2e10 Final Report

Part A – Self Assessment

- Top 3 strengths

Communication: As a team our effective communication was one of our biggest strengths, throughout the completion of this project. We utilised various communication methods that aided us to complete all three of the challenges, some of which were the creation of a WhatsApp group chat which was used throughout the project for quick communication between all team members. This form of communication allowed us to share updates, ask questions and schedule meetings outside of labs to discuss any issues or assign new tasks for each group member. The two labs we had each week were used as a time for us to meet together as a group and work towards completing each challenge, as well as a time to delegate tasks for each member for the week. Any problems a group member had were also raised in the labs which resulted in us becoming a unified group capable of tackling any issues that arose.

Time Management: Each individual member of the group displayed exemplary time management skills while undertaking this project. There were many benefits to our time management skills, some of which were improved productivity, increased quality, improved teamwork and the ability to meet deadlines for each challenge. As a group we feel our time management skills were one of the main reasons we were able to complete the gold challenge. Time management skills helped to increase our productivity by encouraging members to stay focused and motivated by setting a clear plan and timeline for completing tasks. This also had an effect on the quality of work each member was presenting to the group, as each member was allocated plenty of time to complete their individual tasks. As a result of each team member being aware of the project timeline and their tasks it consequently reduced the stress of each member and enhanced our teamwork. All these benefits allowed us to meet all deadlines with good quality work that we as a group were proud to submit.

Flexibility: Flexibility as a group was another one of our main strengths. One of the key benefits to having flexible group members is that it allows each member to work effectively with people who have different personalities to themselves. Each member of the team has their own strengths and weaknesses, by being flexible we were able to accommodate these differences and found ways that we could work together as a team and utilise everybody's strengths. Another benefit to our group being flexible was any time deadlines weren't met due to unforeseen circumstances such as illness or the task being more difficult than previously expected, we were able to come together as a group to try and resolve the issue and offer help when needed. As a group we were also open to change and willing to try new things, this was particularly beneficial when brainstorming ideas for the gold challenge, as it allowed us to choose an idea that suited us as a group.

- 3 weaknesses

Knowledge about Arduino: As a group we feel that we did not have many weaknesses throughout the completion of this project. However, as a result of not having used Arduino to a sufficient degree before, there were times as a group we had to try and learn various new aspects of this software. As a result of spending time learning a new skill, time was taken away from various other aspects of the project such as working towards the bronze challenge by writing code that implemented the various requirements for the challenge. As a group we feel that we tackled this weakness well, as we were all available to help and guide each other through the learning of this software. This allowed us to overcome the learning curve quickly, allowing for more time to be spent on other aspects of the project that required attention.

Inadequate resources: While attempting to complete the silver challenge our group was faced with an inconvenience of non-functioning equipment. The motor of one our buggy's stopped working, at first we thought that this was a result of improper code and spent a lot of time investigating the code and trying different things in the hopes of fixing the motor. Later we discovered that there was nothing wrong with the code and that it was the motor that was the issue, this was important for us to discover as we had spent a lot of time focusing on this area of the project when we had other important deadlines that were approaching. Another inconvenience we were tasked with was that one of our group members computers stopped allowing us to upload code to the buggy. This was less of a problem as we had three other members of the group with computers that were able to upload code to each of the buggy's. Despite these setbacks we managed to group together as a team to complete our various tasks.

Announcing changes made: Another weakness we had as a group came at the beginning of the project where we were not keeping track of the changes each member was making to the buggy in both a coding and electronics sense. This made it difficult for members of the team to keep working on individual sets of code for the project as changes were made that they were unaware about. We soon realised that this was becoming a greater issue and quickly became better at communicating the changes that each member was making to the buggy. This resulted in exponential progress in each challenge as no time was lost to attempting to discover changes that were made to the code or electronics of the buggy.

Three changes to our approach and design for the Silver Challenge that would allow us to achieve the Gold challenge if given two more weeks:

Processing: For the Silver challenges we sent all three variables to processing on one port. For example, we sent the Encoder speed the Ultrasonic speed and the distance on the one port which meant it was received on the other end in the same order it was sent out as. However, for the Silver Challenge, this did not work very well. This meant we had to spend more time during the Gold Challenge changing this so that the three variables could be sent on three different ports. If we changed this during the Silver Challenge, it would allow us more time to improve our buggy for the Gold Challenge.

PID: After reading week we only had a short amount of time before the Silver challenge was due. This meant that some aspects of the buggy had to take priority. One that did not was in relation to the tuning of the P.I.D. At this time, we did not realise how important the tuning of

the P.I.D. would be in our aspirations to pass the Gold Challenge. This meant we spent a lot of time during the Gold Challenge to fix and understand this and if given two more weeks, we feel that this change would allow us to achieve the Gold Challenge.

Time Management: Our approach and ideas for the Silver and Gold Challenges mainly took place inside the designated lab slots on Monday and Wednesday afternoons. As previously stated, a lot of our problems could have been solved if we had longer to do the Silver Challenge. Looking back, a way we could have found a solution for this was simply just to meet outside of the designated lab slots. This would have given us more time to understand the buggy and conjure ideas for the Gold Challenge, which would have helped create priorities in the Silver Challenge that would in future help us for Gold, such as the tuning of the P.I.D. and our problem with processing.

Ethical Issues

Using artificial intelligence will always lead to possible ethical issues. In our case, the main two questions regarding ethical misuse are if the buggy is being abused and whether it stores and collects sensitive data.

Throughout the design process, the group has considered these issues and taken note if any happen. Fortunately, there was never a worry about the buggy collecting and storing sensitive data. In terms of abuse of the buggy, nothing suggests that the buggy was being abused. While parts were broken such as the motor, H-bridge and power bank. It was not due to the abuse of the buggy but simply through wear and tear.

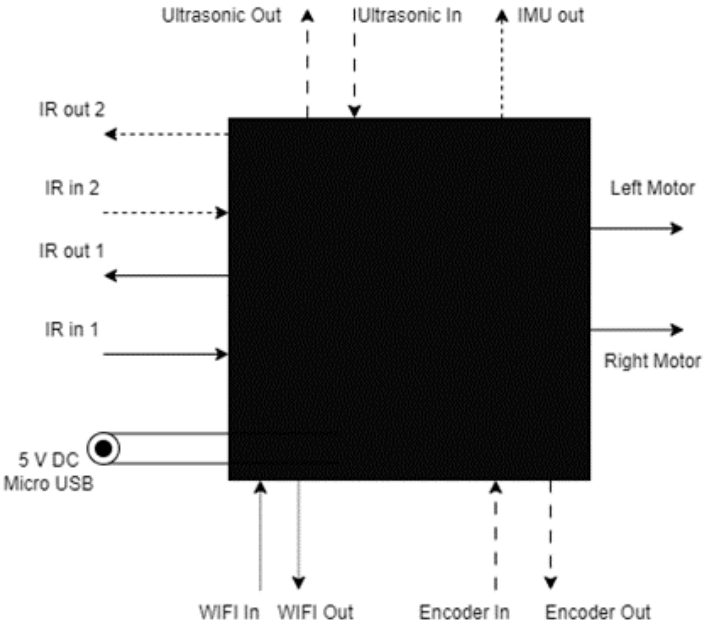
To allow for complete design optimisation, the buggy parts could be replaced year after year to allow the students a proper chance to attempt the Gold Challenge. However, in this day and age, throwing away working parts is ethically incorrect in terms of recycling and making a clean world. This means that the parts of the buggy are reused and would often undergo a lot of wear and tear causing some of them to be faulty.

In our case, we faced a couple of incidents where the parts became faulty which resulted in time being wasted in figuring out what parts were faulty and which new we needed to get. Ethically the right thing to do is to continue reusing parts every year. However, it would work better if new parts were given to us.

In this case, the buggy collected information and data through the IR sensors and ultrasonic sensor. However, in similar projects the buggy could be equipped with different types of sensors or other data collecting devices which could cause concerns about the collection and use of personal data. It is important to note the equipment we are using and the data we are collecting to ensure no sensitive or private data is being collected.

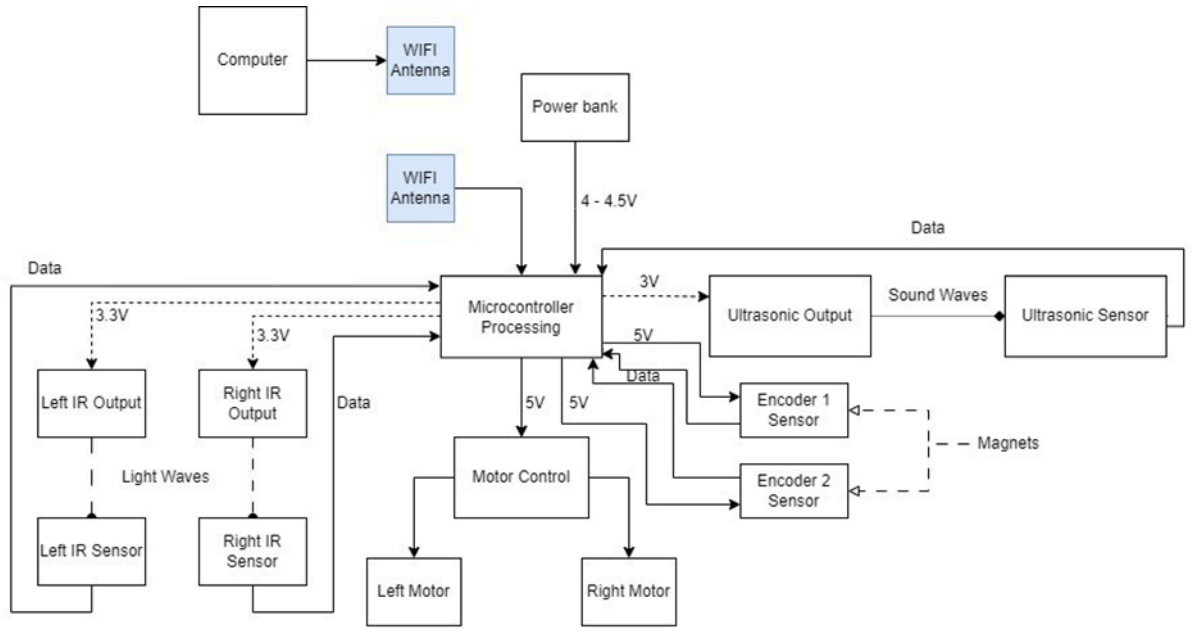
2E10 Final Report (Part B)

Level 0 Block Diagram



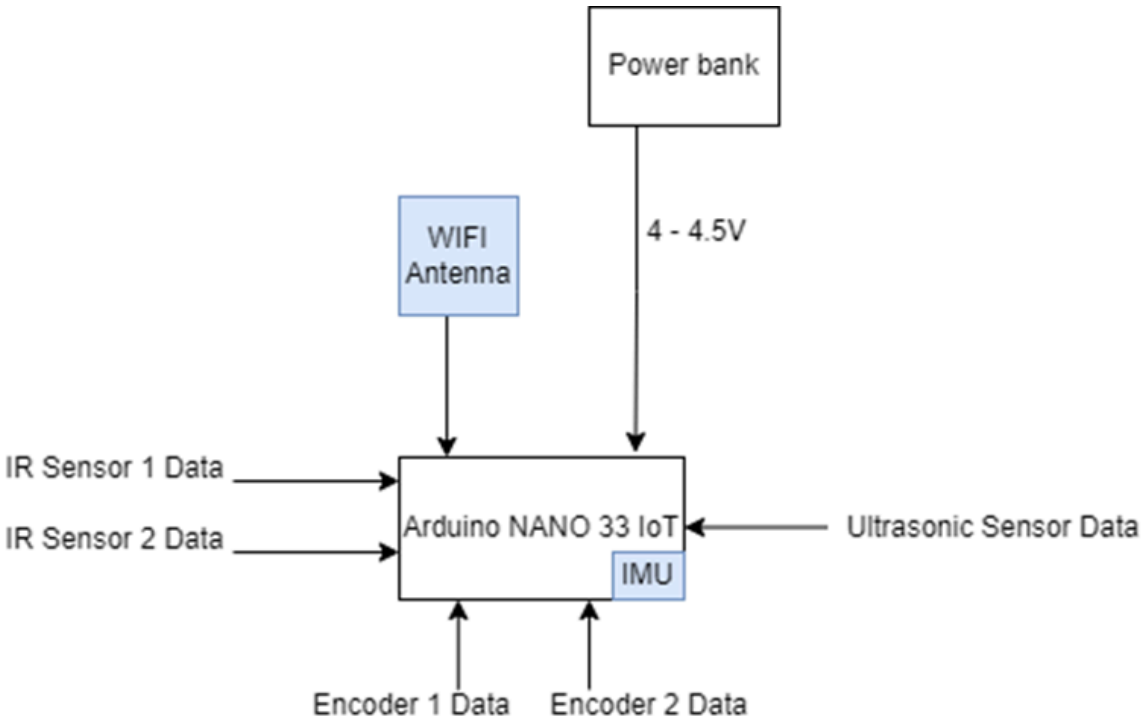
This diagram allowed us to gain an overall view of what signals needed to be sent and received from our buggy for the gold challenge.

Level 1 Block Diagram



Using our Level 0 diagram & our Interim Report Diagram as a base, we were able to create this diagram. It made us consider how our buggy would communicate with the control computer and also determine what data the processing component would need access to in order to complete our gold challenge.

Level 2 Block Diagram



This diagram was created to allow us to further visualise how the processing unit would operate. The data processing is handled by the Arduino code.

Circuit Schematic

Below are the Schematic and the Breadboard Diagrams.

We originally wired the IR Sensors to Digital Pins 2 & 3 to ensure that we could implement interrupts in our code if we felt they were necessary. However we originally used interrupts in the operation of the encoders so they were moved to A0 & A6. We used pins 7 & 12 for the H-Bridge to allow us to implement PWM control. We then filled the other sensors around these requirements as there were no more requirements necessary at this stage.

As shown in the Breadboard Diagram, which is colour coordinated for each sensor, we kept the wires square as we felt it was neater. We kept connections for both IR sensors beside each other for ease of understanding. We also chose to only put the 5V signal through the centre power bar, to ensure that the Arduino board was not accidentally burnt out by a misplaced wire.

We believe that at this point we have created a tidy and clear breadboard that is understandable for other users. We were happy that the choices we had made for wiring in the bronze and silver challenges made it easy for us to implement our plans for gold without much adjustment.

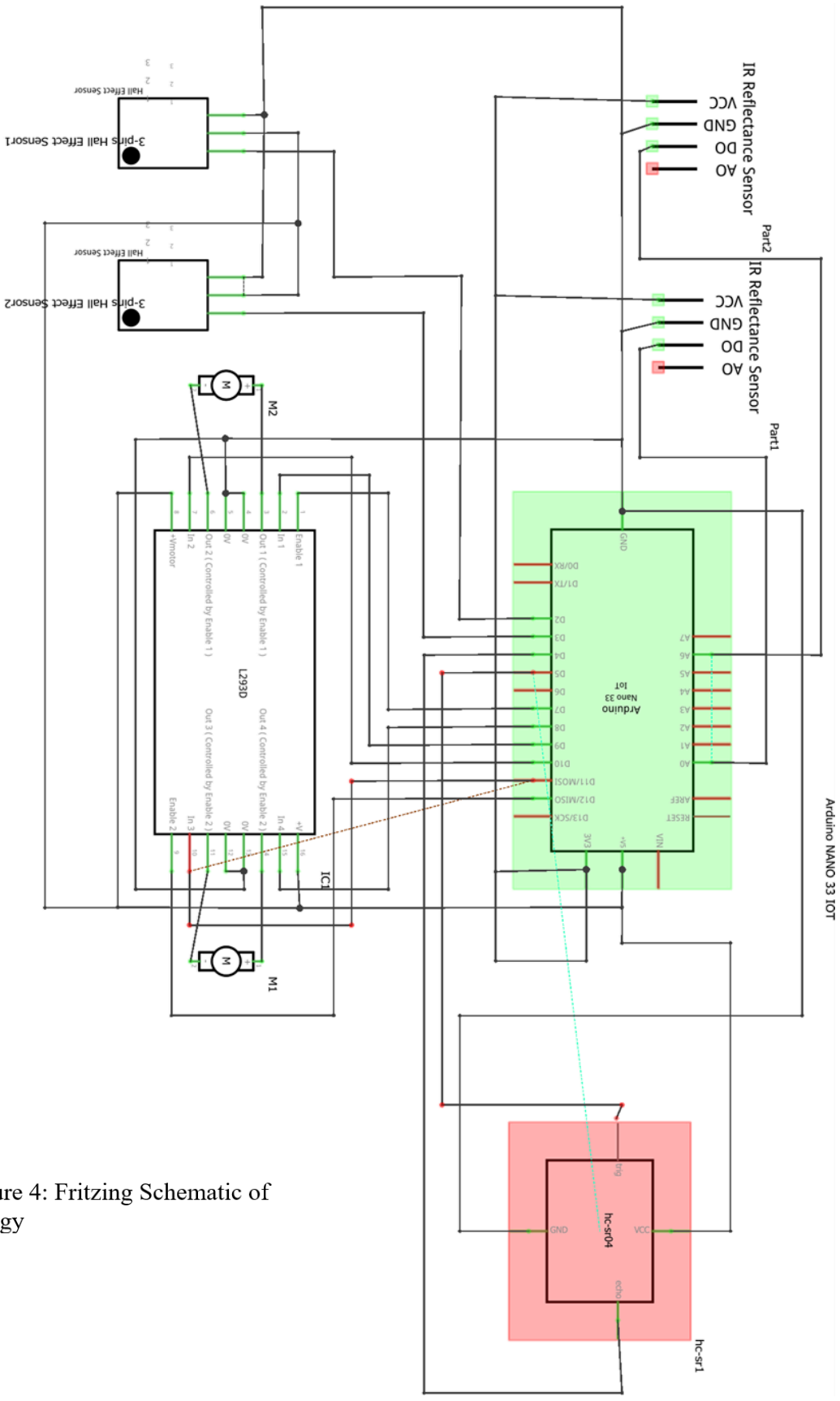


Figure 4: Fritzing Schematic of Buggy

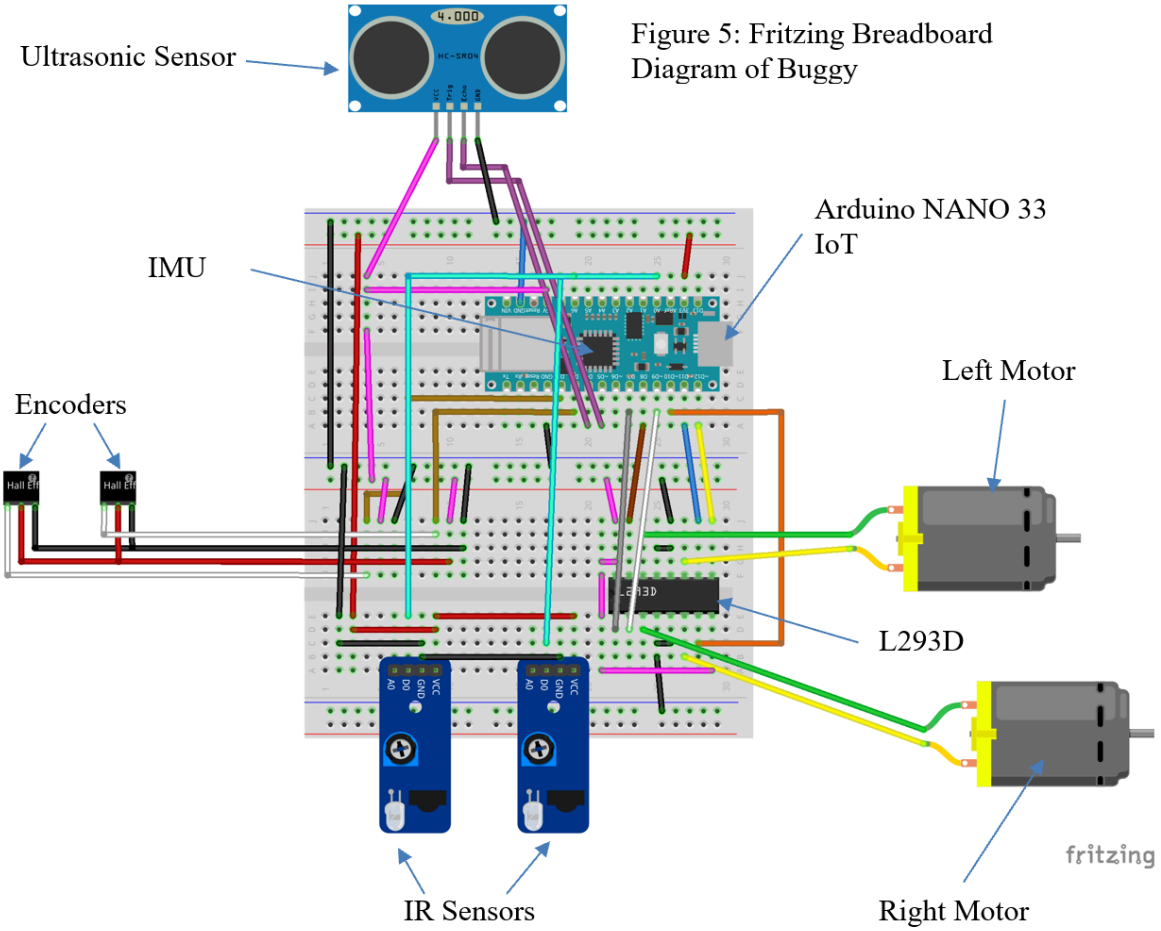


Figure 5: Fritzing Breadboard Diagram of Buggy

Challenges in Design Implementation

Since we had planned our buggy's design prior to implementing it on the breadboard, we did not encounter many challenges with the implantation of the circuit. However, the challenges that were encountered are listed below:

- Broken Motor

  On one of the kits received, the motors were not functional. This caused delays in the implementation of the system as both kits were expected to be functional. After troubleshooting using the other kit, the kit was replaced but this meant the buggy had to be rewired. If redoing the project this could have been prevented by testing both kits prior to implementation.

- Broken H-Bridge

In one of the initial kits given the H-Bridge was missing some of the pins. This also caused delays as it was not noticed right away. Again, this could be prevented by inspecting and testing both kits prior to implementing the code on them.

- Wiring H-Bridge for motors

  The group initially struggled with understanding the H-Bridge circuit. This was an easier challenge to overcome as it could be solved by researching further. This didn't cause delays as it was solved before the group wired the circuit on the breadboard.

- Wiring Encoders

  Initially our group were not sure whether the encoders used 3.3V or 5V power. This answer was found in the encoder data sheet. It also took us some time to realise that the encoders needed to have a Pull-up resistor. After research, it was then realised that this could be solved using the Arduinos internal Pull-up.

- Broken Power Bank

  Just before the deadline for the silver challenge one of our buggy's power banks was losing charge very quickly and was therefore unable to power the Arduino. This caused a delay as we had to wait until the next lab to get a replacement power bank.

Overall, we believe the challenges were solved by the group well and did not affect our ability to reach the deadlines set by the module's challenges.

2E10 Final Report (Part C)

HOW TO STAY ON TRACK:

- The Buggy manages to follow the track using the IR sensors located on the front, which are pointing downwards.
- When the left IR sensor detects the line, the buggy knows to turn left. Vice versa for the right sensor

HOW TO STOP FOR OBSTACLES

- Distance can be calculated using the ultrasonic sensor on the buggy
- By measuring the time interval between sending and receiving an ultrasonic pulse, you can calculate the distance to an object in front

HOW TO CONTROL THE BUGGY AND REPORT EVENTS WIRELESSLY

- Thanks to the Arduino's onboard Wi-Fi chip, you can set up a WAN
- A client can then connect to this WAN to both receive and send data
- In our case, the client can send characters to the WAN
- Logic can be set up on the Arduino to perform actions depending on the character received.
- Our Arduino is also set up to report distance, encoder speed & ultrasonic speed values back to the client

HOW TO FOLLOW OBJECTS

- The buggy follows objects using the ultrasonic sensors and a Proportional-Integral-Derivative Controller
- The distance from the Ultrasonic Sensor is passed to the PID function
- The PID function measures the time interval since the last time it has ran
- Using this interval, the distance and constants Ki, Kp & Kd, it returns a value for the motor speed between 0 & 255
- The value is small if the object is close to 20 cm away and larger if the distance is much further than 20 cm away

BUGGY PSEUDO CODE

Include WifiNana Library
Include Encoder.h Library
Define buggy WAN **name,** **password** & port

Define **LeftEye** & **RightEye** pin numbers

Define **Encoder1** & **Encoder2** pin numbers

Define **Ultrasonic Trigger** & **UltraSonic Echo** pin numbers

Define Motor A Connections (**enA**, **in1**, **in2**)
Define Motor B Connections (**enB**, **in3**, **in4**)

//These variables will be used to control the starting and stopping
Initialise & Set bool value **drive** to false
Initialise distance as integer
Initialise duration as long

//These variables will used for keeping time in the Encoder, US and PID calculations
Initialise & Set int value **DistanceState** to LOW
Initialise **previousMillis** & **currentMillis** as unsigned long
Initialise **interval** as const long
Initialise **distance**, **distance1**, **distance2**, **velocity** & **duration** as doubles

//These variables will used for PID Calculations
Initialise & Set double values **Kp**, **Ki**, **Kd** to PID constants found
Initialise **PID** as unsigned double
Initialise **speed** as const long
Initialise **error**, **lastError**, **input**, **output**, **setpoint**, **cumError**, **rateError** as doubles

//These variables will used for Encoders
Initialise & Set double value **oldPosition** = -999 **(To ensure position updates)**, **speedStart** = 0 **(Buggy starts from Stopped Position)**, **newPosition** = 0 **(To ensure position updates)**, **distancePerStep** = 1.25 **(Distance ratio between Encoder Magnet & Wheel)**, **elapsedTime (Time since last run).**
Initialise **Difference** as a double
Initialise **currentTime,** **previousTime** as longs.

//These variables will used for Sending Data to the GUI
Initialise & Set double value **encoderSpeed**, **usSpeed**, **eyeDist** to zero

//This loop will on repeat once at the start
Void Setup(){

Define serial port as 9600

Set PinMode for all pins //All OUTPUT except IR sensor & U.S. Echo pin

Set **in1, in2, in3** and **in4** to low //Set the motors initial state as off

Start Wifi networking using **name** and **password** defined before

Print IP address to Serial Monitor

}

Void loop(){

Set **currentMillis** to the number of milliseconds since the code started running

IF difference between **currentMillis** & **previousMillis** is greater than the **interval** {

Set **previousMillis** to **currentMillis**

IF **DistanceState** is LOW{

Set **DistanceState** to HIGH

CalculateDistance and assign value to **distance1**

}

ELSE{

Set **DistanceState** to high

CalculateDistance and assign value to **distance2**

Calculate **velocity** using distance

Print data to the GUI

Set **Difference** = difference between **newPostition** & **speedStart**

Set **speedStart** = **newPosition**

}

}

IF client is connected to WAN{

Read character "**C**" received from client

IF **C** = "W" then set **drive** to true IF **C** = "S" then set **drive** to false

}

```
        IF drive is true{
                computePID and assign value to PID
                IF PID is less than 255 & greater than zero {
                        speed = PID
                }

                IF speed is greater than 130{//IE. Set max speed to 230 for good turning
                        Set speed = 230
                }

                 IF distance is greater than 20cm{

                        Depending on value returned by EyeMonitor function{

                        Use Action function to perform appropriate action

                                }

                }
        else use Action function to stop

        Read Encoder newPosition
        IF position has changed (oldPosition!= newPosition)
}

Function computePID(double "inp"){
        Calculates a value for the motor speed based off the distance to object from US
sensor
}

Function CalculateDistance(){

        Set US_Trigger to high for short pulse

        Read US_Echo pulse
        Calculate distance to object in front using time interval between pulses

}

Function EyeMonitor(){

        Returns an int depending on Left & Right eye values

  //EG: if right eye and left eye are both high, return the value "3"
}

Function Action(Integer "i", integer "speed"){
```

Uses switch case to perform an action depending on value inserted. The speed of the action is set by the speed.

//EG: if i = 3, speed = 255 Move buggy forward at speed of 255

}

Function **Print** (){

Compute **encoderSpeed** & **usSpeed**

Set **eyeDist** = **distance**

Send **eyeDist**, **encoderSpeed** & **usSpeed** to processing

}

PROCESSING PSEUDO CODE

```
Import ControlP5 library //For buttons
Import processing.net library //For WiFi

Initialise distance, encoderSpeed, usSpeed and stoppingDistance
Define stoppingDistance as 20

Void Setup(){
        Set canvas size

        Set background colour

        Set framerate



        Connect to Arduino's WAN using its IP Address



        Set up Start & Stop buttons (Position, Size, Colour)

        Set up Speedometer/Knob (Position, Range, Size, Initial Value)

        Set up Text Display Areas 1,2 & 3 (Position, Size, Colour)

}

Void draw(){

        Set distance as first value read from client

        Set encoderSpeed as second value read from client

        Set usSpeed as third value read from client


        If distance is not -1{
                Print <"Obstacle detected at:  " distance " cm"> to Text Area 1
                IF distance is less than 20{
                        Print <"Stopped, Obstacle detected at:  " distance  " cm"> to text Area
1
                }

        }
```

```
        IF encoderSpeed is not -1{
                Print <Encoder Speed:  ” encoderSpeed “ cm/s”> to Text Area 2
                Set Speedometer/Knob to encoderSpeed
        }

        IF usSpeed is not -1{
                Print <“UltraSonic Speed: ” usSpeed “ cm/s”> to Text Area 3
        }
}

Function ReactOnEvent(event){
        If event is START button being pushed
                Send “W” to Arduino

        If event is STOP button being pressed
                Send “S” to Arduino
}
```

PROCESSING GUI

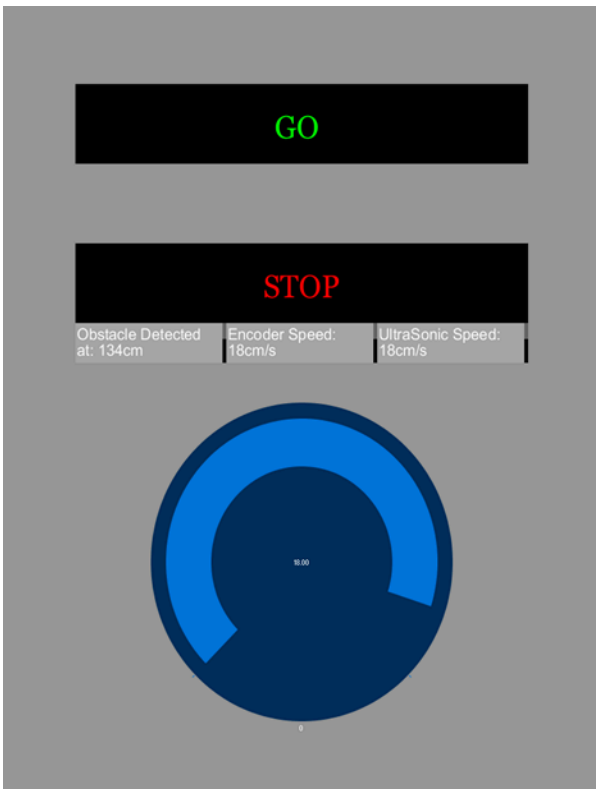


Figure 2: GUI when no obstacle detected

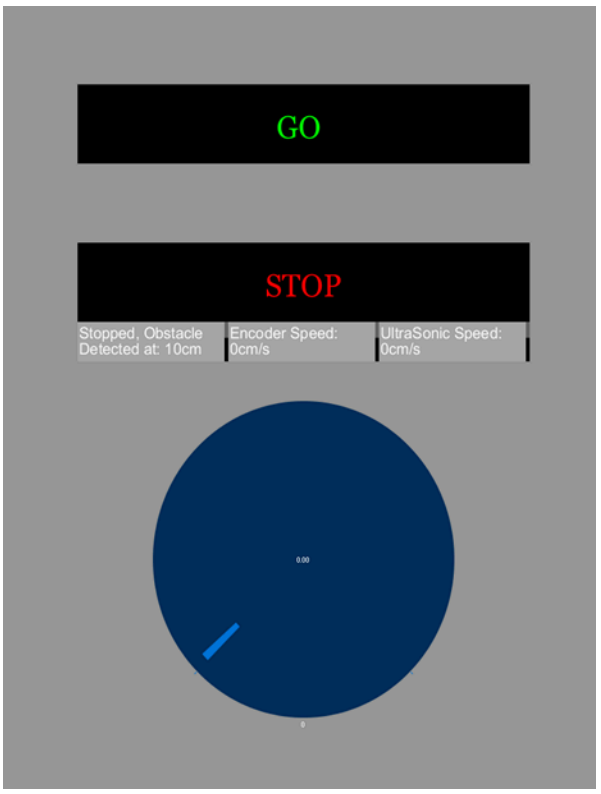


Figure 3: GUI when obstacle detected

As seen above there are 2 buttons to start and stop the buggy. There are also 3 text areas which show the Distance from the US sensor, the Speed measured from the encoders and the Speed of the object moving in front of the buggy (Calculated from the US sensor).

As seen in Figure 3, when an object is detected within 20 cm of the front of the buggy, text area 1 displays “Stopped, Obstacle Detected at: __cm”.

2E10 Final Report (Part D)

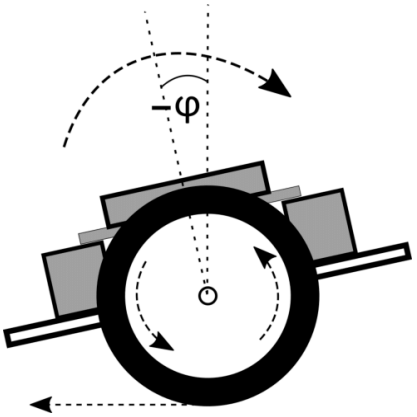
GOLD CHALLENGE IDEA

For the gold challenge, 2 small constraints were given. The buggy must use both the arduino's IMU data, as well as the wheel encoder. We were encouraged to be creative with the orientation/acceleration data. Immediately, we knew we wanted to design a buggy with the ability to balance on 2 wheels. This satisfies our constraints as it heavily relies on the arudino's onboard IMU. The encoder speed could then be reported back to processing. The plan was then to build upon this.

HOW DID WE MAKE THE BUGGY BALANCE

Using the arduino's IMU accelerometer & gyroscope, we could collect data on the buggy's orientation. From experimenting with different angles, we found the angle at which the buggy wants to balance at. This is the angle at which the buggy's centre of gravity (COG) is directly above the wheels. We needed to find a way to keep the buggy at this target angle and to stay in this position.

This was achieved by rotating the wheels in the direction in which the buggy is falling. As seen in the diagram on the right (1), as the buggy tips to the left, the robot attempts to stabilise itself by moving left. Assuming the correct speed is applied, the buggy's angle would return to its target position, and the COG would return back to being directly above the wheels. Finding the correct speed in which to rotate the wheels was a far more challenging task than first thought.



HOW DID WE FIND THE CORRECT MOTOR SPEED

Using a proportional integral derivative (PID) controller, the correct speed can be applied to the motors. The downside of PID controllers is that they require tuning, which can be a tedious process. We heavily underestimated the time we would spend tuning the PID. Unfortunately this hindered our progress.

WHAT FUTURE PLANS DID WE HAVE?

Once we had the PID tuned correctly, we believed we could've easily developed code to move the buggy forward and backward. By altering the target angle a small amount, the buggy would tip in a controlled manner and travel in a chosen direction. We also planned on having the ability to turn while upright. This could've been achieved by independently controlling the speed of each motor. This is similar to how the buggy turned in both the bronze and silver challenge.

REFERENCE: (do properly later)

1. https://www.researchgate.net/figure/The-self-balancing-robot-a-a-schematic-b-the-real-robot-described-in-this-paper_fig1_328243097

BUGGY PSEUDO CODE (GOLD)

Include IMU library
Include Encoder library
Include WifiNina library
Define buggy Wan **name** , **password** .
Define WAN port.

Define Encoder1 & Encoder2 pin numbers

//Variables for PID
Initialise currentTime, previousTime, Interval as long's
Initialise sumOfErrors, Kp, Kd & Ki all as doubles

//Variables for Encoder
Initialise oldPosition, speedStart, newPosition, distancePerStep = 1.25 as long's
Initialise difference & encoderSpeed as ints.

//Boolean variable used to turn on/off the buggy
Initialise bool drive = false

//the angle the buggy balances at
Initialise targetAngle as float = 0.208

Initialise all motor pins
Initialise variable "Speed" as float

Initialise variable motorPower as double

//variables for gyroscope
Initialise accX, accY, accZ as floats.

```
Void Setup(){

        Define Serial port as 9600

        Set PinMode for all pins

        Set motor initial state to off

        Start Wifi network using name and password defined previously

        Print IP address to Serial Monitor
}

Void loop(){
        Set currentMillis to the number of milliseconds since the code started running

        If difference between currentMillis and previousMillis is greater than the Interval{
                Set previousMillis to currentMillis
                Set Difference = difference between newPosition and speedStart
                Set speedStart = newPosition
```

```
            }
        }

        IF client is connected to WAN{
            Read character "C" received from client
            If C = "W" then set bool drive to true
            IF C = "S" then set bool drive to false
        }

        IF drive is true{
            IF the IMU is available{
                Read the IMU accelerometer information
            }
            //This is the angle of the buggy
            Define variable "Angle" to polar coordinates between Z & Y acceleration
            Define speed as MotorPower function with angle as input
            If the speed is greater than 255, set it back equal to 255
            IF the speed is a number{
                Use SetMotorDirectionFunction
                Set motor speeds to variable "speed"
                Use print() function
            }
            Read encoder newPosition
            If position has changed, set oldPosition to newPosition
        } else if drive is false{
            Stop the buggy
        }
}

//Function to change the motor direction
Void setMotorDirection( currentAngle, targetAngle ){
        if(currentAngle < targetAngle){
            Set motor direction in direction that robot needs to go in order to rebalance
        }
        if(currentAngle > targetAngle){
            Set motor direction opposite to above
        }
}


//RETURNS SPEED IN PWM VALUE, USES PID
Double MotorPower(currentAngle){
        Calculates the appropriate motor speed in order to rebalance the buggy
}

Void Print(){
        Prints the current speed to processing
        Also was used for testing purposes
}
```

PROCESSING PSEUDO CODE (GOLD)

```
Import controlP5 library
Import processing.net library //used for wifi

Initialise variables

Void Setup(){

        Set canvas size
        Set background colour
        Set frame rate

        Connect to Arduino's WAN using its IP Address

        Set up Start & Stop buttons (Position, Size, Colour)
        Set up Speedometer/Knob (Position, Range, Size, Initial Value)
        Set up Text Display Areas 1,2 & 3 (Position, Size, Colour)
}

Void draw(){
        Set angle as angle value read from client
        Display angle value in a string.

        Set speed as speed value read from client
        Display speed in speedometer

}

Function ReactOnEvent(event){
        If event is START button being pushed
                Send "W" to Arduino

        If event is STOP button being pressed
                Send "S" to Arduino
}
```

Part E – Appendices

- Pc processing code - Just submit file
- Arduino Code - Just submit file

- A change log of the code

Date:	By Group Member:	Git Commit Descriptions:	Change Log:
30/01/2023	-> Put this in because it was in the Git commit thing do we just stick in random names?		What the lab sheets told us to do [Was learning how to use arduino here}
01/02/2023			[was just learning to use fritzing in this lab]
06/02/2023			[Bank Holiday]
08/02/2023			Wired encoders and Wheels - Set up code for basic bronze
13/02/2023			[Learning to use processing & Networking]
15/02/2023			Wired H-Bridge and Motors
20/02/2023			Made a WAP on arduino and got processing to control it -> Wired US sensor??
22/02/2023			[Learned how to graph stuff in arduino]
27/02/2023		-Uploaded Bronze Challenge Ardiuno & Processing Code	(Bronze Challenge Demo)-> This was the day we had the lecture on GIT

01/03/2023		- Added encoders to code	[Wired encoders]
06/03/2023		- Experimented with encoders and added PULLUP	
08/03/2023		-Added interrupts to Encoder Code -Added in PID Function	
13/03/2023		-Tuned PID	
15/03/2023		- Swapped our Encoder Function for Encoder Library	
20/03/2023		-Added code to send Ultrasonic Distance to Processing	(Silver Challenge)
22/03/2023		- Uploaded new files for Gold Challenge	
27/03/2023		-Improved code run speed	
29/03/2023		-Integrated Encoder Library into the Gold Code	
03/04/2023			MCQ Exam
05/04/2023		-Worked on tuning PID	
10/04/2023		-Changed Target Angle and Tuned PID	
12/04/2023		-Fine Tuned PID	Gold Demo