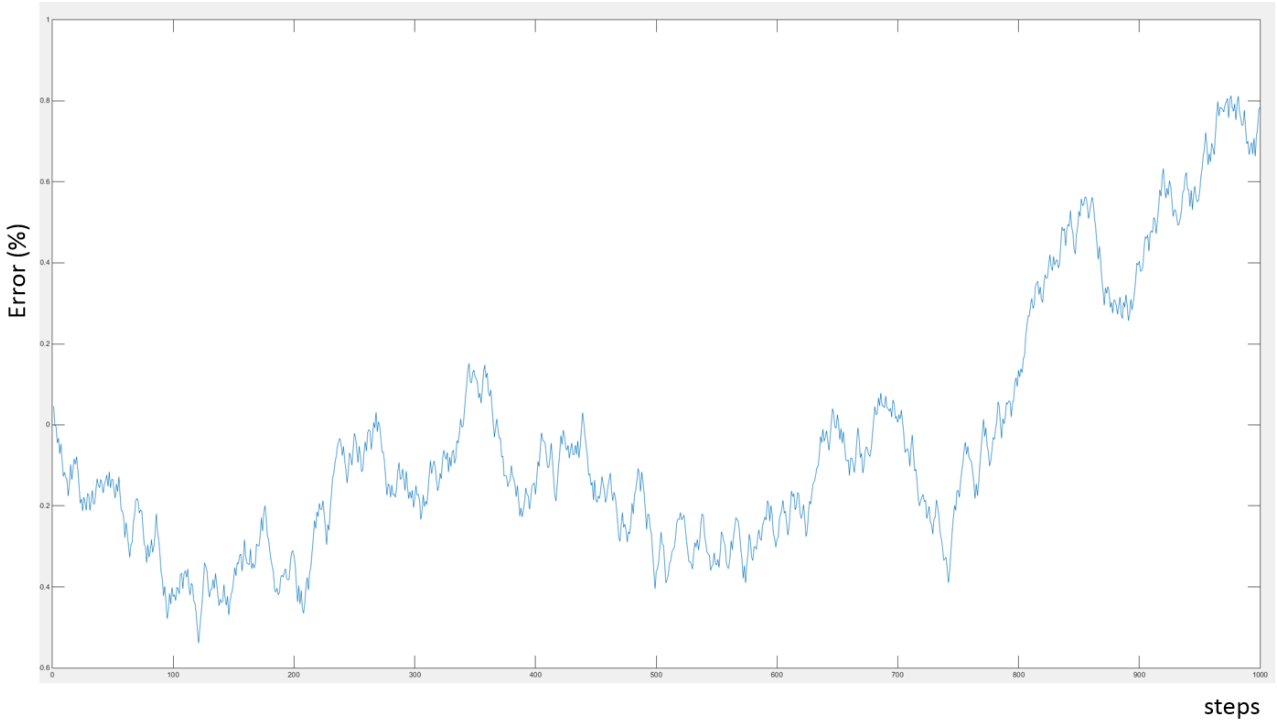

Figure 11: Error escalation in random numbers

This means that the error at step t can be defined as:

$$Error_t = \sum_{n=0}^{t} E_n$$

Additionally, as a further step, the A* algorithm could be modified so that it finds paths along walls and obstacles, which it uses in order to recalibrate the position, assuming that the obstacles are always aligned to the discretization, and there is a sufficient amount of obstacles to calibrate to.

2.6 Discretization

Another issue that needs to be defined is how the terrain is discretized. This mainly implies defining the size of the discretization.

While a smaller size will allow better precision, it will also mean more processing time, longer communication and more memory required. It will also however take less steps for the error to become a large percentage of the discretization.

On approach could be to make discretization the size of the drone. This is a good minimum size of discretization, since it will mean the drone will not fill more than one block. One disadvantage is that in order to make diagonal movement work we would need to implement another concept, which we call safety regions.

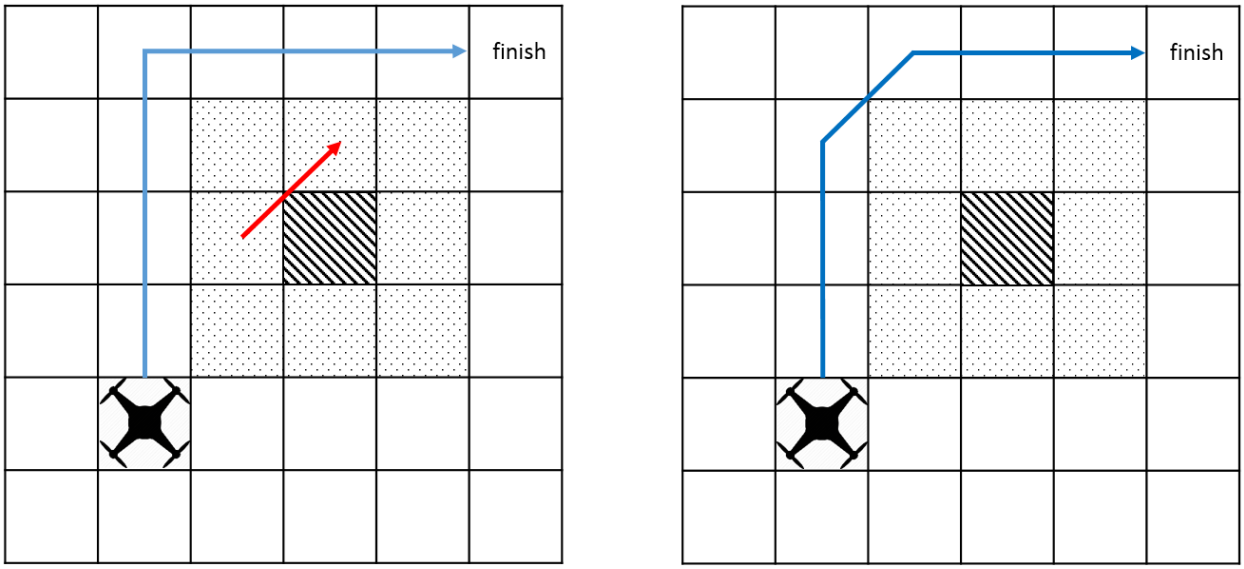

Figure 12: Safety zone with and without diagonals

As the figure above shows, safety zones wrap obstacles to prevent diagonal movement that would cause a collision, which is shown by the red line. While this might be a necessity when allowing diagonal movement. On the other hand, in a system which does not allow diagonal movement it would increase the amount of steps necessary to go around an object by 2 steps. Another disadvantage is that it will result in potentially less possible paths, since it blocks 8 extra discretization nodes. On another note, it will also make our system more error tolerant, since the error will have to be larger to cause a collision.

For our project, we take the concept of safety zones, but implement it in an alternative way. Since we have control of the obstacle size, we assume that obstacles are much smaller than the discretization, so diagonals through obstacles are allowed.

The discretization size depends on the settings selected on the drone, and can be scaled by changing the throttle value or the impulse duration. For the testing, we used the minimum throttle possible where the drone would still consistently move. More on this is discussed on the testing section.

2.7 Taylor Series for square root

Taylor series is a powerful mathematical tool which is used to approximate a function around a point.[20] In the case of the project, the Taylor series for a square root function is particularly powerful as it enables one to control the number of iterations when applied for square root functions in comparison with the standard math libraries in which the number of iterations cannot be controlled when applied for square root.

This comes in handy as with the control of number of iterations for solving a square root, the number of decimal points of the value obtained can also be controlled, thus obtaining a better memory and CPU time management. Needless to say, as it is a powerful mathematical tool, the Taylor series is just an approximation of the square root of a function and the degree of accuracy can be manipulated by increasing or decreasing the number of iterations.

In case of a multivariable function, the Taylor expansion formula is:

f(x,y) = f(x_0 , y_0) + f_x (x,y)(x-x_0) + f_y (x,y)(y-y_0) + $\frac{1}{2!}$ [f_{xx}(x,y)$(x - x_0)^2$ + 2f_{xy}(x,y)(x-x_0)(y-y_0) + $f_{yy}(y - y_0)^2$] + … [21]

Where f_x, f_y are the partial derivatives of the function with respect to the variable of interest andx_0, y_0 are called the initial conditions of the variables of interest. Furthermore, f_{xx} and f_{yy} represent the partial derivatives of x with respect to f_x respectively y to f_y.

The Taylor series formula for square root can be expressed as:

$$f(N, d) = \sqrt{N^2 + d}$$

Then using multivariable Taylor:

$$f(N,d) \approx f(N_0,d_0) + f_N(N_0,d_0)(N - N_0) + f_d(N_0,d_0)(d - d_0) + \frac{1}{2}f_{Nd}(N_0,d_0)(N - N_0)(d - d_0) + \frac{1}{2}f_{dN}(N_0,d_0)(N - N_0)(d - d_0) + \frac{1}{2}f_{NN}(N_0,d_0)(N - N_0)^2 + \frac{1}{2}f_{dd}(N_0,d_0)(d - d_0)^2 \ldots$$

Optimize $\frac{|d|}{N^2}$ → Lowest $N_0 = N$ → $(N - N_0) = 0$ → $f_n(N_0,d_0)(N - N_0) = 0$

Smallest $d_0 = 0$

So all partial differentiations relative to N are 0 so →

$$f(N,d) \approx f(N_0,d_0) + f_d(N_0,d_0)d + \frac{1}{2}f_{dd}(N_0,d_0)d^2 + \frac{1}{6}f_{ddd}(N_0,d_0)d^3 \dots$$

$$\approx N_0 + \frac{1}{2\sqrt{N_0+d_0}}d + \frac{1}{2}\left(-\frac{1}{4\left(\sqrt{N_0^2+d_0}\right)^3}\right)d^2 + \frac{1}{6}\left(\frac{3}{8\left(\sqrt{N_0^2+d_0}\right)^5}\right)d^3 \dots$$

$$\approx N + \frac{d}{2N} - \frac{d^2}{8N^3} + \frac{d^3}{16N^5} \dots \quad = \sum_{n=0}^{\infty} \frac{(-1)^n(2n)!d^n}{(1-2n)(n!)^2 4^n N^{2n-1}}$$

As the number of elements in the Taylor series increase, so does the accuracy as each element will become smaller and smaller approaching 0, but never actually reaching it until one of the elements in the Taylor series will be dividing infinity.

As it was mentioned above, the Taylor series is an approximation of some function around an equilibrium point. With respect to the project, the function involved may be the square root of a multivariable function or a function with one variable.

The Taylor expansion formula is important with respect to this project as it is essential in calculating the length of path from the start node to the goal node, as diagonal movement is determined using Pythagoras's theorem. Due to the fact that the diagonal length from one node to another is represented by a square root, this Taylor expansion formula allows one to approximate the value of the square root and thus numerically approximate the length of path.

Given a practical example, say the square root of interest that needs to be approximated is 120. The closest perfect square is 121 (11^2). This means that if $\sqrt{120}$ is written as $\sqrt{11^2 + (-1)}$,the expression is perfectly correct from a mathematical point of view.

Applying the Taylor expansion formula, the result outcome would be $\sqrt{11^2 + (-1)}$=$\sqrt{N^2 + d}$ and thus:

$$f(11,-1) \approx 11 - \frac{1}{22} - \frac{1}{10648} = 10.95445154$$

The real answer is 10.95445115, so just with two iterations; we have already gotten six decimal points accuracy.

2.7.1 C++ code implementation of Taylor expansion formula for square root:

As Taylor expansion is a series used to approximate complicated functions, when implementing this particular series in software, one point of interest is the order of convergence of this series. Therefore, the main point of interest is how fast the series can converge to an acceptable result. As the square root function can have the format $\sqrt{N^2 + d}$, then the rate of convergence of the series can be determined by using the smallest possible $\frac{|d|}{N^2}$ (when d is the smallest relative to N^2). As the ratio $\frac{|d|}{N^2}$ becomes smaller and smaller, this means that the series can converge faster as it is close to the end point.

The first approach in software in implementation of this series is iterating through numbers starting from 1 until it reaches the optimal N element in the square root. After reaching the number N, the program will run a while loop which will iterate through all the elements of the Taylor expansion formula until it reaches either the machine epsilon threshold or a defined precision (1.0e-100). Then the loop will break. This behaviour of the code can be described by the following C++ structure:

```
  while(true){

 nextit =
(pow(1,n)*fact(2*n)*pow(d,n))/((12*n)*pow(fact(n),2)*pow(4,n)*pow(N,2*n-1));

        res += nextit;

        n++;

        if (nextit < 1.0e-100){

            break;

        }
```

Optionally it could also be implemented using a do while loop.

2.8 Longest Path

To begin with, let one describe why it might be important to take the longest path into consideration. Tie breaking technique (explained later in the report), for instance, has a factor variable, chosen with respect to maximum amount of nodes in the path. Another application of the longest path knowledge is memory management, because the maximum size of path holding data structure is known and there is no need to reserve extra-space. It can also help with predicting stack overflow problems. Thus, in this subchapter the dependence between map size and the longest path will be discovered.

To begin with, it is necessary to give a definition to the longest path and what it implies. Basically, it is a worst case (in terms of distance to travel) for the algorithm, where the path found by A* algorithm includes as many nodes as possible on a certain map. It was empirically determined, that a spiral-obstacle designed map makes the algorithm output the longest possible pass. The formulas available further in the subchapter were only found for square dimension grids.

There are two types of obstacle combinations, which can constitute the longest path. These are a spiral and a tunnel. Some examples are supplied below. Colour notation is shown in the figure. Width, height and path length variables are given in node units. Finish node is not included in the path length.

Node color	Meaning
	Space
	Start
	Finish
	Obstacle
	Path

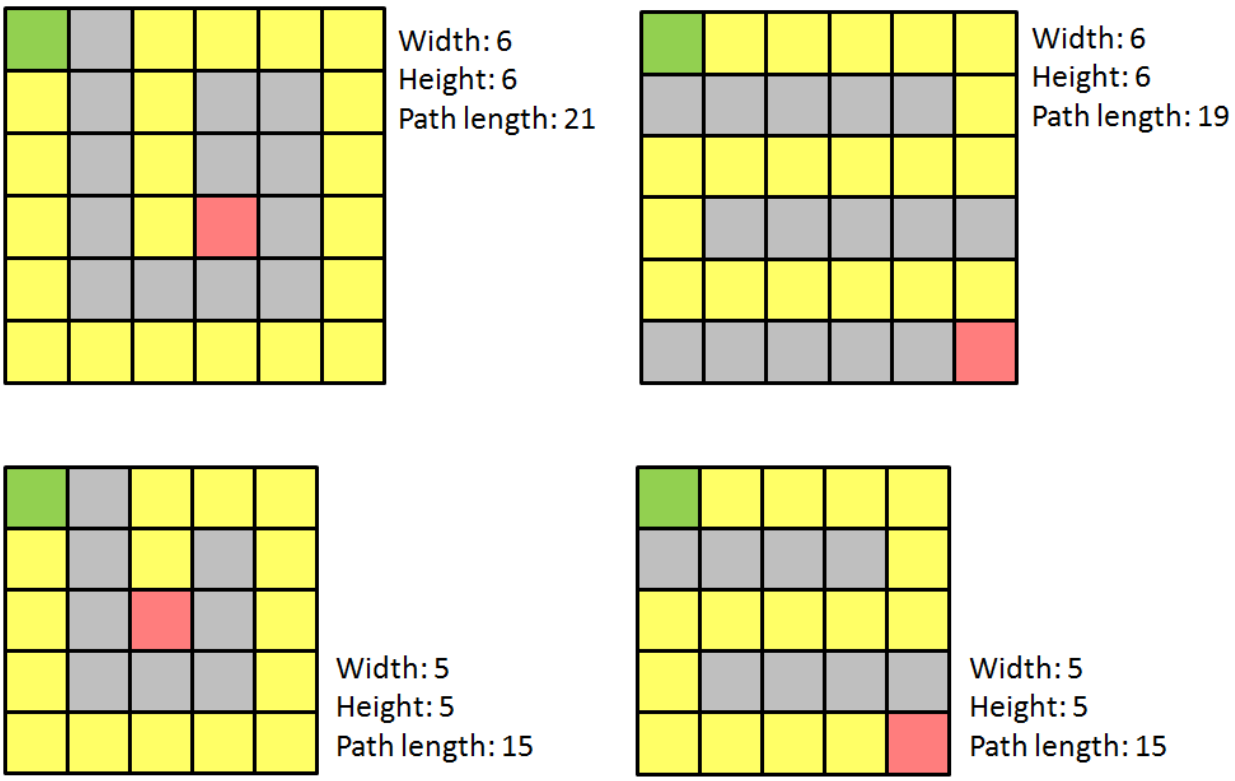

Figure 13. Different map designs.

There is a conclusion, which can be made from the provided picture. Odd-dimensional maps (5 by 5) give equally long paths for both spiral and tunnel, so there is no difference which map is used. Even-dimensional maps (6 by 6) give paths of different lengths for tunnel and spiral obstacle combinations.

The mathematical aspect comes now. The following formula works for odd-dimensional map with tunnel obstacle, where “pl” stands for path length and “N” for one dimension (width or height).

$$pl = \underbrace{(N-1)}_{\text{First row without starting node}} + \underbrace{\frac{N-1}{2}N}_{\text{Full rows}} + \underbrace{\frac{N-1}{2}}_{\text{One path node rows}} - 1 \quad \text{Goal node}$$

Spiral obstacle gives the longest path for even-dimensional maps. The following figure shows the pattern in obstacle wrapping.

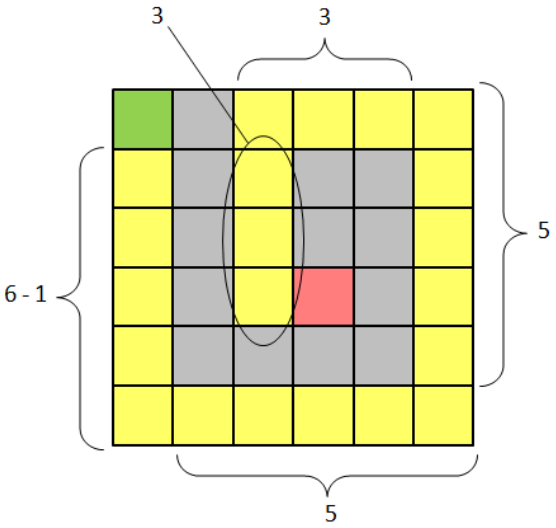

Figure 14. Regularity in the path.

It is possible to judge from the picture that the length of path is N-1 plus the summation of all odd numbers from N-1 to 3 multiplied by two. The formula can be immediately obtained.

$$pl = (N-1) + \sum_{i=2}^{N/2}(2i-1)2$$

These formulas of course are not the only possible. An alternative formula, obtained for L-shaped obstacles, is presented below and includes the finish node in the path length.

$$pl = \begin{cases} \text{N is even:} & \left(\sum_{i=0}^{\frac{N}{2}} 2(N-2i)-1\right) - 2 + \frac{N}{2} \\ \text{N is odd:} & \left(\sum_{i=0}^{\frac{N-1}{2}} 2(N-2i)-1\right) - 1 + \frac{N-1}{2} \end{cases}$$

2.9 Security

Since our system is written in C, C++ and Java, which means software security is a very relevant in varying degrees depending on different parts of the system.

One way to determine the security of the system would be to examine memory safety and type safety of each part.

“Memory safe” is a term used to describe a program for which all executions are safe in terms of memory access errors. Examples of memory access errors are buffer overflows, null pointer dereference, using memory after free (), using uninitialized memory or illegal freeing of memory. A memory error for example happens when a program accesses memory that is not part of the stack, the heap or static data areas.[22]

Type safety is linked to the program possible outcomes, which should be well defined. For example, if an array of **int** datatypes “a” is declared to have a size of 5, and then you call a [6] = 9, then the behaviour will be undefined in C. It is important to note that type safety is ensured in some programming languages. The following are some examples of safe and unsafe languages.

- C/C++ are unsafe: Accessing and writing memory using arrays outside their size is an example of this.
- Java and C# (and by extension C++ managed classes) are considered type safe, since they will call exceptions (f.e ArrayIndexOutOfBoundsException). While it is hard to prove type safety in the full-blown language implementation[23], it is proven that its main parts are type safe by using subsets of the languages like Featherweight Java.
- Python and Ruby are type safe, because they are dynamically typed languages. Since they are null-type languages, they will not have any semantically correct code that will not be well defined.

In terms of memory safety, the simulation part is memory safe, since Java does not allow the use of pointers directly. On the other hand, the C++ part of the system is type safe in all its classes by using managed classes, with the exception of the A* algorithm classes, which are unmanaged (the definition of a managed class will be given later). This means that the whole C++ program could be potentially vulnerable to attacks by exploiting unmanaged classes.

Some potential vulnerabilities are format string attacks to print internal data for example Printf(“%s”) will print until it encounters an end to a string, potentially leaking confidential

information. Another exploit could be buffer overflows, which can be used to overwrite local variables, using designed inputs to the system. This can also be further exploited to inject code to the system which would be done by altering the instruction return pointer (eip) to point to a part of the memory where attacker code (in assembly) is. A way for the attacker to ensure that the program will end in the desired instruction is to add **nop** sled (x86 assembly) via buffer overflows.

A way to solve type unsafety in C programs could for example be to implement projects such as those, which redefine pointers as capabilities.

3. Communication

3.1 Universal asynchronous receiver/transmitter (USART)

A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is a microchip is used to simplify communication through a computer's serial port by using the RS-232C protocol. UART works to provide the computer with important interfaces to connect to other modems and serial devices, it also provides synchronous option. UART aims to convert bytes from PC parallel to a serial bit, where serial port is sending one bit at a time. When you insert a character, the terminal gives it further to the transmitter, where it sends that byte out of the serial line, bit by bit.[24]

USART has two primary forms of transmission, Synchronous and Asynchronous. Synchronous serial transmission has a timing signal that allows the receiver to know when to read the next data bit. Synchronous communication is normally more efficient for sending and receiving data bits. This form of transmission is used with printers, where the data is transmitted on one group of wires. Printers are not serial devices, because devices are sending data in parallel, which mean when insert word of data for each clock signal, it using separate wire for each bit of the word.

Asynchronous transmission can transmit data without having to send a clock signal to the receiver from the transmitter, but agreeing on timing parameters, and relying on special bits to synchronize the units. When a word sending by UART for Asynchronous transmission, a first bit called Start Bit, is added to the beginning of the word that will be sent. Start Bit is used to give an alert to the receiver that data will be sent. The sender and the receiver clock must be accurate enough and the frequency drift should not exceed 10% during the transmission.[25]

After the Start Bit is sent, the bits of a word are sent individually, with each bit having an equal duration. The receiver looks at the signal in the half point of the period, determining if its value is 0 or 1.

When the receiver has received all the bits of a word from the transmitter, it will check the Parity Bit, where Parity Bit is an used to check simple errors. No matter the result of the parity check it will also receive a Stop Bit from transmitter.

The basic function of UASRT is converting parallel data to serial data for transmitter and converting serial data to parallel data for the receiver. This concept can be extended to cause continuous communication, and is therefore not limited to sending data once.[1]

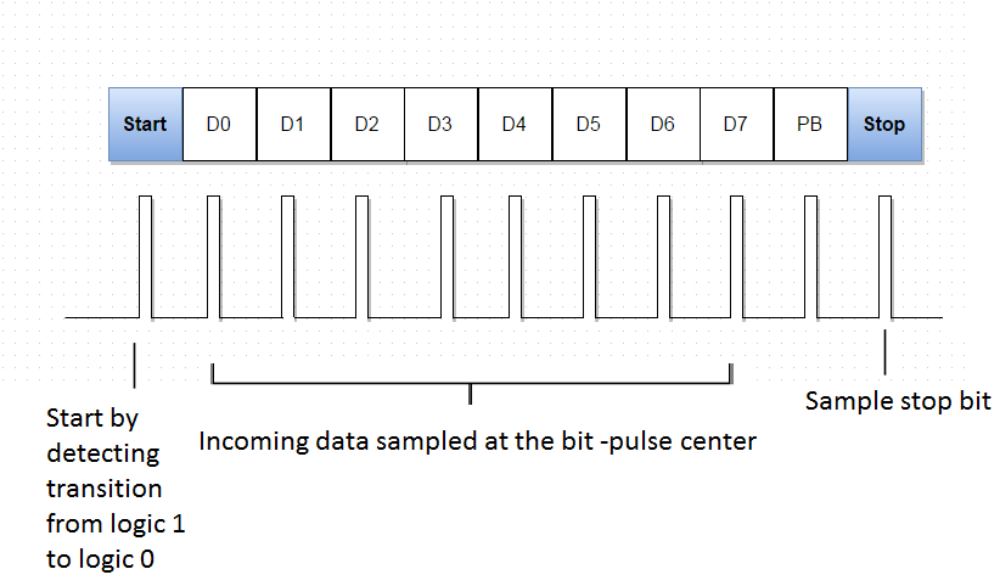

Figure 15. UART data frame

Data framing is showing a word bit, which consists of 8 bits (D0 to D7), Start is Start bit logic 0, where Start bit tells receiver that the will be sent. Stop is Stop bit logic 1, where it refers to the end of a word bit. PB refers to Parity Bit, which is working to check simple errors. Word bits will be sent after the Start bit, and word bits are sent with a constant duration. Thus, the receiver looks at the voltage, if it is a high logic, it will record a binary digit 1, and 0 if is low.[26]

3.2 MAVlink

In order to be able to determine from the GCS (Ground Control Station)[27] in every moment in time where the quadcopter is on the given terrain, the flight controller mounted on the quadcopter must communicate with the GCS through telemetry radio.

In order to be able for the GSC and the flight controller to communicate through the telemetry radio, there must be an “agreement” between the two pieces of technology on how the information is going to be sent and received. This “agreement” is called a protocol and in the case of this project,

the MAVlink protocol is the one of interest. MAVlink stands for Micro Aerial Vehicle link.[28] It is an open-source protocol[28], meaning that it can be modified as the design is publicly available.

The way this protocol works is that the Mission Planner software encodes the message in packages of minimum 17 bytes each. [27] Every encoded message contains 6 header bytes, 9-256 payload bytes and 2 checksum bytes.[27] The header bytes contains information such as message length, sequence number, system ID, component ID and message ID.[27] Basically the header bytes contain all the information about the actual information transmitted (the payload bytes).[27] The checksum bytes contained within the actual encoded message are used for error detection and in case the message is corrupted, then it is discarded and the software waits for the next encoded message to be received. [27]

3.2.1 How MAVlink Protocol works

The MAVlink Protocol works between the flight controller and GSC. It is a bi-directional protocol [27] meaning that it enables the communication between the devices in both directions (flight controller communicates with the GSC and vice versa).

When the flight controller receives an encoded message, with the help of the method handlemessage(msg) reads the System ID and Component ID in order to check the validity of the received message. After the System ID and Component ID have been checked, the actual data located in the payload bytes is acquired and included in the appropriate data structure containing the same information. [27]

Another important aspect which needs to be mentioned with regard to the MAVlink protocol is the overall security offered by this protocol.

In a thesis written by Joseph A. Marty called “Vulnerability Analysis of the MAVlink Protocol for Command and Control of Unmanned Aircraft”, the author states that “Since this protocol is designed with attention to availability and safety, its security might have been overlooked”.[28] Rephrasing the previous statement, the author believes that this protocol trades off the security of transmission, meaning anybody can access the data log in order to see the information sent/received, for reliability of transmission, meaning trying to minimize any information loss might occur during transmission and reception of the information. The author of the paper states that side channel attacks can be enough to find the communication configuration of the unmanned aerial vehicle.[28]

One way described in the paper that could solve this issue is either by managing to implement a cipher that could encode the communication configuration (such as the Rabbit cipher) [28] or use a library for APM that could ensure a safe transmission using encryption as well (Network and Cryptography library) [28].

Due to the study nature of the project, a safe transmission of information is not required (not for University level), but this problem was stated due to the fact that this issue is serious and needs to be taken into consideration when important payload is being carried.

When dealing with information transmission, errors may occur when transmitting or receiving the data.[29] There are different ways of error checking when one is speaking about information transmission, such as parity check, CRC bits etc. [29] The one method of interest is CRC, which stands for “cyclic redundancy check” which processes the binary information and produces a residue to check the integrity of the message. What happens next is that the binary information is repeatedly XOR evaluated with a predefined binary number, and the remainder is added to the block of characters as a cyclic redundancy check byte(s).[29] The CRC byte(s) are then verified by the receiver, by successive XOR operations. If the residue of the information and CRC byte(s) is 0, then no error is detected. Otherwise, a request will be sent in order to resend the package (also called ARQ or automatic repeat request). This is the way the actual software ensures that there is little to no information loss due to the transmission or reception of the package.[29]

3.3 Error detection and correction

3.3.1 Cyclic Redundancy Check

As described in the MAVLINK section, the protocol implements cyclic redundancy check (CRC). CRC is a very popular error detection code, and there are many different implementations. This section will focus on CRC theory and specifically CRC-16-CCITT also known as KERMIT. The concept of CRC is based on agreeing on a polynomial, which is used to verify the integrity of data.

As an example, we assume the polynomial agreed on is $x^3 + x + 1$ then the bit representation of it would be 1011. The data sent is 11010011101100, then it will require 3 bits for residue, since the polynomials maximum power is 3. Form here the algorithm simply does a successive XOR operation like shown next:

```
11010011101100 000
1011
-----------------
01100011101100 000
 1011
-----------------
00111011101100 000
  1011
-----------------
00010111101100 000
   1011
-----------------
00000001101100 000
       1011
-----------------
00000000110100 000
        1011
-----------------
00000000011000 000
         1011
-----------------
00000000001110 000
           1011
-----------------
00000000001110 000
           1011
-----------------
00000000000101 000
             101 1
-----------------
00000000000000 100
```

Figure 16.Successive XOR operation with polynomial

It is worth noting however that the most commonly used powers are 9, 17, 33 and 65, since the residues fit exactly in 1, 2, 4 and 8 bits. In the case of CRC-16-CCITT it uses 16 bits, that is 2 bytes for CRC and has an error detection of 99.9984%. The polynomial has a large impact on the detection range of the CRC protocol, and the polynomial used in the CCITT protocol is $x^{16} + x^{12} + x^5 + 1$.[30]

Additionally, it is important that both systems agree on MSB or LSB endian, or implement an endianness agnostic system, that is that it guarantees successful interpretation by sending both residues.[31]

3.3.2 Hamming Code

Hamming code is one of the simplest forms of error detection and correction. It relies on the simple concept of parity checks, in order to locate where the error is happening. It relies on placing a parity bit in every 2^n value in-between the data. The method, checks n bits from bit 2^n and then skips n bits as shown in the following figure:

Data: 0 1 0 0 1 1 0 1

P1 P2 0 P4 1 0 0 P8 1 1 0 1

P1 P2 **0** P4 **1** 0 **0** P8 **1** 1 **0** 1 = **0**

P1 **P2 0** P4 1 **0 0** P8 1 **1 0** 1 = **1**

P1 P2 0 **P4 1 0 0** P8 1 1 0 **1** = **0**

P1 P2 0 P4 1 0 0 **P8 1 1 0 1** = **1**

Sent Data: 0 1 0 0 1 0 0 1 1 1 0 1

Figure 17. Hamming code Data processing

To show the error correction protocol works we change the sent data bit 11 to 1, and we display the process of finding the error in the next figure:

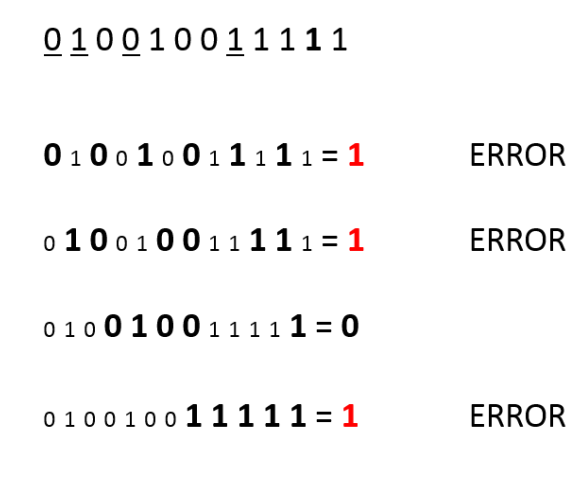

Figure 18. Error index detection

3.3.3 Comparison

It is very complex to compare CRC as a concept, with Hamming codes. This is mainly because there are many different options when using CRC, this section will be limited to comparing the version called CRC-16-CCITT with Hamming codes.

To start we will compared the amount of bits that the protocols use, as a percentage, as the amount of data increases. In the following graph the blue line is CRC and the red line is Hamming codes.

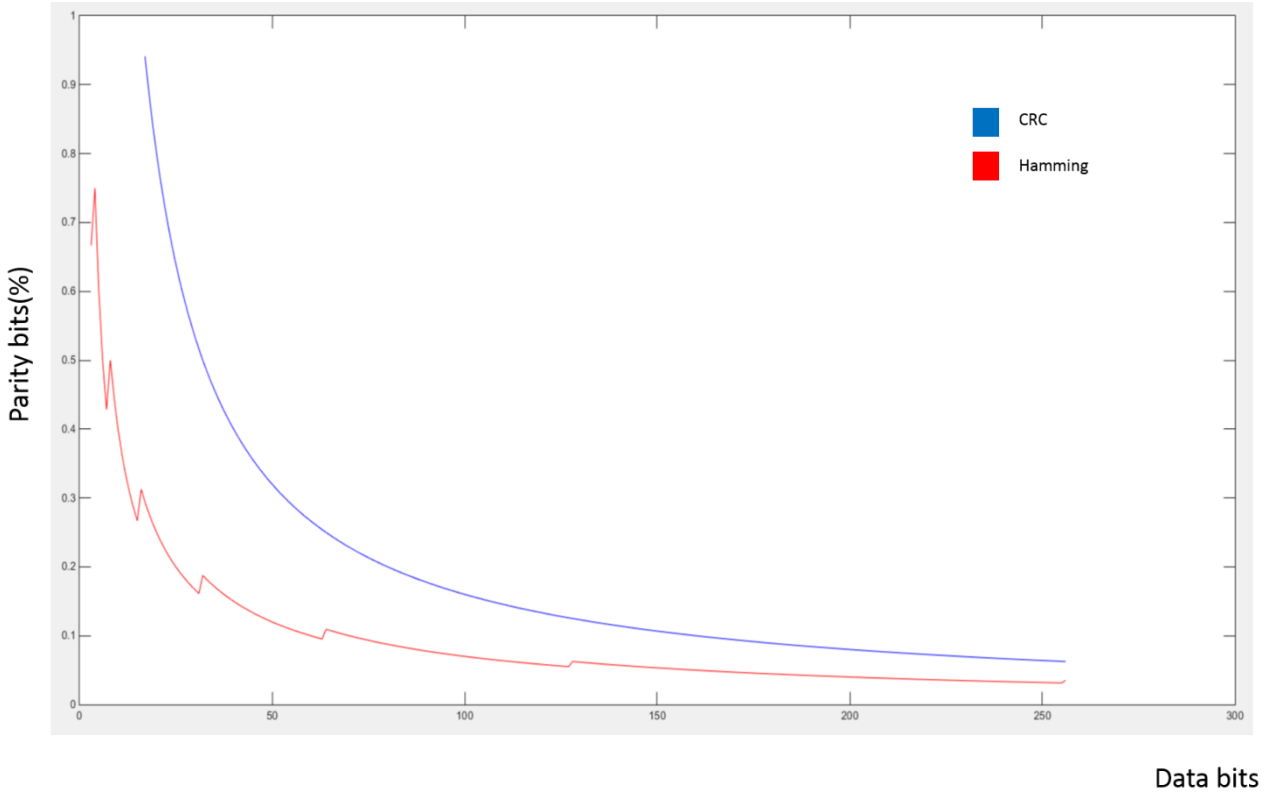

Figure 19. Graph of CRC versus Hamming code

The graph shows that the efficiency of Hamming has peaks for every power of two, but is always better than CRC. The exact point where CRC becomes more efficient than Hamming codes is at 65 536. That is 8 kB of data, which is probably too large for 16 bit CRC to be effective.

In terms of operations, it is easy to predict the behaviour of Hamming codes. In the worst case, each parity will check half of the bits sent (data bits + parity bits). This is shown in the following formula:

$$bits\ checked = \left(\frac{sentbits}{2}\right) * paritybits$$

This means that at 16 bits or 2 bytes of it will check 2.5 times the data, and at each power of two it will increase by 0.5.

On the other hand, CRC has a constant worst-case checking factor, since it will have to compare 16 bits in the data with the 16 bits of the polynomial. This means that it will check 32 bits at each step. This is however deceiving since the XOR operation is an instruction in x86 architectures, meaning it

will only be one operation per step. In practical terms, CRC will in the worst-case have to do one operation for each data bits:

$$Operations_{CRC} = databits$$

From this we can conclude that CRC-CCITT will be much more efficient in regards to the amount of operations it has to do, though you would have to have very large datasets in order for the Hamming code performance to be slow.

It also needs to be noted that this version of Hamming codes can only correct one bit errors, and is usually used in applications where a retry mechanism (resending package) is either not possible, or will affect the system. We can additionally also extend the Hamming code to be able to detect two-bit errors, though it will not be able to correct them (SECDED).

While our project can easily implement a retry mechanism, we will implement Hamming codes in the simulation communication, in order to get acquainted with the concept in practice.

For the communication with the drone, we use a much simpler concept. Since the radios are half-duplex, we simply send the message back to the sender as received, so it can compare what it receives with what it sends to establish if an error happened.

3.3.4 Hamming code implementation

Hamming code was implemented both in java as well as in C++.

The first thing to define would be the encoding of instructions in terms of bytes. Since the drone can move in 8 directions, then it would be sufficient to have a 3 bit representation for each instruction. This however was not true, since we also needed a null instruction, so we needed 4 bits per instruction. As the analysis before shows, Hamming is more efficient just before the multiple of 2, for example 15 bits or 31. The problem is this was not possible using data that is a multiple of 4. For example, 12 bits data caused 17 bits sent and 5 parity checks, while 11 bits only required 15 bits and 4 parity checks. With this in mind, it was decided to have all instructions start with a 0 bit, and messages would contain between 1-3 instructions. This means that all messages would start with 0, so we did not need to send this bit, or verify it with Hamming codes.

This means that the message will have the following structure:

Figure 20. bit processing 3 instructions

The C++ Hamming class is an object which receives the data string, and returns the string that must be sent via the serial communication (data and parity bits). The java hamming class verifies the integrity of the message, and removes the parity bits, returning the data bits only.

3.4 X-Bee

For the communication between the Quadcopter and the computer that is processing the map and gives the instructions for prototype movement, two X-Bee's modules where used. The specific model used was the XBee Series 2(ZigBee Mesh). This model combine with the "X-Bee Explorer USB" shield allows a direct communication with the computer with just a USB cable. And on the quadcopter side, a "XBee Explorer Regulated" shield was used and connected to the Arduino Uno. To be able to use two X-Bee's together some settings are necessary. One of the module needs to be set as a "Coordinator" (the one connected to the computer) and the other one will be the "end device". The two modules are binded together using the Coordinator serial number and copying it to the end device. They need also a common ID so they be part from the same "node". After both X-Bee are connected, the one from the computer with the USB cable, and on the quadcopter with the Voltage, Ground, Data In and Data Out. The computer can send, through simple serial communication data to the X-Bee connected to the Arduino. The Data In and Data Out from the X-Bee is connected to the TX and RX on the Arduino making this a simple serial communication. The data is received byte by byte and pushed to a queue for later usage.

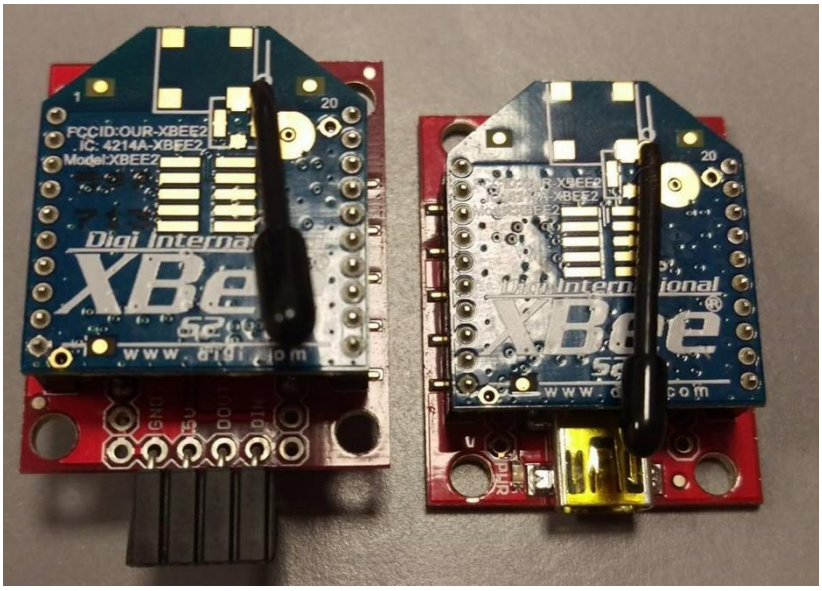

Figure 21. Quadcopter X-Bee Module(Left) Computer X-Bee module(Right)

The x-Bee serial is receiving from the computer bytes that are read with the Serial.Available() command and transformed after to char for easy usage.

3.5 Collision avoidance

If more than one drone is to be connected to the system, then it is important that we consider that there is a chance of collision happening, unless we design our system to avoid collisions.

The simplest form of collision is when two drones try and communicate to the Ground Control Station. This is be shown in the following picture:

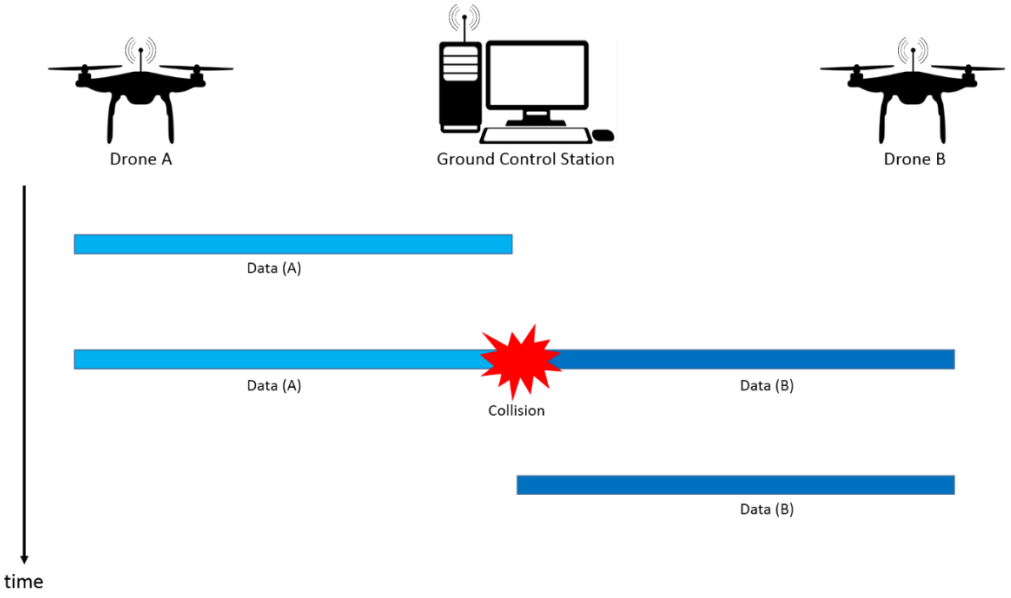

Figure 22. Simple collision

3.5.1 CSMA

One of the simplest forms of preventing collisions is using carrier sense multiple access (CSMA). What this means is that the sender will listen to the channel, and only send if it is empty as displayed by next figure:

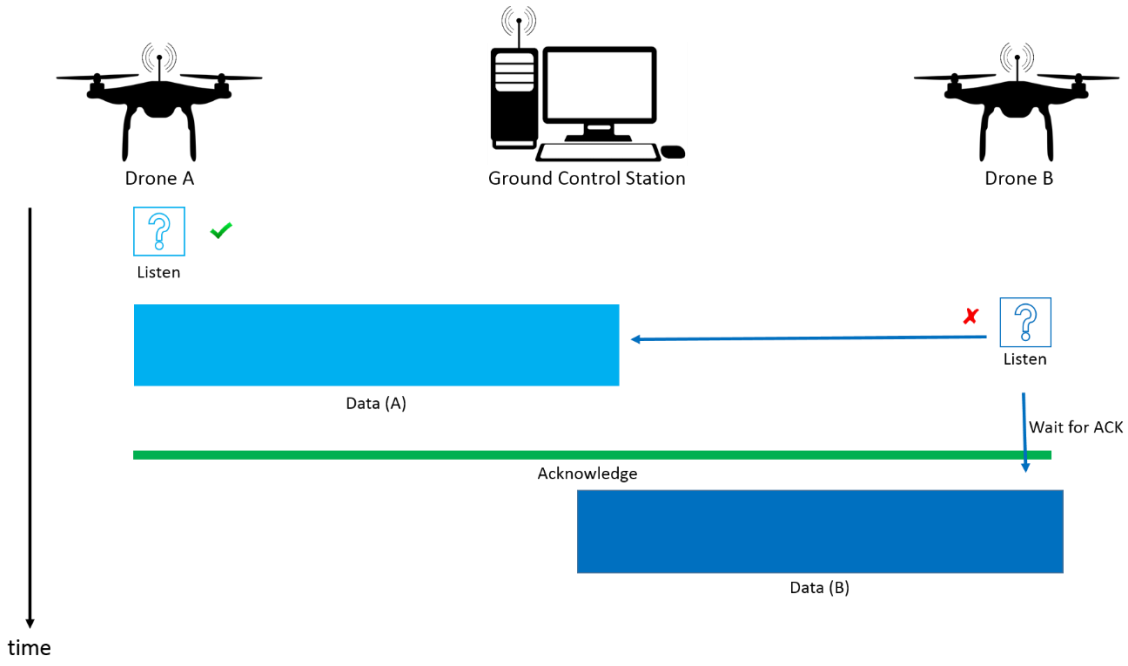

Figure 23. CSMA collision prevention

Waiting for acknowledgment is one variation of CSMA (1-persistent), which has the advantage that there will be no lost time in between signals, but if more than one drone is waiting to send, then it will cause collisions, since both drones will assume it is their turn to communicate. Another variation includes issuing a random delay (0-persistent) and then checking if the channel is free, which makes it harder for two signals to start sending simultaneously. In terms of how exactly CSMA works, it follows the following flow chart:

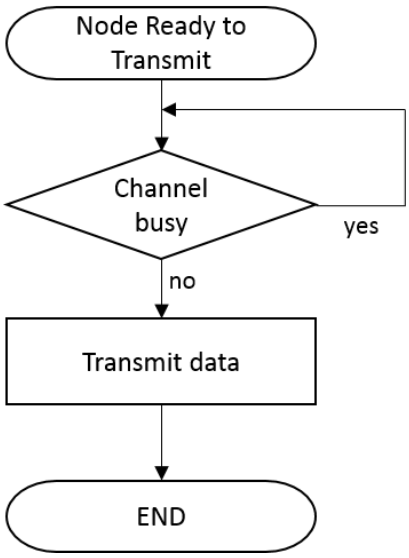

Figure 24. CSMA (0-persistent) flow chart

Unfortunately, there are two main issues with CSMA, the first one is that as the distance of the communication increases, so does the delay between when the signal is sent and when other nodes detect it, meaning that collisions will still happen as shown in the next diagram:

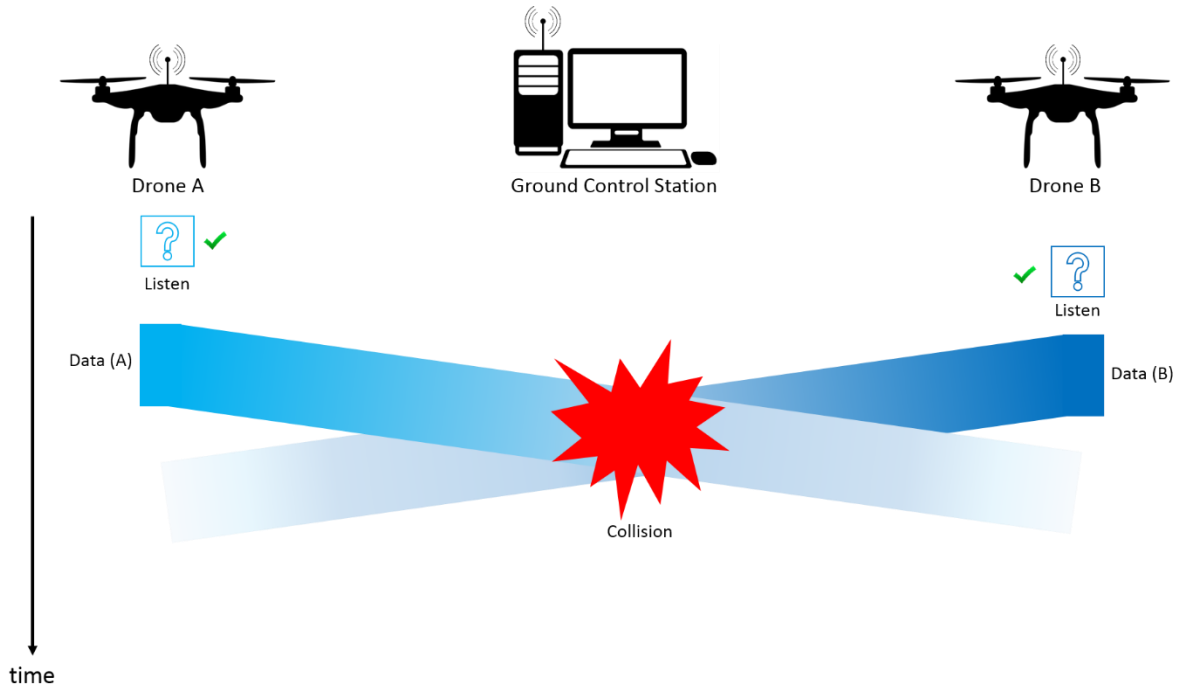

Figure 25. Collision caused by delay in signal

The second issue is an even more complex problem, called the exposed and hidden node problems.

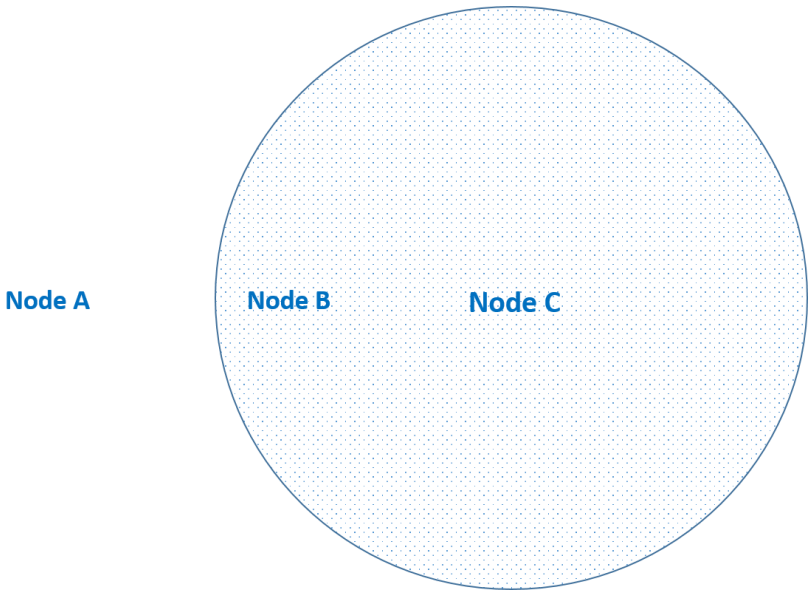

Figure 26. Exposed and hidden node setup

As we can see in the previous figure, assuming the circle is Node C's range, then Node C and B can be busy because of C talking to B, but Node A thinks Node B is available, and will starts sending simultaneously causing collisions. This is called the hidden node problem. Additionally there is a

scenario where C is sending to another node, and B assumes A is busy, although it is not. This is the exposed node problem.[32] [33]

Next subsections will discuss modifications to CSMA in order to avoid some of these issues, as well as other possibilities without CSMA.

3.5.1 CSMA/CD

CSMA with collision detection (CD) is a variation of the protocol. It deals with detecting collisions as fast as possible, to avoid wasting time. While 0-persistent and 1-persistent CSMA will send the entire package, CSMA/CD checks for collisions at every sent byte, like the following flow chart shows:

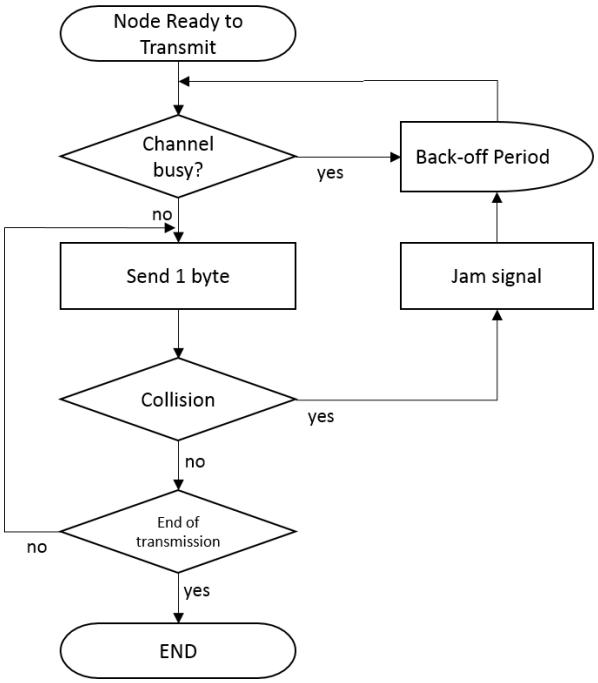

Figure 27. CSMA/CD flowchart

If a collision is detected it will jam the signal, so all other nodes can detect the collision. While this saves time in case of collisions, it will not avoid the exposed or hidden node problems, which will still cause collisions or innefficiency. In addition, wireless networks can usually not send and receive simultaneously, which means a node will not be able to detect collisions.[34]

3.5.2 CSMA/CA

CSMA with Collision Avoidance (CA) implements the mechanism described in the standard 802.11. It relies on Request to Send (RTS) and Clear to Send (CTS) messages to control traffic, and will avoid many collisions. The flow chart of the protocol is the following:

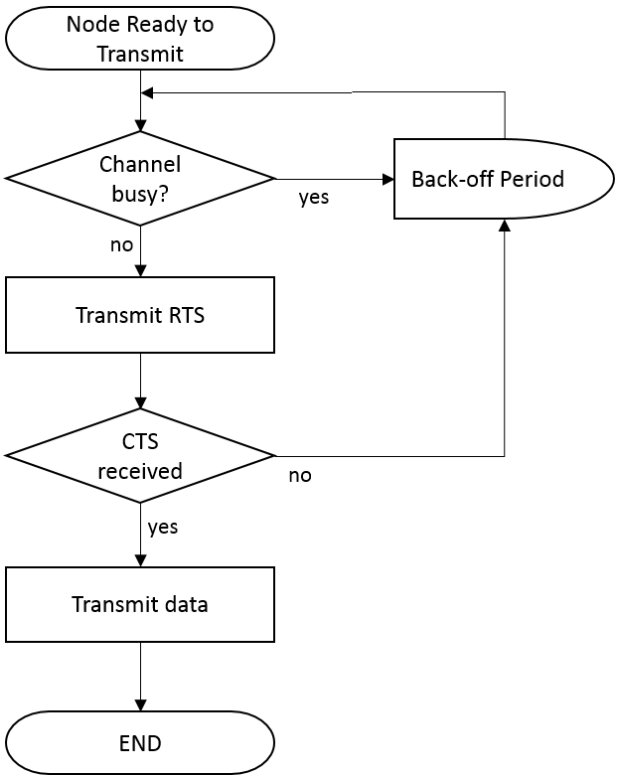

Figure 28. CSMA/CA flow chart

In terms of how it would work in practice, it is shown in the diagram below:

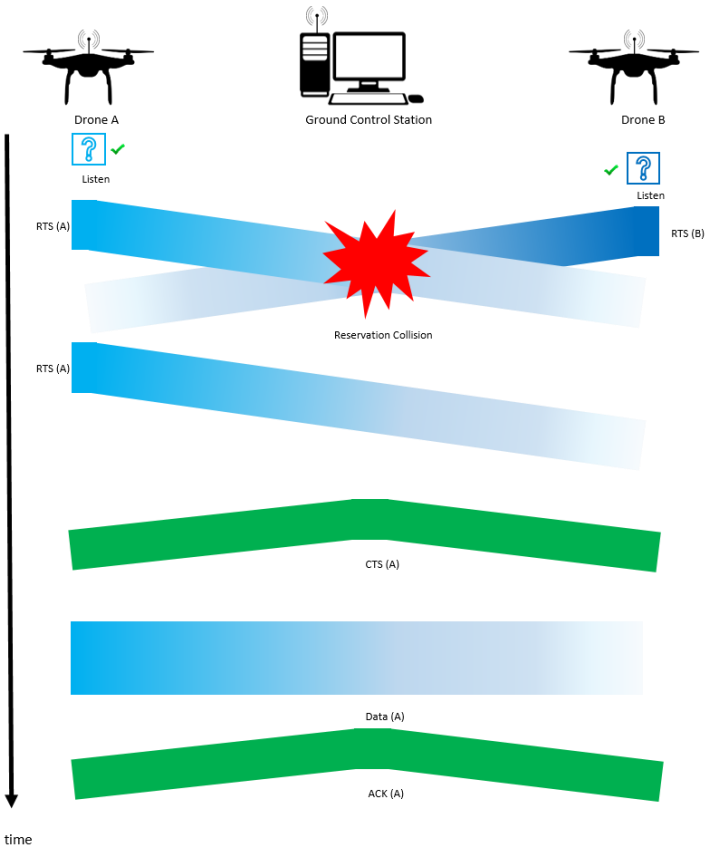

Figure 29. CSMA/CA with a RTS collision

As the diagram shows, you would still need collision detection for collisions in RTS, and possibly anywhere else, caused by colliding RTS messages with data in a hidden node context. Additionally, other things need to be addressed such as multiple GCS systems, and keeping the drones in range, which could make the communication much more complex.[34] [35]

3.5.2 Non-CSMA solutions

We will analyse 3 alternatives to CSMA protocols, and discuss their viability.

The first and perhaps most common solutions is using channel partitioning also known as Time Division Multiple Access (TDMA). This means that each node is allocated a time period. This is very efficient at high loads, since there will be no free time, but at lower loads it will be inefficient, since channels that need to send must wait for their time allocated, even if other channels do not need to send. Also, adding or removing nodes requires a redesign of the whole system, and reconfiguring of all nodes. The system will for example work the following way for three nodes.[36]

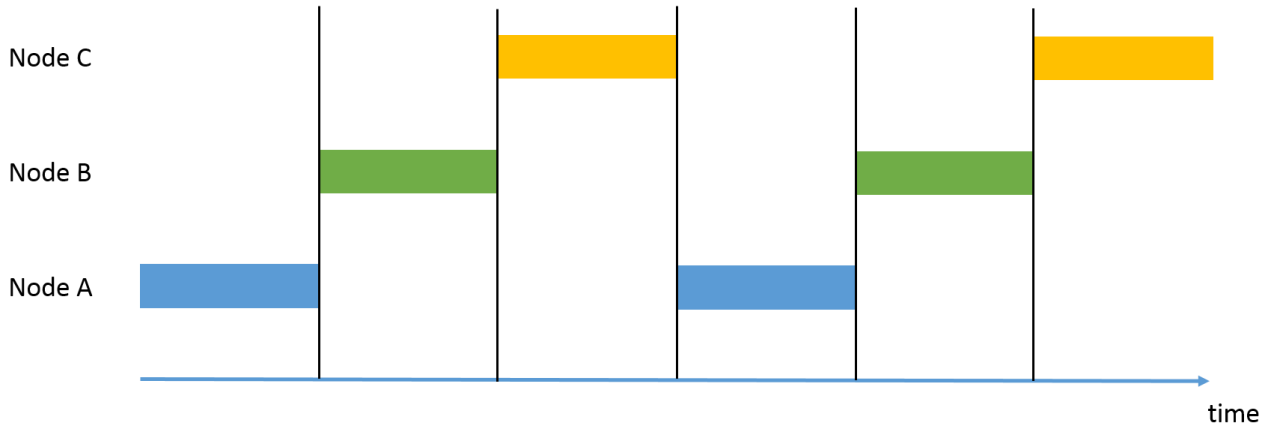

Figure 30. TDMA system with 3 nodes

The second option would be having a system where the master, which is GCS does polling on the drones. This means if a drone does not need to send, it will only reply with a short message, and the master will move on to the next node. This means that adding or removing nodes will just require changing the master node (GCS), assuming you still have addresses free to allocate more nodes. While this might seem like a good solution, it has the disadvantage that it will have an overhead because of the polling, there will be a latency and the architecture has a single point of failure in the GCS. This last point is however not a big issue in our case, since the system by definition requires the GCS to be working. The system will work the following way for a master node and three nodes.[37]

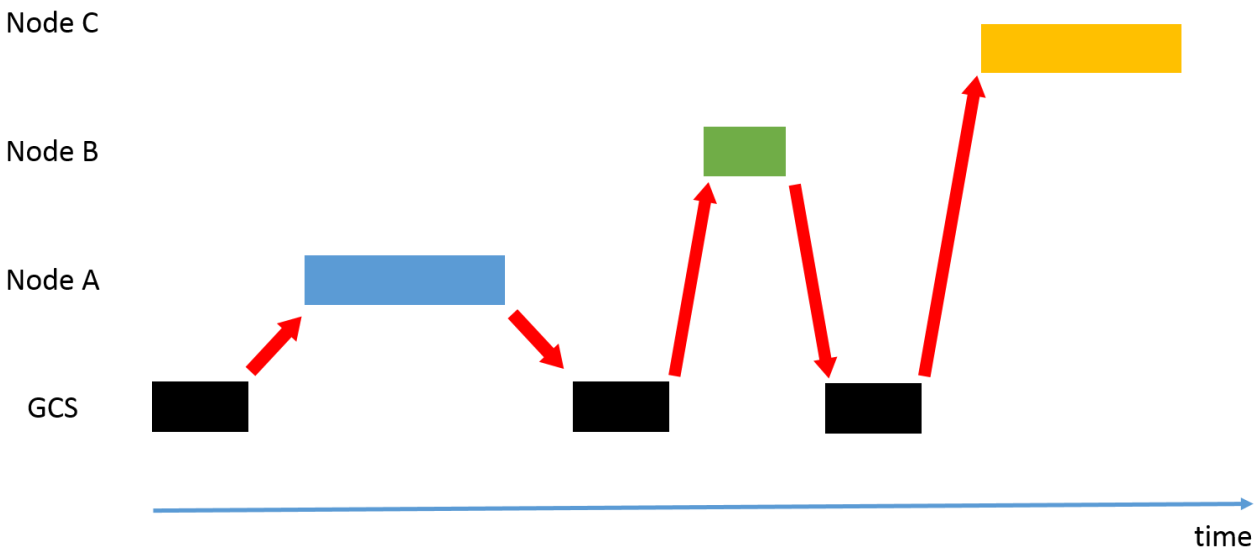

Figure 31. Polling system

The last option analysed is token passing. This option relies on nodes passing a token in order to send data, and when the communication is done, they will pass the token, as the figure shows.

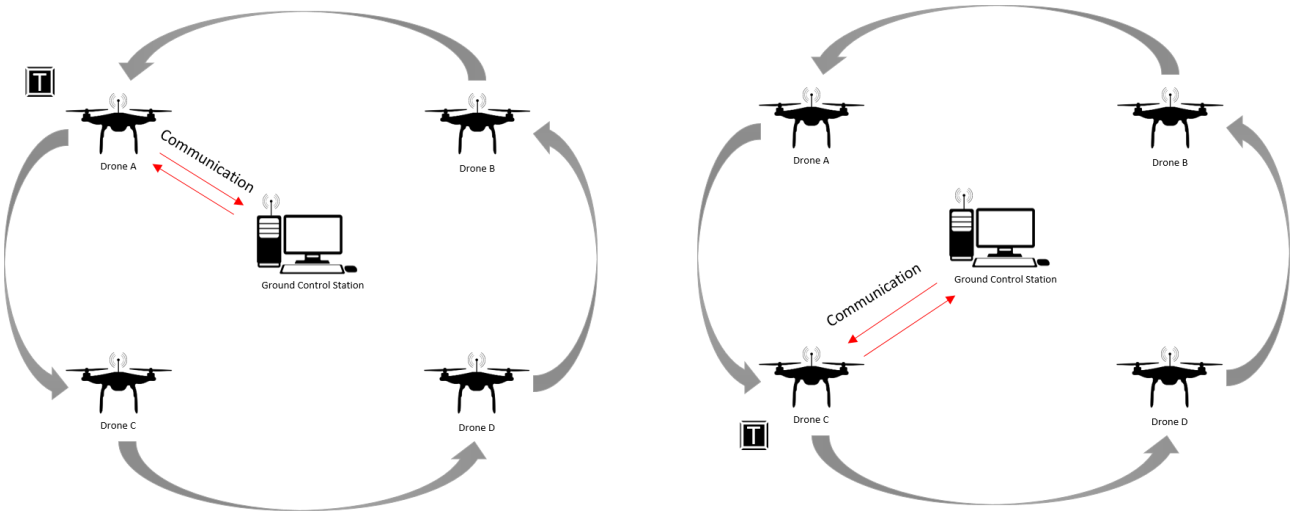

Figure 32. Token passing two steps

This option has very similar problems as polling does, overhead because of token sending, latency and a single point of failure in the token. For example if a node with the token fails to send it to the next node, the system will stop working.[38] [39]

3.5.3 Conclusion

In conclusion, there are many way to detect and avoid collisions in wireless systems, and they all have different weaknesses and strengths. Our system does not require messages to be sent very fast, so we are only interested in avoiding collisions and resending messages that fail.

With this in mind, TDMA seems like a great solution, since it will avoid collisions, and we do not care about efficiency. Nevertheless, if the systems controls many drones, then the time allocated to each drone might be too low, so a polling solution could be a viable solution, or perhaps CSMA/CA.

Regarding the hidden and exposed node problems, it is hard to solve, since drones will be dynamically moving, therefore causing changing conditions. Another issue by extension could be getting out of range of GCS, and a clever way to deal with it is a similar way to how MAVLINK does it. A heartbeat message is sent by the master, and in case a drone does not receive the heartbeat, it is out of range, and should for example land or retrace its steps till it encounters the heartbeat again.

4 Solution

This chapter will present the overall view of the solution. The developed system consists of the following components:

1. GCS with implemented GUI and algorithm.
2. Drone prototype and remote control for it.
3. A simulation machine.

The solution processes different parts of the task at hand at each node, until it gets carried out in the final node. The chapter will start with describing the connections in the system, followed by a subchapter on the A* algorithm. Afterwards it will describe the GCS program and simulation programs in their own subchapter. The drone part of the project will be explained in depth in the last subchapter.

4.1 Solution description

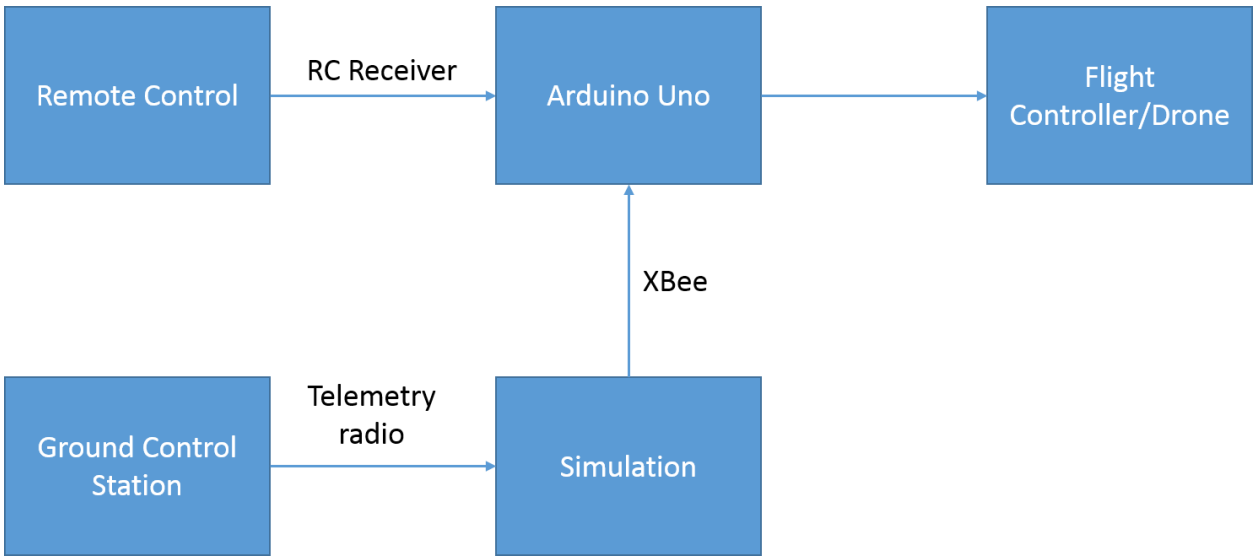

Figure 33. System Diagram

The diagram above simply describes how the system is distributed and how each component of this distributed system communicates with its counterparts. The GCS also called the Ground control station is the main control component in this distributed system. The GCS runs the main GUI and algorithm program, which will calculate the optimal path, and work as a master. This device can communicate via telemetry radio with another device, which is running the simulation of the drone performance. The computer or device which contains the simulation software can communicate directly with Arduino Uno which is basically used to forward remote control messages and send virtual remote control commands via PWM (pulse-width modulation) signals to the flight controller in order to control the quadcopter. This is done to avoid modifying the flight controller software.

4.2 A* search algorithm

This section will describe the algorithm, whose purpose is to find the shortest path for the prototype on discretized space. The choice of A* has its reasoning in its wide use in AI development (for example, in computer games) and is known for its superior performance in pathfinding[40].

A* is an algorithm dealing with graphs. In the current project, a graph is composed with respect to a node and its neighbours (pointers to nodes in the graph). Another thing which is worth mentioning is conversion. Space is discretized considering two dimensions. These are latitude and longitude, so altitude is not covered. That is why it was efficient to use two-dimensional array mapping of the grid, which implies known **[i]** and **[j]** notation. Since the algorithm is carried out in C++ programming language, the graph is represented by **std::vector** type variable (one dimension, say, **[k]**), which the array was converted to by applying a certain formula to current coordinates (i, j). Thus,

$$k = i * amount\ of\ columns + j$$

This is also a best-first search algorithm, which means that the next node to be processed is considered best according to a specific rule. The rule will be described later in this chapter. Furthermore, the current implementation exploits heuristics – a preliminary estimation of distance between a particular node and a finish node. There are three types of heuristics available in the current version of the program: Manhattan, Diagonal and Euclidian. Some optional parameters (for instance, enabling diagonally-adjacent nodes consideration, breaking-ties) are also included. All these topics will be discovered in details.

Basically, the main idea of A* is built around summation of two functions [40].

$$f(n) = g(n) + h(n)$$

Where h(n) is the heuristic estimation of cost from a particular node to the goal node. In its turn, g(n) is the cost from the current node back to the starting node. The important fact is that g(n) is not calculated arbitrary, but considering the whole path from the starting node to current node, which the algorithm keeps track of.

4.2.1 Distance to start calculation.

The initial g(n) cost values are 10 for vertical/horizontal and 14 for diagonal. It is clarified in the picture below. Notation is given.

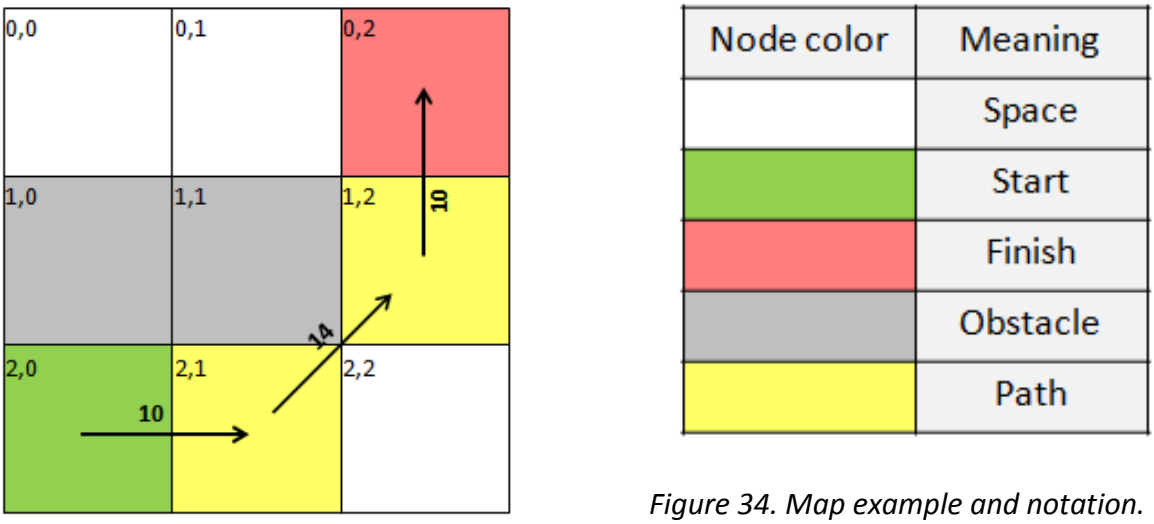

Figure 34. Map example and notation.

Cost coefficients are not chosen at random. As it is possible to notice, that line of cost 14 is a hypotenuse in an isosceles right triangle with other sides equal to 10 both.

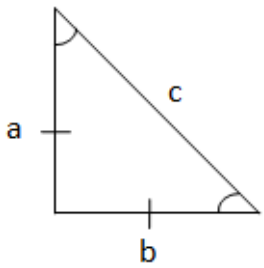

Figure 35. An isosceles triangle.

An isosceles triangle has the following properties.

$a = b$

$c = a * \sqrt{2}$

So, **a** and **c** coefficients are selected in such a way, that $\frac{c}{a} = \sqrt{2}$. With **c = 14** and **a = 10**, it results into 1.4. Now this property of an isosceles right triangle is considered and can be used, if necessary. For better precision, larger scalars could be used. For instance, **c = 577** and **a=408**, resulting into about 1.414216, which is closer to the square root of two.

In the map in the figure above, g(n) for Node[0][2] is 10+14+10 = 34. If diagonal movement was not allowed, then Node[2][2] would also be included into the path, and g(n) for finish node would be 10+10+10+10 = 40.

4.2.2 Heuristic estimation.

As it was stated, heuristic estimation is an assumed distance from a particular node to the goal node, so in the actual code it is known as distance to finish. Unlike g(n), distance to finish (noted as h(n)) is measured in units equal to changing coordinates. Three types of heuristics are available in the program.

Manhattan distance takes into account 4 possible directions (2 horizontal and 2 vertical) and is considered a standard approach on a square grid. Manhattan distance is precomputed in the following way:

$di = abs(node.i - goal.i)$

$dj = abs(node.j - goal.j)$

$manh = di + dj$

Where **di** is the difference in [i] between a particular node and the goal node and **dj** is the difference in [j] between the same nodes.

Diagonal distance is used when diagonal movement is allowed in the map, which implies eight possible directions (2 horizontal, 2 vertical, 4 diagonal). As expected, the program has its features when implemented in real life, different from virtual space. The first issue is that diagonal movement is dangerous, because the drone can simply touch an edge of an obstacle, which would disturb its

movement and, worst-case scenario, cause a crash. Another thing is even more problematic. Consider a part of a map.

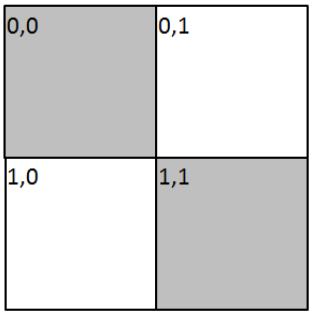

Figure 36. An impassable obstacle.

Even if in virtual reality a leap from Node[1][0] to Node[0][1] is acceptable, it is still not a thing in the real world, because Node[0][0] and Node[1][1] have a common edge in the middle of the map fragment. Thus, diagonal movement from Node[1][0] to Node[0][1] would not be physically possible for a drone, unless a third dimension (altitude) is included.

Back to the algorithm, diagonal distance is calculated like this (using Chebyshev distance):

$$di = abs(node.i - goal.i)$$
$$dj = abs(node.j - goal.j)$$
$$diag = (di + dj) + (1 - 2 * 1) * min(di, dj)$$

Where **min(di,dj)** returns the smallest change.

At last, Euclidean distance calculation is also carried out. According to the given article 40, one can benefit from Euclidean distance, when any angle movement is allowed. Even though it is not realized (directions are predefined), it could be possible. The author, however, warns that g(n) function would be devaluated in case of any angle movement: it does not match the other calculations anymore. Euclidian distance is calculated according to the formula below.

$$di = abs(node.i - goal.i)$$
$$dj = abs(node.j - goal.j)$$
$$eucl = sqrt(di * di + dj * dj)$$

As it is possible to notice, Euclidean distance calculation involves square root computation, which is a demanding procedure. The author warns us again, that squaring the Euclidean distance (avoiding square root computation) is not a solution: "*This definitely runs into the scale problem. The scale of g and h need to match, because you're adding them together to form f. When A* computes f(n) = g(n) + h(n), the square of distance will be much higher than the cost g and you will end up with an overestimating heuristic. For longer distances, this will approach the extreme of g(n) not contributing to f(n), and A* will degrade into Greedy Best-First-Search*".

Nevertheless, square root calculation can potentially be simplified in the current program. Instead of using standard **sqrt()** from C++ Math library, a custom function was written. The function is Taylor series expansion for $\sqrt{N^2 + d}$.

$$\sqrt{N^2+d} = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)! d^n}{(1-2n)n!^2 4^n N^{2n-1}} = N + \frac{d}{2N} - \frac{d^2}{8N^3} + \frac{d^3}{16N^5} - \frac{5d^4}{128N^7} + \cdots$$

This method of calculating square root is iterative, so by changing the amount of iterations in the loop it is possible to control the precision and converge further towards the correct result. The mathematical aspect of the method is covered in 2.4.

4.2.3 Breaking ties technique.

It is a common phenomenon for a map to have several optimal paths of the same length to the finish, caused by the same f(n) value. This means, that more nodes will be processed by the algorithm, but no benefits will be gained – the path will not be shorter. It is not very easy to notice on a relatively small map, but when dealing with large input data, performance could become a key factor. An example will be described in the algorithm development further steps section.

The solution of considering paths of the same length is to force f(n) to change. As is was stated earlier, f(n) is a summation of g(n) and h(n). It is not reasonable to change g(n), because this function is precise – no matter what, vertical/horizontal or diagonal movement costs stay the same. Function h(n), in its turn, is a prediction, a pre-calculated estimation. Thus, tie breaking becomes a heuristic modification technique. It is applied to already calculated heuristic like this:

$$heuristic \ *= \ (1.0 \ + \ p)$$

Where factor **p** is not an arbitrary number. Note, that the factor should be chosen in such a way, that it does not push a smaller f(n) towards being equal to a larger f(n). In other words, breaking-ties technique is only for nodes with the same f(n), making it slightly different, and should not affect nodes with different f(n) functions. As the author of the article states,

$$p < \frac{minimum\ step\ cost}{expected\ path\ length}.$$

So, if in heuristics one step costs in heuristic at least 1 and overall path not supposed to have more than 1000 steps, **p** is equal to 0.001 as it grows with expansion of a graph. As it could be concluded, the factor value is dependent on the longest path value, which is linked to map size and of course obstacle location. The longest path analysis was discussed in chapter 2.5. Deeper analysis of these relations and developing a formula for the factor could be one of further steps. A remark is that the factor should be greater than machine epsilon in order to have an effect – make a difference, if a machine can process a path of $\frac{1}{\epsilon}$ step length of course.

Another way of applying breaking ties technique is "to prefer paths that are along the straight line from the starting point to the goal"[40]:

$$di1 = current.i - goal.i$$
$$dj1 = current.j - goal.j$$
$$di2 = start.i - goal.i$$
$$dj2 = start.j - goal.j$$
$$cross = abs(di1 * dj2 - di2 * dj1)$$
$$heuristic \mathrel{+}= cross * 0.001$$

As the author continues, "This code computes the vector cross-product between the start to goal vector and the current point to goal vector". This implementation was used in current algorithm, because it demonstrated higher performance. However, with this code applied to heuristics, a path tends to stay along the straight line between start node and finish node. This behaviour can be strange and even lengthen the path.
This behaviour is shown in the following figure.

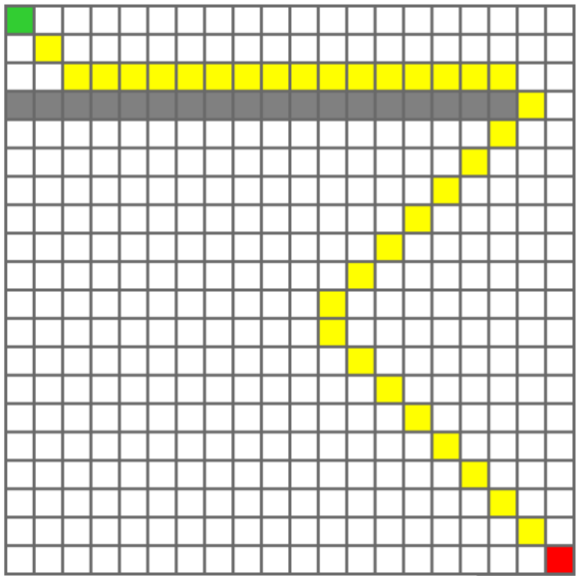

Figure 37. Breaking ties trade-off

This image shows how breaking ties in specific situations can be a trade-off by reducing considered nodes, but increasing path length.

4.2.4 Data structures.

The whole procedure involves performing operations on data storages. There are two data storages, where nodes are put. Open list contains nodes waiting to be processed. On the other hand, already processed nodes go to closed list. In C++ implementation of algorithm, both lists are represented as **std::vectors<Node*>** and contain pointers to nodes of the graph, as the programming language allows. The selection of **vector** over other data structures was mainly motivated by complexity and C++ compiler (MinGW, to be precise) specifics.

As it will be explained later in the chapter neither open list nor closed list are sorted– nodes are added to lists as the graph is being explored. There is no need to make a regular rule for ordering and insert a node in a particular place in **vector**. That is why nodes are added to the end of a **vector**. It is performed by calling **vector.push_back(node)** command, which has O(1) complexity. Removal from the end has O(1) complexity too[41]. Moreover, random access also has O(1) complexity. At last, insertion or removal of elements is linear in distance to the end of the **vector**, so complexity is O(n). These circumstances fit our case, where **vector** in C++ can be compared with **arraylist** in Java.

A **vector** could be picked over a **list,** because, as C++ creator Bjarne Stroustrup says in the presentation in C++11 style 42, vector is a default sequence of elements in C++, stating a **list** to have slower traversal and using more memory.

Unlike Java compiler, C++ compiler does not allow to declare an array of unknown size. A size of an array should be strictly determined at compilation time. Nevertheless, we do not know the size of storages in advance, so a resizable data structure is required. In this sense, **vector** is a suitable option.

4.3.5 Actual algorithm.

The algorithm begins with calling calculation function with the starting node as a parameter known as current node. First, its neighbours are obtained. Neighbours of a node are its adjacent nodes of any type, but not obstacles. The algorithm goes through a list of neighbours, which has a maximum size of eight, when diagonal movement is enabled. If goal node is adjacent to current node, then finish is reached and path is found. If there was no finish node in neighbours, then each single neighbour node goes through the following process:

1. If the neighbour node was already processed earlier (is in closed list), then do nothing.
2. If the neighbour node is not in open list, it is the first time that node has been discovered. Add it to open list. A pointer to the current node is written in the neighbour node object. Thus, current node is a parent of the neighbour node. Distance to finish (heuristics) and distance to start are calculated for the neighbour node afterwards. So, function g(n) = (g(n) of current node) + (distance from current node to the neighbour node, it is the cost of edge connecting them).
3. If a neighbour node is already in open list, it means the node was discovered before and has its parent and calculated g(n). Now, it is necessary to check, whether the new path, which the neighbour node was reached by, is shorter than the older path. If (g(n) of current node) + (distance from current node to the neighbour node) is smaller than current g(n) of the neighbour node, it implies that the new path is shorter than the old one. We rewrite the parent of the neighbour node to the current node and assign new g(n) to the neighbour node.
4. Open list now likely has elements and is ready to be gone through. If open list is empty, then a path from the start node to the goal node does not exist. The node with the smallest f(n)

= g(n) + h(n) is found, removed from open list and added to closed list. Calculation function is called recursively with this node as a parameter, repeating these instructions for it.

A plain text explanation of the algorithm could be difficult to comprehend. A flowchart of the calculation process is offered below.

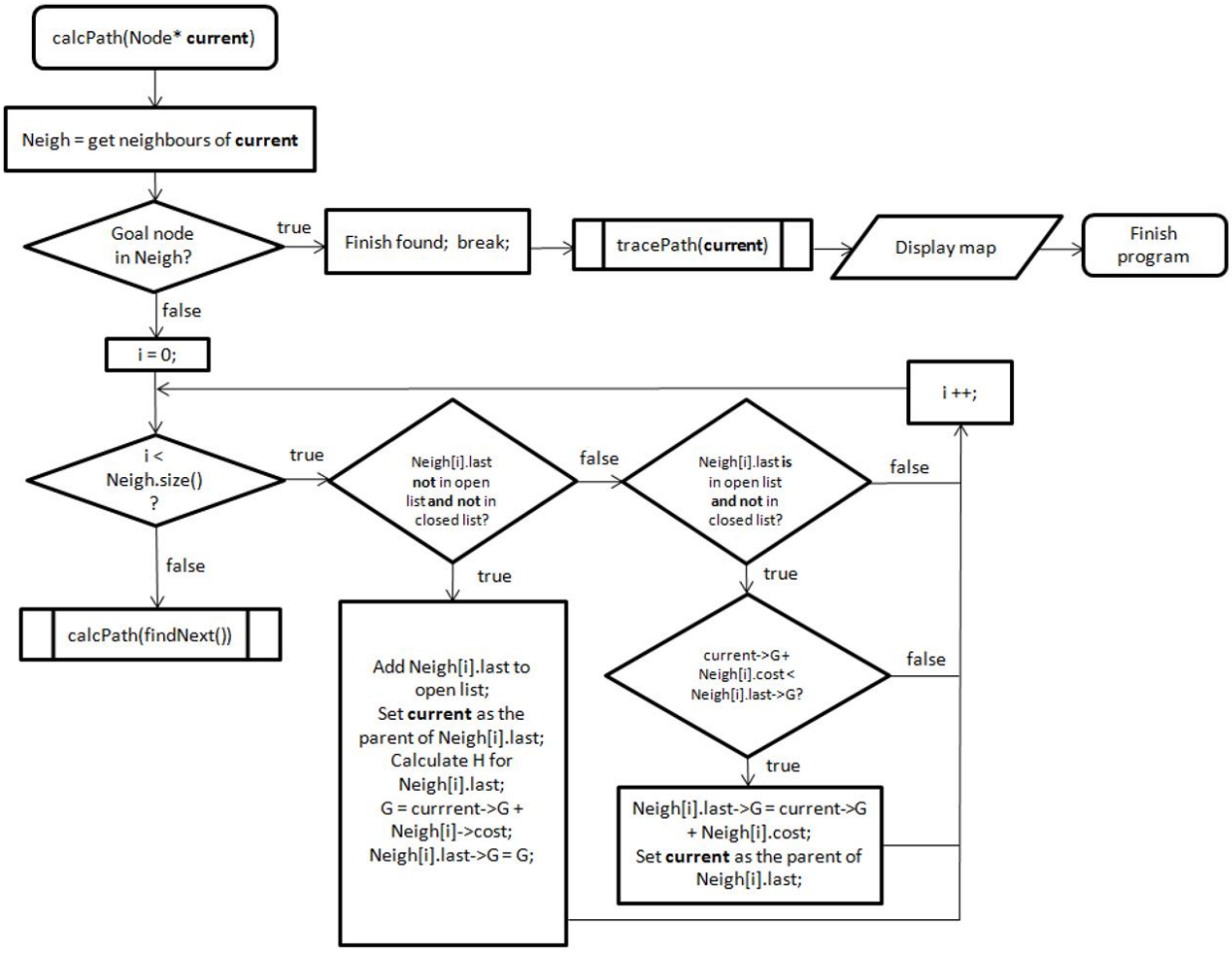

Figure 38. Algorithm flowchart.

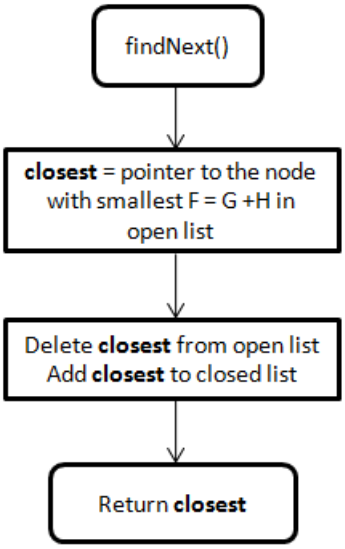

Figure 39. Flowchart continued.

The provided flowchart explains the work of the algorithm's main calculation, it is **calcPath(Node* current)**. A single remark is about neighbour data storage, which contains edges, having member variables for an actual neighbour (**Node* last**) and for cost of movement (g(n)) from current node to a neighbour (**int cost**). All member variables and functions will be mentioned further.

4.3.6 Detailed description.

Let us first take a look at UML class diagram of the whole program. Note that some variables or functions might be kept for testing purposes.

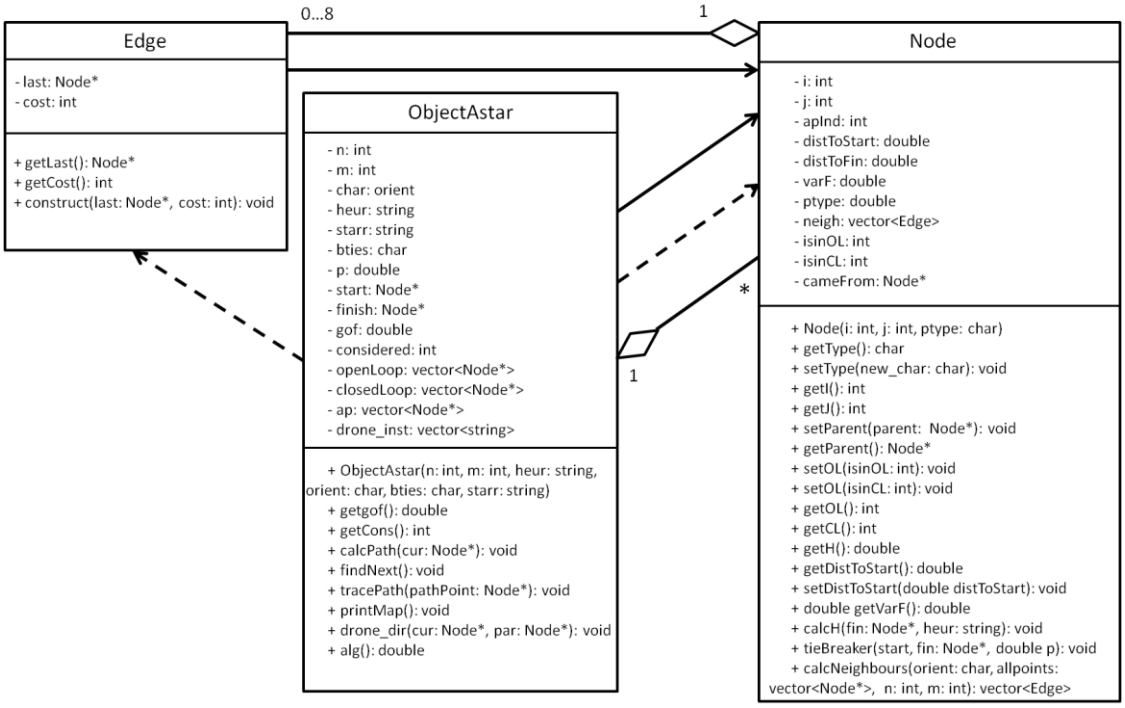

Figure 40. Algorithm UML diagram.

The figure above supplies the reader with class member functions as well as with relations between these classes. However, the point of some functions are not clear from their names in the first place and, in addition, there are some local functions, which normally are not included in a UML diagram. A short description of some functions is provided below to make sure the functions are explained. Since the point of getters and setters is known from object-oriented programming paradigm, they are mostly not included in tables.

Class	Function	Remarks on parameters	Description
Node	getOL()/getCL()	See UML.	A function for a quick check of a node being in a list, not to iterate through the latter.
Node	calcNeighbours()	char orient: diagonals on/off vector<Node*> allpoints: the graph int n, int m: sizes of a map (not to get out of bounds)	Obtains neighbours of a node.
Node	long double fact()	double fnum	Two **local** functions for custom calculation of a square root.
Node	long double sqroot()	int number	
ObjectAstar	ObjectAstar()	Introduced by a user.	A constructor used to create an object with user input in GUI.
ObjectAstar	alg()	See UML.	Translates a string representation of a map to an actual graph with start and goal nodes assigned.
ObjectAstar	tracePath()	See UML.	Traces the path from finish to start by accessing parents of nodes.
ObjectAstar	drone_dir()	See UML.	Converts the path to a set of directions for the prototype to move in.

Figure 41. Supplementary function description.

4.3.7 Fails, improvements and further steps in algorithm development.

The most obvious thing to mention is orientation in space. As it was stated earlier, the current program for the prototype considers only two dimensions – latitude and longitude. Altitude was not taken into account. If to take a pragmatic look at the problem of path finding, it could be enough to avoid obstacles by handling altitude only since an aerial vehicle allows it. A work-around is to round any obstacle atop. Nevertheless, it is a brute-force solution. In addition, it can cause difficulties indoors, where the space above an obstacle is not guaranteed to be free and, at last, some environments can require a drone to keep a constant altitude.

However, it is possible to find a situation, where going up is an efficient and justified action. It happens, when rounding an obstacle atop makes the whole path shorter than if the obstacle was avoided on right or left side. It goes without saying, that involving altitude in the algorithm is a solid further step.

Such an addition would make a serious impact on memory and data structures in particular. The grid becomes three-dimensional, implying cubic increase of nodes in a cubic room. This grid should still be represented by data holders. An **array** of **arrays** of **arrays** could be used with a new formula developed for 3D **array** to 1D **vector** conversion. The amount of neighbours a node can have grows too. In a three-dimensional map a node can have maximum 6 adjacent nodes with horizontal/vertical movement only and 26 neighbours with full diagonals enabled – versus 4 and 8 nodes in a two-dimensional map respectively. The current version of program simply iterates through the neighbours of one node, when looking for the goal node among them. So, it takes **n** comparisons to find the goal node at the n-th position. It is not efficient to attach some kind of advanced algorithm when dealing with only 8 elements maximum. An increase to 26 elements can change the situation a bit. A more efficient method can be carried out, for instance, simultaneous exploring of a vector from its start and from its end via multithreading declaring the data storage as a shared resource. Regardless to the circumstances, a similar approach can also be chosen over iteration when finding a node with the smallest f(n) in open list. Making list sorted, instead of iterating through the whole list, could be another improvement.

As it has been mentioned in the testing section, the program crashes at map size about 80 x 80 with tie breaking off. It was logically supposed (because the language is type unsafe), that the reason of

this behavior is stack overflow, taking into account recursive nature of A* algorithm. To be precise, calcPath(Node* cur) function eventually causes a call of itself. To understand the issue, application memory management must be comprehended.

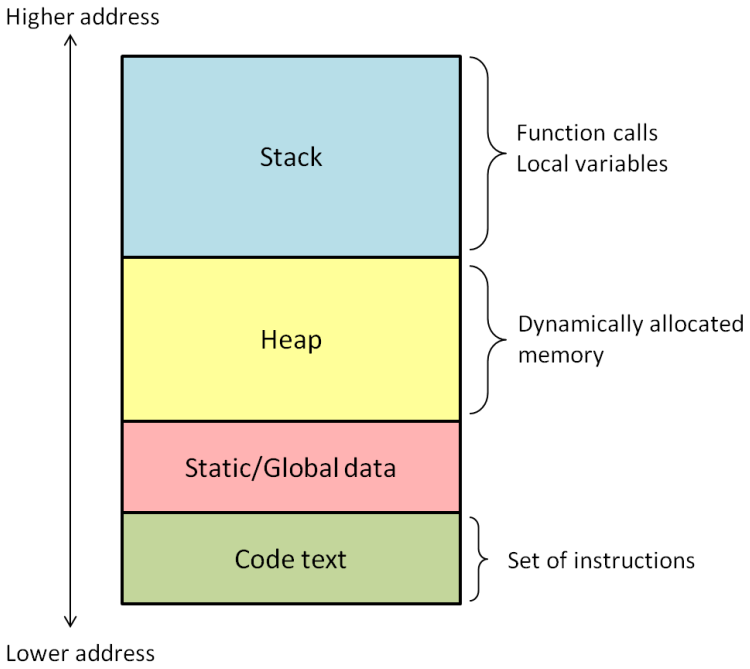

Figure 42. Memory management.

Recursion implies a function keeps being called until a certain condition is met, which in our case is either found finish or empty open list, that is, no path existing. So, recursion is appealing to call stack memory region by its definition.

A common function is pushed to the stack on its call and popped off the stack when it is finished (**return** statement reached). Recursive call, in its turn, demonstrates a special behaviour. Assume, that **calcPath(Node* closest)** is called multiple times with a different node from open list as the parameter every time. These nodes, can, for example be, B, F, R, A, C, with the smallest **f(n)** of node B increasing to node C. Note, that letter notation was chosen for better comprehension. The stack will then look like this.[43]

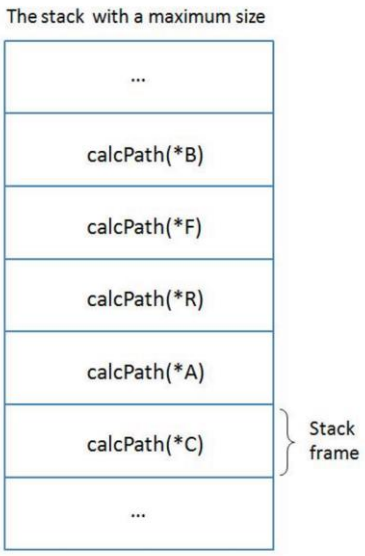

Figure 43. The stack state after a set of recursive calls.

As it can be figured out from the picture, recursive call of a function causes this function be pushed to the stack, but not popped. So, the function and its associated data will remain in the stack until the condition is met and recursion is finished. This becomes an issue.

One of the differences between the stack and the heap is allocated memory size.[44]

1. The stack grows with pushed functions and shrinks with function popping off.

2. Memory management is not custom. Memory is managed automatically by CPU.

3. Lifetime of stack variables is supported by running the related function.

4. The stack has limited size (1MB for Visual Studio, for instance).

So, paying special attention to the fourth point, it is possible to conclude, that if recursive calling of a function eventually exceeds the stack size, a stack overflow happens, leading the program to crash. Any kind of recursion splitting, implying performing a recursion partially, could be an attempt to solve the issue. Another solution could be dealing with the heap memory region instead of the stack. The figure below depicts how different the heap and the stack are.[44]

The stack	The heap
Limited stack size	Unlimited size (unless run of RAM)
Local variables only	Global access variables
Non-resizable variables	Variables can be resized with realloc()
Faster access	Slower access
CPU-controlled space management	Developer is responsible for memory management (using malloc() and free())
Memory does not become fragmented	Memory fragmentation can happen due to allocation and freeing

Figure 44. The stack and the heap differences.

Thus, custom memory allocation is what is needed to run the algorithm with large data input. Nevertheless, it allows no recursion then. So, the algorithm should be rewritten in an iterative way.

One more solution is to reserve memory somehow. Another form of A* algorithm can help with that. It is called Iterative deepening A* (IDA*) and is considered an iterative deepening depth-first search (IDDFS) algorithm, different from best-first search classic A*. “An iterative deepening search operates like a depth-first search, except slightly more constrained-there is a maximum depth which defines how many levels deep the algorithm can look for solutions. A node at the maximum level of depth is treated as terminal, even if it would ordinarily have successor nodes. If a search "fails," then the maximum level is increased by one and the process repeats. The value for the maximum depth is initially set at 0 (i.e., only the initial node)”[45].

A crucial difference between them is that IDA* does not introduce open and closed lists. So, it only takes care of the current path and does not remember already visited nodes, so the same node can end up being checked several times. That is why IDA* is relatively slower than A*. A trade-off between memory usage and calculation speed is clear.

A more advanced further step is the shortest path finding in real time, meaning that an environment can have moving objects, so it can change, which the algorithm should dynamically react to.

Finally, as an actual topic in robotic technology, machine learning could be introduced for the drone to explore a terrain and construct a map according to its measurements.

4.3 GUI design and Implementation

4.3.1 Initial design

The GUI was designed at the start of the project, including necessary features. The initial design is shown in figure 45.

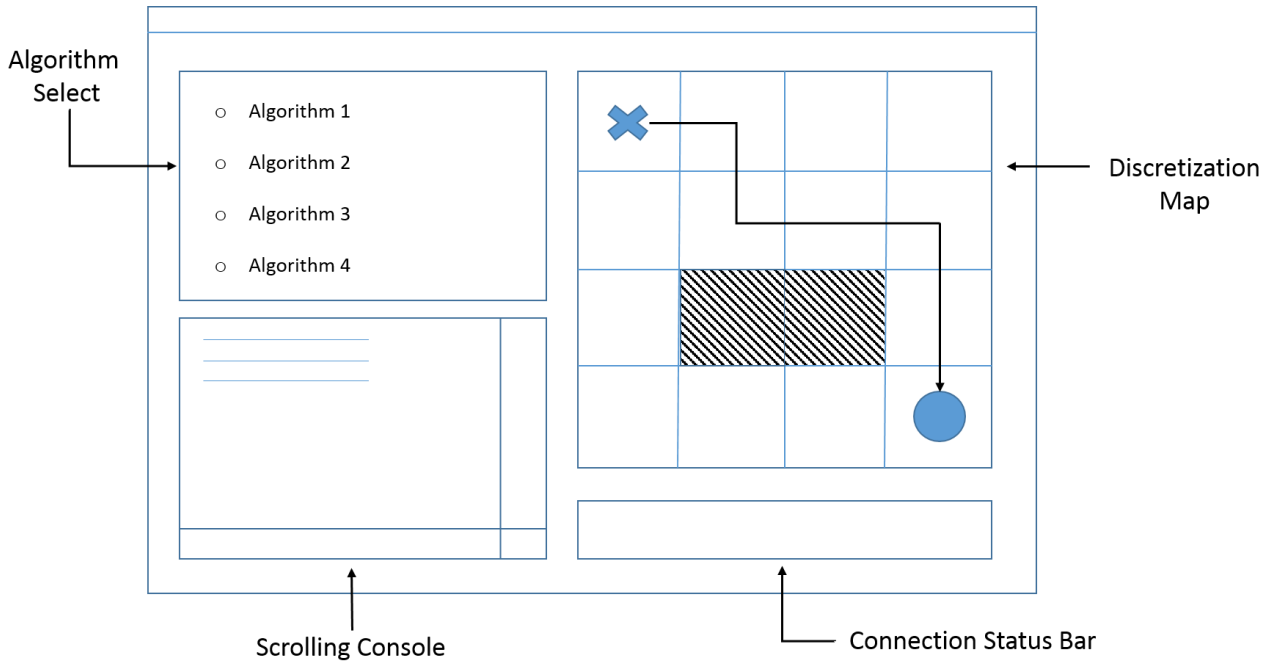

Figure 45: GUI initial sketch

The parts shown in the figure were defined in terms of uses and features:

- Algorithm select: This panel would involve selecting any algorithm available using radio buttons, and include all heuristics for these algorithms.
- Scrolling console: In this section, the user could type commands in. Some of the discussed commands were:
  - Confirm Path: Send message to drone with the instructions.
  - Stopping and hovering command.
  - Stopping and landing command.
  - Error messages that occur.

  - Mapping commands.
  - Show connection state.
- Connection status bar: This bar would show if the program was connected to the drone, as well as data about last received/sent data, date/time of connection and some way of attempting to connect or disconnect from the drone.
- Discretization map: This map would be in a discretized format, and it would change according to parameters given. The path would also be generated depending on the algorithm and heuristics selected. The map would show the initial drone position or start point, the goal node or finish, the optimal route calculated and obstacles that the terrain had. Optionally it would also support managing several drones/maps simultaneously.

4.3.2 Implementation

The GUI was developed using the .NET framework. A Visual C++ WinForms application was developed, using the Common Language Runtime (/CLR), which is a virtual machine. As an extension of this, multithreaded debug with DLL (/MDd) was used. It could be useful to utilize /MTd which does not use ".dll" extension files, so the code could be run in any computer running windows using the generated .exe, but it is not possible, since /CLR relies on using a ".dll" file. Warning level 3 (/W3) was turned on to increase the verbose level of warnings, since some encountered issues were not being displayed at the standard warning level. The Windows Runtime Compilation (/ZW) compiler option was enabled, which uses the C++/CX extension, which is compatible with WinRT. This enables making classes managed, meaning that C++ code, which is type unsafe, is ensured to be type safe. By this, one can avoid most security vulnerabilities that are usually in C/C++ programs, at the cost of having to use Windows object, instead of "std" objects. More on type safety and security vulnerabilities is discussed in the Security section.

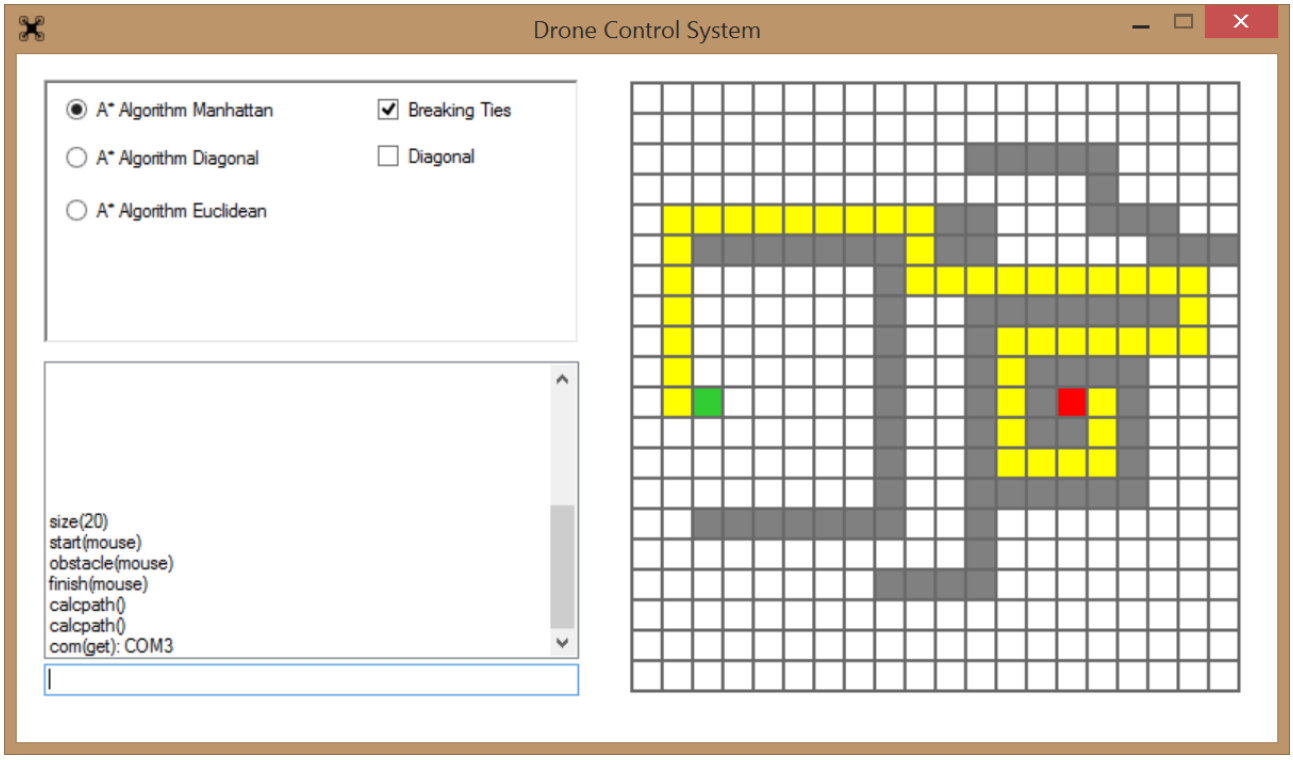

Figure 46: GUI implementation

The program uses Managed classes developed in Visual Studio, while the A* implementation classes are not managed, since they were developed in another IDE. The C++11 standard was used, since it provides many useful improvements to vectors. Initially, compatibility with C99 was also required, but the code was adapted to be compatible with C11, which does not allow declaring array sizes with variables.

Precompiled headers were deactivated. These improve compilation time by starting from results already gotten by compiling big headers like windows.h, this way skipping the recompilation of these headers.

The GUI is triggered using an instance of MyForm from the main class DroneGUI.cpp. In MyForm.h GUI components are initialized and paint sections are defined. The listeners and other functions are in the final part. The paint section takes care of the scaling map as well as all the elements in it.

The console has listeners for keys, mainly for sending a command to the protocol object via pressing ENTER and using arrows to go through the previously used commands, in a similar way the MATLAB

and Linux terminals work. Once the protocol receives the input, it processes the information and returns a state, so the form can determine which action to take.

Mouse actions were enabled to allow clicking on the map to set an obstacle, start or finish, using mouse states.

The strToP() and pToStr() functions are for the obstacles, since it is not possible to create a list of Point^. Finally the consoleMemoryManager limits the memory usage of the list box, which, if exploited, could end up occupying a large amount of RAM.

The UML for the program is in the following figure.

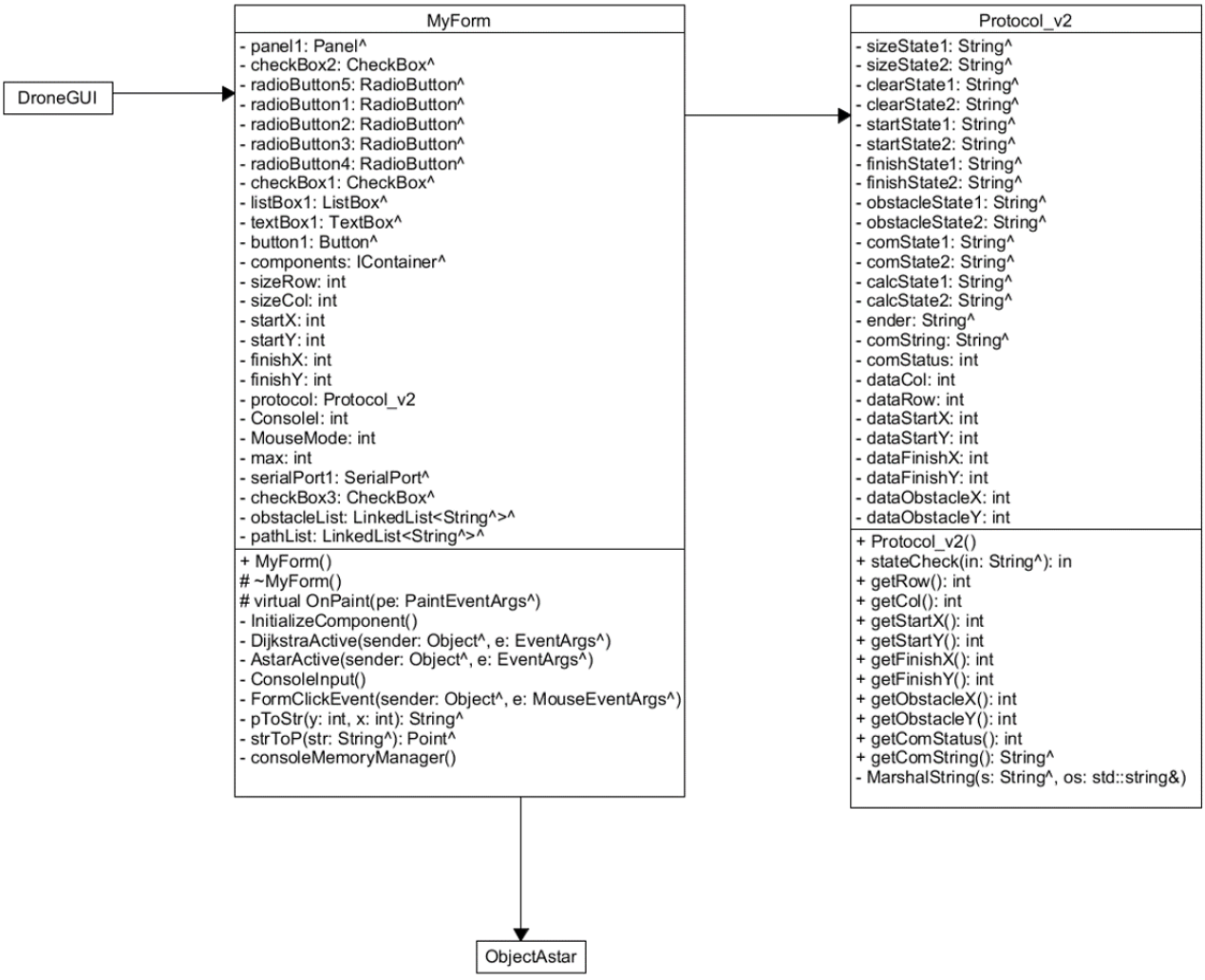

Figure 47: GUI UML class diagram

4.3.3 Further work

Although the GUI works as intended, there is a list of optional features that could be included in a possible 2.0 version.

One of these optional features was described in the initial design part, which was the idea of supporting more than one map/drone simultaneously. The idea of more than one map could be implemented by being able to click arrows besides the map to switch to another map.

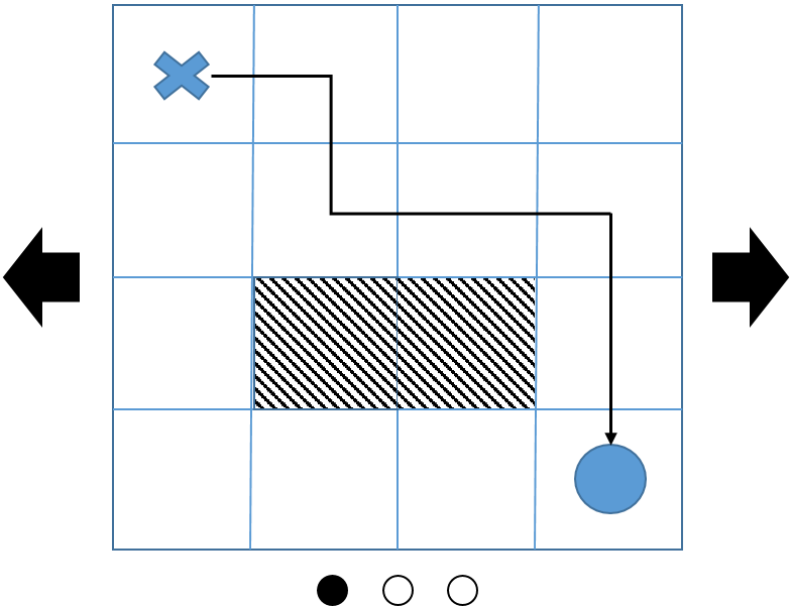

Figure 48: Extension to multiple maps concept

If this was the case then it would also be beneficial to have a map object, since the program would need more than one. This proposed class would just contain the integers for the size, start coordinates, finish coordinates, a list of obstacles, and a list of path instructions.

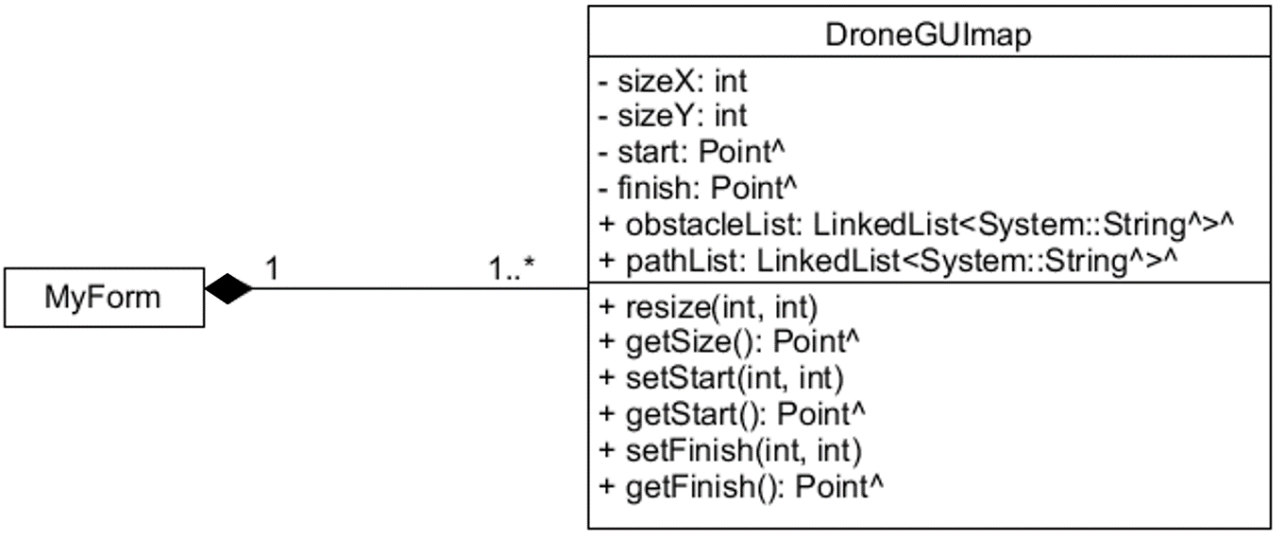

Figure 49: UML design of map to form relation

The relation between the MyForm and the DroneGUImap is composition (strong aggregation), and one MyForm can contain one or more DroneGUImap's.

More importantly than this, however, would be handling the communication in a multi-drone system. Although the packet anatomy could include a system ID section, it would be important that drones do not send messages simultaneously, since the ground control station (GCS) could miss an important message while processing the previous one, even if the communication has its own thread, assuming they share hardware to receive messages. Ideally, the program would create a thread for processing each message separately from the radio hardware thread, however not even this would guarantee successful receiving of messages, since messages that intersect will still cause issues. The communication issue was discussed in chapter 3.5.

4.4 Simulation

Since there were some issues with the real prototype, a software simulated system was considered as a back-up plan and was successfully developed in Java programming language. A relatively short description is provided in this chapter. Different languages were chosen to have a multilingual system. Java has powerful libraries for graphics and certain use was made of multithreading supported in Java. Eventually, multilingual system demonstrates that there was no problem in telecommunication between two programs written in different languages.

The simulation program is not run on the same computer as the algorithm program – a standalone computer is used instead. Radio communication happens via the telemetry, plugged in serial COM ports. For further text, a short notation is introduced. The computer with shortest path calculation and GUI for setting algorithm parameters will be called “the station”. The simulation running computer will be named as “the simulator”.

First of all, it is necessary to understand that simulated environment represents a real discretized terrain; it does not need to know a map, because it is an actual map. Therefore, the station should know the map it is going to deal with. In terms of this project the map is known in advance and both programs need to agree on it. Once this is handled, the algorithm performs its job and finds the shortest path, which is then translated into a set of movement directions, in other words, a bunch of instructions the simulated drone should carry out. Instructions are generated in terms of drone graphical object X and Y coordinates this time. The picture below shows which symbols are attached to which directions.

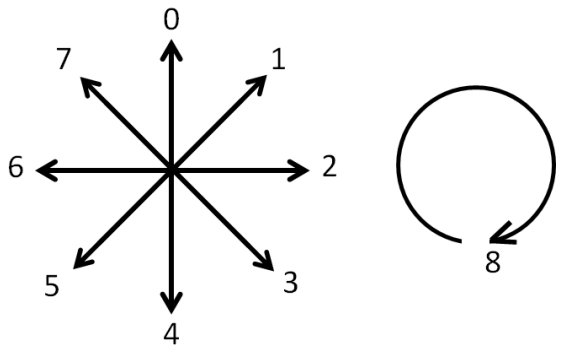

Figure 50. Direction notation.

As it is possible to notice, directions are represented by integers, where 2 means drone **forward** direction as a reference, and 8, stands for **stop**. There is a reason for using integers and not chars or even strings, which are easier to understand intuitively. On one hand, the libraries used have their own already implemented protocol to handle reading and writing via a serial port. So, sending a plain text message would not be a problem. On the other hand, it is possible to say, that the quality of simulation would then suffer to some extent. Working with bits gives more freedom in both learning and reality approximating. Techniques such as encoding or error-detecting could be applied to a sequence of bits. In this way, it was decided to use binary code (a sequence of zeros and ones,

stored in **string** variable type nevertheless), which integer numbers can be converted to. One more reason to use integers is program branching (9 branches for 9 directions to be precise) avoidance, which will be mentioned later.

Three commands (directions) are sent per message. Each command is coded with 4 bits, which are from 0000 to 1000. Total length of one message is 12 bits before encoding. If amount of commands division by 3 does not give a zero remainder, then the rest of a message is just filled with 1000, which means 8 and tells the drone to stop. So, assume there are 8 nodes in the path including the goal node giving the simulated drone 8 directions to move. In this case there will be 3 messages sent, which are “xxx”, “xxx”, “xx8”, where “x” is a number from 0 to 7.

Moreover, Hamming code error-detection method is implemented. The message length becomes 16 then. The dependence between initial message length and encoded message length was demonstrated in 3.3.

Let one have a simple example. Say, the following simple map is processed by the algorithm on the station. The screenshot was taken while simulation was running with node names assigned later. Colour notation stays the same.

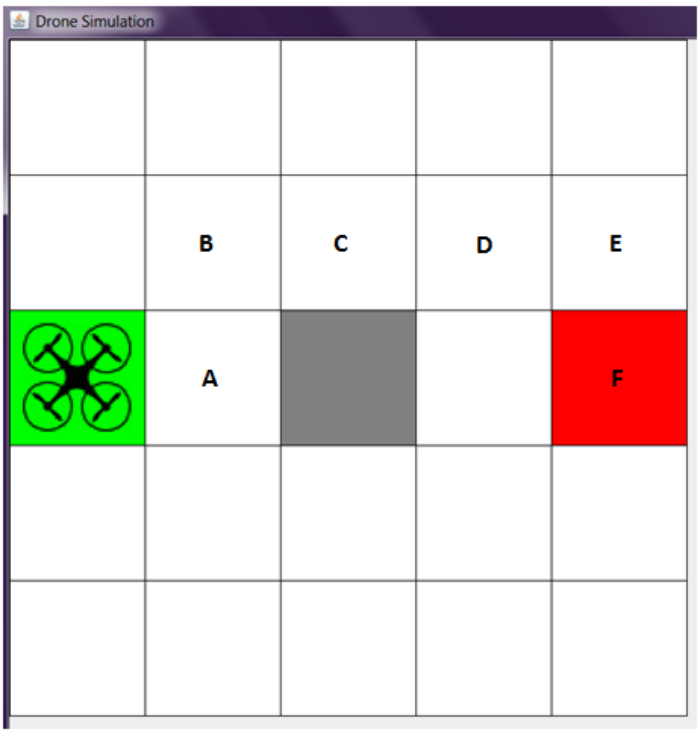

Figure 51. Simulation process screenshot.

The process can be step by step described like this.

The station transmission.

1. A, B, C, D, E, F nodes in path.
2. Direction to move: forward, left, forward, forward, forward, right.
3. Directions in integer numbers: 2,0,2,2,2,4. 6/3 = 2 messages to be sent.
4. Before Hamming: 001000000010 and 001000100100.
5. After Hamming: 0110010010000010 and 0010010000100100 sent one after another one.

The simulator receiving.

1. Received a message with applied Hamming code. The message is checked for errors.
2. Messages are decoded: [“001000000010” “001000100100”].
3. Each four digits of the message are converted from binary to decimals, which are concatenated in a string: ”0010|0000|0010” = “202”, “0010|0010|0100” = “224”. Messages are pushed to the queue one be one.
4. Each decimal digit in the message string is an **index** of an array element, which describes the behaviour of the simulated drone in terms of X and Y coordinates.
5. Messages are peeked (popping the first element) from the queue and changes in X and Y (velocities) over time are respectively assigned to the drone picture in simulation: [(1, 0) (0,-1) (1,0)] and [(1,0) (0,1) (1,0)]. If the queue is empty, the drone stops.

One remark to point number 5 of the simulator is that accessing the instructions by array element is an alternative to program branching, which implies having an if-statement for each single command. The whole instruction table is available below.

i in directions[i]	Change in X	Change in Y
0	0	-1
1	1	-1
2	1	0
3	1	1
4	0	1
5	-1	1
6	-1	0
7	-1	-1
8	0	0

Figure 52. Directions[] array table.

The conceptual model of the system communication inspired by OSI model is provided below.

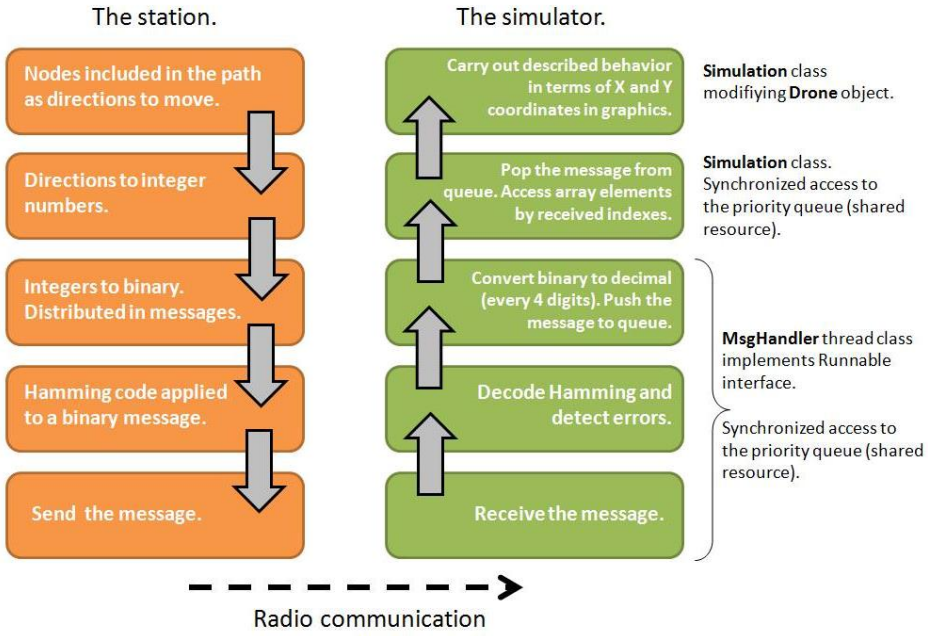

Figure 53. Communication in simulation model.

Message handler class works as a thread and has a **run()** method, where message receiving happens in real time via **while(true)** loop. It is done for simulated drone movement and message receiving

to happen simultaneously and not wait for each other to finish. The queue is a resource shared by **MsgHandler** thread (pushes elements to the queue) and main thread (reads and pops elements from the queue). Therefore **synchronized** statement is used to prevent race conditions – a known issue in distributed and multithreaded computing, when two or more threads are trying to perform changes on a common resource at the same time.

Drawing and animation are performed via standard means available in Java such as "java.awt.Graphics" library and "java.awt.event.ActionListener" together with "java.awt.event.ActionEvent" complemented with "javax.swing.Timer". UML class diagram can be checked out below for further details.

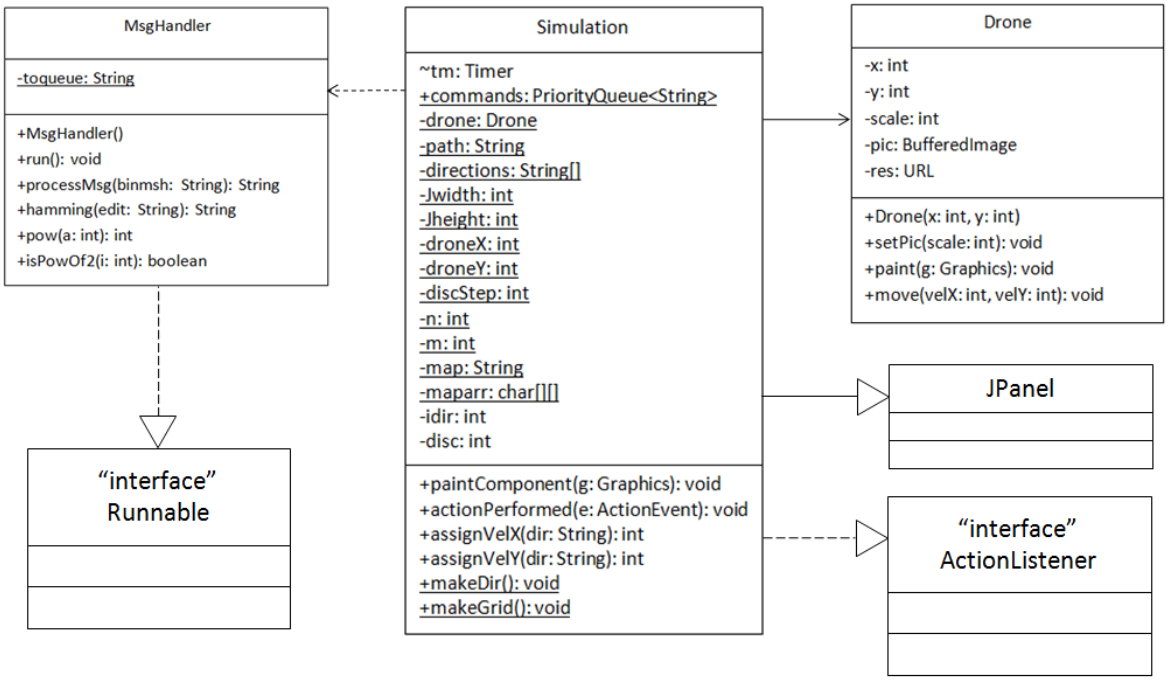

Figure 54. Drone simulation program UML diagram.

Finally, discretization is an important part. It was not really applied to a real environment – the development process simply did not reach that point because of some issues explained in chapter 4.5. Simulation, in its turn, lets one implement discretization, but from a different prospective.

Real drone has fixed size and is supposed to fly in different environments of different sizes. Software simulated environment is, however, limited by the size of a screen containing **JFrame**, which drone

picture and map size are scaled with respect to. The process of finding the discretization factor is the following.

1. Provide the program with map size in width (variable **m**) and height (variable **n**) units, which can, for example, be extended to meters. **JWidth** and **JHeight** represent **JFrame** dimensions.

2. Calculate discretization step according to the formula.

if (n>m) discStep = (jheight-50) / n;
else if (m>=n) discStep = (jheight-50) / m;

Note, that **JHeight** is used as a reference, because most screen resolutions have larger width and smaller height, and 50 is an arbitrary number for not going above window borders.

When discretization factor or step (variable **discStep**) is calculated, it is applied to grid drawing, drone picture scaling and, of course, to its movement, so the coordinates change (by ones, as it can be seen from Figure X – table) **discStep** times, and proportionality is respected. The video demonstrating the simulation in work was captured and is available in the project folder.

4.5 Drone Solution

For this project, the prototype is made from a 45 by 45 centimetres frame, 30-amp speed controller, 980 Kv motors, 6000 mAh battery and 10x4.5 inch propellers; all of them controlled by the APM flight controller module. These features create an agile quadcopter that can also carry some extra mass that may include the electronics necessary to make it autonomous, compared to the options discussed in chapter 1. This configuration has been chosen with the knowledge acquired over the past semester projects. The prototype assembling process was quite straightforward based on the fact that this process was done few times in the past. After the construction of the prototype, a few tests were performed to check if all the main parts of the prototype were working. More tests were necessary for the fine tuning of the quadcopter, but the weather was not suitable for outdoors flight. These tests were supposed to help tune and see how the quadcopter would fly in open space and were important as it would have offered some beneficial insight for the final testing.

4.5.1 The assembly:

The prototype is made from the following parts:

The frame is the main part of the quadcopter and is made from glass fibre to give it structural resistance, but in the same time to keep it as light weighted as possible. The bottom side of the frame has also included a simple PCB that connects the battery and the ESC (Electronic speed controller) that is connected forward to the motors.

Figure 55. Frame[46] [47]

The 980Kv brushless motors can be considered medium sized motors by their power. These motors can produce a rotational motion of 980 rpm for each applied volt, resulting in a total angular velocity of the motors of around 13000 rpm if the maximum voltage the battery can apply is 12.6V.

Figure 56. Brushless motor[48]

The electronic speed controllers transform the DC current from the battery to 3 phase AC for the motors and they can also handle fast bursts of current for when a big change needs to be made in flight.

Figure 57. ESC[49]

The APM flight controller is an open source board used for controlling RC vehicles and can be considered the brain of the quadcopter, containing all the sensors necessary for keeping the prototype in the air. The board has premade connections ready to be used for GPS and telemetry.

Figure 58. APM controller[50]

The 10 x 4.5 inch propellers are attached to the motor mounts, and are the biggest size of propellers that the 980 Kv motors can handle, with an acceptable flight time with the current battery.

Figure 59. Propellers[51]

A radio receiver is used for communication between the remote control and quadcopter. Having 9 channels available, just five of them need be used in this case. Having these channels is still enough for all directions movement and an extra switch that can change the flight mode of the prototype.

Figure 60. RC Receiver[52]

The Arduino Uno is the microcontroller that will make possible changing the behaviour of the prototype with simple commands from the computer sent through the X-Bee module. A program will make the translation from the computer commands to direct input values to the APM.

Figure 61. Arduino Uno[53]

A wireless communication between the two parts of the system is made with the help of the X-Bee modules. The modules connection was explain in chapter 3.4.

Figure 62.X-Bee module[54]

4.5.2 Prototype versions

From all drone prototypes, which were so far developed by the group members, the current one experienced the most versions before it went on the right track.

Version 1.0

For the first version of this quadcopter, the way of making it work was by creating a new flight mode in the code of the APM controller. With this new flight mode and communication between the drone and the computer that calculates the shortest path that can be used, it would control the drone. Based on the fact that the APM is an open source controller, all the code was available online and all that was necessary to do was to use already made methods from the code to make the quadcopter work in the required way. The fact that landing and take off had predefined methods made this first idea seem simple. The communication between the two devices would use the telemetry module to communicate via the MAVlink protocol.

The open source code was poorly organized and documented. While the board receives almost monthly updates and improvements, these are not backward compatible with other required tools and hardware. The flight controller was only compatible with an older version of the firmware, and the whole documentation for it had been replaced with newer incompatible one. Furthermore, changes in the uploading program meant that it was necessary to modify all classes from the no longer compatible programmable chars to standard char. Also the command to assemble using C++ command has changed from “–assembler-with-cpp” to “–x assembler-with-cpp”. Another issue

encountered was that the firmware was larger than the memory of the flight controller, and not explicitly said by the compiler. Finally the configure file for the compiler had to be replaced to upload to the flight controller.

Once the flight mode was created, it was not possible to select it, since the standard GCS program retrieved flight modes from the online firmware. This meant that an internal XML had to be modified in the program to allow selection of the custom flight mode. Still then, there was an error in the firmware which was not detected by the compiler which prevented use of this custom flight mode. After all these attempts it was decided to take an alternative approach, as the timeline would not allow the prototype to work as planned.

Version 1.1

After the first attempt was unsuccessful, a way of controlling the quadcopter outside the APM board was a better idea, since it did not require changing the code, but just affecting the input from the remote. This will make a better solution to the problem because all that needs to be done is replacing the human input from the remote control with direct input from the computer. Before trying to send commands directly to the prototype, four ultrasonic sensors were used to observe whether an obstacle avoidance system could be implemented without communication. The prototype will work in the way that, being in a maze, the quadcopter will receive up, down, left or right instructions, and the ultrasonic sensors will take care of quadcopter's safety. After multiple instruction sent, the quadcopter might have some distance errors, not necessarily big, but even a relatively small error per instruction will sum up to a bigger error and can eventually cause a collision. With the implementation of the ultrasonic sensors and the obstacle avoidance system the quadcopter can "calibrate" itself when an obstacle is on two opposite sides of its up, down or left, right for example. If the path cannot be reached before a certain amount of error is accumulating, then the quadcopter could look first for the first spot that can be found to calibrate the position in space of it. For the communication, it is necessary to send the instruction from the computer to the prototype via radio. A basic radio frequency module was used next, but sadly during the tests with these modules, a technical mistake caused both modules to be burnt. The last attempt to make the

communicate between the two devices work was using two X-Bee modules. After the modules had been configured to work together the communication successful.

The ultrasonic sensors and the X-Bee module were connected to the Arduino Uno board that was set between the radio receiver of the remote control and the APM board. In this way human control was still possible but in the same time, the computer could send instructions to the quadcopter without affecting the capability of manual control of the quadcopter and keeping it safe from collision.

An improvement for the position of the sensors was designed but the part could not be 3D printed in time for the final testing. Below is a 3D sketch of the part that is holding the sensor.

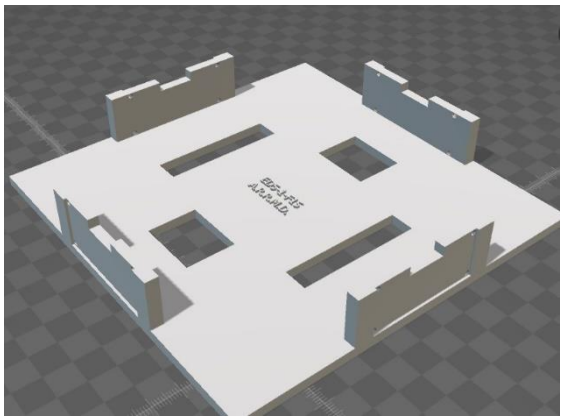

Figure 63. Quadcopter Ultrasonic holder

Both the sensors and the X-Bee were working separately. When put together, the delay caused by the time that the ultrasonic sensors needed to read the distance made the system have an overall input to output delay of about 0.8 seconds. It may not look like a significant delay, but all the human control and instructions will be delayed by at least 0.8 seconds. Even that small value that is under one second, makes the quadcopter not able to fly utilizing human input. In terms of safety, a human controller used as a safeguard was not a viable option anymore because of the reliability of the human input commands. A different approach was again necessary to find a solution to this problem.

Version 1.2/Final Version

This is the last version for this prototype which was also used for testing. By eliminating the ultrasonic sensors, which were giving the delay in the system, a direct connection was made between the instructions sent from the computer to the APM. Without the ultrasonic sensors the movement of the quadcopter could not implement calibration while flying and the amount of thrust had to be more carefully calibrated. The Arduino in the quadcopter is taking the instructions from the X-Bee and transform them into movement by changing the PWM values like the remote control does. It is important that all the instructions are uniform in terms of distance moved. This was hard within the time available for the testing, so the results obtained from testing were as precise as desired but for sure in the right direction. The system was able to control the quadcopter with only the input from the computer. Because the precision of the commands was not yet accurate enough, the testing was not in a high altitude flight but sliding the quadcopter on the floor and making it follow a certain path sent from the computer.

Conclusion of Versions

After having considered all the versions described above, and after running tests on them, it is obvious that there are many improvements to be made, which were discovered thanks to this trial and failure process. In the following section the future, possible upgrades and improvements for the quadcopter will be presented.

4.5.3 Future Steps/Improvements:

Even though the final prototype is not perfect and not behaving exactly as intended, it is sure that the process itself brought knowledge. Every version that was tried and every time something was not going as expected extra knowledge was acquired.

From the first version the most important lesson is that if something is called open source is does not necessarily imply that the code is maintainable, some parts of the code may not work as expected, and it is not guaranteed that there is documentation available for our specific application. This forced the project towards an alternative approach and attempt to understand the system as a whole.

The second version was a simpler concept to make the prototype work because it can be better to split the problem into smaller modules, and solve them by the “divide and conquer” method. One

of the improvements that can be made on the sensor side would be having the part that was designed for this project but not printed in time. This part will make the sensors more precise by keeping them perfectly straight: a feature that is very important for the ultrasonic sensors. Also, in order to be able to use communication and sensors simultaneously, these two processes need to work in different threads or ideally on different microcontrollers.

Some last improvements that can be made for the overall prototype could be better calibration for the quadcopter. This calibration needs to be as precise as possible to obtain the best results from the prototype, reducing the frequency of corrections.

For the overall idea of autonomous drone, the following electronics may be added and may enhance the movement, detection and stability of the quadcopter:

- GPS Module
- Camera (for image recognition)
- Infrared camera (for low light environments)
- RTL feature (RTL- return to home)

5 Testing/Analysis

5.1 Time Testing

Elapsed time testing is a test of A* algorithm, was performed to find out how algorithm parameters affect the performance and scalability. The testing process includes three entries: diagonal movement on and breaking ties on (Y, Y), diagonal movement on and no breaking ties(Y, N), and no diagonal movement and no breaking ties (N, N). Testing will be performed with square maps changing in size, where start node takes top-left corner and goal node takes bottom-right corner. A simple map example is provided below.

s 0 0

0 0 0

0 0 f

The first test is with (Y, Y) parameter, where Y (refers) says “Yes” to “Diagonal”, and second Y (refers) says “Yes” to “Breaking Ties”. The whole test war done for Manhattan heuristic type.

In the map above, “s” stands for start node, “f” represents finish node and “0” is used to show empty space. So, string representation of the map will be “s0000000f”. Thus, the following object will be constructed for testing:

ObjectAstar calccom(3,3,"manh",'Y','Y',"s0000000f ");

Its alg() method will be called then. The process is looped 100 times and each iteration elapsed time is recorded, which allows us to obtain average time taken by calculation. This approach is used because time resources, taken by an application, depend on many factors (such as computer characteristics, CPU usage), so will be different. Note, that the whole testing was performed on one machine.

Average time is then written down and map size is increased. The map size starts at 3x3 and increases to 20x20 one by one. Then it is further increased by 5 until 90x90, where it takes 14.84 ms.

The graph was made in MATLAB program by running a simple code. X axis represents the size in terms of nodes and Y axis represents elapsed time in miliseconds. Average time data was passed to MATLAB and plot(x,y) command produced the graph.

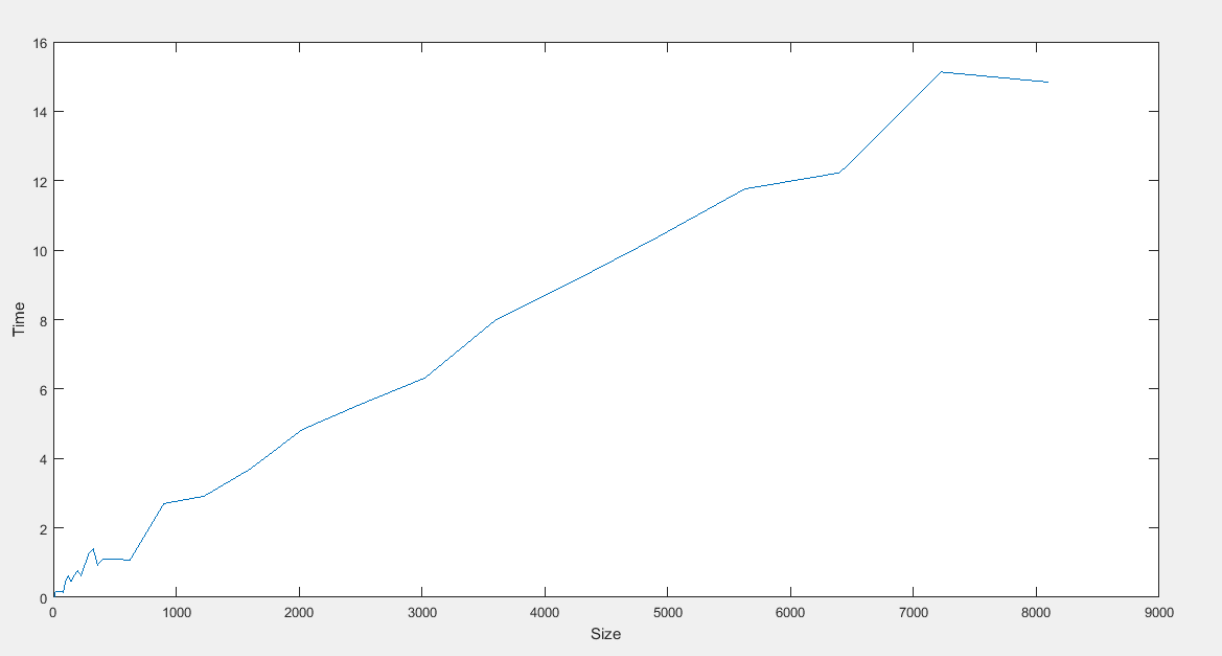

Figure 64. (Y, Y) test MATLAB

The parameters second test are “Diagonal” activated (Y), and “Breaking Ties” deactivated (N). This test will only go up to 65x65 (4225 nodes) to avoid stack overflow (covered in chapter 4.2), since deactivated breaking ties will make the algorithm consider more nodes, and thus cause a larger stack. Comparing the (Y, Y) result to the new results, we see that the first set of results are much faster overall. The maximum time of (Y,N) is for 65x65 which has a value of 40.65 ms.

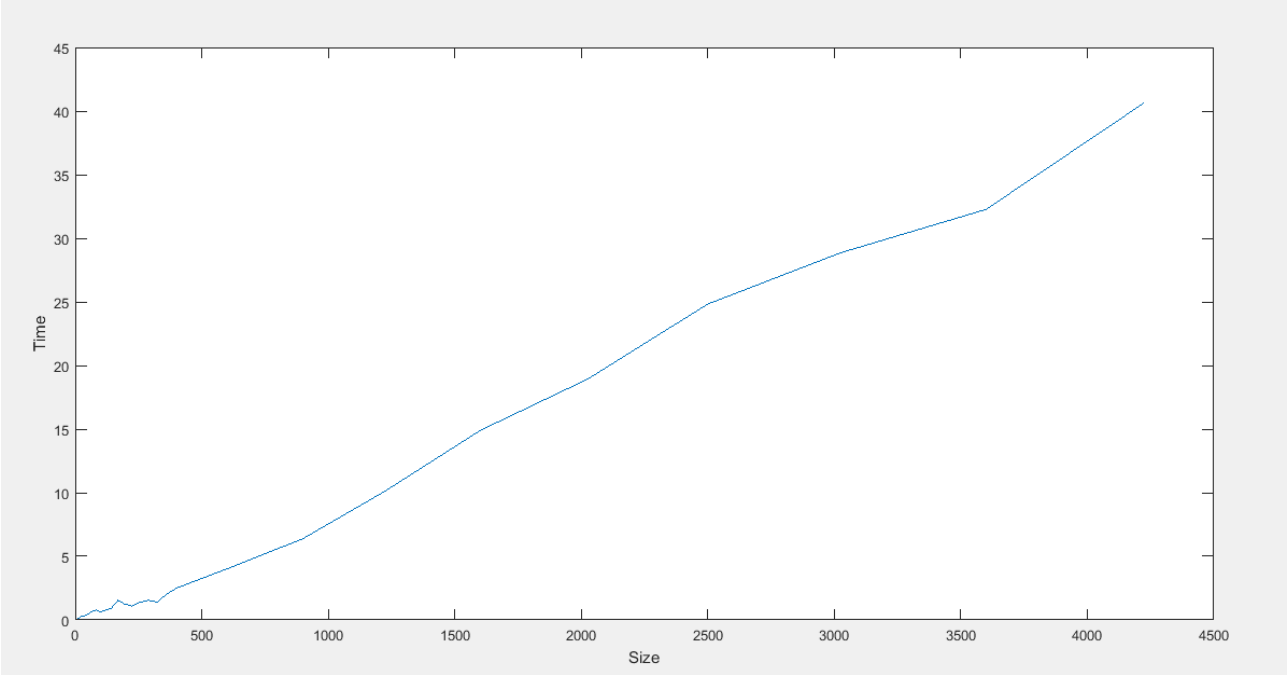

Figure 65. (Y, N) test MATLAB

The third test is (N, N), which mean the program will set both “Diagonal” and “Breaking Ties” as NO. This test uses the same size scaling as the second test. The maximum result for (N,N) is for 65x65 with a value of 34.82 ms, which is smaller than (Y, N) test and larger than (Y, Y) test.

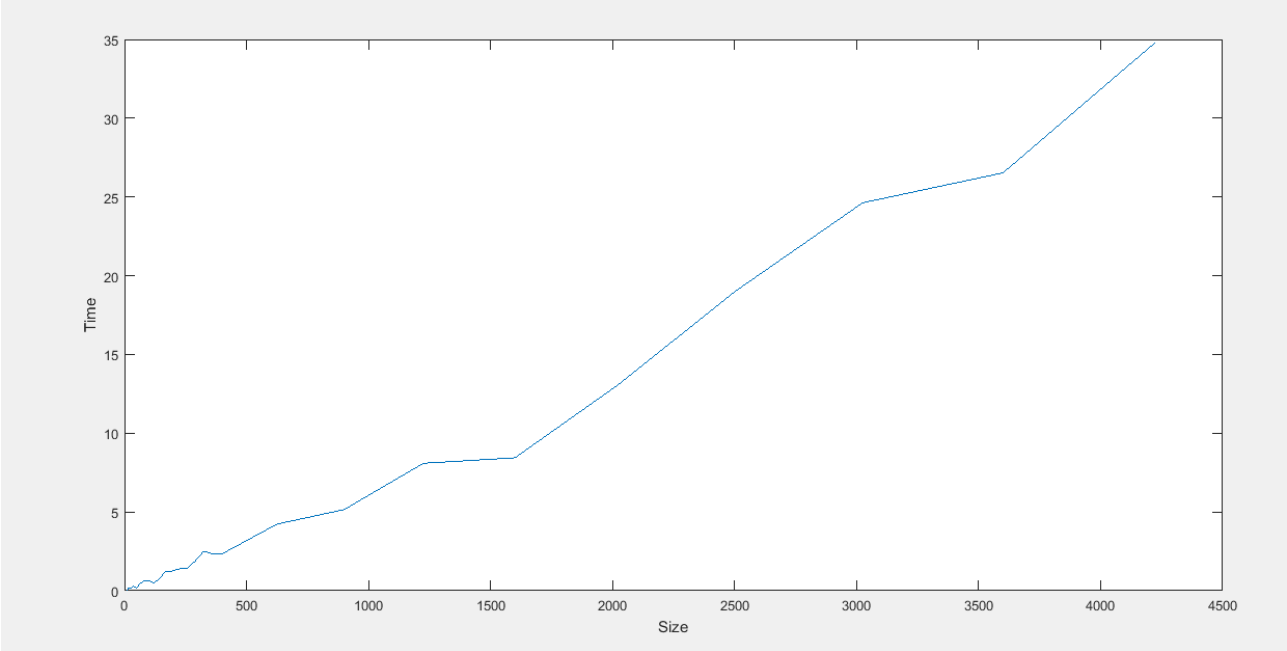

Figure 66. (N,N) test MATLAB

5.1.1 Comparison of results for (Y, Y), (Y, N), and (N, N) parameters.

After those three tests, the difference in the results can be noted. First test (Y, Y) shows that the program can be faster when “Diagonal” and “Breaking Tie” are enabled, which means it can do the calculation in less time compared to the other two tests. The table below shows average elapsed time for different parameters for map size from 40x40 until 65x65 for being able to compare them. Note, that (Y,Y) parameter function can, as it was stated earlier, go up to 90x90, which is simply not displayed in the figure.

Map sizes	Time for (Y,Y)	Time for (Y,N)	Time for (N,N)
40x40	3.69	14.89	8.44
45x45	4.83	18.93	13.1
50x50	5.56	24.83	18.98
55x55	6.32	28.86	24.63
60x60	7.99	32.27	26.53
65x65	9.09	40.65	34.82

Figure 67. Selected testing results.

As it is possible to conclude, the behavior tends to be as expected: the elapsed time is regularly growing with increasing the map size and breaking ties technique boost the speed of calculation. There, however, is a suspicious behavior for (Y,N). It was supposed to be either faster, or approximately the same as (N,N) parameter function testing. As it can be seen, the difference in time for these to is almost always around 6 seconds, so it can be assumed, that a certain operation takes this additional time. A visual representation of testing is available in the following picture.

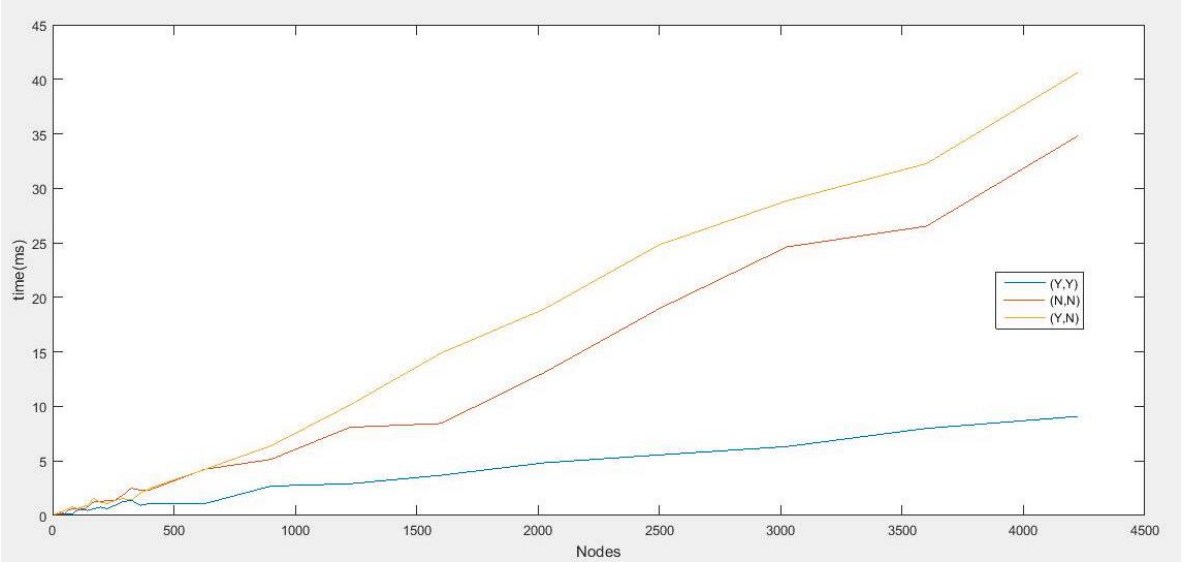

Figure 68. Comparison between algorithm options

5.1.2 Big O Notation

Big O notation is a mathematic way to describe the behaviour of functions. Big O notation shows a tendency of how change in input affects the resources used. In our case Big O notation was used to estimate elapsed time. The following functions are used:

Notation	**Function**
*O(log(*log(n)*)*	*f(x) = log(log(x))*
O(log (n))	*f(x) = log(x)*
O($\sqrt{n}$)	*f(x) =$\sqrt{x}$*
O(n)	*f(x) = x*

Figure 69. Big O Notation

We placed the data gotten from the tests into graphs with the functions to estimate how they perform relative to each other. The following graph displays the (Y,Y) data with big O functions for comparison.

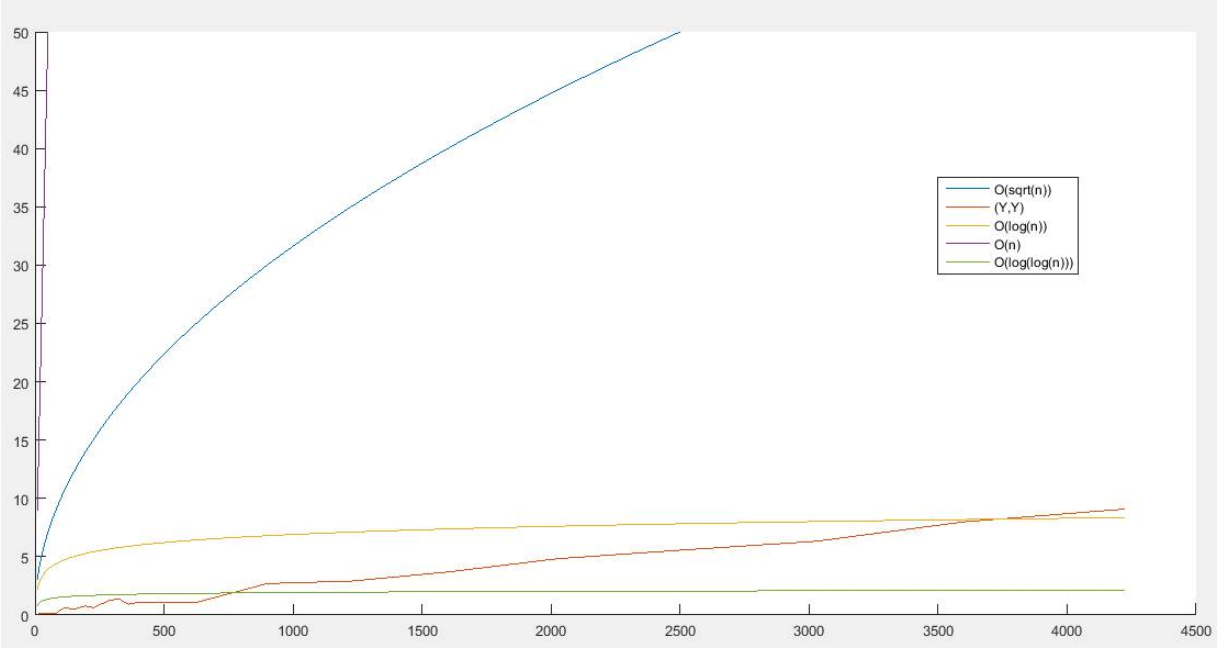

Figure 70. (Y,Y) performance

From this graph we can conclude that the performance seems to be better than square root, but worse than the logarithmic function. From this, we can assume that the performance could be polylogarithmic or a fractional power.

The next graph compares the (N,N) data.

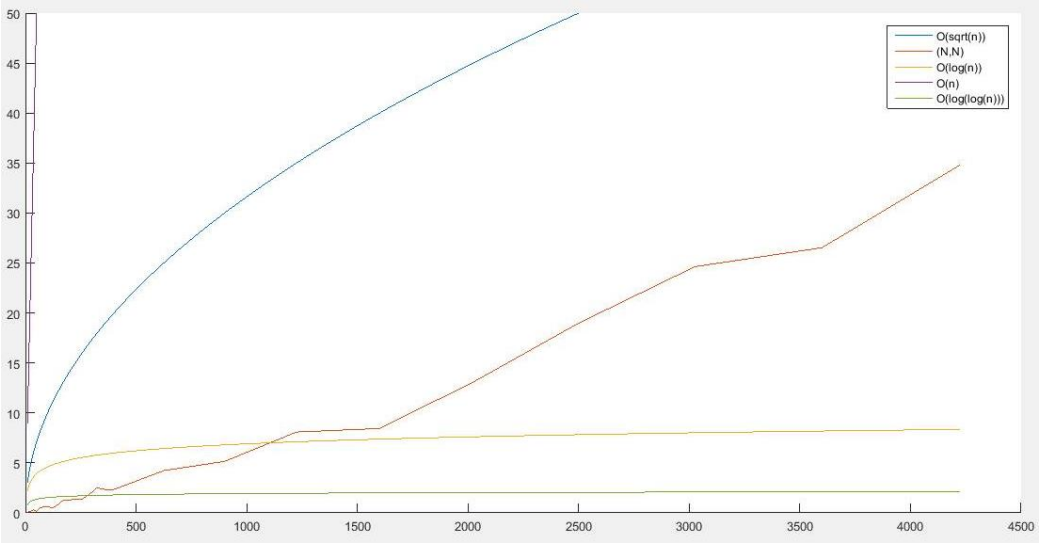

Figure 71. (N,N) performance

This is clearly worse that the previous, and its gradient is clearly very close to the square root function.

The last graph is about the (Y,N) parameters:

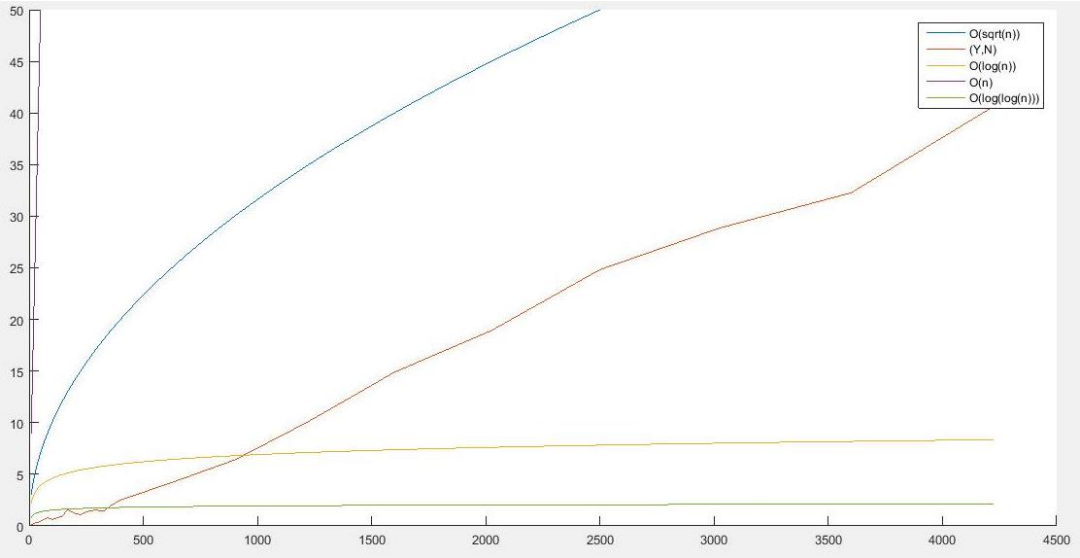

Figure 72. (Y,N) performance

This function has a very similar gradient to the last test, and thus can also be approximated to the square root function.

From these test we can conclude that (Y,Y) is the superior parameters in terms of complexity, and the other two options are approximately equal.

One additional note is that the algorithm actually has a $O(n^2)$ performance in terms of size, since the amount of nodes will increase in that ratio. This was avoided by graphing nodes in the x-axis, to be able to compare the parameter option complexity in more detail.

5.2 Percentage Error of Time Testing

The idea of error percentage of elapsed time testing is to discover how the error scales with an increasing map size. The same algorithm parameters are used as in the previous testing section.

Percentages of Time testing will include those three tests (Y, Y), (Y, N), and (N, N), where it will calculate the error percentage of time at each size.

The size 3 x 3 is the start size of all three tests. The method to find the percentage of time will be by finding the ratio of values of each time and evaluates what the potential error could be in a worst case scenario.

To find the percentage in a 3 x 3 size in (Y,N) where the time is 0.15 ms, the ratio of values will be between two numbers (0 or 1). The ratio is 85% of 1 and 15% 0. This is the lowest possible time, and the maximum is where 0 is $0.\hat{9}$ and 1 is $1.\hat{9}$. This means that there is a variability of approximately 1, so the value could be expressed as 0.65±0.5. From this we can determine that the error of this particular time is 76% relative to the middle point.

Since the error will always be one, by logic, the percentage error will decrease at the value increases.

To make sure that the results are understood correctly, it is necessary to consider that increase in time leads the errors decrease. The following table shows the pattern described above, and displays how the error percentage is decreasing as the time increases.

5.15	900	30x30	8%
8.09	1225	35x35	5,57%
8.44	1600	40x40	5,34%
13.1	2025	45x45	3,55%
18.98	2500	50x50	2,58%
24.63	3025	55x55	1,98%
26.53	3600	60x60	1,84%
34.82	4225	65x65	1,42%

Figure 73 - (N, N) Percentages of Time Testing

This behaviour will also affect the tests we ran, and by extension we can determine that the (Y,Y) results have the largest error percentage, since it has the fastest performance. It is important to note though that the error percentage is near negligible at most results past 40x40. We can explain the oscilating nature in the graphs at the lower map sizes by the relatively large error, while the fact that they become increasingly stable, proves that the error is relatively smaller at the larger maps.

ObjectAstar calccom("manh",'N','N')			
5.15	900	30x30	8%
8.09	1225	35x35	5,57%
8.44	1600	40x40	5,34%
13.1	2025	45x45	3,55%
18.98	2500	50x50	2,58%
24.63	3025	55x55	1,98%
26.53	3600	60x60	1,84%
34.82	4225	65x65	1,42%

ObjectAstar calccom("manh",'Y','N')			
6.39	900	30x30	6,87%
10.13	1225	35x35	4,51%
14.89	1600	40x40	3,25%
18.93	2025	45x45	3,03%
24.83	2500	50x50	2,44%
28.86	3025	55x55	1,70%
32.27	3600	60x60	1,51%
40.65	4225	65x65	1,21%

ObjectAstar calccom("manh",'Y','Y')			
2.7	900	30x30	14,40%
2.91	1225	35x35	14,37%
3.69	1600	40x40	11,21%
4.83	2025	45x45	9,16%
5.56	2500	50x50	7,79%
6.32	3025	55x55	6,96%
7.99	3600	60x60	6,76%
9.09	4225	65x65	4,98%

Figure 74 - Percentage of (N, N), (Y, N) and (Y, Y) time testing

5.3 Machine epsilon

One point of interest in computer science and engineering is accuracy of number representation. This deals with how one can represent a number in floating-point format without losing precision and occupying extra-memory. However, this can never be achieved as there is a trade-off between precision and memory usage. As a number is represented with more precision (it will have more decimal points in its representation), it will take more memory.

The way computers represent real numbers is by using the single or double IEEE floating-point arithmetic where the floating-point number has the following format:

$$x = \sigma \cdot \bar{x} \cdot 10^{e} \quad (1)^{1}$$

Where σ is called the sign bit; $\bar{x}$ is called the significant or Mantissa bit and e is the exponent. It must be mentioned that the IEEE single floating-point representation has a precision of 24 bits, whereas the double floating-point representation has a precision of 53 bits. [55]

The machine epsilon can be defined as "the difference between 1 and the next larger number that can be stored in that format." [55]

Basically, machine epsilon defines the limit of floating-point numbers precision which a particular machine can have. In the single IEEE floating-point representation, the precision is limited by 24 bits meaning that everything that requires more than 24 bits will be rounded towards a 24-bit number implying some precision loss.

For example, if a number has a precision of $1+2^{-24}$, then, according to the IEEE single floating-point standards, the precision is out of bounds and the number is chopped to a precision of $2^{-23} = 1.19 \cdot 10^{-7}$, where 10^{-7} describes that successfully only 7 decimal digits of a decimal number. [55]

An example of C++ code for finding a machine epsilon for is the following:

```
while (num != 0) {
        oldnum = num;
        num = num / 2;
}
 std::cout << oldnum << std::endl;
```

The provided code works in the following way. Variable **num** has a type, which supports floating-point. This variable is divided by 2 on each iteration of the while() loop until **num** becomes so small that it cannot be tracked by a computer, so is rounded to zero. Last value of **num** is stored in another variable and gets printed after the loop.

5.4 Efficiency of A* algorithm using different heuristics

A heuristic is simply an "estimate of the minimum cost from any vertex n to the goal." [56]. Based on the choice of heuristic, the performance and efficiency of the algorithm can be manipulated in order to perform better. The testing session of the algorithm will give an insight which heuristic and parameters will optimize the A* algorithm for both an empty map and a map filled with obstacles. When testing the algorithm, the size of the map was predefined to be 20x20. But there is a trade-off as some heuristics can enable the algorithm to be faster or more precise. So there must be equilibrium between the two as accuracy must be prioritized but speed must be also of interest.

Each heuristic tested has a list of parameters which can be either enabled or disabled. The heuristics that have been tested are Manhattan, Diagonal and Euclidian. In the parameter list of each of these heuristics are diagonal movement (can be either enabled or disabled), breaking ties (either enabled or disabled) and last but not least, the map which can be either empty or with obstacles. By testing

different combinations of these parameters, the optimized configuration of the A* can be determined.

Parameters	Time(milliseconds)	Length of path	Nodes considered
Manh(N,N, empty)	1.63	380	360
Manh(N,N, obstacle)	1.61	640	255
Manh(N,Y, empty)	0.77	380	216
Manh(N,Y, obstacle)	1.39	640	255
Diag(Y,N, empty)	1.74	266	372
Diag(Y,N, obstacle)	1.4	470	277
Diag(Y,Y, empty)	0.78	266	132
Diag(Y,Y, obstacle)	1.4	484	257
Eucl (N,N, empty)	2.02	380	218
Eucl(N,N, obstacle)	1.86	640	255
Eucl(N,Y, empty)	1.87	380	218
Eucl(N,Y, obstacle)	1.72	640	255
Eucl(Y,N, empty)	2.03	266	373
Eucl(Y,N, obstacle)	2.34	470	277
Eucl(Y,Y, empty)	0.93	266	132
Eucl(Y,Y, obstacle)	1.41	484	257

Figure 75. Testing of the algorithm with different heuristics with different parameters

As it can be seen from Figure 75, the algorithm is the most efficient with respect to time when using the Manhattan heuristic with diagonal movement disabled, breaking ties enabled in a 20x20 empty map. The algorithm with this particular heuristic can go from a starting node to the goal node in 0.77 milliseconds average. In contrast, the least efficient configuration for the A* algorithm is with the Euclidian heuristic, with diagonal movement enabled, breaking ties disabled and performing in a map filled with obstacles. This particular configuration is completing the route from the start node to the goal node in 2.34 milliseconds average.

In the first testing session, the start node and the goal node were placed in opposite corners. In order to assess even further the behaviour of the algorithm, the start node and goal node must be placed in different positions (on the same row, but in different columns – a figure is provided).

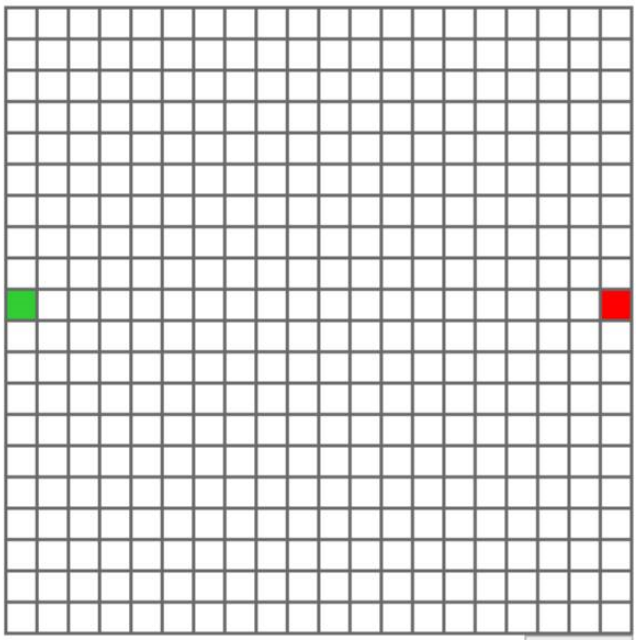

Figure 76. Second test map.

Parameters	Time(milliseconds)	Length of path	Nodes
manh(n,n, empty)	1.06	190	230
manh(n,y, empty)	0.49	190	84
diag(y,n, empty)	1.64	190	306
diag(y,y, empty)	0.72	190	108
eucl(n,n, empty)	1.74	190	235
eucl(n,y, empty)	0.4	190	86
eucl(y,n, empty)	2.13	190	301
eucl(y,y, empty)	0.67	190	106

Figure 77. Algorithm performance for different heuristics when the position of the start node and goal node has been changed

In comparison with the first test which was performed for both an empty map and a map filled with obstacles, this test was performed only for an empty map. The reason why this test was performed only for an empty map was solely due to the fact that it would offer an insight of which algorithm configuration would boost its performance when the start node and the goal node are in a straight line from one another. As it can be seen in the Figure 77 Euclidian heuristic with diagonal movement disabled, breaking ties enabled, allows the algorithm to traverse from the start node to the end node in an average time of 0.4 milliseconds.

In contrast, the least efficient configuration for the A* algorithm is the Euclidian heuristic with diagonal movement enabled and breaking ties disabled. The algorithm will go from the start node to the goal node in an average of 2.13 milliseconds.

5.5 Printing time

The A* algorithm was first tested in an older version of the program which printed the map node by node to the console, and also printed the attributes of the nodes when they changed. The problem with this is that C++ std::cout uses line buffering when printing using std::endl. This means that the programs runtime was highly increased by the time spent on printing into the terminal. Once the printing section of the code was removed, the program ran much faster. Thus, larger testing maps would be necessary to get more accurate data.

This in turn led to discovering the issues with recursive programs in terms of stack growth, which were discussed in the A* section. Only the results from the program which only prints necessary information is included in this report, since the old results did not represent the performance of the algorithm.

5.6 Testing map design

In order to carry tests on the combinations of movement and heuristics, it was decided to test the algorithms with 3 different maps. Attention was focused on a 20x20 map, because it was large enough to see noticeable differences.

Tests were run for three maps, which had to be theoretically biased towards a certain algorithm option. The first testing map was an empty map, with the start point at 1,1 and the finish at 20,20. This map is theoretically favouring algorithms that can move diagonally, since they will have a shorter path, and consider fewer nodes than Manhattan heuristic. This map results can also be used to compare the best case diagonal heuristic with Euclidean.

The second map is also an empty map, but this time the start is at 1,1 and the finish at 1,20. This map should favour Manhattan heuristic in terms of runtime, since they will not need to compute the square root.

Finally, the third map is an arbitrary map with obstacles.

5.7 Drone testing

A drone needs a large area for testing without dangerous consequences, which are difficult to avoid outdoors due to weather. The test was instead performed indoors: in a football hall, which had enough space for safe testing.

Before the test can begin all the connections were double checked and the battery was charged so the longest flight time could be obtained, with a fully charged battery. First tests were done for the general checking of the quadcopter operability. Before allowing the computer control the drone, one should know how to control it in case something goes wrong. Few runs on the field were made and after everything was checked and a user was able to control the prototype properly, further tests could be carried out. The drone was able to fly for around 25 minutes before running out of battery.

To be able to see the motion of the quadcopter with the instructions from the computer, the flight mode “AltHold” (altitude hold) was necessary. For the first test this mode was attempted, but because of too late known reasons it did not work as planned: the motors were simply stopping when this flight mode was selected. Because of this problem, all the tests were made at ground level, since the pilot was not able to hold the quadcopter at a fixed position in the air.

The communication between the prototype and computer was tested one day before the testing session to ensure that the commands are were received and can be executed by the drone. The first

test was made by using just one command, for example, move forward or move backwards. The instructions were sent in character format: “l” (left), “r” (right),” u” (up),” d” (down).

Because the test was made at ground level, the quadcopter was not flying, but gliding. Because each instruction can be represented by the movement of the stick on the remote control, the distance that the quadcopter needed to travel had to be calibrated to be fixed, taking into account some factors, like friction and the irregularities on the landing platform. These factors caused an error in the distance moved, as it can be seen in the video.

After the calibrations were made for each of the movements, a series of instructions were sent to observe how the quadcopter reacted when multiple instructions were sent in a single package. It was observed that, because of the inertia, the prototype was still sliding when the next instruction was executed, creating a diagonal movement effect. This unwanted effect was eliminated by including more delay after each instruction, so the drone has time to stop before executing the next command.

With this problem solved, the quadcopter reacted well to both one instruction and a series of instructions. Unfortunately, the flight mode problem was not solved before the testing time available had ended, so the test was made just at ground level.

Overall the test went as expected with a lot of data that could be used for future improvements of the prototype.

The next step would be making the flight mode work, keeping the quadcopter in a fixed position in the air, and repeat the testing process for the drone in flight. Also some anti-inertia movement should be included in the instruction execution, in the way that after each movement a movement in the opposite direction should be triggered for a short period of time. This will make the quadcopter have a better precision for each move.

A drone needs a large area for testing without dangerous consequences, which are difficult to avoid outdoors due to weather. The test was instead performed indoors: in a football hall, which had enough space for safe testing.

Before the test can begin all the connections were double checked and the battery was charged so the longest flight time could be obtained, with a fully charged battery. First tests were done for the general checking of the quadcopter operability. Before allowing the computer control the drone, one should know how to control it in case something goes wrong. Few runs on the field were made and after everything was checked and a user was able to control the prototype properly, further tests could be carried out. The drone was able to fly for around 25 minutes before running out of battery.

To be able to see the motion of the quadcopter with the instructions from the computer, the flight mode “AltHold” (altitude hold) was necessary. For the first test this mode was attempted, but because of too late known reasons it did not work as planned: the motors were simply stopping when this flight mode was selected. Because of this problem, all the tests were made at ground level, since the pilot was not able to hold the quadcopter at a fixed position in the air.

The communication between the prototype and computer was tested one day before the testing session to ensure that the commands are were received and can be executed by the drone. The first test was made by using just one command, for example, move forward or move backwards. The instructions were sent in character format: “l” (left), “r” (right),” u” (up),” d” (down).

Because the test was made at ground level, the quadcopter was not flying, but gliding. Because each instruction can be represented by the movement of the stick on the remote control, the distance that the quadcopter needed to travel had to be calibrated to be fixed, taking into account some factors, like friction and the irregularities on the landing platform. These factors caused an error in the distance moved, as it can be seen in the video.

After the calibrations were made for each of the movements, a series of instructions were sent to observe how the quadcopter reacted when multiple instructions were sent in a single package. It was observed that, because of the inertia, the prototype was still sliding when the next instruction was executed, creating a diagonal movement effect. This unwanted effect was eliminated by including more delay after each instruction, so the drone has time to stop before executing the next command.

With this problem solved, the quadcopter reacted well to both one instruction and a series of instructions. Unfortunately, the flight mode problem was not solved before the testing time available had ended, so the test was made just at ground level.

Overall the test went as expected with a lot of data that could be used for future improvements of the prototype.

The next step would be making the flight mode work, keeping the quadcopter in a fixed position in the air, and repeat the testing process for the drone in flight. Also some anti-inertia movement should be included in the instruction execution, in the way that after each movement a movement in the opposite direction should be triggered for a short period of time. This will make the quadcopter have a better precision for each move.

6 Conclusion

Obstacle avoidance techniques and path finding are some large areas in computer science that have seen significant development in the latest years and drone technology might have a big impact on life as one sees it in modern times. This was one of the reasons the group attempted to build a drone that can avoid obstacles using distributed systems. The way distributed systems can come in handy when it comes to this subject is that the necessity for an experienced human operator will not be required anymore.

Furthermore, “distributing” this whole assembly into multiple processing units means that the memory management and computational time needed in order to process the instructions the drone sends/receives will be optimized.

Delving into distributed systems and the way one can be implemented, the group faced a lot of difficulties throughout the project time but not without any rewards. During this period, knowledge has been acquired about subjects of little interest beforehand such as communication protocols, other programming languages besides Java (which was studied as a course in the past semesters), how compilers work, how to optimize an algorithm and how to choose the proper algorithm for a specific purpose.

It can be said that although the first approach with respect to prototype implementation was not successful (the use ultrasonic sensors in order to detect obstacles), the group managed to find another practical solution such as providing the prototype with a map of the terrain in which obstacles are known and are pointed out. Basically this approach means inserting “virtual” obstacles in an empty room and the quadcopter could move in the discretized space according to the provided map. Furthermore, using the provided map, the algorithm can determine the shortest path from a certain point on the map to the end point.

In terms of software implementation, the A* algorithm and graphical user interface (GUI) were implemented in C++ using Visual Studio IDE whereas the quadcopters behaviour simulation was implemented in Java using IntelliJ. Through this software implementation in different programming

languages, the group attempted to get a better grasp at the difference between the programming paradigms these languages use.

The learning goals of this project were strongly related to both hardware and software. New hardware modules were used for the first time such as radio communication with the use of X-Bee modules (transceiver) as well as telemetry radio. The group had some encounter with the APM flight controller software as well, but it turned out that the software cannot be changed. Strictly speaking about software, the group has attempted to create high quality software in order to make it easier to grasp and optimize its performance. Some numerical methods procedures were also implemented in software, such as Taylor expansion formula for square root. With this software implementation of a mathematical concept, the group attempted to have a better understanding of some of the concepts which were taught in the Numerical Methods course.

Although this semester project is considered to be a success by the group members, there are obvious improvements that can be made. One of these improvements is uploading a map of virtual obstacles and control the quadcopter in such a way that it will autonomously avoid them in an open space. Further improvements might represent image recognition software which can detect patterns and shapes and in case of an imminent collision, it can autonomously avoid that particular obstacle. Without a doubt, there are many improvements which can be implemented with regard to this subject. As this subject may be not attempted in the future by the group, is it yet to see how engineers and hobbyists throughout the world will attempt this problem and find an optimized solution to it.

References

1) Luukkonen T. Modelling and control of quadcopter [Internet]. Aalto University. 2011 [cited 2015]. Retrieved from: http://sal.aalto.fi/publications/pdf-files/eluu11_public.pdf
2) History of Quadcopters [Internet]. Quadcopters. [cited 2015]. Retrieved from: http://ffden-2.phys.uaf.edu/webproj/212_spring_2014/clay_allen/clay_allen/history.html
3) Goldberg AV. Efficient Point-to-Point Shortest Path Algorithms [Internet]. Efficient Point-to-Point Shortest Path Algorithms. [cited 2015]. Retrieved from: http://www.cs.princeton.edu/courses/archive/spr06/cos423/handouts/epp shortest path algorithms.pdf
4) Kruse GW. Algorithm Efficiency [Internet]. Algorithm Efficiency. [cited 2015]. Retrieved from: http://jcsites.juniata.edu/faculty/kruse/cs2/ch12a.htm
5) *Roberts E. Motion Planning in Robotics [Internet]. Robotics: Motion Planning. [cited 2015]. Retrieved from: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html*
6) *rcgroups. ArduPilot Mega APM 2.6 Flight Controller With 6M GPS From Banggod. http://www.rcgroups.com/forums/showthread.php?t=2043121*
7) *Korey Smith. Best Flight Controllers And Why. http://myfirstdrone.com/tutorials/buying-guides/best-flight-controllers/ (accessed).*
8) *APM 2.5 and 2.6 Overview.copter.ardupilot http://copter.ardupilot.com/wiki/common-apm25-and-26-overview/*
9) *Brown W. Brushless DC Motor Control Made Easy [Internet]. [cited 2015]. Retrieved from: http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf*
10) *Giorgos Lazaridis. How Brushless Motors Work (BLDC Motors). http://pcbheaven.com/wikipages/How_Brushless_Motors_Work/*
11) *Atmel AVR443: Sensor-based Control of Three Phase Brushless DC Motor [Internet]. 2013 [cited 2015]. Retrieved from: http://www.atmel.com/images/atmel-2596-sensor-based-control-of-three-phase-brushless-dc-motors_application-note_avr443.pdf*
12) *Mat Dirjish | Electronic Design. What's The Difference Between Brush DC And Brushless DC Motors. http://electronicdesign.com/electromechanical/what-s-difference-between-brush-dc-and-brushless-dc-motors (accessed).*

13) Dragos Cristian Danila, Radu Nicolae Marica, Rolan Abdulrazzak Ossi,
Paul Claudiu Stoian, Hans Dhariwal , [A study into the control of a Quadcopter on axis of movement], (ED4-2015).

14) Brushless DC Motor, How it works ? [Internet]. ~ Learn Engineering. [cited 2015]. Retrieved from: http://www.learnengineering.org/2014/10/brushless-dc-motor.html

15) *FAQ_Speed Controllers [Internet]. FAQ_Speed Controllers. [cited 2015]. Retrieved from: http://www.gws.com.tw/english/service/faq/speed controllers.htm*

16) *Plush [Internet]. Turnigy. [cited 2015]. Retrieved from: http://www.turnigy.com/esc/plush/*

17) *kedamodel. ElectronicSpeedController Users' Manual SAKER MODEL Advance ESCs for plane. http://www.kedamodel.com/product/doc/ESC%20Users'%20Manual%20for%20saker%20model%20advance.pdf (accessed).*

18) IMAX B6AC version 2 [Internet]. pololu.com. https://www.pololu.com/file/0J857/manual-iMAXB6AC-v2.pdf; 2014 [cited 2015]. Retrieved from: https://www.pololu.com/file/0j857/manual-imaxb6ac-v2.pdf

19) Rolan Abdulrazzak Ossi, Paul Claudiu Stoian, Hans Dharwil, [Study Of Two Wheels Mini-Segway], (ED4-2015).

20) Erwin Kreyszig Advanced Engineering Mathemathics 10 edition p691 s15.4

21) Taylor Series [Internet]. math.ucdenver.edu. http://www.math.ucdenver.edu/~esulliva/Calculus3/Taylor.pdf; [cited 2015]. Retrieved from: http://www.math.ucdenver.edu/~esulliva/calculus3/taylor.pdf

22) The Programming Languages Enthusiast. pl-enthusiast.net. http://www.pl-enthusiast.net/2014/08/05/type-safety/

23) The Programming Languages Enthusiast. pl-enthusiast.net. http://www.pl-enthusiast.net/2014/08/05/type-safety/

24) Texas Instruments Incorporated, KeyStone Architecture Literature Number: SPRUGP1 November 2010 Universal Asynchronous Receiver/Transmitter (UART) http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf

25) Frank Durda, Serial and UART Tutorial. https://www.freebsd.org/doc/en/articles/serial-uart/

26) Electricimp, Introduction to the standard serial bus. https://electricimp.com/docs/resources/uart/

27) Balasubramanian S. MavLink Tutorial for Absolute Dummies (Part –I) [Internet]. http://dev.ardupilot.com/wp-content/uploads/sites/6/2015/05/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf. [cited 2015]. Retrieved from: http://dev.ardupilot.com/wp-content/uploads/sites/6/2015/05/mavlink_for_dummiespart1_v.1.1.pdf

28) "Joseph A M. Vulnerability Analysis of the MAVLink Protocol for Command and Control of Unmanned Aircraft [Internet]. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA598977. 2013 [cited 2015]. Retrieved from: http://oai.dtic.mil/oai/oai?verb=getrecord&metadataprefix=html&identifier=ada598977

29) Encyclopedia of Computer Science, John Wiley, 4th Edition, pages 476-479

30) Felic B. Implementing The CCITT Cyclical Redundancy Check [Internet]. Dr. Dobb's. 2007 [cited 2015]. Retrieved from: http://www.drdobbs.com/implementing-the-ccitt-cyclical-redundan/199904926

31) Numerical Recipes in C: The Art of Scientific Computing, Second Edition: Chpt 20.3 (896-903)

32) Singh A. Resolving data collision in csma via protocols [Internet]. Resolving data collision in csma via protocols. 2011 [cited 2015]. Retrieved from: http://www.slideshare.net/anu103/resolving-data-collision-in-csma-via-protocols

33) Introduction to LAN/WAN [Internet]. http://web.cs.wpi.edu/~emmanuel/courses/cs513/S10/pdf_slides/mac3.pdf. [cited 2015]. Retrieved from: http://web.cs.wpi.edu/~emmanuel/courses/cs513/s10/pdf_slides/mac3.pdf

34) Medium Access Control (MAC) Protocols for Ad hoc Wireless Networks - II [Internet]. http://www.cs.jhu.edu/~cs647/mac_lecture_2.pdf. [cited 2015]. Retrieved from: http://www.cs.jhu.edu/~cs647/mac_lecture_2.pdf

35) IEEE 802.11 MAC [Internet]. https://www.cs.purdue.edu/homes/park/cs536-wireless-3.pdf. [cited 2015]. Retrieved from: https://www.cs.purdue.edu/homes/park/cs536-wireless-3.pdf

36) The Data Link Layer [Internet]. http://people.ee.duke.edu/~romit/courses/f07/material/5-link.pdf. [cited 2015]. Retrieved from: http://people.ee.duke.edu/~romit/courses/f07/material/5-link.pdf

37) Bonaventure O. Computer Networking : Principles, Protocols and Practice [Internet]. The datalink layer and the Local Area Networks —. 2014 [cited 2015]. Retrieved from: http://cnp3book.info.ucl.ac.be/1st/html/lan/lan.html#carrier-sense-multiple-access-with-collision-detection

38) Lesson 1: Access Methods [Internet]. Lesson 1: Access Methods. [cited 2015]. Retrieved from: http://pluto.ksi.edu/~cyh/cis370/ebook/ch03b.htm

39) Token Ring/IEEE 802.5 [Internet]. - DocWiki. 2012 [cited 2015]. Retrieved from: http://docwiki.cisco.com/wiki/token_ring/ieee_802.5

40) Algorithmics of Large and Complex Networks pp 117-139

41) Introduction to A* From Amit's Thoughts on Pathfinding [Internet]. Introduction to A*. 2015 [cited 2015]. Retrieved from: http://theory.stanford.edu/~amitp/gameprogramming/astarcomparison.html

42) std::vector [Internet]. - cppreference.com. 2015 [cited 2015]. Retrieved from: http://en.cppreference.com/w/cpp/container/vector

43) C Without Fear Second Edition [Internet]. 2012 [cited 2015]. Retrieved from: http://ptgmedia.pearsoncmg.com/images/9780132673266/samplepages/0132673266.pdf

44) 7. Memory : Stack vs Heap [Internet]. 7. Memory : Stack vs Heap. 2012 [cited 2015]. Retrieved from: http://gribblelab.org/cbootcamp/7_memory_stack_vs_heap.html

45) Algorithms [Internet]. Algorithms. [cited 2015]. Retrieved from: http://web.stanford.edu/~msirota/soco/inter.html

46) http://www.hobbyking.com/hobbyking/store/catalog/49725s6.jpg

47) http://www.hobbyking.com/hobbyking/store/catalog/49725.jpg

48) http://www.hobbyking.com/hobbyking/store/catalog/66416.jpg

49) http://www.hobbyking.com/hobbyking/store/catalog/TR-P30A.jpg

50) http://cdn6.bigcommerce.com/s-xkoep7/products/771/images/3439/amp2.8_flight_controller_1__98021.1434627348.1280.1280.jpg?c=2

51) http://cdn.shopify.com/s/files/1/0264/2161/products/MRP.jpg?v=1415205820

52) http://www.rcparts.eu/images/detailed/2/turnigy-9x-reciever-9x8cv2.jpg

53) http://d2rormqr1qwzpz.cloudfront.net/photos/2013/06/12/48912-arduinouno_r3_front.jpg

54) http://ep.yimg.com/ca/I/yhst-27389313707334_2252_127389036

55) Numerical analysis [Internet]. math.pitt.edu/. [cited 2015]. Retrieved from: http://www.math.pitt.edu/~trenchea/math1070/math1070_2_error_and_computer_arithmetic.pdf

56) Heuristics From Amit's Thoughts on Pathfinding [Internet]. Heuristics. 2012 [cited 2015]. Retrieved from: http://theory.stanford.edu/~amitp/gameprogramming/heuristics.html