Text, logo

Description automatically generated

Internet of Things Project

NaviCar

Nathan Ferry

Bachelor of Software & Electronic Engineering

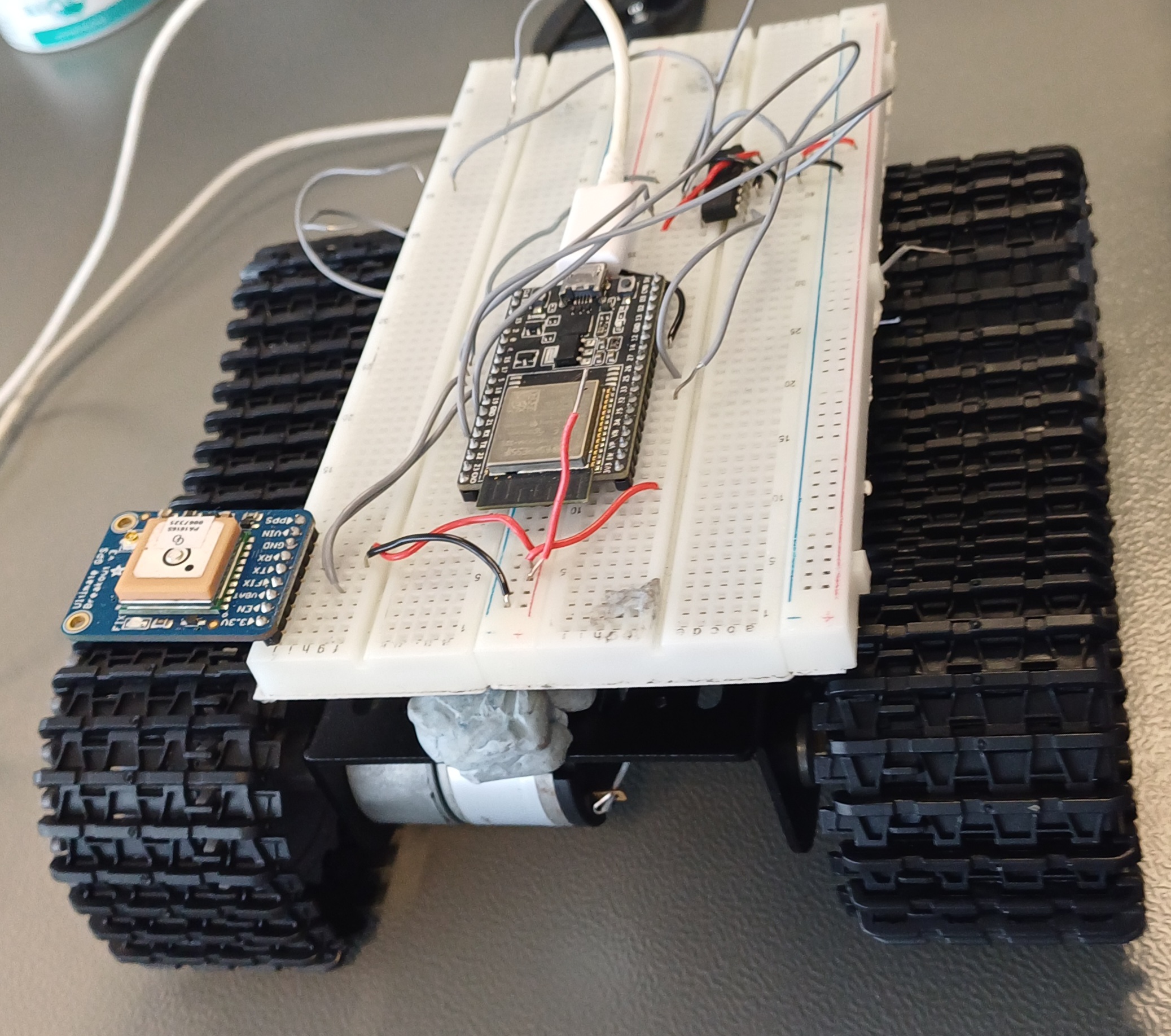
Atlantic Technical University

2023/2024

A screenshot of a computer program

Description automatically generated

IoT Project Poster



NaviCar Hardware

**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at the Atlantic Technical University, Galway campus.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

\_\_Nathan Ferry\_\_\_\_

**Acknowledgements**

Use this section to acknowledge anyone, if you wish to, who might have helped during your project.

**Table of Contents**

[1 Summary 8](#_Toc163897774)

[2 Project Architecture 9](#_Toc163897775)

[3 Development Platform and Tools 10](#_Toc163897776)

[4 Sensors 13](#_Toc163897777)

[4.1 DHT11 13](#_Toc163897778)

[4.1.1 DHT11 Code 14](#_Toc163897779)

[4.2 SEN0017 16](#_Toc163897780)

[4.2.1 SEN0017 Code 17](#_Toc163897781)

[4.3 HC-SR04 Ultrasonic sensor 17](#_Toc163897782)

[4.3.1 HC-SR04 code 18](#_Toc163897783)

[5 GPS Ultimate V3. 20](#_Toc163897784)

[5.1 GPS Code 20](#_Toc163897785)

[Nitty gritty, code pic 20](#_Toc163897786)

[6 Interrupts 21](#_Toc163897787)

[7 Motors 22](#_Toc163897788)

[7.1 DC Motors 22](#_Toc163897789)

[7.1.1 DC Motor Code 22](#_Toc163897790)

[7.2 Servo Motors 23](#_Toc163897791)

[7.2.1 Servo Code 23](#_Toc163897792)

[8 3D printed components 24](#_Toc163897793)

[8.1 Designing the components 25](#_Toc163897794)

[9 Web Server 26](#_Toc163897795)

[9.1 Wi-Fi 27](#_Toc163897796)

[9.2 Web Server 28](#_Toc163897797)

[9.3 HTML & CSS Code 28](#_Toc163897798)

[9.4 Web accessibility 28](#_Toc163897799)

[10 Problem Solving 29](#_Toc163897800)

[11 Impact of Project on Sustainability 29](#_Toc163897801)

[12 Conclusion 31](#_Toc163897802)

[13 References 31](#_Toc163897803)

[Appendix 1: Code 32](#_Toc163897804)

[Appendix 2: Bill of Materials 33](#_Toc163897805)

[Appendix 3: Schematic 35](#_Toc163897806)

# Summary

Sources-ides mentioned,hardware mentioned,sdgs,

The NaviCar was intended to be a self-navigating vehicle that would also update the user through a website on sources of information believed to be useful such as the location and temperature.

The NaviCar has a wide range of features such as a DHT11 temperature and humidity sensor, live location with the Ultimate GPS Breakout v3, line following capabilities utilising two SEN0017 line sensors and collision avoidance thanks to a HC-SR04 ultrasonic sensor that is capable of being rotated via a servo motor. The NaviCar is driven forward by two DC motors on the NaviCar chassis, connected to an L293D driver chip and along with all other hardware (except the DC motors) the chip is connected to an ESP32 microcontroller.

There are a range of functions and software that allows the project to host a webserver and using Thingspeak, update information on display graphs and a map as well as let the NaviCar to make decisions as to whether to turn to stay on the line it is following or to stop to avoid a collision. The collision avoidance is powered by the HC-SR04 taking regular readings and the main focal point of the self-navigation is the interrupts that run when either line sensor no longer detects a line. In this circumstance the NaviCar will turn to find the line again.

The project aims to tackle the U.N. Sustainable development goals (SDGs) 3 and 11 which focus on good health and wellbeing as well as sustainable cities and communities. Indicator 3.6 which is to road deaths and accidents by 2030 is targeted by the NaviCar as its self-navigating technology aims to provide safer transport than a human driver would as humans are prone to mistakes. Indicator 11.2 is to provide access to safe, affordable accessible transport as well as improve road safety for all. The NaviCar’s ability to follow strict routes will improve road safety in any given city.

The NaviCar was made by employing the philosophy of agile development. It is modular and while at its core it was designed to follow specific routes, it has turned into much more. Once several aspects of the project were complete, remaining time was used to not only polish the project but add extra features. All components were programmed using the Arduino IDE but also utilising past knowledge of 3d drawing. Two components that project hardware now mount to were designed and printed. All web design was written on the IDE Visual Studio Code to test my website, ThingSpeak to graph temperature and humidity the DHT11 read live. In total a website that utilises ThingSpeak to graph temperature and humidity over time as well as displays current GPS co-ordinates was developed and functional DC motors, line sensors, servo motor and ultrasonic that can notify if there are obstacles ahead were programmed. In conclusion, I believe the project is successful in what it aimed to do with further added features.

# Project Architecture

Sources-ide’s and esp32

The project architecture of this project includes the ESP32-DevKitC, Visual studio code IDE, Arduino IDE, Thingspeak and Onshape. Everything runs through the ESP32, and all hardware is connected to it. Visual studio code was used for all HTML/CSS programming. The Thingspeak API was used for graphing and displaying data to the webpage. Onshape was used for designing models to 3d print and the Arduino IDE was used for all other programming.

A diagram of a car

Description automatically generated

Figure 2‑1 Architecture Block Diagram

# Development Platform and Tools

Sources,Pictures,ides,esp32 website

The ESP32 has a host of features such as built-in Wi-Fi, Bluetooth, 45 programmable GPIO pins,3 Universal asynchronous receiver/transmitters (UART) and LED pulse width modulation (PWM) controller with up to 8 channels among so much more. All hardware was connected to the EPS32, and code uploaded to it to bring the project to life. Several IDEs were used to programme components and a website which was uploaded to the ESP32 which in turn operated as the brain of the NaviCar, telling each component what it was to do and when while also uploading data to ThingSpeak and hosting the webserver.

Figure 3‑1 ESP32

The Arduino IDE contains a text area for writing code and a console for outputs to be printed among other built-in features. It has a verify button to check code before uploading it to a microcontroller. All NaviCar components were programmed using Arduino functions and a main code loop. Code was separated into functions and multi file programming was used to store sensitive information. Passwords and network names were stored outside of the main project file.

A screenshot of a computer

Description automatically generated

Figure 3‑2 Arduino IDE

Thingspeak allows the user to send sensor data privately to the cloud to analyse and visualise it. Results against time can be graphed, a light programmed, that when a value is within a specific range its green but if it exits the range, it turns red alongside other features. Iframes of graphs can be embedded onto a website. Thingspeak was used to graph the temperature and humidity received live from the DHT11 mounted to the NaviCar chassis.

A screenshot of a computer

Description automatically generated

Figure 3‑3 ThingSpeak API

Visual studio code allows you to write HTML/CSS and then test it with a browser. It has built in debugging and allows various extensions for ease of code writing. The NaviCar website was created using Visual Studio Code.

A screenshot of a computer

Description automatically generated

Figure 3‑4 Visual Studio Code IDE sample

Sketches were written on Onshape that would later be 3D printed and used to mount the NaviCar’s ultrasonic, as well as the servo motor and both SEN0017s. The sketches were planned out on paper before being drawn on Onshape with precise measurements that were taken before putting pen to paper. Once the sketches were drawn on a plane extrude commands formed the final designs to be printed.

A screenshot of a computer

Description automatically generated

Figure 3‑5 Onshape sample sketch

# Sensors

There are several sensors involved in the NaviCar. There is a DHT11 that records the live temperature and humidity, two SEN0017s that monitor if the NaviCar is on the line or not and a HC-SR04 which measures the distance between the NaviCar and possible hazards.

## DHT11

(dht11 link 1) The DHT11 measures temperature using a thermistor and humidity via a capacitive humidity sensor. It operates in the voltage range of 3-5 volts and for humidity readings of 20% - 80% it has a 5% error. It has a +-2-degree Celsius error for a range of readings from 0-50 degrees Celsius. (Link 2 for thermistor) The DHT11 uses a negative temperature coefficient thermistor. This means that as temperature increases, so does resistance, this is how the DHT11 measures the live temperature. (Link 3 on humidity) The capacitance of the capacitive humidity sensor varies with humidity and this is how the DHT11 measures its surrounding humidity.

There are two main operatives of the DHT11. It passes the live value of the temperature and humidity to be recorded on the NaviCar website but also pass the temperature to the HC-SR04 for calculations. The DHT11 is connected to a 3.3-volt power supply, ground and pin 33 on the ESP32.

A white rectangular sign with black text

Description automatically generatedSources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to,

Figure 4‑1 DHT11 temperature sensor circuit

### DHT11 Code

The DHT11 code in the NaviCar project has three functions. getTemp() and getHumi take readings of the temperature and humidity when called and the values read are put to the webpage via a String named message which will be explained further on. DHT.read() tells the DHT11 to read a value and DHT11\_PIN refers to the signal pin the DHT11 has connected to the ESP32. A string variable then stores either the temperature or humidity based on what function is being used(getTemp would store temperature in the temp string). The value is then returned which will be a part of the String message explained earlier. The last function is similar to the previous two apart from the type of value returned. In tempy() the DHT11 stores the temperature in the integer temp and returns the variable. This value will be passed to the ultrasonic sensor function Usensor(). The reason for calling the tempy() function in Usensor will be explained in the HC-SR04 code section.

A screen shot of a computer code

Description automatically generated

Figure 4‑1-1 DHT11 sensor code

## SEN0017

Sources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to

The SEN0017 sensor can detect light and can provide a stable transistor transistor logic output signal. (i.e., 1 for light and 0 for dark) This data is used to tell if the sensor is over the line or not. It operates in the voltage range of 3.3-5 volts. It detects from a range of 1-2cm. The SEN0017s are connected to 3.3 volts, ground, and pins 22 and 23 on the EPS32.

If a sensor doesn’t detect the line, the motors are told to turn left or right depending on which sensor reads that it is not on the line. For example, if the left sensor doesn’t detect the line code will cause the function to turn right to get back on the line. Interrupts are used to achieve this. Interrupt functionality will be described in further detail later in this report.

A white rectangular sign with black text

Description automatically generatedA white rectangular sign with black text

Description automatically generated

Figure 4‑2 SEN0017 sensor circuit

### SEN0017 Code

Nitty gritty,

A screen shot of a computer program

Description automatically generatedThe SEN0017 code works through interrupts and tests through Sense(). r1 and r2 are integers representing each sensor. digitalRead() is used to read specific named pins(in this case pins 22 and 23). Strings are printed followed by the values r1 and r2 store. This function is void as it returns no values and is used to test the operation of the interrupts. The SEN0017 interrupt code will be explained in full in the Interrupts section.

Figure 4‑2-1 SEN0017 sensor code

## HC-SR04 Ultrasonic sensor

Sources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to,

The ultrasonic sensor sends out pulses via a trigger pin which in turn bounce off an object and are received by the echo pin. The HC-SR04 is given a low pulse before a 10us high pulse. This low pulse ensures a clean high pulse. The high pulse triggers the sensor which sends out 8 pulses at 40kHz. Through a formula explained further on, the distance is calculated between the sensor and an object. The ultrasonic trigger pin is connected to pin 5 and echo pin 18 on the ESP32. The sensor operates at 5 volts; therefore, a potential divider circuit is used to step down the voltage to be compatible to the ESP32’s 3.3-volt supply. The sensor is also connected to ground. The sensor reads a range of 2cm to 4m with an accuracy of 3mm.

A diagram of a circuit board

Description automatically generatedThe ultrasonic sensor is used to measure the distance of the NaviCar from obstacles. There are an echo and trigger pin that receive and send out small pulses which reflect off objects and come back to the sensor. The sensor takes the time taken for the pulse to return and the speed of sound to get the distance travelled. This distance is halved to get the distance of an object from the sensor. The DHT11 passes the temperature to the HC-SR04 to calculate the speed of sound with the actual temperature of the air rather than an estimate. This is used to get a more accurate distance. The ultrasonic stops the NaviCar if the measured distance is too small to avoid a collision. The Servo will then rotate, and the ultrasonic will take several readings to determine the best path forward. For example, ahead it might read 5 centimetres, on the left 12 centimetres and the right 25 centimetres, in this scenario the NaviCar will turn right.

Figure 4‑3 Ultrasonic sensor circuit

### HC-SR04 code

nitty

The HC-SR04 is passed the live temperature from the DHT11 and uses the formula: v=331+(0.6T) to measure the speed of sound in air. We can estimate the speed, but I would prefer to pass the live temperature(T) and use the formula to attempt to get a more accurate distance. The formula: (duration/2) \* (speed of sound/10000) is used to calculate the distance from an object. The function is void as it prints to the serial monitor the distance read. The ultrasonic sets the trigger pin low via the digitalWrite() method. It is kept low for 5 microseconds to ensure a clean high pulse. The trigger pin is then set high for 10 microseconds which triggers the sensor to send a burst of 8 pulses at 40kHz from the sensor after the trigger pin is set low once more. The burst is received at the echo pin after hitting an object and the time taken for the pulse to hit off an object and come back to the echo pin is used to calculate distance. The variable duration is set to the duration of the high pulse received at the echo pin. tempVal stores the temperature that is read from the DHT11 after calling the tempy() function. The temperature is needed to calculate the speed of sound in air. We could use an estimate but getting the actual temperature is more accurate. airVal is this speed of sound in air and uses the formula 331.5+(0.6\*tempVal). The distance is stored in cm and calculated by dividing the duration for the pulse to return over 2 (as duration is the time to bounce off on object and come back and dividing it by 2 gives us the time for sound to travel from the NaviCar to an object). This is then multiplied by airVal which is divided by 10,000. The value of cm is then printed to the serial monitor.

A screenshot of a computer program

Description automatically generated

Figure 4‑3-1 Ultrasonic sensor code

# GPS Ultimate V3.

Sources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to,

The GPS Ultimate V3 is a 3.3-5-volt device that can track up to 22 satellites on 66 channels. It can tell you your altitude, latitude, longitude, date, and time with many other features. It can do up to 10 updates a second for high performance tracking, with a built-in antenna. It has a LED that blinks at 1Hz while searching for satellites and once locked on it conserves power by blinking every 15 seconds. The GPS has a receiver pin connected to a transmit pin on the ESP32 (pin 17). The GPS has a transmit pin connected to a receiver pin on the ESP32 (pin 16). The GPS is also connected to 3.3 volts and ground.

A diagram of a gps device

Description automatically generatedThe GPS passes the live value to the NaviCar website once it locates a connection to satellites. The website also allows the user to have a direct link to google maps where the co-ordinates will show your location.

Figure 5‑1 GPS circuit

## GPS Code

## Nitty gritty, code pic

# Interrupts

Interrupts are used to keep the NaviCar on the line it is following. attachInterrupt() is used to attach the interrupt to a pin, state the interrupt service routine (i.e. turn left/right to go to the line) and the condition for the interrupt to enact (i.e. falling, rising etc). For the SEN0017 the pins 22 and 23 are attached, based on which side of the NaviCar they are on the interrupt service routine is either the Left or Right function which tells the motors to turn in those directions. Both interrupts are triggered by a falling edge. The SEN0017s work on a falling edge where if they were to go from reading a 1 to 0 then an interrupt occurs. If one of the sensors read they are off the line, they call the interrupts to run a function to turn back onto the line and continue to follow it.

A screen shot of a computer program

Description automatically generated

Figure 6‑1 Interrupt code

# Motors

Include a subsection on your motor’s power requirements.

Sources,broad overview of all components say will go further into detail later

I have several types of motor on my project. I have two DC motors and a servo motor. The DC motors drive the NaviCar, and I have several functions written to control them. The Servo motor rotates the ultrasonic if need be and has multiple of its own functions.

## DC Motors

Sources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to,

The DC motors drive the chassis forward, backward, left, and right. The ESP32 tells the motors what to do through specific function code i.e. go left will turn on the right motor and make it go forward while keeping the left motor off.

Figure 7‑1 DC motors circuit

### DC Motor Code

The DC motors each have three pins to control each motor. The enable pin turns them on, this must be high for the motor to do anything, and there are two other pins which tell the motor to go forward or back (i.e., one high and the other low mean go forward). All the motor functions are void as they return no values. The M1ON and M2ON constant integers are the enable pins, MotorPin1 to MotorPin4 are the non-enable pins. The pins are set to pin numbers before coded in functions. The Reverse function is one example of several that exist. There is also a Left, Right, Stop and Forward function. All functions print using Serial.println(“”); the mode of the motors i.e. in Reverse it is “Reversing” but in Forward it is “Going Forward” etc. The enable pin of the first motor is then set followed by the other two pins and the other motor pins. The motor function Forward is the NaviCar’s default mode unless an interrupt is called.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

#### DC Motor power requirements

Explain L293D here

## Servo Motors

Sources,Picture of component+circuit diagram +pins+what it does overview of projects,what its connected to,

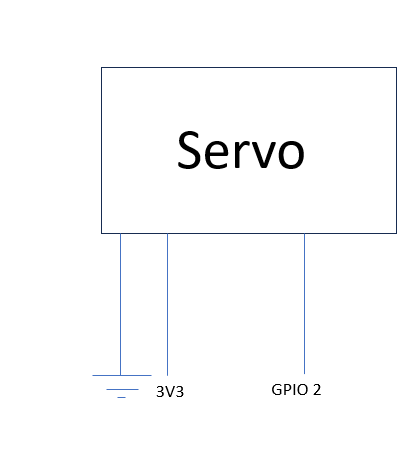
The servo motor operates by being fed a pulse of variable width. If the pulse if 20ms and it is high for 1ms and low for 19ms the servo could be held at 0 degrees, for 2ms high it could be 180. The duration of the high pulse tells us how far the servo is rotated and held at that specific degree. The NaviCar has functions to turn the servo left and right. The servo motor is a volt component and connected to pin 2, ground and 3.3 volts. It operates from 3.3-5 volts.

Figure 7‑2 Servo motor circuit

### Servo Code

Nitty gritty

The servo motor control code works by using for loops. Servo1.write tells the servo a number between 0-180 degrees to put the servo to, therefore a simple for loop turns the servo left or right.

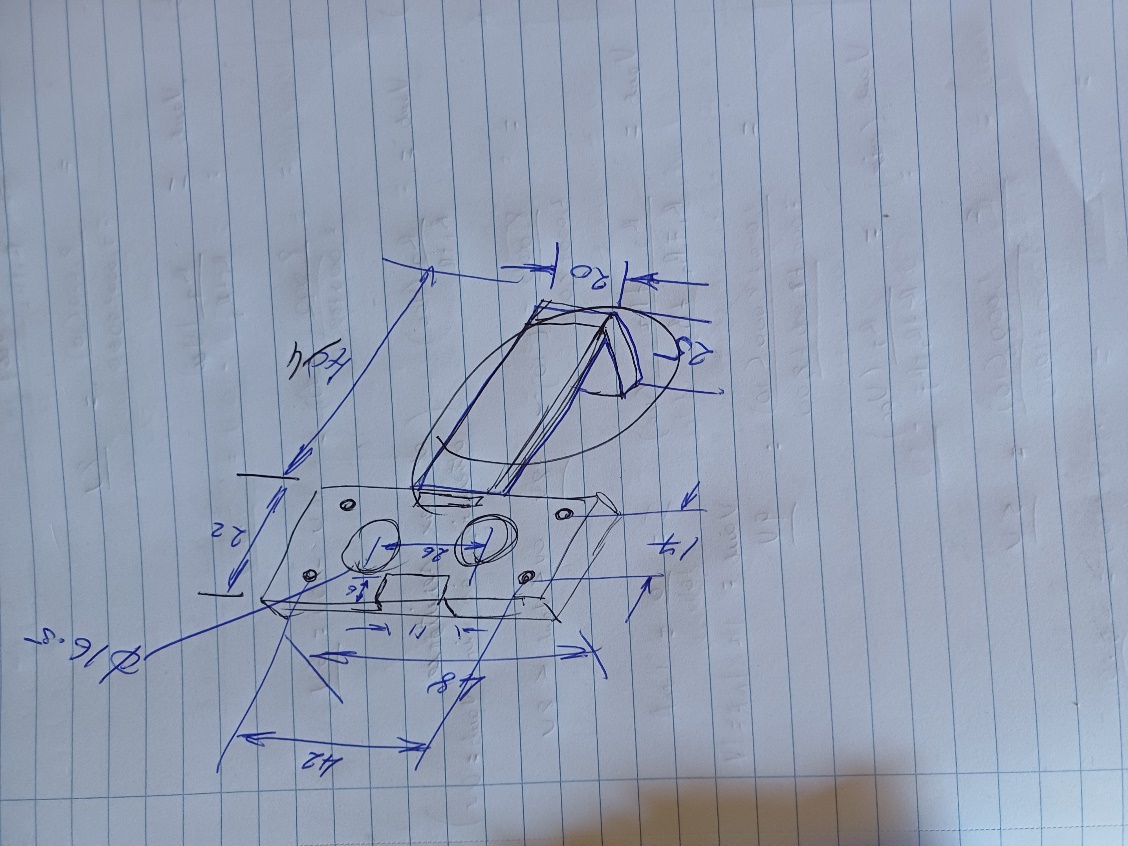
A computer screen shot of a program code

Description automatically generated

# 3D printed components

Sources for Onshape description , detail of problem solved

3D components were sketched on paper and then drawn in software once problems arouse in how the ultrasonic sensor would be mounted on the top of the servo motor. The servo had to be able rotate the ultrasonic via a part that was functional and sturdy. A second part was drawn as the SEN0017s had to be 2 centimetres from the ground to get a reading and the servo had to be held steady on the breadboard. Below are the rough starting sketches of the ideas and the finished software sketches.

A piece of paper with a drawing on it

Description automatically generated

Figure 8‑1 Component concept sketches

A blue rectangular object with a hole

Description automatically generatedA blue object with circles

Description automatically generated

Figure 8‑2 Component 3D models

Paste models

**Figure 8‑3** Component Printed models

## Designing the components

In depth on functionality of Onshape and how you did this, sent info etc

I first designed a holder for the ultrasonic to be mounted to the servo motor. This was easier to design and draw. It consisted of several extruded cuts for the ultrasonic to fit into and a mounted frame to attach to the top of the servo. I then created a design for my SEN0017s and servo motor to fit to. These are easily placed on my chassis and worked well the first time I printed them. The second part has fits neatly on top of the breadboard and gives the project a better sense of layout for the servo motor and SEN0017s. It consisted of an extruded frame similar to the first part and an extruded cut for the servo to fit into.

# ThingSpeak

Explain dht11 is read and how etc explain thingspeak uplioad

A screen shot of a computer code

Description automatically generated

# Web Server

Sources, talk to other about how they approached this section ,

My ESP32 hosts a webserver that my webpage is sent to as a message containing functions and raw string literals. I have an ssid and password in a separate header file.

A screen shot of a computer program

Description automatically generated

## Wi-Fi

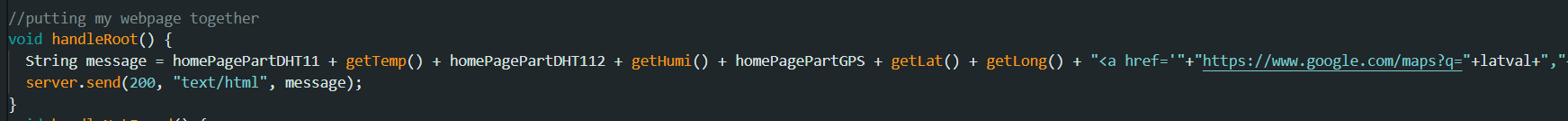
The ESP32 has built-in Wi-Fi capabilities that I use to connect and host a webserver on using a mobile hotspot.

## Web Server

A Webserver is hosted on the ESP32 using an ssid and password (the name and password of my hotspot). The ssid and password are in a separate header.

## HTML & CSS Code

I wrote the HTML/CSS code to split the website into rows of information. There are headings and data given for each row and it shrinks and grows with the webpage. A header file called homepage.h holds my html code as it is split into a message as seen below where segments of my webpage and function calls are added together.



## Web accessibility

Detail what you have included to ensure that your website is accessible.

You could break it down to the following headings:

* Perceivable - Web content is made available to the senses - sight, hearing, and/or touch
* Operable - Interface forms, controls, and navigation are operable
* Understandable - Information and the operation of user interface must be understandable.
* Robust - Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies

Perceivable:  
e.g. For Screen readers to know what language to use:  
<html lang="en" >   
All links and alt tags were given meaningful names e.g.   
<p><a href="http:some big long address" >Make link text meaningful</a></p>

<p><img src="/image path.png" alt="Include meaningful description/make empty if unnecessary"></p>   
etc….

# Problem Solving

Describe the major problems / challenges encountered during your project, and how you solved them. Highlight your approach to problem solving and the steps you took to successfully solve problem(s).

Overview of problem, photo of solution before vs after prints etc

There were several examples of my problem solving from within this project, some were easier to diagnose than others while some were easier to fix once the key to the problem was identified.

The NaviCar started out with only the goal of self-navigation, but as the scope of the project increased, so did the number of active components in use. A problem arouse as the breadboard and chassis was becoming increasingly busier with added components mounted with blu tack and wires tangling around parts such as over the ultrasonic sensor and between the servo, it was clear that a solution was needed to cut down on the use of blu tack and give strength to the project. There was also a problem of keeping the SEN0017s above the ground but not haphazardly dangling off the front of the breadboard. They needed to be mounted to something. The same could be said for the ultrasonic that had to be attached to the servo motor somehow. Until that point a makeshift but unsteady mount was attached to the ultrasonic via blu tack. This is where an idea to 3d print pieces for the chassis developed. Models were sketched on paper, then in software and finally printed. The designs were simple as it was clear what components were needed to be below the chassis and what was designated to be near or on the breadboard. Two parts were drawn. The first solved the problem of the line sensors. They had a part that they could now be attached to sturdily but also held them above the ground at the required 2 centimetres for reading distance. The part also had a holder for the servo motor so that the breadboard had a clearer layout for the sensors in use. The first part elegantly overhung on the front of the breadboard and maximised space by having a servo motor holder. Until then the servo was held to the board via blu tack. The second part was designed to hold the ultrasonic sensor to the servo motor. This was simpler to draw and removed more placeholder blu tack from the project but also gave strength to the ultrasonic as it sat above the servo, ready to turn with it.

The second problem the NaviCar encountered was before the Christmas demo to showcase the project. The code to follow a line was working and the interrupt code worked but due to an error, were the SEN0017s were connected to a power supply that gave an excess of current, they no longer functioned. The demo had to be reworked to focus on the ultrasonic stopping the NaviCar if it got to close to a wall instead of the preferred line follower idea as the ICs on the sensors were broken by overloading current. It was a good lesson as it showed how costly simple mistakes can be. Preventative double checking have been taken when testing any components since this error.

Another problem the NaviCar encountered was a faulty breadboard. Through each iteration of code, the DC motors would be tested to make sure they still functioned as intended. Through inspection, it turned out that the pin X on the L293D motor driver was not connected to the wire on the track of the breadboard used for connections. The track had broken and for hours the problem remained unsolved until a wire was directly placed on the motor driver pin. It was at this point that it was clear that the breadboard was the issue.

Another issue the NaviCar faced was to do with interrupts. There was an issue where the interrupts would cause a reboot if the serial monitor was used. The watchdog would terminate the programme as serial communication would take longer than the watchdog would allow. One the serial communication was commented out the interrupts would not cause the reboot.

# Impact of Project on Sustainability

Consider

* ~~Project application~~

~~Impact of project on United Nations Sustainable Development Goals UN SDGs. Pinpoint the exact SDG targets impacted negatively or positively.~~

* ~~Accessibility e.g. website accessibility. Include a screenshot.~~
* ~~Power Budget~~
* Health and Safety
* Programming Style
* ~~Component reuse and recycling~~
* Plagiarism/referencing
* Sources ,

The ESP32 was built with the sustainable development goals (SDGs) in mind. It was set to Xtarget 3.6 which is to half the number of road deaths and accidents by 2030. The NaviCar uses sensing technology to avoid collisions when moving forward while never deviating from a set path. The NaviCar uses 2 SEN0017s for path following. Target 11.2 is to provide access to safe, affordable, accessible, and sustainable transport systems as well as improve road safety for all. The NaviCar’s ability to follow a predetermined path and sense upcoming collisions leads to improved road safety. The NaviCar can avoid possible collisions by braking and senses with the ultrasonic sensor. The NaviCar has an accessible webpage that uploads an array of live data. The webpage was checked for accessibility using lighthouse and it was found that X. The NaviCar’s power budget looks like X. All of the components can be reused from the NaviCar as they consist of wires and components attached to a breadboard. At the end of the Internet of Things module, the NaviCar will be deconstructed to its bare parts. All sensors can be reused, the wiring may be harder to reused but some can be, the only problem would be the 3d printed components. The chassis and ESP32 would also be easily reused.

# Conclusion

Write a short conclusion. What is the outcome of the project. Perhaps you have a product prototype, or some analysis of data from sensors. Keep it focussed on what you have done. You can mention future opportunities for the work but keep this part short.

The NaviCar can tell you where you are, how hot and humid you are, tell you if you are about to hit something and lead you along a specified line. This one project does all of that as well as host a webpage with all the information measured and allows us to see insights into the self-autonomous future. The NaviCar is the culmination of a lot of effort, but all worth it in the end thanks to the final product.

# References

My references:

Research on 12/04/24:

Esp32 sheet: <https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf>

Arduino IDE: <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/>

Thingspeak: <https://thingspeak.com/pages/learn_more>

Vscode: <https://code.visualstudio.com/docs>

Onshape: <https://www.onshape.com/en/>

13.04.24

DHT11: <https://www.adafruit.com/product/386>

what thermistor dht11 uses: <https://sites.duke.edu/memscapstone/temperature-and-humidity-sensors/#:~:text=DHT11%20is%20a%20low%2Dcost,is%20negatively%20correlated%20with%20temperature>.

capacitive humidity sensor: <https://www.electricity-magnetism.org/how-do-capacitive-humidity-sensors-work/#:~:text=Capacitive%20humidity%20sensors%20operate%20based,capacitance%20increases%2C%20and%20vice%20versa>.

Sen0017: <https://wiki.dfrobot.com/Line_Tracking_Sensor_for_Arduino_V4_SKU_SEN0017>

HC-SR04: <https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/>

GPS: <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf>

Servo: <https://www.pololu.com/product/2818>

|  |  |
| --- | --- |
| [1] | Espressif, "ESP32\_Datasheet," 2022. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32\_datasheet\_en.pdf. [Accessed 4 April 2022]. |
| [2] | Mathsworks, "IoT Analytics - Thingspeak Internet of Things," [Online]. Available: https://thingspeak.com/. [Accessed 28 March 2023]. |
| [3] | Texas Instruments, "www.ti.com," 1999. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm35.pdf. [Accessed 4 April 2022]. |
| [4] | Arduino, "www.arduino.cc," Arduino, 6 April 2022. [Online]. Available: https://www.arduino.cc/reference/en/libraries/servo/. [Accessed r April 2022]. |

Some example references are given above. For example, the section in your report on your servo motors should have a sentence that includes [3] in it, referring to reference [3] here.

Use the References facility in MSWord as described in the Lecture. Use the IEEE style.

Include all your Datasheets.

# Appendix 1: Code

Include all the code you have written here - your top-level Arduino .ino file, and any .cpp/.h files you created. Do not include third party code.

It is worthwhile including your website as HTML/CSS code which can be easily viewed from any browser.

# Appendix 2: Bill of Materials

Include a Bill of Materials table. State whether you got your part from ATU stores by placing a ‘y’ or ‘n’ under the ATU Stores column.  
If you can include the price of all components to show the approximate cost of your project. It’s not mandatory to have all costs.  
Mandatory sections to complete are Item, Quantity and ATU Stores.  
Don’t forget to multiply the unit cost by the number of components you use in your project e.g. the HC-05 module costs 19.45 each so the total cost for 2 is 38.90.  
You don’t need to cost resistors, breadboards, stripboard and wiring.  
It’s useful to create the table in Excel and then copy and paste here.   
Some students sourced parts themselves and you can indicate this by using the ‘\*’.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Quantity** | **Manuf** | **Manuf No** | **ATU Stores** | **Sourced from** | **Order No** | **Cost Euros** |
| Arduino Uno | 1 | Arduino | A000066 | y | Radionics | 715-4081 | 18.86 |
| 3K3 | 1 |  |  | y |  |  | 0 |
| 1K8 | 1 |  |  | y |  |  | 0 |
| ESP8266 | 1 | Sparkfun | WRL-17146 | y | Mouser | 474-WRL-17146 | 5.89 |
| HC05 Module | 2 | Velleman | 158 | y | Robot Shop | RB-Vel-158 | 19.45 |
| Fingerprint Sensor | 1 | Adafruit | 4690 | n | Mouser | 485-4690. | 16.9 |
| LCD | 1 | Displaytech | 162KBCBW | y | Radionics | 210-9031 | 11.94 |
|  |  |  |  |  |  |  |  |

Components marked with an asterisk indicate that the student sourced this component themself.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Quantity** | **Manuf** | **Manuf No** | **ATU Stores** | **Sourced from** | **Order No** | **Cost Euros** |
| HC-SR04 | **1** | **Kitronik** | **46130** | **y** | **Radionics** |  | €3.03 |
| GPS Ultimate V3 | **1** | **Adafruit industries** | **746** | **y** | **Radionics** |  | **€44.72** |
| SEN0017 | **4** | **DF Robot** | **SEN0017** | **n** | **Farnell** | **2946112** | **€19.48** |
| DC Motor | **2** | **RS Pro** | **n/a** | **n** | **Radionics** | **238-9737** | **€15.72** |
| Servo | **1** |  |  | **y** | **Radionics** |  | **€5.61** |
| DHT11 | **1** | **Seeed Studio** | **101020011** | **y** | **Radionics** |  | **€8.61** |
| ESP32 | **1** |  |  | **y** |  |  |  |
| Chassis | **1** | tbc | n/a | y | n/a | n/a | **tbc** |
| L293D | **1** | STMicroelectrics | L293D | y | STMicroelectrics | 714-0622 | **€5.12** |

Components marked with an asterisk indicate that the student sourced this component themself.

# Appendix 3: Schematic

A screenshot of a computer

Description automatically generated