SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY

MURDOCH
UNIVERSITY
PERTH, WESTERN AUSTRALIA

Autonomous Tracking Device based on Luggage Carrier

ENG470 - Engineering Honours Thesis

Yvonne Kaub

11/13/2015

Supervisor: David Parlevliet

This page has intentionally been left blank.

Acknowledgements

Firstly, I would like to thank my supervisor and mentor David Parlevliet for always lending an ear and the enthusiasm with which he approached the proposed project.

Secondly, thanks goes to Iafeta “Jeff” Laava for his ongoing assistance throughout the course of this project. He was always eager to hear about my progress and help out with little bits and pieces that were needed to make the project come together.

Furthermore, I would also like to thank Murdoch University staff and my fellow students for providing me with the knowledge needed to undertake a project like this, as well as cheering me up in time of need. I truly will miss the long nights in the laboratories, not for the exam and test preparations, but for the fun times shared with people that I now call my friends.

Lastly, I would like to thank my family at home for the ongoing moral and financial support throughout my whole degree. I would also like to mention Aharon Cunta, who was not only my mentor throughout this degree, but also my rock during the stressful time of this thesis.

Abstract

Creating intelligent systems that can sense and interact with their environment has been a huge factor behind the work on artificial intelligence and automation. It involves multidisciplinary and complex technology that has proven to be challenging for researchers working in that area. Despite this, great progress has been made in this area of study over recent years and automated systems can be found everywhere in our daily lives from the automobile industry to manufacturing.

The aim of this project involves investigating the idea of an autonomous tracking device. This device will be based on an autonomous luggage piece which would make a traveller's journey a lot easier. It is important to keep in mind that this idea could be implemented on more than just luggage. Other systems this could be implemented on would be shopping trolleys, golf bags or even for film making.

In order to achieve autonomy, background research on triangulation was conducted and is presented in this document. The sensors used for distance measurements were modified ultrasonic sensors used in combination with radio and Bluetooth transmitters.

The whole project involved the design of two devices, one carried by the person and hence referred to as 'the personal device' throughout this document and one tracking robot. The tracking device's control circuit was placed on a rigid body with two DC motor driven wheels at the front. Power was supplied by using a battery pack and the device's brain consisted of Arduino microcontrollers.

The motor shield purchased to drive these motors failed to work properly and hence the device could not be tested as an autonomous device. It is still shown that the control algorithm implemented can modify the PWM frequency inputs to the motors according to the direction the device needs to travel in. Hence the use of triangulation in combination with feedback control to drive an autonomous device is indeed feasible. Collision avoidance was implemented using Infrared sensors.

Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

Table 1: List of Abbreviations

AI	Analog Input
AUD	Australian Dollar
A/D	Analog to Digital
BFS	Breadth First Search
CPP	C++ (C plus plus)
CPR	Counts Per Rotation
CRC	Cyclic Redundancy Check
DC	Direct Current
DI	Digital Input
DO	Digital Output
IC	Integrated Circuit
IDE	Integrated Development Environment
IR	Infrared
ISM	'Industrial, Scientific, and Medical' (frequency bands)
FIFO	First In, First Out
GFSK	Gaussian Frequency Shift Keying
GPIO	General-purpose input/output
Mbps	Megabits Per Second
MDF	Medium-Density Fibreboard
PID	Proportional, Integral and Derivative (Controller)
MCU	Microcontroller Unit
PWM	Pulse Width Modulation

RPM	Rotations Per Minute
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

1 Introduction

Creating intelligent systems that can sense and interact with their environment has been a huge factor behind the work on artificial intelligence and automation. It involves multidisciplinary and complex technology that has proven to be challenging for researchers working in that area. Despite this, great progress has been made in this area of study over recent years and automated systems can be found everywhere in our daily lives from the automobile industry to manufacturing [1].

Most conventional systems have all their sensors on-board, but the aim is to evolve those systems into open network systems with distributed processing power, actuators and sensors. A good example of this is the development of driver-less vehicles that will transport passengers from A to B navigating on data acquired passive sensor inputs integrated into the road network [1]. Autonomous vehicles are already used in the industry where large haul trucks transport iron ore from A to B, autonomously making incidents less likely and increasing productivity. The paths these vehicles take were so accurate that randomly generated errors had to be programmed into the code in order to avoid the trucks digging into the tracks [2].

The aim of this project involves investigating the idea of an autonomous tracking device. This device will be based on an autonomous luggage piece which would make a traveller's journey a lot easier. It is important to keep in mind that this idea could be implemented on more than just luggage. Other systems this could be implemented on would be shopping trolleys, golf bags or even for film making. Since designing and prototyping a project like this can be very expensive, extra focus is put on finding components that will fulfil the technical requirements whilst remaining relatively low in cost.

Reading on, the reader will find information about the different sensors used and the reason they were chosen, mathematical background, the physical layout of the tracking device and the implementation and testing of components and software.

1.1 Objectives

This section presents the objectives to be completed during the thesis period. At the end, the outcomes can be compared to these objectives to determine whether the overall project was a success.

- The first objective that will have to be fulfilled is the design and construction of a platform on which the control centre, sensors and drive system can be installed.
- The second objective is the successful identification of the person to be followed and the actual tracking of the sensors placed on the user's wristband or belt.
- The third objective is collision avoidance. A system will have to be designed and implemented that allows the device to track a person without running into other obstacles.

2 Background

2.1 Literature Review

Tracking devices have been designed and built previously by engineers. There are numerous devices that track either a path along the ground or a person.

Even the idea of autonomous luggage has been thought of before. In the following paragraphs some examples of what has been done before are listed.

2.1.1 Hop! Suitcase by Rodrigo Garcia Gonzalez:

The Hop! suitcase was designed by Rodrigo Garcia Gonzalez and won the James Dyson National award in 2012 [3]. It is a self-driven suitcase that is supposed to follow its user, another suitcase or airport staff at the airport. It tracks the user's mobile phone using three in-built sensors, a combination of computers/microcontrollers and triangulation operating a caterpillar drive system [3, 4]. No information could be found on the exact sensors and microcontrollers used. In case of a lost signal, the suitcase is able to lock itself and the user's phone will start to vibrate. The final product is still in the development phase [4].

2.1.2 Automated Luggage Carrying System:

The objectives of this research paper is the identification of efficiency of conventional luggage carrying systems when compared to the proposed system and the demonstration of the benefits of this proposed system while considering cost, speed, time and human labour. This luggage system was designed to reduce manpower, increase efficiency and maintainability while considering a flexible design for future changes [5].

The basis of this design is a six-wheeled robot with each wheel driven by its own motor. All motors had to be synchronized to ensure a mechanically simple yet robust rigid body design. During the process, the designers found that their choice of battery was flawed, as the battery chosen did not supply sufficient current for the application [5].

The control design involves the use of ultrasonic sensors on the robot itself and a smart card on the user. Since the ultrasonic sensors are only able to provide distance, not the exact location, the user's exact location had to be calculated by different means. In addition, the robot did not only have to be able to track the user, but also be able to avoid obstacles [5].

The different types of motion were achieved by rotating the different wheels in different directions. Hence, the forward motion is achieved by spinning all six wheels in the forward direction and the backwards motion is achieved by spinning all six wheels in the opposite direction. Right and left motions are achieved by turning the wheels on the right hand side forwards by and the wheels on the left hand side backwards and vice versa, respectively [5].

A smart card is a device that features an embedded IC and its inside usually contains a microprocessor hidden under a golden plate. These cards can identify the user, as well as provide authentication, application processing and data storage. Its signal can be detected by a nearby tower and information about the user's position can then be conveyed to the robot. Another idea behind this design was the return of the robot to its charging station once the user puts the smart card on its holder [5].

The return of the robot to the charging station is important as it is powered by a battery. Hence, once the battery charge is at 20%, the robot will automatically return to its charging station. This process will also be initiated when the robot has finished its work [5]. During the charging period, multiple robots will line up in a queue and use their power connectors on each side of their rigid bodies, allowing for multiple robots to be recharged by the same charger. If a robot is needed by a user, the first robot in the queue will follow the request and the second robot will take its place in the queue and so on [5].

The designers' idea is to have a waiting room for these robots next to restricted areas such as restrooms. Once the user exits such a restricted area, the robot could use triangulation to find its way back to its user [5].

As navigation can be a major challenge in Artificial Intelligence, the implementation of a powerful algorithm is essential. The designers propose the implementation of Breadth First Search (BFS) for traversing, which means the area used will have to be divided up into an m*n (rows by columns) space and then be converted into a tree structure. This tree grid can then be used to find the next position whilst remembering the last position the robot was in [5].

The environment that the robot could be used in could have an actual grid painted on the floor. Additional sensors on the underside of the robot could then tell the difference between the colour patterns and hence grid-map the space according to the Cartesian coordinate system where the initial node is set as the origin (see Figure 1). The rest of the grid can be mapped according to the origin and travel direction [5].

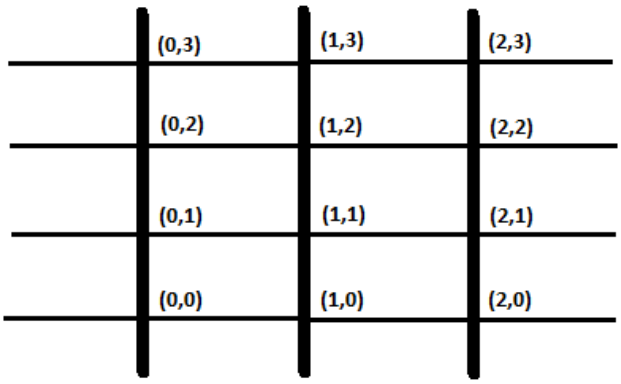


Figure 1: 3x4 Cartesian coordinate grid (black on white) [6]

In order to avoid nodes that have already been visited, the designers propose a stack dedicated to visited nodes. They apply the same principle when it comes to obstacles; nodes where an obstacle was encountered will be remembered and avoided. The direction that was taken, the coordinates of the grid can be updated and memorised [5]. Further information on the BFS algorithm can be found in Section 2.3.4.

The method of triangulation is proposed by the designers in order to detect and identify the user and the robot. A wire-less approach was taken using active beacons for absolute detection. Different types of triangulation algorithms exist and the design choice to use the 'Generalized Triangulation Algorithm' by Joao Sena Esteves was made [5]. This allows the robot to self-triangulate itself on a plane. For this approach the active beacons in the environment have to be placed intelligently since the algorithm will only work reliably as long as the robot is located within the triangle formed by said beacons [5].

After implementing and testing their proposal, the designers came to the conclusion that most of the limitations of their design are caused by lack of resources and could be worked around with more powerful equipment. This would also enable the robot to master stairs, therefore increasing the feasibility of this design in airports. They also found that the robot was not navigating obstacles reliably, if there were too many at once. This slowed the process down and hence decreased efficiency [5].

2.1.3 Wireless Controlled Luggage Carrier:

In this project, the engineers are proposing the design and build of a luggage carrier that follows its user wirelessly at a certain distance. The intention behind the design is the increase of comfort for the user. Therefore, the following device has to be capable of travelling at different speeds, as well as stopping in busy environments such as an airport [7].

The wireless communication link is achieved through sensor on the tracking unit connecting up to sensors worn on a bracelet by the user. In order to make the design secure, an alarm was implemented warning the user in case of loss of the tracking device or low battery. In addition, a flashing LED on the tracker is alerting nearby passengers of its location [7].

A design choice was made to allow a safe distance of 90-120cm between the tracking device and the user. Moreover, the device is outfitted with an on-board battery charger that allows the user to plug it into a wall in case of a dead battery. The designers chose an 18V DC motor and are proposing to

connect three 18V batteries in parallel for longer operation. These batteries can be connected to an on-board battery charger that can be connected into a wall outlet for recharging purposes [7]. Two limit switches are used to feed full left and full right steering information back to a microcontroller. A 12V DC motor is used for steering purposes and the IR beacon manufactured by Pololu is used to sense distance and direction [7]. The beacon's detection range lies between 15.2cm to 609.6cm with a supply voltage of 6-16V [8].

In order to control the steering motor's movements, the 10Motor microcontroller by SyRen with an input voltage of 6-24V nominal and 30V absolute maximum was chosen [9]. It can supply a continuous current of 10A and allows peak currents of up to 15A supplying a synchronous regenerative drive. It can also return power to the battery in case of deceleration or reverse motor operation, which in turn will save power. It features PWM frequency of 32kHz [9].

The designers chose the MC7 Motor Controller-Drive to drive the device. Its specifications are an input voltage of 12-36V with a continuous current output of 35A. It can be controlled by applying a PWM input and will use ramp speed to accelerate forwards or backwards [7].

To sensor the distance, ultrasonic sensors with an input voltage of 5V were used. These are able to measure a distance from 2cm to 3m [7].

The main part of the device's logic relies on the IR sensor pair which alternate between sending and receiving infrared light in order to not get confused with their own signals. This signal is sent more than 1000 times per second and the microcontroller monitors all four IR detectors in each beacon to decide on their position in relation to each other [7]. The IR beacon's outputs indicate to the microcontroller, which of its side receives the most light meaning that the other IR beacon would be situated in that direction [7].

One clutched DC motor is in charge of the orientation of the front wheels which is a similar design as the rack and pinion system on a conventional car. The motor in charge of drive, applies torque to a single front wheel using the friction of a smaller wheel that is mounted on the driving motor [7].

In this design, the battery is mounted underneath the robot with access to charger's transformer plug. The siren and the charger are mounted on plexi-glass situated at the bottom of the robot. The drive and steering controllers are also situated on the bottom of the device in order to prevent overheating by maintaining proper ventilation [7].

In order to find the centre point of the steering axis, a full sweep to the left and the right is applied in start-up mode. The sensors will pick up which direction to go in and then wait for the distance signal to be processed before correcting any direction errors [7].

The software components were developed individually and then brought together in the end. The distance sensor works in combination the drive motor, whereas the steering is dependent on the direction sensors. In order to ensure reliability when using the distance sensor, three values were stored and then compared to the setpoint. Only if all three values are valid (all smaller or all bigger than setpoint value), the feedback will start compensating for the error [7].

The engineers' future works include obstacle detection and handling, group robot control and 'social interaction' between multiple robots [7].

2.2 Component Review

2.2.1 Distance Ranging

Table 2: Distance Sensor Comparison

Name and Maker	Range	Price (as of November 2015)	Power Req.	Voltage Output
IR Sensor by Adafruit	100cm-500cm (1-5 meters)	$AUD 43.12	5VDC 30mA	1.4-3VDC
Ultrasonic Sensor by Pololu	2cm - 3m	$AUD 57.13	5VDC 35mA active	TTL pulse
IR Sensor by Polulu	10 - 80 cm	$AUD 13.93	4.5- 5.5 V 33 mA	0-3.1VDC
HCSR04 Ultrasonic Sensor	2cm – 400cm	$AUD 3.50	5VDC 15mA	Pulse width 5V

[10-12]

Listed in Table 2 above are common distance sensors. At first glance it becomes obvious that there is a huge difference in the price between some of the sensors. Hence, it was immediately clear that either the IR Sensor by Pololu [11] or the HCSR04 Ultrasonic Sensor [10] needed to be looked at closer. Since the IR sensor only works over a distance range of 10 – 80 cm, the HCSR04 sensor was chosen. If one wanted to choose an IR sensor with a greater distance range, one would have to look at spending twice the amount of money. Another strong reason for choosing this sensor was the amount of online projects that had used this sensor in the past, hence providing an online support platform for this project. Two of the IR sensors were also purchased for use in a collision avoidance system.

2.2.2 Microcontrollers

Table 3: Microcontroller Comparison

Name and Maker	Price (As of November 2015)	Clock Speed (Processor)	I/O	Supply Power (PC Connectivity)
FRDM-K64F by mbed	$AUD 49.76	120MHz (ARM)	16 DI/DO, 6 AI	5V (via USB)
Raspberry Pi 2, Model B	$AUD 38.95	900 MHz (ARM)	27 GPIO	5V (via Micro USB)
BeagleBone Black	$AUD 65.38	1GHZ (ARM)	65 GPIO	5V (via mini USB, USB)
Arduino Mega 2560	$AUD 86.00	16MHz (AVR)	54 DI/DO, 16AI	5V (via USB)
Arduino Uno	$AUD 48.95	16MHz (AVR)	14 DI/DO, 6AI	5V (via Micro USB)
Funduino Mega 2560	$AUD 49.95	16MHz (AVR)	54 DI/DO, 16AI	5V (via USB)

[13-18]

An overview of the computer systems considered for this project is given in Table 3 and having to choose one of these systems to control the project was not a hard task. It became clear quickly, that the BeagleBone Black [13] is well beyond the computing speed needed. It also includes 65 GPIO and nowhere near as many are needed for this project. The Raspberry Pi's [14] clock speed is also too much for the project and it was sold out. That left the Arduino microcontrollers as well as the mbed microcontroller. The mbed microcontroller [17] is not sold in the local electronic stores and the website was down at the time of component acquisition. The Arduino microcontrollers [16] were

readily available in all the local electronic stores. With their huge online community [16] and support, they form the perfect platform for any new developer.

An Arduino Nano was already purchased for home projects and was supposed to be used for the personal device. However, it was later decided to purchase an Arduino Uno due to its advantages over the other components listed in Table 3 above, but it was not powerful enough to work with the motors. It was later used on the personal device instead of the Arduino Nano. The reader can find more information about this in Section 5.2.6.

Hence, a Funduino [18] was purchased from an electronics store to control the tracking device. This Funduino development board has exactly the same features as the Arduino Mega [18], which had been researched earlier. The only difference is its considerably smaller price tag of $AUD 49.95 [18] compared to the Arduino Mega's $AUD 86.00.

2.2.3 Drive System

Table 4: Motor Comparison

Name and Maker	Price	Power	RPM	Inbuilt Encoder?
Pololu 34:1 Metal Gear Motor	$AUD 48.93	6V, 80mA free-run, 2.4 A stall	165	48 CPR Encoder
Pololu 30:1 Micro Metal Gear Motor	$AUD 22.33	6V, 40mA free-run, 0.36 A stall	440	No
Pololu 29:1 Metal Gear Motor	$AUD 27.93	6V, 250mA free-run, 3.3 A stall	450	No

[19-21]

Table 4 presents the three motors that were considered for this project. Choosing the right motor was a more difficult task, as no previous projects included having to choose a motor. Hence, a lot of the motors were researched and the choice fell on Pololu's 34:1 Metal Gear Motor. Its features are further described in Section 3.1.3.2. The main reason for choosing this motor was its included 48 CPR

encoder, which was not needed for this project, but might be useful in future works. Moreover, its RPM were sufficient and a stall current of 2A is high enough to carry the small load of the tracking device. The distributor's website also had a direct to the wheels chosen in the next section ensuring these two components worked correctly together.

Table 5: Wheel Comparison

Name and Maker	Price	Number of Wheels	Diameter	Adapter
Pololu	$AUD 40.93	2	120mm	4mm adapters x 2
Sparkfun	$AUD 4.54	2	65mm	No
Pololu	$AUD 11.13	2	60mm	No

[22-24]

The main features of three of the considered wheels can be found in Table 5 above. The Pololu Dagu Wheels were chosen for this project due to their design for almost any terrain. They were also one of the only ones that included 4mm adapters, making a direct connection to the motors chosen in the previous section possible. They were by far the most expensive pair of wheels, yet they save costs when it comes to research time and price of correct adapters for the chosen motors.

It was also calculated that with a wheel diameter of 120 mm and a motor with 165RPM:

Equation 1: Circumference of Wheel

$$Circumference_{Wheel} = \pi x 0.12 = 0.38m$$

Then,

Equation 2: Max Speed

$$Speed_{Max} = 165x0.38 = 62.7\ \frac{m}{min} x \frac{1min}{60s} = 1.045 \frac{m}{s}$$

This is more than enough speed to track a person at walking pace.

Table 6: Motor Driver Comparison

Name and Maker	Price	Current Output	Number of Motors	Ease of Connectivity
Sparkfun	$AUD 46.12	4A (2A per motor)	2 DC motors	Soldering required
Seeed Studio	$AUD 31.87	2A per Channel	2 DC motors	Wires
Sparkfun	$AUD 75.46	2.5A per motor	4 DC motors	Shield
Seeed Studio	$AUD 59.29	4A (2A per motor)	2 DC motors	Shield

[25-28]

Table 6 displays the main characteristics of the four different motor drivers that were considered for this project. It is handy to have the motor driver on a shield, as it connects straight to the microcontroller and no soldering or wiring is needed. Due to its ease of connectivity and its low price compared to Sparkfun's motor shield [26], the four channel motor driver by Seeed Studio [25] was chosen. Its current ratings are sufficient to supply the motors with enough power.

2.3 Software and Mathematical Research

2.3.1 Atmel Studio & Atmel Software Framework (ASF)

Atmel Studio is an environment in which a programmer can debug, test as well as compile code and then download it to any ARM or AVR based microcontroller made by Atmel. The ASF is integrated into Atmel Studio and consists of a library of production-ready source code which includes 1600 projects written by experts and tested in production [29].

A free plugin, called Visual Micro, can be downloaded and installed which implements features targeting Arduino development into Atmel Studio. The advantage of using this environment is its much more user-friendly interface and other convenient development tools [30] compared to using the Arduino IDE on its own.

Different '.ino' sketches and 'cpp' source codes can be used and worked within the same project ensuring the compatibility of the projects with the open source code community [30].

It is due to its simplicity and user interface that the Atmel Studio in combination with the Visual Micro plugin was chosen for this thesis instead of using the more basic Arduino IDE on its own.

2.3.2 C / Objective-C / C ++ Programming Language

The programming language 'C' was first developed by Dennis Ritchie between 1969 and 1973 [31]. Due to its design providing constructs that can be mapped to machine instructions efficiently [31], it kept being used in applications that were originally programmed in assembly language, such as the 'Unix' operating system. The standard ANSI C was published in 1989 by the Institute [31].

The language was called 'C' since its features were derived from a previous programming language called 'B' which was a scaled back version of the BCPL programming language [31]. The first edition of 'The C Programming Language' was published by Brian Kernighan and Dennis Ritchie in 1978 [31] and was used as an informal specification of the language for a few years. The second edition of 'The

C programming language' covers the ANSI C standard. Features introduced in this language were the standard I/O library, the 'long int' data type and the 'unsigned int' data type [31].

Objective-C was developed by Tom Love and Brad Cox in the early 1980s [31]. Both developers had been exposed to the programming language 'Smalltalk' [31] and realized quickly that this language would not be useful for building development environments for the place they worked at. Still, backward compatibility was important and so Cox began working on a pre-processor for C [31] adding some of 'Smalltalk's capabilities. An object-oriented extension for C and a commercial venture by Love and Cox followed, which used an Objective-C compiler and class libraries [31].

C++ was developed by Bjarne Stroustrup [31], who recognized the helpful features of the programming language 'Simula' but found that it was too slow, whereas BCLP was too low-level for large software developments. He therefore decided to enhance the general-purpose, portable and widely used C language with Simula-like features [31]. Its name changed from C with classes to C++ in 1983, including new features such as virtual function names, improved type checking, references, constants and user-controlled free-store memory control [31].

Due to C++ being derived to overcome issues found in C and Object-C, it is the most commonly used programming language used nowadays. The Arduino projects are coded using a subset of C++, hence the code for this thesis is also in C++.

2.3.3 PID Controller

A PID controller is one of the basic feedback controllers used to keep a process under control and can be found in almost all industrial environments. It is particularly useful when the dynamics of a process are relatively stable and not too nonlinear. Even more advanced control systems have PID control at the most fundamental layer of their hierarchy. Some of its important features include its ability to provide feedback about the process, the capability to eliminate steady state offsets and being able to anticipate the future using its derivative action. Most of the PID controllers

manufactured today are based on microprocessors providing the opportunity of implementing additional features such as auto-tuning, gain-scheduling and adaptive parameters [32].

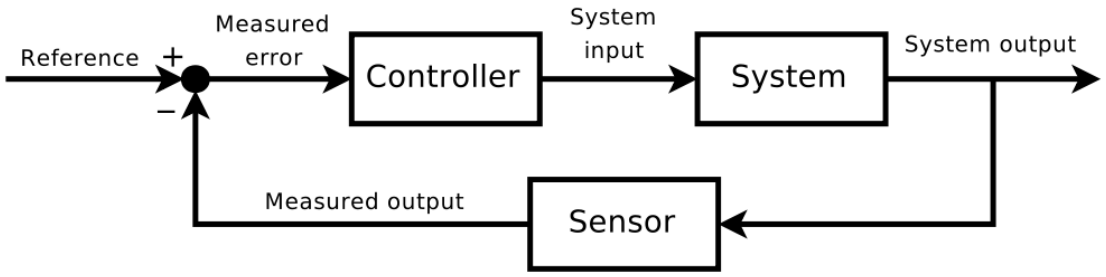


Figure 2: Feedback Loop [33]

In Figure 2 above, the reader can see a simple negative feedback loop. It is called a 'negative feedback' loop because the manipulated variable (the system input) and the process variable (system output) are compared with opposite signs. The system output is compared to the reference or setpoint. The error between the process variable and the setpoint is then mathematically manipulated by the controller in order to adjust the system input enough to ensure a stabilized system output that meets the setpoint [32].

There are three control modes that most closed loop controllers are capable of using: Proportional (P), Integral (I), and Derivative (D). There is only a limited number of combinations of these three different modes:

- P- only, which is used for the most basic controllers
- P in combination with I, where I is used to eliminate the offset caused by the P-only mode
- P in combination with I and D, where the D term is used to stabilize the process
- P in combination with only D, which is used in a special application called 'Cascade Control'
- I only, which is the primary controller of the special application 'Cascade Control'

[34].

The PID controller makes use of mathematical models of the process to be controlled. The process dynamics are expressed in the form of a transfer function which is in the s-domain and can be converted back to the time domain by using Laplace transformations.

Most process models can be estimated by a first order transfer function:

Equation 3: Transfer Function [32]

$$G(s) = \frac{K}{s\tau+1}$$ [32],

where K is the process gain and τ is the system's time constant.

The PID algorithm can be described as:

Equation 4: PID algorithm [29]

$$u(t) = K\left(e(t) + \frac{1}{\tau_i}\int_0^t e(\tau)d\tau + \tau_D\frac{de(t)}{dt}\right)$$[32],

with the controller parameters K = controller gain, τ_i = integral time and τ_D = derivative time.

Since nowadays, control is handled by computer systems, there has to be some means by which to convert these continuous time process dynamics into a discrete time form. The velocity form is simple and easy to implement:

Equation 5: Velocity Form [35]

$$\Delta u(k) = K_c\left[\left(1 + \frac{\Delta t}{\tau_i} + \frac{\tau_D}{\Delta t}\right)\varepsilon(k) - \left(\frac{2\tau_D}{\Delta t} + 1\right)\varepsilon(k-1) + \frac{\tau_D}{\Delta t}\varepsilon(k-2)\right]$$ [35],

where k = discrete time instant, Δt = sampling time, ε = error, K_c = controller gain, τ_i = integral time and τ_D = derivative time

2.3.4 Breadth First Search (BFS)

Artificial Intelligence applications these days cover a wide range of space which is why graph algorithms have become more and more important [36]. State-space representation is the most basic problem representation technique used in AI, yet only a small subset of problems can be

described by it. There are too many possibilities to continue from one point to the next, hence the solutions that have already been visited have to be remembered in order to prevent repetition [37]. Marking ssolutions that have already been visited can only be done, if the developer has additional knowledge about the problem, even though describing this problem in form of a mathematical function causes further challenges [37].

BFS's goal is to visit all nodes on a plane without visiting the same one twice, as this could lead the algorithm going around in circles, and without going backwards [38]. This works by the algorithm dividing up a space into a graph composed of set of vertices. The size of the graph is then defined as the number of vertices. Each vertex is then assigned a neighbour vertex. The algorithm explores all the neighbour vertices in levels starting from that first vertex. The neighbouring vertices are then marked and their level is stored in memory [39].

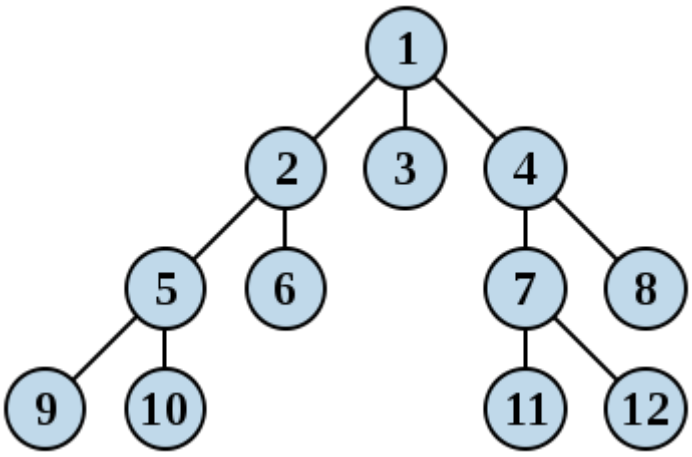


Figure 3: Breadth First Search Levels [40]

In Figure 3 above, the starting vertex would be Number 1 with the neighbouring vertices being number 2, 3 and 4 which then would be saved as situated in level 1. The following level would consist of vertices number 5, 6, 7 and 8 and so forth. The BFS algorithm is often used in robotics and AI for path finding or collision avoidance.

2.3.5 Triangulation Methods

The triangulation methods explained in the following sections are based on the problem of finding a point **x** in space. Two images taken by two cameras with a certain view give the point's position. The intersection point of the known lines (**u** and **u'**) in space has to be found in order to solve this problem [41]. **P** and **P'** are the camera matrices.

2.3.5.1 Linear Triangulation

This is the most common triangulation method described. If $\boldsymbol{u} = P\boldsymbol{x}$ and we write in homogeneous coordinates that $\boldsymbol{u} = w(u, v, 1)^T$ where w is an unknown scaling factor and u and v are the point coordinates observed. If denoting by $\boldsymbol{p}_i^T$ the i-th row of the matrix P can be written as $wu = \boldsymbol{p}_1^T\boldsymbol{x}$, $wv = \boldsymbol{p}_2^T\boldsymbol{x}$, $w = \boldsymbol{p}_3^T\boldsymbol{x}$ [41].

Using the third equation to eliminate w, we get

Equation 6: Linear Equations in Coordinates of x [41]

$$u\boldsymbol{p}_3^T\boldsymbol{x} = \boldsymbol{p}_1^T\boldsymbol{x} \text{ and } v\boldsymbol{p}_3^T\boldsymbol{x} = \boldsymbol{p}_2^T\boldsymbol{x} \text{ [41].}$$

Overall, four equations in the **x** coordinates can be obtained from two views and can be written in the form of $A\boldsymbol{x} = \boldsymbol{0}$ for suitable 4x4 matrices. However, these equations can only define **x** up to a certain scale factor and the solution for **x** should be nonzero. Since there is noise in every data set, the solutions will never be solved completely, yet we seek for the best possible solution. Methods of solving these equations are either the linear eigenvalue approach or the least-squares method [41].

2.3.5.2 Iterative Linear Methods

Some inaccuracy is caused by the lack of geometric meaning of the minimized values in the linear and least-squares methods. The iterative method is changing the weights of the equations in order to achieve correspondence of the weighted equations to the errors [41].

Looking at the first equation in Equation 6, one can see that **x** can never satisfy the equation, it can only come close to the exact solution due to errors in measurement. This error can be described as

Equation 7: Error in Measurement [41]

$$\varepsilon = u\boldsymbol{p}_3^T\boldsymbol{x} - \boldsymbol{p}_1^T\boldsymbol{x}$$ [41].

However, the interesting value to calculate in this situation is the distance between u (measured image coordinate) and the projection of **x**. This can be expressed by $(\boldsymbol{p}_1^T\boldsymbol{x})/(\boldsymbol{p}_3^T\boldsymbol{x})$, and in particular the minimizations of the projected error

Equation 8: Equation to be minimized [41]

$$\varepsilon' = \frac{\varepsilon}{p_3^T x} = u - (\boldsymbol{p}_1^T\boldsymbol{x})/(\boldsymbol{p}_3^T\boldsymbol{x})$$ [41].

If a weighting factor of 1/w, with $w = \boldsymbol{p}_3^T\boldsymbol{x}$, is then applied, the error resulting from this would have matched exactly what is supposed to be minimized [41].

The same weighting factor could then be applied to the other part of Equation 6. The equations cannot be weighted like this though, because **x** is needed and this variable is unknown until the equations are solved. Hence, an iterative approach can be taken until the exact solution $\boldsymbol{x}_0$ is found. This can then be used to calculate the exact weights and be repeated until the process converges. This method can be used in combination with the Eigenvalue- or Least-Squares Method [41].

2.3.5.3 Midpoint Method

Another way to compute position of a point in space is by making use of its midway point. This means finding the midpoint of the common perpendicular of the two projection lines that correspond to the matched points. One can use a linear algorithm to solve the problem at hand making this method a simple one to use [41].

If $P = (M \mid -M\boldsymbol{c})$ is a decomposition the first camera matrix, then the centre is $\begin{pmatrix}\boldsymbol{c}\\1\end{pmatrix}$ in homogeneous coordinates. The point that maps to **u** in the image at infinity is given by $\begin{pmatrix}M^{-1}\boldsymbol{u}\\0\end{pmatrix}$ and hence, any point on the projected line that maps to **u** can be written as $\begin{pmatrix}\boldsymbol{c} + \alpha M^{-1}\boldsymbol{u}\\1\end{pmatrix}$. In non-

homogeneous coordinates, this expression takes the form of $\boldsymbol{c} + \alpha M^{-1}\boldsymbol{u}$, for some α. Using the second image leads to $\alpha M^{-1}\boldsymbol{u} - \alpha' M'^{(-1)}\boldsymbol{u}' = \boldsymbol{c}' - \boldsymbol{c}$ resulting in three equations and two unknowns ($\alpha \; and \; \alpha'$). These two values can be found using a linear least-squares method minimizing the distance squared between the two lines. The midpoint (P_M) between the two lines is hence given by

Equation 9: Midpoint Equation [41]

$$P_M = (\boldsymbol{c} + \alpha M^{-1}\boldsymbol{u} + \boldsymbol{c}' + \alpha' M'^{(-1)}\boldsymbol{u})/2 \; [41]$$

3 Electrical Hardware

3.1 Components and Devices

3.1.1 Arduino Platform

Arduino is an open source prototyping platform that is based on hardware and software and that is designed to be used intuitively. With a world-wide community of students, professionals and programmers that have gathered around the platform, it has a vast amount of accessible knowledge [16].

As it was first designed at Ivrea, the Interaction Design Institute, for students without any background knowledge, it had to adapt to new needs as its community grew. The platform, as well as its software, is open source and hence accessible for everyone to build on [16].

Arduino is the preferred choice amongst users, due to its simple and clear programming environment, with a flexibility that makes it popular amongst advanced users also. It is a cross-platform environment that runs on Mac, Linux as well as Windows operating systems [16].

Arduino's software is called an IDE, which stands for 'Integrated Development Environment' and connects to the hardware in order for the user to upload programs and allow communication with them. It contains a text editor which is used for coding, a toolbar with buttons and a series of menus [16].

The IDE supports third party hardware that can be accessed through libraries and downloaded from the particular hardware provider's website. Its serial monitor can be used to display serial data by using the serial class in the code and setting the right baud rate [16].

3.1.1.1 Arduino Nano

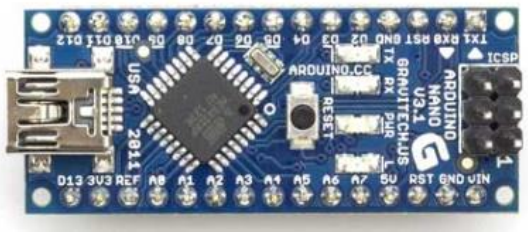


Figure 4: Arduino Nano Board [16]

The Arduino Nano development board (refer to Figure 4) is based on either the ATmega328 or ATmega168 integrated circuit depending on the version in use. Designed and produced by Gravitech, the Arduino Nano is a small and breadboard-friendly package that works with a Mini-B USB cable [16].

Its logic level voltage is 5V and it features 14 digital I/O pins of which six are capable of PWM output. It also comes with eight analogue input pins as well as a clock speed of 16MHz. This microcontroller board features two external interrupts on pins 2 and 3 which can trigger an interrupt on a low value, falling and rising edge, or when a change in value occurs [16].

The Arduino Nano has 16KB of flash memory for the ATmega168 or 32KB for the ATmega328. 2KB of this flash memory is used by the bootloader. It also features a single UART [16].

For a list of the technical specifications, as well as the pin mapping, please refer to the relevant section in the Appendix.

3.1.1.2 Arduino Uno

Figure 5: Arduino Uno Board [16]

The Arduino Uno (refer to Figure 5) platform uses an AVR 8-bit microcontroller (AT168/328) that supplies the development board with 14 digital inputs and outputs. Six of these digital pins can be used as PWM outputs [16]. This board also includes six analogue inputs, a 16MHz quartz crystal and a USB connection. The AVR chip on the board is replaceable, in case of destruction due to incorrect wiring [16].

The AVR chip provides the developer with two external interrupt pins (Digital pin 2 and 3) which can be configured within the IDE to trigger interrupts on a low, falling, rising or changing signal. It also includes PWM functionality on Pins 3, 5, 6, 9, 10 and 11 providing the developer with an 8-bit PWM output function that can be used as an analogue output and configured within the IDE. If SPI is required, the user can connect to Pins 10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK) in combination with the SPI library [16]. Also included are six analogue inputs named A0 through to A5, providing a 10 bit resolution which means a resolution of 1024 levels. By default, all pins measure from Ground up to 5V. It also features a single UART for PC connectivity [16].

For a list of the technical specifications, as well as the pin mapping, please refer to the relevant section in the Appendix.

3.1.1.3 Arduino Mega

Figure 6: Arduino Mega Board [16]

The Arduino Mega development board (refer to Figure 6) is based on an AVR chip microcontroller (ATmega 1280/ATmega2560). It provides the Mega board with 54 digital input and output pins which can be configured within the IDE. 15 of these digital pins can be configured to provide PWM functionality. As well as digital pins, the AVR also provides 16 analog inputs, 4 UARTs (hardware serial ports), a USB connection as well as a 16MHz crystal oscillator, an ICSP header and a power jack. One advantage of the Mega is its compatibility with nearly all of the shields that were designed for the Arduino Uno [16].

The Mega board, as well as the boards mentioned in the previous sections, come with a bootloader that is pre-programmed and communication is made possible via the original STK500 protocol. This allows for uploading of code without the need for an external hardware programmer, yet the bootloader can be bypassed through the ICSP [16].

In order to prevent shorts and overcurrent in the computer the board is connected to, its USB port comes with a resettable polyfuse, which will temporarily disconnect the board once more than 500 mA is applied to it. Power can be applied via the USB connector, but also via an external power supply. The power supply range can be found in the appropriate Appendix section, however if less than 7 V is supplied to the board, the 5 V pin might supply less and the board can therefore become unstable if the operating voltage drops below 4.5 V [16].

Some of the pins have specialized functions and it is worthwhile mentioning the ones needed for this project. The Arduino Mega comes with six external interrupts located on pins 2, 3, 18, 19, 20 and 21. These can be configured to trigger interrupts on a low, rising, falling or changing signal via the IDE [16].

Pins 2 through to 13, as well as pins 44 to 46 can be used as 8-bit PWM outputs providing an analog output capability [16].

The SPI pins are located on pins 50 (MISO), 51 (MOSI), 52 (SCK) and 53 (SS) and are supported with the SPI library available to be downloaded online. All pins usually measure from Ground to 5 V [16].

For a list of the technical specifications, as well as the pin mapping, please refer to the relevant section in the Appendix.

3.1.2 Sensors

3.1.2.1 Ultrasonic Sensors

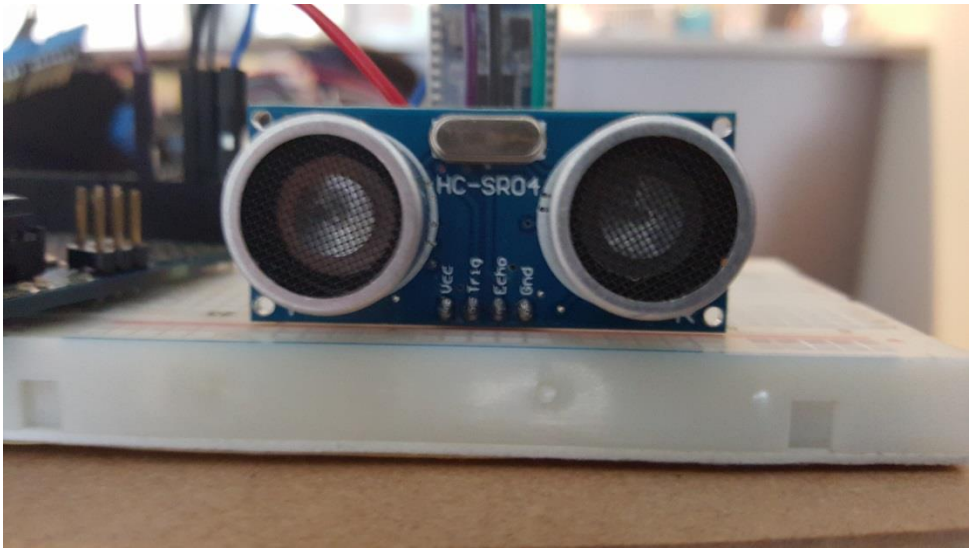


Figure 7: HCSR04 Ultrasonic Sensor [42]

As the name suggests, the ultrasonic sensor HCSR04 [10] (refer to Figure 7) uses sonar to detect the distance to an object and is perfect for non-contact detection. Since its readings are not affected by dark material or sunlight, its readings are stable and accurate. This makes these sensors more

reliable than Sharp rangefinders, even though soft cloth can be hard to detect and therefore interfere with the readings [10].

The HCSR04 module includes a transmitter and a receiver which means that this arrangement will save solder, space and the implementation of one's own protocol. In order to initiate a measurement, the trigger pin of the ultrasonic module has to be set high for at least 10 µS. This, in turn, will send out eight cycles of ultrasonic bursts at a frequency of 40 kHz. The module then waits for the returning, reflected signal [10].

Once the returning signal is sensed, the ultrasonic module's echo pin will be set to 5 V and held there for a period of time that is proportional to the measured distance. The distance can be obtained by either dividing this pulse width by 58 for measurements in centimetres, 148 for a measurement in inches or utilize the speed of sound with a value of $340\,\frac{cm}{s}$ [10]. Slight variations can occur since these numbers are dependent on air temperature and air pressure.

3.1.2.2 IR Sensor

Figure 8: Infrared Sensor [42]

The Infrared Sensor GP2Y0A21KY0F [11] (refer to Figure 8) is a distance measuring device that measures within a range of 10-80 cm. It operates with an integrated combination of an IRED (infrared emitting diode), a PSD (positive sensitive detector) and a signal processing unit [11].

The measurement is presented in form of an analog voltage signal that is inversely proportional to the detection distance. Hence, this sensor can also be used as a proximity sensor. Some of its other applications include, but are not limited to touch-less switches, amusement equipment such as arcade gaming and robot cleaners [11].

For a list of the technical specifications, as well as the pin mapping, please refer to the relevant section in the Appendix.

3.1.2.3 NFR24 Radio

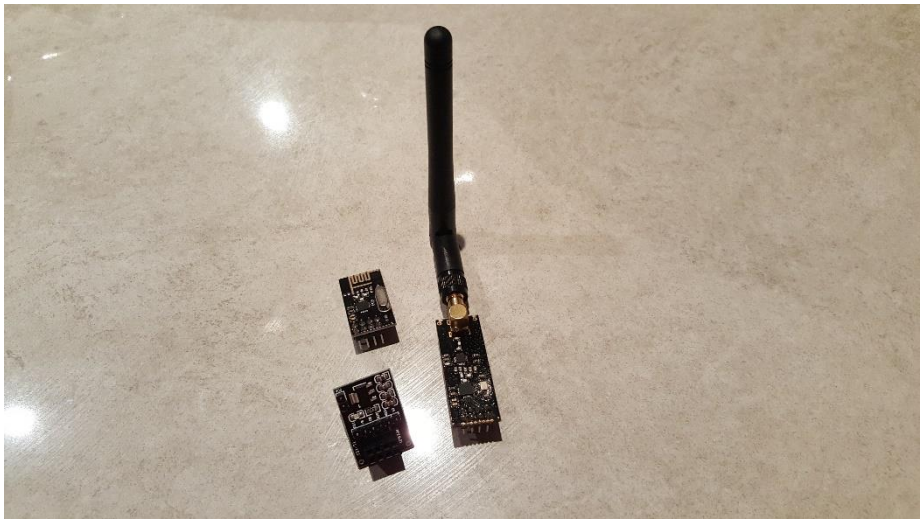


Figure 9: NRF24L01 Radios plus Breadboard Adapter [42]

The nRF24L01 Radio device (refer to Figure 9) consists of a single chip 2.4GHz transceiver. A baseband protocol engine that was designed for low power applications is embedded in this device. The radio was designed to operate in a range between 2.400-2.4835GHz, also called the ISM range. [43]. No license is required to operate on them and their usage is free of cost.

The device's configurations can be accessed and it may be operated via SPI, through which the register map is available. This map holds the configuration registers and can be accessed in all of the operation modes. To ensure a smooth flow from the MCU to the radio's front end and vice versa, internal FIFOs (First In, First Out) are used, which store transmitted or received payloads that are ready to be clocked out. This ensures a smooth data flow between the device's front end and the MCU. The radio front end uses GFSK modulations with parameters such as frequency channel, air rate and output power, which the user can configure [43].

The air data rate can be sped up to 2 Mbps which, combined with two power saving modes, makes this transceiver suitable for designing low power systems [43].

One of the parameters that can be configured is the RF Channel Frequency, which determines the centre of the channel used by the nRF24L01. The channel has a bandwidth of 1 MHz with a speed of 1 Mbps and a bandwidth of 2MHz when the speed is set to 2 Mbps. This means that at 2Mbps the channel's bandwidth is wider than the resolution of the setting [43]. Hence, the channel spacing should be set to 2MHz or wider to ensure that neighbouring channels do not overlap. This is set by the RF_CH register according to the formula $F_0 = 2400 + RF\_CH\ [MHz]$. The transmitting and receiving device should have the same RF Channel setting to be able to communicate [43].

Another implementation that makes this device so reliable is the embedded protocol engine Enhanced ShockBurst™, which is based on packet communication. It is used for automatic packet handling and timing. It clocks the bits in the data pocket during transmit. While receiving, ShockBurst™ continuously searches for a valid address in the signal and once it finds a valid address, it uses CRC to check the validity of the data packet that has been read [43].

It also controls high speed timing and bit handling and handles automatic packet transaction handling that ensures reliability for a bi-directional data link [43].

3.1.2.4 Bluetooth

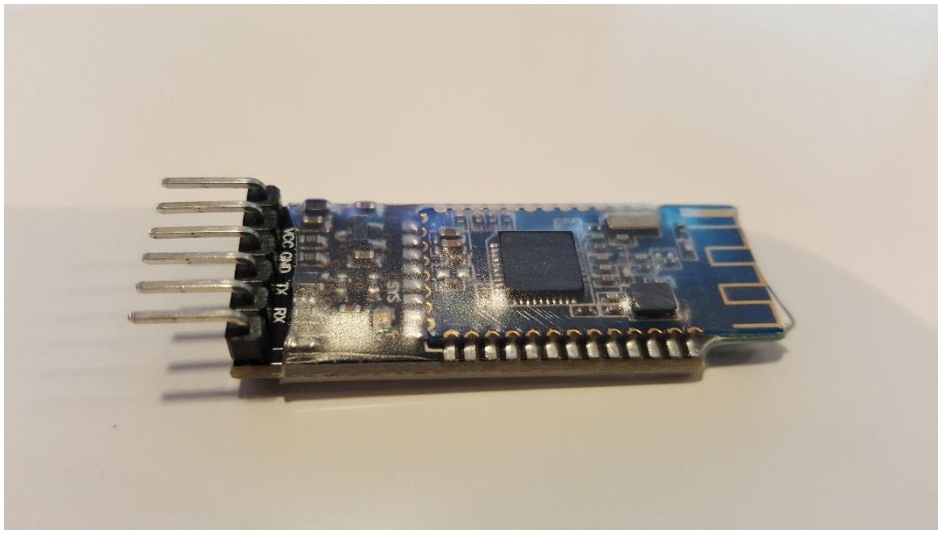


Figure 10: Bluetooth Module [42]

The Bluetooth module (refer to Figure 10) was a last minute purchase from a local electronics store in order to replace the radios. The reader can find more information on this in Section 5.2.3.

The chip on the USB module was used to identify the brand as none of this was advertised in store. The module discussed is the Tinysine Bluetooth 4.0 BLE module with a working frequency of 2.4 GHz [44]. It uses GFSK (Gaussian Frequency Shift Keying) as its modulation method and has an RF power of 0.01-5mW which can be modified using AT commands.

It needs a supply of 3.3 V, but since it is soldered onto a board with an in-built DC-DC converter, a 5 V supply can also be used. This Bluetooth module works up to a range of 100 m [44].

3.1.2.5 LCD Screen

Figure 11: Nokia Screen [42]

This LCD is screen (refer to Figure 11) is made by Nokia and has a resolution of 84x48 pixels on a monochrome LCD display. Its diagonal is about 3.81 cm, but since it was designed with a backlight, it is very readable. It can be used for displaying text, bitmaps or graphics and it is easy to use as well as inexpensive to purchase. The display is low power and only needs a small amount of digital I/O pins to operate [45].

A library can be found online to operate this screen within the Arduino environment, but can also easily be used with other microcontrollers [45].

This display was used in the Nokia3310 and 5110 cell phones and uses a PCD8544 controller chip made by Philips. Since this chip is designed for a 3.3 V supply, microcontrollers that operate on a 5 V logic level will need a logic level shifter in order to not damage the display. When using the backlight, the LCD screen can draw up to 80mA due to four white LEDs drawing 20 mA each [45].

3.1.2.6 Buzzer

To warn the user when the tracking device loses signal, an inexpensive Piezo buzzer was purchased from Adafruit. These elements convert voltage to a vibration or vice versa. This device is rated up to

12 V peak to peak but square waves of 3 V or 5 V will also ensure a loud enough signal. The two wires for voltage supply and ground are already attached.

3.1.3 Drive System

3.1.3.1 Motor Shield

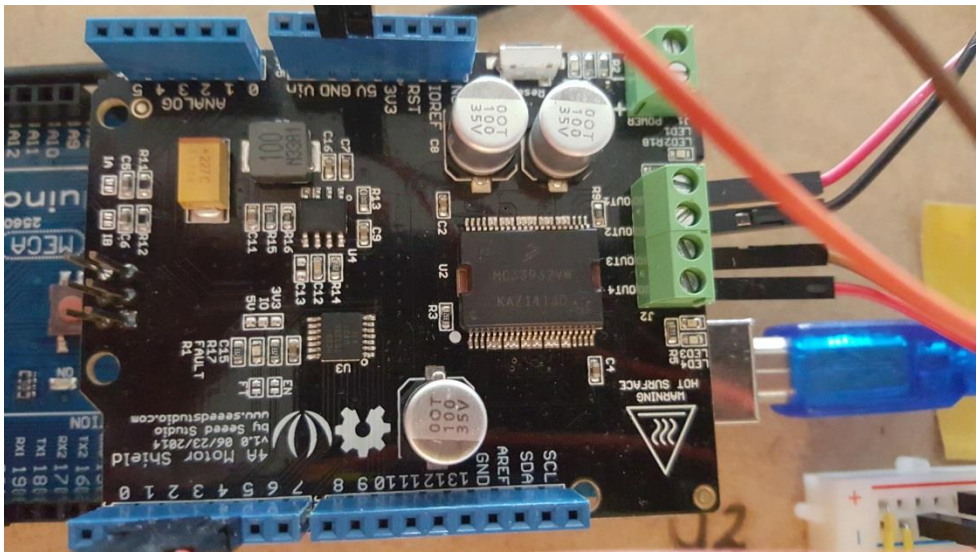


Figure 12: Seed Studio Motor Shield [42]

This motor shield (refer to Figure 12) manufactured by SeeedStudio is based on the Freescale MC33932 dual H-Bridge Power IC. Currents up to 5.0 A per bridge can be controlled with this shield, which is compatible with Arduino boards. Each motor's direction can be controlled independently and measuring the current absorption is one among other features [25].

The shield is powered through the DC input connector and the input power can range from 6-28 VDC. One of the most advantageous features is the shield's on-board DC to DC converter, which allows the user to step the voltage down to 5 VDC and power the microcontroller [25].

The motors can be connected to the pins Out1 and Out2 for one motor and the pins Out3 and Out4 can be used for the second motor. Caution must be taken during operation of the motors, as the PCB board can heat up to 100°C when a full load is applied [25].

The 33932 chip itself consists of two monolithic H-Bridge Power ICs in the same robust and thermally enhanced package. Originally made for electronic throttle control, they can be applied to any low voltage DC motor control system as long as the voltage requirements and limits are met. Its outputs can be switched at frequencies up to 11 kHz [46].

The motor shield has a current feedback feature with a signal output that is suitable for a microcontroller's A/D input and under-voltage, over-temperature and over-current are reported by a status flag register. Its two independent inputs allow the control of voltage polarity in order to control two half-bridge totem pole outputs [46].

A list of the motor shield's specifications can be found in the corresponding Appendix section.

3.1.3.2 Motor & Encoder

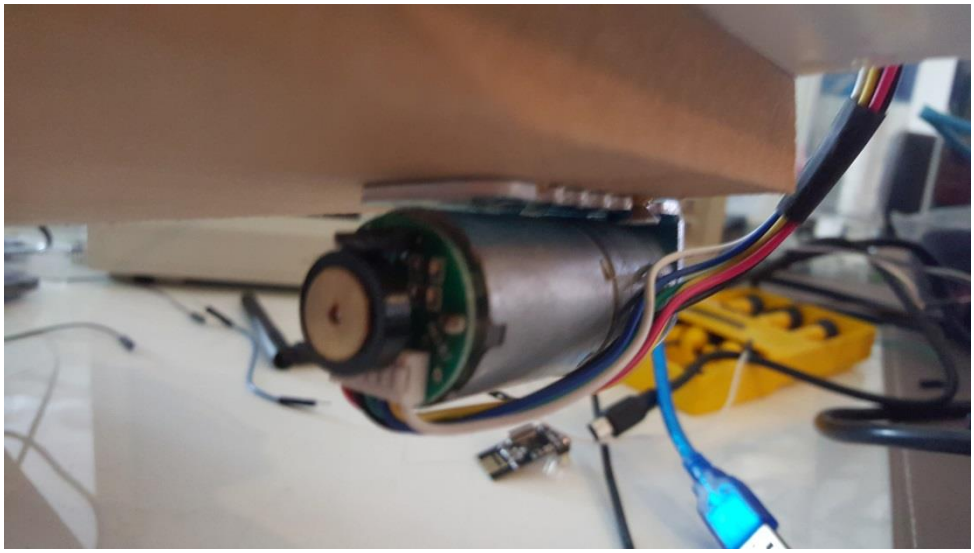


Figure 13: Motor and Encoder on motor bracket [42]

The high-power brushed DC motor (refer to Figure 13) operates on 6 V and comes in combination with a 34.014:1 metal spur gearbox. It is equipped with a 48 CPR cylindrical encoder on the motor's shaft and can provide 1632.672 counts per revolution of the output shaft on the gearbox [47].

The gearmotor has a diameter of 25 mm and its output shaft is D-shaped and has a diameter of 4 mm. It extends 12.5mm from the gearbox's face plate. At 6 V, the motor has 285 RPM with a free-run current of 450 mA and a stall current of 6.5 A [47].

This motor also comes with different power variants in the 6 V range as well as three more power versions in the 12 V range. They all have a diameter case of 25 mm and a 4 mm diameter gearbox shaft. This makes swapping motors easy in case of requirement changes. Not all these models come with the 48 CPR encoder though [47].

In order to mount these motors, a metal gearmotor bracket pair can be purchased which mounts onto the two mounting holes in the motor's face plate using M3 screws. Next to these brackets, a universal aluminium mounting hub can be purchased for 4 mm shafts in order to custom fit bigger wheels to the output shaft of the gearmotor [47].

More wheels can be fitted directly to the output shaft with a hex adapter which is included in the 'Wild Thumper Wheels' package [47].

3.1.3.3 Wheels

Figure 14: Dagu Wild Thumper wheel with adapter [42]

The 'Dagu Wild Thumper' wheels (refer to Figure 14) from Dagu Electronics measure 120 mm in diameter and are primarily made as replacement wheels for robots operating complex and rugged terrain. Their tires are made of spiked rubber for an increase in traction and their adapters make it easy to mount them on motors with 4 mm output shafts [22].

3.1.4 Power

Figure 15: FS Racing 7.2V 1300mAh Ni-MH battery [42]

For the battery on the personal device, a 7.2 V 1300 mAh Ni-MH battery by FS Racing was chosen (refer to Figure 15). This battery can supply a sufficiently high voltage to allow the regulator on the microcontroller to operate and with a capacity of 1300 mAh, it should last long enough to test the tracking system. The main reason for choosing this battery is due to its low cost thanks to student pricing at a local hobby shop, as well as it being readily available.

Figure 16: SIK RC 7.2V 4600mAh Ni_MH battery [42]

For the tracking device, a 7.2 V 4600 mAh Ni-MH battery by SIK RC was chosen (refer to Figure 16).

Again, 7.2 V is enough voltage to supply the microcontroller. More than three times the capacity was chosen for the tracking device, as the motors will drain a lot more current. This battery should last long enough to test the overall system later on.

Again, the choice of this battery was due to its availability at the local hobby shop, as well as its reasonable cost.

Figure 17: LiPro Balance Charger [42]

In order to be able to recharge the batteries, a LiPro Balance Charger (refer to Figure 17) was borrowed from a peer. Unfortunately, it was later found that the FS Racing batteries could not be charged using this device.

The LiPro Balance Charger iMAX B6 includes a high performance microprocessor with software specialized for charging batteries, allowing for a fast recharge rate. It can charge batteries of different types and cell sizes. As the name suggests, it is able to balance the charge each cell receives and hence batteries with multiple cells.

For a list of the specifications, please refer to the appropriate section in the Appendix.

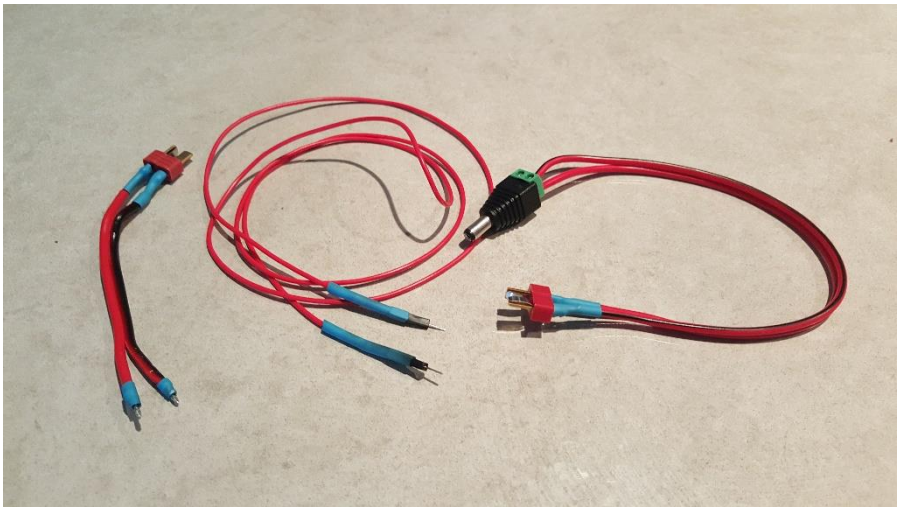


Figure 18: Charging Cables [42]

Since the battery came without leads to attach to the microcontroller or battery charger, a few cables were custom made. Some of these can be seen in Figure 18 above.

4 Design Strategy

The tracking device as well as the personal device have been developed by testing and programming the sensors individually and then modifying the corresponding code to suit the purpose. Each program is written in C++.

The hardware was programmed first, i.e. interrupts for timing were implemented. An example for this would be the interrupt controlling the triggering of the ultrasonic sensors. After this was achieved, the software of what happens when the interrupt is triggered was added. The software was then implemented around these interrupts.

A good example of the individual testing and then the combination of sensor is the development of the ultrasonic and radio transmission code. Here, the distance range was first found using coding and testing the Ultrasonic Sensors. Then the radios were programmed, first for one way communication, then for bi-directional communication. Once communication was established, values from the ultrasonic sensors were sent across.

Once, it could be confidently said that the devices were working, they were transferred to a MDF platform and connected. The MDF platform functions as the body of the device; all major components such as the motors and sensors will be attached to this platform giving the device stability. The entire system was then again tested for correct operation. The actual tracking control was then tested when the robot itself was complete.

For an overview of how the system is set up, please refer to Figure 19.

During the whole process, a pin map was kept in order to remember which pin was used for what purpose.

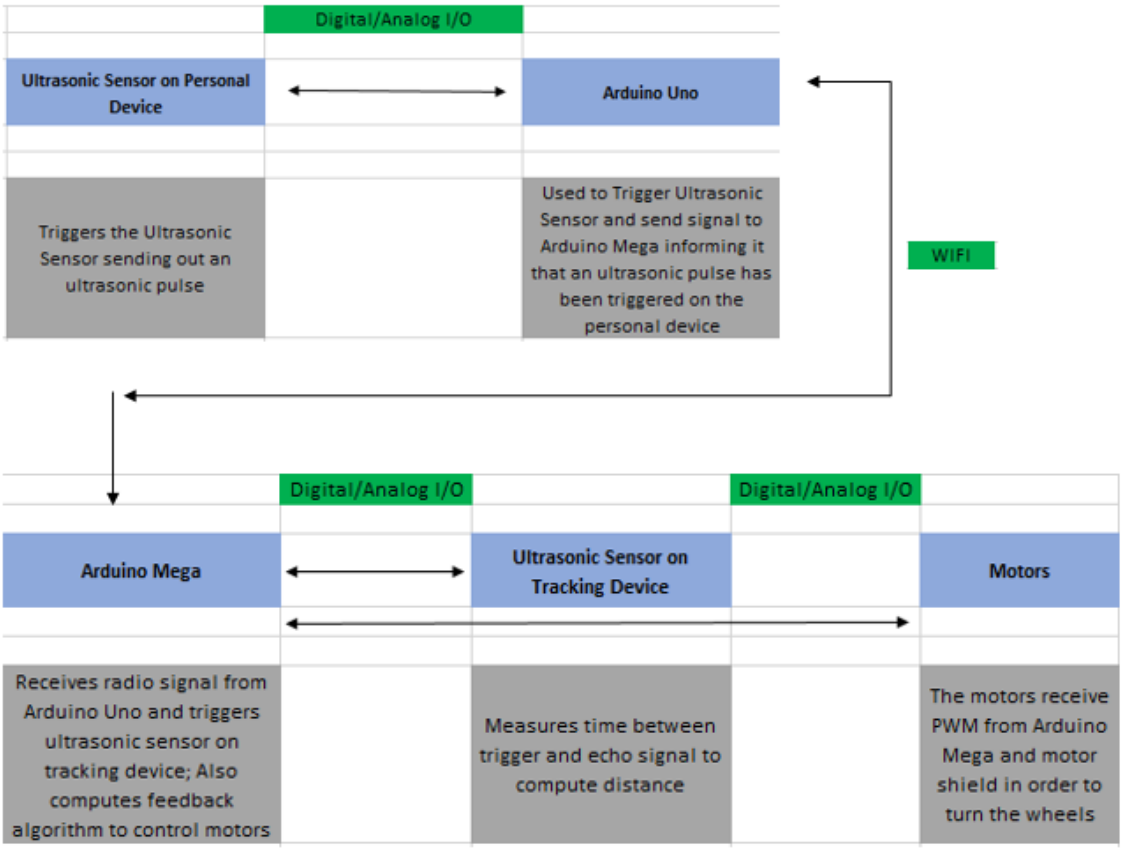


Figure 19: Design Overview [42]

5 Results

5.1 Physical Structure of Device

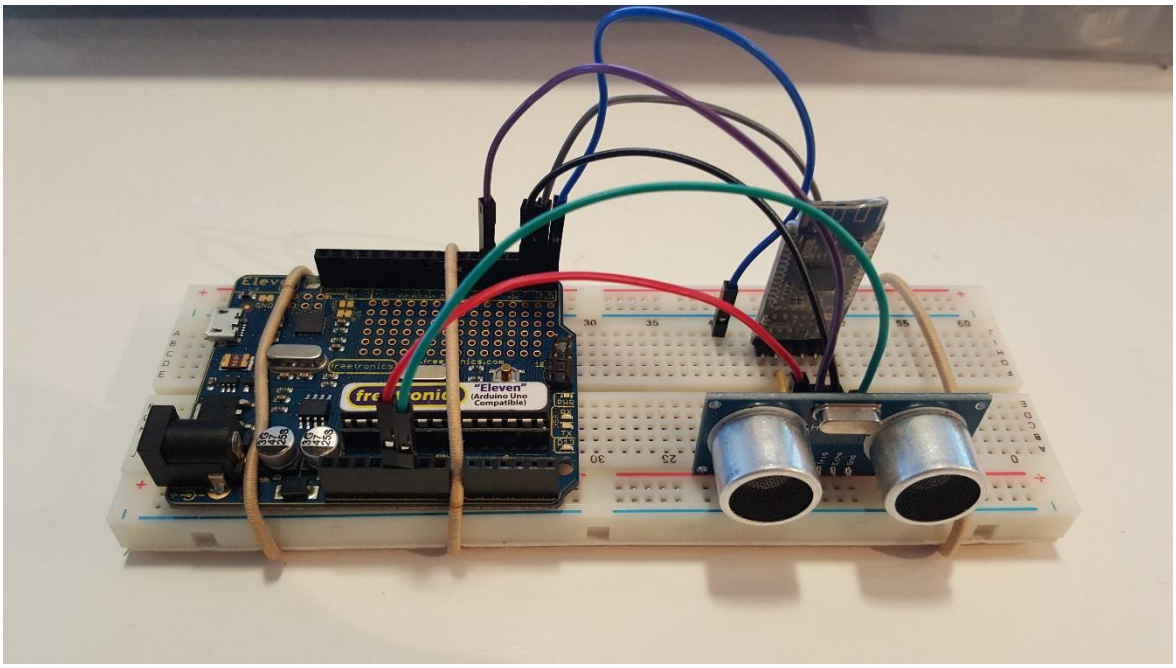


Figure 20: Personal Device [42]

Figure 20 above shows the personal device to date. The reader can see the ultrasonic sensor to the right, the Bluetooth device straight behind it and the Arduino Uno on the left hand side of the breadboard.

For a better handing, it is suggested to solder the sensors onto a shield that can then be attached to the microcontroller and stored on a belt or even on the wearer's ankle.

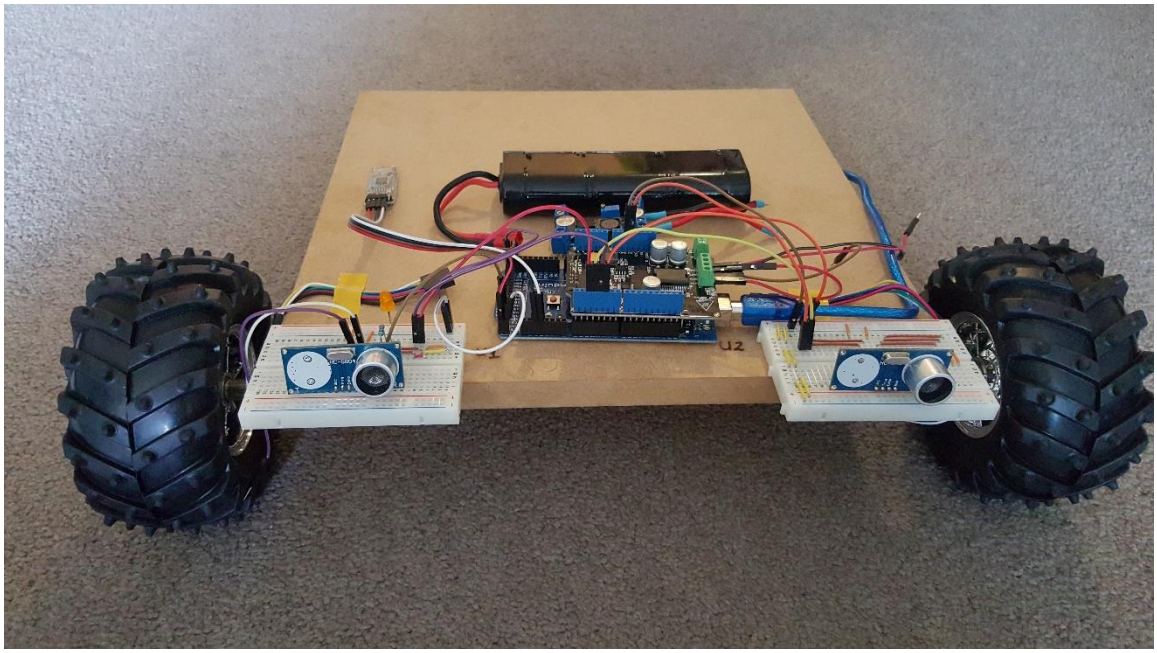


Figure 21: Tracking Device Top [42]

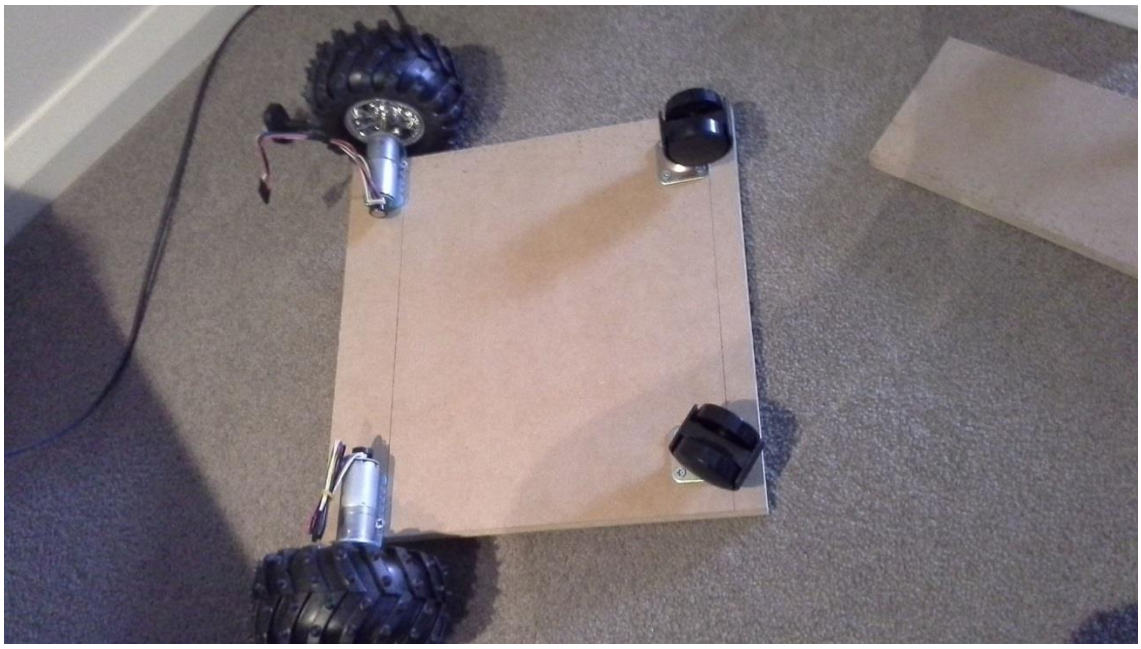


Figure 22: Tracking Device Bottom [42]

Figure 21 and Figure 22 above show the tracking device to date. The platform is made out of MDF material and measures an area of 30cmx30cm. As one can see, the motors are mounted to the underside of the device with the mounts described in Section 3.1.3.2. Two trolley wheels were purchased from a local timber and hardware store to provide free movement and balance of the

device's rear end. All mounts are installed using M3 screws. The wheels are then directly attached to the motors using the 4mm adapters.

Two breadboards were mounted on top of the MDF platform to provide an area on which to attach the ultrasonic sensors and the radio/Bluetooth device. The microcontroller and the motor shield attached to it, sit in between the two breadboards to allow access to all pins from both breadboards. The encoder and motor cables that can be seen in Figure 22 are have been bent around the edges of the MDF board and are now attached to the top of the board, located in close vicinity to the motor shield. In order to minimise wire movement, tape was stuck over the encoder cables holding them in place.

In order to balance the battery's weight evenly across the platform, it was placed in the middle of the board. Its custom-made cables (see Figure 18) are fed into a DC-DC converter to drop the voltage down to 6 V. This supplies the motors with the 6 V they require.

The IR sensors used for collision avoidance are installed behind each wheel on the edge of the device, in order to measure the distance of the object to its surroundings.

5.2 Implementation

These sections will describe the methods used in order to implement the major components of this project.

5.2.1 Ultrasonic Sensors

The Ultrasonic sensors were first tested as a range finding sensor, so the way they operate could be studied. This means that the soundwave that is sent out by the sensor, will have to bounce off an object and return to its source. The time between sending and receiving the time of flight signal can then be measured.

The Arduino's data sheet and multiple online forums were used in order to learn how to use timer interrupts and to find out more about how to implement this system using C++.

In order to measure the time that an ultrasonic sound wave takes to travel between the personal device and the tracking device, a triggering ultrasonic sensor has to be placed on the personal device. On the tracking device, there are to ultrasonic sensors placed on each side of the MDF board (see Figure 21). These are placed as far away as possible from each other to maximise the difference in time it takes for the sound signal to reach each device. This will later on help in the triangulation algorithm.

The ultrasonic sensor works in such a way that its trigger pin has to receive a high for at least 10 μs before it sends out an ultrasonic burst and sets the echo pin high. The echo is then either received by the sensor or times out. If the echo is received, the echo pin resets to a logic level of low [10].

To implement this in the code, an output compare interrupt was first implemented on the personal device triggering its ultrasonic sensor at a frequency of 1Hz (please refer to Figure 23 below). An external interrupt can then be attached to the echo pin to detect a change in signal and use the time it takes for the signal to change to calculate the echo's pulse duration (see Figure 24). The pulse

duration is calculated in microseconds and by dividing this time by the speed of sound, the distance can be calculated.

```
//Set interrupts
OCR1A = 15624;//7812;//15624; //1Hz = 15624 //3Hz = 1302
TCCR1A |= (1 << COM1A1) | (0 << COM1A0); // Compare Output
TCCR1B |= (1 << CS12) | (1 << WGM12) ; // 256 prescaler and CTC
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
```

Figure 23: Timer Output Compare Interrupt [42]

```
//External Interrupt for Receiver
attachInterrupt(digitalPinToInterrupt(2), measureSens1, CHANGE);

void measureSens1()
{
    switch (digitalRead(EchoPinSens1))
    {
        case HIGH:
        endTimeSens1 = 0;
        startTimeSens1 = micros();
        break;

        case LOW:
        endTimeSens1 = micros();
        distanceSens1 = (endTimeSens1-startTimeSens1)/33;
        break;
    }
}
```

Figure 24: External Interrupt Routine for measuring Echo [44]

All interrupts were setup with the help of the Atmel ATmega168P/ATmega328P and the Atmel ATmega2560 datasheets [48, 49].

The triggering was monitored on an oscilloscope in order to verify correct behaviour. Once this was implemented, the ultrasonic sensors on the tracking platform were programmed using the Arduino Mega's equivalent. Since the trigger pin needs to be set high in order to initiate the echo pin, the same interrupt as shown in Figure 23 was once again implemented. This caused the sensors on the

tracking device to pick up their own signal bouncing off the surroundings, so the design choice to remove the transmitter from the ultrasonic sensor was made (see Figure 25).

This still allowed the trigger pin and therefore the echo pin to be initialised without having to worry about interference from unwanted soundwaves.

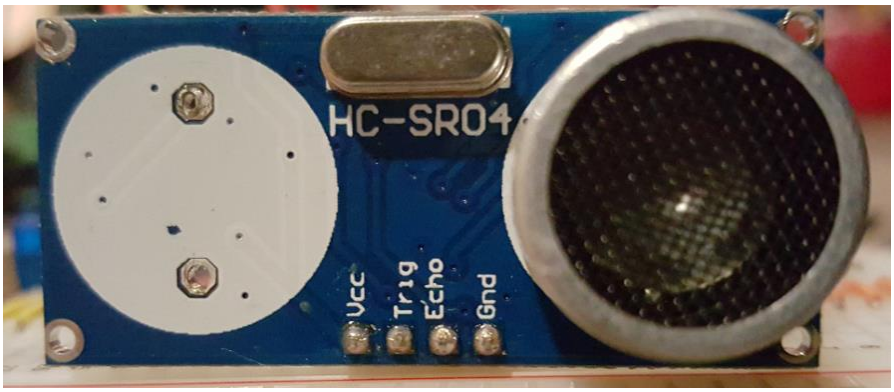


Figure 25: Ultrasonic Sensor with unsoldered Transponder [42]

Unfortunately, when designing the project in the early stages, it was not taken into account that the timing of the triggering could now not be controlled by the same microcontroller. This meant the triggering was not synchronized and hence the echo pins on the tracking device would either time out or pick up the wrong length of signal.

Figure 26 and Figure 27 show the waveform of the triggering signal (trace on upper half of oscilloscope screen) and the waveform of the echo signal (trace on lower half of oscilloscope screen). It can be seen that the ultrasonic sensor starts listening for an echo once the triggering pulse has finished. Figure 26 shows what the echo signal looked like when a correct reading was received. Figure 27, on the other hand, displays what the echo signal looked like when no signal was received.

To work around this, the radio transmitters originally purchased for implementing telemetry, were now used to send a signal from the personal device to the tracking device in order to tell the triggers on the tracking device when the ultrasonic sensor on the personal device was triggered.

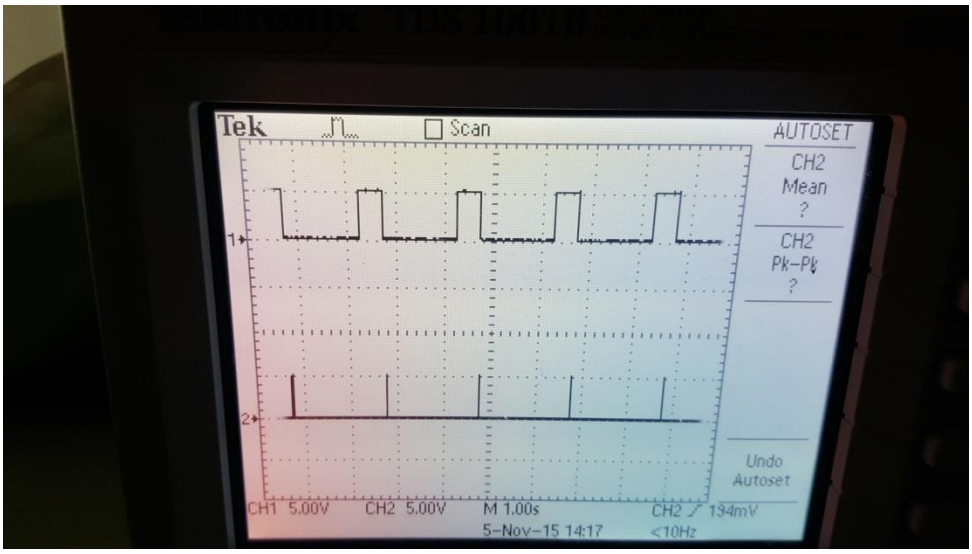


Figure 26: Ultrasound Sensor synchronized [42]

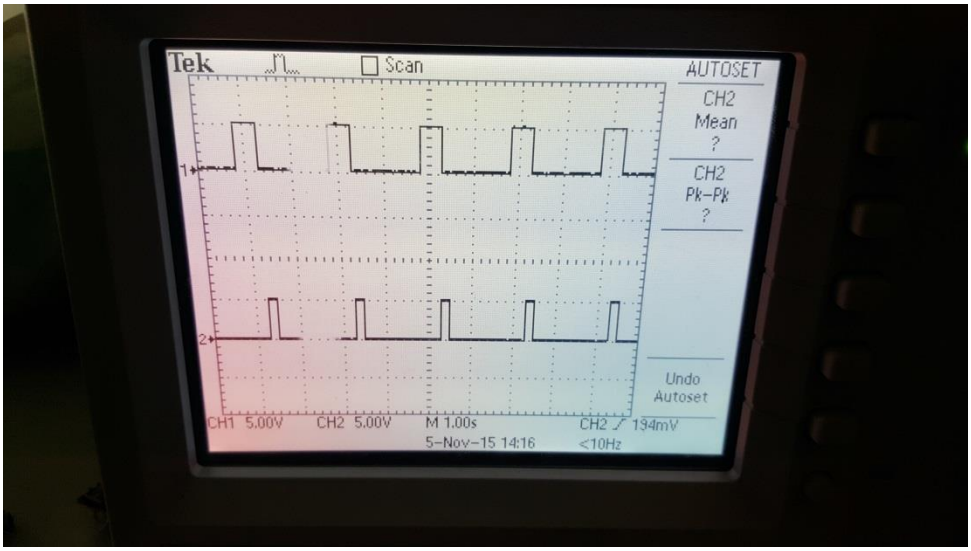


Figure 27: Ultrasonic Sensors not synchronized / Timed out

5.2.2 Radios

The radios were tested prior to their use with the ultrasonic sensors. To implement the radios, an inbuilt SPI and RF24 library was used. This takes advantage on the microcontroller's ability to talk

over the SPI protocol. Please find the required SPI pins for each microcontroller in Table 7 and Table 8 below. This pin assignment can be found in the pinout diagram in Section 0 and Section 8.1.3 [16].

7	CE NRF24L01
8	CSN NRF24L02
11	MOSI NRF24L01
12	MISO NRF24L01
13	SCK NRF24L01

Table 7: Uno Radio Pins

40	CEPin
50	MISO NRF24L01
51	MOSI NRF24L01
52	SCK NRF24L01
53	CSN NRF24L01

Table 8: Mega Radio Pins

To control the speed at which the radios were transmitting data, another compare output interrupt was implemented. The Arduino's Timer 0 was used for that. When the system was then tested, it did not seem to work anymore. It took quite some time and a lot of searching through the libraries installed to find that one of them was using Timer 0 for delays and other internal timing functions. When this timer was used to control the radios, it interfered with the code in the library and hence stopped the radios from working. The solution to this problem was to use Timer 1 instead. After this the program was running fine.

However, to trigger the ultrasonic sensors, the fastest possible speed was desired. This is the reason the interrupt was taken off and the radio was transmitting at its highest rate.

When the trigger of the personal device was then compared to the trigger on the tracking device, it could be seen that there was a delay in transmission causing the echo pulse to time out. In order to further speed up transmission, parameters in the radio's libraries were changed (see Figure 28).

It was attempted to speed up the process by disabling the radio's automatic acknowledge code, by setting the data rate to 2 MPS (as fast as it can possibly transmit data), by disabling the cyclic redundancy check, by setting its power to the maximum and by disabling all retries [50].

```
//Radio Transmit
radio.begin();
radio.setPALevel(RF24_PA_MAX);
radio.setAutoAck(0);
radio.setDataRate(RF24_1MBPS);
radio.setRetries(0,0);
radio.disableCRC();
```

Figure 28: Radio Parameters [42]

Enabling all the delays possible in the underlying library code still did not eliminate all the delay. To work around this problem, the trigger pin signal on the personal device was compared to the trigger pin device on the tracking device on an oscilloscope. A static delay as well as a compounding delay could be detected with the trigger of the tracking device slightly lagging the personal device's signal. The static delay was eliminated by delaying the triggering of the personal device by 52 ms while still sending the radio signal as soon as the compare timer interrupt was triggered.

```
//Interrupt Routines
ISR(TIMER1_COMPA_vect)
{
  testSend = !testSend;
  intReady = 1;
}
```

Figure 29: Interrupt toggling trigger signal [42]

```
void triggerUltra()
{
    if (intReady)
    {
        if (distSet)
        {
            Serial.write(testSend);
            delay(52); //52
        }
        if (testSend)
        {
            PORTD |= B100000;
            intReady = 0;
        }
        else
        {
            PORTD &= ~B100000;
            intReady = 0;
        }

    }
}
```

Figure 30: Triggering Personal Device with Delay [42]

Figure 30 shows the sending of the Boolean which is toggled in the interrupt service routine shown in Figure 29. This will send the high or low signal to the radio on the tracking device, therefore triggering the sensors on the other side. All setting of the pins was done using bit operations, as the 'digitalRead()' and 'digitalWrite()' functions have further delays in the lower levels of coding. This minimised unnecessary delays and hence optimised the microcontroller's operating speed.

It is thought that the compounding delay stems from the signals getting more and more out of sync. It was measured that this delay gains 5ms every three minutes:

Equation 10: Compounding delay calculation

$$\frac{0.005}{3x60} = \frac{0.005}{180} = 28\mu s$$

Since the interrupt to trigger the interrupt was set at 2 Hz which corresponds to 0.5 s, a delay of $14\ \mu s$ was added in each loop in the tracking device's code.

The ultrasonic sensors were then calibrated. During this calibration phase it was noticed that even though the relationship between the values was always linear, the values themselves changed after each microcontroller reset.

Figure 31 shows three lines, with all points having a linear relationship. Each line is the scaled calibration curve of the ultrasonic sensor after a microcontroller reset. One can clearly see that none of the values lie in the same range.

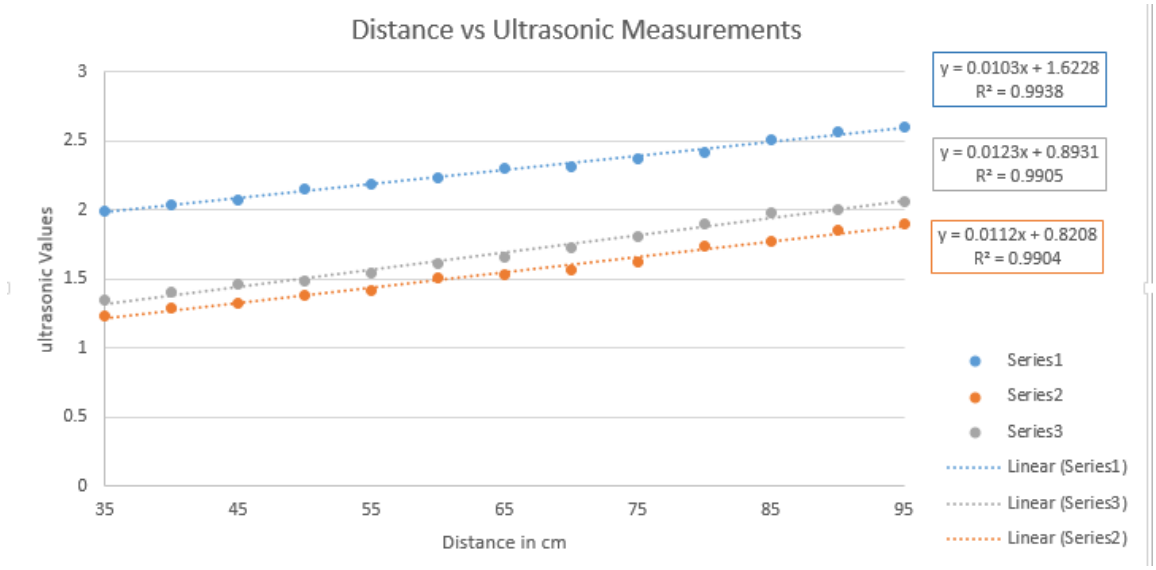


Figure 31: Calibration of Ultrasonic Sensor [42]

In order to work around this issue, code was implemented that first allowed the ultrasonic sensor to sync and then worked out an eight point moving average to minimize noise. It was then attempted to take the first value produced by the moving point average and take it off the following values, so that the curve could be normalized and started at zero. Due to noise, overflow occurred and the number that was supposed to be the initialised zero ended up being the thousand range.

It was therefore decided, to divide the average by 1000 once off and the resulting number was then used to divide every new averaged value by.

E.g.: If the first averaged value was 5200, and it was then divided by 1000, the resulting number would be 5.2. If the every new averaged value is then divided by this number, the result should be a number around 1000. This takes the risk of overflow away and scales the numbers to around 1000 reliably, even after a microcontroller reset.

To ensure the user knows what range they are operating in, the personal device's microcontroller has code in place that ensures that the initial triggers only get send across when the tracking device and personal device are roughly 95 cm apart. This means that the scaled number 1000 then corresponds to a measurement value of 95 cm. Since the calibration curves are linear, further distance calculation can be easily done.

In case of the sensors losing sync, code was implemented that allowed those high time out values to be ignored and push the last useful value through until synchronisation is once again achieved.

This seemed to be working, so the circuit was put onto the MDF board in order to start programming the motors. During the implementation, a short must have occurred in the circuit and damaged the radio on the tracking device.
After some research it was decided to buy two Bluetooth devices at a local electronics store, since no radios were available.

5.2.3 Bluetooth

In order to set up the Bluetooth, the serial monitoring program 'Termite' was used. AT commands can be typed into the serial monitor setting the baud rate, enabling the device to lock onto the first other Bluetooth device it finds and to start sending straight away once it receives power [44].

The approach in using the Bluetooth devices was identical to the approach used for the radios. A signal was sent from the personal device to the tracking device allowing the two devices to

synchronise before distance measurement was initiated. The tracking device received this Bluetooth signal and started listening for an echo. It was found that the using the Bluetooth signal resulted in the sensors losing synchronization quite frequently. It was attempted to reset the microcontroller in order to gain synchronization, but the system was too unreliable to layer a control system on top.

It was therefore decided to use a wire to trigger the tracking device's ultrasonic sensors at the same time as the personal device. This worked perfectly.

5.2.4 Motors

A library was downloaded in order to be able to control the motor shield. Using this library, it became quickly clear, that it was set up completely wrong. The pins used did not correspond to the pins on the motor shield's datasheet. Hence the library was neglected and it was attempted to get the motors running through setting the pins on the datasheet either high or low and using PWM outputs to control the speed of the motors (for code please refer to Figure 32).

```
pinMode(11,OUTPUT);      //SpeedB
pinMode(10,OUTPUT);      //SpeedA

pinMode(6, OUTPUT); //INT
pinMode(7, OUTPUT); //INT
pinMode(8, OUTPUT); //INT4
pinMode(9, OUTPUT); //INT1

analogWrite(9, 0);  //Output1 high
digitalWrite(6, LOW);
digitalWrite(11, LOW);

analogWrite(10, 0); //Output2 high
digitalWrite(8,LOW);
digitalWrite(7, HIGH);
```

Figure 32: Code controlling motors [42]

At this time it was noticed that only one of the motors could be controlled. The other one did not respond even after setting different combinations to high or low or trying to swap the two motors. The reason for this is unknown. Hence, it was decided to implement a control system and monitor the PWM frequency. The results are presented in Section 5.3.

The highest PWM value that can be fed to the motors is the value 127. This is due to the PWM being an 8bit number and half of this number being used to spin the motors backwards. This is not needed in this project and therefore not used.

5.2.5 IR Sensors

The IR sensors were implemented by supplying the required input voltage and reading their analogue output. Their output was then scaled into a voltage and converted into a distance measurement using the calibration curve that can be seen in Figure 33.

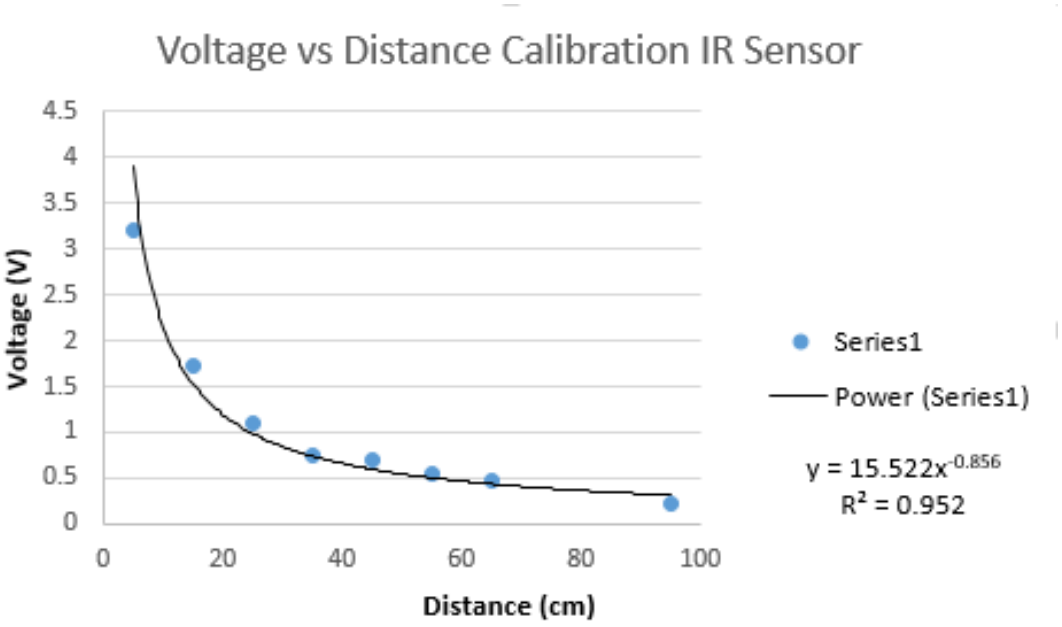


Figure 33: Calibration of IR Sensor [42]

Their functionality was to detect any objects within a certain range of the tracking device. This was then used to stop the motor if anything within a range of 10 cm of the tracking device was detected.

Once the obstacle had passed, the motor could then start again. It was also planned to alert the user of this event by sending a signal over the radios and triggering the buzzer on the personal device.

5.2.6 Encoders

The encoder's code uses more external interrupts. This was the main reason why the decision was made to upgrade from the Arduino Uno, originally chosen for the tracking device, to the Arduino Mega.

The code in Figure 34 compared two pins. If both of them are set to the same logic level, the motors are spinning forward and the rotation value is increased. It the pins do not have the same logic level, the motors are spinning backwards and the rotation value is decreased (Idea from the Arduino's online 'Playground' [16]).

Since the encoders attached to the motors are 48CPR encoders, 48 rotation value increased in the code stand for one full rotation of the motor.

By counting the amount of rotations over a minute, one should then be able to calculate the RPM value. This was not implemented in this project due to time restrictions.

```
ISR (PCINT0_vect)//handle pin change interrupt for d8 to D13, triggered by
{
    if( bitRead(PINB,0)== bitRead(PINB,1)) //compare Pin 50 and 51
    {
        encoderValue++;
    }
    else
    {
        encoderValue--;
    }
}

ISR (PCINT2_vect)// handle pin change interrupt for d8 to D13, triggered by
{
    if( bitRead(PINC,2) == bitRead(PINC,3)) //compare Pin 52 and 53
    {
        encoder2Value++;
    }
    else
    {
        encoder2Value--;
    }
}
```

Figure 34: Interrupt Code for Encoders [42]

5.2.7 LCD Screen

In order to be able to implement telemetry, an Adafruit LCD screen was programmed (the Nokia screen took too long to be delivered) using Adafruit's LCD screen libraries, as well as the Arduino's 'Wire.h' and 'SPI.h' libraries. In combination with the ultrasonic sensors the distance range could be converted into a level indicator as well as a number displayed on the LCD screen. The code for this can be seen in Figure 35.

```
void loop()
{
    dist = echo_duration / 58;
    thick(dist);
    text(dist);

    //Serial.println(echo_duration / 58);
    delay(20);

}

void thick( int l)
{
    if (l > prev)
    {
        for (int h = prev; h < l; h++)
        {
            for (int i = 0; i<16; i++)
            {
                display.drawPixel(h, i, WHITE);

            }
            display.display();
            delay(1);
        }
    }
    else if(l < prev)
    {
        Serial.println("l < prev");
        for (int a = prev; a > l; a--)
        {
            for (int b = 0; b<16; b++)
            {
                display.drawPixel(a, b, BLACK);

            }
            display.display();
            delay(1);
        }
    }
    else
    {

    }
    prev = l;
}
```

Figure 35: LCD Screen Code [42]

5.3 Results & Discussion

In this Section, the functioning of the control system will be presented and discussed.

In order to be able to implement a PID controller into a microcontroller, the velocity form was used as described in Equation 5. The setpoint was compared to the distance measured by the ultrasonic sensors. That error was then used to calculate the PWM value that would be fed to the motor shield in order to adjust the speed of the motors.

Since the Velocity form requires three error and process variable values from the current time step as well as the two previous time steps, the values were stored in an array. It was then possible to call the values needed from the previous time steps.

It can also be noted that the array u[] is a signed integer, in order to allow calculations with negative errors. It is then converted into an unsigned integer and mapped into the motor's PWM input range of 0-127.

```
void speedPID()
{
    y[0] = uint16_t(distanceSens1);
    e[0] = setpoint - uint16_t(distanceSens1);
    y[1] = y[0];
    e[1] = setpoint - y[1];
    y[2] = y[1];
    e[2] = setpoint - u[2];

    setpoint = 40;
    error = setpoint - uint16_t(distanceSens1);
    u[1] = Kc*(1+(deltaT/taui)+(taud/deltaT))*e[0] - ((2*taud/deltaT)+1)*e[1] - e[2] + u[0];
    transform = uint16_t(u[1]);
    VelocityForm = map(transform, -32767, 32768, 127, 0);
}
```

Figure 36: Velocity form implementation [42]

The parameters were left as Kc = 1, taui = 1 and taud =1 since proper tuning was not possibly due to the motor shield failure. For a reference to the implemented code, please refer to Figure 36. Nonetheless, by looking at Figure 37 and Figure 38 it can be seen that the PWM value increases as

the distance value moves above from the setpoint value. One can also see that the PWM decreased when the distance is smaller than the specified setpoint.

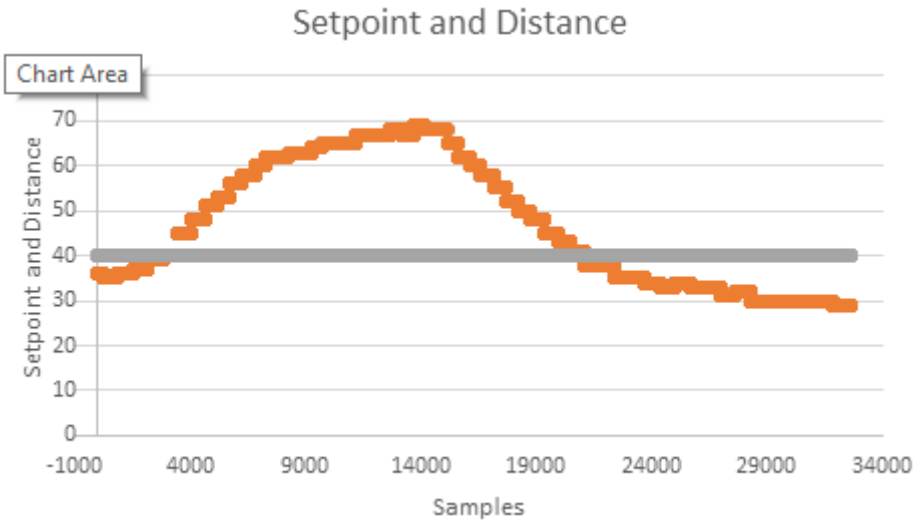


Figure 37: Setpoint and Distance Curves [42]

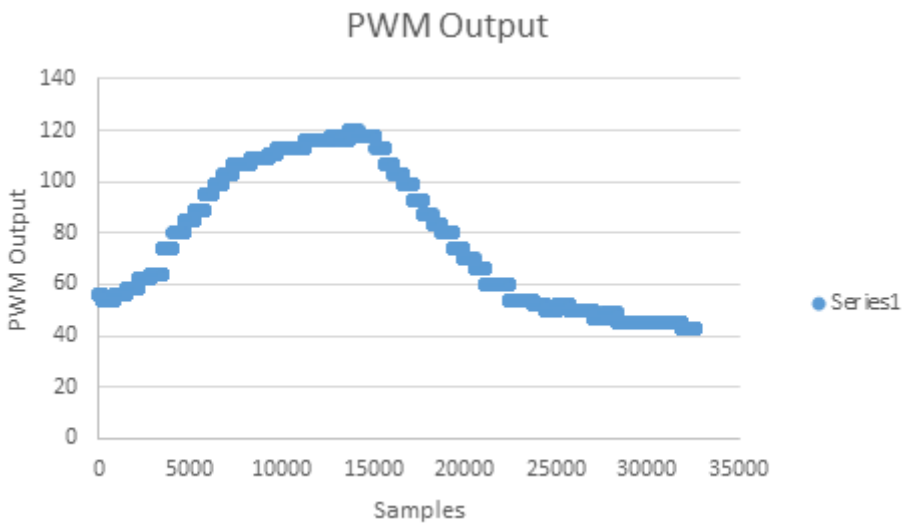


Figure 38: PWM Output Curve [42]

It can therefore be concluded that the implementation of a working PID controller is definitely achievable with a working drive system.

In order to implement direction control, the midpoint method discussed in Section 2.3.5.3 was implemented in the code (see Figure 39) using the method described in Equation 9.

```
void triangulation()
{
    differenceSensors = ((distanceSens1)) - ((distanceSens2));
    yT1[0] = (differenceSensors);
    eT1[0] = setpointT - (differenceSensors);
    yT1[1] = yT1[0];
    eT1[1] = setpointT - yT1[1];
    yT1[2] = yT1[1];
    eT1[2] = setpointT - uT1[2];

    yT2[0] = (differenceSensors);
    eT2[0] = setpointT - (differenceSensors);
    yT2[1] = yT1[0];
    eT2[1] = setpointT - yT1[1];
    yT2[2] = yT1[1];
    eT2[2] = setpointT - uT1[2];

    setpointT = 0;

    errorT1 = setpointT - (differenceSensors);
    errorT2 = setpointT - (differenceSensors);

    uT1[1] = KcT1*(1+(deltaTT1/tauiT1)+(taudT1/deltaTT1))*eT1[0] - ((2*taudT1/deltaTT1)+1)*eT1[1] - eT1[
    uT2[1] = KcT2*(1+(deltaTT2/tauiT2)+(taudT2/deltaTT2))*eT2[0] - ((2*taudT2/deltaTT2)+1)*eT2[1] - eT2[

    transformT1 = uint16_t(uT1[1]);
    transformT2 = uint16_t(uT1[1]);

    VelocityFormT1 = map(transformT1, -32767, 32768, 0, 127);
    VelocityFormT2 = map(transformT2, -32767, 32768, 127, 0);
}
```

Figure 39: Triangulation Code [42]

The difference between the two sensors was calculated and that error was fed into the Velocity form. The Velocity form's output is then again transformed into the PWM range, but one of the motors' range is inversed in order to achieve one motor to slow down while the other one speeds up, hence achieving a change in the device's direction.

In Figure 40, Figure 41 and Figure 42 the distance sensor values, the error curve and the PWM compensation curve can be seen, respectively. In the error graph, it can be observed when the personal device is moving to one side of the tracking device, the other side of the tracking device or when it is at the midpoint. Referring to the PWM output graph, it can be seen that one of the curves follows the error. This would be the motor that is trying to make up for a distance measurement that is bigger than the setpoint value. The other motor curve moves in the opposite direction of the error

curve, which means that this distance value is smaller than the specified setpoint value. The motor then compensates by reducing its speed.

Even though none of these controllers could be tuned and tested on the tracking device, it is obvious that the difference in signal between those two motors is big enough to allow the tracking device to be controlled by midway point triangulation.

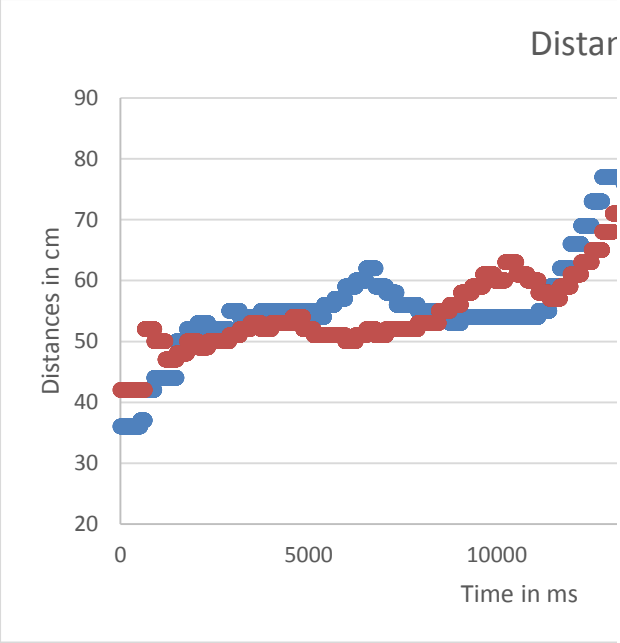


Figure 40: Distance Value Curves [42]

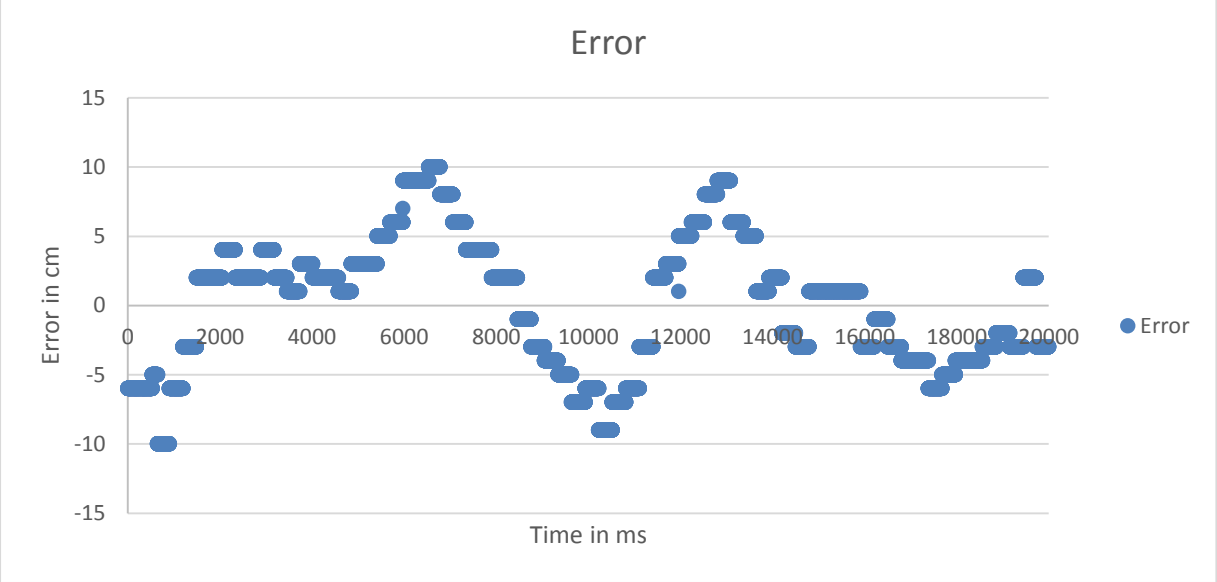


Figure 41: Error Curve [42]

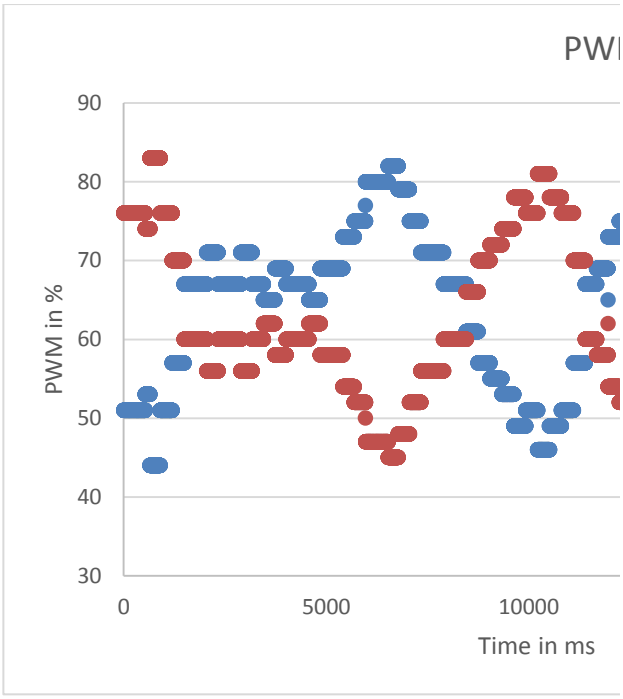


Figure 42: PWM Output of Motor 1 and Motor 2 [42]

6 Recommendation & Future Improvements

For future work, it is recommend choosing different sensors to measure the distance between the tracking device and the personal device. Too much time was lost trying to synchronize the trigger pins in order to get reliable readings. An IR beacon, in combination with IR detection sensors, is highly recommended as they measure the strength of a light signal received and do not need triggering or timing. They could then be used in a similar way as the ultrasonic sensors were used in this thesis, i.e. by calculating the midway point of their distance readings.

Moreover, the motor shield should be swapped to one that come with a correct datasheet. The time trying to get this one to work could have been used to improve the triangulation algorithms.

In addition, it would be helpful for the user to get some information on the status of the tracking device. Battery voltage, distance measurements and the buzzer need to be implemented on the personal device.

A more sophisticated collision avoidance algorithm could be researched and tried to implement in future works.

7 Conclusion

Concluding this report, it can be said that with a working motor shield it is possible to use a triangulation algorithm in order to track a person. This system could then be applied to a suitcase. With different distance measurement sensors, achieving autonomy would be a lot easier. On top of that, additional time might have made the synchronization of the ultrasonic sensors using the Bluetooth devices possible.

The objective of building a platform on which the tracking system could be installed was achieved, as well as the objective of collision avoidance.

Even though, running the device autonomously was not achieved, proofing the concept of using triangulation to track a person has still been achieved.

8 Appendix A – Technical Specifications

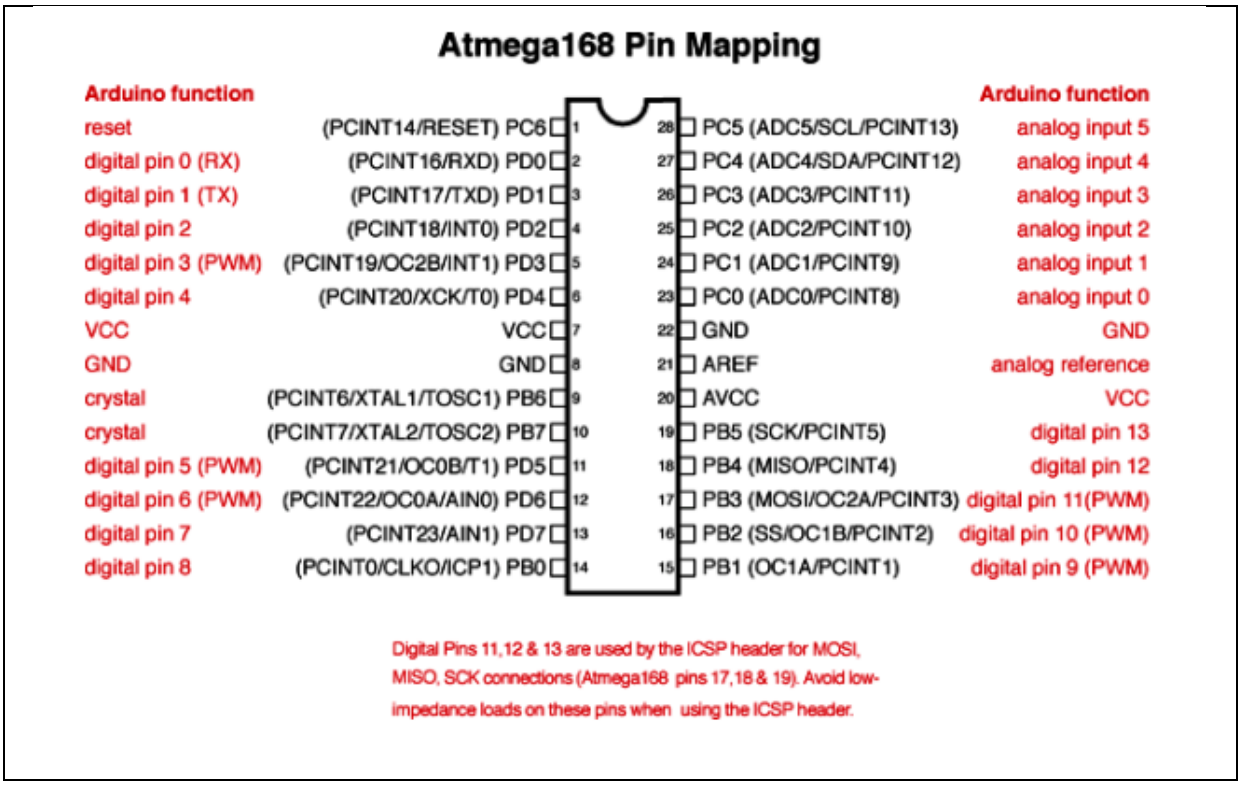
8.1 Arduino Platform

8.1.1 Arduino Nano

Specifications:

Microcontroller	Atmel ATmega168 or ATmega328
Operating Voltage (logic level)	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40mA
Flash Memory	16KB (ATmega168) or 32KB (ATmega328) of which 2KB used by bootloader
SRAM	1KB (ATmega168) or 2KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1KB (ATmega328)
Clock Speed	16MHz
Length	45mm
Width	18mm
Weight	5g

Pin Mapping:

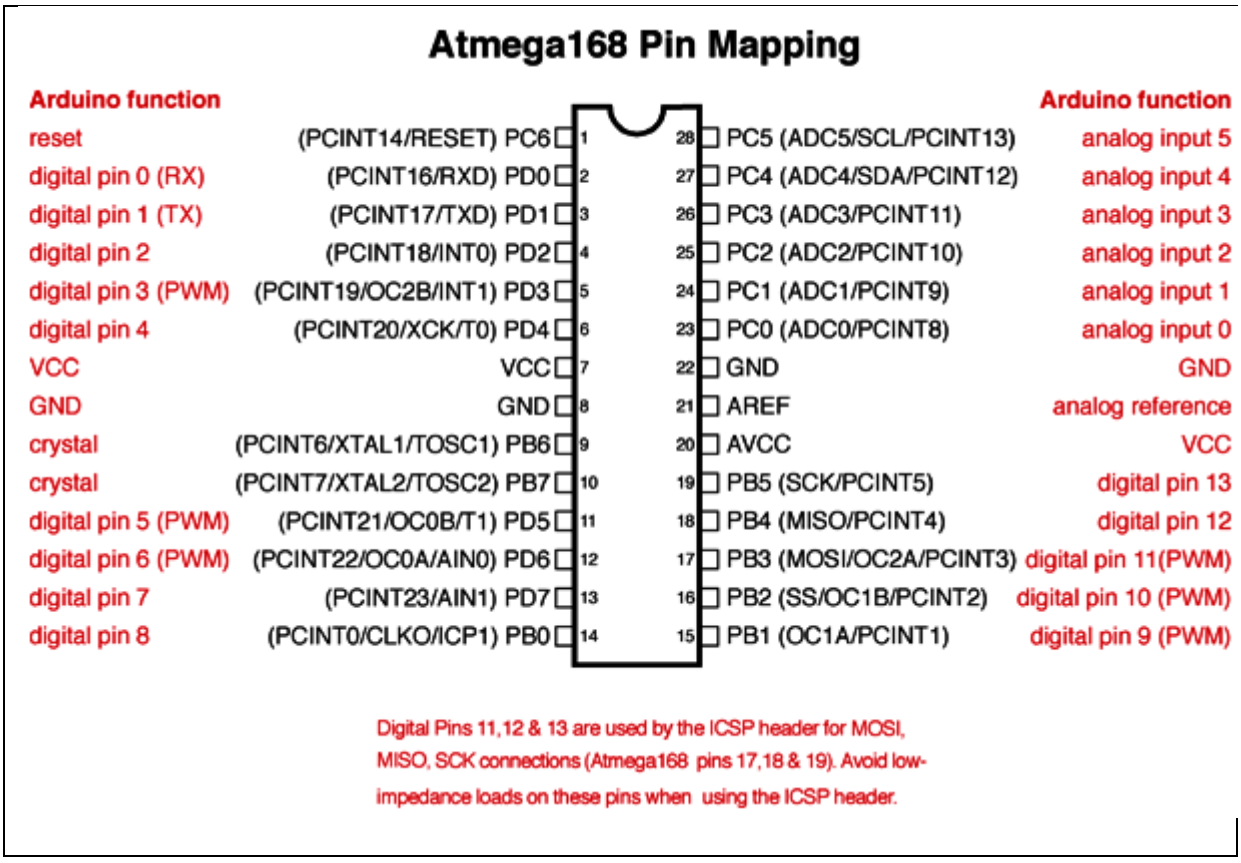

Atmega168 Pin Mapping
Arduino function
reset
digital pin 0 (RX)
digital pin 1 (TX)
digital pin 2
digital pin 3 (PWM)
digital pin 4
VCC
GND
crystal
crystal
digital pin 5 (PWM)
digital pin 6 (PWM)
digital pin 7
digital pin 8
(PCINT14/RESET) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 (SS/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)
Arduino function
analog input 5
analog input 4
analog input 3
analog input 2
analog input 1
analog input 0
GND
analog reference
VCC
digital pin 13
digital pin 12
digital pin 11(PWM)
digital pin 10 (PWM)
digital pin 9 (PWM)
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

8.1.2 Arduino Uno

Specifications:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20mA
DC Current for 3.3V Pin	50mA
Flash Memory	32KB (ATmega328P) of which 0.5KB used by bootloader
SRAM	2KB (ATmega328P)
EEPROM	1KB (ATmega328P)
Clock Speed	16MHz
Length	68.6mm
Width	53.4mm
Weight	25g

Pin Mapping:

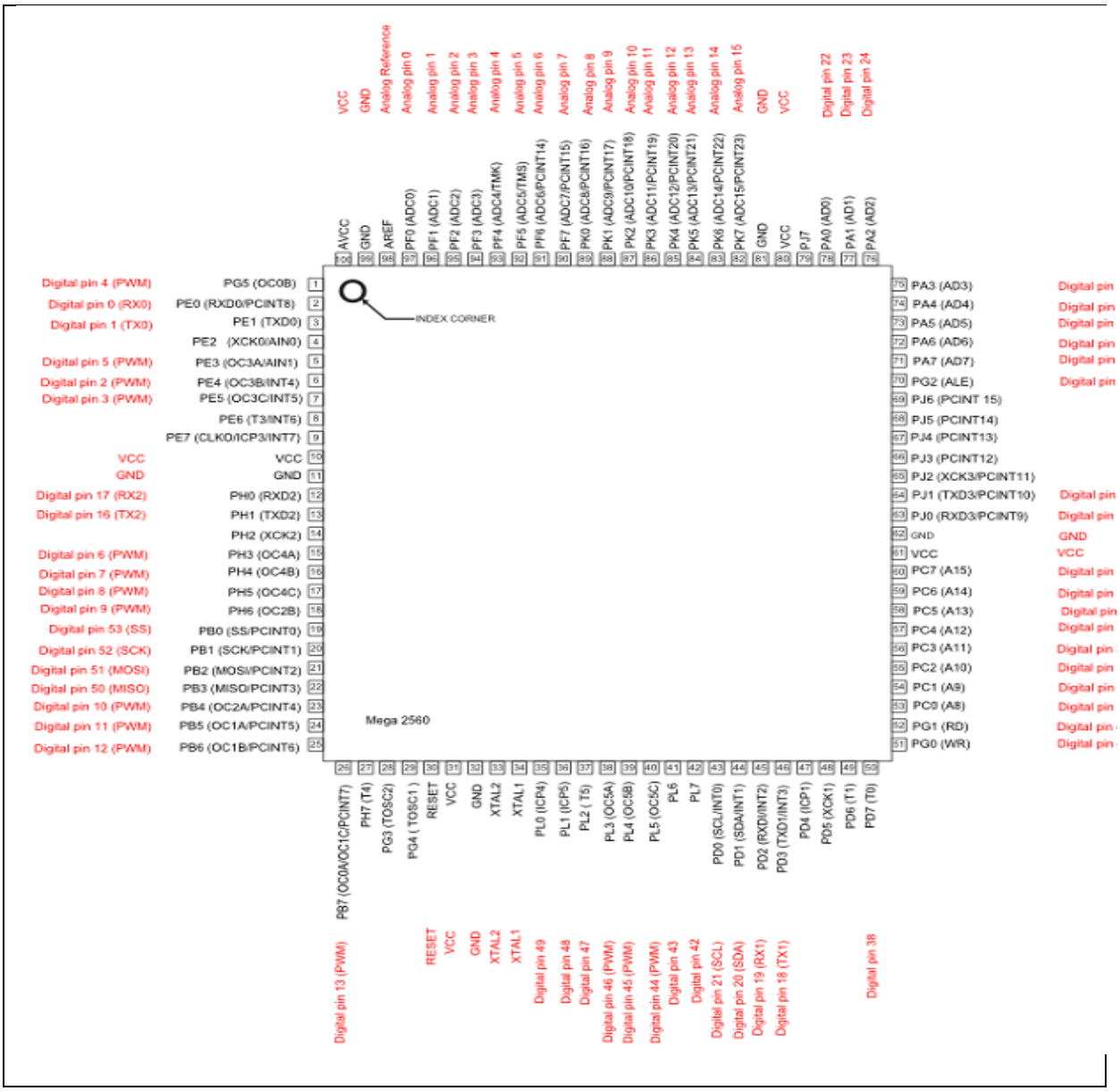

Atmega168 Pin Mapping
Arduino function
reset
digital pin 0 (RX)
digital pin 1 (TX)
digital pin 2
digital pin 3 (PWM)
digital pin 4
VCC
GND
crystal
crystal
digital pin 5 (PWM)
digital pin 6 (PWM)
digital pin 7
digital pin 8
(PCINT14/RESET) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 (SS/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)
Arduino function
analog input 5
analog input 4
analog input 3
analog input 2
analog input 1
analog input 0
GND
analog reference
VCC
digital pin 13
digital pin 12
digital pin 11(PWM)
digital pin 10 (PWM)
digital pin 9 (PWM)
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

8.1.3 Arduino Mega

Specifications:

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Inputs Pins	16
DC Current per I/O Pin	20mA
DC Current for 3.3V Pin	50mA
Flash Memory	256KB of which 8KB used by bootloader
SRAM	8KB
EEPROM	4KB
Clock Speed	16MHz
Length	101.52mm
Width	53.3mm
Weight	37g

Pin Mapping:



8.2 Sensors

8.2.1 Ultrasonic Sensor HCSR04

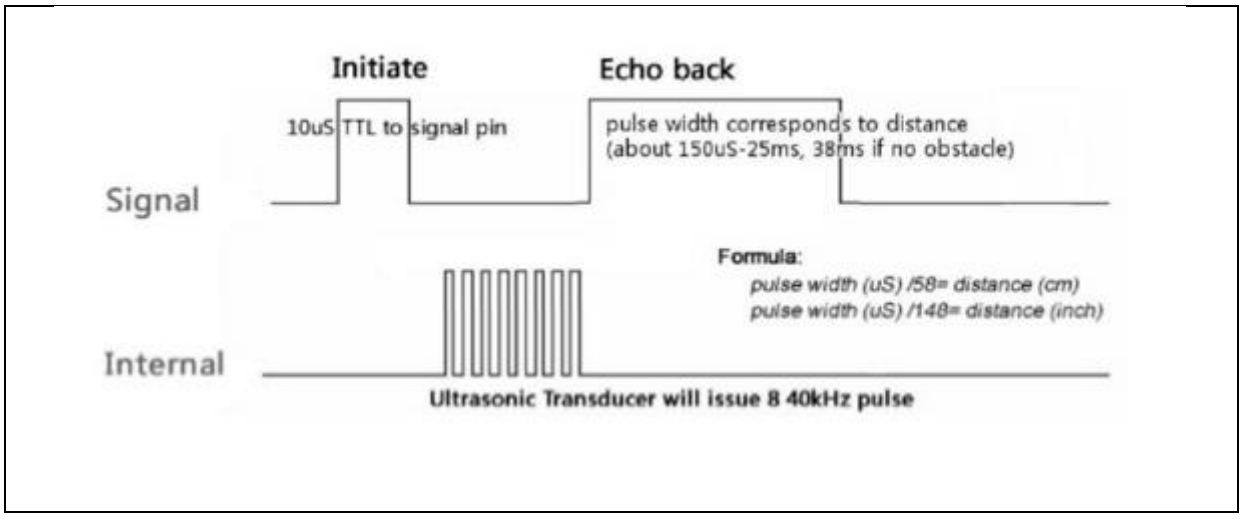
Features:

Power Supply	+5V DC
Quiescent Current	< 2mA
Working Current	15mA
Effectual Angle	< 15°
Ranging Distance	2cm – 400cm / 1” – 13ft
Resolution	0.3cm
Measuring Angle	30 degree
Trigger Input Pulse Width	10µS
Dimension	45mm x 20mm x15mm

Specifications and Limitations:

Parameter	Min	Typ.	Max	Unit
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz

Timing:


Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

8.2.2 IR Sensor

Absolute Maximum Ratings:

Parameter	Symbol	Rating	Unit
Supply Voltage	V_{cc}	-0.3 to +7	V
Output terminal voltage	V_o	-0.3 to Vcc+0.3	V
Operating Temperature	T_{oper}	-10 to +60	°C
Storage temperature	T_{stg}	-40 to +70	°C

Electro-optical Characteristics:

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Average Supply Current	I_{cc}	L = 80cm (Note1)	-	30	40	mA
Distance measuring	ΔL	(Note1)	10	-	80	cm
Output Voltage	V_o	L = 80cm (Note1)	0.25	0.4	0.55	V
Output voltage differential	ΔV_o	Output voltage difference between L=10cm and L=80cm (Note1)	1.65	1.9	2.15	V

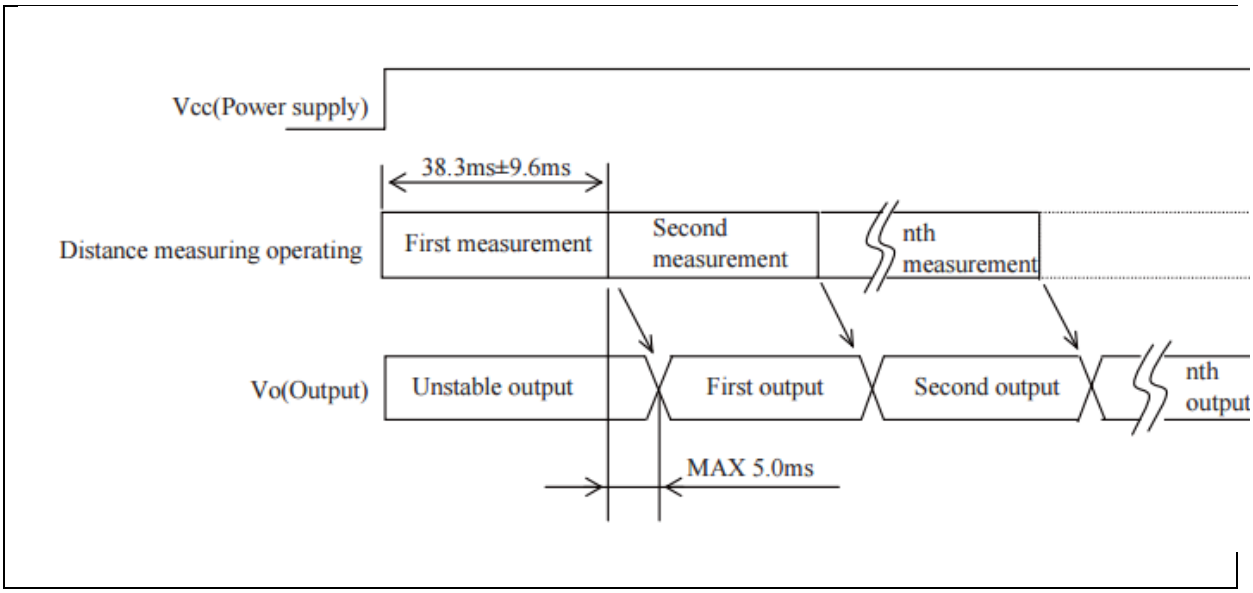
Note1: When using a reflective object such as white paper

All measurements assume T=25°C and $V_{cc} = 5V$

Recommended operating conditions:

Parameter	Symbol	Rating	Unit
Supply Voltage	V_{cc}	4.5 to 5.5	V

Timing:


Vcc(Power supply)
38.3ms±9.6ms
Distance measuring operating
First measurement
Second measurement
nth measurement
Vo(Output)
Unstable output
First output
Second output
nth output
MAX 5.0ms

8.3 Drive System

8.3.1 Motor Shield

Specifications:

Operating Voltage	6V-28V
DC/DC output	5V 100mA @ “5V” Pin
Output current (For each channel)	2A (continuous operation) / 5A (peak)
Output Duty Range	0%-100%
Output short-circuit protection (short to VPWR or GND)	
Over-current limiting (regulation via internal constant-off-time PWM	
Temperature dependant current limit threshold reduction	

8.3.2 Motors and Encoders

Dimension:

Size	25D x 64L mm
Weight	101g
Shaft Diameter	4mm

General Specifications:

Gear Ratio	34.014:1
Free-run speed @ 6V	285RPM
Free-run current @ 6V	450mA (250mA without gearbox)
Stall current @ 6V	6500mA
Stall torque @ 6V	60oz-in (0.42Nm)
Lead Length	8in (20.32cm)
Motor Type	6.5A stall @ 6V

8.3.3 Wheels

Dimensions:

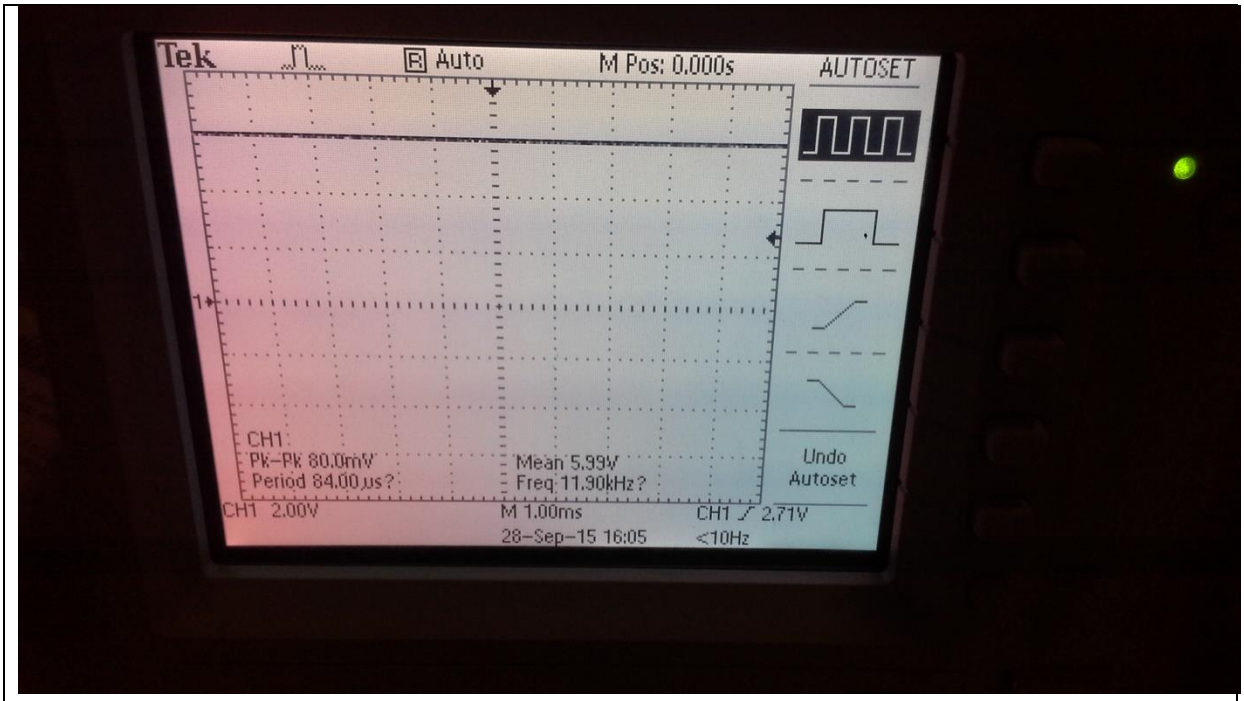
Size	120x60mm
Shaft Diameter	4mm

8.4 Power

8.4.1 LiPro Balance Charger

Operating Voltage Range	11-18.9VDC
AC to DC adapter	11-18V/5S
Circuit Power	Max. 50W for charging, Max. 5W for discharging
Charge Current Range	0.1-5.0A
Discharge Current Range	0.1-1.0A
Current drain for balancing Li-Po	300mAh/cell
NiCd/NiMH battery cell count	1-15cell
Li-ion/Polymer cell count	1-6 series
Pb battery voltage	2-20V
Weight	277g (NetWeight)
Dimensions	133x87x33mm

8.5 Physical Structure of Device



6V Output of DC-DC converter at Motor Shield Inlet

9 References

[1] F. L. Lewis, *Autonomour Mobile Robots*. Boca Raton, FL: CRC/Taylor & Francis, 2006.

[2] Rio Tinto, "Rio Tinto improves productivity through the world's largest fleet of owned and operated autonomous trucks," ed, 2014.

[3] Gaurav, "Hop automatic moving suitcase follows you wherever you go," in *DamnGeeky* vol. 2015, ed, 2012.

[4] R. G. Gonzalez. (2012, 4/11/2015). *Hop! The following suitcase*. Available: http://cargocollective.com/ideactionary/hop

[5] I. e. a. Khan, "Automated Luggage Carrying System," *American Journal of Engineering Research (AJER),* vol. 2, pp. 61-70, 2013.

[6] Y. Kaub, "3x4 Cartesian coordinate grid (black on white)," in *Paint*, ed, 2015.

[7] G. M. Karastoyanov D, "Wireless Controlled Luggage Carrier " *PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS,* vol. 64, 2011.

[8] Pololu. (2007, 12/01/2016). *Pololu IR Beacon*. Available: https://www.pololu.com/file/0J31/irb02a_guide2.pdf

[9] DimensionEngineering. (2007, 12/01/2016). *SyRen 10 / SyRen 25 motor driver user's guide*. Available: https://www.dimensionengineering.com/datasheets/SyRen10-25.pdf

[10] Cytron Technologies. (2013, 30/10/2015). *User's Manual*. Available: https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit?pli=1

[11] SHARP. (30/10/2015). *GP2Y0A21YK0F*. Available: http://www.sharpsma.com/webfm_send/1489

[12] Parallax Inc. (2013, 10/11/2015). *PING))) Ultrasonic Distance Sensor (#28015)*

[13] G. Coley. (2013, 11/11/2015). *BeagleBone Black System Reference Manual*. Available: http://www.adafruit.com/datasheets/BBB_SRM.pdf

[14] Adafruit. (2015, 11/11/2015). *Introducing the Raspberry Pi 2 - Model B*. Available: https://learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-2-model-b.pdf

[15] Freescale. (2015, 11/11/2015). *Freescale Sensor Fusion Library for Kinetis MCUs*.

[16] Arduino. (2015, 30/10/2015). *Getting Started*. Available: https://www.arduino.cc

[17] mbed. (12/01/2016). *FRDM-K64F*. Available: https://developer.mbed.org/platforms/FRDM-K64F/

[18] Altronics. (2015, 11/11/2015). *Funduino Mega 2560 R3 Compatible Development Board*. Available: http://www.altronics.com.au/p/z6241-funduino-mega-2560-r3-compatible-development-board/

[19] Pololu. (2001-2015, 11/11/2015). *34:1 Metal Gearmotor 25Dx52L mm LP 6V with 48 CPR Encoder*. Available: https://www.pololu.com/product/2284

[20] Pololu. (2001-2015, 11/11/2015). *30:1 Micro Metal Gearmotor*.
[21] Pololu. (2001-2015, 11/11/2015). *29:1 Metal Gearmotor 20Dx41L mm*.
[22] Pololu. (2001-2015, 4/11/2015). *Dagu Wild Thumper Wheel 120x60mm and 4mm Shaft Adapter*. Available: https://www.pololu.com/product/1558
[23] Sparkfun Electronics. (2015, 11/11/2015). *Wheel - 65mm (Rubber Tire, Pair)*. Available: https://www.sparkfun.com/products/13259
[24] Pololu. (2001-2015, 11/11/2015). *Pololu Wheel 60×8mm Pair - Black*. Available: https://www.pololu.com/product/1420
[25] SeedWiki. (2015, 4/11/2015). *4A Motor Shield*. Available: http://www.seeedstudio.com/wiki/4A_Motor_Shield
[26] Sparkfun Electronics. (2009, 11/11/2015). *Serial Motor Driver User Guide* Available: https://www.sparkfun.com/datasheets/Robotics/SFE03-0012-UserGuide-ROB-09571-serialmotordriver.pdf
[27] SeedWiki. (2015, 11/11/2015). *Grove - I2C Motor Driver V1.3*. Available: http://www.seeedstudio.com/wiki/Grove_-_I2C_Motor_Driver_V1.3
[28] Sparkfun. (2001-2015, 11/11/2015). *ComMotion*. Available: https://cdn.sparkfun.com/datasheets/Robotics/ComMotion-Manual.pdf
[29] Atmel. (2015, 8/11/2015). *AtmelStudio7*. Available: http://www.atmel.com/microsite/atmel-studio/
[30] Visual Micro Ltd. (2015, 9/11/2015). *Arduino programming with Atmel Studio*. Available: http://www.visualmicro.com/page/Arduino-for-Atmel-Studio.aspx
[31] J. e. al., "Comparative study of C, Objective C, C++ programming language," *International Journal Of Engineering And Computer Science,* vol. 2, pp. 202-206, 1/1/2013 2013.
[32] K. J. Aström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, 2 ed. Research Triangle Park, NC: Instrument Society of America, 1995.
[33] W. Commons, "Feedback loop with descriptions," ed, 2008.
[34] W. Altmann, *Practical Process Control for Engineers and Technicians*. Jordan Hill, GBR: Newnes, 2005.
[35] B. A. Ogunnaike, *Process Dynamics, Modeling and Control*. New York: Oxford Press, 1994.
[36] B. e. al., "Direction-optimizing breadth-first search," *Scientific Programming,* vol. 21, pp. 137-148, 01/2013 2013.
[37] T. P. Kadek, J., "Extended Breadth-First Search Algorithm," *IJCSI International Journal of Computer Science Issues,* vol. 10, 2/11/2013 2013.
[38] E. Demaine, "13. Breadth-First Search (BFS) " in *MIT 6.006 Introduction to Algorithms*, ed: MIT Open Course Hardware, 2013.
[39] R. Kala, A. Shukla, and R. Tiwari, "Robotic path planning in static environment using hierarchical multi-neuron heuristic search and probability based fitness," *Neurocomputing,* vol. 74, pp. 2314-2335, 2011.
[40] Wikimedia Commons, "Breadth-First Tree," B.-F. Tree, Ed., ed, 2007.
[41] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding,* vol. 68, pp. 146-157, 1997.

[42] Y. Kaub, "Project Components," ed, 2015.
[43] Nordic, "nRF24L01 - Single Chip 2.4GHz Transceiver," ed, 2007.
[44] TinySine. (2015, 11/11/2015). *Tinysine Bluetooth 4.0 BLE module*. Available: http://www.tinyosshop.com/datasheet/Tinysine%20Serial%20Bluetooth 4%20user%20manual.pdf
[45] Adafruit. (2015, 4/11/2015). *Nokia 5110/3310 Monochrome LCD*. Available: https://learn.adafruit.com/downloads/pdf/nokia-5110-3310-monochrome-lcd.pdf
[46] freescale. (2012, 4/11/2015). *5.0A Throttle Control H-Bridge* [Datasheet]. Available: http://www.seeedstudio.com/wiki/images/0/07/MC33932.pdf
[47] Pololu. (2001-2015, 4/11/2015). *34:1 Metal Gearmotor 25Dx52L mm HP 6V with 48 CPR Encoder*. Available: https://www.pololu.com/product/2273
[48] Atmel. (2014). *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V*. Available: http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
[49] Atmel. (2009, 14/01/2016). *ATmega168P/ATmega328P*. Available: http://www.mouser.com/pdfdocs/Gravitech_ATMEGA328_datasheet.pdf
[50] TMRh20. (2015, 12/11/2015). *Optimized High Speed NRF24L01+ Driver Class Documenation*. Available: http://tmrh20.github.io/RF24/