Lab Report 1: Communications and Sensors

*Mohamed Ghali*

*Student ID: 20381397*

# Abstract

Whilst microcontrollers have significant potential independently, their potential is further enhanced with the capability of communication. In this report, there will be an analysis of how microcontrollers (more specifically the Arduino Nano [1] and ESP32 [2]) can communicate with sensors and other microcontrollers to achieve set tasks, that require them to adapt to their surroundings.

For example, Motion data can be retrieved from sensors to analyse the motion of an autonomous vehicle and this data can be used to autonomously park a vehicle. Some aspects of this microcontroller application will be described in this document.

# Contents

[Abstract 2](#_Toc92714100)

[Contents 3](#_Toc92714101)

[1 Introduction and Background to Autonomous Vehicles 4](#_Toc92714102)

[1.1 Rotary Encoders to Measure distance travelled 6](#_Toc92714103)

[1.1.1 Coding with the Rotary Encoder 6](#_Toc92714104)

[1.2 Ultrasonic Distance Sensor (HC-SR04) to measure range 7](#_Toc92714105)

[1.2.1 Coding with the HC-SR04 7](#_Toc92714106)

[1.3 IMU to measure motion 8](#_Toc92714107)

[1.3.1 I2C, SDA and SCL 8](#_Toc92714108)

[1.3.2 Coding with the MPU9250 9](#_Toc92714109)

[2 Modelling Motion Using MATLAB 10](#_Toc92714110)

[2.1 Velocity under a Constant Acceleration 10](#_Toc92714111)

[2.2 Displacement under constant acceleration (I) 11](#_Toc92714112)

[2.3 Displacement under constant acceleration (II) 11](#_Toc92714113)

[2.4 Displacement under constant acceleration (III) 12](#_Toc92714114)

[2.5 Angular Displacement under Constant Acceleration 13](#_Toc92714115)

[3 Lab Work 15](#_Toc92714116)

[3.1 Rotary Encoder 15](#_Toc92714117)

[3.2 HC-SR04 15](#_Toc92714118)

[3.3 IMU 16](#_Toc92714119)

[3.4 Reverse Parking Manoeuvre Challenge 16](#_Toc92714120)

[3.4.1 Programming the microcontroller 17](#_Toc92714121)

[3.4.2 Trials and results 20](#_Toc92714122)

[4 Conclusion 21](#_Toc92714123)

[5 Table of Figures 22](#_Toc92714124)

[6 Appendices 23](#_Toc92714125)

[6.1 Appendix 1: Code for Rotary Encoder Activity 23](#_Toc92714126)

[6.2 Appendix 2: Code for ultrasonic sensor activity 24](#_Toc92714127)

[6.3 Appendix 3: Code for IMU Activity 25](#_Toc92714128)

[7 References 26](#_Toc92714129)

# Introduction and Background to Autonomous Vehicles

The human reaction speed can sometimes be far too slow to prevent danger. Autonomous vehicles, however, can respond to their surroundings at a much faster speed. Autonomous vehicles can rapidly collect data from their surroundings using sensors, then rapidly process them using microcontrollers, to result in an almost automatic response.

Autonomous vehicles could soon boost human productivity, as they would mean that humans no longer need to be present at the wheel when commuting in a private vehicle.

However, because autonomous vehicles are still starting to roll out, and they use quite expensive and accurate sensors, right now they are generally unaffordable to the working-class public. These cars are also relatively difficult to maintain, due to the sensitivity of the sensors.

Even though this topic area is still under research, significant research has been made by academics and private institutions to develop this topic area at an extraordinary rate.

There are several types of autonomy, as shown in Figure ‎1.A, below. The EEEBot being built during this lab falls into level 4 of automation. However, level 5 automation, on public roads, is well under its way.

Although the prototypes seem to be very successful, a fully autonomous car that is reliable enough to be on the streets has not been constructed yet. This is mostly because of the difficulties involved in controlling a vehicle in the unpredictable traffic conditions of urban areas [3]

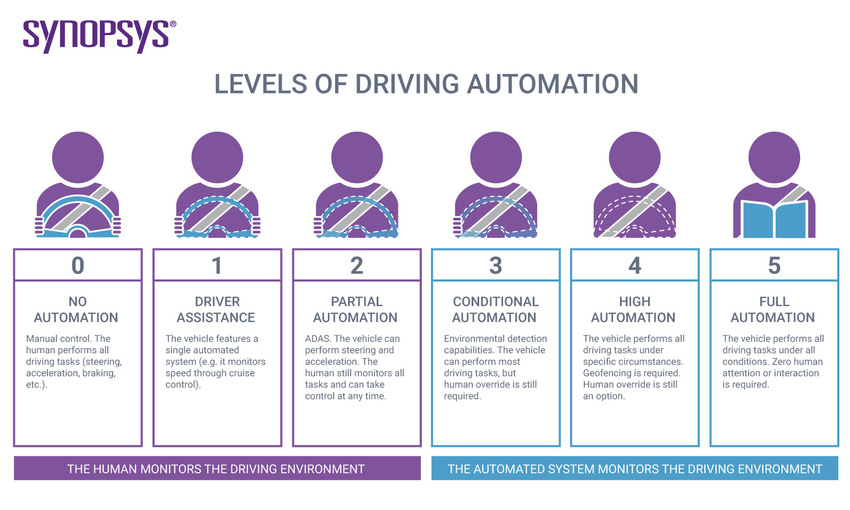


Figure .: The levels of autonomy [4]

Also, the economy would benefit dearly from the presence of fully autonomous vehicles, as they would provide a means of transportation without the need for a driver. This means that driver shortages would no longer have the huge effect it does today.

Companies can also provide vehicle services to customers, with more profit, as they’ll no longer need to pay for a driver.

Giving the wheel to the computer, also means that drivers with a rebellious mind are now more controlled, as they are no longer able to force the car above any restricted limits e.g. speed limits.

Learning how to drive could also be dearly advanced if needed at all, because the car would be more aware of its surroundings and potential dangers than a driving instructor could be.

People drive their cars to work, to go shopping, to visit friends and to many other places. Children take the bus to school. The economy depends largely on the goods that are delivered by trucks. [3]

Even though autonomous vehicles have had a short history dating back to 1977, they have been very successfully advanced within the last decade or so by the well-known Tesla, Inc [5].

The first known worthy attempt to build an autonomous vehicle was in 1977. [3]

## Rotary Encoders to Measure distance travelled

Rotary encoders can be used to measure the distance travelled by an autonomous vehicle. They are attached to the wheel, and when the wheel turns, a count is interpreted by the microcontroller, telling it how far the wheel has moved.

An application of a rotary encoder could be to make a vehicle stop after it has travelled a set distance.

**The number of counts needs to be scaled, to give the distance in the unit wanted. To do this, take a trial run of set length (the longer the more accurate) and measure how many counts the encoder interpreted over that distance.**

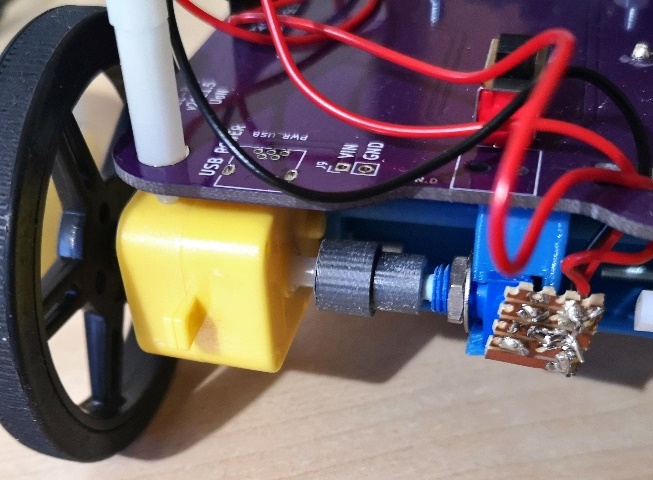
­­­

Figure .: A rotary Encoder attached to the EEEBot’s wheels

Using a Veroboard, for example, the central pin on the encoder is attached to the microcontroller’s common ground, and the other 2 pins can connect to any 2 IO pins, in this case, they have been attached to IO32 and IO33 of the **DOIT ESP32 DEVKIT V1** [2].

### Coding with the Rotary Encoder

You can take readings from the rotary encoder using libraries, appropriate to the microcontroller, which are available online. In this case, the library used was **ESP32Encoder** [6].

In this library:

**ESP32Encoder::useInternalWeakPullResistors = UP** is used to activate the pull resistors within the rotary encoder. Setting this equal to UP activates the pull-up resistors.

**encoder.attachHalfQuad(pinA, pinB)** is used to declare where the two wires are connected.

**encoder.count()** is used to extract the current count of the rotary encoder.

**encoder.clearCount()** is used to set the count of the encoder to 0.

## Ultrasonic Distance Sensor (HC-SR04) to measure range

The HC-SR04 [7] can be used to find the distance between the sensor and an obstacle, using ultrasound. The distance can then be processed and used by the microcontroller. When attached to the back of an autonomous vehicle, it could be used to find the distance between the vehicle and a nearby object.

One application for the HC-SR04 [7] could be to make an autonomous vehicle stop when it approaches an obstacle.

**However, be aware that there is a maximum range that the HC-SR04** [7] **can measure and it wouldn’t work as expected for any distance further than that. For the HC-SR04 the maximum range is 4m** [8]**, but the accuracy depletes with distance, so it will be used at a maximum 2m range. Also, consider that the accuracy is quite low at very short ranges.**

A picture containing text, electronics

Description automatically generated

Figure .: A HC-SR04 [8] attached to the back of an EEEBot

There are 4 pins on the HC-SR04 [8], which are connected to the microcontroller. A ground pin (GND) is connected to the common ground of the microcontroller. A Vcc pin is connected to the output 5V of the microcontroller. An echo pin is connected to any microcontroller pin, which is defined as the echo pin in the code. A trigger (Trig) pin is connected to any microcontroller pin, which is defined as the trigger pin in the code.

### Coding with the HC-SR04

You can take readings from the HC-SR04 [8] to an Arduino Nano [1] using the **NewPing** [9] **library**.

In this library:

**NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE)** is used to initiate a connection with the sensor, at the maximum distance desired.

**sonar.ping\_cm()** is used to retrieve the distance, in cm of the intercepting obstacle.

## IMU to measure motion

An inertial measurement unit (in this case, MPU9250 [10]) is used to analyse the motion of a system, in a 3-axis cartesian method. It has a built-in accelerometer that can measure the acceleration of the object and it has a gyroscope that can measure the angular velocity of the object. It also has a magnetometer to measure the strength of a magnetic field, however, this is not used in the lab.

An application of this could be to make an autonomous vehicle rotate at a specific angle, such as 90 degrees right for a right turn.

A close-up of a circuit board

Description automatically generated with medium confidence

Figure .: An MPU9250 [10] attached to the EEEBot

There are only 4 pins, which are needed, to use the IMU for the task below. That is Vcc, GND (ground), SCL and SDA.

The ground is attached to the common ground of the microcontroller, and Vcc is attached to the 5V output from the microcontroller. SCL and SDA are attached to the relevant pins, according to the microcontrollers used. Please refer to the pinout of the microcontroller. In this case, the microcontroller is an Arduino Nano [1] and the SDA pin is A4 and the SCL pin is A5.

### I2C, SDA and SCL

An I2C bus is used for communication between master and slave microcontrollers, and other peripheral devices in a system. The SDA is a 2-way data line that sends the data required between the devices. The SCL is a clock line, which prepares all the devices for synchronous transmission. The SCL signal is only sent by the master devices.

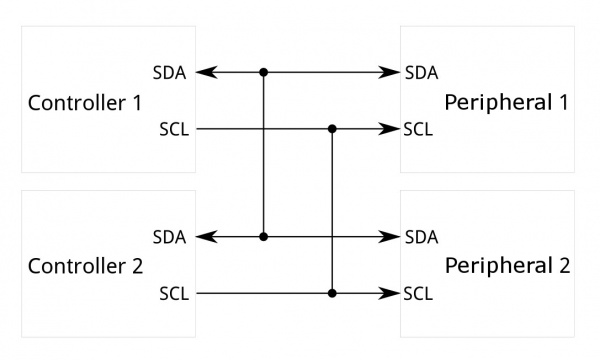


Figure .: A diagram demonstrating a system using the I2C bus [11]

### Coding with the MPU9250

With an Arduino Nano [1], the **MPU6050\_tockn library** [12] (works with MPU9250 [10]) can be used to analyse data from the IMU.

Figure .: The coordinate axes given on the MPU9250 [10]

With the **Wire** [13] library active, the following functions can be used:

**MPU6050 mpu6050(Wire)**

**mpu6050.calcGyroOffsets(true)** to determine the angle offsets from where the system started.

**mpu6050.update()** to get updated information from the MPU.

**mpu6050.getAngleX()**, **mpu6050.getAngleY**() and **mpu6050.getAngleZ**() to get the angle rotated from the initial position around the respected axes. These give the angles rotated around the axes given on the MPU below, where Z is out of the page.

# Modelling Motion Using MATLAB

MATLAB [14] is a powerful tool, which can manipulate data in its masses, using arrays. It is then able to plot graphs using these arrays. This part of the lab report will be using this software.

In the case where acceleration is constant:

Equation : Acceleration under constant acceleration

* Where a is a variable representing the acceleration, across all times

If this is integrated, in terms of time, to get an expression for velocity, under constant acceleration:

Equation : Velocity under constant acceleration

* Where v is a variable representing the velocity at time t
* Where u is a constant representing the velocity at t equals zero
* Where t is a variable representing time

If this is further integrated, in terms of time, to get an expression for displacement under constant acceleration:

Equation : Displacement under constant acceleration

* Where s is a variable representing the displacement at time t
* Where s0 is a constant representing the displacement at t equals zero

## Velocity under a Constant Acceleration

Chart, line chart

Description automatically generated

In this activity, the task was to plot a graph, onto MATLAB [14], which represents velocity under a constant acceleration of 2 ms-2.

First, any constants were defined.

u = 0; %metres per second

a = 2; %metres per second^2

Then an array was created for the times, with a 0.1s difference, in the range zero to ten. The velocity for each point was then calculated, using Equation 2.

Figure .: Velocity-Time graph for constant 2ms-2 acceleration

t = linspace(0,10,101); %seconds

v = u + a\*t; %metres per second

The graph was then plotted, and formatted:

plot(t,v,"r") %plots the graph, in red

title("Velocity vs Time"); %adds a title to the graph

xlabel("Time"); %labels the horizontal axis with "Time"

ylabel("Velocity"); % labels the vertical axis with "Velocity"

tmin = 0; % defines the minimum time

tmax = max(t); % defines the maximum time, using the greatest value in t

vmin = 0; % defines the minimum velocity

vmax = max(v); % defines the maximum velocity, using the greatest value in v

axis([tmin tmax vmin vmax]) % defines the horizontal and vertical axis ranges

## Displacement under constant acceleration (I)

Chart

Description automatically generated with medium confidenceIn this activity, the task was to plot a graph, onto MATLAB [14], which represents displacement under a constant acceleration of 2 ms-2.

First, any constants were defined.

s0 = 0; %metres

u = 0; %metres per second

a = 2; %metres per second^2

Then an array was created for the times, with a 0.1s difference, in the range zero to ten. The displacement for each point was then calculated, using Equation 3.

Figure .: Displacement-Time graph for constant 2ms-2 acceleration

t = linspace(0,10,101); %seconds

s = u\*t + 0.5\*a.\*t.\*t + s0; %metres

The graph was then plotted, and formatted:

plot(t,s,"r") %plots the graph, in red

title("Displacement vs Time"); %adds a title to the graph

xlabel("Time"); %labels the horizontal axis with "Time"

ylabel("Displacement"); % labels the vertical axis with "Displacement"

tmin = 0; % defines the minimum time

tmax = max(t); % defines the maximum time, using the greatest value in t

smin = min(s); % defines the minimum displacement

smax = max(s); % defines the maximum displacement, using the greatest value in s

axis([tmin tmax smin smax]) % defines the horizontal and vertical axes

## Displacement under constant acceleration (II)

In this activity, the task was to plot a graph, onto MATLAB [14], which represents displacement under a constant acceleration of 2 ms-2. The graph stops once the displacement reached is 5m.

Graphical user interface

Description automatically generated with low confidenceFirst, any constants were defined.

s0 = 0; %metres

u = 0; %metres per second

a = 2; %metres per second^2

smax = 5; % maximum displacement

tincrement = 0.1;

Then an array was created for the times. The displacement for each point was then calculated, using Equation 3. The loop calculates the next displacement and checks if the limit is reached.

Figure .: Displacement-Time graph (II) for constant 2ms-2 acceleration

t = [0]; %seconds

s = [0]; %metres

for m = 1:(10/tincrement)

t(end + 1) = m\*tincrement;

s(end + 1) = u\*t(end) + 0.5\*a\*t(end)\*t(end) + s0;

if max(s) >= smax

break

end

end

The graph was then plotted, and formatted:

plot(t,s,"r") %plots the graph, in red

title("Displacement vs Time"); %adds a title to the graph

xlabel("Time"); %labels the horizontal axis with "Time"

ylabel("Displacement"); % labels the vertical axis with "Displacement"

tmin = min(t); % defines the minimum time

tmax = max(t); % defines the maximum time, using the greatest value in t

smin = min(s); % defines the minimum displacement

%smax has already been defined above

axis([tmin tmax smin smax])

## Displacement under constant acceleration (III)

In this activity, the task was to plot a graph, onto MATLAB [14], which models a particle undergoing constant acceleration for 0.5s, and then travels at a constant velocity until 5m is travelled.

First, any constants were defined:

s0 = 0; %metres

u = 0; %metres per second

a = 2; %metres per second^2

v = a\*0.5; %this the velocity when time = 0.5 (when linear velocity begins)

smax = 5; %the maximum displacement in the modelling

tincrement = 0.1; %the time increment from which the measurements are taken

Chart, line chart

Description automatically generatedThen an array was created for the times, with a 0.1s difference, in the range zero to six. The displacement for each point was then calculated, using Equation 3. slin0 is the displacement at the start of constant velocity. The loop calculates the next displacement and then checks if the limit is reached, and then breaks when necessary.

Figure .: Displacement-Time graph (III) for constant 2ms-2 acceleration, then linear velocity

t = linspace(0,0.5,6); %generates the first set of t values

s = u\*t + 0.5\*a.\*t.\*t; %generates the displacement for part 1

slin0 = s(end); %the displacement achieved in 2nd part

for m = 6:(10/tincrement)

t(end + 1) = m\*tincrement; %generates the next time

s(end + 1) = v\*t(end) - slin0; % finds the next displacement

if max(s) >= smax %breaks the loop when 5m reached

break

end

end

The graph was then plotted, and formatted:

plot(t,s,"r") %plots the graph, in red

title("Displacement vs Time"); %adds a title to the graph

xlabel("Time"); %labels the horizontal axis with "Time"

ylabel("Displacement"); % labels the vertical axis with "Displacement"

tmin = min(t); % defines the minimum time

tmax = max(t); % defines the maximum time, using the greatest value in t

smin = min(s); % defines the minimum displacement

%smax has already been defined above

axis([tmin tmax smin smax])

## Angular Displacement under Constant Acceleration

Chart

Description automatically generated with medium confidenceIn this activity, the requirements were to plot a graph of angular displacement when a constant angular acceleration of 0.5 rad s-2 was given. The plot would end when 90 degrees has been travelled. Even though this is angular acceleration, the same equations can be used, to their angular definition, because the acceleration is constant.

First, any constants were defined:

s0 = 0; %metres

u = 0;

a = 0.5; %metres per second^2

smax = 90;

tincrement = 0.1;

Figure .: Angular Displacement/Time 0.5 rad s-2 acceleration

Then the displacements are found sequentially. If the next displacement is past the limit, no more displacements are found.

t = [0]; %seconds

s = [0]; %metres

for m = 1:(360/tincrement)

t(end + 1) = m\*tincrement; %seconds

s(end + 1) = u\*t(end) + 0.5\*a\*t(end)\*t(end) + s0; %metres

if max(s) >= smax

break

end

end

The graph was then plotted, and formatted:

plot(t,s,"r") %plots the graph, in red

title("Angular Displacement vs Time"); %adds a title to the graph

xlabel("Time"); %labels the horizontal axis with "Time"

ylabel("Angular Displacement"); % labels the vertical axis with "Angular Displacement"

tmin = min(t); % defines the minimum time

tmax = max(t); % defines the maximum time, using the greatest value in t

smin = min(s); % defines the minimum displacement

%smax has already been defined above

axis([tmin tmax smin smax])

All these code files can be found, in their entirety by following this [link](https://uniofnottm-my.sharepoint.com/:f:/g/personal/eeymg2_nottingham_ac_uk/Eh3FiTOXdONFq0YWE_YWqswBaQ4OVa3UAOe-20ZuRk8YFA?e=SoQilU).

# Lab Work

## Rotary Encoder

To familiarise with the rotary encoder before the challenge, an application of the rotary encoder, where the autonomous vehicle was set to travel 650 counts (about 10m) before stopping, was developed, where an ESP32 [2] was used. Pins A and B were connected to microcontroller pins 26 and 27.

A picture containing text, indoor, monitor, desk

Description automatically generated

Figure .: An image of the Rotary Encoder activity

For access to the full video, where Figure ‎3.A was obtained, please refer to this [link.](https://uniofnottm-my.sharepoint.com/:v:/g/personal/eeymg2_nottingham_ac_uk/EST3mXJGgwZIt-HnHo4NyQ8BTpgp9pGNqmLCXxjLDYJKrA?e=gexyHo)

In the background console, the encoder count is incrementing. As soon as the count reaches 650, the wheels stop.

For the code used, please refer to Appendix 1.

## HC-SR04

Graphical user interface, text, application, Word

Description automatically generatedA picture containing text

Description automatically generatedTo familiarise with the ultrasonic sensor before the challenge, an application of the HC-SR04, where an LED’s brightness depends on the distance measured by the Ultrasonic Sensor was developed, where an Arduino Nano [1] was used. The trigger pin was attached to pin D11 of the microcontroller and the echo pin was attached to pin D12. The LED was connected to D3 and the common ground of the system.

Figure .: Console during the HC-SR04 [8]/LED activity

Figure .: Photo of the results of the HC-SR04 [8]/LED activity

For access to the full video, where this experiment was conducted please refer to this [link.](https://uniofnottm-my.sharepoint.com/:v:/g/personal/eeymg2_nottingham_ac_uk/Ef8AT-qQYFJDr-ZDyY_i_TgB_3kw14n8QirAa4OJ0FGSuQ?e=lGeXo7)

In the console above, the distance is first given, and then the value inside the fadeValue variable, which stores the ratio of the voltage supplied to the LED (maximum value of this variable is 255). As the hand moved away, the distance between the hand and the HC-SR04 [8] increased, and so did fadeValue. Therefore, the brightness of the LED increased.

For the code used, please refer to Appendix 2

## IMU

To familiarise with the IMU before the challenge, an application of the IMU, where an LED’s brightness depends on the rotation around the z-axis of the IMU, was developed, where an Arduino Nano [1] was used. The pins were attached to the default pins on an Arduino Nano [1] which can be found under the heading IMU to measure motion, in this document.

A picture containing text, indoor, computer, person

Description automatically generated

Figure .: An image of the IMU activity

In the console above, as the angular displacement (around the z-axis) of the IMU, from the initial position, increased, it directly proportionally increased the value inside the fadeValue variable, which stores the ratio of the voltage supplied to the LED (maximum value of this variable is 255). Therefore, the brightness of the LED increased. For access to the full video, please follow this [link](https://uniofnottm-my.sharepoint.com/:v:/g/personal/eeymg2_nottingham_ac_uk/EbZMeY1kU4xJso8ipPyUfewBkZkNc3nm1S8jCnikEtOUxw?e=aD1pE4).

For the code used, please refer to Appendix 3.

## Reverse Parking Manoeuvre Challenge

The code required for the slave in communication was given as support, by the teaching staff, and can be found by following this [link](https://uniofnottm-my.sharepoint.com/:f:/g/personal/eeymg2_nottingham_ac_uk/EhIjTeXGkn9BukVqb5G8YegBGeX5milkQGdWwX2pYYTWag?e=P2QgGK).

As an example of an automatic process conducted by the EEEBot, the requirements were to make the vehicle perform the following tasks:

1. Drive for one second
2. Rotate 180 degrees anticlockwise about the z-axis
3. Reverse until the vehicle is ten centimetres away from the closest obstacle
4. Rotate a further 90 degrees anticlockwise in the same direction
5. Reverse again until the vehicle is ten centimetres away from the closest obstacle

To achieve these, the following sensors are used:

* The HC-SR04 [8] gives data about how far the car is from the closest obstacle.
* MPU9250 [10] returns how far the vehicle has turned.

The pin connections between the peripherals and the microcontroller are the same as they were in the previous examples.

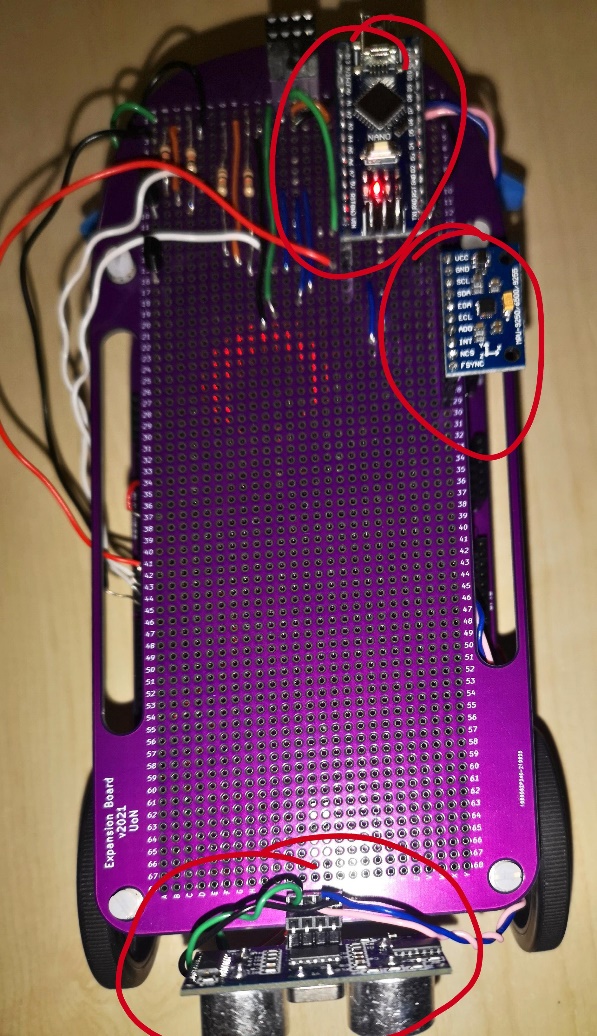


Figure .: The EEEBot's main parts are circled

### Programming the microcontroller

In this section, the different parts of the Arduino Nano [1] program are described.

First, the following libraries were used for this exercise:

#include <MPU6050\_tockn.h> //IMU

#include <NewPing.h> // HC-SR04

#include <Arduino.h>

#include <WirePacker.h> //to send data to ESP32

#include <Wire.h>

To use the HC-SR04 [8], first, the microcontroller pins that the HC-SR04 [8] was connected to are defined. Also, the address of the ESP32 [2], on the I2C bus needed to be defined, for the Arduino Nano [1] to communicate with it.

The IMU and sonar sensors are initiated.

// defines for the HC-SR04

#define TRIGGER\_PIN 11

#define ECHO\_PIN 12

#define MAX\_DISTANCE 100

#define I2C\_SLAVE\_ADDR 0x04 // ESP32 Address

MPU6050 imu(Wire); // IMU connected to the I2C Wire

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE); // HC-SR04 established connection

Some variable declarations for later use…

//determines the delay, until the car starts moving forward for the manoeuvre

unsigned int startForwardMotion = 10 \* 1000;

unsigned int endForwardMotion = startForwardMotion + 1000;

//variables used for conditional statements later, to advance to the next parts of the manoeuvre

unsigned int doRotate\_1 = 1;

unsigned int doReverse\_1 = 1;

unsigned int doRotate\_2 = 1;

unsigned int doReverse\_2 = 1;

The Arduino needs to wait for the gyroscope to calculate its offset, and the initial z-angle is calculated.

void setup()

{

  Serial.begin(115200);

  Wire.begin();

  imu.begin();

  imu.calcGyroOffsets(true); //calibrates the gyroscope on the IMU

}

int startAngle = (int)imu.getAngleZ(); // finds the angle, which the car is initially in

Now, the main loop will be described:

There are 5 tasks that the vehicle needs to complete to fully complete the manoeuvre. For each of these tasks, there is an accompanying if statement. Once the task has been completed, the next if statement is called in the next loop.

void loop()

{

  int \_time = (int)millis(); //finds the time since the start

  if (\_time <= startForwardMotion) {

    slaveWrite(0, 0); // no motion until the end of the delay

  }

  else if (\_time > startForwardMotion && \_time < endForwardMotion)

  {

    Serial.println("start");

    slaveWrite(255, 255); // makes the car go forward, for 1 second.

  }

  // goes through all of these if statements.

  // when the task finishes, zero is returned, so it will be skipped in the next loop

  else if (doRotate\_1 == 1)

  {

    // rotates the car 180 degrees anticlockwise

    doRotate\_1 = rotate(startAngle + 180);

  }

  else if (doReverse\_1 == 1)

  {

    // reverses the car until the closest obstacle is 10cm away

    doReverse\_1 = reverse(10);

  }

  else if (doRotate\_2 == 1)

  {

    //turns a further 90 degrees anticlockwise

    doRotate\_2 = rotate((startAngle + 180) + 90);

  }

  else if (doReverse\_2 == 1)

  {

    // reverses the car till the closest obstacle is 10cm away

    doReverse\_2 = reverse(10);

  }

  else {

    // when everything has been conducted, puts into an infinite loop

    Serial.println("end");

    while (1) {}

  }

  delay(500);

}

There are several function calls in this code. These can be found with the full code [here](https://uniofnottm-my.sharepoint.com/:f:/g/personal/eeymg2_nottingham_ac_uk/EqdmEbrlH6xGkC1PN5P4bDsBszJ-gJ8M3WZST7uE-TonWw?e=QmcBjY).

### Trials and results

During this section, the trial of the system will be described, with the problems encountered, and how they were overcome. The following table summarises the trial steps:

|  |  |  |
| --- | --- | --- |
| Test | Results | Attempted solutions |
| Communication between the microcontrollers | **Not working** | * Checked example programs supplied on Moodle (no problem with the code) * Checked wiring and connections and all were visibly perfect * Visit to the lab **technicians**, but nothing could be done before the 17th January |
| There seems to be no communication between the Arduino Nano [1] and the ESP32 [2], for no known reason. Possibly a hardware problem. Because of this, a different approach had to be taken. To see if the Arduino Nano [1] was sending the correct signals to the ESP32 [2], alternative tests have been planned.  Alternative tests (results shown in the video): | | |
| Testing that the HC-SR04 [8] sensor tells the Arduino Nano [1] to stop 10cm away | **Working** | The data in the video shows that the car reverses until the distance between it and an obstacle is 10cm. |
| Testing that the gyroscope detects when 90 degrees has been turned | **Working** | The data in the video shows that the car roughly does the appropriate next steps when the car rotates the set amount. |
| The motors run according to the ESP32 [2] signals | **Working** | When the motors are supplied with signals from the ESP32 [2], they behave accordingly. |
| In conclusion, because of the lack on measuring instruments at home, I couldn’t solve the hardware problem. And the lab technicians couldn’t help in due time. The microcontrollers couldn’t communicate even using the provided example code on Moodle.  Communication simulation:  The data obtained at the master during the alternative tests, could be used, in a loop, as input, in the slave program to simulate the receiving of data and demonstrate how the vehicle would behave. This couldn’t be achieved as I don’t have an appropriate storage on the master. | | |

A machine on the floor

Description automatically generated with low confidence

Figure .: Image of the manual manoeuvre

The video of the “manoeuvre”, in greater quality to the image above can be found through this [link](https://uniofnottm-my.sharepoint.com/:v:/g/personal/eeymg2_nottingham_ac_uk/ESBwvWOH2gdEtfUqPxpv5wEB7an5Cb9VlMLXxw1UmR0xkw?e=l3LxdA).

# Conclusion

This has been a rather successful laboratory, where sensors were used to control the manoeuvring of an autonomous vehicle. Despite the microcontrollers not being able to communicate with each other in the end, the final manoeuvre challenge could still work as expected if the circumstances were slightly different. This has been confirmed by the success of the sensor, trial run. This communication problem will be fixed as soon as possible.

We have seen how sensor data could be implemented in the code, and how such data can be processed to calibrate the data entered to better represent real-world dimensions.

However, techniques on how to improve the accuracy of the existing sensors could have been employed, so that the sensors are more sensitive to changes, and are less likely to cause a collision. This is vital, due to the nature of any vehicle in a real-world situation, where a collision could be fatal. The lack of precision demonstrated by the IMU (in the manoeuvre video), where the given angle was quite different to what was visually expected is an example of this.

An example of a source of problems is how the ultrasonic sensor is not perfectly perpendicular to the car surface, as can be seen in Figure ‎3.E. This could mean that the returning ultrasonic waves are not returned at their fastest time, creating a false impression that the distance is greater than it is.

Another reason for the lack of accuracy could be the weak connections that the sensors have with their female pin headers. Due to this, the sensors can pivot about the pin headers, and this may distort the readings from the sensors.

Also, some research could be done to find more robust sensors that can measure data more accurately, and measure more samples of data, with more precision.

Considering the rate of change of momentum, the abrupt stops which the vehicle is made to make, as a reaction to the sensor readings, can cause a lot of force, which can damage the autonomous vehicle in the long run. Ideally, the vehicle should be made to decelerate slowly as it approaches a stop so that this force can be greatly reduced, as the momentum decreases gradually.

# Table of Figures

[Figure ‎1.A: The levels of autonomy [4] 4](#_Toc92714130)

[Figure ‎1.B: A rotary Encoder attached to the EEEBot’s wheels 6](#_Toc92714131)

[Figure ‎1.C: A HC-SR04 [8] attached to the back of an EEEBot 7](#_Toc92714132)

[Figure ‎1.D: An MPU9250 [10] attached to the EEEBot 8](#_Toc92714133)

[Figure ‎1.E: A diagram demonstrating a system using the I2C bus [11] 9](#_Toc92714134)

[Figure ‎1.F: The coordinate axes given on the MPU9250 [10] 9](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714135)

[Figure ‎2.A: Velocity-Time graph for constant 2ms-2 acceleration 10](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714136)

[Figure ‎2.B: Displacement-Time graph for constant 2ms-2 acceleration 11](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714137)

[Figure ‎2.C: Displacement-Time graph (II) for constant 2ms-2 acceleration 12](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714138)

[Figure ‎2.D: Displacement-Time graph (III) for constant 2ms-2 acceleration, then linear velocity 13](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714139)

[Figure ‎2.E: Angular Displacement/Time 0.5 rad s-2 acceleration 13](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714140)

[Figure ‎3.A: An image of the Rotary Encoder activity 15](#_Toc92714141)

[Figure ‎3.B: Console during the HC-SR04 [8]/LED activity 15](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714142)

[Figure ‎3.C: Photo of the results of the HC-SR04 [8]/LED activity 15](https://uniofnottm-my.sharepoint.com/personal/eeymg2_nottingham_ac_uk/Documents/Year%201/Applied%20Electrical%20and%20Electronic%20Engineering%20Construction%20Project/Reports/Report%201/GHALI_20381397_EEEE1002LR1.docx#_Toc92714143)

[Figure ‎3.D: An image of the IMU activity 16](#_Toc92714144)

[Figure ‎3.E: The EEEBot's main parts are circled 17](#_Toc92714145)

[Figure ‎3.F: Image of the manual manoeuvre 20](#_Toc92714146)

# Appendices

## Appendix 1: Code for Rotary Encoder Activity

#include <ESP32Encoder.h>

#define PWMa 27 // Pin connected to left motor

#define PWMb 26 //Pin connected to right motor

const int frequency = 5000; // frequency of pulses

const int resolution = 10; // resolution of voltages to select

const int PWMChannelA = 0; // channel which sends signal to PMWa pin

const int PWMChannelB = 1; // channel which sends signal to PMWb pin

int countime = 650;

ESP32Encoder encoder;

void setup() {

 ESP32Encoder::useInternalWeakPullResistors = UP;

 // initiate encoder use with pins 32 and 33

 encoder.attachHalfQuad(32, 33);

 // reset the encoder count to 0

 encoder.clearCount();

**Serial**.begin(9600);

 // configure the motor control pins as outputs

 pinMode(PWMa, OUTPUT);

 pinMode(PWMb, OUTPUT);

 // establish voltage connection with both channels

 ledcSetup(PWMChannelA, frequency, resolution);

 ledcSetup(PWMChannelB, frequency, resolution);

 // attaches each channel to the pin

 ledcAttachPin(PWMa, PWMChannelA);

 ledcAttachPin(PWMb, PWMChannelB);

}

void loop() {

 signed long encoderCount = encoder.getCount();

 do

 {

   encoderCount = encoder.getCount();

**Serial**.println(encoderCount);

   ledcWrite(PWMChannelA, 1023); //NON-ROTARY

   ledcWrite(PWMChannelB, 1023); //ROTARY

   //every 180 cycles, turns right a bit, to help it go in a straight line (random motion so not fully predictable)

   if ((encoderCount % 180) == 100)

   {

     ledcWrite(PWMChannelA, 0); //NON-ROTARY

     ledcWrite(PWMChannelB, 1023); //ROTARY

     delay(20);

   }

 } while (encoderCount <= countime);

 //when the required distance has been travelled, ends the motion

**Serial**.println("end");

 while (encoderCount > countime)

 {

   stopMotors();

 }

}

void stopMotors() {

 ledcWrite(PWMChannelA, 0); //NON-ROTARY

 ledcWrite(PWMChannelB, 0); //ROTARY

}

## Appendix 2: Code for ultrasonic sensor activity

#include <**NewPing**.h>

#define TRIGGER\_PIN 11

#define ECHO\_PIN 12

#define MAX\_DISTANCE 100

#define LED\_PIN 3 // defines the pin, which the LED is connected to (the LED also connected to ground)

**NewPing** sonar(TRIGGER\_PIN,ECHO\_PIN,MAX\_DISTANCE); // initiating a connection to the ultrasonic sensor

float fadeValue;

void setup()

{

**Serial**.begin(9600);

 pinMode(LED\_PIN, OUTPUT); // sets the microcontroller pin connected to the LED as an output

}

void loop()

{

 delay(50);

 unsigned int distance = sonar.ping\_cm(); // finds the distance, in cm

**Serial**.print(distance);

**Serial**.println(" cm");

 fadeValue = (255 \* ((float)distance/1000)); //uses the distance to find the ratio of voltage to give to the LED light (out of 255)

**Serial**.print("fadeValue = ");

**Serial**.println(fadeValue);

 analogWrite(LED\_PIN, fadeValue); //sends a voltage to the LED

 delay(100);

}

## Appendix 3: Code for IMU Activity

#include <MPU6050\_tockn.h>

#include <Wire.h>

#define LED\_PIN 3 // defines the pin, which the LED is connected to (the LED also connected to ground)

float fadeValue;

MPU6050 mpu6050(Wire); // states that the MPU is connected to the I2C Wire

void setup() {

**Serial**.begin(9600);

 Wire.begin();

 mpu6050.begin();

 mpu6050.calcGyroOffsets(true); // activates the Gyroscope of the MPU

 pinMode(LED\_PIN, OUTPUT); // sets the microcontroller pin connected to the LED as an output

}

void loop() {

 mpu6050.update(); // gets new readings from the MPU

 float angle = abs(mpu6050.getAngleZ()); // gets the angle around the z-axis

 fadeValue = 255 \* (angle/1000);

**Serial**.print("Angle: ");

**Serial**.println(angle);

**Serial**.print("Fade Value: ");

**Serial**.print(fadeValue);

 analogWrite(LED\_PIN, fadeValue); //sends a voltage to the LED

}

# References

|  |  |
| --- | --- |
| [1] | Arduino, “Arduino Nano,” Arduino, [Online]. Available: https://store.arduino.cc/products/arduino-nano. |
| [2] | “DOIT ESP32 DEVKIT V1,” [Online]. Available: https://makeradvisor.com/tools/esp32-dev-board-wi-fi-bluetooth/. |
| [3] | A. Forrest and M. Konca, “Autonomous Cars and Society,” Worcester Polytechnic Institute, Worcester, 2007. |
| [4] | Synopsys, “The 6 Levels of Vehicle Autonomy Explained,” [Online]. Available: https://www.synopsys.com/automotive/autonomous-driving-levels.html. |
| [5] | Tesla, Inc., “Tesla,” Tesla, Inc., [Online]. Available: https://www.tesla.com/en\_gb. |
| [6] | K. Harrington, “ESP32Encoder,” [Online]. Available: https://www.arduino.cc/reference/en/libraries/esp32encoder/. [Accessed 5 January 2022]. |
| [7] | ElecFreaks, “Ultrasonic Ranging Module HC - SR04,” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed 2022 January 5]. |
| [8] | LastMinuteEngineers.com, “How HC-SR04 Ultrasonic Sensor Works & Interface It With Arduino,” [Online]. Available: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/. |
| [9] | T. Eckel, “NewPing,” [Online]. Available: https://www.arduino.cc/reference/en/libraries/newping/. |
| [10] | “MPU-9250,” [Online]. Available: https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/. |
| [11] | SparkFun Electronics, “I2C,” [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all. |
| [12] | Tockn, “MPU6050\_tockn,” [Online]. Available: https://www.arduino.cc/reference/en/libraries/mpu6050\_tockn/. |
| [13] | Arduino, “Wire Library,” [Online]. Available: https://www.arduino.cc/en/reference/wire. |
| [14] | MathWorks, “MATLAB,” [Online]. Available: https://uk.mathworks.com/products/matlab.html. |
| [15] | Espressif, “ESP32 Board on Arduino IDE (DOIT ESP32 DEVKIT V1),” [Online]. Available: https://github.com/espressif/arduino-esp32. |