AALBORG UNIVERSITY

STUDENT REPORT

ED3-1-E15

Autonomous Bale Collector

Students:
Alexandra Dorina Török
Allan Gjerlevsen
Andrius Kulšinskas
Emil Már Einarsson
Thomas Thuesen Enevoldsen

Supervisor:
Akbar Hussain

December 15, 2015

AALBORG UNIVERSITY

STUDENT REPORT

School of Information and Communication Technology
Niels Bohrs Vej 8
DK-6700 Esbjerg
http://sict.aau.dk

Title:
Autonomous Bale Collector

Theme:
Scientific Theme

Project Period:
Fall Semester 2015

Project Group:
ED3-1-E15

Participant(s):
Alexandra Dorina Török
Allan Gjerlevsen
Andrius Kulšinskas
Emil Már Einarsson
Thomas Thuesen Enevoldsen

Supervisor(s):
Akbar Hussain

Copies: 3

Page Numbers: 91

Date of Completion:
December 15, 2015

Abstract:

Contents

Preface

The project entitled —– was made by five students from the Electronics and Computer Engineering programme at Aalborg University Esbjerg, for the P3 project in the third semester.

From hereby on, every mention of 'we' refers to the five co-authors listed below.

Aalborg University, December 15, 2015

Allan Gjerlevsen
<agjerl13@student.aau.dk>

Emil Már Einarsson
<eeinar14@student.aau.dk>

Andrius Kulšinskas
<akulsi14@student.aau.dk>

Alexandra Dorina Török
<atarak14@student.aau.dk>

Thomas Thuesen Enevoldsen
<tten14@student.aau.dk>

Chapter 1

Introduction

For this project, we decided to have a collaboration with a group from another field of study, that being the mechanical engineering group **M3-1-E15**.

The project proposes a solution for the problems around the process of collecting bales. Both groups agreed that the appropriate solution would be to design and build an **Autonomous Bale Collector (ABC)**. This was decided in the early meetings, where we also discussed the semester's requirements to make sure that they will be fulfilled in the case of each group. To do so, we have made a rough draft of the work division (see Figure 1.1) and came up with a few most important topics, which should be covered in either one of the reports. In short, we were in charge of creating the autonomous part of the vehicle and M3-1-E15 was in charge of creating all of the mechanical design. Even though the design was mostly assigned to them, we did have a few discussions on it together since the sensors we planned on using require a certain specific placement.

Figure 1.1: Work division.

To not only analyse and expand the problem, but to also help us decide on the best possible solution to it, we decided to use a 6W-diagram as a model and also did some preliminary research regarding some of the parts. This can be seen in the Problem Analysis chapter of this report.

1.1 Initiating problem

Is it possible to solve the problem of collecting bales by trying to implement the Autonomous Bale Collector?

Chapter 2

Problem Analysis

6000 years ago, most people in Northern Europe were hunters, but as the population rose, hunting and gathering food were not enough to sustain them anymore. They turned to agriculture and started to exploit the soil in order to get extra food. Many years later, this area had improved enormously and it became the primary source of food. The Stone Age farmers had now included domestic animals into their farms, which provided them milk, meat, clothes and manure. When farmers started producing more food than they could consume, more advanced cities were developed. People could now specialize as blacksmiths, bakers and many other professions. By trading and working more efficiently, people got more time to gain new knowledge. Since then, farming efficiency continued to increase and the modernization of our society would probably have been impossible without agriculture. [1]
Nowadays one of the most important crops is grain. People use it as a very important food source in many different ways, but the most common one would probably be its use in bread. In addition, when the grain is harvested, we get straws, which can be fed to the domestic animals. They can also be used to make heat in a boiler or to produce electricity by burning them. Another way to exploit the straws is to use them as biomass in order to create energy.

Figure 2.1: Harvesting procedure.

The procedure of harvesting grain and straws has 3 main steps, which are illustrated in figure 2.1. A combine harvester cuts the straws and shakes off the grain, which will be collected in a wagon and driven to a storage space - first picture. From here, the straws will be pressed into bales by a baler - second picture. Finally, the bales will get picked up by a tractor with a front loader and loaded onto a wagon that will transport them to a barn, or some assembly point, where it will be unloaded -

third picture.
To see the shape of the pressed bales look at figure 2.2.

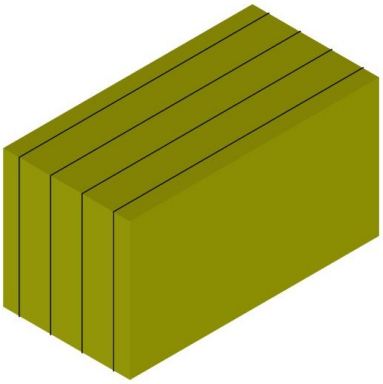


Figure 2.2: The shape of a pressed bale

The size of the big bales is 131 cm x 121 cm x 238 cm, whereas the size of the mini big bales is 88 cm x 82 cm x 220 cm (H x W x L).

The big bales weigh between 440 and 600 kg and the mini big bales weigh between 156 and 235 kg, depending on how much they are pressed and also the amount of moist in the bale.

2.1 6W-diagram

To get an overview of the problems concerning bale collection we created a 6W-diagram (See figure 2.3) which will be analysed further below.

Figure 2.3: 6W-diagram for bale collection

What is the problem?

Bale collecting takes up many work-hours since one worker drives a tractor with a front loader that picks up the bales while another worker drives a tractor with a wagon connected to it where the bales are stored. When the wagon is fully loaded, it is driven to a barn or a place on the field where the bales will be temporary stored. There, the bales need to be unloaded and this whole process will continue until all the bales have been collected.

Where does the problem occur?

The problem occurs on the farmers' fields where grain has been harvested.

When does the problem appear?

Usually the optimal start of harvesting is in the beginning of August and it should be done before the beginning of September. Of course, it depends on when the grain is mature, which again depends on the amount of sun and rain in the season. After the harvest, the bales are pressed and then they can be collected and stored. This will probably be done in late August or in September, all depending on the weather.

How is this a problem?

Late summer is a very busy time of the year for farmers since they need to harvest, press and collect bales in a relatively short period of time. There are several potential problems with exploiting the grain. First of all, it needs water so it does not dry out. It also needs the right amount of sun to mature. Heavy winds could blow the straws down to the ground, where they will be impossible to harvest. In addition, when the grain is finally mature it needs to be dry because if the grain contains more than 15% water, its value decreases. It can be dried, but this costs money. The bales can be pressed if the straws are dry enough after being harvested. Then, the bales can be collected if they contain less than 17% water. Higher water levels decreases the value of the bales. So, with all these potential delays, it is impossible to plan out the whole process. Also, the farmers often get very busy since they still need to take care of their domestic animals on the farm. [2]

Why is this a problem?

Many farmers have economic problems because of lower sales prices for their goods. In some years the prices for bales of straws are so low that it is a better economical solution to plow down the straws instead of pressing them to bales, if the farmer does not need the resources himself to feed his domestic animals.

Who has this problem?

The ones having the most economic problems are the farmers with least soil, since it is cheaper to run farms of big sizes. Also, they have less workers and not so advanced machines to help them do the work. They might depend on a machine pool to do some of the work.

2.2 Problem description for an automatic bale collector

In order to make a fully automatic bale collector, there are several issues that need our attention.

Mapping the field

Before the machine can start working, it needs to get the location of the bales. They will be spread on the field in a certain pattern which can vary from field to field depending on the amount of straws on the field and also the route of the baler.

Calculate optimal travel route

To increase the efficiency of the bale collector, the optimal travel route has to be calculated. The less distance the machine has to travel to get all the bales transported to the storage, the less time will be spent. The ideal way to minimize transport time would be to store the bales somewhere in the middle of the field; however, this could be a problem since the bale collector would have to avoid other bales on its way to the unload area. Also, there could already be bales at that point. Additionally, the farmer may find the unload area to be inconvenient for further transportation of the bales.

How to approach a bale and validate it

When a bale has been selected to be picked up, it is important to approach it in the right direction since it needs to be lifted up on their small side. The bale also needs to be measured in order to make sure that it is in fact a valid bale and not some other stationary object.

Storage data

When a bale has been picked up and delivered to the storage point, the storage data has to be updated so that the bale collector will not try to put the next bale in the same place. A manual access to the storage data will be needed so that the farmer can move bales as he wishes.

Safety and error handling

The procedure will continue in a loop while constantly scanning for people, animals and other possible obstacles. Safety procedures should prevent any accidents. The system's fail check should run in certain intervals so an alarm can be sent to a human worker if a trouble is encountered. The bale collector should have a manual control for these situations.

Environmental challenges

The machine should be able to navigate in the dark to optimize its efficiency. Also, the electronic system should be shielded so that the machine is able to function

under any weather conditions.

2.3 Preliminary Research

2.3.1 Communications

Since the machine we have in mind is expected to be an autonomous one, it is highly important that it has the ability to communicate data. What this means is that it should be able to send or receive information over a certain distance. This transfer of information could happen, for example, between the machine and an external computer which a farm employee will use in order to check what is happening on the field. The information can consist of the data received from the different types of equipment which we may use, such as cameras, sensors etc.

We instantly excluded the types of communication that required cables because of the fact that our machine will be in a continuous movement on the field. Therefore, we thought about the types of wireless communications that we could use and came up with the following:

- **Wi-fi**
- **Cellular data services**
- **Mobile satellite communications**
- **High Frequency Radio communications**

Before deciding anything on the communication topic, we agreed that the most important aspects to take into consideration are the range and the cost of both implementation and maintenance that the system would require.

So, how do wireless radio based systems work? The concept is simple. There are two main components:

- **Transmitter** - from which the information is being sent
- **Receiver** - which picks up the information carried by an electromagnetic wave

Wi-fi

Wi-Fi is a wireless local area network which uses radio waves; it enables devices to connect easily to the Internet. A computer's wireless adapter translates the data into a radio signal and transmits it by using an antenna. A wireless router then receives the signal, decodes it and sends the information to the Internet using a wired Ethernet connection. This process also works in reverse.
Wi-fi radios transmit at frequencies of 2.4 GHz or 5 GHz. These are considered high values compared to the frequencies used for cell phones or television and they make it possible for more data to be carried. [3]

Cellular data services

Cellular data services offer coverage only within a certain range from the closest cell tower. Technologies have now evolved to 4G networks, but 3G are also being used. Today, 4G networks boost transfer speed by up to 30 times more than the 3G ones.

One advantage of using cellular data services is that they use multiple low-power transmitters (100 W or less). Each area divided into cells is served by its own antenna and by a base station consisting of a transmitter, a receiver and a control unit. The cells are set up such that the antennas of all neighbors are equidistant - hexagonal pattern (See Figure 2.4).

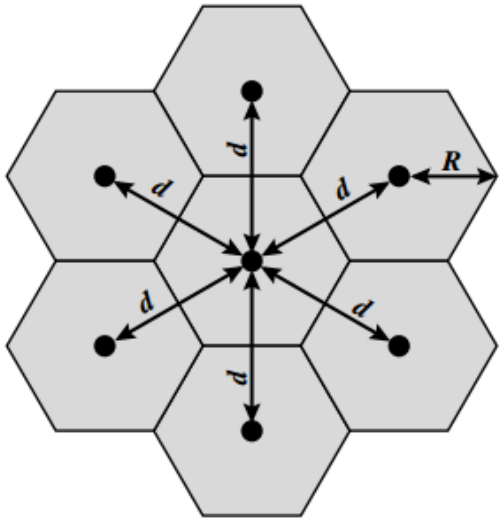


Figure 2.4: Hexagonal pattern of cellular networks

Usually, cell towers are grouped in areas of high population density where there are the most potential users. They can be placed hundreds of meters to kilometers apart depending on this. In urban areas, the spacing between the cell towers is smaller than in the suburban or rural ones.

Mobile satellite communications

Mobile Satellite Communications is often used where other wireless connections are unavailable, such as largely rural areas or remote locations. So, even though this might sound like a good way to go, the costs would probably be too high and not worth it, especially for smaller farms.

High Frequency Radio Communications

Military organizations have used HF radios for both strategic and tactical communications for more than 80 years; however, with the advent of satellites, HF had been de-emphasized and fell into disuse. HF radio can be used for:

- **Line-of-Sight (LOS)**: Range, typically less than 30 km, is limited by terrain obstructions and/or earth curvature.
- **Ground (surface) Wave**: Useful range is up to 50 km on land, 300 or more km over the sea.
- **Beyond Line-of-Sight (BLOS)**: Range to about 400 km using Near Vertical Incidence Skywave (NVIS).
- **Long Range Communications**: Communications up to ranges of 4000 km and beyond.

Principle

"Radio waves belong to the electromagnetic radiation family. Radio signals radiate outward, or propagate, from a transmitting antenna. Radio waves propagate at the speed of light. We characterize a radio wave in terms of its amplitude, frequency, and wavelength (See Figure 2.5). Radio wave amplitude (or strength) can be visualized as its height being the distance between its peak and its lowest point. Amplitude, which is measured in volts, is usually expressed in terms of an average value called root-mean-square, or RMS. The frequency of a radio wave is the number of repetitions or cycles it completes in a given period of time. Frequency is measured in Hertz (Hz); one Hertz equals one cycle per second. Radio wavelength is the distance between crests of a wave. The product of wavelength and frequency is a constant that is equal to the speed of propagation. Thus, as the frequency increases, wavelength decreases, and vice versa. Radio waves propagate at the speed of light (300 million meters per second). To determine the wavelength in meters for any frequency, divide 300 by the frequency in megahertz. So, the wavelength of a 10 MHz wave is 30 meters, determined by dividing 300 by 10." [4]

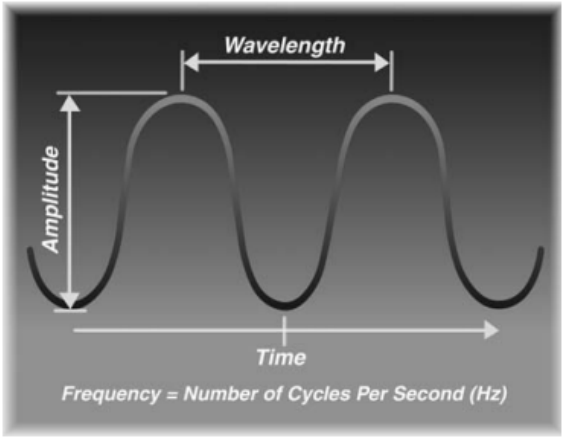


Figure 2.5: Radio wave properties [4]

After analyzing our options, we came to the conclusion that HF radio communications might be the right choice for our project because it fulfills both range and cost requirements, while also being easy to implement.

2.3.2 Safety

For any autonomous machine that works both in a closed and open space, it is important to have some safety measures installed to ensure that no accidents occur. If the device performs tasks around both humans and animals, it needs to make sure it avoids causing any harm to them. There is also a need to ensure that the machine is not a safety risk due to faulty parts which could cause a sudden combustion or similar problems.

Obstacle detection

Our autonomous machine needs to be able to detect living beings - humans and animals - as well as stationary objects in order to avoid collision with them. The easiest way to detect living beings is to look for heat signatures, since warm blooded animals and humans emit certain amount of heat. There are a few choices between special cameras and sensors, such as:

- Thermal cameras
- Passive infrared (PIR) sensors
- Kinect

Thermal cameras

Thermographic (or infrared) cameras are able to use infrared radiation emitted by objects to form a heat map in an image (See figure 2.6). Therefore, they can be used to measure the amount of heat in the area, essentially detecting if there are any living beings present.

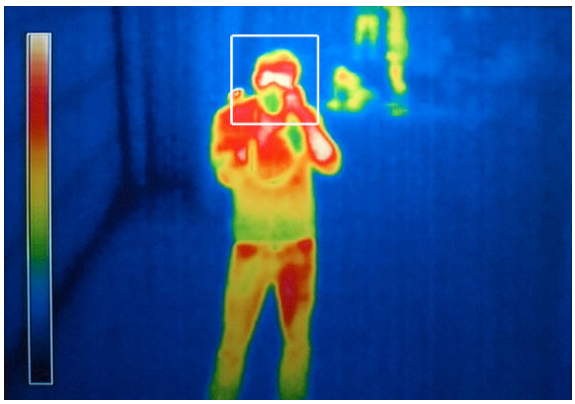


Figure 2.6: An image of a person captured with a thermal camera. [5]

These cameras are extremely accurate and some can detect human-sized objects over a 20 kilometer distance. However, these cameras usually have narrow detection angle, meaning they need to point directly towards where the objects are in order to

detect them. This technology is also expensive with prices ranging anywhere from $1.000 to $2.000. Cheaper, but less effective technologies are already in the making, but the amount of living being detection needed for the machine in the area it is operating does not justify the huge costs.

PIR sensors

Passive infrared sensors, same as thermographic cameras, also detect infrared light emitted by objects in its detection area. These sensors do not emit any amount of energy in order to detect the objects, thus they are called passive. These sensors are commonly used as motion detecting sensors for burglar alarms as well as for lighting systems that turns on when someone enters a room. This is because PIR sensors detect infrared radiation sent out by the humans or warm blooded animals. These sensors usually have a detection range of about 10 meters in a 180° angle. They are fairly cheap too, with most being priced under $10. There are more powerful sensors as well, having detection distance of up to 30 meters and sometimes an angle of 360°. The disadvantage of these sensors is, however, that they only detect change in the area, therefore the object needs to move while in the sensor's detection range. Even though it is very unlikely that there would be a living object standing still in the field while the machine is running, the risk is there.

Kinect

Kinect, a motion sensing device for use with Xbox game consoles makes use of a combination of two cameras in order to detect a human being, its position and gestures.

- **Infrared depth camera** which can detect a human being as well as the depth level of the surrounding area, determining the position of object in 3 dimensions. It sends out an infrared signal from one end and receives it on the other, which is then processed.
- **RGB camera** which can see the gestures of a human and send them for processing.

While Kinect's primary use is to essentially turn a human into a game controller, who can play the game by making gestures, it can also be used to simply detect humans and their location. It can detect objects within a range of 50 centimeters to 5 meters. Usage of Kinect has its disadvantages, though. It is about 30 centimeters wide and requires a computer to process the data, therefore it can be difficult to fit it on the machine and impossible to set it up on a small prototype. The sensor itself is also rather expensive, costing $150 alone. Additionally, the infrared radiation from the sun will make it problematic for the Kinect to work properly in daylight.

Stationary object detection

It is important for an autonomous machine to be able to avoid collision with stationary obstacles just as much as with living ones. However, it is safe to assume that there would be little to no stationary objects on the field, so we only need minimal amount of detection.

An active infrared sensor can be used to detect an object in front of the machine. The infrared LED sends out an infrared signal, which then can reflect from an object in front and bounce back to the receiver on the sensor (See figure **??**).

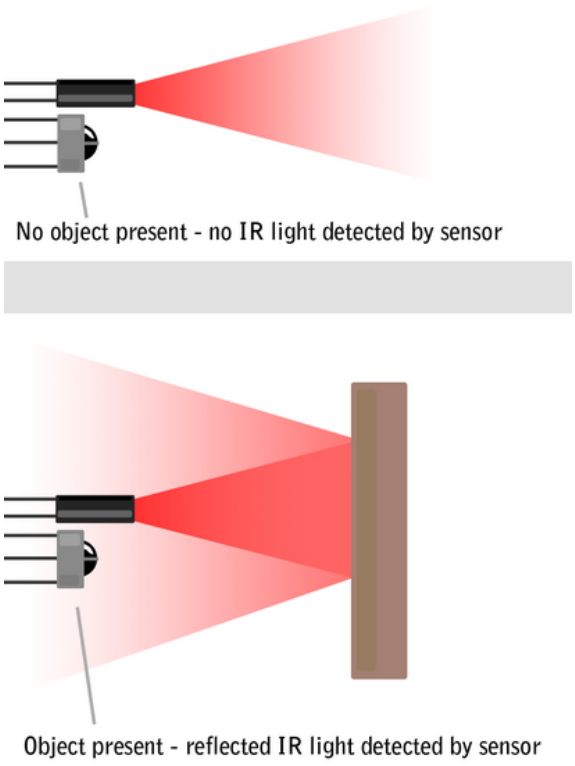


Figure 2.7: A demonstration how active infrared sensor can detect objects. [6]

The detection distance and angle of these sensors varies depending on one's needs, but detecting a stationary object in about 5 meter distance from the machine is possible. The sensor could, however, fail to function if the object does not reflect infrared light, nevertheless, it is highly unlikely that such objects would be present on the field.

There are several alternatives to the infrared sensor such as ultrasound or laser sensors. They perform in a similar manner and the only major differences are the signals that are used for the detection.

2.3.3 Computer Vision

Humans use their eyes to be able to sense the world around them. The brain then processes all that information. Computer vision is the same process, but for computers. The camera represents the eyes and the computer used for processing represents the brain (See figure 2.8). The computer can see things faster and in more detail, but its boundaries lie within its original programming and parameters. [7]

Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding. [7]

Figure 2.8: A demo of computer vison [8]

The different uses of computer vision can be, and not finalized:

- Agriculture
- Augmented reality
- Autonomous vehicles
- Biometrics
- Character recognition
- Forensics
- Industrial quality inspection
- Face recognition
- Gesture analysis
- Geoscience
- Image restoration
- Medical image analysis
- Pollution monitoring
- Process control

- Remote sensing
- Robotics
- Security and surveillance
- Transport

[7]

2.3.4 OpenCV

OpenCV stands for Open Source Computer Vision Library. *The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.* [9]

The library is built up by more than 2500 algorithms. The algorithms include a set of both classic and state of the art computer vision and machine learning. It is a good starting platform for many projects that can then be built on top of. These projects can range from face detection to form analysis. [9]

2.3.5 Laser and Ultrasonic rangefinders

A rangefinder is an instrument used for estimating the distance to an object, from the range finding device. Rangefinders come in many different shapes and sizes, and are also used for many other different applications. The distance calculation is made by timing how far away the measured object is, and then multiplying by the speed of light or sound, depending on what type of rangefinder is used.

$$D = \frac{ct}{2} \tag{2.1}$$

where c is the speed of light/sound and t is time it takes to reach the object and return.
Both sound and laser rangefinders work by firing a small pulse in the desired direction. If the pulse hits a surface that is uneven or at an angle, the pulse is unable to return correctly to the rangefinder. Though, on impact with the object the pulse will scatter, allowing the rangefinder to pick up some of the scattered pulse, if it is not returned directly to the rangefinder. [10]

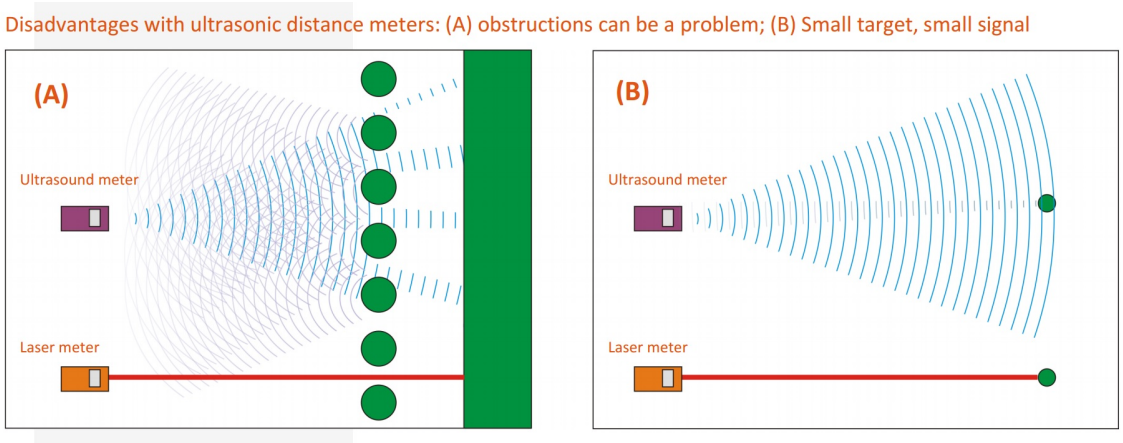


Figure 2.9: A comparasion between ultrasonic and laser distance equipment [10]

Ultrasonic sensors often use much wider pulses than lasers, since the pulses from the ultrasonic devices are in the form of sound waves. Inaccuracy may occur, by hitting an object in the path towards the desired object. The scatter that occurs when hitting the smaller objects can cause inaccurate readings, this can be seen in figure 2.9. Where a lasers measurement is done using a beam, and can more precisely be aimed at a target.

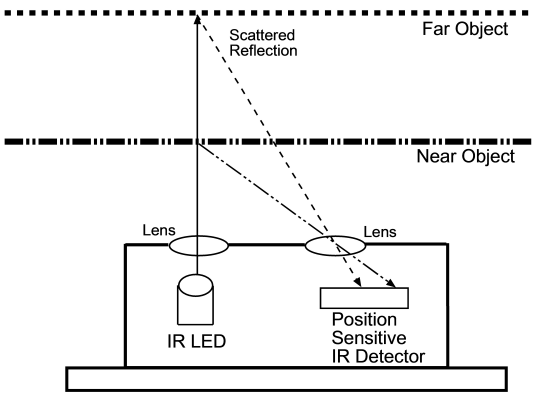


Figure 2.10: A rangefinder using angles to calculate distance [11]

Other than measuring the time it takes for a pulse to travel to and from an object, different equipment exists that instead uses trigonometry to calculate the distance to objects.
Figure 2.10 shows how the distance to an object can be calculated using trigonometry. This type of rangefinder uses the distance between the location of which the pulse is emitted to the point of absorption. Using this distance and the angle of entry, it is possible to calculate the distance from the rangefinder to an object. [12]

2.3.6 GPS

Global Positioning System (GPS) is a navigational system that consists of 20+ satellites. This space-based navigation system was originally intended for different military applications, but was later made free for public use. Because of the widespread satellites, the system works everywhere in the world and also during any kind of weather conditions.

GPS has the capabilities to supply the users with five different positional data. [13]

- Current position (A)
- Direction
- A second-parties position (B)
- Optimal route from A to B
- Time of arrival

The user interacts with GPS by using different kinds of receivers. Based on the desired application, these receivers come in different shapes and sizes.

Smaller GPS receivers are used to track animals or for smaller electrical systems. The most common GPS receiver is the one used for navigational systems for vehicles.

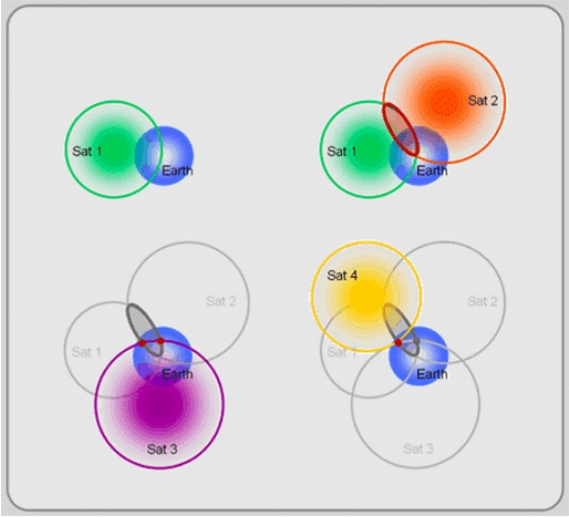


Figure 2.11: GPS trilateration process [14]

GPS works by using the satellites in space as reference points for locations on earth. Whilst figuring out the current location of a receiver, it is narrowed down by finding the distance to four of the satellites. The first satellite determines the receivers location somewhere on a sphere, as shown by Sat 1 on figure 2.11. The second satellite narrows down the locations to the possibilities within the intersection of the two spheres. The third satellite then limits the large area created by the satellite prior, to two locations. A fourth satellite then acts as a correction, to calculate

the timing and then selects one of the two points determined by the previous three satellites. [15]

To figure out the distance from the satellites to the receiver, the receiver receives Pseudo Random Code [16], which is a complicated series of digital code. The receiver has its own Pseudo Random code that it attempts to match with the code received from the satellite. The amount of time the code from the satellite is delayed, multiplied by the speed of light equals the distance from the receiver to the satellite. [17]

When the GPS signal travels from the satellite to the receivers, errors may occur when it is slowed down by the particles in the Ionosphere and in the Troposphere. Mathematical modelling can correct most of the errors caused by the atmosphere. Since the GPS receiver uses four satellites to find its position, the amount of errors caused by the different layers of the atmosphere is multiplied by 4. [18]

To combat the errors which follow the regular GPS and to improve its accuracy, Differential GPS was created. DGPS uses two receivers, where one receiver is dedicated to correcting errors created during the transmission through the atmosphere. The receiver doing the error correction is a stationary GPS receiver, which instead of calculating its position calculates the timing instead. It is then able to compare the actual GPS signal travel time with the measured travel time, this is the error correction. The stationary receiver then transmits this information to the secondary receiver which in most cases is a moving receiver. [19]

Latitude and longitude

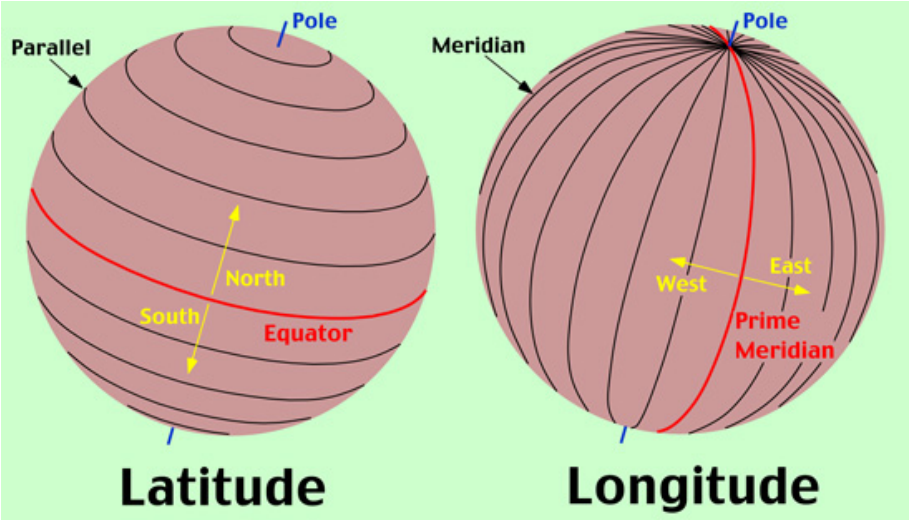


Figure 2.12: Latitude and Longitude [20]

Latitude are the horizontal lines shown on the left sphere in figure 2.12. These lines measures the position between the north and the south poles. The equator is found at 0 degrees and the poles are found at 90 degrees north and 90 degrees south. One

degree of latitude is 111km and one minutes of latitude is 1.85km.
Longitude are the vertical lines shown on the right sphere in figure 2.12. These lines run from the north to the south pole and are used to measure the east-west position. The longitude at 0 degrees runs though Greenwich, England. The degree measured to the west of Greenwich are degree west and the ones measured to the east are degrees east. [21]

When dealing with positions given in latitude and longitude there are three common formats.

- DDD° MM′ SS.S” (Degrees, Minutes and Seconds)
- DDD° MM.MMM′ (Degrees and Decimal Minutes)
- DDD.DDDDD° (Decimal Degrees)

2.4 Stakeholders

2.4.1 Interested parties

This is the group of people that can be effected by building and using an autonomous bale collector. Also including people who would have an interest or an influence of our project.

Politicians

Since there are only few special laws for autonomous vehicles in agriculture, because of the fact that this technology is in a very early stage of development, politicians will have influence on new laws in the area.

Farmers

They will be the ones using the vehicle and it will change their work routines. Also it should lower the human work load which could affect the amount of workers on the farm.

Machine pools

Machine pools might take interest in the project since the automatic bale collector could reduce their need for human workers in the busy harvest season.

2.4.2 Actors

This group will include those who work on the project and those who helps with the development of it.

Supervisor

Our supervisor Akbar Hussain can be consulted about technical issues as well as our written part of the project.

Group ED3-1-E15

The members of this group will be in charge of producing a solution for the project and it includes:

- Allan Gjerlevsen
- Andrius Kulšinskas
- Alexandra Dorina Török
- Emil Már Einarsson
- Thomas Thuesen Enevoldsen

2.4.3 Technology holders

Machine producers

Companies that have produced machines for agriculture in years will of course have a lot of valuable knowledge in this area.

2.5 Legislation

Before starting the process of building a solution to our problem there are some legislation that we need to consider.

The Road Traffic Act

All vehicles driving in public areas need to follow the Road Traffic Act but since our robot will only be driving on farmers' private property, these rules do not apply so we will not research this any further.

Health and safety (At a workplace)

There are certain rules for working with autonomous vehicles. Among them are safety precautions to prevent accidents between humans and machines. The vehicle has to make a warning sound or blink with some yellow light before start driving to alert nearby people. Besides that, it needs emergency stops and some system that can stop the machine if people are approaching. The area where the machine works within has to be labeled to warn people of the possible danger.

2.6 Risk management

The implementation process of the proposed Autonomous Bale Collector (ABC) has a number of risk factors which we have to take into account, since they could have a negative impact on our end-result. We came up with a plan for each individual risk, which can be seen in Table 2.1.

Risk factor	**Probability**	**Impact**	**Mitigation approach**
Exceeding the budget	M	L	Make sure to cautiously select the desired products before handing in the excursion request form
Insufficient knowledge on hardware and/or software	M	H	Consider a few contact persons who have both the knowledge and the availability to guide us
Damaging certain electrical components during the development stage	L	H	Purchase the same product again or find a similar one which could replace it
Too high of a scope compared to the allocated time	H	H	Level down the scope, but also make sure to give a valid reason why this has been done and write about it in the discussion chapter
Parts not arriving on time	M	H	Purchase the parts from a local store ourselves and get the money back from the University or find similar materials that would serve our needs at the lab.

Table 2.1: Risk management and mitigation

The risks are categorized into two different scales. One of them is **Probability**, which represents the chance of occurrence and another one is **Impact**, which represents the level of damage that a risk would cause in case it would actually turn up. Each scale has 3 levels going from **Low (L)** to **Medium (M)** to **High (H)**, the first one being considered as a minor issue and the last one as the most severe.

Chapter 3

Problem Definition

3.1 Conclusion of Problem Analysis

During the research and collaboration with the group M3-1-E15, we have figured out what technologies we will use to solve the problem stated in the 6W-diagram (See figure 2.3)

M3-1-E15 will be designing the mechanics for picking up and delivering the bales. Meanwhile, our focus will be on the electrical and autonomous part of the project. Throughout the development phase multiple discussion will have taken place regarding sensor placement and other designs they will have to take into consideration when designing the vehicle.

We will be developing a vehicle that autonomously navigates and collects bales based on given GPS coordinates. Besides navigating with GPS, the vehicle will use a close proximity system to sense nearby objects. The vehicle will also be equipped with PIR sensors to be able to detect humans and animals for safety reasons. A camera will be used to identify a bale for pickup, once the vehicle arrives at the given GPS location. To increase the precision of the vehicle movement, encoders will be used to more accurately determine the movement of the vehicle.

For computer vision we will use an HD camera, OpenCV and a laptop. The use of a laptop is only for prototyping purposes and will be re-discussed in the respective chapter. A microcontroller will be used to control the vehicle and its navigation using the GPS and laser sensors.

We will attempt to build a prototype similar to the design supplied from the mechanical engineering group, but without building a functioning bale picking up mechanism. Our main focus will be on navigating to the location of the bale and then correctly positioning the vehicle so that it can be picked up correctly.

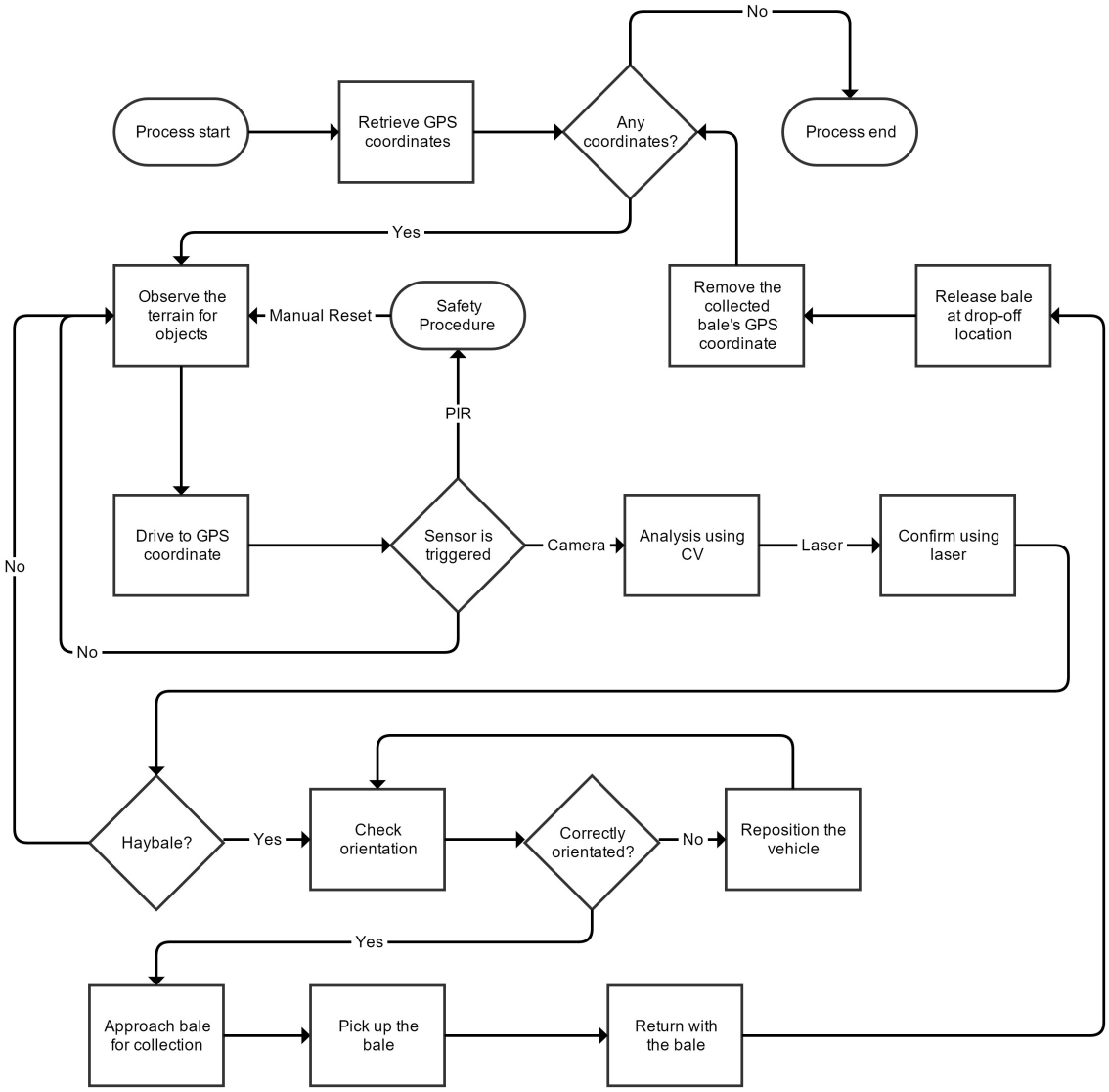


Figure 3.1: Flow chart describing our desired system. [22]

The flowchart shown in figure 3.1 displays the main process of our initially planned autonomous navigation system.

It will be using a two factor recognition system where the computer vision along with a laser will confirm that the observed item is a bale for pick up. After the bale is correctly recognized the CV and laser will then confirm that the bale is appropriately orientated, and the initiating the pick-up process. The bale collection will continue until there are no GPS coordinates left in the database, which means that all the bales have been collected.

3.2 Problem Delimitation

Two of the major limiting factors for us during this project period is time and funds. We are limited to only having the given time during this semester to work on the product and will therefore not be able to develop a full-scale version of our solution. Thus, we will only focus on building a down-scaled version to demonstrate the key concepts that will help solve our problem stated previously in the report.

M3-1-E15 will not be building a prototype, to demonstrate the lifting mechanisms and other features of the vehicle. This is because they have the tools available to simulate their entire prototype and therefore save time. This means that we will be modifying our prototype to better fit our needs, to help it work as a proof of concept. Our prototype will not be equipped with a working lifting mechanism, but a navigation and recognition system.

For our prototype, we have decided to use a laptop and a webcam for image processing as a proof of concept. The reasoning behind using a laptop instead of a microcontroller is to harvest the greater computational power, which will increase the processing of OpenCV. A LIDAR-laser will also be used to detect bales as a proof of concept.

When aligning the bale correctly within the vehicle, the camera loses vision of the bale. To combat this issue we will be using some close proximity laser sensors to correctly align the bale.

As a safety feature, we will use PIR sensors to aid in the detection of animals and humans. This will help ensure that the vehicle does not harm any living creatures.

For the navigation from bale to bale, we will use a combination of a compass, a gyro and a GPS. The vehicle will retrieve a list of GPS coordinates from a database, where a route will be planned for the vehicle. Then, based on its current state, the vehicle will either move towards the next bale or drop off the recently picked up bale.

To navigate, the vehicle will compare its current coordinate with the ones from the database, then use the compass to ensure it is heading towards the desired point. If somehow the coordinates from the database are corrupted or non-existent, a search-mode will be created. This mode will search the field in a brute-force manner to locate the bales.

To use our given time as efficiently as possible, the GPS navigation system will be separate from the detection system using computer vision and the laser sensors.

3.3 Requirements

Below are our determined requirements for the prototype and testing. Some of them are based on the requirements that are set by M3-1-E15. We will be using the testing requirements to evaluate the prototype once the development has been completed. Below is a list requirements set by the mechanical engineering group that are relevant to our development process.

- Collection of 130 bales within 8 hours without the driving distance exceeding 60km
- Recognition, positioning, lifting and unloading of the bale should all take place within 2 minutes per bale
- "Overholde Arbejdstilsynets krav til automatiserede kørertøjer/arbejdsredskaber"
- Be able to recognize damaged/unusable bales
- Be able to identify obstacles such as humans and animals

Based on the above requirements, we have selected and modified the most important ones, which we will use as our success criteria. The additional ones will be addressed later in the report. These success criteria are separate for the prototype requirements and testing requirements.

Prototype requirements

- The vehicle should navigate autonomously between coordinates using GPS.
- The computer should detect a bale using a camera.
- The prototype should use computer vision to process the bale orientation.
- The prototype should use a laser to process the bale orientation.

Testing requirements

- The vehicle should navigate autonomously between coordinates using GPS.
- The laser sensor should detect the orientation of the bale.
- The camera and OpenCV should detect the bale and its orientation.
- The vehicle should be able to align the bale correctly for pick-up.
- The vehicle should be able to navigate back with the bale for drop-off.
- The vehicle should stop when the PIR sensors detect a living creature.

3.4 Problem Definition

The delimited scope for this project will be to develop three separate prototypes, each one acting as a proof-of-concept for the entire solution. A prototype showcasing the navigational capabilities and pathfinding aspects, a prototype using computer vision to detect the bales and do measurements and a prototype using a laser to detect, measure and check if the bale has been correctly aligned.

By testing our three prototypes separately and making them compatible for combination, it is possible for us to use the requirements written in the previous section to determine whether or not the prototyping was a success.

The parts left out of M3-1-E15's requirements will be addressed in the perspective part of the report, showing our reflection and future design choices regarding implementation.

Chapter 4

Development

4.1 Bale storage placement

A lot of time can be saved when collecting and transferring the bales to their storage spot by placing the storage spot somewhere in the center of the bales. This ideal center can be identified by calculating the centroid (C) of all bales by using the formula:

$$C = (\frac{x_1+x_2+...+x_n}{n}, \frac{y_1+y_2+...+y_n}{n})$$

where $(x_1, y_1), (x_2, y_2)...(x_n, y_n)$ are the bales' coordinates.

A simple example of four bales on a field is shown in Figure 4.1.

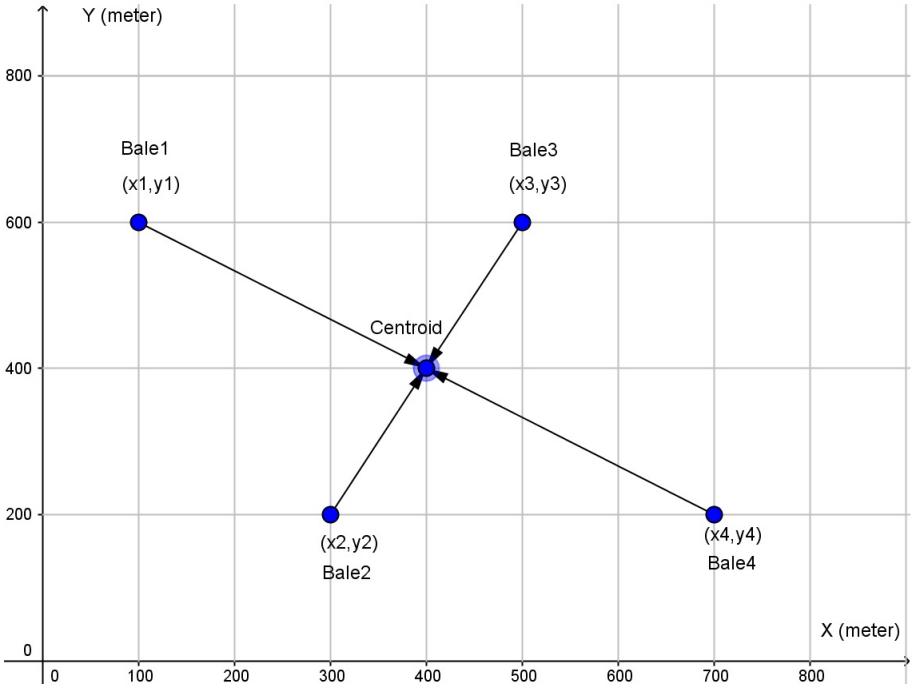


Figure 4.1: A field with centroid (C) and four bales shown as blue dots

In this case:

$$C = (\frac{x_1+x_2+x_3+x_4}{4}, \frac{y_1+y_2+y_3+y_4}{4}) = (\frac{100+200+500+800}{4}, \frac{600+200+400+400}{4}) = (400, 400)$$

The total travel distance to collect the four bales and bring them to the centroid (C) is equal to twice the length of the four vectors between C and each individual bale. This is because the bale collector moves from the center to a bale, then picks it up and goes back (See figure 4.1).

The length (L) of each individual vector can be calculated by using the general formula:

$$L_n = \sqrt[2]{(x_n - x_C)^2 + (y_n - y_C)^2}$$

where n represents the bale number and (x_C, y_C) the centroid's coordinates.

The lengths of the four vectors represented in Figure 4.1 are: $L_1 = 361m, L_2 = 224m, L_3 = 224m, L_4 = 361m$, which brings us to a total travel distance of 2340 m.

If instead the storage spot is placed at the corner of the field (See figure 4.2), the lengths of the four vectors are: $L_1 = 608m, L_2 = 361m, L_3 = 781m, L_4 = 728m$, resulting in a total travel distance of 4956 m.

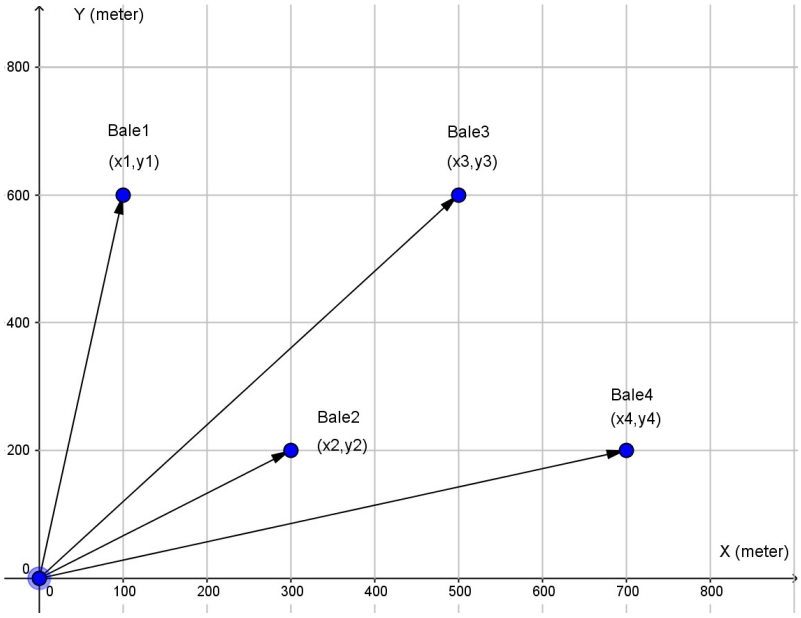


Figure 4.2: A field with the center in (0,0) and four bales shown as blue dots

The difference between the two travel distances is 2616 m. Hence, more than half of the travel time can be saved just by picking the ideal storage spot. However, the farmer may prefer to keep all the bales in his barn or in some other more convenient place.

4.2 Movement pattern and pathfinding

It is also important to cover the movement patterns of the machine. In our case, the autonomous machine must be able to not only determine the optimal drop-off point, but the most optimal pattern and path to pick up bales in the first place.

In a perfect scenario in 2-dimensional plane, where the machine starts at the drop off point, the pick up pattern is unnecessary, because we can pick up and carry one bale at a time. Therefore, it does not matter which bales we target first, since, in the end, we still have to travel the same distance – the distance from starting point to the bale and back. Figure 4.3 displays the scenario where there are no obstacles. No matter which pattern we use to pick up and drop off all 4 bales, we will always travel the same distance.

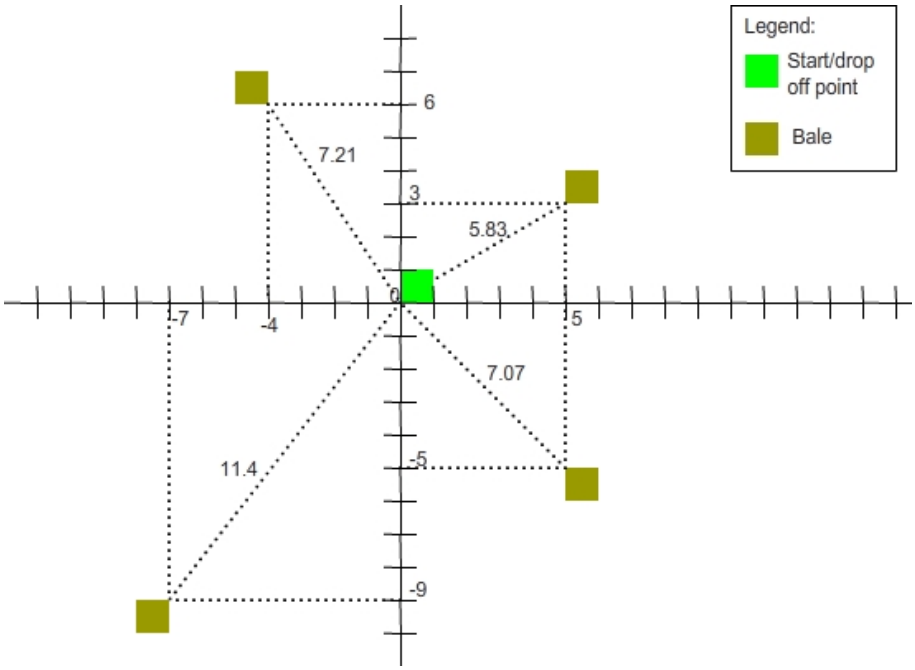


Figure 4.3: An example of perfect bale placement

However, we need to consider two things that may occur. One – the fact that the bales themselves are obstacles for the machine and two – the machine might not start at the drop off point. Therefore, it is more efficient to first target the closest bales. This way, we can avoid additional movement and time wasted trying to move around the closer bales when going for further ones. We can also save some time if the machine happens to start further from the drop off point by targeting the closest bale instead of the furthest one. The noticeable difference can be seen in figure 4.4, which shows two movement patterns for picking up and dropping off same bales. The left image shows the optimal way to do the task, which spends the least amount of time traveling, while the right image shows the inefficient way, which sacrifices time driving around other bales instead of targeting the closest ones first.

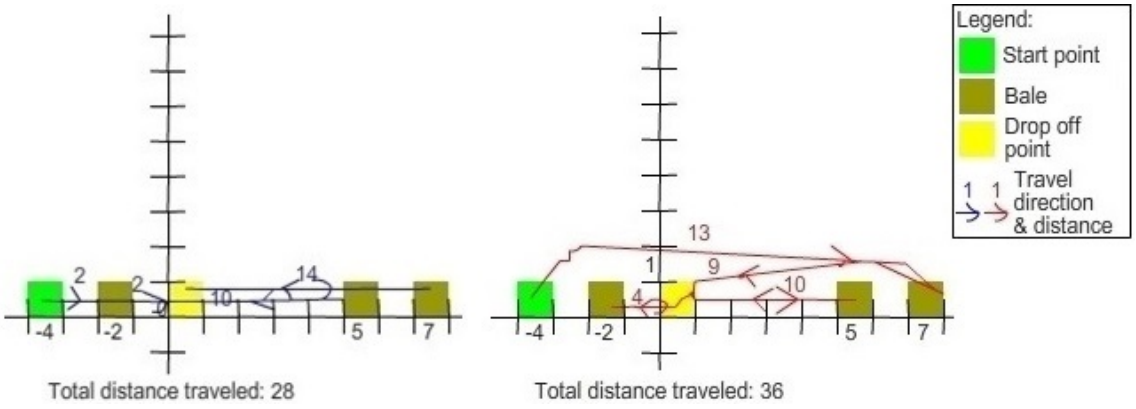


Figure 4.4: A comparison between the two travel patterns

Last thing to take into consideration are other possible obstacles. While they are similar to bales in a sense that they block movement, we do not possess any initial data about them – their location, amount, size. We can only detect an obstacle when it is within the range of the machine's proximity sensors, so the task to create the most optimal pick up pattern for the machine becomes difficult.

In turn, we need to introduce a pathfinding system which can use the data gathered by the proximity sensors to determine the best move possible. A common pathfinding algorithm is called A* (A star), which finds the most optimal path from origin to destination in a map by dividing the area into even sections and giving them values (see figure 4.5).

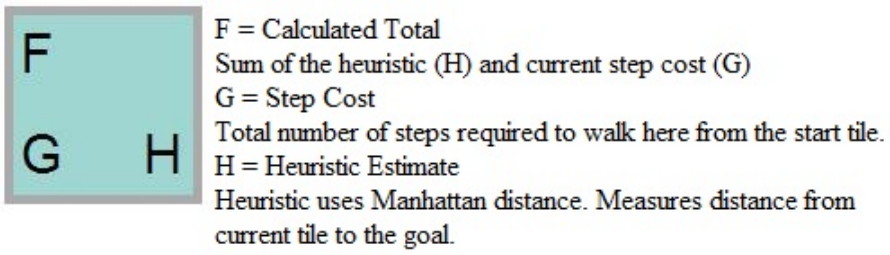


Figure 4.5: An image showing the description of numbers found on each tile [23]

This value is the sum of the amount of moves we made and the distance between us and the end point (see figure 4.6).

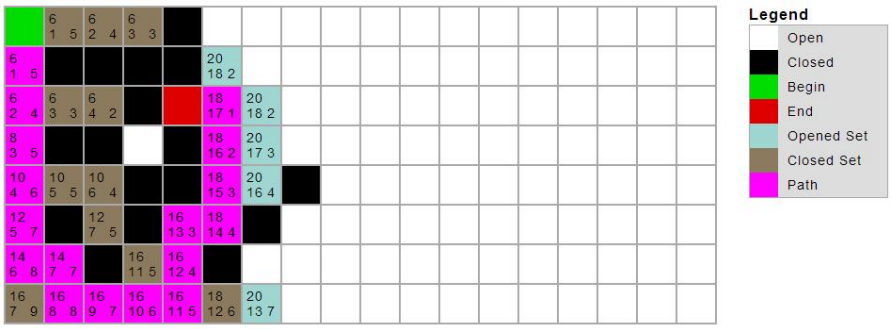


Figure 4.6: An example of a map showing the A* algorithm in action [23]

This algorithm analyses all possible movement points and gives them the previously mentioned values. As the algorithm moves one tile at a time in each possible direction, it determines which paths lead to a dead end, which paths lead to the end point and which are inefficient. While the algorithm itself is not the fastest one, it is a flexible, easy to implement algorithm which can be upgraded to fit a 3-dimensional plane without any problems. [24]

4.3 Arduino Uno Microcontroller

We decided on using an **Arduino Uno Microcontroller board** (see figure 4.7) for both navigation and bale approaching prototyping.

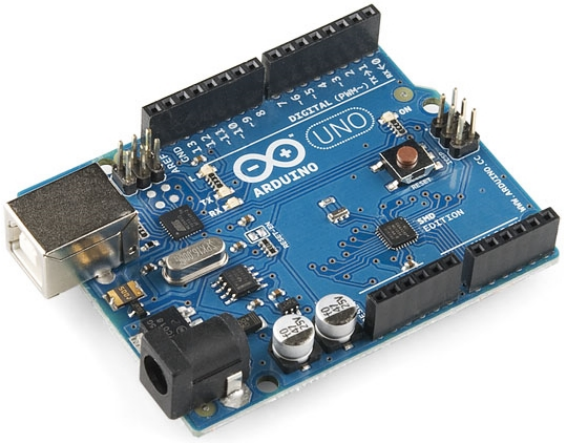


Figure 4.7: Arduino Uno Microcontroller. [25]

Specifications

The board is based on **ATmega328**. It has 14 digital I/O pins (out of which pins 3, 5, 6, 9, 10 and 11 can be used PWM - Pulse Width Modulation outputs) and other 6 analog inputs. Its operating voltage is 5 V. The recommended input voltage is 7 - 12 V. Every pin is allowed to get a maximum of 40 mA. There are two ways to

power the Arduino Uno Microcontroller; either via the USB connection, either with an external power supply, which could be a battery. Besides the I/O pins, there are other pins such as: VIN, 5V and GND. The VIN pin is the input voltage to the Arduino board when it is using an external power source; the voltage can be supplied through this pin or it can be accessed through it while supplying voltage via the power jack. The 5V pin outputs a regulated 5V from the regulator on the board. [26]

Besides the specifications listed above, a very handy feature of Arduino that we took advantage of is the **Serial Monitor**. We used it mostly for debugging, since it allows us to print variables' values on the screen in order to make sure tat they are what we expect. We also made sure we printed the name of the function which is being called, just to make sure that that happens and see whether the program gets stuck in a loop somewhere inside the program or if it runs the way it should.

4.4 4-Wheel Robotic Platform w/ DC Motors

In the building process, we used the **DFRobot 4WD Arduino-Compatible Platform** seen in Figure 4.8. This is a mobile robotic development platform which provides a 4 wheel drive system by using 4 DC Motors.

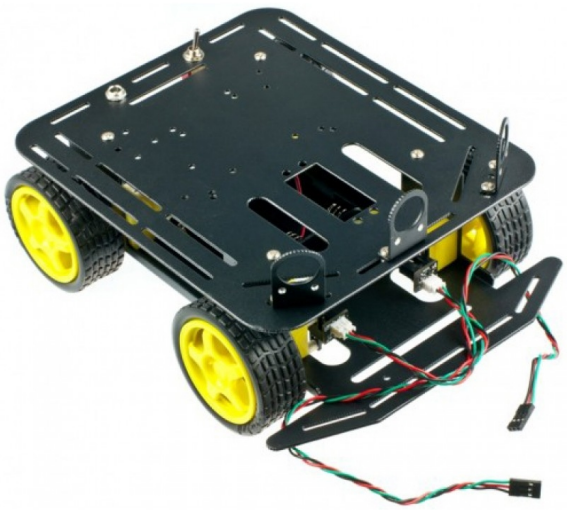


Figure 4.8: DFRobot 4WD Arduino-Compatible Platform. [27]

It is a convenient platform for our scope because of its' dimensions - 200mm x 170mm x 105mm and installation compatibility with other electronic components such as the Servo Motors, IR sensors and more. [28]

Each DC motor has a wheel attached to it. By running the left pair of wheels backwards while the right pair goes forwards, the rover turns left around its center. Similarly, the rover turns right if the left pair of wheels goes forwards while the right pair moves backwards. We also attached a servomotor in front of the rover,

centered in the middle. It is able to move at an 180° angle. On top of it, an IR-sensor is attached, which is used to measure distances. A buzzer and an LED are used as a warning for when the rover starts moving and when an object which is not a valid bale is detected. Also a PIR-sensor is attached to each corner to stop the rover if animals or humans are close by for safety reasons. Everything is connected to the Arduino Uno, which is programmed to take care of all operations without human intervention. The electronic diagrams of the system's parts can be seen in the Appendix.

4.5 Servo motor

To be able to read distances from the IR sensor in a range of 180°, we attached it to a servomotor. The servomotor will change the IR sensor's position, in order for us to be able to get more readings at certain angles - between 0 and 180°.

The specific type of servomotor that we used is the **SM-S2309S**, which can be seen in figure 4.9.

Figure 4.9: SM-S2309S Servo motor.

The working principle of servomotors is fairly simple. Generally, servomotors can be controlled by using PWM - Pulse Width Modulation, but Arduino's software makes it easier to control them since it comes with a special servo library. In this library, there is a *myServo.write()* function, which takes as a variable the angle at which the servo will be placed.

Wiring

The servo motor has 3 pins: **Sgn**, **VCC** and **GND** with which it is connected to the Arduino board. The Power pin is connected to the 5V pin, the Control pin is

connected to pin 9 and the Ground is connected to the GND pin. The connection is illustrated in the Appendix.

Programming

The first steps in our code regarding programming the servomotor were to include the *Servo library*, create a *servo object* and attach the servo to the pin to which its signal wire is connected by using the *myservo.attach()* function as seen in listing 4.1.

```
#include <Servo.h>
Servo myServo;  // create a servo object

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo
    object
}
```

Listing 4.1: Servomotor initialization

The code in listing 4.2 shows how we made the servo turn 180°. As it can be seen, the *angle* variable is set to a value of 1 before the *do while()* loop starts. Then, inside this loop, the *myServo.write()* function will tell the servo to go to the position stored in the *angle* variable, which is currently 1. Now, the *angle* will increase its value with 5 and so, when the *do while()* loop is being executed once more, the *angle* inside the *myServo.write()* function will be 6. When the *angle* variable will get to a value lower than 180, the loop will stop and the servo will have completed its 180° turn.

```
  angle=1;//IR start scaning at this angle
  do{
    myServo.write(angle);//move IR
    angle=angle+5;//set IR's pointing angle 5 degrees to the left
  }while (angle<180); //end of scanning
```

Listing 4.2: Servomotor 180° turn in step of 5°

4.6 SHARP IR Ranger Sensor GP2D12

An overview of the working concept of infrared proximity sensors has been given in the Preliminary Research chapter of the report. A specific type of infrared sensor that we used - **GP2D12** is illustrated in Figure 4.10.

Figure 4.10: SHARP IR Ranger Sensor GP2D12. [29]

Specifications

- Sensing range: 10-80 cm
- Supply voltage: 4.5-5.5 V
- Power consumption: 35 mA
- Operating temperature: -10-+60°C [30]

The specifications listed above prove that the GP2D12 IR Sensor is Arduino compatible. Besides the compatibility with the microprocessor, we also have to take into account some installation limitations. First of all, we have to make sure that the sensor is properly aligned with the surface of the object to which it is measuring the distance. Secondly, the surface of the object has to be non-reflective, otherwise the operation of the sensor may be affected.

Wiring

The connection with the Arduino microcontroller will be made using the sensor's 3 pins: **GND** - connected to Arduino's GND pin, **OUT** - connected to the analog pin A0 and **VCC** - connected to Arduino's 5V pin (See Appendix).

Programming

The first thing that we did was variable declaration. We declared the pin to which the IR sensor is connected - *IRpin* and the *distance* as seen in listing **??**.

```
1  int IRpin = 0; //analog pin for reading the IR sensor
   float distance;
```

Listing 4.3: Defining variables for the IR sensor

The *get_distance()* function is used to calculate the distance between the IR sensor and the bale. In this function, we used a *for* loop in order to get 10 readings from the sensor and then calculate the average of those, to make sure the value that we

end up with for the reading is as accurate as possible. The pin reading is done using the *analogRead()* function. The *distance* uses the value stored in the *sum* variable by the formula [31] in the code listing **??**.

```
void get_distance(){
  float sum = 0.00;
  for (int i=0; i<10; i++)//get 10 readings for higher precission
    {
      sum += analogRead(IRpin);
      delay(40);
    }
    sum=sum/10;
    distance=6787.00/(sum-3.00)-4;
    delay(100);
  }
```

Listing 4.4: Function used for measuring the distance

4.7 Motorcontroller

Alongside the GPS module and compass, we used a Polulu DRV8833 motorcontroller. The motorcontroller is used for directional control of the motors mounted in the chassis for the vehicle. The chassis used for prototyping the navigation system, uses tracks instead of wheels and is powered by two DC motors. Whereas the prototype for the bale approach system uses four DC motors, where the two motors on each side are connected in parallel. (See wiring diagram 9.1)

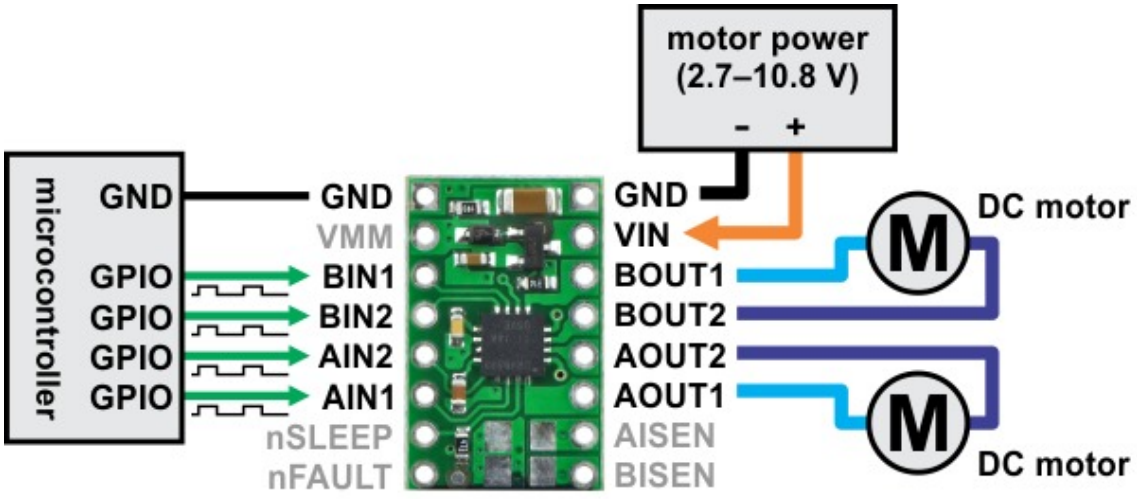


Figure 4.11: The connections for the DRV8833 [32]

The DRV8833 requires 5 connections to the microcontroller and uses a a separately connected battery to control the motors, this can be seen in figure 4.11. The microcontroller is connected to the DRV8833 to ensure that it operates at the correct logic levels given to it by the unit.

xIN1	xIN2	xOUT1	xOUT2	FUNCTION
0	0	Z	Z	Coast/fast decay
0	1	L	H	Reverse
1	0	H	L	Forward
1	1	L	L	Brake/slow decay

Table 4.1: H-Bridge Logic used to control the motor directions [33]

As seen in table 4.1 the DRV8833 uses the same logic as most H-bridges. Each motor is controlled by two separate GPIO pins and the direction is determined by what pin is high and the other low.
The motorcontroller also supports PWM, which allows the microcontroller to adjust the speed of which the motors are moving at. To use PWM the pins that are usually set to a logic level 1 are then instead pulse width modulated.

Programming and control

```
//The connected pins for the DRV8833 motorcontroller
int motor_pins[] = {8, 9, 10, 11};

//The truth table for the directional movement of the vehicle with the
    current connected pins.
//          Pin: 8  9  10 11
int forward[] = {0, 1, 0, 1};
int reverse[] = {1, 0, 1, 0};
int left[]    = {1, 0, 0, 1};
int right[]   = {0, 1, 1, 0};
int halt[]    = {0, 0, 0, 0,};
```

Listing 4.5: Definition of the motor states

To control the different possible directions of the motors, we defined their pin configurations as arrays, this means that depending on how the input and outputs are connected on the motorcontroller, the truth table will change. The purpose of the arrays is so that we can easily loop through in the *drive()* function shown in code listing 4.6.

```
void drive(int pin_array[], int dir_array[]){
  //Loops the passed array so set the drive functions state
  for(int i = 0; i < 4; i++){
    digitalWrite(pin_array[i], dir_array[i]);

      //Debug
      Serial.print("Pin num: ");Serial.print(pin_array[i]);Serial.print
    (" - ");Serial.print("Pin State: ");Serial.println(dir_array[i]);
    }
```

Listing 4.6: The function setting the direction

The *drive()* function takes in the *pin_array[]*, this array contains the pin connections to the xIN1 and xIN2 terminals on the motor controller and also the *dir_array[]* , which is the output figuration for a certain direction.

4.8 GPS Navigation

The main objective of the GPS navigation is to successfully navigate between coordinates from a database. The coordinates consist of all of the bale locations and also the coordinate for the drop-off point.

To create a prototype for the navigation system a HMC5883L Triple Axis Magnetometer Breakout and [insert name] GPS breakout board was used. It is necessary to keep track of the vehicles current heading with use of a magnetometer, to create a more efficient GPS navigational system. [34]

4.8.1 Magnetometer (Compass)

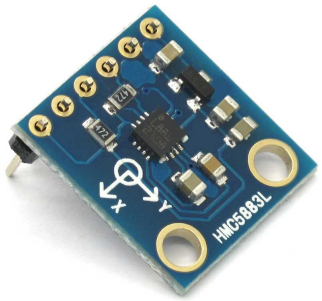


Figure 4.12: HMC5883L Magnetometer. [35]

The HMC5883L is communicated with through an I^2C interface and has an operating voltage in range of 3.3V-6V DC [36]. A wiring diagram for the magnetometer can be found in the appendix9.6.
Calibration of the HMC5883L is needed to receive accurate readings, and it must be re-calibrated each time the board is mounted. The calibration and processing of the magnetometer data is done using a library created specifically for the board [37]. The library handles the I^2C interaction and also the processing of the data from the x and y-axis, but omits the z-axis since it is not needed to find the current heading of the vehicle.

An important thing to note when dealing a compasses is that the heading points towards the magnetic north pole and not the true north pole. When calculating the heading for navigation system, the magnetic declination must be taken into account. The declination can be found at websites such as NOAA's, which is the National Centers for Environmental Information [38]. The library for the magnetometer returns the heading with a north based on the magnetic north and not the true north, therefore this must be taken into consideration when writing the navigation algorithm.

4.8.2 GPS module

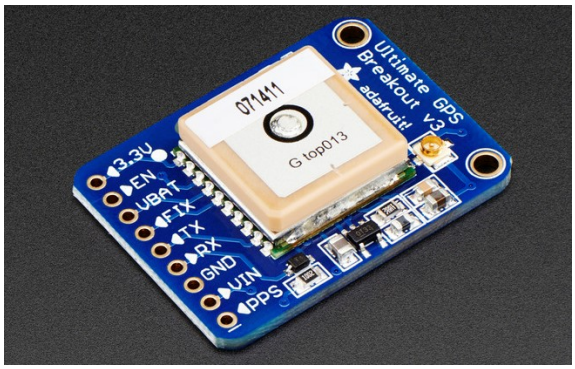


Figure 4.13: Adafruit Ultimate GPS Breakout. [39]

The GPS module used for this project is the Adafruit Ultimate GPS Breakout board (Version 3). It is built around the MTK3339 chipset and offers the ability to track up to 22 satellites on the 66 channels [40]. The voltage range for controlling the board is 3.3-5VDC. A physical status LED is also found on the board, indicating whether or not the module has gotten a GPS fix (a connection to the satellites). The LED will be blinking at 1Hz while searching for a fix and will then blink at 15 second intervals once it has established its connection. The module requires connections Vcc, GND and two additional pins for Tx and Rx, a wiring diagram can be found in the appendix 9.6.

NMEA is a protocol that has been created by the National Marine Electonics Association. This system was originally designed for use with boat navigation systems. GPS systems uses a subset or some modified parts of the NMEA protocol. One of the NMEA sentences that the module outputs will supply the navigation system with useful information when interpreted. An example of a NMEA sentence could be the following:

$$\$GPGGA, 161229.487, 3723.2475, N, 12158.3416, W, 1, 07, 1.0, 9.0, M, , , , 0000 * 18 \tag{4.1}$$

Name	Example	Decription
Message ID	$GPGGA	GGA protocol header
UTC Time	161229.487	hhmmss.sss
Latitude	3723.2475	ddmm.mmmm
N/S Indicator	N	N=north or S=south
Longitude	12158.3416	ddmm.mmmm
E/W Indicator	W	E=east or W=west

Table 4.2: Table showing the interpretation of the NMEA sentence. [41]

Table 4.2 shows only some of the information that is available from NMEA sentences, in this case the sentence is called GPGGA. The information found within GPGGA is the most relevant data for this project, since it contains some of the required date for successful GPS navigation. The latitude and longitude will be used to obtain the current position of the vehicle and will also be compared to its target position. The target position in this case will also be in the form of latitude and longitude. [42] [43]

```
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();
  //Will write the NMEA senteces to serial monitor
  //if (GPSECHO)
  // if (c) UDR0 = c;
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt
    somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
```

Listing 4.7: Setting up the GPS module

To interact with the GPS module we used the Adafruit GPS library [44]. The GPS library uses software serial, this allows the Arduino to serial communicate on other digital pins. Code listing 4.7 shows an interrupt vector named *TIMER0_COMPA_vect*, which is an 8-bit timer. Timer0 is by default set up to generate a millisecond interrupt and is also used for the *millis()* function, this is because it creates an interrupt by default when it overflows and a rate at almost 1KHz.
The interrupt is set up so that it generates an interrupt whenever then the timer counter is equal to the comparison register. In this case the comparison register is set to 0xAF. Whenever the interrupt occurs it stores the current GPS information, which then can be parsed by the library. This is a crucial part of the data gathering process, to ensure that the GPS information is kept up to date as often as possible.

```
// approximately every 2 seconds or so, grab the current GPS data
if (millis() - timer > 2000) {
  timer = millis();

  if (GPS.fix) {
    //Uses the GPS library to grab the latitude and longitude in
  Decimal Degrees
    //But only if there is a GPS fix
    currentLat = GPS.latitudeDegrees;
    currentLong = GPS.longitudeDegrees;
  }else{
    Serial.println("No GPS fix");
  }
}
```

Listing 4.8: Storing the latitude and longitude

The key feature of the library is that we are able to continuously and easily update our coordinates. As seen in code listing 4.8 the *GPS.latitudeDegrees* is a variable that is found within the GPS library, that is reassigned to the *currentLat* and *currentLong*. The library handles all of the extraction and conversion from the NMEA statements that the GPS receiver receives.

*$PGTOP,11,2*6E*
*$GPGGA,094738.000,5529.3973,N,00826.8020,E,1,09,1.03,14.1,M,44.2,M,,*53*
*$GPRMC,094738.000,A,5529.3973,N,00826.8020,E,0.01,221.78,141215,,,A*61*

Time: 9:47:38.0
Date: 14/12/2015
Fix: 1 quality: 1
Location: 5529.397460N, 826.802001E
Location (in decimal degrees)55.489955, 8.446701
Satellites: 9

Some sample data achieved from parsing the NMEA sentences received by the GPS module.

4.8.3 Navigation system

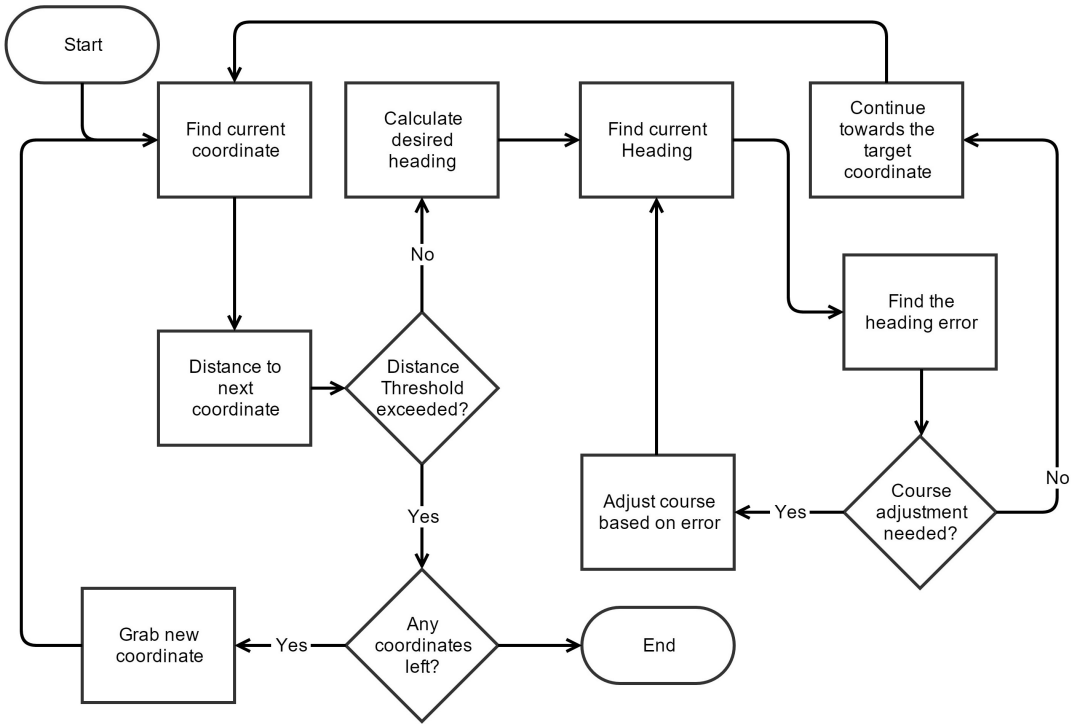


Figure 4.14: Flow chart describing the GPS process

Figure 4.14 displays a flow chart explaining the navigation algorithm for the autonomous bale collector. Since this part of the prototype only focuses on coordinate to coordinate navigation, it does not take bale processing and placement into account. The main purpose of this particular navigation system is to navigate between given coordinates, to show that it is possible to reach the target locations and successfully identify when there are no coordinates left to be navigated towards.

The navigation system starts of by finding its current location, this being the current GPS location, and then calculates the distance towards its target location. Since it requires a high amount of precision and more expensive equipment to reach an exact location with the vehicle, we decided to create a distance threshold, this being the distance between the current location of the vehicle and the target location. If the vehicle is sufficiently close to the target location, the navigation system will treat it as an arrival and continue with the next target coordinate.
But if the threshold is not exceeded and the vehicle still needs to navigations towards the target, the algorithm then calculates the optimal heading from its current location to the target location. This value is then compared to the current measured heading of the vehicle, using a compass. The heading error is then calculated and is then compared to a set value. This set value also acts as a threshold, but for the heading. The value allows the vehicle to be off its course by a certain amount

degrees before attempting to re-adjust it course. Once the error value exceeds its limit, the vehicle will then adjust its heading so that it corrects the course towards the target location.
Once the vehicle is on the correct course, the current location of the vehicle and distance towards the target location is updated. The previous mentioned calculations and comparisons are then continously performed until there are no target location left.

```
compass_heading();
heading += 1.96; //Adding the declination of Esbjerg, Denmark.
```

Listing 4.9: Calculating the heading between two coordinates

Since we are using library to calibrate and process the compass data, the current heading of the vehicle is easily obtained by calling the *compass_heading()* function. On the second line in code listing 4.9 we take the magnetic declination into account. [45]

```
float calculateHeading(float currentLat, float currentLong, float
    targetLat, float targetLong){

  //Calculating the x and y arguments needed for the atan2 trig
    function
  //The heading is based on the target coordinates and the current
    coordinates
  float x = sin(radians(targetLong-currentLong)) * cos(radians(
    targetLat));
  float y = cos(radians(currentLat)) * sin(radians(targetLat)) - (sin(
    radians(currentLat)) * cos(radians(targetLat)) * cos(radians(
    targetLong-currentLong)));
  float heading = atan2(x, y);
  //If the heading is negative it can't be used, this can be corrected
    by adding two pi.
  if (heading < 0.0){
    heading += 2*PI;
  }

  return degrees(heading);
}
```

Listing 4.10: Calculating the heading between two coordinates

To calculate the current heading of the vehicle, the formula found in equation 4.2 was used. [46]

$$\begin{aligned} \lambda &= Latitude \\ \phi &= Longitude \\ \Theta &= atan2(sin\Delta\lambda \cos\phi_2, cos\phi_1 sin\phi_2 - sin\phi_1 cos\phi_2 cos\Delta\lambda) \end{aligned} \tag{4.2}$$

The function *calculateHeading()* uses the current location, which is received from the GPS module, and the target location, which is defined in two arrays. Both the current and target locations are given in latitude and longitude, in the form of

decimal degrees with a 6 decimal precision. Calculating the heading is an essential part of the navigation system, without this function the vehicle is unable to figure out what direction to navigate towards. [47]

```
float calculateDistance(float currentLat, float currentLong, float
    targetLat, float targetLong) {
    // Uses the haversine formula to calculate the distance between two
     points.
    // The distance is returned in meters
    float a = sq((sin((radians(currentLat - targetLat)) / 2.0))) +
              (cos(radians(currentLat))*cos(radians(targetLat)) * sq((
    sin((radians(currentLong - targetLong)) / 2.0))));
    float distance = 2.0 * 6371000 * asin (sqrt(a));
    return distance;
};
```

Listing 4.11: Calculating the distance between two coordinates

To find the distance between the current location and the target location we used a formula called the "Haversine" distance formula, this can be found in equation 4.3.

$$\begin{aligned} &\lambda = Latitude \quad \phi = Longitude \quad R = 6371000(meter) \\ &a = sin^2\left(\frac{\Delta\phi}{2}\right) + cos\phi_1 cos\phi_2 sin^2\left(\frac{\Delta\lambda}{2}\right) \\ &c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \qquad d = R \cdot c \end{aligned} \tag{4.3}$$

The function *calculateDistance()* also is passed the coordinates at 6 decimals of precision, to ensure that there is the same amount of precision for computing all of the necessary calculations. Finding the distance between the current coordinate and the target coordinate is crucial for the navigation system, since this distance when compared to the threshold distance will let the vehicle know that it has arrived within an acceptable distance of its target location. [47]

```
  if(currentTarget == noOfCoords){
    Serial.println("No coordinates left in the array.");
    drive(motor_pins, halt);
    return;
  }
  if(distance < distanceThreshold){
    Serial.println("Arrived at the target location.");
    Serial.println("If there are additional coordinates it will
    continue");
    currentTarget++;

  }
```

Listing 4.12: Checking the distance Threshold

When the current distance is less than what the distance threshold is, the algorithm will increment the currentTarget so that a new location can be passed to the *calculateHeading()* and the *calculateDistance()* functions. Along side the two arrays an

integer is defined to be a stop number. Whenever the *CurrentTarget* integer has incremented enough to be equal to another integer named *noOfCoords* the vehicle will enter stop mode.

```
  float error = desiredHeading - heading;
2 if(error < -180) error += 360;
  if(error > 180) error -= 360;
```

Listing 4.13: Calculating the error

To calculate what the optimal direction of turning is, or in other words calculating how to adjust the vehicles heading to the desired heading, we used the statements found in listing 4.13. These three lines first calculates the angle difference between the two headings, it then normalizes them so that they are within a range of $-180, 180$. If the heading error is negative then the desired heading is achieved by turning left, or if positive it achieved by turning right instead.

```
  if(abs(error) <= headingThreshold){
2   Serial.println(" - On course");
    drive(motor_pins, forward);
4 }else if(error < 0){
    Serial.println(" - Adjusting towards the left");
6   drive(motor_pins, left);
    delay(50);
8   drive(motor_pins, halt);
  }else if (error > 0){
10  Serial.println(" - Adjusting towards the right");
    drive(motor_pins, right);
12  delay(50);
    drive(motor_pins, halt);
14 }else{
    drive(motor_pins, forward);
16 }
}
```

Listing 4.14: Adjusting the course based on the error value

When the vehicle is moving it has three possible directional choices, this is shown in code listing 4.14. If the vehicles current heading is within the threshold for the desired heading, it will continue moving straight forward until the heading difference exceeds the threshold. If the threshold has been exceeded, the vehicle has the choice of either turning to the left or right. As previously mentioned this is determined by the heading error, where a negative error results in a left turn. The current turning of the vehicle is done in short 50ms bursts, to ensure that the vehicle does not over-steer.

4.9 PIR Motion Sensor HC-SR501

A brief presentation on the working concept of such sensors has been given in the Safety chapter of this report. For human detection, we will be using 4 PIR sensors, which will be mounted on the chassis of the rover; 2 of them will be placed in the front and 2 of them in the back. The specific model of sensors we are using is the **HC-SR501** shown in Figure 4.15.

Figure 4.15: Front view of the HC-SR501 PIR Motion Sensor. [22]

Specifications

- Operating Voltage Range: 5-20V DC
- Output Level: High (3.3V)/Low (0V)
- Power Consumption: 0.65 mA
- Sensing range: <120° cone angle, within 7 meters
- Operating temperature: -15-+70°C [48]

The HC-SR501 PIR Motion Sensor has two extra useful features, which can be used by manipulating the two potentiometers shown in Figure **??**. Turning the delay potentiometer clockwise increases the sensing distance to a maximum of 7 meters, while turning it counterclockwise decreases the sensing distance to a maximum of 3 meters. Turning the delay potentiometer clockwise increases the time of delay to a maximum of 300 seconds, while turning it counterclockwise decreases the time of delay to a maximum of 0.5 seconds.

For our scope, we will set the distance to a maximum of 3 meters, since it is just a small-scale product and the delay to a maximum of 0.5 seconds to be able to avoid collision in due time.

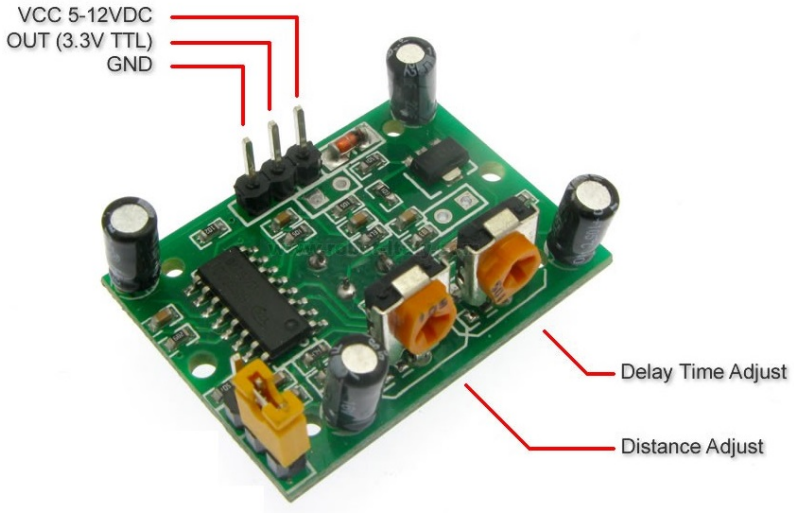


Figure 4.16: Back view of the HC-SR501 PIR Motion Sensor. [49]

Circuit analysis

Before physically placing the PIR sensors on our rover, we have made a series of calculations with regards to the electrical installment in order to avoid causing any damage to the components. Knowing that Arduino's input and output pins have a maximum of 200 mA current and that the 5V pin has a maximum output of 500 mA, we had to make sure to not exceed those values. Each HC-SR501 PIR Sensor will be connected to the 5V pin, which will create a parallel connection between all 4 sensors (See Figure 4.17).

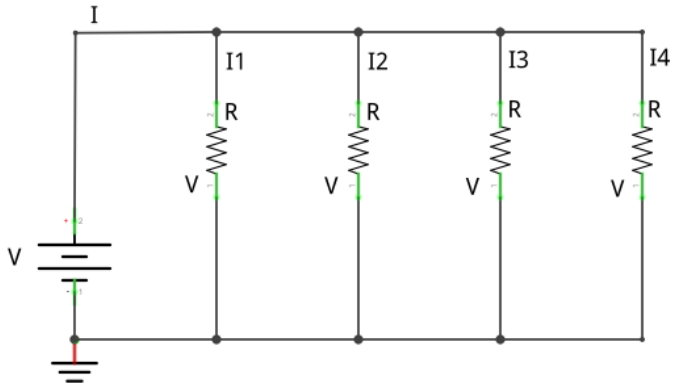


Figure 4.17: The PIR Motion Sensors' Electrical Circuit.

The HC-SR501 PIR Sensors' resistance varies from 100 kΩ when inactive and 7813 Ω when active. Therefore, using our previous acquired knowledge on electrical circuits, we have made calculations for both cases and ended up with the results listed in Table 4.3.

	Inactive PIR Sensors	**Active PIR Sensors**
V	5 V	
R	100 kΩ	7813 Ω
R_{eq}	$\frac{R}{4} = 25$ kΩ	$\frac{R}{4} = 1953.25$ kΩ
I	$\frac{V}{R_{eq}} = 0.2\ mA$	$\frac{V}{R_{eq}} = 2.56\ mA$
$I_1 = I_2 = I_3 = I_4$	$\frac{I}{4} = 0.05\ mA$	$\frac{I}{4} = 0.64\ mA$

Table 4.3: The circuit's calculations and results.

All the results above show a match between the HC-SR501 PIR Sensor's specifications and Arduino's. Hence, it is impossible to cause damage to either of the components.

Placement

Being the component which ensures the safety related aspects of the ABC prototype, the HC-SR501 PIR Motion Sensors' placement has to be decided carefully. To come up with a final solution, we took into account the the sensors' sensing range and both width and height of the rover. Figure 4.18 below gives a good visual understanding regarding a sensor's range (120° cone angle) while monitoring its' surrounding.

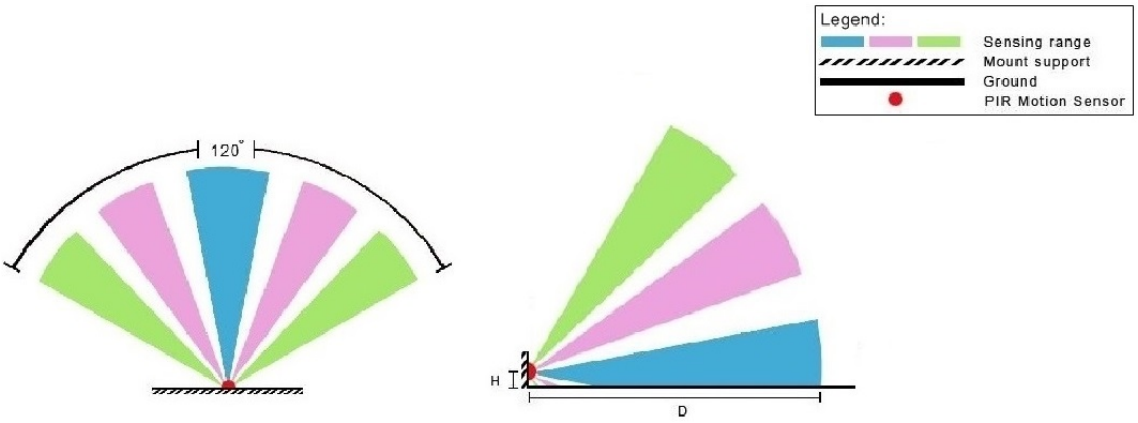


Figure 4.18: Top and side view of the HC-SR501 PIR Motion Sensor's range.

Figure 4.18 also displays the sensing range of the HC-SR501, but this time with consideration to H - the height at which each sensor will be mounted. The value of H will be larger than 5.5 cm, which is the actual height of the rover we are using. As mentioned before, D - the sensing distance will be set to a maximum of 3 meters.

The whole final safety system setup, which uses 4 HC-SR501 PIR Motion Sensors is displayed in Figure 4.19. Depending on the value of W - the width of the rover's

chassis, both front and back pair of sensors will be tilted with a certain angle to the exterior of the rover in order to increase the detection area on the sides.

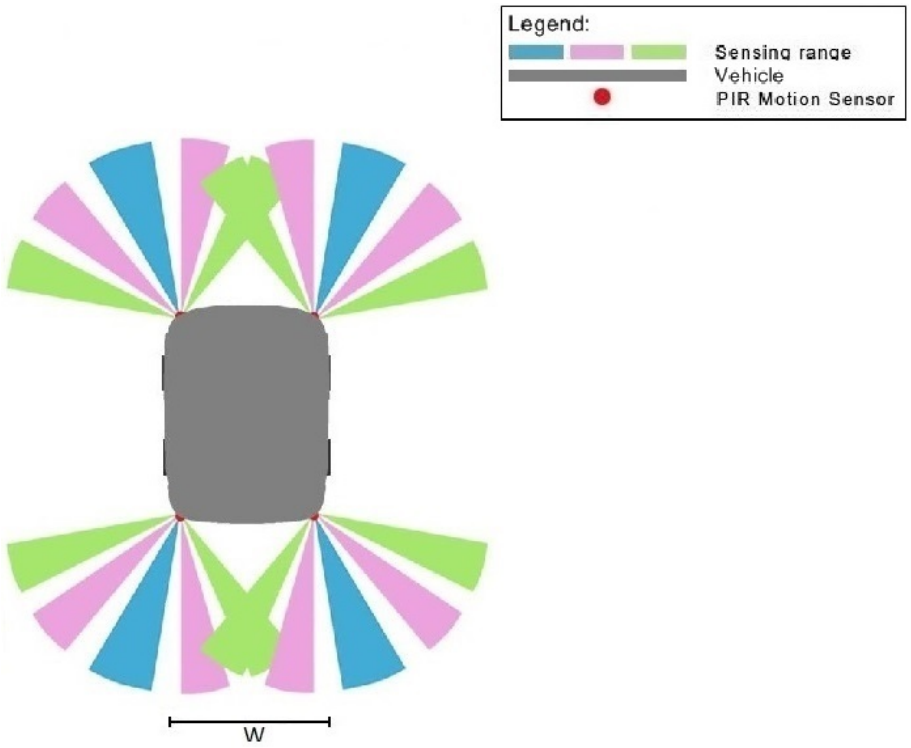


Figure 4.19: Top view of the HC-SR501 PIR Motion Sensors' placement.

Note! In our case, D is 20 times larger than W and 50 times larger than H (with approximation). Therefore, it is obvious that the proportions of the measurements in Figure 4.19 are not right, but they still prove the point we are making and help with visualizing the entire setup.

Wiring

The connection with the Arduino microcontroller will be made using the 3 pins of each sensor: **GND**, **TTL** and **VCC** (See Figure 4.16). GND will be connected to Arduino's GND pin, VCC will be connected to Arduino's 5V pin and TTL will be connected to a digital pin. The connection of all 4 PIR sensors can be seen in Appendix.

Programming

We started by declaring the pins connected to the PIR sensors, by using the *pir_pins[]* array. Since the sensors need between 10 and 60 seconds of calibration according to the datasheet, we also declared the *calibration_time* variable for the time calibration and stored the value 10 in it, which will be used further in the code - see listing 4.15.

The next step was to set each pin as an input by using the *pinMode*() function and to also set the pins to LOW (no movement - LOW; movement - HIGH) by using the *digitalWrite*() function. Then, we used 2 interrupts because that's how many the Arduino Uno board supports for 2 of the pins connected to the PIR sensors, those being pins 2 and 3. The function used for creating the interrupts is the *attachInterrupt()* which has as its first parameter is the pin number. Normally, you use *digitalPinToInterrupt(pin)*, however, older sketches often have direct interrupt numbers. For example, Arduino Uno has number 0 (for pin 2) or number 1 (for pin 3).

```
int calibration_time=10; //the time we give the sensor to calibrate
  (10-60 secs according to the datasheet)

int pir_pins[]={3,7,2,8}; //digital pins connected to the PIR sensors
  ' outputs

for(int i=0;i<4;i++) {
  pinMode(pir_pins[i],INPUT);
  digitalWrite(pir_pins[i],LOW);
}

attachInterrupt(0, warning, RISING);
attachInterrupt(1, warning, RISING);
```

Listing 4.15: Defining variables for the PIR sensors' setup

The second parameter is the function to call when the interrupt occurs , which in our case is *warning()*. The role of this function is to start the alarm when any of the 2 pins go from LOW to HIGH and to also stop the vehicle. This function is shown in listing 4.16. The last parameter of the *attachInterrupt()* is the *mode*, which defines when the interrupt should be triggered. We used *RISING* in order to trigger when the pin goes from LOW to HIGH. Other modes available on the Arduino Uno board are: LOW, CHANGE and FALLING. [50]

```
void warning() {
    Serial.println("-----------------------------------");
    Serial.println("function warning started");
    start_alarm(3000);
    stop_robot();
}
```

Listing 4.16: Function to start alarm and stop the rover

We used the Serial Monitor to be able to see how the program was running, therefore we needed to see when the calibration of the sensors was finished. This is shown in the code listing 4.17.

```
Serial.begin(9600);          // start the serial port

Serial.println("-----------------------------------------------");
Serial.println("          Hale bale collector                  ");
Serial.println("-----------------------------------------------");
Serial.println("           Calibrating sensors                 ");
```

```
7   for(int i = 0; i < calibration_time; i++){
      Serial.print(".");
9     delay(1000);
    }
11  Serial.println("          Calibration is done          ");
    Serial.println("        PIR sensor are  active         ");
```

Listing 4.17: Serial Monitor of the PIR sensors' setup

4.10 Bale approach process description

When moving the servomotor from 0 to 180° in small turns, the nearest object can be detected. By moving the sensor to the closest point of the object and moving the sensor right until it shoots past the object distance and angle to the right edge is obtained. Similarly, by moving the sensor to the left side to get the angle and distance of the left edge. The size of the object can now be calculated and if the size is valid, the travel path will be calculated. See figure 4.20 and 4.21.

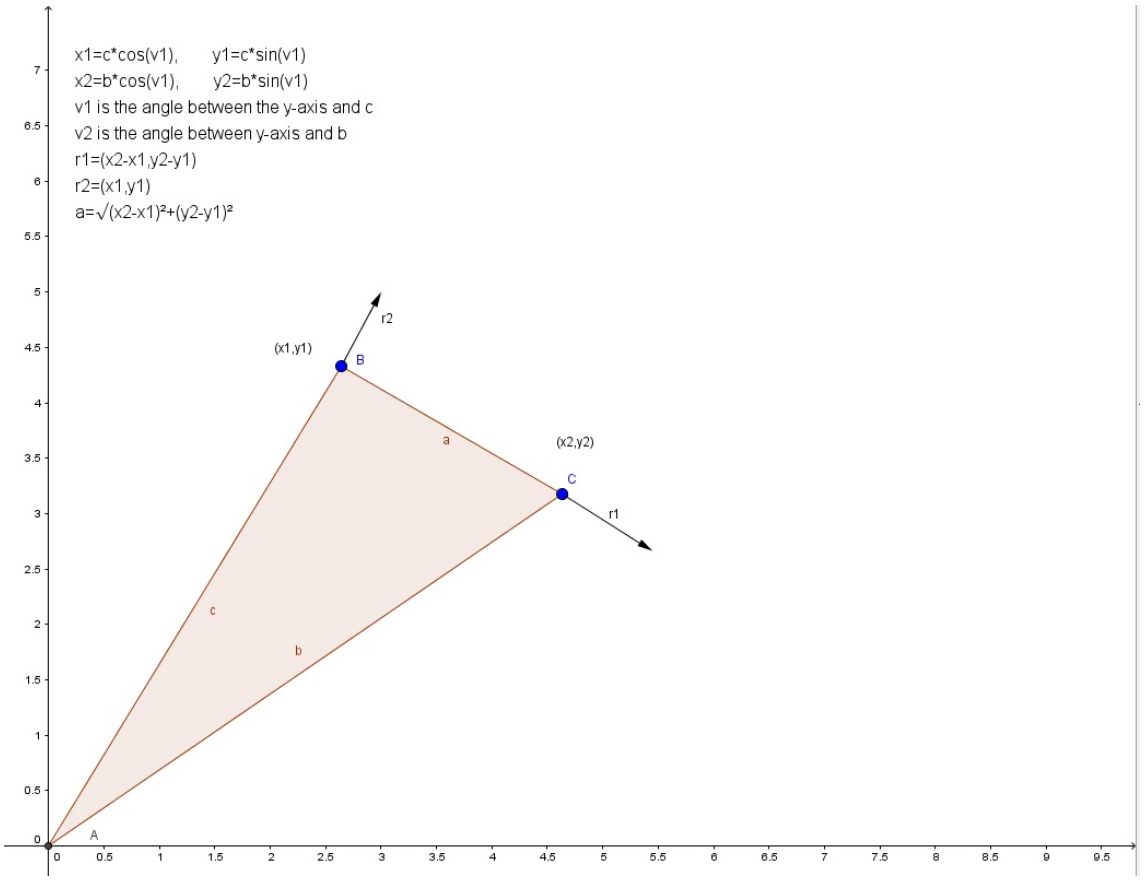


Figure 4.20: A laser scanning the bale width.

Point A represents the position of the laser and points B and C represent the edges of a bale. So, by using these equations, balewidth would equal the length of a. The

first edge of the bale would be at point (x_1, y_1) and the second one at (x_2, y_2).

For the rover to get in front of the small side of the bale, it has to move the length of a in the direction of $\vec{r_1}$, turn 90° to the left and then move the length of c in the direction of $\vec{r_2}$ and turn 90° to the left. See figure 4.21.

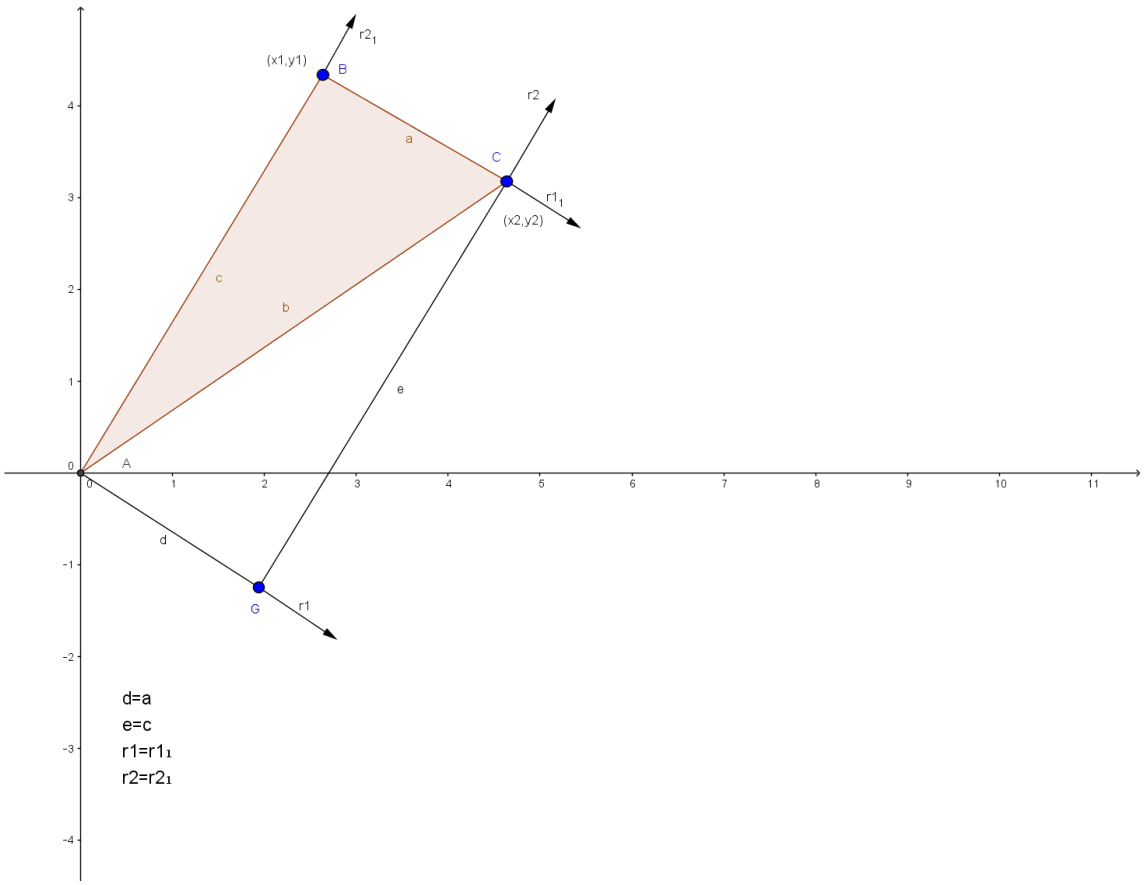


Figure 4.21: The path to get in front of the small side of the bale

Programming

We have "translated" all the math above into code. Outside Arduino's predefined functions - the *setup()* and *loop()*, we declared all the variables that the code will use. Each variable is declared accordingly to which kind of data it will have to store (*int*, *float*). The scope for which all the variables are being used is described in the comments of the listing **??**.

```
unsigned long m_sec;
int IRpin = 0; //analog pin for reading the IR sensor
float centimeter;
float distance;
int angle;   // variable to hold the angle for the servo motor
int closest_angle; //laser angle to closest object
long closest_dist;//distance to closest object
float dist1;//distance to first edge on the bale
float dist2;//distance to second edge on bale
```

```
float bale_width;
int angle1;//store the angle  between bales first edge and the laser
int angle2;//store the angle  between bales second edge and the laser
float degree_left;//time in miliseconds to turn servo motor 1 degree to
    the left
float degree_right;//time in miliseconds to turn servo motor 1 degree
   to the right
```

Listing 4.18: Variable declaration

For the motor movement, we had to assign values to the *centimeter*, *degree_left* and *degree_right*, values which we decided on based on practice. We have input some random values at first and analyzed what the robot does - how much it moves forward, how much it turns left/right. The values stored by each variable can be seen in listing **??**.

```
  centimeter=500/30;//time in milliseconds to drive one centimeter
  degree_left=1950.00/360.00;// time in millideconds to turn one degree
    to the left
  degree_right=2050.00/360.00;//time in milliseconds to turn one degree
    to the right
  Serial.begin(9600);         // start the serial port
```

Listing 4.19: Variable value assignment for the servo's movement

Inside the *loop()* function, we have used *Serial.print()* to print function calling and variable values on the Serialmonitor. Also, we have called the *find_closest_object()*, *calc_width()* functions - see listing **??**, functions which will be discussed further on.

```
void loop() {
  Serial.println("-----------------------------------");
  Serial.println("function get_distance started");
  find_closest_object();
  calc_width();
  Serial.println("-----------------------------------");
  Serial.println("function calc_width started");
  Serial.print("Bale width: ");
  Serial.print(bale_width);
  Serial.println(" cm");
}
```

Listing 4.20: Arduino's *loop()* function

The first function being called in the *loop()* is the *find_closest_object()* function, which, as its name suggests, has the scope of measuring distances at an 180° angle and decide how far away the nearest object to the rover is. The whole function is illustrated in listing 4.21. The role of *do while()* loop is to turn the servo motor, therefore the IR sensor attached to it as well, with one degree after each distance reading. In the same loop, the value of the *distance* variable, resulted from calling the *get_distance()* function will be compared every time with the previous one. In case the value is smaller then the one before, the value of *distance* will now be stored in the *closest_dist* variable and the value of *angle* will be stored in the *closest_angle* variable.

```
void find_closest_object(){
  Serial.println("-----------------------------------");
  Serial.println("function find_closest_object started");
  int i;
  closest_dist=1000;//initial value to prevent the program to end the
   loop emidiatly
  angle=1;//laser start scaning at this angle
  i=0;
  do{
    myServo.write(angle);//move laser
    get_distance();
    if(distance<closest_dist){//compare readings to the closest object
   so far
      closest_dist=distance;//new closest object
      closest_angle=angle;//new closest angle to object
    }
    angle=angle+5;//set laser's pointing angle 5 degrees to the left
  }while (angle<180); //end of scanning
}
```

Listing 4.21: Function deciding the nearest object to the rover

Moving on to calculating the actual width, we have created the *calc_width()* function - see listing 4.22, which first calls *get_dist1()*, which returns the value of *dist1* - the distance to the first edge of the bale and *get_dist2()*, which returns the value of *dist2* - the distance to the second edge of the bale. The value of these two variables will be used to finally calculate the width of the bale by using the math described in this section.

```
void calc_width(){
  double cos_v;//cos(v) in degrees
  double angle_tri;//the angle between the two edges of the bale
  get_dist1();//get distance to first edge
  get_dist2();//get distance to second edge
  angle_tri=(angle1-angle2);//the angle between the edges
  cos_v=cos(angle_tri/360*2*3.14);//cos(v) in degrees
  bale_width=sqrt(pow(dist1,2)+pow(dist2,2)-(2*dist1*dist2*cos_v));//
   cos-relation
}
```

Listing 4.22: Function returning the width of the bale

get_dist1() and *get_dist2()* are two very similar functions. They do the same thing, but mirrored. The main difference between them can be seen in line 12 from listing 4.23 and line 11 from 4.24. In the *get_dist1* function, the angle increases with one degree, making both the servo and the IR sensor turn to the right side of the bale, while in the *get_dist2()* function the angle decreases with one degree, resulting in a turn to the left side of the bale.

```
void get_dist1(){//same as get_dist2, just for the right edge of the
   bale
    Serial.println("-----------------------------------");
    Serial.println("function get_dist1 started");
    int x=0;
```

```
    angle=closest_angle;
    angle1=angle;
    myServo.write(angle);
    delay(300);
    get_distance();
    dist1=distance;
    do{
      angle=angle+1;
      x++;
      myServo.write(angle);
      delay(100);
      get_distance();
        if (distance>dist1+2){//if there is a big difference the IR-
  sensor is no longer pointing at the same object
          if (x<3){//be certain to get a valid reading
            angle=closest_angle-5;
          }else{
            angle=180;//to exit loop
          }
        }else{
          dist1=distance;//distance measurement
          angle1=angle;//to get angle between laser and the second edge
   of the bale
        }
      }while(angle<180);
      Serial.print("dist1 meassured is: ");
      Serial.print(dist1);
      Serial.println(" cm");
 }
```

Listing 4.23: Function returning the distance to the first edge of the bale

```
void get_dist2(){//dist2 is the distance to the left corner of the
  bale
    Serial.println("-------------------------------");
    Serial.println("function get_dist2 started");
    angle=closest_angle;
    angle2=angle;
    myServo.write(angle);//point the IR-sensor at closest object
    delay(300);
    get_distance();
    dist2=distance;//distance to object's second edge
    do{
      angle=angle-1;
      myServo.write(angle);//turns the servo-motor to the left
      delay(100);
      get_distance();
      if (distance>dist2+2){//if there is a big difference the sensor
  is no longer pointing at the same object
        angle=0;//to exit loop
      }else{
        dist2=distance;
        angle2=angle;//to get angle between servo-motor and the second
  edge of the bale
      }
    }while(angle>0);
```

```
22        Serial.print("dist2 measured is: ");
          Serial.print(dist2);
24        Serial.println(" cm");
      }
```

Listing 4.24: Function returning the distance to the second edge of the bale

Lastly, the function which is in charge of moving the rover by taking into consideration the width of the bale that we had just calculated using all the other functions discussed above, is the *approach_bale()* one. This function calls the functions in charge of the movement - *turn_left()*, *turn_right()*, *drive_forward()*, *stop_robot()* in a certain defined sequence and at a certain angle or for a certain distance. The sequence in which this functions are called can be seen in listing 4.25. The result of calling these functions is as follows: the rover will first make turn to the left so it is in parallel with the bale; then, it will drive forward for the sum of the bale width and the half the rover's width. After this, it will make a 90° turn to the right, it will drive forward for the distance that initially was between the rover and the bale, using polar coordinates, and finally make another 90° turn to the right and find itself in front of the bale. In between all these movement, we called the *stop_robot()* function, so that the robot stops and we are able to see the sequence of movements clearly.

```
void aproach_bale(float _vec_angle,float x,float y){//x and y compared
    to the direction of the servo-motor
2   Serial.println("-----------------------------");
    Serial.println("function approch_bale started");
4   turn_left((_vec_angle)*degree_left);
    stop_robot();
6   delay(1000);
    drive_forward(x*centimeter);
8   stop_robot();
    delay(1000);
10  turn_right(90*degree_right);
    stop_robot();
12  delay(1000);
    drive_forward(y*centimeter);
14  stop_robot();
    delay(1000);
16  turn_right(90*degree_right);
    stop_robot();
18 }
```

Listing 4.25: Function in charge of the rover's movements to approach the bale

4.11 Computer Vision

In this project, we decided to use computer vision and a webcam to recognise and calculate the angle of the bale. To do this, we wrote the code discussed in this section. The program first looks for a yellow object (a sponge during the testing) and checks if it is placed within the desired distance. If that is true, the program starts looking for a blue object instead, which is a blue tape we put on on the of the corners of the sponge. This is necessary to determine the angle of the object, since the tape is mostly a two-dimensional object, whereas a three-dimensional sponge would make this process difficult.

The main part of the program mainly consists of two functions. The first one loads the image from the webcam and stores it into a matrix, then processes it using given values to detect desired colour. The second part uses that image to detect bale's position.

The first lines of the code, as usual, are used to load necessary libraries and declare global variables we will be using. Most of these variables will be explained later in the section.

```
VideoCapture cap(0); //capture the video from web cam
if(!cap.isOpened()){// if not success, exit program
  cout << "Cannot open the web cam" << endl;
  return -1;
}
  while (true){
      bool bSuccess = cap.read(imgOriginal); // read a new frame from
   video
      if (!bSuccess){ //if not success, break loop
          cout << "Cannot read a frame from video stream" << endl;
          break;
      }
```

Listing 4.26: Reading from webcam

Going into the main function, we first request webcam content to be stored into a variable called 'cap'. Then, we jump into the main loop, which contains the rest of the program's functions and store the contents of 'cap' into a matrix called 'imgOriginal', so that we can process it later. See code 4.26.

```
      if(detEdge == 1){//blue line
          iLowH = 101;
          iHighH = 129;
          iLowS = 93;
          iHighS = 255;
          iLowV = 77;
          iHighV = 255;
      }else{
          iLowH = 0;//sponge
          iHighH = 56;
          iLowS = 77;
          iHighS = 255;
          iLowV = 100;
```

```
14              iHighV = 255;
            }
```

Listing 4.27: Assigning HSV values

Moving on, based on the value of variable 'detEdge', which tells the program whether we are looking for a yellow colour object or a blue one, we assign values to 6 variables. See code in 4.27.

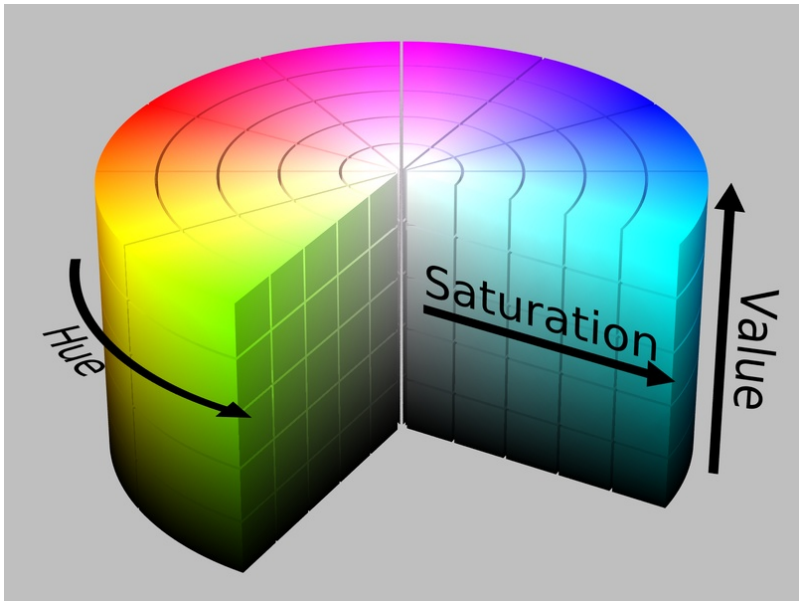


Figure 4.22: HSV spectrum explanation

These variables determine in what range of HSV spectrum we are working. HSV stands for **Hue**, **Saturation** and **Value**. The roles of these values can be seen in figure 4.22.

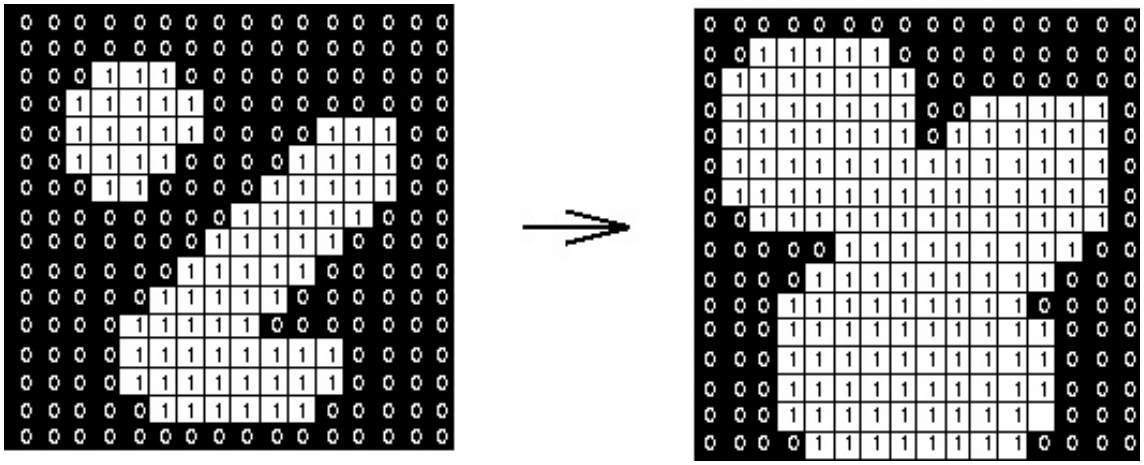


Figure 4.23: Mathematical morphology

We then create another matrix called 'imgHSV'. We convert the matrix with the webcam image from RBG spectrum to HSV spectrum and store it in the new matrix.

We then threshold this image based on the previously assigned HSV values. In other words, we turn the whole image into a black screen and then turn some pixels wide based on their intensity, which is determined by the HSV values. Afterwards, we perform mathematical morphology (dilation and erosion) on the thresholded image to remove small objects as well as fill in small gaps inside the objects. An example of mathematical morphology can be seen in figure 4.23.

```
1       Mat imgHSV;
        cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the
    captured frame from BGR to HSV
3       inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH,
    iHighS, iHighV), imgThresholded); //Threshold the image
        //morphological opening (remove small objects from the
    foreground)
5       erode(imgThresholded, imgThresholded, getStructuringElement(
    MORPH_ELLIPSE, Size(5, 5)));
        dilate( imgThresholded, imgThresholded, getStructuringElement(
    MORPH_ELLIPSE, Size(5, 5)));
7       //morphological closing (fill small holes in the foreground)
        dilate( imgThresholded, imgThresholded, getStructuringElement(
    MORPH_ELLIPSE, Size(5, 5)));
9       erode(imgThresholded, imgThresholded, getStructuringElement(
    MORPH_ELLIPSE, Size(5, 5)));
```

Listing 4.28: Thresholding the image

This means that we first remove 5x5 squares of white pixels in the image to remove small objects, then turn 5x5 squares of black pixels into white ones around the existing white ones to fill small gaps inside objects. See the full code of these operations in 4.28.

```
  Mat threshold_output;
2 vector<vector<Point> > contours;
  vector<Vec4i> hierarchy;
4 // Detect edges using Threshold
  threshold( imgThresholded, threshold_output, thresh, 255,
   THRESH_BINARY );
6 // Find contours
  findContours( threshold_output, contours, hierarchy, CV_RETR_TREE,
   CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

Listing 4.29: Finding contours

Moving on, we go into a function called 'thresh_callback', which is constantly being called in the previously mentioned loop. Here, we create a new matrix called 'threshold_output', which we use to find the contours for the objects, or separate clusters of white pixels. See the code in 4.29.

```
1 // Find the rotated rectangles for each contour
  vector<RotatedRect> minRect( contours.size());
3 for(int i = 0; i < contours.size(); i++){
    minRect[i] = minAreaRect(Mat(contours[i]));
5   if(minRect[i].size.height * minRect[i].size.width > maxH * maxW){//
  get the biggest object
      maxH = minRect[i].size.height;
```

```
7         maxW = minRect[i].size.width;
          maxAngle = minRect[i].angle;
9         maxCX = minRect[i].center.x;
          maxCY = minRect[i].center.y;
11      }
    }
```

Listing 4.30: Finding largest rotated rectangle

We then run a loop for each contour and find the rotated rectangles for them. These rotated rectangles are vectors which contain 3 attributes - x and y coordinates of the rectangle's centre, height and width of the rectangle and the tilt angle. We multiply the height and width of each rectangle to get its area and then find the largest one, assigning its attribute values to the variables we will later use. See the code 4.30.

With this process of elimination we determine which object within the desired HSV spectrum is the largest, since there is a possibility of other, smaller objects of similar colour being detected and interfering. Therefore, we make sure that the object we are working with is in fact the right one.

```
1     cout << maxH << " X " << maxW << endl;
      cout << maxCX << "x " << maxCY << "y" << endl;
3     if(detEdge == 1){
        cout << maxAngle << endl;
5     }
      if((maxCY > centerH - errorCH) && (maxCY < centerH + errorCH)){
7       if((maxCX > centerW - errorCW) && (maxCX < centerW + errorCW)){
          if(maxH < targetH - errorH){
9           detEdge = 0;
            swapTarget();
11          cout << "Object too far away, move closer" << endl;
          }else if(maxH > targetH + errorH){
13          detEdge = 0;
            swapTarget();
15          cout << "Object too close, move away" << endl;
          }else{
17          detEdge = 1;
            swapTarget();
19        }
          if(detEdge == 1){
21          if((maxAngle > -0.5) || (maxAngle < -89.5)){
              cout << "OK" << endl;
23          }else{
              cout << "Bad angle, try repositioning" << endl;
25          }
          }
27      }else{
          cout << "Center X coords are off" << endl;
29        detEdge = 0;
          swapTarget();
31      }
      }else{
33      cout << "Center Y coords are off" << endl;
        detEdge = 0;
35      swapTarget();
```

```
    }
```

Listing 4.31: Calculating object's position

Finally, we come to the most important part of the code where we finally determine the position and angle of the object using previously gathered values. For purposes of clarity, we print out the height, width and the x and y coordinates of the largest rotated rectangle, as well as the angle if we are looking for a blue line instead. We first run a few checks to see how close the object is to the centre on the screen and if it is not within the allowed range, the program informs the user that it is off centre and waits for repositioning. We then perform a few 'if' cases to calculate if the object is within desired distance. To do this, we compare the object's height with the fixed number +/- a few more pixels to increase the target area. If the object is not within these numbers, we inform the user that the object is either too close or too far from the camera. We also force the program to switch back to the detection of yellow colour as well as switch the target area, in case it was looking for a blue colour instead. This is needed to determine if the main object is still within the desired distance if we start moving the camera during the determination of the angle later on. If these two previously mentioned 'if' cases fail, the program confirms that the object is in the target area by switching to detection of a blue colour instead. In that case, we then check if the angle is within the desired range, where the angle ranges from 0 to -90 and the closer it is to either of the numbers, the more horizontal or vertical the object is. If the angle is not within the range, the user is instructed that repositioning is needed, otherwise, the program tells the user that the object is ready for pickup. See the 4.31 code.

```
void swapTarget(){
  if(detEdge == 0){//target area for yellow object
    targetH = 150;
    errorH = 7;
    centerH = 240;
    errorCH = 30;
  }else{//target area for blue object
    targetH = 25;
    errorH = 3;
    centerH = 200;
    errorCH = 15;
  }
}
```

Listing 4.32: Swapping target area values

A small part of the code worth mentioning is the function called 'swapTarget' which swaps between the target area values for the yellow and blue object. This is vital to the program due to the fact that those two objects have completely different heights, which are used to determine the distance from camera to the object. See the code 4.32.

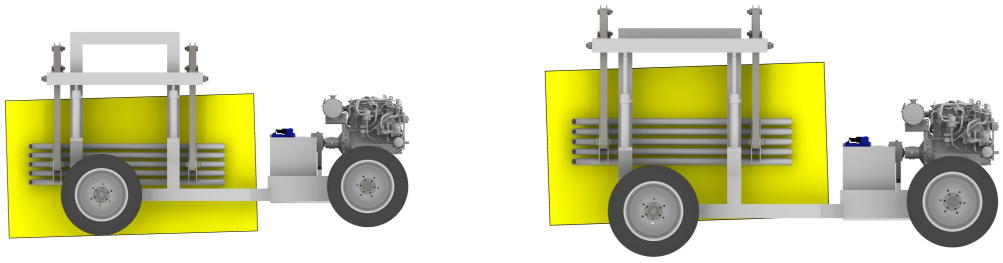
4.12 Mechanical design

In this chapter we are going into detail of the mechanical design that was made by the mechanical group. They designed the mechanical parts for a vehicle capable of collecting a single bale at a time. Both the powering and steering of the wheels is rear-ended. The vehicle has a total weight of 1600 kg, but with a safety factor added on top it is estimated to weigh around 2100 kg. The actual bale weighs around 900 kg so in total the vehicle will weigh 3000 kg when carrying a bale. Their final design has taken all of our sensor placements into account. The two images in figure 4.24 shows how the clamp in an open and closed position, it needs 26kN of force when clamping the bale to be able to lift it of the ground.

(a) Clamp in open position **(b)** Clamp in closed position

Figure 4.24: Clamping of the bale

Besides the clamping motion, the vehicle also has a mechanical lifting movement, this can be seen in figure 4.25. The bale is lifted 30 cm from the ground, before the vehicle begins to move and is then driven to the designated drop-off location.



(a) Bale in down position **(b)** Bale in up position

Figure 4.25: Lifting of the bale

Figure 4.26 shows the full frame picture of the vehicle. The mechanical group did not need to build a prototype, their work was all theoretical and could be simulated

using their CAD tools. But from what they have gathered and researched then this is the best suitable design for an autonomous bale collector. During their design process, we collaborated with them and gave them insight into requirements and optimal sensors placements.

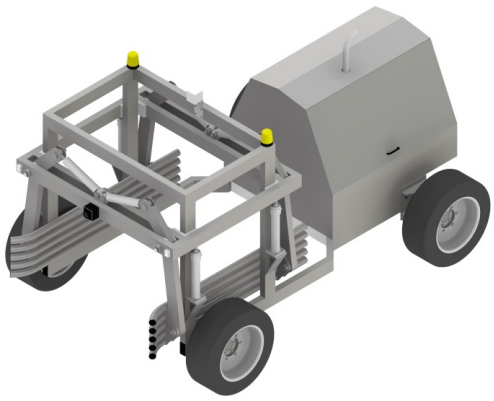


Figure 4.26: The final mechanical design

Braking safety

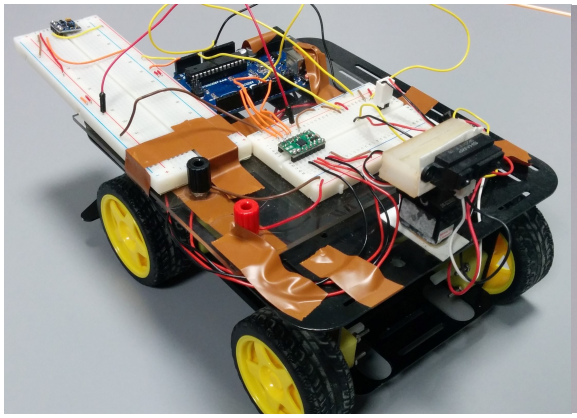
For safety reasons we need to calculate the braking distance of the vehicle. Our sensors will need to extend further than the minimum safety factor. In equation 4.4 v stands for the speed, g is the force of gravity, F is the friction of the road and G is angle of the road. The vehicles max speed is 19 km/h, that is 5.28 m/s. The friction on the tractor tires on a field is given to be 0.6 [51] [52]

$$\begin{aligned} d &= \frac{v^2}{2 \cdot g(F+G)} \\ d &= \frac{5.28\frac{m}{s^2}}{2 \cdot 9.8\frac{m}{s^2}(0.6+0)} \\ d &= \frac{27.8\frac{m}{s}^2}{11.76\frac{m}{s^2}} \\ d &= 2.37m \end{aligned} \tag{4.4}$$

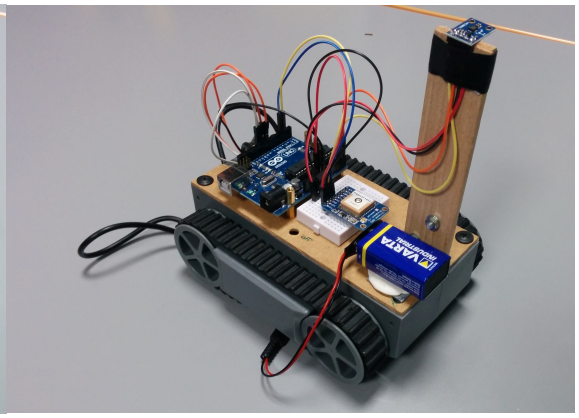
The result is then that it takes the vehicle around 2.4 m to brake to a complete stop. For safety reasons we add a safety factor of 2. This mean that the sensors will have a range greater than or equal to 4.8m, whilst sensing the area at the front of the vehicle. The PIR sensors used in our prototype has the capabilities of ranging up to 7m. [48]

Chapter 5

Testing



(a) Bale Approach Prototype



(b) Navigation Prototype

5.1 Bale Approach Prototype

To obtain an objects position, we planned on using a laser.

Early tests showed that this solution might work in practice. However, when working on the coding for Arduino, the laser stopped working properly. A substitute for it had to be found fast and for a small amount of money. The choice was an IR-sensor with a max. range of 150 cm and min range of 20 cm. Therefore, the vehicles working area had to be scaled down since the laser had a max range of 40 meters and no min. range. After some changes in the programming, early tests showed that it was not as precise as the laser especially at the edges of an object where it was difficult to tell if the sensor still measured the width or it had reached the other side of the bale.

Also, it had difficulties in measuring sharp angles since the beam would be reflected away from the receiver. To avoid those inaccurate readings from the edges, the vehicle should move to a better position in front of the bale, scan again, and then use readings on two points as a vector in the direction of the bale. Even so, the first

bad readings sometimes sent the vehicle in a wrong position. The objects, which were small boxes that looked like scaled down bales, were then replaced with thinner objects and the problem with inaccurate readings got better; at least when the angle was not too small. Unfortunately the IR-sensor stopped working too so a new sensor was needed. This sensor was even less accurate and had a working range of 10 to 80 centimetres.

There was also early testing of driving vehicle forward for a certain distance and turning to the sides at a certain angle by measuring the time it spent to do these operations. The tests turned up to be almost accurate, but in later tests the time to drive one meter for instance had changed. The problem occurred because the batteries lost power and the motors got a smaller voltage through. This problem could be fixed by using the encoders on the wheels to measure travel distance. However they were not very accurate. The problem was solved by using an adapter to power the vehicle with a steady voltage. Later tests on the final approach of the bale where still inaccurate. The problem was that the vehicle turns around its center and not where the sensor is placed. This was fixed in the coding by adding half the vehicles length to the calculations.

In the final test the success criteria was if the sensor could get an accurate measurement of the size of an object and the vehicle then would move in front of the small size of the vehicle. See setup for sensor test in figure 5.2.

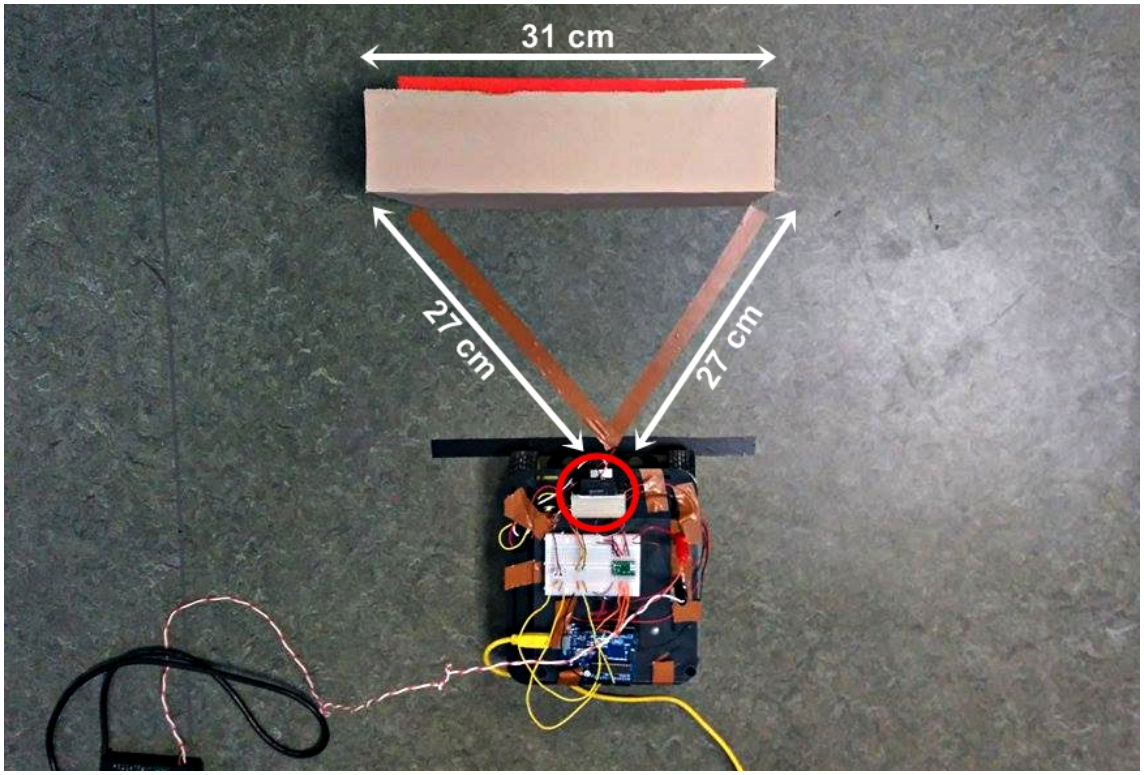


Figure 5.2: Setup for test of IR-sensor.

Also a test of the security system should reveal if the vehicle would stop if there was movement around it.

The sensor test showed some almost accurate readings but it also got bad readings sometimes. When the vehicle was placed in front of an object the sensor got better readings than when it was placed within a small angle. When we tested if the vehicle could use the sensor readings to drive in front of the bale we got different results. Since the sensor readings was some times inaccurate the vehicle would off course not go to the right location. also the wheels some times lost the grip of the ground which also resulted in a wrong approach. When sensor readings where right and the vehicle drove correctly according to these data the tests was successful. To document this we used Serial.print to illustrate that the program actually worked as it was supposed to. See figure 5.3.

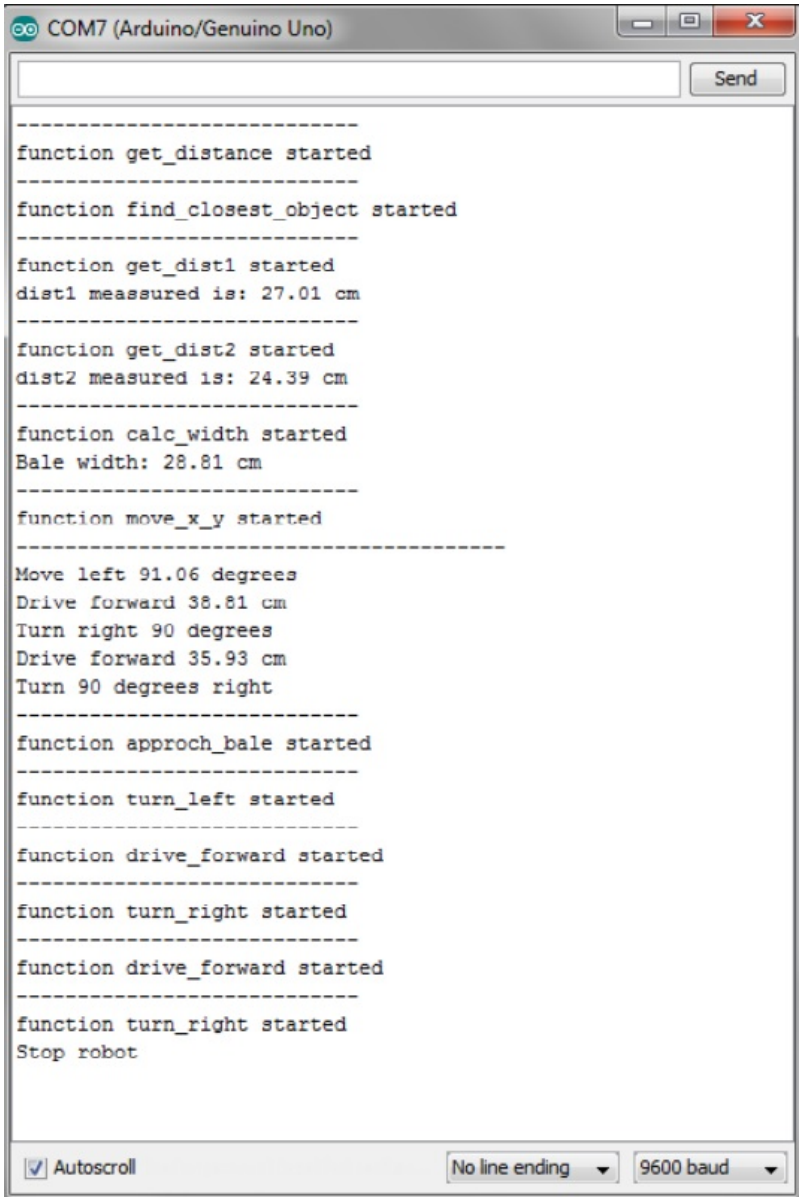


Figure 5.3: Serial Monitor illustrating the different steps in the vehicles working routine.

So comparing to the flowchart in appendix x the behaviour of the vehicle was correct. Also the programs math which is described in section ?? was executed correctly.

The safety test was done by testing if human movements would stop the robot and turn on the alarm until a reset button had been activated. This test worked out fine without any problems.

5.2 Navigation testing

All of the GPS navigation tests were done on the basketball court on Aalborg University's Campus, this place was an optimal testing area since it was outdoors and also the surface of the basketball court was really even. It was quite important to us that we could find an area in which our equipment would not get damaged by the terrain. Since all of our tests were conducted in mid December, we were limited by the rainy weather and also the short day-light hours.

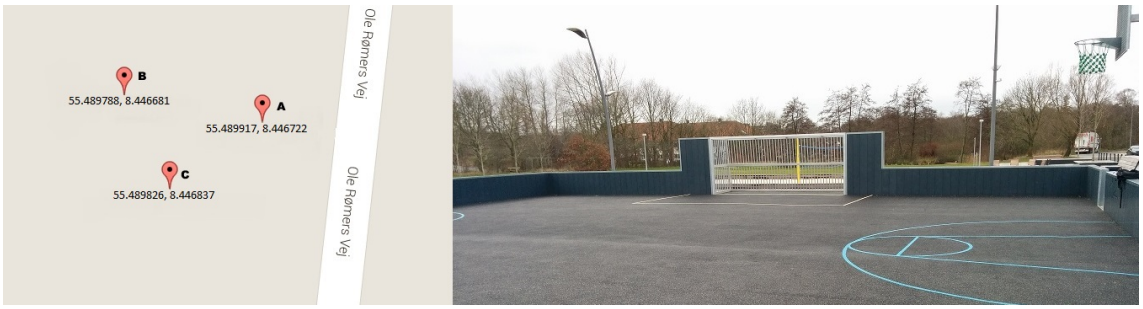


Figure 5.4: Outdoor testing area

To test the GPS navigation system, the vehicle received three GPS coordinates. These three coordinate were visited in the following order $A \rightarrow B \rightarrow C \rightarrow Stop$.
During the first test, the vehicle had a lot of issues finding an optimal course, the mean reason of this being when adjusting based on the error heading, the vehicle would over-steer, causing it to adjust its heading again but in the opposite direction. This caused it to constantly turn left and right, doing a sort of zigzag motion. By introducing the small 50ms delay found in code listing 4.14 reduced the inaccuracy of the heading adjustment.
Initially the heading threshold was set to 10 degrees and with a distance threshold of 3 meters. A heading threshold of 10 degrees was way too many degrees, since the vehicle would only adjust its desired heading to be slightly within the threshold margin. This caused the vehicle to constantly adjust it self to being 10 degrees off its optimal course, because of this obvious mistake we terminated the test prematurely and tweaked the settings.

The main test was conducted using a heading threshold of 4 degrees and a distance threshold of 3 meter. This test was a complete success, since the vehicle was able to maintain its course without having to adjust too often and with an heading error margin of only 4 degrees the vehicle was able to always have a reasonable heading even after adjusting it based on the desired heading calculations.
One thing that also became really obvious during the test was the slight inaccuracy

of the GPS receiver. For some of the coordinates the vehicle would progress to a new target coordinate almost precisely 3 meters before reaching the location, this being what we expected if the GPS receiver would be as accurate as possible. At other times it would seem that the readings of the received current GPS location were shifted slightly, causing the vehicle approach the target coordinate further than the 3 meter threshold. Debugging the navigation system through the serial monitor in situations like these would show that the current location of the vehicle was not entirely accurate. The reasoning being that the average accuracy written in the datasheet is approximately <1.9m.

```
---------------------------------------------
Current heading/heading: 259.77 Degree
Current Location: 55.489955, 8.446860
Distance to next coordinate: 9.64
Target Location: 55.489917, 8.446722
Desired heading: 244.23 - Error: -15.54 - Adjusting towards the left
---------------------------------------------
---------------------------------------------
Current heading/heading: 250.89 Degree
Current Location: 55.489955, 8.446860
Distance to next coordinate: 9.64
Target Location: 55.489917, 8.446722
Desired heading: 244.23 - Error: -6.66 - Adjusting towards the left
---------------------------------------------
---------------------------------------------
Current heading/heading: 246.67 Degree
Current Location: 55.489948, 8.446860
Distance to next coordinate: 9.29
Target Location: 55.489917, 8.446722
Desired heading: 249.54 - Error: 2.87 - On course
---------------------------------------------
---------------------------------------------
Current heading/heading: 246.43 Degree
Current Location: 55.489948, 8.446860
Distance to next coordinate: 9.29
Target Location: 55.489917, 8.446722
Desired heading: 249.54 - Error: 3.11 - On course
---------------------------------------------
```

Figure 5.5: Navigation system output

Figure 5.5 shows some of the debugging information we used whilst testing the vehicle in an outdoor environment. The most interesting pieces of information in terms of influence when it comes to the decision making of the navigation system is the *Error* and the *Distance to next coordinate*.

5.3 Computer vision testing

The program used for bale recognition was constantly tested during its development. The end goal of the program was to use a yellow coloured sponge to act as a bale, with a line of blue tape taped to one of the sides of the sponge to be used for angle calculation. Therefore, the various values used in the program were based on these two objects. These objects can be seen in figure 5.6.

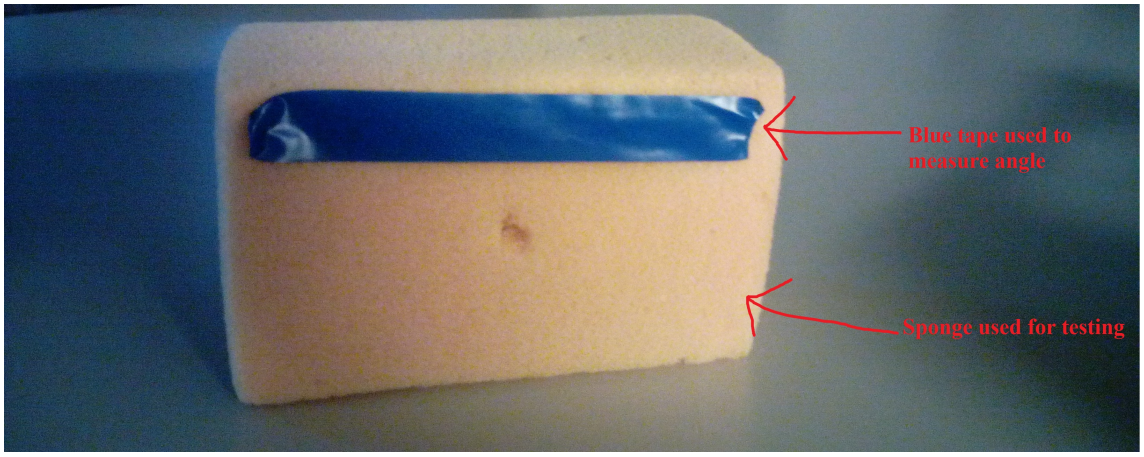


Figure 5.6: Image of sponge used in testing

The initial program contained two goals: load the image from the camera and then threshold the image to find the specific colour. The program was partially successful: the program's output gave us both the stream from the webcam and the thresholded image of it, however, other objects of similar colour would also be detected. During testing, this meant that the skin would occasionally be detected as well and if it was too close to the sponge on the camera, it would be merged with the sponge and recognised as one object. We could not completely eliminate the detection of skin or any other similar colour objects, so we had to be careful during further testing to avoid interference. See figure 5.7 for the demonstration.

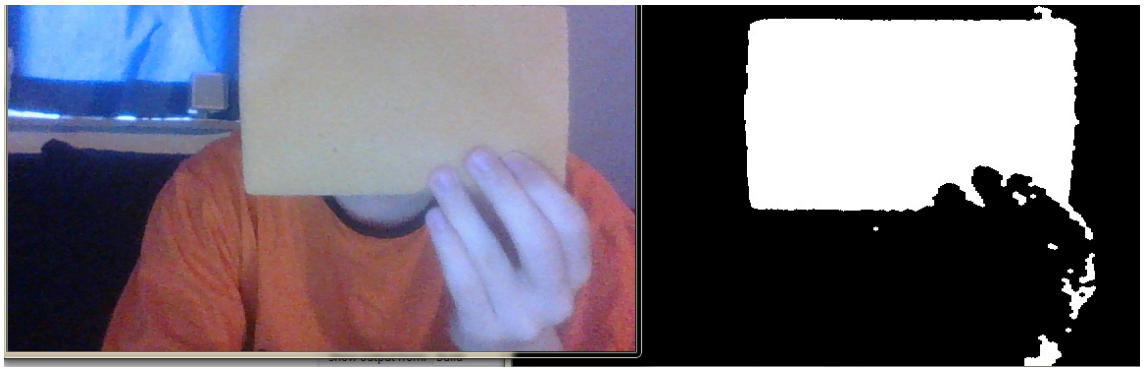


Figure 5.7: Detecting sponge, with human skin interfering

The program was then expanded to detect contours in the thresholded image, show

them and then draw rotated rectangles around them. The results were given in another screen, as seen in figure 5.8.

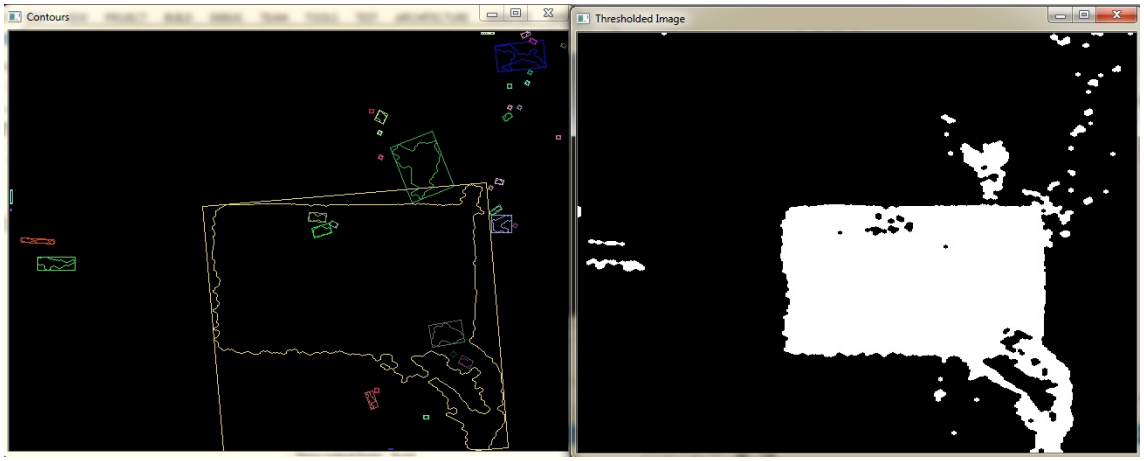


Figure 5.8: Detecting and drawing contours and rotated rectangles

We could now detect all objects of the desired colour on the screen, however, since there were objects of similar colour interfering, we had to modify the program to find the biggest object. Therefore, while the output screens for both thresholded image and contour detection showed us all of the objects, the math would be done only on the biggest object detected.
To move forward towards our goal, we needed to know the approximate distance from the camera to the object. That had proven to be difficult to achieve, though, since we were working with two-dimensional images. In other words, we had no sense of depth of the image, so we had no way of calculating the distance between two objects. Instead, we decided to use the height of the object in pixels on the camera screen to estimate how close the object is to the camera. We chose to use height instead of width because the object would be in the same or similar height level as the camera. If we were to use width instead, we would get inaccurate results when the angle between the object and the camera changed. The results were displayed on one of the output screens and an example can be seen in figure 5.9.

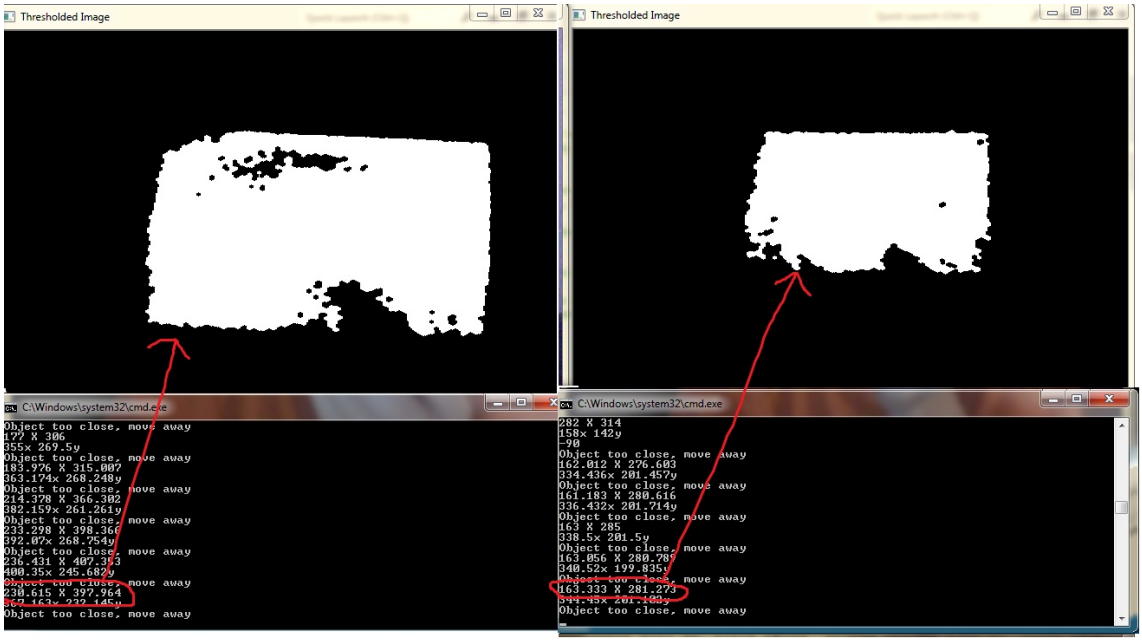


Figure 5.9: Estimating object's distance from the screen

The last parts of the program were to determine if the object was close to the center of the screen and if all checks passed, the program would then switch from detection of yellow colour to blue. The blue coloured tape would then be used to check the rotation angle of the sponge and if it was within the parameters, the program would inform us that it has fulfilled its goal. However, if at any time the position of sponge would go outside the parameters, the program should switch back to detection of yellow colour so we should be able to reposition it based on comments. In order to save resources used by the program, though, it could only detect one colour at the time. This meant that we had to create new parameters position parameters for the blue tape. In the end, we could not pinpoint the exact values that would match the ones used for sponge detection because the results would wildly vary based on the environment, amount of lighting and position of all objects involved. Therefore, the values used in the program had to be changed nearly every testing session.

In the end, the program partially fulfilled its goal - it could estimate the position of the sponge on the camera screen, but the interference from other objects and the environment forced us to recalculate the parameters or risk encountering inaccuracies. An example of program returning message after successful run can be seen in figure 5.10.

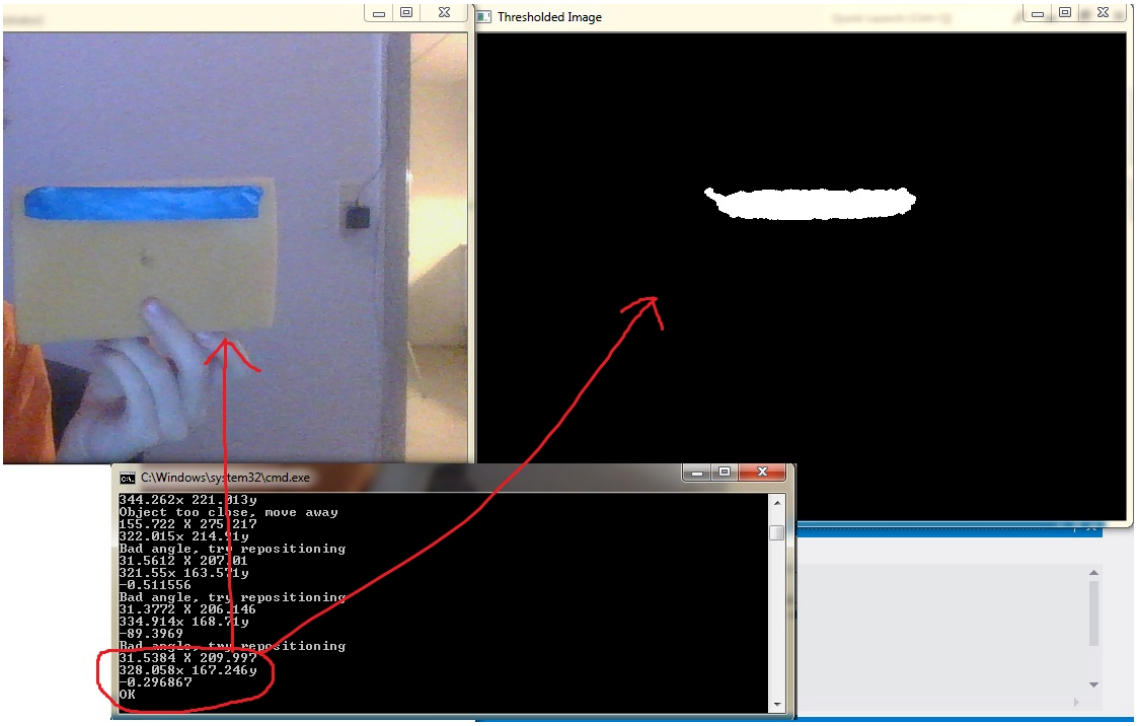


Figure 5.10: Program achieving its goal

Chapter 6

Discussion

Points to go through in this chapter:

- Collection of 130 bales in 8 hours without the driving distance exceeding 60 km
- Recognition, positioning, lifting and unloading processes should all take up 2 minutes per bale
- Meeting Arbejdstilsynets requirements on autonomous vehicles/working tools
- Invalid bale recognition
- Humans and warm-blooded animals identification
- Using a laser to process the bale orientation; correct alignment with the small side of the bale

The task of collecting 130 bales within 8 hours while maintaining the driving distance below 60 km seems fairly impossible since it would mean that driving to the bale, approaching the bale, picking-up the bale and driving to the drop-off point would all have to be done in 4 minutes. Only driving from the storage point to the bale and from the bale to the drop-off point could take up more than 2 minutes, that depending on the distance between them. One thing that we cannot control is keeping the driving distance under 60 km.

As for the bale recognition, rover positioning, bale lifting and bale drop-off processes, they have all been tested using the prototype that we built and it is, in fact, possible for them to take up maximum 2 minutes per bale since: returning the distance to the closest object takes 14.4 seconds, returning both the size of the bale and the distances to its edges take 16 seconds and approaching the bale takes only 4 seconds. These add up to a total of 34.4 seconds, which means that the vehicle has 85.6 second left to drive the last 10 centimeters to the bale. We did not test the alignment in regards to the loading process, since we did not get receive the laser sensors that we planned on using in due time.

Since the rover only drives on private ground, one of the requirements regarding Arbejdstilsynets that it needs to meet is turning on an alarm and start blinking an LED before the movement of the vehicle starts. We have implemented this on our prototype by using an LED and a buzzer, which ended up working out fine. Also, for safety, the PIR sensors will constantly check whether there is any movement around the vehicle. If there is, the vehicle will stop. Some important things to mention about the PIR sensors that we have used is that they are sensitive to direct sunlight, which will probably affect their functionality. Also, their life span would have to suffer from too much light exposure.

The recognition of invalid bales was a problem since the IR sensor are inaccurate. We did not manage to get positive testing result while measuring sharp angles. When the rover was placed right in front of an object, the IR sensor was able to measure more precise distances and calculate the width with a precision of $\pm$ 3 cm.

As mentioned above, PIR sensors are continuously scanning the surrounding area. They can, however, only detect humans and warm-blooded animals. We planned on using two laser sensors to scan the area too, but since the sensors' arrival was so delayed, they were never implemented in the final prototype.

As far as our prototype is concerned, we had some issues with the laser installment. The laser sensor that we had from a previous project broke down and we had to replace it with an IR-sensor, which is not as precise. The final testing was downgraded and the vehicle was placed in front of an object in a 90° angle, so that we could test if the measurements to the edges of the object were being measured correctly and also if the vehicle would drive to the small side of the object. This test was not a complete success because we sometimes got some incorrect readings of the distances' values.

Whenever the IR-sensor returned bad readings, the vehicle would, of course, make its movements according to those bad readings and, therefore, it would drive to a wrong position. But, even when the readings were right, the vehicle would still not be able to perfectly align with the small side of the bale because of a series of factors. One of them would be the grip of the ground, which made the wheels spin without moving the vehicle. Also, because the movement was done according to how much time it should take to drive a certain distance, the accuracy was low. The encoders that we planned on using to measure the distance traveled were, however, even more inaccurate. The different steps of the running program were documented in the Serial Monitor and it showed that the program did make the right calculations and the right sequence of movements while trying to approach the object in the way it was supposed to.

We received the GPS module towards the end of our development, this limited the amount of time we were able to put into the navigation system. Our requirement for creating a successful navigation system was to enable the vehicle to autonomously navigate between coordinates.
In the end we managed to create functioning prototype that with ease navigated between our target coordinates, with only minor amount of setbacks. To improve the navigation system even further steering accuracy would have to improved, since on the current vehicle there is no proper speed control and the actual chassis used is kind of cheap and does not provide a good grip on the surface. The lack of speed control had the greatest impact on the performance of the navigation system, it forced us to slow down the entire navigation process by adding the small delays in the turning, which we mentioned in the testing chapter, to avoid the vehicle from over-steering.
Besides implementing better speed control, further changes to the compass would also improve the navigation system. In its current state, at an idle position the compass heading returns a varying heading in the range of $\pm 2°$, this limits the algorithms ability to have an error threshold for the heading below $3°$, since the range of $\pm 2°$ would continuously trigger the *adjustCourse()* function seen in code listing 4.14 and send the vehicle into an endless loop correcting its heading. It is a possibility that the compass has not been calibrated correctly, and that its the library causing some inaccuracy in regards to the heading output.

Since the development time was so limited, we did not have the time to adjust the accuracy of the GPS module, one of the reasons being that the default settings worked surprisingly well out of the box. During the testing phase the accuracy of the GPS proved to be much better than expected, which caused us therefore to on creating a working prototype that would showcase the navigational properties, instead of delving deeper into improving its accuracy. There are a multiple of ways we could investigate further whether or not it is possible to increase the accuracy of the module, some of them being trying to use DGPS and WAAS, and even investigating the floating point accuracy when converting the data from the GPGGA NMEA sentence to decimal degrees.

While designing the navigation system we kept in mind that it should be compatible for combination with the bale-recognition and pick-up systems. When the navigation system reaches a target location, this being when the threshold distance is exceeded (See code listing 4.12), the navigation algorithm can be modified to pause the navigation and instead start the process of location the nearest bale and positioning the vehicle correctly for the pick-up procedure.

6.1 Computer vision improvements

One of the limiting part of the computer vision is different lighting. Since the machine will be running outside the whole time for long periods of time, the amount of light will constantly change. These changes will heavily impact the recognition of the bale, since the HSV-based method used in the computer vision program is made

for a fixed amount of lighting. Therefore, a workaround must be found to create a more dynamic recognition. One solution would be to use a photoresistor. It is a light-controlled variable resistor, which , as the name suggests, changes resistance depending on the amount of light hitting its surface. The more intense the light is, the lower the resistance is. Using a photoresistor, the computer vision program could be expanded to change the HSV values based on the resistance of a photoresistor by creating different sets of values for resistances at certain intervals. Another idea would be to use active lighting to create a fixed amount of lighting being given away by the machine. This light would be directed at the bale and would always be acknowledged by the camera, so, in a certain area, the camera would read the same amount of light no matter the time of day. Third option would be the use of an infra-red camera. However, the exposure to daylight could affect the camera's effectiveness. Thus, the infra-red camera should only be used when sun cannot be seen.

We would also have to make a final decision on what processing unit we would use to run the entire system, that could also handle computer vision well.

The system also needs more field testing to tackle the different lighting and how to handle the problem with the bale and its surrounding being the same colour. It has been already mentioned in the report that the current program sometimes detects objects of similar colour to the target. On a field, this issue would be clearly visible when the program will not be able to tell the bale apart from the rest of the field and instead recognise them as one object. Therefore, the current detection algorithm would need to be improved. One possibility could be to estimate the part of the bale where it has different HSV values than the field on the horizon and then instead use only a part of the bale for the calculations in the program. Another idea could be to use the strings that hold the bale together for the recognition and calculations instead. In the end, if everything fails, the very last solution could be to look for an alternative algorithm.

The combination of the computer vision code, GPS, navigation and laser detection is also a big project in itself.

Chapter 7

Conclusion

7.1 Conclusion

Chapter 8

Perspective

8.1 Perspective

One of the limiting factors in therms of prototyping and testing, was the fact that the mechanical group did not build a prototype. We decided to make 3 prototypes for our project, optimally we would prefered to combine them into a single prototype showing all of the functions, but because of time restrictions we were not able to fulfil that desire. A crucial next step would also to be looking into the laws and regulations surrounding autonomous vehicles, before moving onto the next stage of the project. These regulations would have a massive impact on our design choices and plays a crucial role in terms of the safety system and its design. We know the limitations of the PIR sensor, when used in an outside environment, which has been covered in the discussion. If developed further, more testing of the PIR would be needed in an outside environment and perhaps even a change of sensor.
Previously in the report, we mentioned that the we will need the GPS locations of every bale from the baler as they are created. This means that it is necessary for us to also create a system that will be placed on the baler to add coordinates of the bales to a database. The navigation system would then need to be modified to retrieve this information and thereafter calculate the optimal drop-off location, before heading out on the field to collect the bales. We would also need to implement a searching algorithm in case of losing the GPS locations of the bales, or if there is bale missing at a target location. The searching algorithm would follow in the same pattern as the baler, see figure 8.1, since the pattern is not always consistent the searching algorithm should be able to detect the bales it meets along the way. The searching algorithm would be much slower and less efficient than if all the bales have their own separate GPS locations, since its possible to calculate the optimal routes and thereby become as efficient as possible.

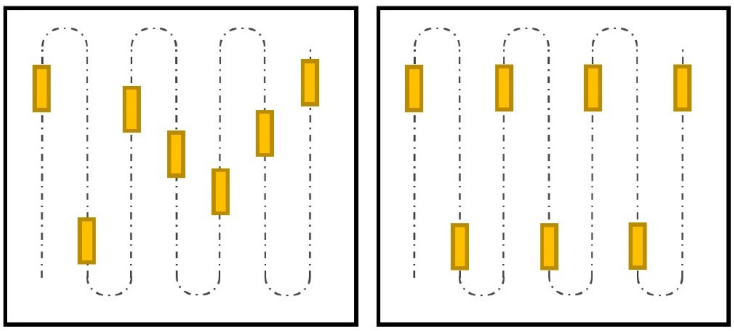


Figure 8.1: Baler driving pattern

The design of a user interface for the farmer is also needed. This interface should allow changes to different configuration options such as the number of bales, manual input of a collection place and size of bales. This interface system should also have the capabilities to connect with the camera in case of emergency or for navigation scenarios. It would then be possible for the owner to see a picture or video feed from an incident on the field, and remotely decide whether or not to reset the vehicle using the interface or going out on the field to the actual vehicle.

We would implement a system that uses both computer vision and a laser for detection of bales and size recognition. That is done to take the best out of both system. The computer vision system can more easily confirm that we are currently looking at is a bale in front of the vehicle, but it has a hard time detecting the edges of the bale, since the bale and its surroundings are the same colour. Also changes in outdoor lighting is a limited factor for the camera but not the laser. For further development it is also necessary to look into other types of cameras and possibly different ways of processing the imagery with the computer vision, to ensure we are using the best possible solution.

Bibliography

[1] Jens Aage Poulsen. Det historiske overblik. http://www.syntetisktale.dk/pdf/det_historiske_overblik.pdfl, 2007.

[2] Johannes Ravn Joergensen. Hvad er det optimale høsttidspunkt? https://www.landbrugsinfo.dk/Planteavl/Plantekongres/Filer/pl_plk_2014_shw_33_1_Johannes_Ravn_Joergensen.pdf, January 2041.

[3] Rajdeep Kaur Gurveen K.Sandhu, Gurpreet Singh Mann. Benefit and security issues in wireless technologies: Wi-fi and wimax. http://www.ijircce.com/upload/2013/june/27_Benefit.pdf, June 2013.

[4] Harris RF. Radio communications - in the digital age. http://rf.harris.com/media/Radio%20Comms%20in%20the%20Digital%20Age%20-%201_tcm26-12947.pdf, 2005.

[5] Person captured with a thermal camera picture. http://cdn.bgr.com/2015/07/thermal-face.jpg.

[6] Infrared sensor object detection picture. http://www.education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/images/409px-IR_Sensor_Principles.png.

[7] BMVA. What is computer vision? http://www.bmva.org/visionoverview.

[8] Computer vision picture. http://www.rcs.ei.tum.de/fileadmin/tueircs/www/title_slideshow/adas_vision_ohne_titel.jpg.

[9] OpenCV. http://opencv.org/about.html.

[10] Transcat. Why use a laser distance meter? - understanding the technology. https://sites.google.com/site/todddanko/home/webcam_laser_ranger.

[11] Figure for the rangefinder using angles. https://electronics.stackexchange.com/questions/83328/real-time-short-distance-range-finder-and-visual-output.

[12] Todd Danko. Webcam based DIY laser rangefinder. https://sites.google.com/site/todddanko/home/webcam_laser_ranger.

[13] The Library of Congress. What is GPS? http://www.loc.gov/rr/scitech/mysteries/global.html, June 2011.

[14] Giscommons. Gps sphere figure. http://giscommons.org/files/2010/01/2.141.gif.

[15] Trimble. Triangulating. http://www.trimble.com/gps_tutorial/howgps-triangulating.aspx.

[16] Trimble. Pseudo random code. http://www.trimble.com/gps_tutorial/sub_pseudo.aspx.

[17] Trimble. Measuring distance. http://www.trimble.com/gps_tutorial/howgps-measuring.aspx.

[18] Trimble. Differential gps. http://www.trimble.com/gps_tutorial/dgps-how.aspx.

[19] Australian Maritime Safety Authority. Differential global positioning system. https://www.amsa.gov.au/navigation/services/dgps/, November 2011.

[20] Latitude and longtitude figure. http://www.physicalgeography.net/fundamentals/images/globe5.jpg.

[21] MapTools. Latitude and longitude definitions. https://maptools.com/tutorials/lat_lon/definitions.

[22] HC-SR501 PIR motion sensor. http://s3.amazonaws.com/thmb.inkfrog.com/pix/noodlehed/hc-sr501_2.jpg/596/0.

[23] Ash blue. Figures related to the a* pathfinding. https://ashblue.github.io/javascript-pathfinding/.

[24] Ray Wenderlich. Introduction to a* pathfinding. http://www.raywenderlich.com/4946/introduction-to-a-pathfinding, September 2011.

[25] Arduino uno microcontroller figure. https://blog.arduino.cc/wp-content/uploads/2011/01/10356-01b.jpg.

[26] Arduino uno microcontroller specifications. https://www.arduino.cc/en/Main/ArduinoBoardUno.

[27] Dfrobot 4wd arduino-compatible platform figure. http://www.robotshop.com/en/dfrobot-4wd-arduino-platform-encoders.html.

[28] 4-wheel robotic platform w/ dc motors datasheet. http://www.robotshop.com/media/files/pdf/datasheet-rob0025.pdf.

[29] Sharp ir ranger sensor gp2d12 figure. http://g02.a.alicdn.com/kf/HTB1_8BIXXXXXQXXXXq6xXFXXX9/-font-b-GP2D12-b-font-IR-Ranger-Sensor.jpg.

[30] Sharp ir ranger sensor gp2d12 datasheet. http://www.geeetech.com/wiki/index.php/SHARP_IR_Ranger_Sensor_GP2D12_for_Arduino.

[31] Infrared sensor distance calculation formula. http://hackcooper.org/static/arduino.pdf.

[32] DRV8833 wiring diagram. http://www.hobbytronics.co.uk/image/data/pololu/drv8833-dual-motor-driver-4.jpg.

[33] Dual H-bridge motor driver DRV8833 - datasheet. http://www.hobbytronics.co.uk/datasheets/drv8833.pdf.

[34] GISGeography. Magnetic north vs geographic (true) north pole. http://gisgeography.com/magnetic-north-vs-geographic-true-pole/, June 2015.

[35] Figure HMC5883L magnetometer. http://arduinotech.dk/wp-content/uploads/2013/11/HMC5883L-Triple-Axis-Compass.jpg.

[36] 3-Axis digital compass HMC5883L - datasheet. http://www.hobbytronics.co.uk/datasheets/sparkfun/HMC5883L-FDS.pdf.

[37] Omer Ikram ul Haq. Hmc5883l header arduino with auto calibration. https://github.com/helscream/HMC5883L_Header_Arduino_Auto_calibration.

[38] NOAA. Magnetic field calculators - declination. http://www.ngdc.noaa.gov/geomag/declination.shtml.

[39] Adafruit. Figure adafruit ultimate GPS breakout. https://www.adafruit.com/images/970x728/746-08.jpg.

[40] Datasheet for the PA6B (MTK3329) GPS module itself. https://www.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf.

[41] NMEA reference manual. https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual1.pdf.

[42] Eric S. Raymond. NMEA revealed. http://www.catb.org/gpsd/NMEA.txt, April 2015.

[43] Dale DePriest. NMEA data. http://www.gpsinformation.org/dale/nmea.htm.

[44] Adafruit. Adafruit gps library. https://github.com/adafruit/Adafruit-GPS-Library.

[45] Site used to find magnetic declination. http://www.magnetic-declination.com/.

[46] Movable Type Scripts. Calculate distance, bearing and more between latitude/-longitude points. http://www.movable-type.co.uk/scripts/latlong.html.

[47] Movable Type. Haversine and heading formula. http://www.movable-type.co.uk/scripts/latlong.html.

[48] HC-SR501 PIR motion sensor - datasheet. https://www.mpja.com/download/31227sc.pdf.

[49] HC-SR501 PIR motion sensor. http://www.robot-italy.com/media/catalog/product/cache/4/image/d43192dcd82ea942982b4b1d2a6e2479/p/i/pir_iso_botm_annot.jpg.

[50] Arduino's interrupt service routine. https://www.arduino.cc/en/Reference/AttachInterrupt.

[51] Braking friction. http://www.webpages.uidaho.edu/niatt_labmanual/chapters/geometricdesign/theoryandconcepts/BrakingDistance.htm.

[52] Braking formula. http://hpwizard.com/tire-friction-coefficient.html.

Chapter 9

Appendix

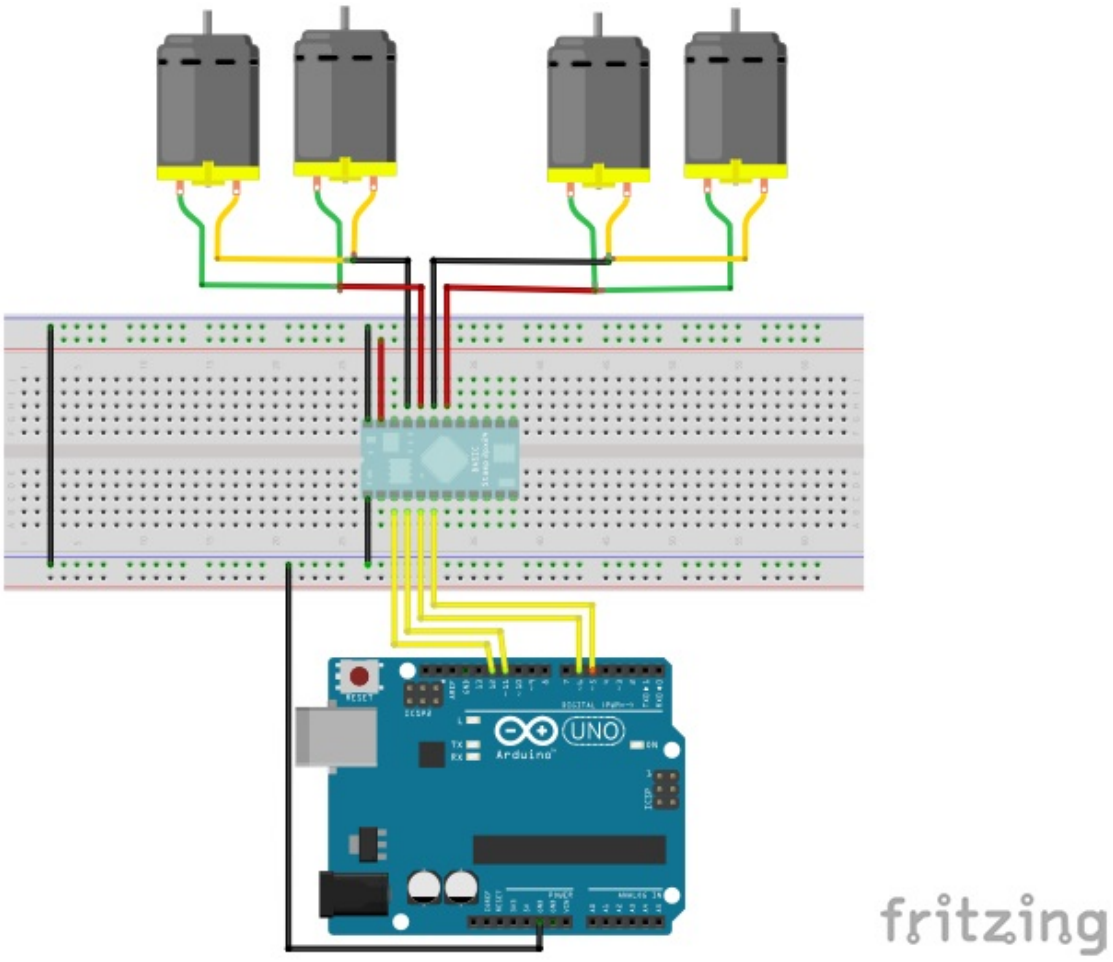


Figure 9.1: Electronic diagram of the motors and the motor controller.

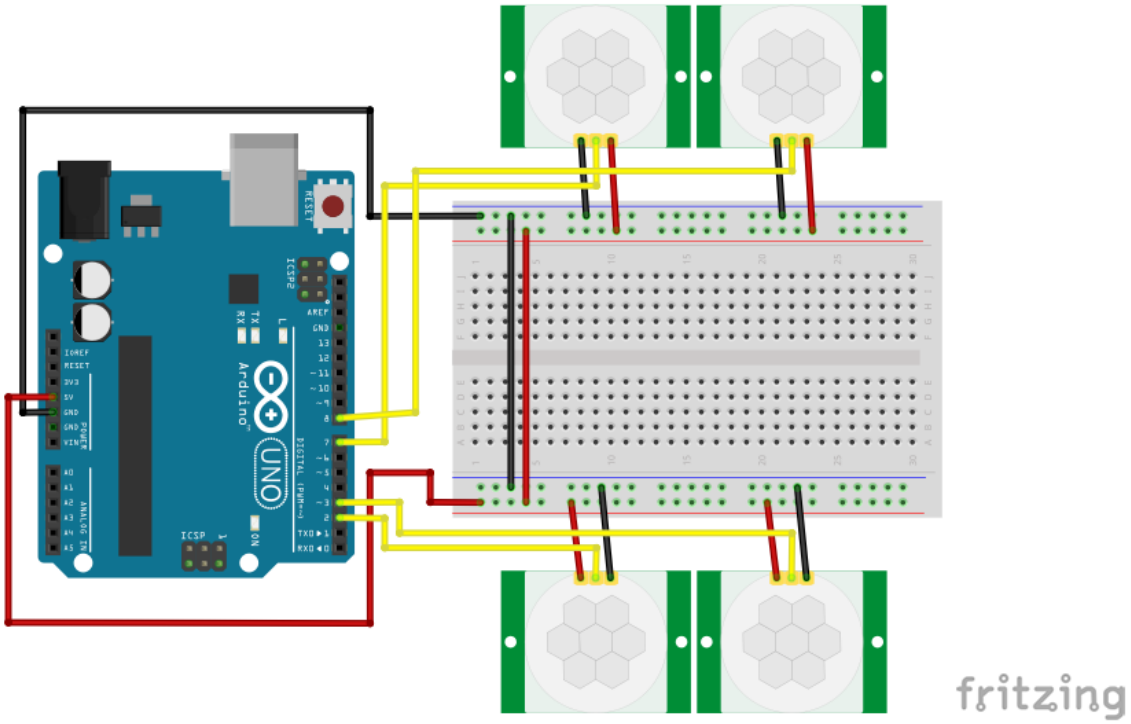


Figure 9.2: Wiring of the PIR setup.

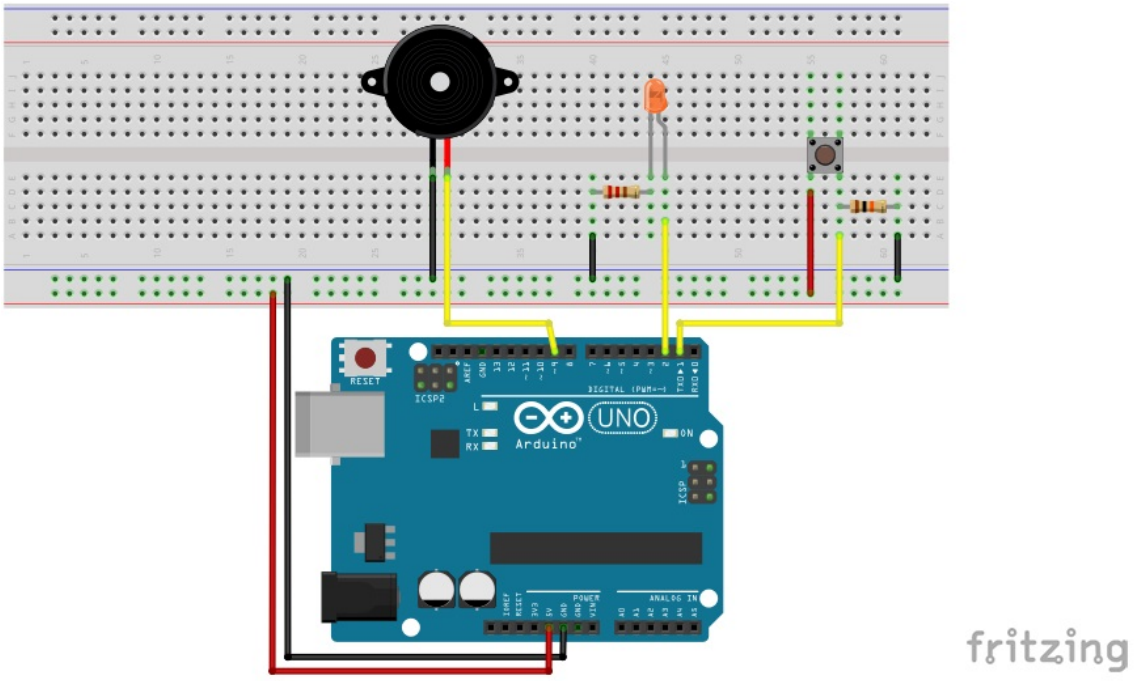


Figure 9.3: Electronic diagram of the alarm.

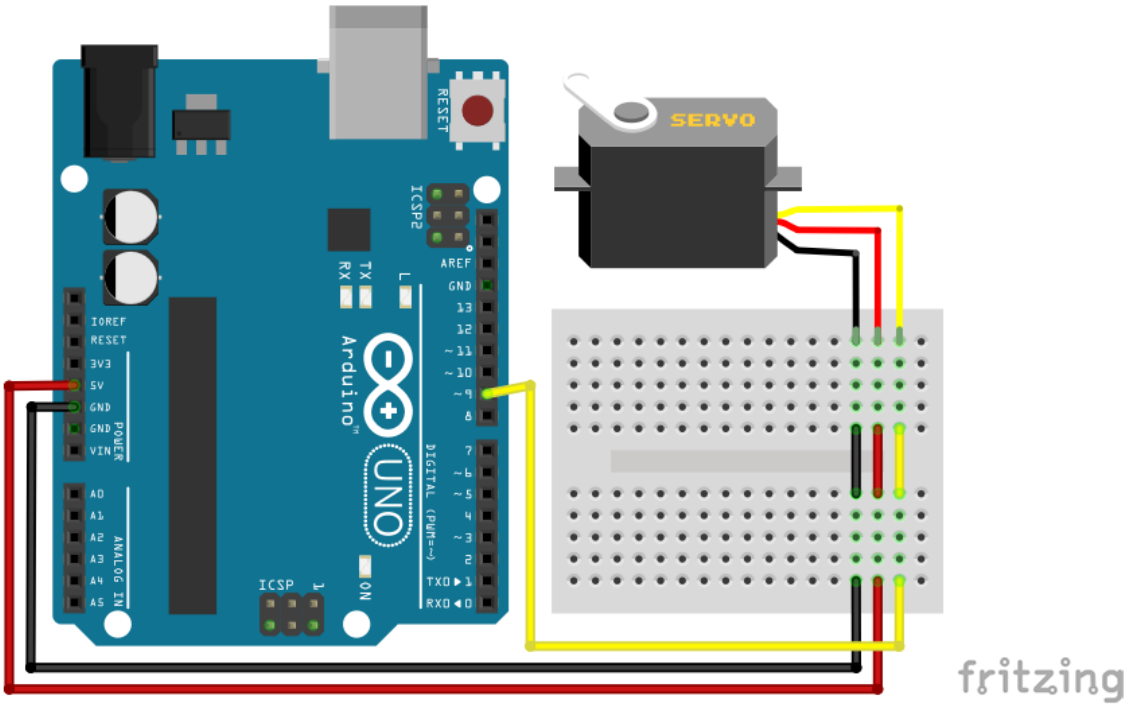


Figure 9.4: Wiring of the servo motor

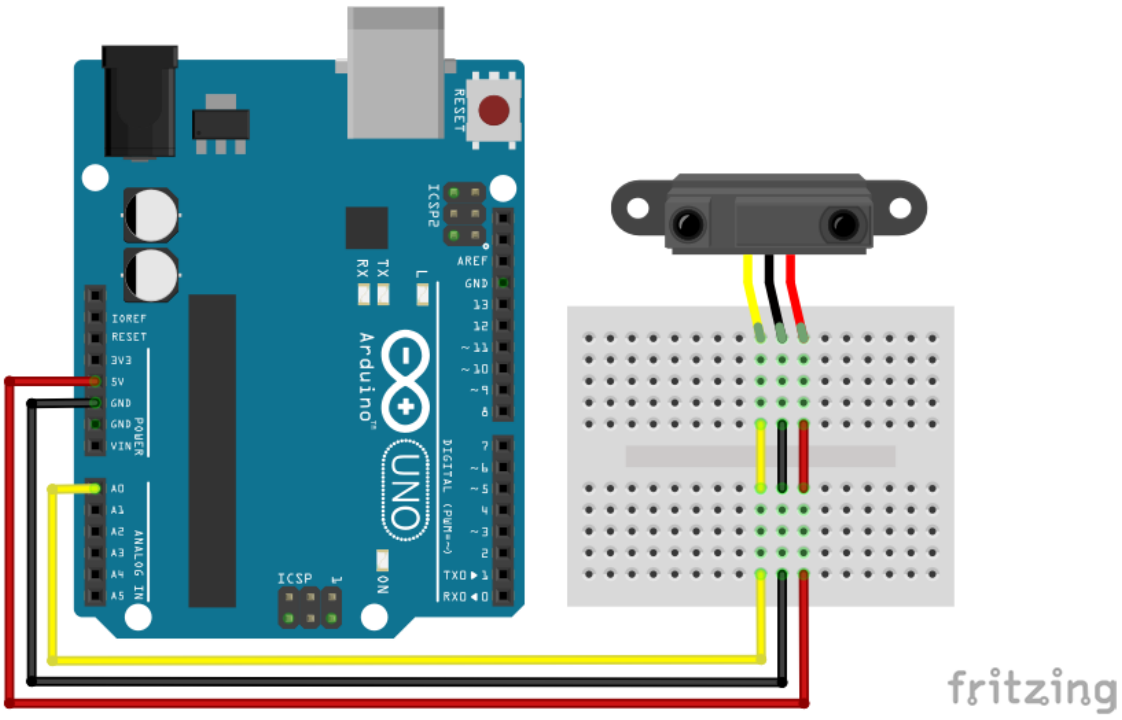


Figure 9.5: Wiring of the GPD12D12 IR Sensor.

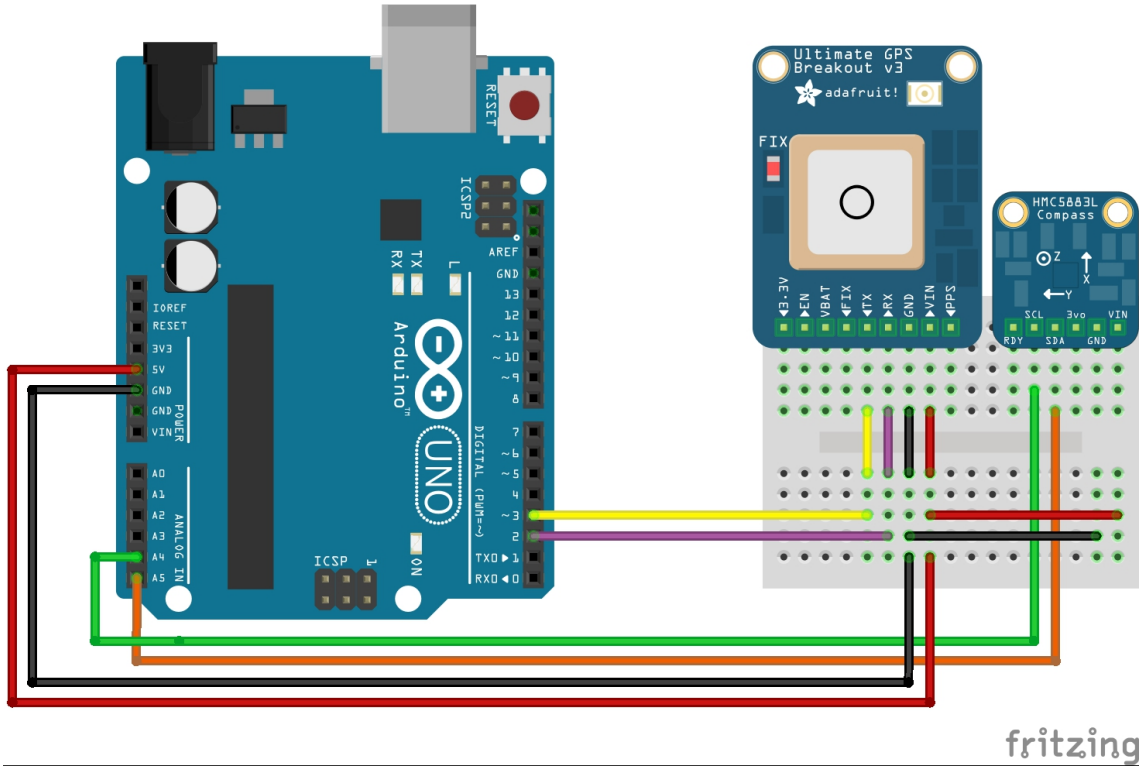


Figure 9.6: Wiring of the Compass and GPS

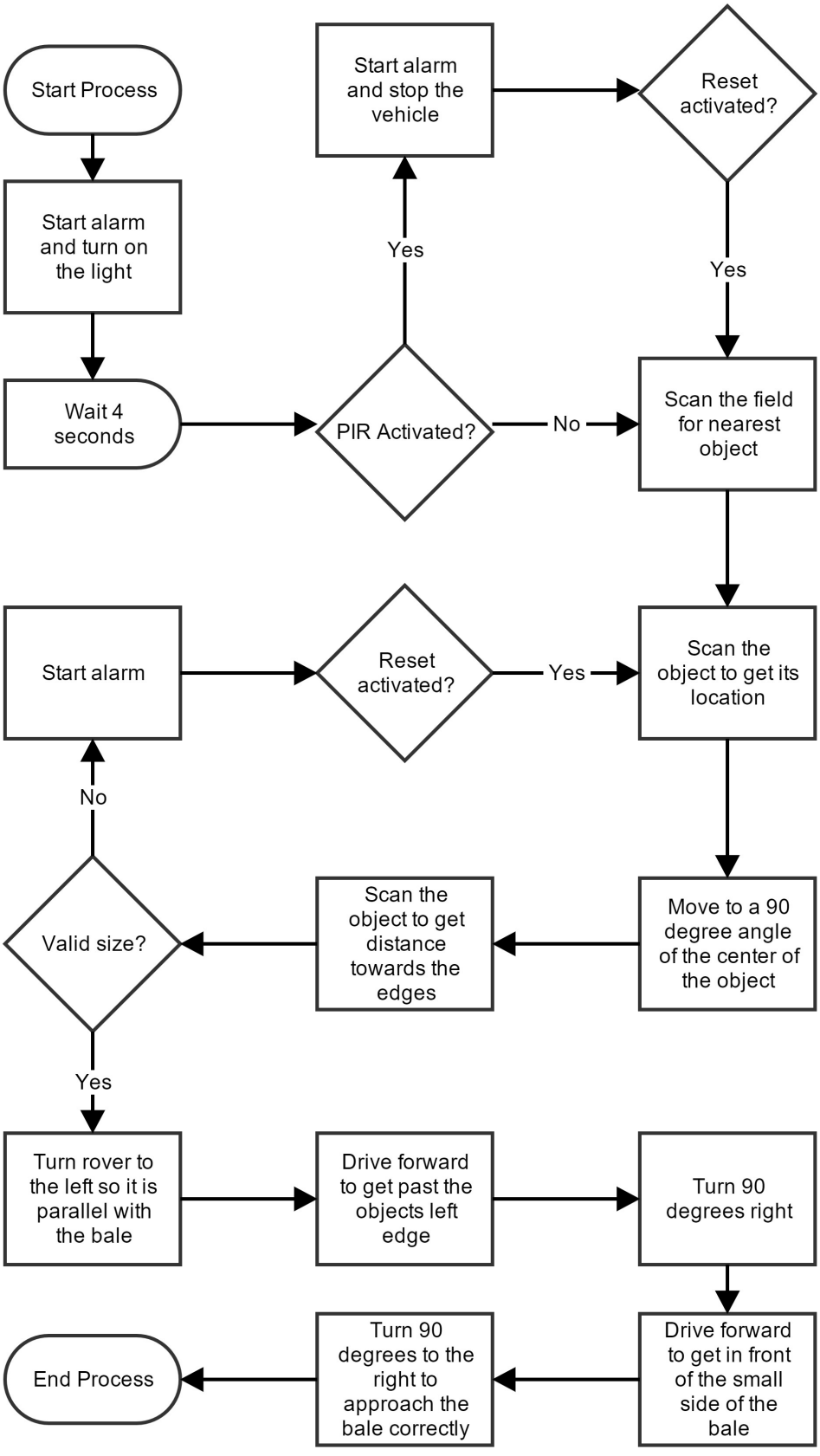


Figure 9.7: Flowchart of laser recognition process