Drones for offshore spare part delivery

Why are drones not used for delivering small packages to offshore locations?

Boris Fabula, Iasonas Theodoros Tomadakis-Bamford, Mateusz Matejko, Nils Albin Emil Nolemo, Norbert Pap, Tamás Halasi, Tue Burmeister Jensen

Applied Industrial Electronics, AIE21-B324b, 2022-05

2nd Semester Project

AAU Energy
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Drones for offshore spare part delivery

Theme:
Microprocessor based systems

Project Period:
Spring Semester 2022

Project Group:
AIE21-B324b

Participant(s):
Boris Fabula
Iasonas Theodoros Tomadakis-Bamford
Mateusz Matejko
Nils Albin Emil Nolemo
Norbert Pap
Tamás Halasi
Tue Burmeister Jensen

Supervisor(s):
Mads Valentin Bram

Cosupervisor(s):
Dil Muhammad Akbar Hussain

Page Numbers: 63

Date of Completion:
May 25, 2022

Abstract:

The offshore industry is currently going through changes in how it operates, transitioning from mainly oil and gas production to a larger share of renewable energies. The industry is making efforts to lower its carbon footprint in many areas, one of them being their logistical operations to the offshore platforms. The aim of this report is to investigate how the environmental footprint of offshore spare part delivery can be reduced by designing an autopilot and monitoring system for drones. This is done in order to find out if it could potentially reduce the use of fuel-run helicopters and sea vessels. The system is designed in three parts, take off and landing, navigation and monitoring. This is so that it could potentially be integrated into a physical drone and fly by itself, and at the same time be monitored at all times by an operator. The system was built and put through certain tests in order to verify that they act according to the intended function. The results are promising, meaning that further research into this area would be worthwhile.

Preface

This project report was done by second semester Applied Industrial Electronics students in the laboratory of the Esbjerg campus of Aalborg University. Collaboration with the innovation department of Semco Maritime and Verdensmålshuset helped steer the direction of the project by giving an insight into the needs of a maritime transport and service provider, together with current sustainability concerns of industry development.

Contents

Chapter 1

Introduction

Logistics is a field in which large systems of planning and transportation come together in order to make sure everything is in the right place at the right time. The global logistics industry reached a market size of 6 trillion euros in 2019, highlighting its size and importance[47]. Given the size of the industry, the importance of sustainability and lowering of emissions is rising, as transportation currently stands at an estimated 16.2% of global emissions[22]. In order to meet with several of the UN's sustainable development goals[38], such as:

9. Industry, Innovation and Infrastructure

12. Responsible Consumption and Production

13. Climate Action

It is important that companies within the industry and customers of said industry focus on lowering of emissions in this sector.

A challenge initiated by Verdensmålshuset and maritime transport and service provider Semco to Aalborg University Esbjerg led to a collaboration on how to reduce transportation emissions within the offshore industry. The importance of logistics issues were highlighted with a focus on the offshore industry in which Semco operates, mainly in offshore wind and oil and gas applications. This challenge and discussion with the challengers led to the following initiating problem in order to research this topic further:

Is it possible to reduce the environmental footprint of small item transport in relation to offshore logistics?

Chapter 2

Problem analysis

2.1 Offshore logistics

The offshore energy industry is one that has comprised mainly of oil and natural gas extraction, but increasingly offshore wind power is being set up throughout the world. A report by the US Department of Energy gives an estimate of offshore wind potentially growing to 215GW installed capacity by 2030, compared to the existing 33GW in 2020[35]. A report by the International Energy Agency predicts oil demand falling in the long term, but it also predicts continued growth for natural gas, meaning the offshore energy industry will be working with renewable technologies and fossil fuels side by side for the foreseeable future, continuing to give access to large amounts of oil, gas and increasing amounts of renewable energy[21]. Innovations in offshore energy production are also arising, which may create new challenges with regards to planning and logistics. An example of innovation in this sector is the Energy Island proposed in the North Sea in Denmark, which will be used as a hub for wind power production and potentially Power-To-X applications[2]. Offshore projects constantly need to transport items to and from the sites, both during construction, commissioning, maintenance and decommissioning of the sites. Here anything from large turbine wings to small screws may need to be delivered, which presents an array of problems, with different size requirements and potentially differing time pressures. Changing the ways in which logistics is done in this industry using novel technologies could potentially lower costs and emissions.

Semco explains that the current modes of transportation for the largest building blocks to small part deliveries are done using varying sizes of marine vessels and helicopters. Deliveries can be subject to differing time constraints and size

which may influence the choice of vehicles. Larger and less critical deliveries generally fit the choice of vessels, while time critical parts may be deployed on a helicopter, particularly if delays in production could harbour substantial financial consequences. Faster and more flexible deliveries are thus an important aspect the industry could improve on.

2.2 Critical delivery

One example of small, critical package delivery is the delivery of spare parts, tools and equipment to offshore sites. From more routine, well-defined operations such as regular maintenance, the rising need for decommissioning of oil rigs and the development of new projects like the Energy Island could come with uncertainties that raise the need for small and flexible deliveries. According to the IEA, offshore oil and gas decommissioning will rise in both amount and complexity up until the 2040s, as more and more deep sea platforms will need dismantling[21]. This added on top of the already dynamic nature of running these platforms, which may also derive benefit from novel solutions.

Offshore sites are critical for the energy supply of the population of their respective countries, thus maintaining consistent production is especially important, particularly now in a time when the European Union has stated it is aiming for energy independence and lowering carbon emissions simultaneously[41].

From these motivations, investigating novel logistical solutions for smaller item deliveries is worthwhile, seen both from an environmental and economical standpoint. To investigate it further it is first necessary to examine the current solutions.

2.3 Current solutions

2.3.1 Vessels

Seafaring vessels are a natural choice for offshore environments, and they are common in these industries. They have high load capacities, and range is not a limiting factor, which makes them the ideal heavy load carrying mode of transportation. They also require crews to operate them, who could potentially be exposed to considerable risks moving through unfavourable weather and sea conditions[37]. Vessels are also relatively slow moving, make them unsuitable for time critical use-cases.

2.3.2 Helicopters

Helicopters are frequently used vehicles for the transportation of humans and smaller items to and from offshore sites, and are expected to grow in usage in offshore applications the coming years[27]. Helicopters come with the immediate benefit of being faster than boats, but have a far lower carrying capacity than vessels, and substantially higher costs. A study by the World Bank explains that air freight is typically 12-16 times more expensive per weight than sea transport[6]. Although this study observes airplane freight, this likely holds in principle for helicopters as well. The same is true for emissions, where air freight emits on an order of magnitude 36 times the amount of CO_2 than sea freight does[45]. They are better suited for last minute type deliveries for this reason, but are hindered by the fact that they are more expensive to operate and are substantially higher in emissions.

Even though there are solutions for supply deliveries, there is a potential market gap with regards to smaller deliveries, especially for cases where the size of the load does not justify the usage of an entire helicopter or vessel. In particular, there is a need for a quick, flexible option with lower emissions.

2.4 Connecting to the UN goals

Connecting the information acquired from the studies, articles and a meeting with Semco together with the UN sustainable development goals lead towards the following:

- 9. Industry, Innovation and Infrastructure
  Economic benefit could be gained by changing the way smaller deliveries offshore are made, as helicopter flights are both expensive and emission heavy. These benefits could potentially propagate into other logistical industries as well.

- 12. Responsible Consumption and Production
  Delivering only what is necessary in a more efficient manner, reducing the amount of non-full capacity trips with vessels or helicopters could lower the amount of trips and fuel consumption, consuming the minimal amount of resources needed to deliver these items. Streamlining resource usage could lead to lowering the impact in the offshore industry specifically, but the knowledge gained could also be utilized elsewhere.

- 13. Climate Action
  Lowering the usage of emission heavy vehicles and utilizing lower emission

options whenever possible would be an environmental benefit. This is crucial in order to meet with Paris Agreement goals[39].

Sea vessels are able to carry large volumes of material at a relatively low emission count, but are limited by their speed, while helicopters have the reverse benefit. Therefore a transport method with high speed and lower emissions that can carry smaller items could potentially fill a hole in the market. Unmanned aerial vehicles (UAVs) could provide several advantages as they are relatively fast, can carry small loads and can potentially be built to run on batteries, benefiting both economically and environmentally.

2.5 UAVs for offshore delivery

In this section the following words and abbreviations will be used with respective meanings.

- **UAV** - Unmanned aerial vehicle.
- **Drone** - A multi-rotor UAV that depends on propellers to generate lift.
- **Airplane** - A UAV that depends on wings to create lift, also called fixed-wing frame or configuration.
- **VTOL** - A hybrid airplane that has a frame equipped with multiple motors aimed in a vertical direction allowing for vertical take-off and landing.

2.5.1 Overview of available products

This section provides the results of a market research of drones and VTOLs. The aim of the research is to compare the specifications and discover if a product capable of offshore delivery tasks is readily available for purchase. The results indicate that a sufficiently large UAV able to carry a reasonably sized payload can be bought and modified for the purpose. Moreover, the results are used for a comparison of types of UAV, namely a drone or a VTOL. Fixed wing configuration of a VTOL performs the best in range, however, airplanes have lesser capacity than drones in payload size, and also the payload has to be loaded in a way that does not negatively affect the center of gravity of the airplane. In offshore delivery the payload size has higher priority than achieving the longest range, rendering drones more suitable. A comprehensive comparison of selected relevant specifications of some commercially available drones is shown in Table 2.1.

Product name	Payload	Flight time	Range	Wingspan or size	Type
DJI Matrice 300 [9]	4 kg	55 min	15 km	0,81 m	Drone
DJI Matrice 600 Pro [10]	6 kg	16 min	5 km	1,7 m	Drone
Nightgale Security [42]	–	33 min	12 km	0,6 m	Drone
Deltaquad Pro [8]	1,2 kg	170 min	100 km	2,35 m	VTOL
L3Harris FVR-90 [32]	10 kg	16 hrs	1500 km	4,7 m	VTOL
L3Harris FVR-55 [31]	4,5 kg	10 hrs	100 km	4 m	VTOL
L3Harris FVR-10 [30]	1 kg	1 hr	–	2,03 m	VTOL
Ukrspecsystems PD-2 [48]	11 kg	10 hrs	–	5 m	VTOL

Table 2.1: Product comparison Dash lines are entries which were not publicly available

Table 2.2 summarizes found specifications into a comparison of UAVs based on the type of frame. Firstly, electric current requirements are described relatively to other UAV types. Since fixed-wings generate lift during flight by their design, this configuration requires less power than a drone that has to generate lift with its motors constantly, or a VTOL capable plane during vertical take-off and landing. This allows for the long flight time of fixed-wings. Furthermore, for payload and speed, only the maximum values were selected for the table, which are naturally achieved by the larger drones. Attention should be paid to the fact that drones are capable of carrying significantly heavier payloads while still being smaller in size compared to a fixed wing UAV. It can therefore be concluded that in a use-case where payload capacity is prioritized, drones are more suitable even though their range and flight time is shorter.

Drones comparison	Multi rotor	Hybrid fixed-wing (with VTOL)	Fixed-wing
VTOL	Yes	Yes	No
Flight time	15 - 60 min	25 min - 16 hours	30 min - 18 hours
Electric current requirements	High	High only during take-off and landing	Low
Size	0.5 - 3 m	2 - 5 m	2 - 5 m
Payload	90 kg[11]	11 kg	19 kg
Speed	90 km/h	120 km/h	120 km/h

Table 2.2: UAV comparison

2.6 Stakeholders

When looking for a solution to a specific problem it is vital to assess the relevance of the researched solution. This section considers industries and companies

that could potentially benefit from this technology.

Drone delivery companies They especially focus on delivery using drones, so any improvement on their technology means potential profit for the companies. Examples include Wingcopter[51], Wing[50] and Zipline[52].

Institutions and companies which provide healthcare Currently in Denmark, crucial medical products are delivered by emergency vehicles, and the usage of drones could be a cheaper and/or more efficient alternative in some cases.

Companies conducting offshore spare part deliveries Examples include Semco Maritime. The company currently conducts deliveries using vessels and helicopters, both of which have disadvantages, described in Section 2.3. Delivery drones could improve on the energy consumption of helicopters while flying with a similar speed, which means a more economical and less environmentally impactful way of transport for the company.

Companies conducting offshore work The workers' efficiency increases if they can get the missing tools delivered to them more quickly and cheaply, thus they might indirectly benefit from the adoption of delivery drone technology.

2.7 Legislation

Knowing the laws and regulations that are imposed on drone flight is crucial in understanding where and how drones could be used to solve a certain problem. When using a UAV there are some specific rules the operator must follow set forth by the European Union Aviation Safety Agency (EASA). These are divided into three different categories based on the risk of the operation: open, specific, and certified[13]. All of these categories require different levels of certification.

The open category, as defined by the EASA, covers the lowest risk operations. It is further divided into three different categories: A1, A2 and A3. A1 covers drones smaller than 500g: they are not allowed to fly over crowds of people. A2 covers drones smaller than 2kg: they are not allowed to fly within a 50m distance of people not involved with the flight. A3 covers drones less than 25kg: they must fly at least 150m away from homes and industrial areas[14].

The specific category, as defined by the EASA, covers operations riskier than the ones in the open category. In this category the operator must get authorisation from the National Aviation Authority (NAA), which in Denmark is the Danish Civil Aviation and Railway Authority, unless the operation is a standard sce-

nario[15]. There are two standard scenarios, implemented by the European Commission in May of 2020: STS-01 and STS-02. STS-01 is defined as any operation that can be performed at most 120m above the ground with visual line of sight (VLOS), in a controlled ground area in a populated area. STS-02 is defined as any operation within a controlled ground area in an area with low population and can be conducted beyond visual line of sight (BVLOS) of at most 2km from the operator, at most 120m above the ground while also having airspace observers[17]. If the operation is covered by these standard scenarios then the operator has to go through a design verification process of the mission. This involves making a thorough design description and notifying the NAA, who then have to approve the mission[15].

The certified category, as defined by the EASA, covers operations with a high level of risk. This category is essentially for all operations requiring exemptions or amendments to regulations. Currently the EASA is working on addressing how these regulation changes should happen. Right now they are focusing on three main areas. The first being using drones to fly cargo internationally. The second is using drones in urban or rural places for operations like drone taxis or package delivery, and The third being the same operations as the second one, but where the pilot is on board at first but later will be remotely operating the drone[12].

While it is possible to get exemptions for some of these regulations, there is a lot of legislation holding back the market for commercial use of drones. One of the challenges is the legislation about BVLOS. The legislation causes the use of drones to be cumbersome, needing to have airspace observers when flying BVLOS in most cases of offshore application. This reduces the viability of such a solution since it makes it less economically viable. The normal operation for offshore is likely to fall into the certified category, making it an unpredictable area to work within since these regulations are in the process of changing.

2.8 Problem definition

Using offshore sites for energy supply generation is expected to increase[24]. That will likely result in an increased need for maintenance[25], which means an increased need for spare part delivery. As the amount of deliveries increase, the impact of the transporting vehicle becomes more visible. Current technologies such as helicopters and vessels either use fuel which emits a non-negligible amount of CO_2, or are not suitable for time critical deliveries to and from offshore platforms. Due to the considerable negative environmental impact of CO_2 emissions, companies conducting deliveries to offshore sites such as Semco Maritime are looking for ways to reduce emissions. Using UAVs might be one of these ways. A prototype of such solution is described in Chapter 3.

Chapter 3

Problem solution

3.1 Problem delimitation

Section 2.8 concluded that a promising solution to the offshore logistical challenges is the usage of UAVs. Potentially suitable UAVs include an unmanned helicopter or a multicopter, possessing high payload capability.

This project solves an important part of the problem of developing a suitable drone, namely the autopilot and continuous monitoring. It will focus on the design and construction of a system which:

1. Sets up sensors that are required for autonomous flight and handles their signals using interrupts and polling
2. Processes the data from the sensors and calculates how the drone should behave in order to reach the target destination
3. Displays this data, and data which might be important for the person monitoring the flight of the drone.

The project showcases how the sensors can be used to calculate autonomous behaviour of the drone. Additionally, testing is conducted in order to verify that all parts of the system are working as intended.

In the case of an unmanned autonomous drone, the data which shows how the drone should behave would be sent to the flight controller instead of just being displayed. Furthermore, interaction with the flight controller and the control theory required to do so, deciding whether the operator or the autonomous algorithm should be controlling the drone at a given moment, and similar problems are not

discussed in this report.

3.1.1 Prototype specification

Following these limitations, specifications for the different goals were defined. These were defined as different sections called take off & landing, navigation, and monitoring. The specifications for each of these were:

- Take off and landing: It is necessary that the system measures distance travelled in vertical direction, and that it detects the ground below it for safe landing. It also needs to calculate the information it would need to provide to the flight controller throughout these procedures for the drone to reach target altitude, and stay on target coordinates at the same time.
- Navigation: The system is required to receive destination coordinates, and based on its current location and compass measurements, it has to be able to calculate the data that would be sent to the flight controller for the drone to reach the destination, while staying within proximity of the target altitude during the procedure.
- Monitoring: Make a system that can automatically receive navigation data and display it on a remote screen.

3.2 Prototype description

To build a drone capable of autonomous flight, a system based on Arduino Mega microcontroller (MCU) programmed with autopilot software was developed. This system does not replace a drones internal flight controller (FC), instead it is intended to command it via PPM bus, as shown in Figure 3.2. The prototype is implemented on two MCUs, with Arduino Mega running the autopilot software being the main one, and the complementary MCU embedded in the on screen display (OSD) module is used for system monitoring. A picture of the prototype is shown in Figure 3.1.

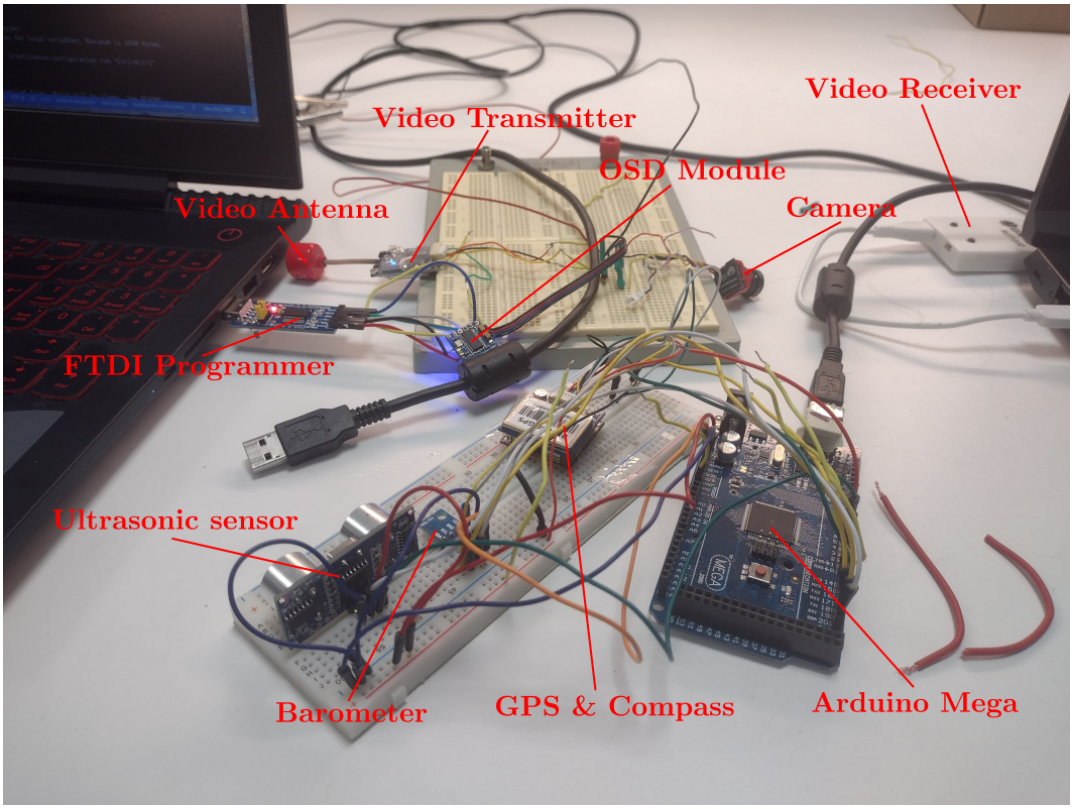

Figure 3.1: The prototype. The bread board in the front contains the sensors used, the board on the right is the Arduino Mega, and the bread board at the back contains the wiring for the OSD module. The video transmitter is the component with the red cylinder next to the laptop on the left, while the video receiver is the white component on the right of the image.

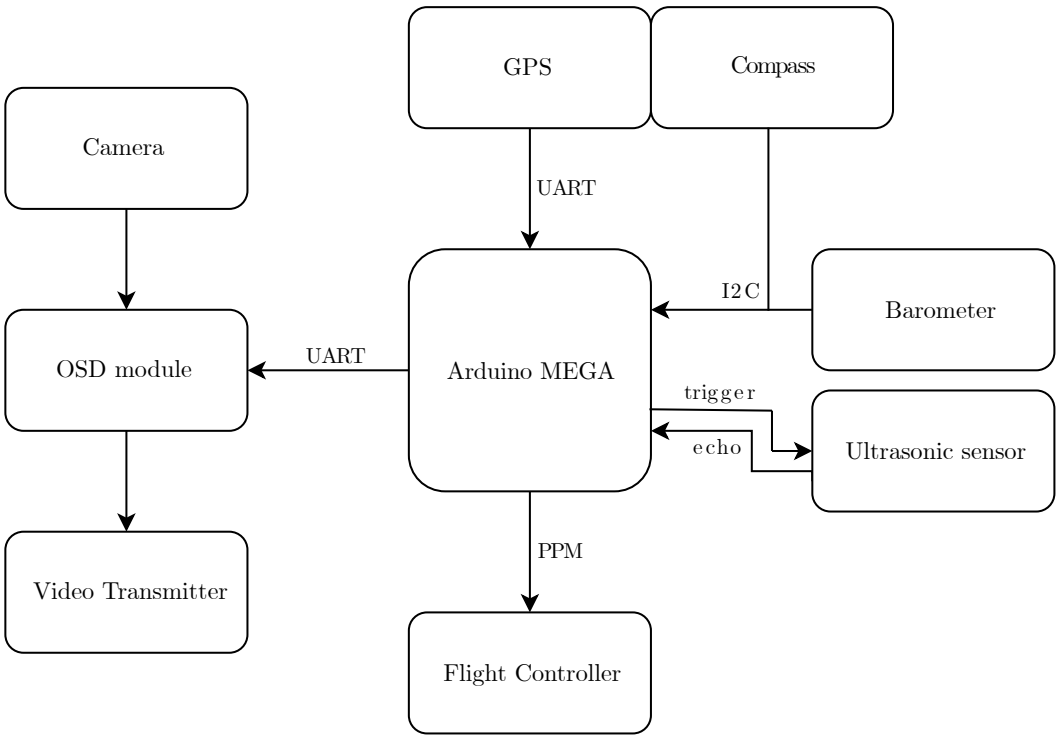

Figure 3.2: The block diagram of the prototype

Figure 3.3 shows the flowchart describing how the prototype behaves.

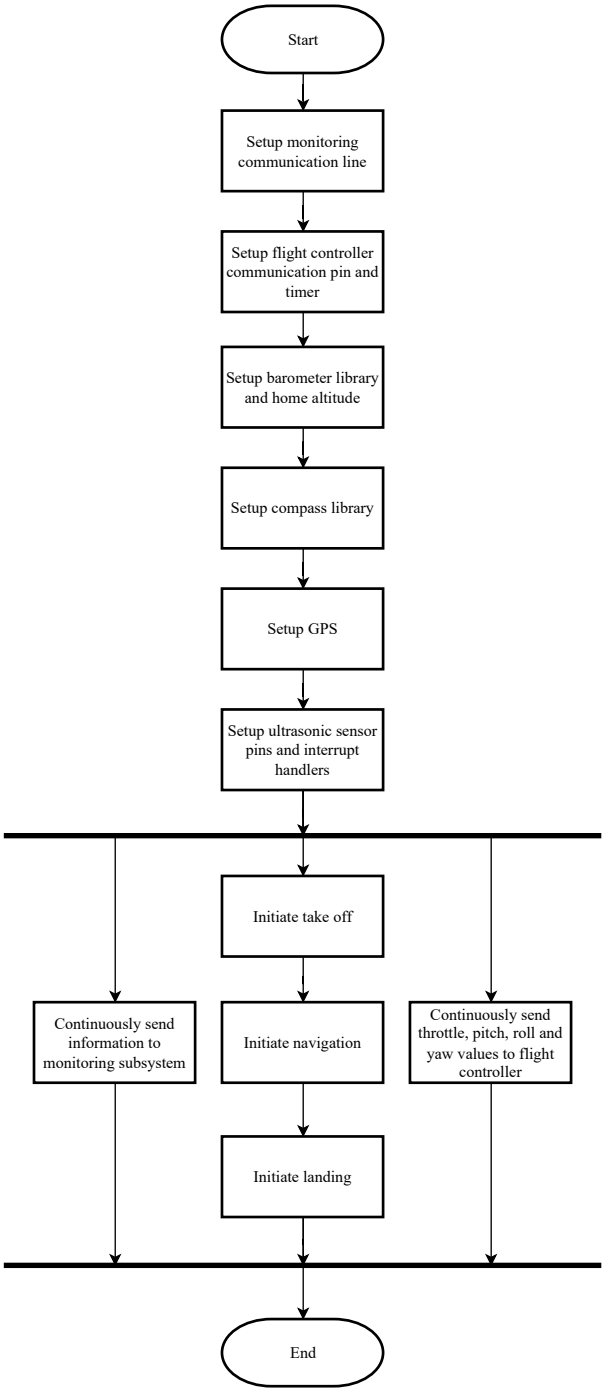

Figure 3.3: Flowchart of the entire prototype

Figure 3.4 shows the wiring of the Arduino Mega, specifics .

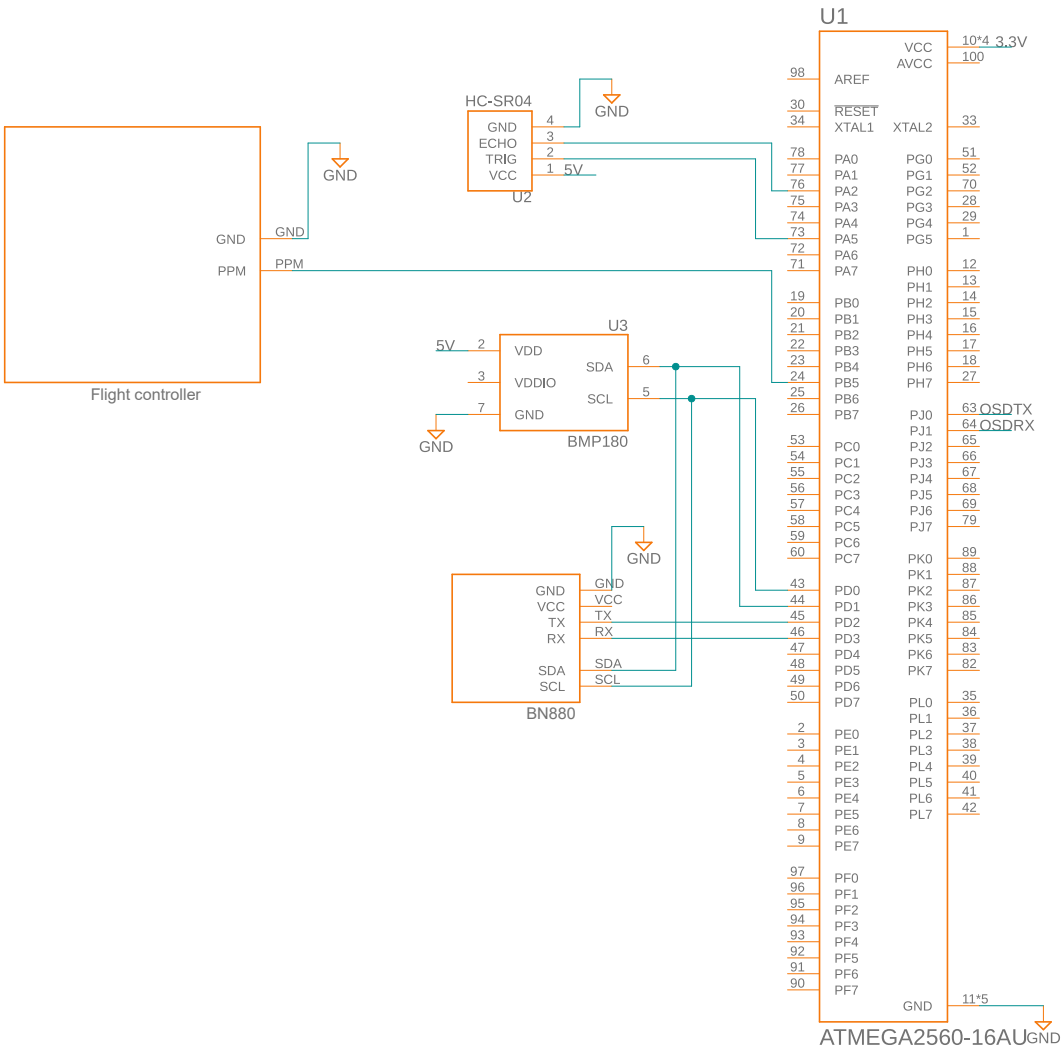

Figure 3.4: Wiring diagram for the Arduino Mega

Figure 3.5 shows the wiring of the OSD module, the specifics of which is discussed in Section 3.4.

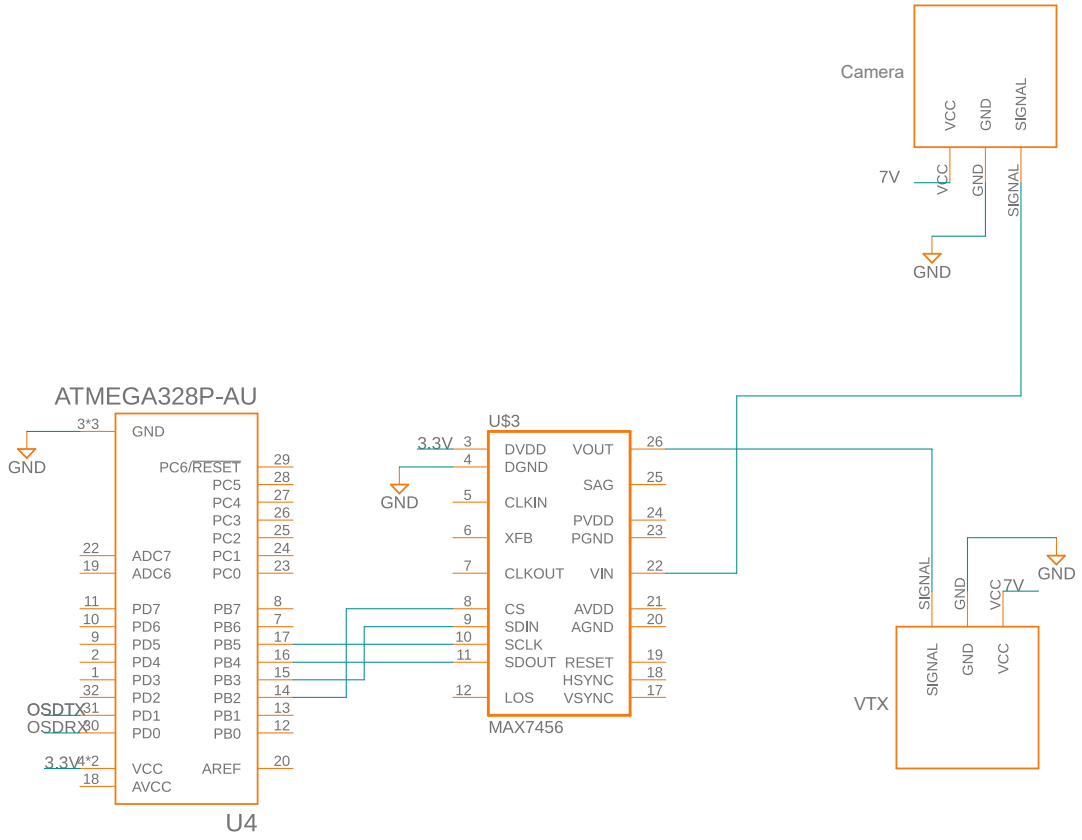

Figure 3.5: Wiring diagram for the OSD module

3.3 Autopilot

This section will use the following definitions.

- **Arm** - Arming the drone means enabling the motors of the drone.
- **Throttle** - The vertical movement of the drone.
- **Pitch** - Pitch is the circular motion on one of the axes, which results in the drone tilting forward or backward.
- **Roll** - Roll is the circular motion on another axis, which results in it rotating clockwise or counterclockwise from the point of view of an observer looking at the drone from the front or back.
- **Yaw** - Yaw is the circular motion on the last axis, which results in it rotating clockwise or counterclockwise from the point of view of an observer looking at the drone from the top or bottom.

The autonomous navigation of the drone consists of three main stages. Firstly take off stage, then navigation and autonomous flight to the destination, and lastly landing stage. Take off and landing are similar in behaviour so they can be described together in one part since they mostly use the same algorithms, with navigation mid-flight being the second part.

3.3.1 Take off and landing

Take-off and landing rely mainly on a barometer and an ultrasonic sensor. However, GPS data is also used to prevent drifting from the desired location caused by wind.

Barometer For altitude measurement a barometer is utilized. A Barometer module based on Bosch BMP180 sensor was chosen for its precision. In addition to altitude data this sensor also provides temperature measurements, which allows for temperature correction of the measured altitude, helping the sensor achieve precision of up to 0.17m[43]. With increasing altitude, atmospheric pressure decreases in a predictable way. At higher temperatures the air expands, thus lowering air pressure. This causes different pressure readings at the same altitude at different temperatures. Therefore, to get the highest precision possible, the sensor combines both temperature and pressure readings to calculate altitude.

The barometer uses a piezo-resistive sensor (a material that changes its resistance based on the change of its shape caused by elastic deformation[18]) to measure the air pressure. The sensor is used in a voltage divider, whose output is connected to an analog to digital converter and a control unit with memory containing calibration data[43]. The unit uses I2C communication to interface with the Arduino Mega, which is a certain type of serial communication, meaning that bits of data are sent through a wire one at a time. To be able to interpret the bits, a clock signal is required for synchronization of intervals when the values change on the data line. This communication protocol has dedicated pins on the Arduino Mega, which are pin 20 for the data line and pin 21 for the clock line. The barometer shares these pins with other devices that use the I2C protocol. The implementation details are abstracted away by the libraries used, provided by Adafruit, an open-source hardware company[1].

Ultrasonic sensor An ultrasonic sensor is used to measure proximity to the ground when landing. This is necessary because the altitude of the drone above sea level is not enough to know where the ground is. In other words, the barometer measures relative altitude change from starting point, and ultrasonic sensor detects the distance from ground for safe landing. The sensor selected for this is the HC-SR04,

because it was available, easy to implement and sufficiently precise when measuring distance to a flat surface perpendicular to the sensor. It has a maximum range of 4 meters[16].

To interface with the sensor, two MCU pins are used, trigger and echo. The internal logic of the sensor starts measuring after a 10 microsecond pulse on the trigger pin, and after the measurement is finished, it sets the echo pin value to high for a time proportional to the measured distance. Interrupt handlers are attached to listen to changes to the value of the echo pin. By default it is set to low by the ultrasonic sensor. Measurement starts with the rising edge, where the Interrupt Service Routine (ISR) records the time of the pin value change. Measurement stops with the falling edge, where by calculating the time elapsed during the high state of the echo pin, the distance can be calculated by the formula Equation (3.1)[16].

$$d = \frac{t_{echo}}{58\mu s \cdot cm^{-1}} \tag{3.1}$$

The unit of d is centimeters, while t_{echo} is in microseconds.

Since the ultrasonic sensor only works up to 4 meters, one measurement cycle can take maximum 23200 microseconds as described in Equation (3.2).

$$400cm = \frac{t_{echo}}{58\mu s \cdot cm^{-1}} \Leftrightarrow t_{echo} = 23200\mu s \tag{3.2}$$

A 24 milliseconds limit was implemented: if the measurement takes more than that, it will be considered to be an error, and is denoted by a distance of -1.

During landing, the system performs continuous measurements with the ultrasonic sensor to calculate the distance to the ground. Thus, pulses on the trigger pin have to be sent out as frequently as possible, and whenever a measurement is ready, it has to be saved into memory. The autopilot cannot simply block the execution of the program until the measurement results are available, since no other process could be executed in the meantime. That would result in losing control of the drone. This means that when the program requires a reading of the ultrasonic sensor, it does not get the current distance from the measurement. Instead, the previous one is used from the memory and it can start a new measurement, which will be processed asynchronously by the interrupt handler, and saved as the last measurement. Since these measurements are finished at most in 24ms, the data is recent enough for the program to decide on what to do next.

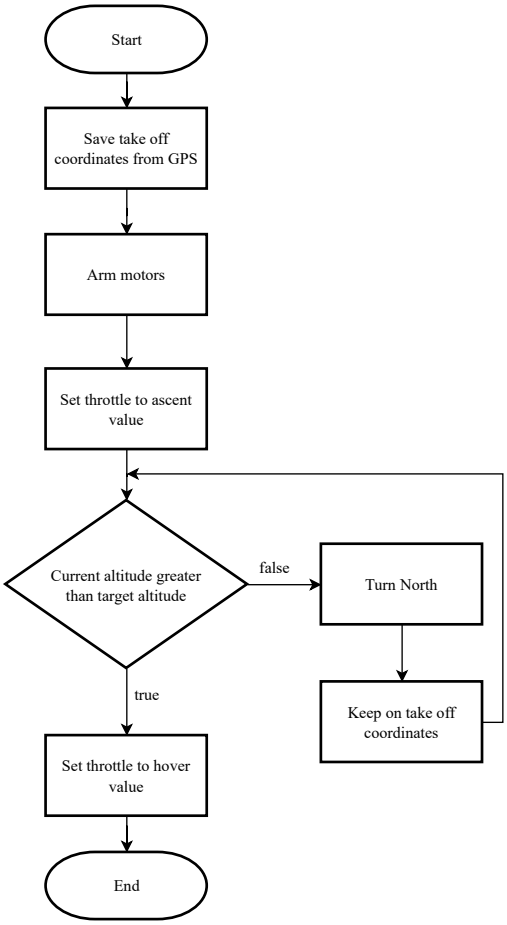

Figure 3.6: Flowchart describing the take-off procedure

Behaviour of system The take off procedure starts after all the systems are set up properly. Once the libraries handling the communication with the barometer are initialized, the home altitude used for all relative altitude measurements is saved, the pins of the ultrasonic sensor are set up with the interrupt handlers registered on them, and the GPS starts reading valid values after it fixes to the satellites, then the take off procedure commences. The flowchart describing the algorithm can be seen on Figure 3.6.

As the first step, the home GPS location is saved, since it will later be used as a reference point where the drone should stay throughout the take off to prevent drifting and colliding with obstructions that might be present around the take off zone. Next, the motors of the drone must be armed so that the propellers can start spinning, and values for the throttle can be set. The throttle has to be set to a value where the propellers start to generate enough lift for the drone to start gaining

altitude.

After the drone is at a state where it is constantly gaining altitude, it continues that until it reaches the target altitude. This altitude is configured to an arbitrary altitude deemed to be high enough for the navigation procedure to begin. It is a number relative to the home altitude recorded previously by the barometer. Until that altitude is reached, the program assures the drone stays above the initial GPS coordinates throughout take-off.

To maintain the position, the drone needs to continually ensure it is heading towards north. This is necessary for the position correcting algorithm, since it simplifies the calculations. If the drone is headed towards north, the latitude and longitude difference can be independently corrected using only single axis motion: to correct the difference in latitude, the drone has to pitch forward to increase its latitude (since latitude increases towards north), or pitch backward to decrease it. Similarly, to correct the difference in longitude, the drone needs to roll to the left to increase its longitude(since longitude increases to the west), or right to decrease it. The flowchart describing the algorithm can be seen on Figure 3.7.

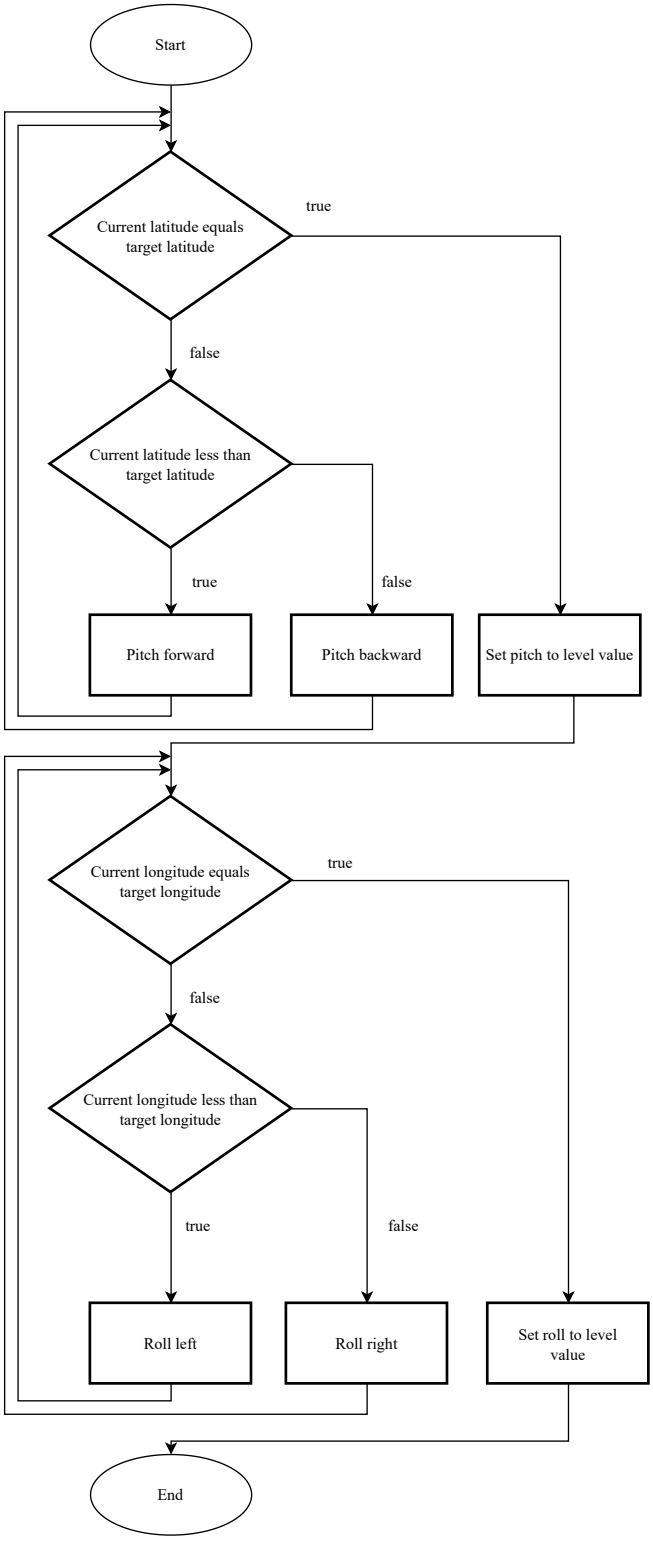

Figure 3.7: Flowchart describing the algorithm used to stay on a target coordinate

Once the drone reached the target altitude, it starts to hover by setting the value of the throttle so that the motors cancel the gravitational force acting on the drone. After this point, it will hand over control to the navigation procedure discussed in Section 3.3.2.

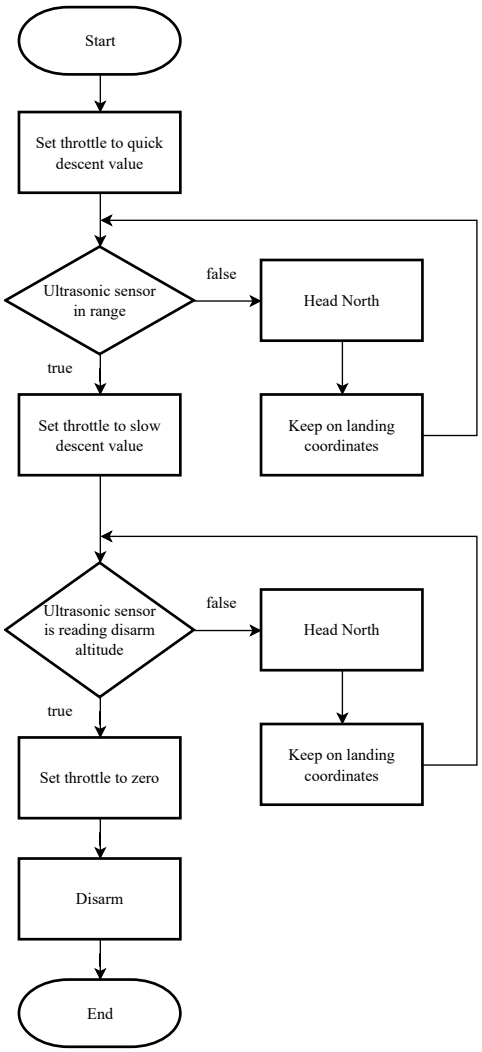

Figure 3.8: Flowchart describing the landing procedure

Once the navigation procedure is over and the drone is above the target location, it proceeds to land. Since the height of the landing zone is unknown, the barometer cannot be used for checking the proximity of the ground during this phase. This is the reason why ultrasonic sensor is used instead, other than the much higher precision that can be achieved by using it. However, since the maximum range of the ultrasonic sensor is 4m, for most of the time during descent, the

ultrasonic sensor cannot be used to measure distance. This breaks up the landing phase into two distinct parts: one where ultrasonic sensor is out of range, and the other where it is in range and thus is used to measure the precise distance between the drone and the landing platform, as seen in Figure 3.8.

To start off the procedure, the drones throttle is set to a value that is not enough for the drone to stay at the same height, thus it starts losing altitude. In this phase the drone is still high above the ground, so the speed of descent can be relatively high to reduce the time it takes to reach the ground. This throttle value is denoted by a constant called quick descent throttle. While waiting for the altitude to be low enough to start measuring distance with the ultrasonic sensor, the drone corrects for drifting by heading north and executing the position stabilizing algorithm discussed before.

Once the ultrasonic sensor starts to read valid distances, it changes the throttle to a value which is high enough to slow down the descent to a speed at which if the drone hits the ground, the structural integrity of it is not compromised. Throughout descent, it still continues the execution of the algorithms that correct for possible drifting.

Once the legs of the drone are touching the ground, the ultrasonic sensor will start reading a constant value, which is near to the height at which it is placed at the bottom of the drone. When values around that constant are read, the drone is on the ground, and it sets the throttle value that corresponds to the propellers not spinning, and disarms the motors. This concludes the algorithm of the landing procedure.

3.3.2 Positioning system

Maneuvering the drone autonomously through space to a destination point requires a navigation system. In this project satellite navigation systems are used. These work by utilising satellites and receivers. Radio waves are emitted from satellites down to the receivers, these waves contain information regarding the position of the satellite and accurate atomic clock time when the signal was sent. Through what is called trilateration the position of the receiver can be calculated[34]. Several satellite navigation systems exist such as the American Global Positioning System, Russian GLONASS and the European Galileo system. All systems of this kind are commonly referred to as the Global Navigation Satellite Systems, GNSS for short[36]. According to the European Space Agency, these systems are based on specific implementations of a reference model of Earth called the International Terrestrial Reference Frame[3]. Because these are based on a common framework, data from these different satellite systems can be combined in order to improve

accuracy and connection[29]. Making use of these systems means a destination can be set and data about the current position of the drone can be acquired and used to calculate course.

GPS module Accessing the GNSS systems is done with a GPS module of type BN-880. This module makes use of multiple satellite systems to gather information on position, speed and altitude of the module. It was chosen as according to the data sheet provided by the manufacturer it has an accuracy of 2 meters[44], which is enough for the intended purposes of this project, and it has a built in compass, allowing for travel direction of the system to be read.

The data from the GPS is transmitted via Universal Asynchronous Receiver/-Transmitter (UART), a serial communication protocol. To connect that to an Arduino, it has to be connected to dedicated pins on it, marked by TX (for transmitting data) and RX (for receiving data). Whenever the GPS receives data, it will send it to the microcontroller, which can then be read at will.

NMEA The data output of the GPS is called NMEA sentences, which refers to a data transmission standard called the NMEA 0183 Interface Standard, created by the National Marine Electronics Association[4]. This standard defines the structure of the data which the GPS outputs. An example of such output is shown in Listing 3.1.

```
$GNGGA,100147.944,5528.544,N,00827.685,E,1,12,1.0,0.0,M,0.0,M,,*6D
$GNGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GNRMC,100147.944,A,5528.544,N,00827.685,E,275701.5,268.9,190522,000.0,W*42
$GNGGA,100148.944,5525.843,N,00612.804,E,1,12,1.0,0.0,M,0.0,M,,*6A
$GNGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GNRMC,100148.944,A,5525.843,N,00612.804,E,275701.5,268.9,190522,000.0,W*45
```

Listing 3.1: Example of the output of the GPS

These sentences contain different information, though the one relevant for the navigation is GNRMC. According to the NMEA 0183 Interface Standard sentence description document and information from the company Advanced Navigation Solutions GmbH, the GNRMC sentence contains the following information[5][20]:

TALKER ID, SENTENCE ID, UTC time, Status, Latitude, Hemisphere of latitude, Longitude, Hemisphere of longitude, Speed over ground, Course over ground, Date, Magnetic variation, Mode indicator. TALKER ID refers to the used system, in this case GNSS, and SENTENCE ID to the type of data, here RMC being a shorthand for Recommended minimum specific GNSS data.

Compass The compass built into the GPS module is HMC5883l. Like the barometer, it uses the I2C communication protocol to interface with other devices, in this case the autopilot microcontroller. The necessary data and clock lines are connected to the same pins as the barometer, and the implementation details of it are handled by the Adafruit library used in the code base of this prototype[1]. It also provides convenience functions for simple access to the data sent by the compass. It can be used for three axis measurements, but in this case, only two were used to identify the direction of magnetic north of the Earth.

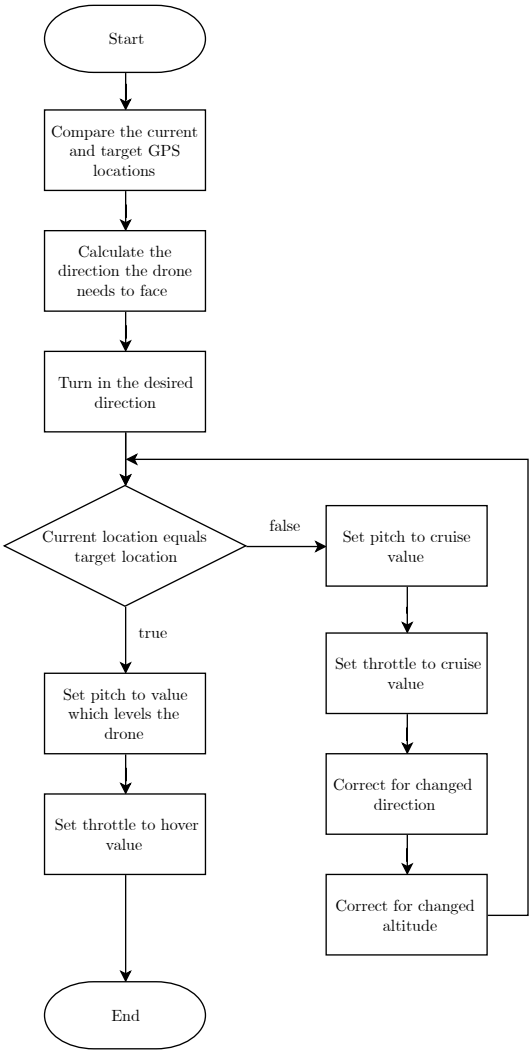

Figure 3.9: Flowchart describing the navigation procedure

Behaviour of system When the control flow of the autopilot is done with take off, the next stage is the navigation part, where the drone flies from the initial location to the destination location. The flowchart of the procedure is shown in Figure 3.9.

The first step is to use the current GPS coordinates and compare them with the destination GPS coordinates, and calculate the angle in which the drone should be headed in order to face the destination. When longitude and latitude have been extracted, this data is passed to the algorithm which calculates which direction the drone has to turn towards. To calculate the angle between the current position and destination position relative to north as seen in Figure 3.10, the equation shown in Equation (3.3) is used.

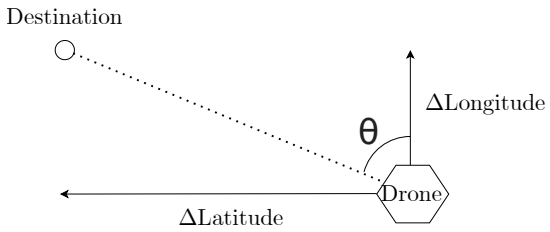

Figure 3.10: Direction difference between drone and destination position

$$\theta_h = \text{atan2}\left(\frac{\Delta Latitude}{\Delta Longitude}\right). \tag{3.3}$$

Where θ_h is the heading angle from true north. Atan2 is a built in function in C++ which acts the same as the inverse tangent, but it is defined for all angles except 0. It is a piecewise-defined function shown in Equation (3.4)[49]:

$$atan2(x,y) = \begin{cases} arctan(y/x) & \text{if } x > 0 \\ arctan(y/x) + \pi & \text{if } x < 0, y >= 0 \\ \pi/2 & \text{if } x = 0, y > 0 \\ -\pi/2 & \text{if } x = 0, y < 0 \\ undefined & \text{if } x = 0, y = 0 \end{cases}. \tag{3.4}$$

Given this angle to turn relative to north, data from the built-in compass can be read in order to compare it with the angle of the drone to see if and how far the drone must turn. The data from the compass will be an angle compared to magnetic north while the calculated angle is with regard to true north, meaning this needs to be accounted for in the calculation. The difference between magnetic north and true north is called magnetic declination, and according to a calculator

made available by the National Oceanic and Atmospheric Administration, the magnetic declination in Esbjerg, Denmark is around 3 degrees[40]. The declination is positive east of true north, thus the turning requirement is given by Equation (3.5).

$$\theta_t = \theta_h - (\theta_c + \theta_d), \tag{3.5}$$

Where θ_t is the turning angle, θ_c is the compass value and θ_d is the declination. The turning instructions are then sent to the system to turn until θ_t is sufficiently close to zero, which is explained further in Section 3.3.2. After the drone has turned in the desired direction, the cruising stage of navigation commences.

For that, the drone first needs to update its current location all throughout the duration of the flight. Since the GPS data is sent by the GPS module to the microcontroller through a serial communication line, it is buffered automatically by the Arduino. This reading takes place in each iteration of the loops contained within all the flight procedures. The GPS data is read, checked if it is the correct NMEA sentence, and is parsed and written into a location from where it can be reached by every part of the code where it is necessary.

To verify if the drone has reached the destination, this updated GPS location is checked against the GPS location of the landing zone. These values can be compared exactly, since the coordinates sent by the GPS module have a precision of up to 2 meters, meaning that even if the drone is 2 meters apart from the destination, it could still result in the same GPS coordinates. This is in contrast to the way other sensors used in this prototype work, given that their sensitivity and precision makes it so that two values cannot be reliable compared, as very small differences in physical quantities result in different values being read from them.

If the destination is not yet reached, the pitch and throttle are set to values which correspond to a forward motion where elevation is maintained. This motion is named cruise. Still, wind is a constant concern, especially with long distances, thus the direction and the altitude are corrected throughout the trip to keep the target course and altitude.

Once the destination location is reached, the drone resets from cruise to hover mode by setting the pitch and throttle to values where it is horizontal, and relatively stable in both position and altitude. This concludes the navigation algorithm.

Maintaining direction and altitude As seen in Section 3.3.2, a critical aspect of navigation is the solution to the changes in direction and altitude throughout the duration of cruising. To do this, two additional algorithms need to be introduced and explained. The compass and the barometer the system uses are both sensors

that need extra handling. Values read by them are too sensitive and their precision too high for the code to simply check equality between them and the desired values.

For both direction and altitude, ranges are introduced compared to which values of the sensors are evaluated, as seen in Figure 3.12 and Figure 3.11. Whenever the current and the target values are compared, the system considers them equal when their absolute difference lies under a set limit called the sensor error boundary. For example, if the target altitude is 50 meters relative to the home altitude, the sensor error boundary could be set to 2 meters, meaning that if the barometer reads a value between 48 and 52 meters, it is considered to be sufficiently close to the target altitude to be considered equal.

The next range is called the correction boundary. If the measurements of the sensor and the target values have an absolute difference greater than the error boundary, they are not considered to be equal, but to prevent constant direction and altitude correction (which might take a considerable amount of time), the program does not yet start doing that within this range.

At last, anything beyond the correction boundary is past the acceptable limits, and the autopilot has to direct the drone back on its proper course. This will result in the drone having its direction or altitude realigned so that the sensor readings and the target values are again within the sensor error range, when they are considered to be equal.

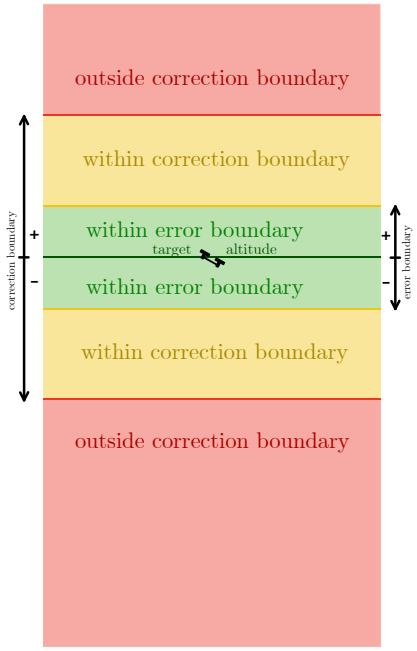

Figure 3.11: Target altitude and the ranges in which measurements in the proximity of it are categorized

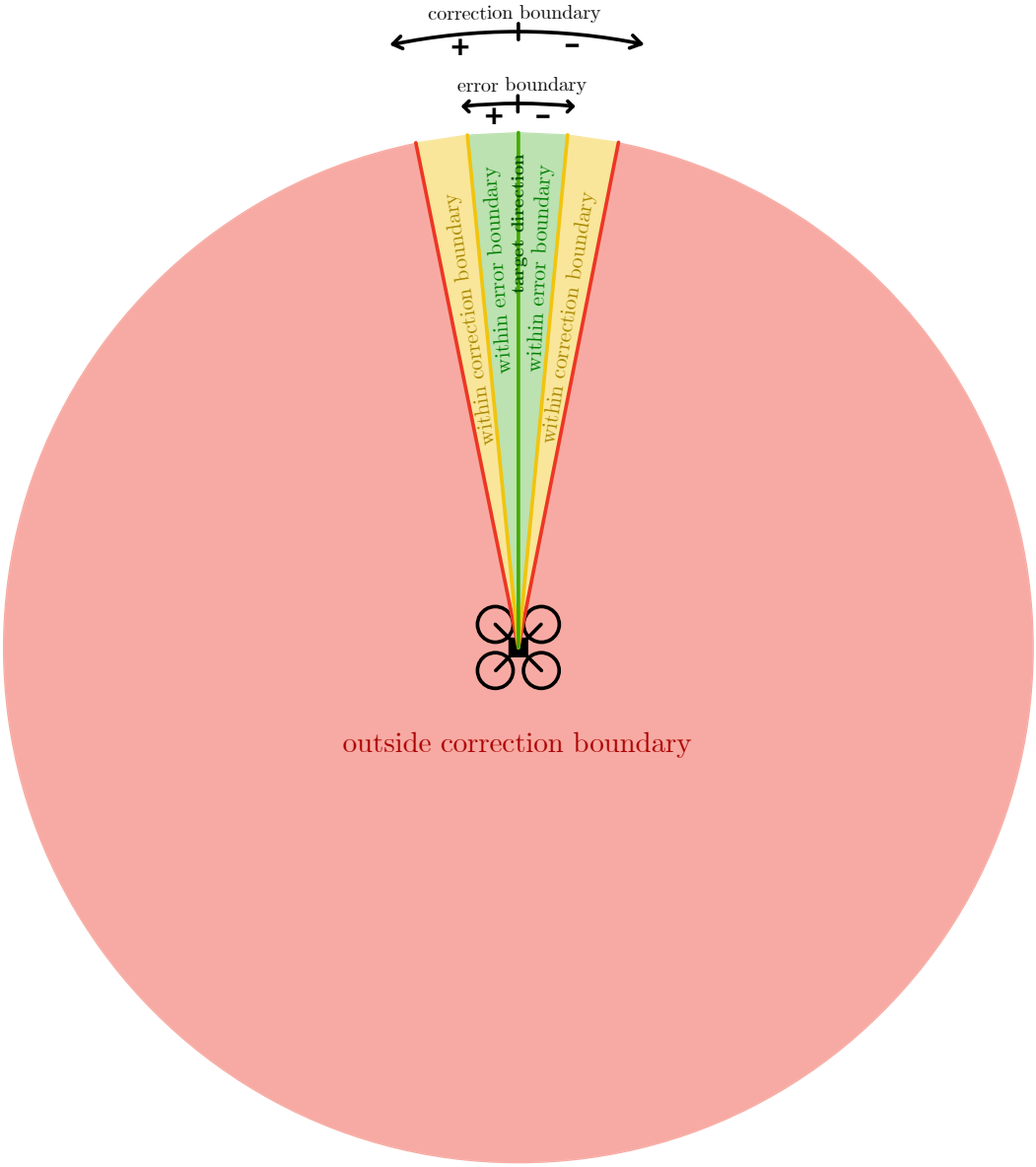

Figure 3.12: Target direction and the ranges in which measurements are categorized

After understanding how and why the sensor value boundaries have been defined, the algorithms for correcting their respective physical quantities can be introduced.

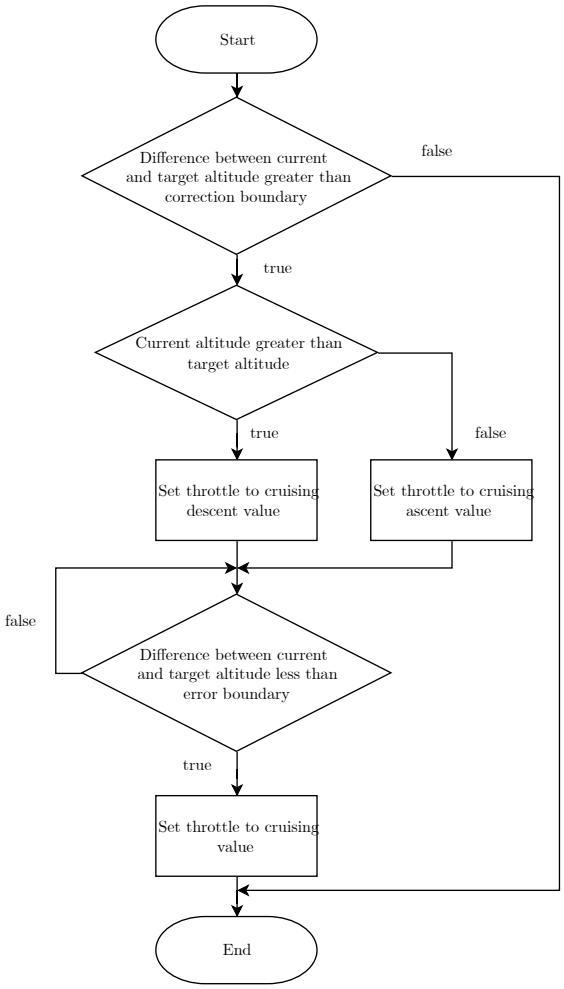

Figure 3.13: The flowchart of the altitude correction algorithm

The flowchart describing how altitude correction works can be seen in Figure 3.13. To keep the drone at or around the target altitude, the drone first has to check the difference between it and the current altitude. If the drone lies outside the acceptable range of measurements, it performs the adjustment. Another check is used to decide if the drone needs to ascend to reach the target, or if it needs to descend.

To make the drone gain or lose altitude, only the throttle value has to be changed, while pitch does not, resulting in continual forward motion throughout the process. Increasing the throttle by a relatively small amount will make the drone gain altitude, while decreasing it will make it lose altitude. These throttle values are called cruising ascent and cruising descent throttle respectively.

After adjusting the throttle value, the drone will keep the value constant as it is getting closer to the target. Once it is within the error boundary, the altitudes are considered to be matching, which means the drone is sufficiently close to the target altitude. After that, throttle is reset to the original cruise value. This concludes the altitude correction algorithm.

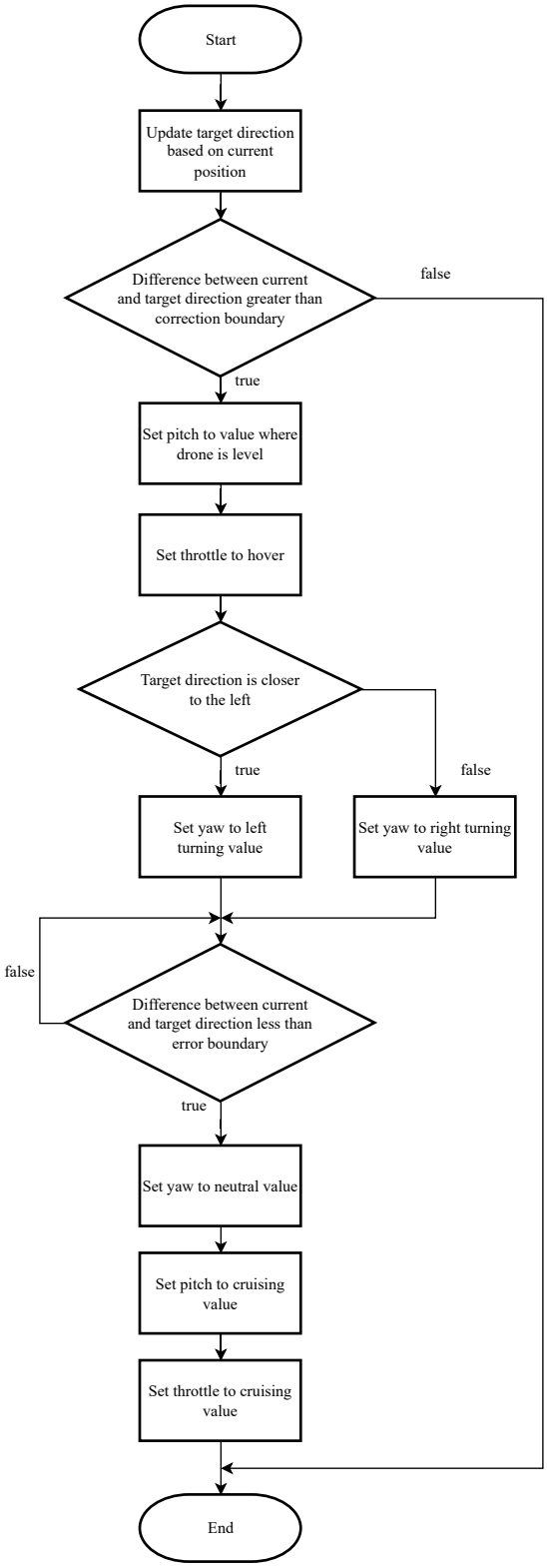

Figure 3.14: The flowchart of the direction correction algorithm

The flowchart describing how direction correction works can bee seen in Figure 3.14. To make the drone correct for the difference between the target and the actual direction, while also keeping it moving forward at the same time, multiple motions on different axes would need to be combined. For this proof of concept, a simplified version was implemented. In this case the drones forward motion is stopped first, then a single axis motion, yaw, is used to correct the direction, after which forward motion can be resumed.

The first step of the process is to recalculate the target direction based on the new position of the drone, then to check if the target and actual directions are in the range where correction needs to take place. When it does, the drone is stopped by setting the pitch to the level where it is horizontal, then the throttle is set to the value where it hovers at the altitude it is currently at. Then the program decides if the drone should be turning left or right, depending on which direction the target is closer to the actual value.

After the drone starts to yaw in the proper direction, the compass is continuously read in order to compare the actual direction value to the target. Once the actual value is within the error boundary, the drone is deemed to be headed in a direction that is sufficiently close to the target. After that, the yaw motion is stopped, so that the drone remains facing in that direction.

The drone can then resume cruising, which is done by setting the pitch and throttle to cruising values. This concludes the direction correction algorithm.

3.3.3 Interaction with flight controller

To command the drone at least five input values are required, four for steering ($pitch$, $roll$, $throttle$, yaw) and one for arming ($aux1$). When the flight controller receives these values, it considers them as channels, one channel is one input. Channels can have values of integers from 1000 to 2000.

A simple way of communicating channel values is via PWM (Pulse-Width Modulation) pins. PWM is a protocol which allows sending and receiving values by modulating the width of a sent pulse. It operates at a switching frequency which is selected depending on application. In information technology it is usually in tens or hundreds of thousands of kilohertz. Based on the width of the modulated pulse, a term duty cycle is introduced. It is the ratio between the time the pulse is in high state and the switching period[23].

In drones, one PWM pin can only carry values for one channel. Consequently abundant wiring is required. To tackle this drawback of PWM communication, PPM protocol can be used (Pulse-Position Modulation).

PPM encodes multiple PWM channels and sends the channel values as one packet. The duty cycles of PWM signals are encoded into durations. The higher the duty cycle, the longer the time frame marked by pulses in a PPM packet is.

To form a packet all PWM values are ordered one after another, see Figure 3.15. PPM sends short pulses every time at the end of the last PWM value and at the start of next PWM value. It encodes the length of a PWM signal into a PPM packet by sending the rising edge of a short pulse to mark the beginning of the PWM signal and another short pulse for marking the end of the signal. Each pulse has static length, it is also called *pausetime*[33].

The cycle length of the whole packet is also predefined, so at the end of the packet there is added *synctime* that fills the gap until next cycle, it makes each cycle last equal time.

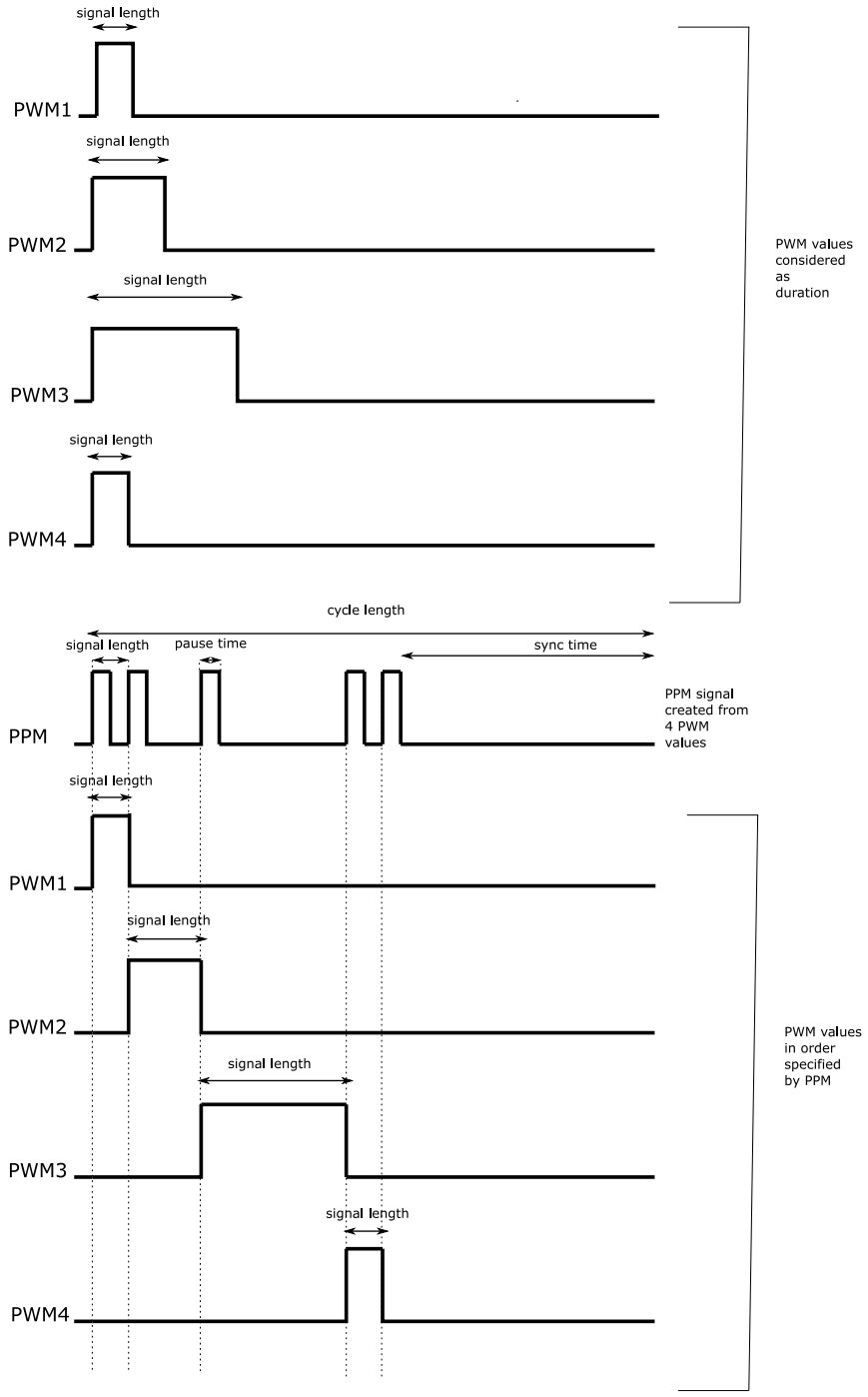

Figure 3.15: Construction of a PPM signal. The first four lines of PWM show one cycle of the respective PWMs. The PPM puts the pulse length of the PWMs after each other—the bottom four lines of PWM show how the PWM values are encoded into the PPM signal. Note that the cycle length of the PPM signal takes more time than the cycle time of PWM (it has to be at least $n_{\text{channels}} \cdot t_{\text{PWM cycle}}$).

In order to send PPM packets with Arduino, a timer with interrupts was set up. Interrupts are crucial because of the need for accurate timing. Comparison mode is used to specify the different time lengths each channel value needs in the PPM signal[46].

3.4 System monitoring

The operator monitoring the drones may need more information than the video feed can provide, such as battery level, a compass, current heading, GPS data etc. to ensure the drone is operating as it should. A way of doing this is to use an OSD. An OSD overlays information on a video feed from a camera. Using an OSD in combination with a microcontroller to read the data and send it to the OSD is a convenient way to solve this problem, since it can provide the operator a visual representation of the data needed.

3.4.1 Overview

This section serves to provide an explanation of how data from the navigation is sent to the OSD and how overlaying that data on a video feed can be done. The focus will be on some of the internal structure of the OSD chip and what abstractions are made to make it easier to work with the OSD.

Figure 3.16 shows how the hardware is connected. The video feed from the camera goes through the MAX7456, which is the OSD chip on the Minim OSD board and will be referred to as OSD from now on, where it overlays information on the video feed using commands send from the ATmega328p, which is the microcontroller on the Minim OSD board and will be referred to as the microcontroller from now on. This modified video feed is then sent to a radio transmitter which send the video to a receiver connected to some type of display.

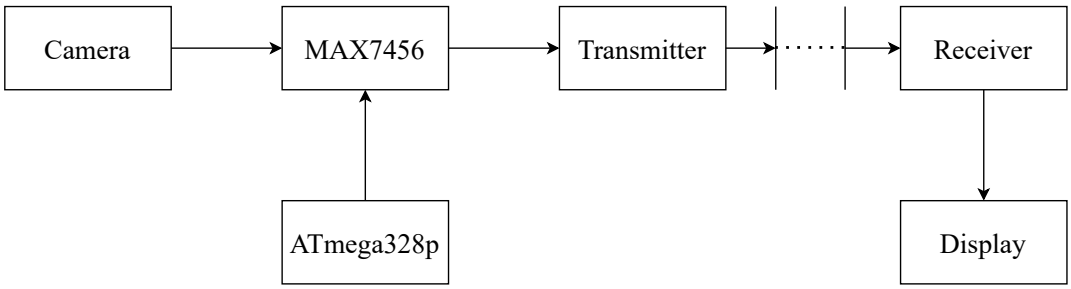

Figure 3.16: System diagram of OSD setup

Figure 3.17 gives an overview about the different parts of the software.

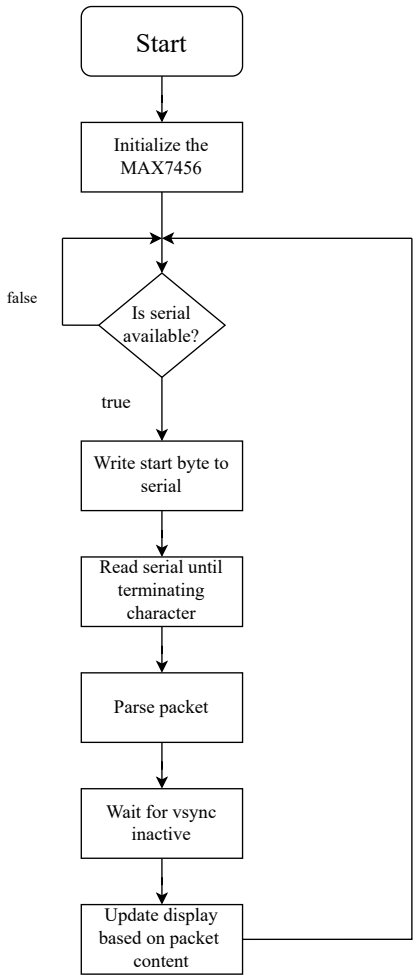

Figure 3.17: Behaviour flowchart for the system monitoring

3.4.2 Communication

The microcontroller is responsible for sending data to the MAX7456 using Serial Peripheral Interface (SPI). SPI is a synchronous way of communicating between a microcontroller and peripherals. There are four pins to be aware of when working with SPI – clock (CLK), Master Out Slave In (MOSI), Master In Slave Out (MISO) and Slave Select (SS). The clock is used to signal both the master and the slave when to sample the data line. The MISO and MOSI pins make up the data line and enable synchronous two-way communication, since there are separate input and output lines. The SS pin is to enable the communication between a specific peripheral and the microcontroller, and is usually active low[7].

To communicate with the OSD, the microcontroller sends a low signal to the

SS pin on the OSD to select the chip, then sends the address of the register that is to be read or written to. This causes the OSD to select that register. Then the data to put in that register is sent. This is then stored in the selected register, and finally a high signal is sent to the SS pin to end the communication. The OSD only sends data back when accessing a read register. Listing 3.2 shows the three convenience functions used to do this: these functions allow for a higher level way of writing the code, thus making the code more readable.

```
void enableChangingOsdRegister () {
    digitalWrite(MAX7456_SELECT, LOW);
}

void disableChangingOsdRegister () {
    digitalWrite(MAX7456_SELECT, HIGH);
}

void changeOsdRegister (byte address, byte value) {
    Spi.transfer(address);
    Spi.transfer(value);
}
```

Listing 3.2: Convenience functions used to communicate with the OSD

3.4.3 Setting up the OSD

Before using the OSD it has to be set up properly, this means setting the video mode to fit with the type of video that the camera produces. The microcontroller configures the pins used for Vertical Sync (Vsync), described in Section 3.4.7) and SS, then it configures Video Mode 0 Register (VM0), see the datasheet for what each bit exactly does[26]. This register enables the video buffer, enables Vsync, enables the OSD image and sets the video standard to PAL, see Section 3.4.5. The setup process is done by the constructor of OsdHandler, see Listing 3.3.

```
OsdHandler::OsdHandler () {
    pinMode(MAX7456_SELECT, OUTPUT);
    pinMode(MAX7456_VSYNC, INPUT);
    digitalWrite(MAX7456_VSYNC, HIGH);

    enableChangingOsdRegister();
    changeOsdRegister(MAX7456_VM0_reg, 0b01001100);
    disableChangingOsdRegister();
}
```

Listing 3.3: Constructor of the OsdHandler class

3.4.4 Character memory and font upload

Before writing characters on the display a font must be uploaded to the OSD. This sections explains the process of uploading characters into the character memory. The font used is from a project called BitsyOSD[19]. The font is loaded into the Non-Volatile Memory (NVM) of the OSD, meaning that it persists even after power is removed. It is stored in a 16x16 grid of characters, where each character is 12x18 pixels. The pixels can be either black, white or transparent and are represented by 2 bits, meaning that each byte is a block of 4 pixels. The way to access the NVM is always through shadow RAM, which is RAM that is hidden from the user. To write a new character into the NVM all the pixel data has to be loaded into the shadow RAM one byte at a time then it can be written to the NVM.

To write the character into memory first the OSD image is disabled, then the address of the character to be changed is written into the Character Memory Address High Register (CMAH). Using the Character Memory Address Low Register (CMAL), the user can pick which block of 4 pixels to write to, then using the Character Memory Data In Register (CMDI) the user can set the pixel data for the 4 pixel block picked with CMAL. This is repeated until all pixels have been written. To put this data into the NVM the Character Memory Mode Register (CMM) is used, by configuring it to write to the NVM.

CMDI [7,6]	CMDI [5,4]	CMDI [3,2]	CMDI [1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]
[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]	[7,6]	[5,4]	[3,2]	[1,0]

Figure 3.18: Visual representation of what a character looks like in memory Based on figure 11 in the datasheet for the MAX7456[26]

3.4.5 OSD display

The OSD overlays the the extra information onto the video stream. The OSD can do this using PAL and NTSC. PAL has a bigger available area vertically—288 lines, opposed to the 234 lines of NTSC—, so it has been used for the first prototype, and since it worked without any problem, this decision did not have to be overwritten. Using PAL, the available display area has 288 lines and is 360 pixels wide. The position of this display area on the video feed can be adjusted using the HOS and the VOS registers, but this was not required in this project—see Section 3.5.2.1. The display area is divided into 16 rows and 30 columns of panels. Each panel can contain the graphical image of a single character.

In order to display a character, the first thing to do is to set the panel index of the OSD cursor according to the desired destination of the character. The algorithm used for this is shown in Listing 3.4. An important note on the algorithm is the usage of auto-increment. By turning it on, the panel index is incremented after each character write. In practice, this means that to write a word onto the screen, one only has to set the panel index in the beginning when using auto-increment.

```
// x and y denote the destination of the character
// The coordinates have to be converted to the index of the panel
unsigned int panelIndex = y * 30 + x;
byte panelIndexHigh = panelIndex >> 8;
byte panelIndexLow = panelIndex;

enableChangingOsdRegister();
// enable auto increment
changeOsdRegister(MAX7456_DMM_reg, 0b1);
changeOsdRegister(MAX7456_DMAH_reg, panelIndexHigh);
changeOsdRegister(MAX7456_DMAL_reg, panelIndexLow);
```

Listing 3.4: Snippet of the openPanel function

After setting the panel index, the character has to be written onto the screen. This is done using `changeOsdRegister(MAX7456_DMDI_reg, c)`, where `c` is the code of the character. Note that this code does not necessarily correspond to the ASCII code of a certain character - see Section 3.4.6.

For example, to display "1" (without quotes) at the current panel index, `changeOsdRegister(MAX7456_DMDI_reg, 0x02)` has to be called (the number of the character depends on the current data held by the NVM – see Section 3.4.4). To display "11" (without quotes), the mentioned command has to be called twice – the two ones are going to be written next to each other because of the auto-increment behavior.

However, to manually move the panel index cursor again, auto-increment has

to be disabled first, otherwise the MAX7456 will keep using the internal panel index generated because of auto-increment[26]. `changeOsdRegister(MAX7456_DMDI_reg , MAX7456_END_string)` is used to disable auto-increment.

So the high level algorithm of writing a word onto the screen is:

1. Go to the specified position and enable auto-increment (done using the `openPanel` function)
2. Output the first character, then the next character, etc.(done using the `writeChar` function)
3. Disable auto-increment (done using the `closePanel` function)

3.4.6 Mapping character codes

For convenience, the writeString function has been created. It takes a String (Arduino data type) and the x and y coordinates of the first character. The String is created using `String("message")`, where the characters can only be a limited section of the ASCII characters, like alphanumeric characters or the space character. Note that the String object will contain the ASCII codes of the desired characters.

In order to display the given String, the algorithm described in Section 3.4.5 must be followed. However, calling writeChar for the characters is not trivial, since the ASCII code of the character does not necessarily correspond to the desired graphical image in the character memory. For example, the ASCII code of "1" is 49, while in the character set used by this project the code for the graphical image of "1" is 2. This means that 49 has to be mapped to 2 – so that using `writeString(2, 3, String("1"))` will in turn call `writeChar(2)`, which will display the graphical image of "1" at (2;3).

The same type of mapping had to be made for all numbers, letters, the space character and the full stop character.

However, there are also graphical images in the character set which are crucial to display but are not present in ASCII. An example of this are the 16 arrows pointing to 16 different directions, which were used to present a compass as well as the heading direction.

In order to display those, constants are defined which hold values which correspond to ASCII characters which are not supported otherwise by the existing program. For example, the ASCII codes 1 to 16 are used for the arrows. A visual representation of the concept can be seen on Figure 3.19.

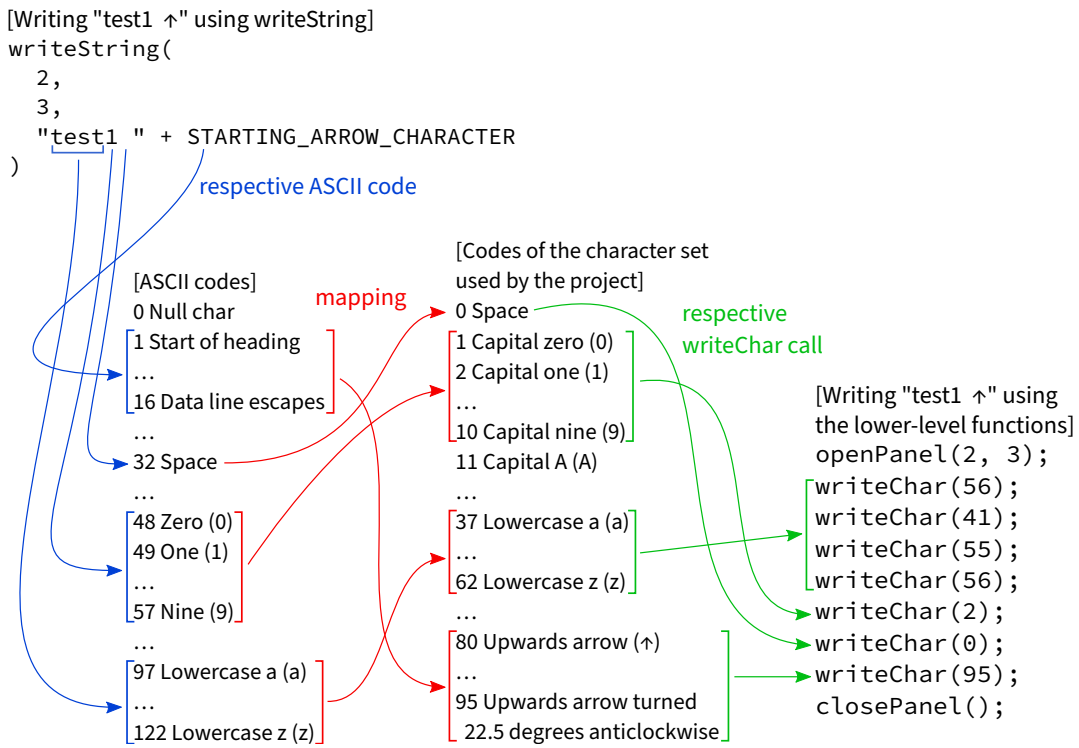

Figure 3.19: A visual representation of how writeString functions

3.4.7 Avoiding screen tearing and text blinking

When writing to the display it is important to be aware of when the characters are being written to the display memory, if not done carefully it can result in text blinking on the screen, also known as screen tearing. In order to prevent screen tearing, the text should only be updated between two refresh cycles (the vertical blanking interval). This is achieved by using Vsync. What Vsync does is to signal to the microcontroller to wait until the display is done updating the image on the screen. When this is done, the microcontroller can freely write the new image to the display memory. However, momentary break up of text (blinking) can still occur if the display memory is updated during a refresh cycle[26]. In order to prevent this from happening, the display memory writes are delayed until Vsync becomes inactive, see Listing 3.5.

```
1 void OsdHandler::waitForVsyncInactive () {
2     enableChangingOsdRegister();
3     while (Spi.transfer(MAX7456_STAT_reg_read) &
      MAX7456_STAT_reg_read_VSYNC_INACTIVE);
4     disableChangingOsdRegister();
5 }
```

Listing 3.5: The waitForVsyncInactive function

Thus the whole updateDisplay function works as follows:

1. Wait for refresh cycle to end if it's ongoing
2. Write data to the display memory

In order to reduce text blinking even more reliably, the panels are only cleared when they are being overwritten. This means that if the text at a given position is ABCD and one writes BCD to that position in the next updateDisplay call, then the resulting text will be BCDD. In order to always remove the obsolete data, a space padding gets added to the text – in the mentioned example, "BCD" (without quotes) will become "BCD " (without quotes, and note the additional space), so that the resulting text on the screen is BCD instead of BCDD.

The size of the padding depends on the maximum possible length of the text – for example, when writing a number between 1 and 999 onto the screen, the extra space padding will be $3 - \textit{the number of digits}$.

3.4.8 Receiving the data which has to be displayed

The ATmega328p chip in the Minim OSD receives the new data to display through serial communication with the Arduino Mega. When the ATmega328p chip is ready to process new data, it sends a 0x01 byte. In response, the Arduino Mega sends a packet with integers and doubles separated by spaces, and the whole packet ends with a slash character. Thus a packet looks like this: `a b c d e f g h i j/`, where the letters get replaced by the actual data points. The data points are in a predetermined order, and they get mapped to the corresponding OsdData properties, which will be displayed during the next updateDisplay.

3.5 Testing

Testing the prototype is a crucial step in validating the success of the project. 12 tests have been devised to validate important parts of the prototype.

1. System monitoring
   (a) Adjusting the OSD's display area
   (b) Padding the text with spaces
   (c) Mapping angles to the respective arrows
   (d) Receiving data from navigation

2. Autopilot
    (a) Barometer
    (b) Ultrasonic sensor
    (c) GPS output
    (d) Compass
    (e) Take off algorithm
    (f) Navigation algorithm
    (g) Landing algorithm
    (h) Flight controller

3.5.1 Autopilot

3.5.1.1 Barometer

Hypothesis It is expected that the barometer will give relative measurements of height within the error range specified on its datasheet($\pm$0.17 meters), compared to the height where it was set.

Method The barometer was set at ground level at the 5th floor of a 12 floor building at Mølleparkvej 1, Esbjerg N. It was first carried up to the 12th floor, then down to the entrance of the building.

Results The barometer readings between the 3 measurements provided a difference of 2.2-3.2 meters between floors, roughly corresponding to the height of one floor in such buildings. Although the results were outside expected error range, the error was still inside a usable boundary. The fact that the results are outside of the error boundary does not necessarily mean that the sensor was faulty, but that pressure depends on other factors than altitude and temperature, for example wind speed.

3.5.1.2 Ultrasonic sensor

Hypothesis It is expected that the ultrasonic sensor measurements will result in distances which have no more than 5 cm error compared to actual distances, and will not return measurements above 400 cm.

Method The ultrasonic sensor was placed perpendicularly to an adjacent wall at less than 4 meter distance. Then the ultrasonic sensor was placed outside the window, pointed towards the sky.

Result Performing multiple measurements showed that the ultrasonic sensor was indeed measuring distances within error boundaries. Once the ultrasonic sensor was pointed towards the sky, it gave -1 as the distance, which was the value used to denote invalid values.

3.5.1.3 GPS output and parsing

Hypothesis The GPS should be able to get a fix on enough satellites in order to output NMEA sentences with the current position of the GPS. That position should be within 2 meters of the actual location of the GPS. The software should be able to specifically parse out the GNRMC sentences and output this to the serial monitor.

Method The GPS was held outside the window of the university and output was read via the serial monitor.

Results The GPS got a satellite fix and the software was able to parse out GNRMC sentences. The GPS output was consistent, with only negligible (under 2m) difference between results. An example of such sentence is shown in Listing 3.6.

```
1 $GNRMC,155821.00,A,5529.49606,N,00826.76122,E,1.880,,230522,,,A
```

Listing 3.6: GPS output while the GPS is being held out of the university window

This was compared to the GPS coordinates found of the current location on Google Maps, which showed the coordinates 55,4915790N and 8,4459516E. The Haversine formula has been used to calculate the distance. The formula is shown in Equations (3.6) to (3.8).

$$a = \sin^2\left(\phi B - \phi\frac{A}{2}\right) + \cos\phi A \cos\phi B \cdot \sin^2\left(\lambda B - \lambda\frac{A}{2}\right), \tag{3.6}$$

$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a}), \tag{3.7}$$

$$d = Rc, \tag{3.8}$$

where ϕ is latitude, λ is longitude, R is the radius of the Earth, and d is the distance, note that angles are in radians[28]. It showed that the distance between the two coordinates was 24.59km. Although this calculation is based in a spherical earth and inherently has some inaccuracy, observing the coordinates on a map showed that the GPS output gave a coordinate far off the coast of Fanø, meaning that the GPS is inaccurate. The large disparity in distance could be from several factors,

including a small amount of satellites in view or scratches on the antenna of the GPS. Since multiple algorithms depend on the readings taken from the GPS to find the position of the drone, those are not able to function properly with the GPS providing data as erroneous as the tests showed. Unless a more precise GPS is installed, the autopilot will not be able to navigate the drone as intended.

3.5.1.4 Compass

Hypothesis The compass should be able to measure 0° angle when facing towards north, and should go up until 359° in one full rotation until it reaches north and 0° again.

Method The compass was set to a position where measurements resulted in 0°, and one full rotation was performed.

Result 0° was not completely aligned with the north facing direction. During the rotation, angle measurements were steadily increasing until it reached 359° not precisely at the end of the rotation, at times by less than one full rotation, at other times by more than one full rotation. After these rotations, shifts in the direction corresponding to 0° were observed in both directions. The test showed that the compass could not be used for precise measurements.

3.5.1.5 Take off

Hypothesis The take off procedure should set the throttle to the ascending value defined as a constant in the code until barometer measurement exceeds the target altitude also defined as a constant in the code. During this time, it should continually make sure to point the drone towards the direction where the compass measures 0°. If GPS data is not the same as it was recorded at the beginning of the procedure, it needs to set pitch and roll values to move in the desired directions.

Method For the purpose of this test the prototype was manually lifted up with the throttle, pitch, roll and yaw values continuously monitored and verified if they match expected values based on sensor measurements. Target altitude was changed to a low value so that lifting it around 1 meters was enough to exceed target altitude.

Result GPS coordinate measurements were not performed, as GPS measurements were deemed too unprecise for the autopilot. The MCU indeed set throttle value to ascending, and proper yaw values to rotate either towards left or right depending on which direction the 0° measurement was closer. By lifting the sensors, target

altitude was reached, at which point the algorithm was finished. This showed that the take off procedure worked as intended compared to the readings on barometer and compass, while the drift avoidance test was not performed.

This test verified that the autopilot code will set throttle and yaw values to the predefined constants in the code as expected based on sensor readings.

3.5.1.6 Navigation

Hypothesis Navigation procedure should first set yaw values correctly to rotate toward the destination, then stop the rotation. Then, it should set pitch and throttle values corresponding to cruise mode. Then it continuously has to change throttle values if the barometer readings are outside of the correction boundary and reset once they are back within error boundary. It should also adjust the direction if the compass reads values outside of the correction boundary, by first changing pitch and throttle to the constants defined for hover mode, then setting yaw values to start rotating in the desired location. After the direction readings are within the error boundary, it resets to neutral yaw, and cruise pitch and throttle. Once the target destination is reached, the navigation process should finish.

Method The array of sensors is lifted up and down around the target altitude which is set to a custom value used for testing. It is also rotated from the target direction, while the throttle, pitch, roll and yaw values are monitored and verified if they match expected values based on sensor measurements. The prototype is moved in the direction of an arbitrarily set destination location to validate that the navigation finishes once it reaches it.

Result After the initial direction was set, throttle and pitch were set to desired values. When the barometer was moved down enough to be outside the correction range defined in the code, it changed throttle values until the barometer measurements were within error range. Then, when the compass was rotated, the system was set to hover mode, with corresponding pitch and throttle values beings set, and was not reset to cruise until compass readings matched the desired direction. Reaching the destination coordinates was not tested, thus finishing the algorithm was not validated.

With this test, it was shown that navigation will start moving the drone in the desired direction depending on compass measurements, and it will correct for both altitude and direction changes during flight based on barometer and compass measurements.

3.5.1.7 Landing

Hypothesis Landing procedure should set throttle to fast descent value while the ultrasonic sensor is out of range, and then set it to slow descent value once it is in range. Once the ultrasonic measurement returns the value of the height at which the sensor is placed at the bottom of the drone, it should set throttle to zero and disarm to disarmed value. Throughout this process, it should continuously check if the compass readings are outside the correction boundary, and rotate the drone to face north by setting yaw values to left or right rotation depending on which direction is north closer to, based on compass measurement, then resetting the yaw to neutral once the compass readings are within error boundary. It should also check continuously if the current and landing GPS coordinates are off, and set pitch and roll values to adjust them until it's back above the target location.

Method The ultrasonic sensor is first pointed in a direction where the nearest surface is beyond the 4 meter maximum range, and then pointed towards a closer surface, while the throttle, pitch, roll and yaw values are monitored and verified if they match expected values based on sensor measurements.

Result The proper throttle value was set to start the fast descent when the ultrasonic sensor measured -1, the value denoting out of range distance. Until the compass showed readings sufficiently close to 0°, it set yaw values to proper rotational directions. Once it did, yaw was reset to neutral value. Throttle was also set to slow descent value once the ultrasonic sensor started measuring distances within 4 meters. Since GPS coordinate measurements were deemed to be too unprecise, position correcting could not be verified.

With this test, it was shown that the landing will start decreasing the altitude in the proper speed depending on ultrasonic readings, and it will continuously try to turn the drone towards north depending on compass readings. The drifting avoidance algorithm was not validated.

3.5.1.8 Flight controller

Hypothesis To control the drone data must be sent to the flight controller using PPM. The expected result is that the data sent to the flight controller will be readable on BetaFlight, a program used to configure and monitor flight controllers.

Method Predefined data, 1500 on all channels, is sent to the flight controller, to easily observe whether it is displayed in BetaFlight.

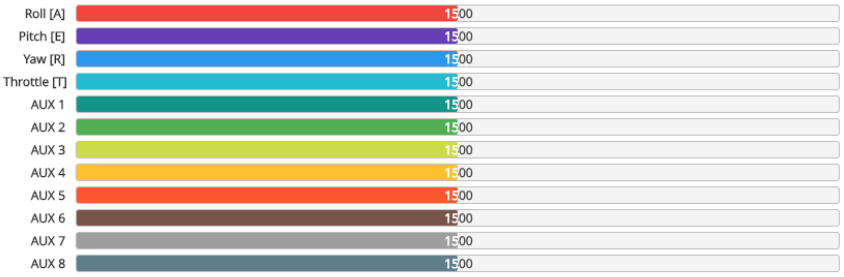

Figure 3.20: Bars in BetaFlight showing the input data

Result The expected data was shown in BetaFlight confirming that the PPM works, seen in Figure 3.20.

3.5.2 System monitoring

3.5.2.1 Adjusting the OSD's display area

Hypothesis The display area does not necessarily align with the video—sometimes extra adjustment is needed so that the OSD's display area covers the whole video. The expectation is that some amount of adjustment needs to be made.

Method To test whether a horizontal or vertical offset was needed, the code has been adjusted so that there is nothing in the loop function, and the setup function contained a writeString call for every row (there are 16 of them), with each of the calls having a String as an argument which is 30 characters long (in this case, all of the lines start with the line index, then a space, then "0123456789" until).

Result The result is shown in Figure 3.21. Although the first row, the first column and the last column is noticeably offscreen, there was a character in every part of the screen—which means the OSD's display area covers the whole video, so no adjustment was needed.

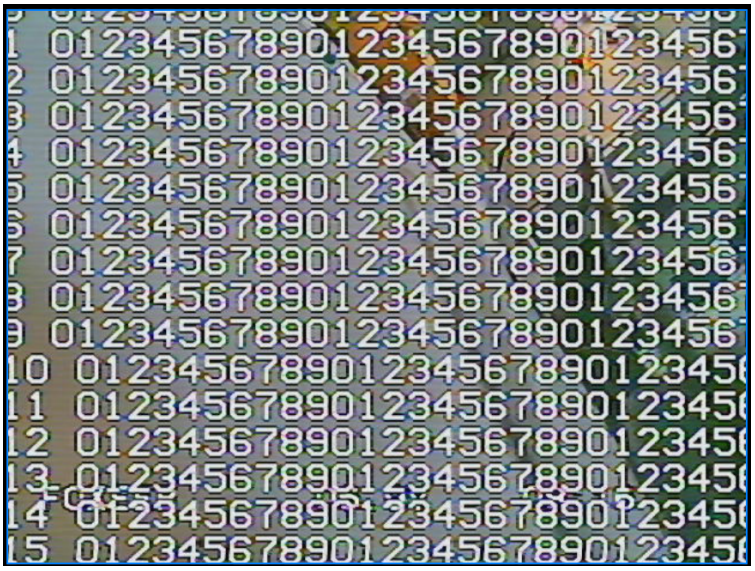

Figure 3.21: Image of video feed from OSD showing characters in every panel

3.5.2.2 Padding the text with spaces

Hypothesis The screen is not cleared to avoid screen tearing, so space padding is used to make sure that obsolete data is not being displayed—as described in Section 3.4.7. The expectation is that left over characters from longer strings will be cleared. An example being going from the 4 character long 1000 to the 3 character long 999. The program should get rid of the last 0 in 1000 and display 999 instead of 9990.

Method The space padding was tested by displaying a countdown from 1050 padded to be 4 characters long, decreasing the value after every updateDisplay call.

Result When the counter went from 1000 to 999, the screen showed 999 instead of 9990, which meant that the padding is indeed working.

3.5.2.3 Mapping angles to the respective arrows

Hypothesis The heading direction and the compass data are in degrees (a number between 0 and 360, the lower bound being inclusive and the upper bound exclusive). These angles have to mapped to the respective arrow, i.e. 0 to the arrow looking upwards, 90 to looking to the right, etc. The expectation is that the arrow will move at even time intervals through the different arrows.

Method A counter was implemented, which counted up one by one up to 359 and then went down to zero in the next iteration. The counter went to its next iteration after every updateDisplay call. The count was sent to the mapArrow function, the result of which was then mapped to the code of the respective arrow, which got displayed after that.

Result The arrow went in a full circle while being in the same position for equal time intervals, which meant that mapping was indeed working as expected.

3.5.2.4 Receiving data from navigation

Hypothesis The Arduino Mega sends the extra textual data for monitoring to the OSD module using serial communication. A packet of data is sent using the protocol described in Section 3.4.8, then it gets overlaid on the video feed by the OSD. The expectation is that all of the sent data gets displayed by the OSD according to the packet specification.

Method A test packet was sent to the OSD where all the data points correspond to different graphical images or image combinations (i.e. the value of the data points are different, and the data points which get mapped to arrows get mapped to arrows pointing to different directions). The function call used for the purpose is shown in Listing 3.7.

```
1 Serial3.write("0 0 1 2 -3 4 90 348 7 8/");
```

Listing 3.7: The code which was used to send the data to the OSD

Result All the data points were correctly displayed, as shown in Figure 3.22, thus serial communication and the handling (encoding and decoding) of packets was working as expected.

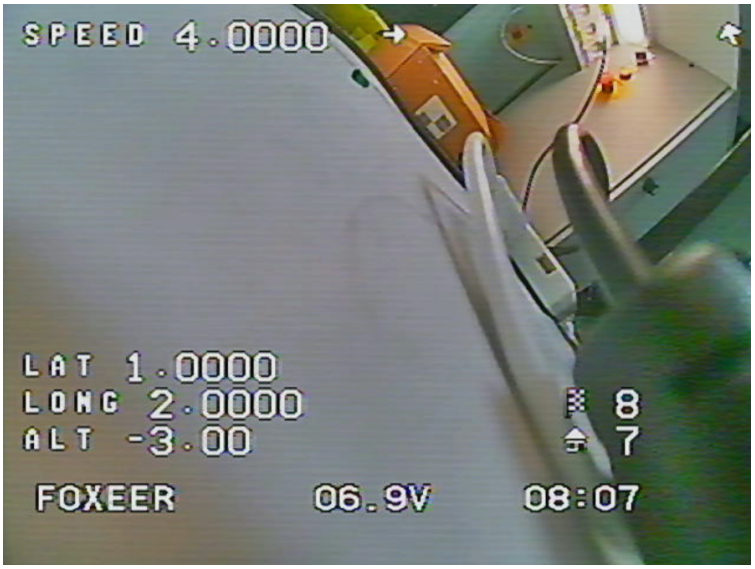

Figure 3.22: The result of displaying the test packet. Note that "FOXEER", "06.9V" and "08:07" are generated by the camera. The rest of the textual output is generated by the OSD.

Chapter 4

Discussion

Throughout the development and testing of the system, changes and improvements were made beyond what was initially specified. Some of these changes were done in order to improve on what was initially planned, and some compromises had to be made due to limitations of time, access to hardware and overall scope of the project.

4.1 Sensors

The sensors used in the project were chosen mainly because of convenience of implementation or their availability. The ultrasonic sensor was available from the Aalborg University laboratory. The accuracy of it was deemed reasonable enough from a prototyping perspective, but might not be suitable if this system were implemented for commercial use. The ultrasonic sensor had quite a low range for the purpose of landing, and even though it was accurate with regard to its output, it would not be reliable in outdoor conditions. The low range was thus compensated for by programming the landing for what would be a very slow descent. This would of course have to be adjusted if the system were implemented in a physical drone. Furthermore, the combined GPS and compass module was used because the navigation subsystem required both, and having them integrated in the same module made it more compact and easy to use. The GPS accuracy of two meters according to the data sheet was deemed enough for the prototype and potentially also for implementation purposes, as landing platforms for helicopters are larger than two meters in radius. However, tests of the GPS showed that the accuracy was far from advertised. In fact, this means that the sensor is highly unsuitable, and would need to be replaced. The compass also suffered from issues with accuracy and would need replacing.

4.2 Software

Software design was initially split up into the take off and landing, navigation and monitoring subsystems, work being done on each part concurrently before integrating them into one program. Throughout the software design process changes were done many times. Both changes in function and in structure. The main microcontroller was initially selected to be an STM32F411. This was chosen as specifications such as clock speed were higher than that of the Arduino Mega. A higher clock speed enables faster processing of data, which may have been a beneficial for sending continuous instructions to the flight controller commanding a physical drone. Moreover, the STM board was significantly smaller, making it easier to fit on a drone. However, due to many issues with the STM32 Cube Integrated Development Environment which was used to set up and program the microcontroller, it was decided that using an Arduino Mega would be sufficient for this system, and was easier to use and interact with. The software was designed using an object oriented programming structure. This was chosen as it is standard for most software development, and was familiar to some group members, enabling them to transfer the knowledge to those who were not familiar with it.

4.3 Communication with drone FC

This part of the program can be considered as the external output of the system, where any drone could be connected. For the purpose of prototyping, there was no need to implement a protocol that is able to control respective channels of the FC, such as throttle, pitch, yaw. Since these commands would have to be adjusted for any particular drone, it could have been considered as a part of drone development, which is out of the scope of this project which focuses on the autopilot program. However, an attempt was made to write a library capable of controlling a drone over PPM protocol. This shifted a part of the resources from more significant parts of development. In the end, this part did not contribute considerably to the working of the autonomous system and for the testing purposes it could have been replaced by printing the commands in serial monitor.

4.4 Testing

The testing successfully validated the most of the modules in the prototype. Testing was designed and planned to allow for each module to be tested independently. This enabled tests to be done in parallel.

Tests were done with an emphasis on the overall function of the sensors, with precise accuracy not being tested. For the ultrasonic sensor and barometer high

precision was not needed for this prototype, and with regards to the GPS and compass they showed in the preliminary tests that they are not consistent enough to merit any further testing.

4.5 Future work

To further research this topic the autopilot described in this report would be placed on a drone to make it autonomously fly to a target location. It is expected that changes would need to be made in order to integrate it on a physical drone. Software would likely need changes to adapt it to circumstances that could only be encountered once the drone is in the air, as for example finding the necessary throttle settings. Other potential changes could be wiring that would need to be more robust, weather shielding of the components, more accurate sensors and likely many other adaptations. Measurements such as energy consumption, range capacity and carrying capacity would need to be tested in order to measure its economic and environmental benefits or drawbacks. Further research into the legislative aspects would be necessary as these are likely some of the largest obstacles to implementing this product.

Chapter 5

Conclusion

The aim of the project was to develop a system designed to be implemented on a drone to enable it to fly autonomously. This was done specifically to research the possibilities of lowering the environmental and economic impacts of offshore spare parts delivery. The tests performed in Section 3.5 were done to test the function of the sensors and subsystems, although implementing them on a drone would require further adjustments in the software. Given that the system was not implemented on a physical drone, experiments to measure the environmental and economic impacts were not done, but the system has shown that work could be continued in this area to answer these questions. Out of the relevant UN Sustainable Development Goals which were analyzed in Chapter 2, it can be concluded that this report has mainly fulfilled goal number

9. Industry, Innovation and Infrastructure,

An important step in furthering this goal has been achieved by the prototype described in this project, yielding a proof of concept that could enable future research into this field in order to contribute to developing solutions that are in line with other sustainability goals.

References

[1] Adafruit. *Bus IO library*. URL: https://github.com/adafruit/Adafruit_BusIO (visited on 05/23/2022).

[2] Danish Energy Agency. *Denmark's Energy Islands*. URL: https://ens.dk/en/our-responsibilities/wind-power/energy-islands/denmarks-energy-islands (visited on 05/18/2022).

[3] European Space Agency(ESA). *Reference Frames in GNSS*. URL: https://gssc.esa.int/navipedia/index.php/Reference_Frames_in_GNSS (visited on 05/25/2022).

[4] National Marine Electronics Association. *NMEA 0183 Interface Standard*. URL: https://www.nmea.org/content/standards/nmea_0183_standard (visited on 05/25/2022).

[5] National Marine Electronics Association. *NMEA 0183 V 4.11 Sentence Descriptions*. URL: https://www.nmea.org/Assets/NMEA\%200183\%20V4.11\%20Sentence\%20Talker\%20Identifiers.pdf (visited on 05/25/2022).

[6] World Bank. "Air Freight: A Market Study with Implications for Landlocked Countries". In: (2009). URL: https://www.worldbank.org/en/topic/transport/publication/air-freight-study (visited on 05/25/2022).

[7] Scott Campbell. *Basics of the SPI Communication Protocol*. URL: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/ (visited on 05/16/2022).

[8] DeltaQuad. *DeltaQuad Pro Cargo specification*. URL: https://www.deltaquad.com/vtol-drones/cargo/ (visited on 02/17/2022).

[9] DJI. *Matrice 300 specification*. URL: https://www.dji.com/dk/matrice-300/specs (visited on 02/17/2022).

[10] DJI. *Matrice 600 Pro specification*. URL: https://www.dji.com/dk/matrice600-pro/info#specs (visited on 02/17/2022).

[11] Gryphon Dynamics. *GD-120X*. URL: http://gryphondynamics.co.kr/product/gd-120x?ckattempt=1 (visited on 02/22/2022).

[12] European Union Aviation Safety Agency (EASA). *Certified Category - Civil Drones*. URL: https://www.easa.europa.eu/domains/civil-drones/drones-regulatory-framework-background/certified-category-civil-drones (visited on 05/25/2022).

[13] European Union Aviation Safety Agency (EASA). *Civil drones (unmanned aircraft)*. URL: https://www.easa.europa.eu/domains/civil-drones (visited on 05/25/2022).

[14] European Union Aviation Safety Agency (EASA). *Open Category - Civil Drones*. URL: https://www.easa.europa.eu/domains/civil-drones/drones-regulatory-framework-background/open-category-civil-drones (visited on 05/25/2022).

[15] European Union Aviation Safety Agency (EASA). *Specific Category - Civil Drones*. URL: https://www.easa.europa.eu/domains/civil-drones/drones-regulatory-framework-background/specific-category-civil-drones (visited on 05/25/2022).

[16] Elecfreaks. *HC-SR04 Datasheet*. URL: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf (visited on 05/16/2022).

[17] Council of European Union. *COMMISSION IMPLEMENTING REGULATION (EU) 2020/639*. 2020. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32020R0639&from=EN (visited on 05/25/2022).

[18] A. S. Fiorillo, C. D. Critello, and S. A. Pullano. *Theory, technology and applications of piezoresistive sensors: A review*. 2018. URL: https://www.sciencedirect.com/science/article/pii/S0924424718308434 (visited on 05/24/2022).

[19] flyandi. *BitsyOSD*. URL: https://github.com/flyandi/BitsyOSD (visited on 05/25/2022).

[20] Advanced Navigation Solutions GmbH. *NMEA Solution Output Format*. URL: https://anavs.com/knowledgebase/nmea-format/ (visited on 05/25/2022).

[21] Tim Gould et al. *Offshore Energy Outlook 2018*. Tech. rep. International Energy Agency, 2018. (Visited on 05/25/2022).

[22] Max Roser Hannah Ritchie and Pablo Rosado. "CO and Greenhouse Gas Emissions". In: *Our World in Data* (2020). https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions. (Visited on 05/25/2022).

[23] D. Ibrahim. *Advanced PIC32 Projects*. 2014. URL: https://www.sciencedirect.com/topics/engineering/pulse-width-modulation (visited on 05/25/2022).

[24] IEA. *Offshore Wind Outlook 2019*. URL: https://www.iea.org/reports/offshore-wind-outlook-2019 (visited on 05/23/2022).

[25] Fortune Business Insights. *Offshore Inspection, Repair, and Maintenance Market Size, Share COVID-19 Impact Analysis, By Service Type*. URL: https://www.fortunebusinessinsights.com/industry-reports/offshore-inspection-repair-maintenance-market-100405 (visited on 05/23/2022).

[26] Maxim Integrated. *MAX7456 Datasheet*. URL: https://datasheets.maximintegrated.com/en/ds/MAX7456.pdf (visited on 05/16/2022).

[27] Data M Intelligence. *Offshore Helicopters Market Share, Insight, Report, Trends, Analysis, Forecast*. URL: https://www.datamintelligence.com/research-report/offshore-helicopters-market (visited on 05/21/2022).

[28] Simon Kettle. *Distance on a sphere: The Haversine Formula*. URL: https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128 (visited on 05/23/2022).

[29] Khomsin Khomsin et al. "Accuracy Analysis of GNSS (GPS, GLONASS and BEIDOU) Obsevation For Positioning". In: *E3S Web of Conferences* 94 (Jan. 2019), p. 01019. DOI: 10.1051/e3sconf/20199401019. (Visited on 05/25/2022).

[30] Inc. L3Harris Technologies. *L3Harris FVR-10 specsheet*. URL: https://www.l3harris.com/sites/default/files/2021-11/as-pes-latitude-resources-fvr-10.pdf (visited on 02/17/2022).

[31] Inc. L3Harris Technologies. *L3Harris FVR-55 specsheet*. URL: https://www.l3harris.com/sites/default/files/2021-11/as-pes-latitude-resources-fvr-55.pdf (visited on 02/17/2022).

[32] Inc. L3Harris Technologies. *L3Harris FVR-90 specsheet*. URL: https://www.l3harris.com/sites/default/files/2020-08/L3Harris_Collateral_FVR-90_SpecSheet_Web_0520.pdf (visited on 02/17/2022).

[33] Oscar Liang. "PWM and PPM Difference and Conversion". In: (2013). URL: https://oscarliang.com/pwm-ppm-difference-conversion/ (visited on 05/24/2022).

[34] geeksforgeeks.com Mayank Kumar. *How GPS works?* URL: https://www.geeksforgeeks.org/how-gps-works/ (visited on 05/25/2022).

[35] Walter Musial et al. *Offshore Wind Market Report: 2021 edition*. Tech. rep. Department of Energy, Office of Energy Efficiency & Renewable Energy, 2018. (Visited on 05/25/2022).

[36] National Coordination Office for Space-Based Positioning, Navigation, and Timing. *Other Global Navigation Satellite Systems (GNSS)*. URL: https://www.gps.gov/systems/gnss/ (visited on 05/25/2022).

[37] National Oceanic National Weather Service and Atmospheric Administration. *Safeboating Marine Forecast*. URL: https://www.weather.gov/safety/safeboating-marine#:~:text=Understanding\%20a\%20marine\%20forecast\%20is\%20critical\%20to\%20safe,roughness\%20of\%20near\%20shore\%20waters\%2C\%20and\%20significant\%20weather. (visited on 05/21/2022).

[38] United Nations. URL: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 05/24/2022).

[39] United Nations. *The Paris Agreement*. URL: https://www.un.org/en/climatechange/paris-agreement (visited on 05/23/2022).

[40] National Oceanic and Atmospheric Administration. *Magnetic Declination Estimated Value*. URL: https://ngdc.noaa.gov/geomag/calculators/magcalc.shtml?#declination (visited on 05/25/2022).

[41] THE EUROPEAN ECONOMIC REPORT FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT THE COUNCIL, SOCIAL COMMITTEE, and THE COMMITTEE OF THE REGIONS. *State of the Energy Union 2021*. Tech. rep. European Commission, 2021. (Visited on 05/25/2022).

[42] Nightgale security. *Nightgale security specification*. URL: https://www.nightingalesecurity.com/specs-faqs/ (visited on 02/17/2022).

[43] Bosh Sensortec. *BMP180 Datasheet*. URL: https://www.mouser.fr/datasheet/2/783/BST-BMP180-DS000-1509579.pdf (visited on 05/16/2022).

[44] Ltd Shenzhen Beitian Communication Co. *BN-880 Datasheet*. URL: http://sz-beitian.com/ProductsDetailEng?product_id=1340 (visited on 05/25/2022).

[45] Shippo.co.uk. *Sea or Air Freight?* URL: https://www.shippo.co.uk/faqs/sea-or-air-freight-what-s-for-me/#:~:text=While\%20sea\%20freight\%20accounts\%20for\%203\%25\%20of\%20the,a\%20massive\%20545g\%20per\%20tonne\%20moved\%20a\%20km. (visited on 05/21/2022).

[46] Pawel Spychalski. "Generate PPM signal with Arduino". In: (2016). URL: https://quadmeup.com/generate-ppm-signal-with-arduino/ (visited on 05/24/2022).

[47] Statista. "Logistics Industry Worldwide". In: (2020), pp. 2–31. URL: https://www.statista.com/study/67390/logistics-industry-worldwide/ (visited on 05/25/2022).

[48] Ukrspecsystems. *PD-2 unmanned aerial system*. URL: https://ukrspecsystems.com/pd-2-uas (visited on 02/17/2022).

[49] Wikipedia. *atan2*. URL: https://en.wikipedia.org/wiki/Atan2 (visited on 05/23/2022).

[50] Wing. *The Press page of Wing's website*. URL: https://wing.com/press/ (visited on 05/21/2022).

[51] Wingcopter. *The About page of Wincopter's website*. URL: https://wingcopter.com/about (visited on 05/21/2022).

[52] Zipline. *The Technology page of Zipline's website*. URL: https://flyzipline.com/technology (visited on 05/21/2022).

Appendix A

Link to the code

Link to the code. The link directs to a repository of the state of the code at hand in.
`https://github.com/BestestTeam/delivery-drone/tree/e0f57364dc48e9ceb97deb34256265afc2de7880`
This link directs to the most up to date version of the code:
`https://github.com/BestestTeam/delivery-drone/`