



Self-driving RC Car

Final Report



Company name: The Autonomous Avengers

Team:

Catalin Chiprian

Andi Serpe

Tiberiu Filip

Ivan Petrov

Place, date:	Enschede, 15 th February 2023		
Created by:	The Autonomous Avenger		
	Ivan Petrov		535237@student.saxion.nl
	Andi Şerpe	526581	526581@student.saxion.nl
	Tiberiu Ştefan Filip	531048	531046@student.saxion.nl
	Cătălin Chiprian	528729	528729@student.saxion.nl
Version number:	0.1		
Clients:	Ali Yuksel		
	Yanin Kasemsinsup		
	Lydia Prieto		
	Felix Hartlieb		
	Teun Hamer		
	Waseem Elsayed		
	Asif Khan		

Table of Contents

Table of Contents	iii
I. Abbreviations	v
1 Introduction	1
2 Analysis of requirements	2
3 Concept Principles	3
3.1 Description of alternative concepts.....	5
3.2 Comparison of concept principles	6
3.3 Choice of most promising concept principle	7
3.4 Elaboration of chosen principle to functional design	8
3.5 Overview of functional design	8
4 Elaboration of functional design	9
4.1.1 Mechanical.....	9
4.1.2 Electrical.....	11
4.1.3 Electronic	12
4.1.4 Software	13
4.2 Functional design integration	13
4.3 Working out one functional concept.....	13
4.4 Platform	14
4.5 Shields	14
4.6 Sensors	15
4.7 Power supply	15
4.8 Propulsion	15
4.9 Drawings	16
4.10 3D model view cover	18
5 Technical Design	20
5.1 Electrical.....	20
5.2 Electronic	20
5.3 Software.....	21
5.4 Test cases	19
5.5 Protocols	22
5.6 Standards	22
5.7 Unit testing	23

Table of Figures

FIGURE 1 - LINE FOLLOWING	10
FIGURE 2 - OBSTACLE DETECTION	10
FIGURE 3 - OBSTACLE AVOIDANCE	10
FIGURE 4 - LINE REDIRECTION.....	10
FIGURE 5 - END OF THE LINE DETECTION	11
FIGURE 6 - ELECTRICAL SCHEMATIC.....	11
FIGURE 7- ELECTRONIC SCHEMATIC	12
FIGURE 8 - SOFTWARE SCHEMATIC	13
FIGURE 9 - CAR BODY.....	14
FIGURE 10 - CAR DRAWING 1.....	16
FIGURE 11 - CAR DRAWING 2.....	16
FIGURE 12 - CAR DRAWING 3.....	17
FIGURE 13 - 3D COVER	18
FIGURE 14 - FINAL 3D COVER.....	19
FIGURE 15 - QTRC ARRAY MOUNT	19
FIGURE 16 - FRONT IR MOUNT	20
FIGURE 17 - ELECTRICAL SCHEMATIC.....	20
FIGURE 18 - ELECTRONIC SCHEMATIC	21
FIGURE 19 - QTRC UNIT TESTING	23
FIGURE 20 - QTRC UNIT TESTING 2	24
FIGURE 21 - MPU6050 UNIT TESTING.....	28
FIGURE 22- MPU6050 UNIT TESTING 2.....	28
FIGURE 23 - IR UNIT TESTING.....	33
 EQUATION 1 - QTR-8RC FORMULA	 36

I. Abbreviations

RC	Radio-controlled
SR	System Requirements

1 Introduction

Chapter with general introduction of the project/product, together with a description of the goal of this document and an overview of the document structure.

The purpose of this functional design document is to outline the steps taken by our team to come up with a system concept for an autonomous RC car that meets the user requirements. The document will consist of various sections, including an analysis of the requirements, generation of concept principles, comparison of these principles, and the selection of the most promising concept principle. Additionally, we will elaborate on the chosen principle by developing a concept design for each discipline, including mechanical, electrical, electronic, and software.

Remarks on goal and structure of this document:

Goal of the Functional Design (FD) is to document the steps that were taken to come to a system concept. This is usually a quite chaotic process involving several iterations of creativity and analysis. Common steps that can be distinguished are:

- *The main requirements of the System Requirements (SR) document are analysed.*
- *Generation of possible concept principles for the main function(s) of the system, in the form of sketches, equations, block diagrams and/or mass-spring systems.*
- *Analysis to motivate compliance to the requirements of a concept principle, preferably quantitatively.*
- *Sometimes the concept principle for a function needs to be tested using some existing hardware.*
- *Evaluation and selection of concept principle, using e.g. a morphologic overview.*
- *Generation of one or several concept designs, in the form of sketches, block diagrams of servo control, electr(on)ical hardware and/or software.*
- *A global dimensioning of the most promising concepts is done.*
- *An analysis on essential aspects is performed, to motivate quantitative compliance to the main requirements.*
- *A choice of the most promising concept is made.*

The CD-document should not only document the outcome of this process, but certainly also the alternative concepts, arguments pro/con and analyses. The paragraph structure proposed below aims at doing so in a logical way, linked to the steps listed above. But if a different structure leads to a better document then the author is free to use his own.

2 Analysis of requirements

The analysis of requirements is a critical step in the process of developing a functional design for the autonomous RC car. The purpose of this step is to evaluate the feasibility of the functional and technical requirements as stated in the System Requirements (SR) document. This is done by conducting an in-depth analysis of each requirement to identify potential issues that may arise during the design and development process.

The analysis of the car's motion is a crucial aspect of the requirements analysis. This involves examining the velocities, accelerations, motion profiles, and cycle time of the car. The goal is to ensure that the car can move smoothly and accurately along the desired path while meeting the required performance criteria.

Another aspect of the requirements analysis is the accuracy of the car concerning disturbance forces and decomposition. This involves examining the car's ability to maintain its position and stability in the presence of external forces and disturbances. This is important for ensuring that the car can navigate safely and reliably through different environments and scenarios.

The mass and vibrations of the car are also important factors to consider during the requirements analysis. The goal is to ensure that the car is lightweight and stable while still meeting the necessary performance requirements. This is critical for achieving the desired speed and agility while maintaining the car's overall safety and stability.

Lastly, the power consumption of the car is an essential consideration during the requirements analysis. This involves examining the car's power requirements in terms of decomposition and internal losses. The goal is to ensure that the car has sufficient power to operate reliably while minimizing the overall power consumption and reducing the environmental impact of the car.

Overall, the analysis of requirements is a critical step in the functional design process for the autonomous RC car. By conducting a thorough analysis of each requirement, the team can identify potential issues and challenges early on, which can help to ensure that the final design meets all necessary functional and technical requirements.

Examples:

- *Motion: velocities, accelerations, motion profiles, cycle time*
- *Accuracy: disturbance forces, decomposition*
- *Mass: decomposition*
- *Vibrations: sensitivity, transfer functions, response spectra*
- *Power consumption: decomposition, internal losses*



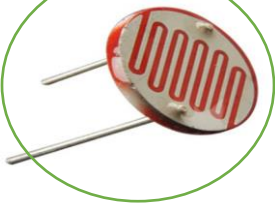
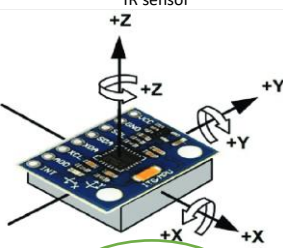





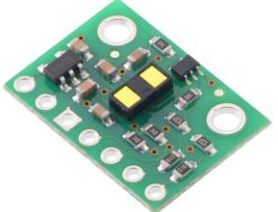
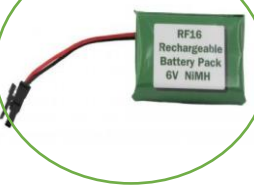





This analysis may require defining an initial concept principle, to which others will be compared (see next chapter).

3 Concept Principles

Several alternative concept principles will be generated for the main function(s) of the autonomous RC car. These principles will be evaluated against functional and technical requirements, including motion, accuracy, safety, mass, dimensions, and power consumption. A description of the alternative concepts will be provided, mainly qualitatively. We will then compare the concept principles, and where possible, this will be done in a quantitative manner. If required detailed calculations will be put in the appendices.

Concept 1:

Table 1 - Concept 1

	IDEA 1	IDEA 2	IDEA 3	IDEA 4
LINE DETECTION				
INCLINE DETECTION				
OBSTACLE DETECTION				
POWER SUPPLY				
MICRO CONTROLLER				
	Arduino UNO	Arduino NANO	Raspberry Pi	

Concept 2:

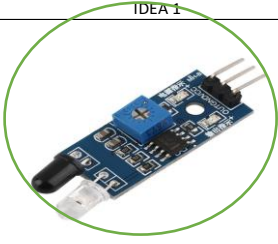


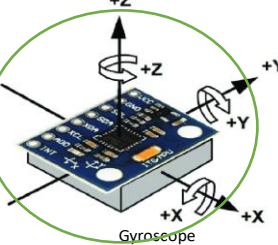










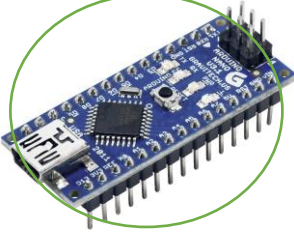

	IDEA 1	IDEA 2	IDEA 3	IDEA 4
LINE DETECTION				
IR sensor		Camera	Light dependant resistor	
INCLINE DETECTION				
Gyroscope		Accelerometer	Magnetometer	
OBSTACLE DETECTION				
Ultrasonic sensor		IR sensor	Camera	LIDAR
POWER SUPPLY				
Rechargeable battery pack		Replaceable battery pack	Solar panels	
MICRO CONTROLLER				
Arduino UNO		Arduino NANO	Raspberry Pi	

Table 2 - Concept 2

Concept 3:



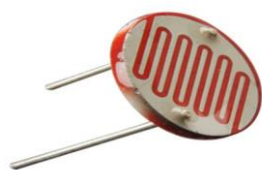
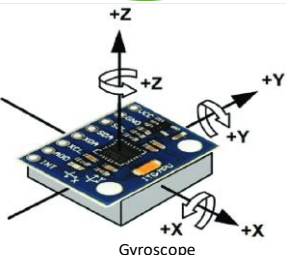
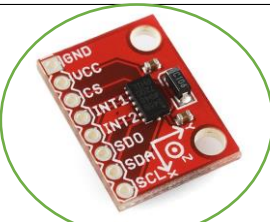









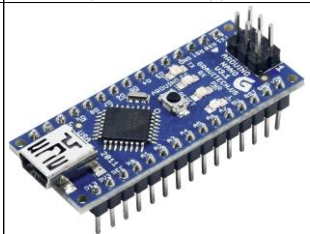

	IDEA 1	IDEA 2	IDEA 3	IDEA 4
LINE DETECTION	 IR sensor	 Camera	 Light dependant resistor	
INCLINE DETECTION	 Gyroscope	 Accelerometer	 Magnetometer	
OBSTACLE DETECTION	 Ultrasonic sensor	 IR sensor	 Camera	 LIDAR
POWER SUPPLY	 Rechargeable battery pack	 Replaceable battery pack	 Solar panels	
MICRO CONTROLLER	 Arduino UNO	 Arduino NANO	 Raspberry Pi	

Table 3 - Concept 3

3.1 Description of alternative concepts

This paragraph describes the result of a creative process: the concept principles for the main function(s) of the system. This description is mainly qualitatively.

Concept 1: This concept uses a light-dependent resistor for line detection, which measures the light intensity on the surface to detect the line. An accelerometer is used for incline detection, which measures the changes in acceleration due to the inclination of the surface. An ultrasonic sensor is used for obstacle detection, which sends out high-frequency sound waves and measures the time it takes for

them to bounce back to detect obstacles. The microcontroller chosen for this concept is a Raspberry Pi, which is a powerful and versatile single-board computer. The power supply for the car is a rechargeable battery pack.

Concept 2: This concept uses an IR sensor for line detection, which detects the infrared radiation reflected by the line. A gyroscope is used for incline detection, which measures the angular velocity and angle of inclination of the car. An IR sensor is used for obstacle detection, which detects the infrared radiation emitted by obstacles. The microcontroller chosen for this concept is an Arduino Nano, which is a compact and cost-effective board. The power supply for the car is a replaceable battery pack.

Concept 3: This concept uses an IR sensor for line detection, which detects the infrared radiation reflected by the line. An accelerometer is used for incline detection, which measures the changes in acceleration due to the inclination of the surface. An IR sensor is used for obstacle detection, which detects the infrared radiation emitted by obstacles. The microcontroller chosen for this concept is an Arduino Uno, which is a powerful and widely-used board. The power supply for the car is a rechargeable battery pack.

3.2 Comparison of concept principles

This paragraph gives a motivated comparison between the alternative concepts, against:

1. Line Detection: All three concepts use sensors that are effective at detecting a white line on a black background. However, Concept 1 uses light-dependent resistors while Concept 2 and 3 use IR sensors. While both are effective, IR sensors have a longer range than LDRs.
2. Obstacle Detection: Concept 1 uses an ultrasonic sensor while Concepts 2 and 3 use IR sensors. Ultrasonic sensors can detect obstacles at a longer range, but they are less accurate than IR sensors. IR sensors are more accurate, but their range is limited.
3. Incline Detection: Both Concept 1 and Concept 3 use an accelerometer while Concept 2 uses a gyroscope. Gyroscopes are more accurate than accelerometers but also more sensitive to vibrations.
4. Microcontroller: Concept 1 uses a Raspberry Pi while Concepts 2 and 3 use different versions of the Arduino board. Raspberry Pi is more powerful than Arduino, making it more suitable for complex tasks such as machine learning. However, Arduino is more efficient and consumes less power than Raspberry Pi.
5. Power Supply: Both Concept 1 and 3 use a rechargeable battery pack while Concepts 2 utilizes a replaceable one. Rechargeable batteries are more environmentally friendly and cost-effective in the long run, but they add extra weight to the car. Replaceable batteries are lighter but need to be replaced more often.

In summary, Concept 1 is the most powerful and accurate, making it suitable for complex tasks. Concept 2 is a good option if you want a lightweight car, while Concept 3 is a good balance between power and simplicity.



3.3 Choice of most promising concept principle

Concept 3 seems to be the best option for the project. The IR sensor for line detection has a high sensitivity and can detect the difference between the line and the background, making it an excellent choice for the project. The accelerometer is a good choice for incline detection as it can measure the car's tilt angle accurately. It is also a cost-effective option compared to a gyroscope. The IR sensor for obstacle detection is also a reliable and cost-effective option that can detect obstacles with precision. Using a rechargeable battery pack ensures that the car has a stable and long-lasting power supply, while the Arduino Uno is a popular and reliable microcontroller that can handle the project's requirements. Overall, Concept 3 provides a cost-effective and efficient solution to meet the project's requirements.

3.4 Elaboration of chosen principle to functional design

After comparing the alternative concepts, we will choose the most promising concept principle and develop a concept design for each discipline. A technical block diagram will be created that includes all mechanical, electrical, and electronic blocks, as well as a safety philosophy that outlines standards and measures. The mechanical discipline will focus on creating a layout of the major components and performing dynamical and thermal calculations if necessary. Dimensions of the drives, transmissions, and flexibility will also be measured. For the electrical discipline, electrical layout will be created that includes modules and interconnections on a wiring level, and for the electronic discipline an electronic layout will be designed that includes modules and interconnections. We will choose a processing platform, such as a PC or microcontroller, and develop algorithms for the main and sub-functions of the software. We will create an architecture diagram with modules and interfaces, outline the system states and state transition diagram, and consider interrupts, programming language, and environment.

3.5 Overview of functional design

Line Detection: For line detection, an IR sensor will be used which detects the infrared radiation reflected by the white line on the black surface. The IR sensor will be placed underneath the car, and it will continuously scan the surface to detect the line. The IR sensor will be connected to the Arduino Uno microcontroller, which will process the data received from the sensor and use it to control the movement of the car.

Incline Detection: To detect the inclination of the surface, an accelerometer will be used. The accelerometer will be placed on the car chassis, and it will continuously measure the changes in acceleration due to the inclination of the surface. The data received from the accelerometer will be processed by the Arduino Uno microcontroller to adjust the car's speed and direction according to the inclination of the surface.

Obstacle Detection: For obstacle detection, two IR sensors will be used which detects the infrared radiation emitted by the obstacles. The IR sensors will be placed on top of each other on the front of the car, and it will continuously scan the surroundings to detect obstacles. The data received from the IR sensors will be processed by the Arduino Uno microcontroller to navigate the car around the obstacles.

Control System: The Arduino Uno microcontroller will serve as the control system for the car. It will receive data from the sensors, process it, and use it to control the motors of the car. The control system will include algorithms to navigate the car along the white line, avoid obstacles, and adjust the car's speed and direction according to the inclination of the surface.

Power Supply: The power supply for the car will be a rechargeable battery pack. The battery pack will be connected to the Arduino Uno microcontroller, and it will provide power to the motors and sensors of the car.

Overall, Concept 3 utilizes IR sensors for line and obstacle detection, an accelerometer for incline detection, and an Arduino Uno microcontroller as the control system, all powered by a rechargeable battery pack. This design will enable the RC car to smoothly follow a white line on a black background, detect obstacles, and navigate around them.



4 Elaboration of functional design

4.1.1 Mechanical

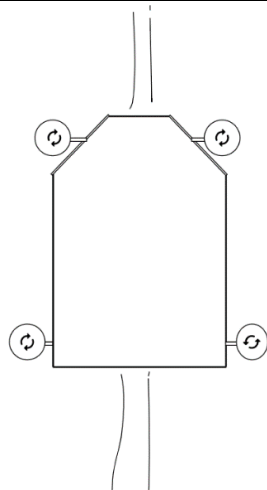


Figure 1 - Line following

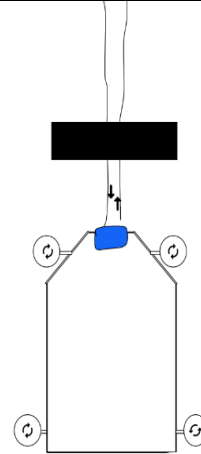


Figure 2 - Obstacle detection

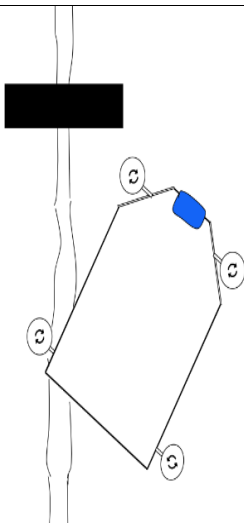


Figure 3 - Obstacle avoidance

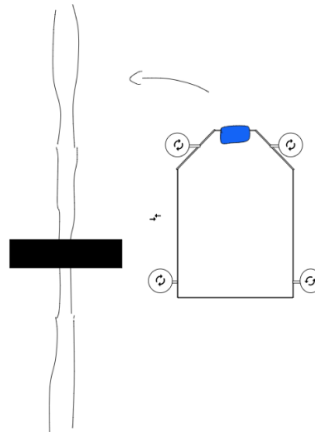
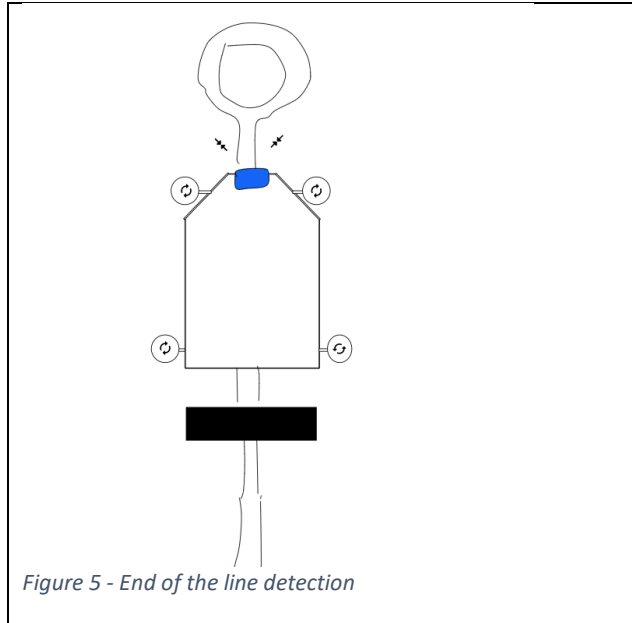


Figure 4 - Line redirection



These mechanical drawings of this project show a visual representation of how the car will react from the feedback of its sensors. The presented mechanical concept meets the desired results and satisfies all the client's requirements.

4.1.2 Electrical

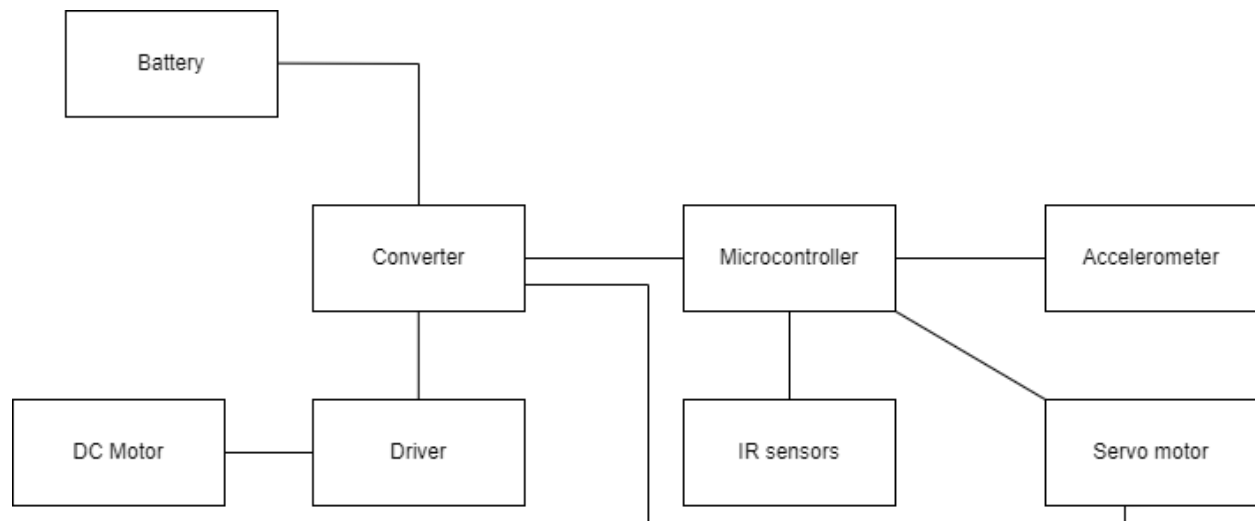


Figure 6 - Electrical schematic

4.1.3 Electronic

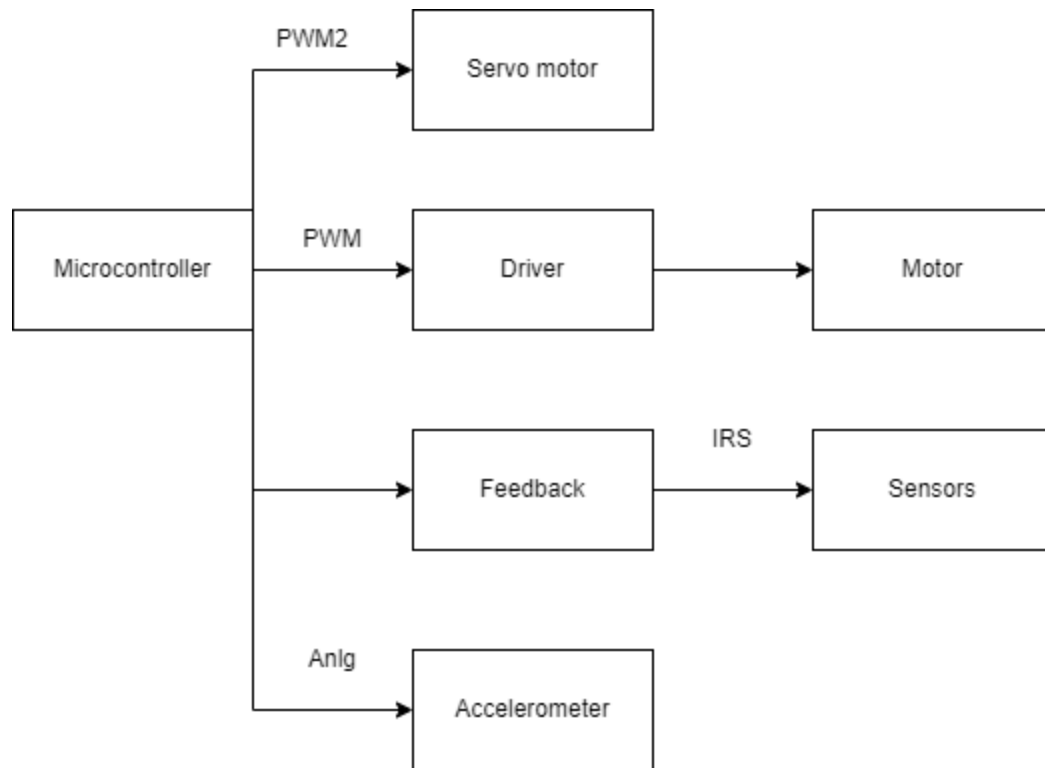


Figure 7- Electronic schematic

4.1.4 Software

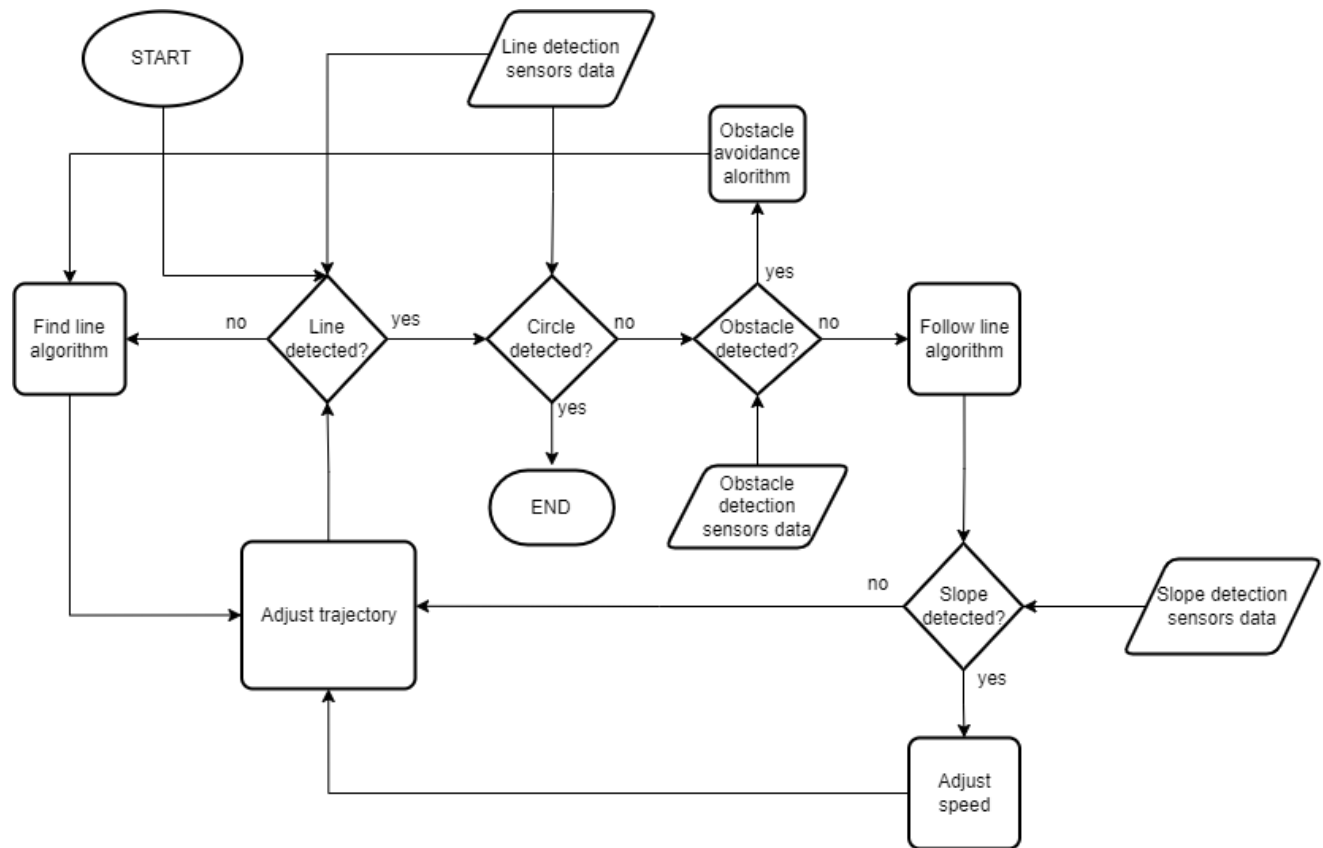


Figure 8 - Software schematic

4.2 Functional design integration

Lastly, group members will consider properties that affect more than one discipline, such as safety, speed, accuracy, cost, and maintenance. The company will ensure that the chosen concept principle meets these requirements and adjust as necessary.

4.3 Working out one functional concept

To create a functioning robotic platform, careful consideration was given to the selection of components. IR sensors were chosen for line following due to their ability to detect changes in reflectivity. An accelerometer sensor was selected to measure the platform's tilt accurately while consuming minimal power. For obstacle avoidance, IR sensors were preferred for their ability to detect nearby objects. A rechargeable battery was chosen as the power supply to reduce waste and promote eco-friendliness. The Arduino Uno was selected as the main controller due to its availability, extensive community support, and programming resources. By evaluating the strengths and weaknesses of each component, a functional concept was developed that aligns with the project's objectives.

4.4 Platform

The platform of the RC car represents the structure of the car and all its components. Considering the following specifications:

- **Materials used:** aluminium and plastic;
- **Dimensions (L/W/H):** 380x255x155 mm;
- **Weight:** 1410g;
- **Drive System:** 4 Wheel Drive, where 2 wheels` direction are controlled by a servo;
- **Tire width:** 33 mm front / 42 mm rear;
- **Tire circumference:** 88 mm.

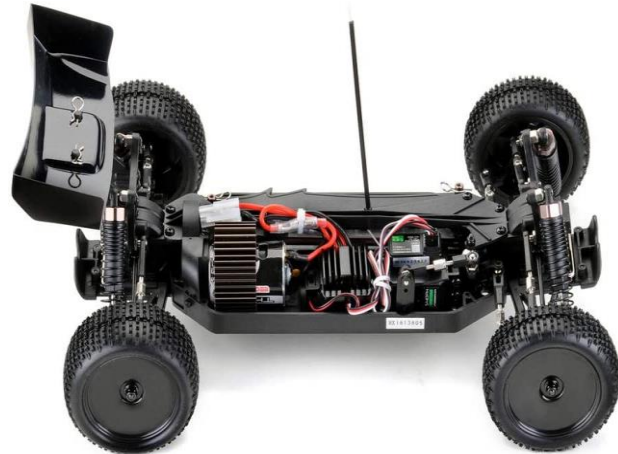


Figure 9 - Car body

Considering the mentioned specifications, the chassis in discussion will hold all the pieces. The company will make sure that the chosen components will reach the project objectives in terms of placing and effectiveness.

4.5 Shields

The motor shield specified for the autonomous car project is a brushed motor controller for RC cars. It has a compact size of 46.5 x 34 x 28.5 mm and weighs 70g, making it suitable for use in crawlers and boats. The motor shield is designed to work with Tamiya plug system and can support a maximum load of 180 A with a continuous current of 40 A. It is compatible with NiCd/NiMH cells of 5-9 and Li-poly cells of 2-3.

The brushed motor controller has a single-click forward/backward function with a quick response motor brake, low power auto cut-off, and fail-safe function. The motor shield has an overheat protection feature that activates at 100°C, ensuring the motor's longevity. The resistance for forward and backward movements is 0.002 Ω and 0.004 Ω , respectively, while the power transfer during backward movement is 100% in crawler mode and 25% in boat mode.

The motor shield has a plug-and-drive system, making it easy to install and use, and comes with a user manual. This specification makes it an ideal motor controller for the autonomous car project.

4.6 Sensors

When deciding on sensors, the company had to look back at the functional requirements and think of all the possible situations our product could find itself in. For line detection, 3 IR sensors are used due to their high sensitivity and ability to differentiate between contrasting colours. Incline detection is achieved using an accelerometer, which can accurately measure the car's tilt angle. For obstacle detection 2 long-range IR sensors were chosen, which are to be mounted at different elevations on the car to give us the ability to differentiate between an obstacle and a ramp.

4.7 Power supply

Choosing the right battery for the autonomous RC car project is crucial to ensure its efficient and stable operation. The 7.4V lithium polymer battery is an excellent choice for this project due to its high energy density, lightweight, and relatively small size. These batteries can supply a high amount of power for their size, making them suitable for powering high-performance vehicles like RC cars. Additionally, LiPo batteries have a low self-discharge rate, which means they can hold their charge for longer periods, making them ideal for long-duration projects. Furthermore, LiPo batteries have a high discharge rate, which allows them to deliver power quickly to the motors when needed. Overall, the 7.4V LiPo battery is a reliable and efficient choice for powering the autonomous RC car project.

4.8 Propulsion

When considering the propulsion system for our project, the group considered the need for both speed and manoeuvrability. After careful consideration, the members decided to use a combination of a DC motor and a servo motor for the wheels. The DC motor will provide the primary driving force for the vehicle, while the servo motor will control the rotation of the wheels, allowing for precise turning and navigation. With this setup, the team can achieve both speed and agility, making our device well-suited for a variety of terrain and obstacles. The motors are powered by a battery pack, which is also connected to the Arduino Uno through a voltage regulator to provide stable and reliable power. Additionally, members will be able to program the motors using Arduino Uno to achieve the desired functionality and performance.

4.9 Drawings

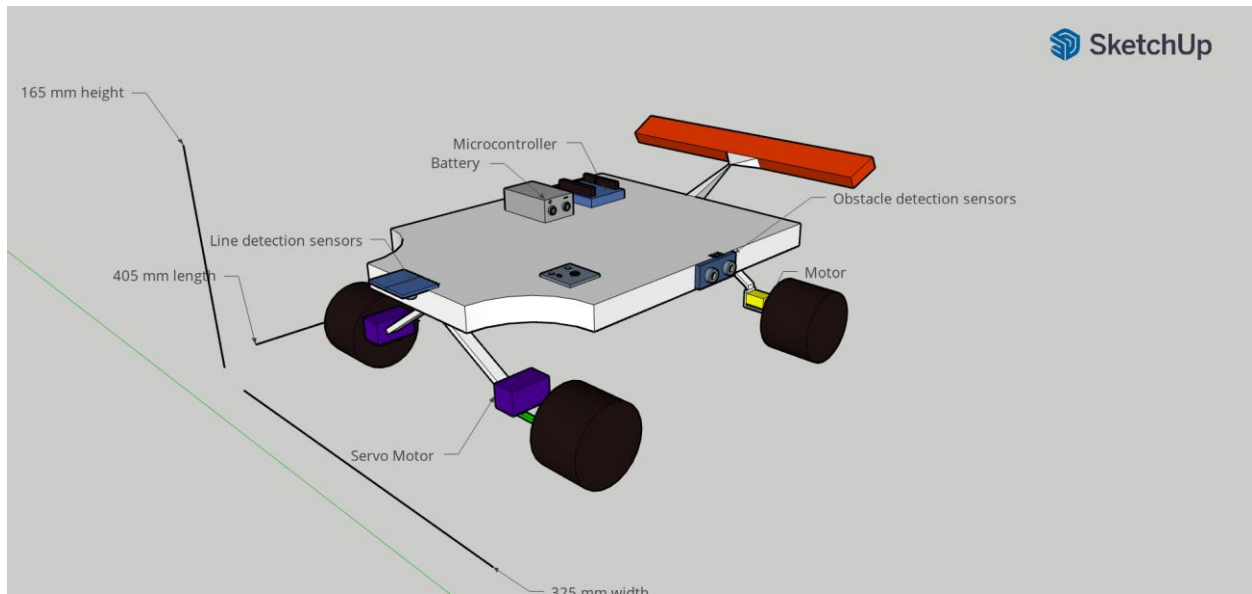


Figure 10 - Car drawing 1

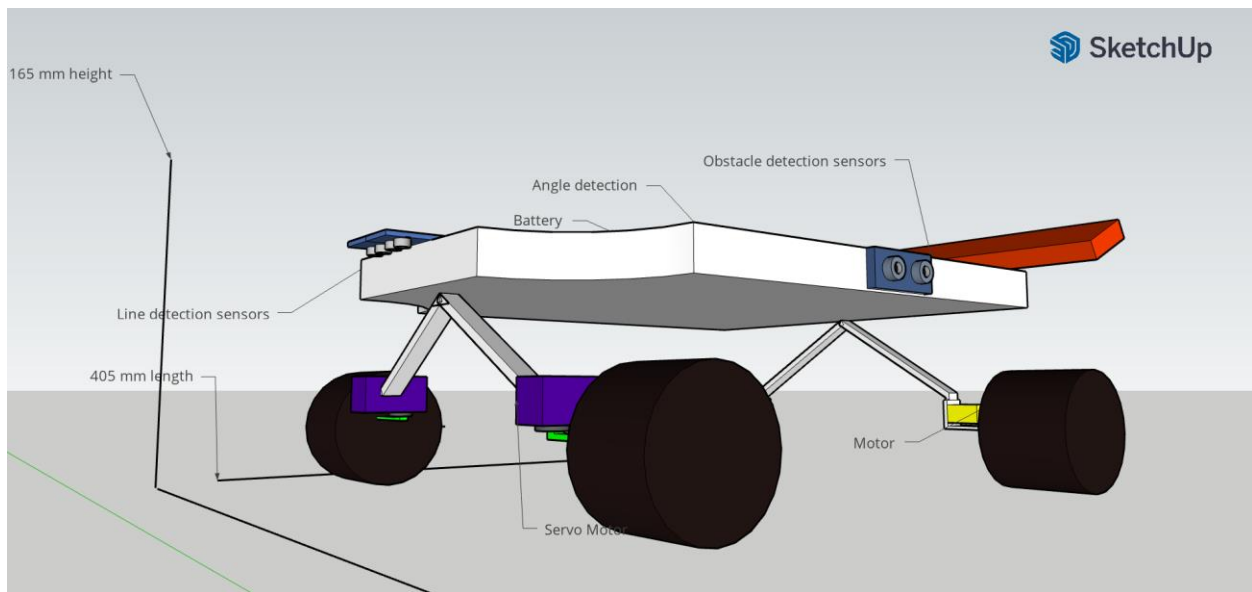


Figure 11 - Car drawing 2

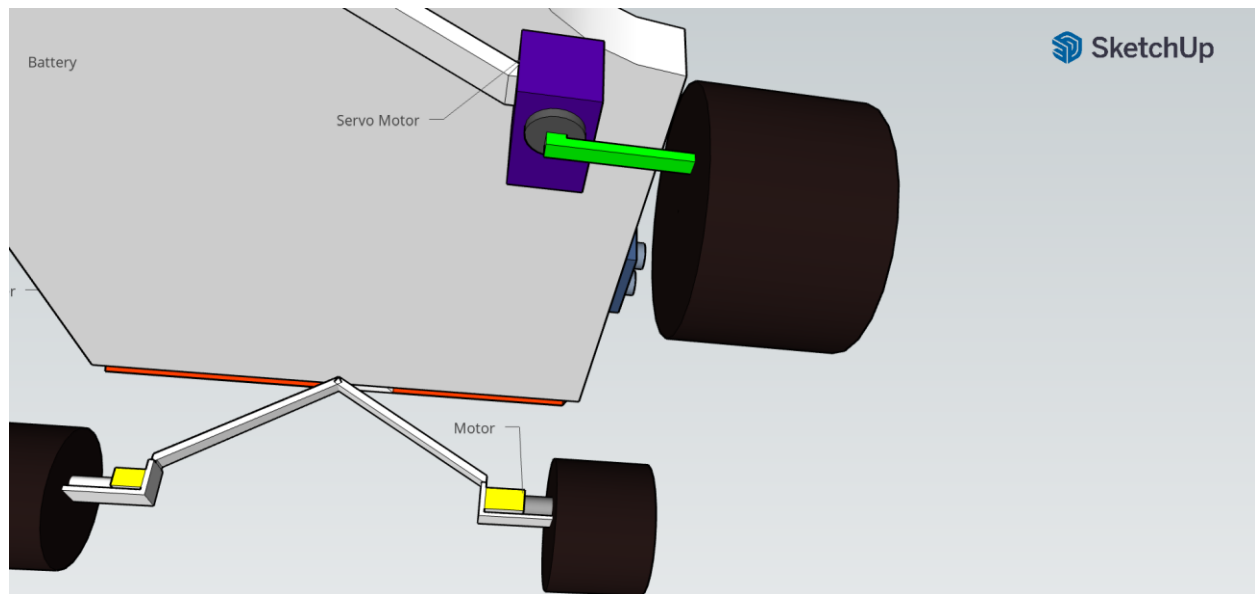
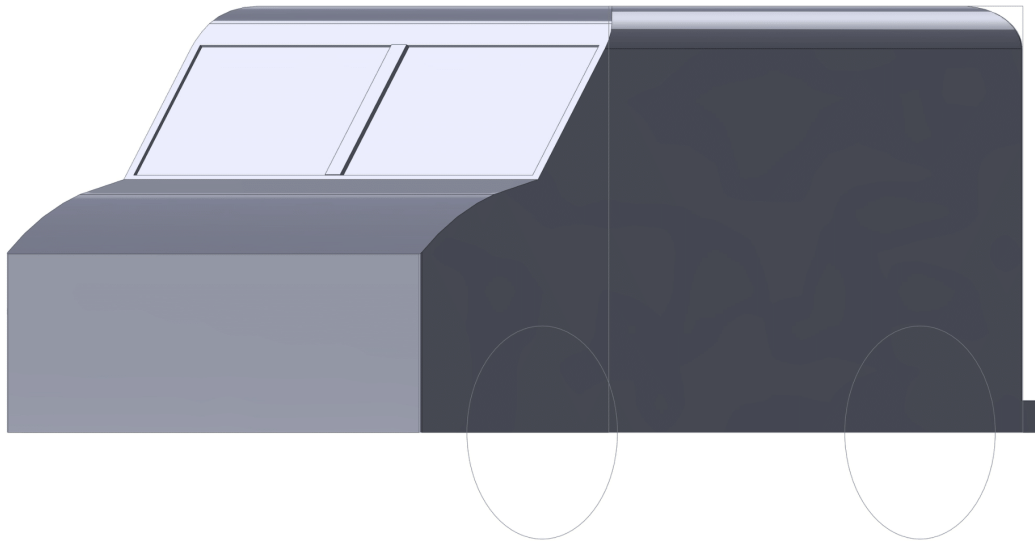


Figure 12 - Car drawing 3

4.10 3D model view cover



SOLIDWORKS Educational Product. For Instructional Use Only.

Figure 13 - 3D cover

4.10.1 Final car cover design

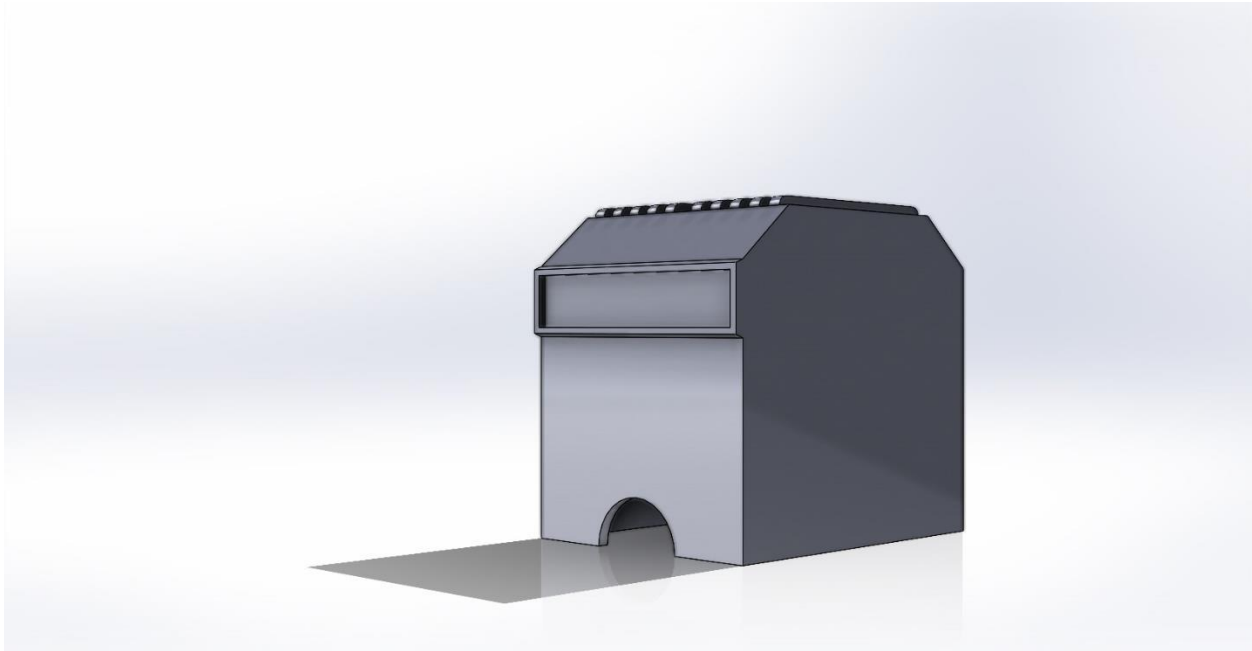


Figure 14 - Final 3D cover

4.10.2 QTRC array mount

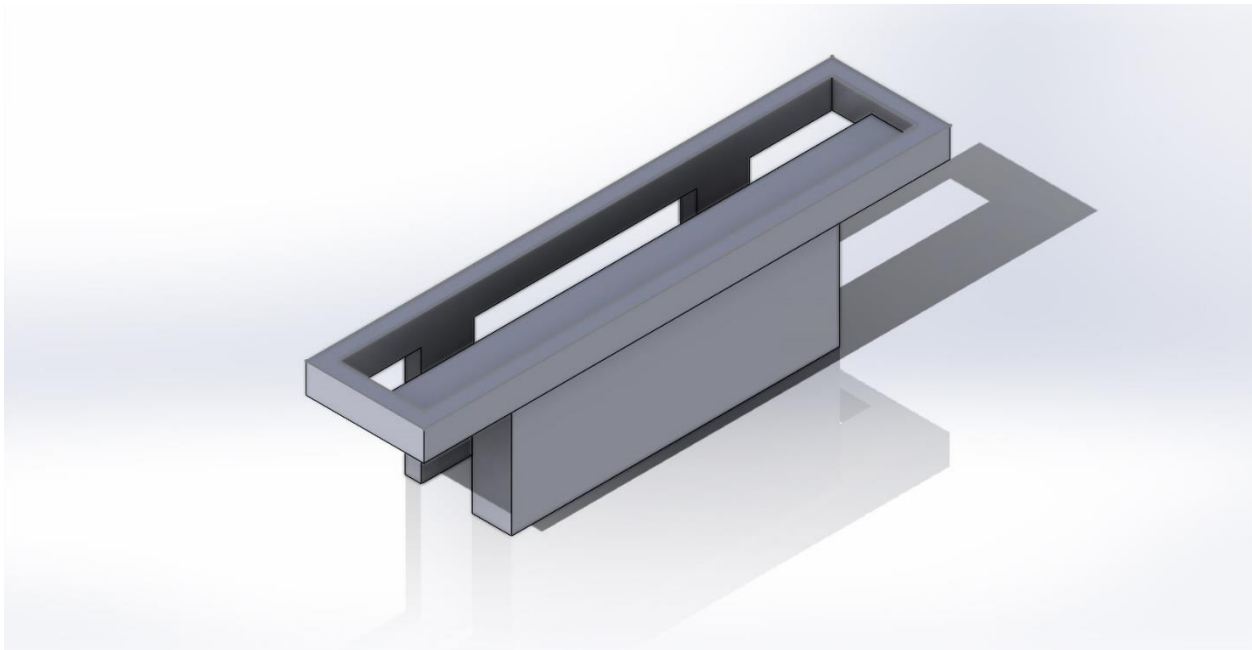


Figure 15 - QTRC array mount

4.10.3 Front IR's mount

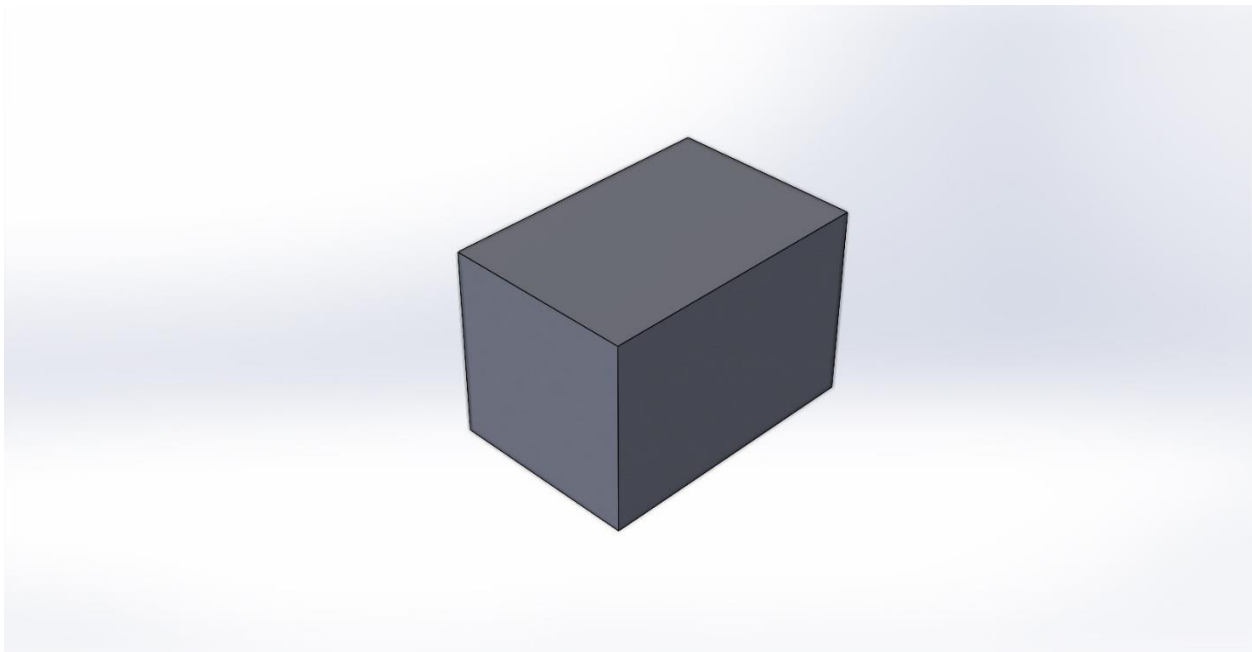


Figure 16 - Front IR mount

5 Technical Design

5.1 Electrical

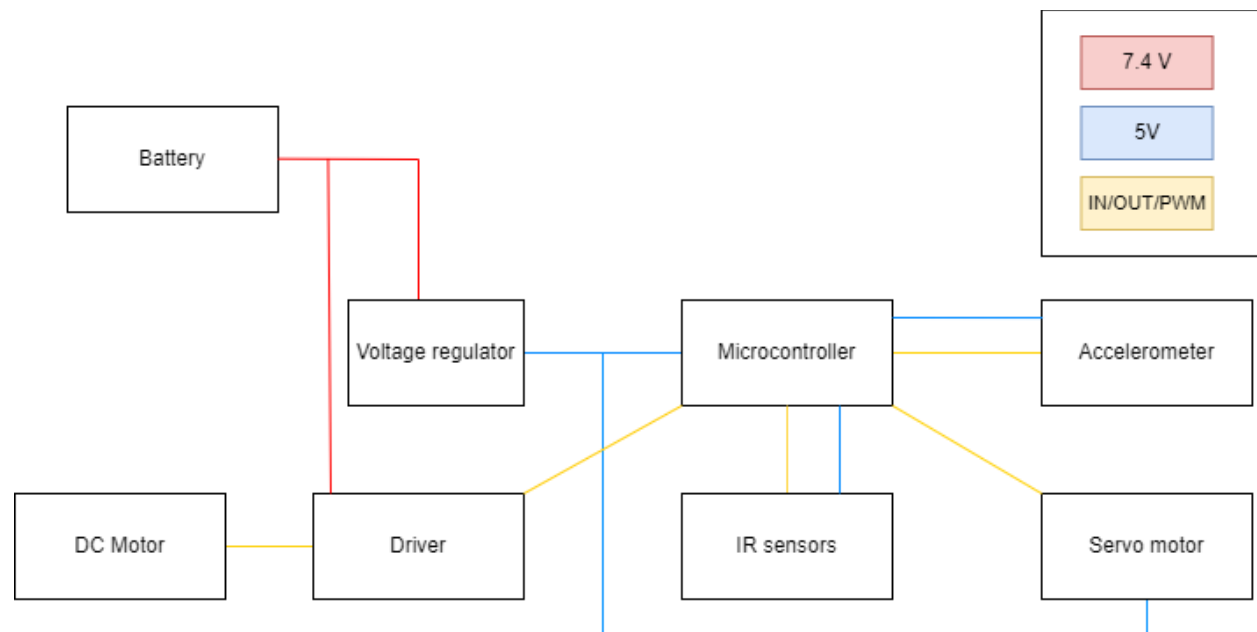


Figure 17 - Electrical schematic

5.2 Electronic

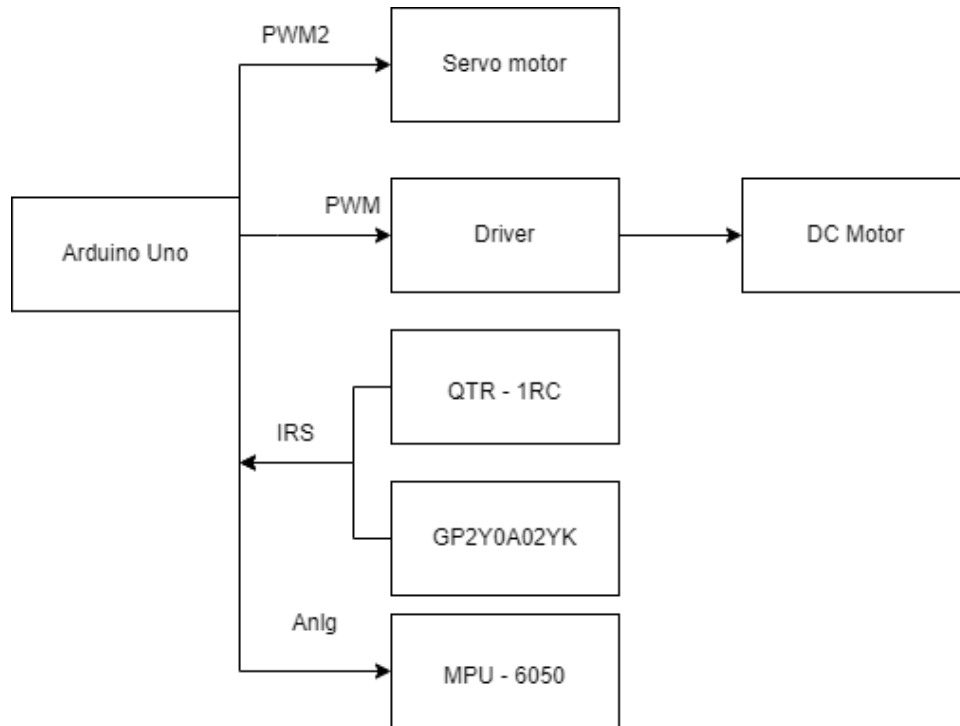


Figure 18 - Electronic schematic

5.3 Software

BEGIN

CALL getSensorsData(lineDetected, obstacleDetected, finishDetected, inclineAngle)

IF lineDetected == true THEN

IF finishDetected == true THEN

CALL setSpeed(0)

END

ENDIF

IF obstacleDetected == true THEN

direction = "forward"

CALL setSpeed(speed)

CALL turn(angle, time)

CALL findLine()

ENDIF

IF inclineAngle != 0 THEN

CALL setSpeed(speed) with speed based on inclineAngle

```
ENDIF  
CALL follow()  
ELSE  
CALL findLine()  
ENDIF  
END
```

5.4 Protocols

1. White line detection protocol: The robot will move along a white line, and the IR sensors will detect the line's position. The protocol should define the speed and direction of the robot.
2. Obstacle detection protocol: The robot will move towards an obstacle, and the IR sensors will detect the obstacle's position and distance. The protocol should define the expected readings from the sensors, such as the distance between the sensors and the obstacle.
3. Tilt measurement protocol: The robot will be placed in different orientations and movements, and the accelerometer sensor will detect the tilt angle. The protocol should define the range of angles, the duration of each movement, and the expected readings from the sensor.
4. Power supply protocol: The robot will run continuously for a specified duration, and the battery's voltage and current will be measured at regular intervals. The protocol should define the duration of the test, the load on the battery, and the expected voltage and current readings.
5. Controller performance protocol: The robot will execute a series of movements based on the input from the sensors, and the controller's response time and accuracy will be measured. The protocol should define the movements, the expected sensor readings, and the expected controller response.

5.5 Standards

1. White line detection standard: The IR sensors should be able to detect a white line on a standard white paper with a minimum width of 5mm at a speed of 0.5m/s with an accuracy of +/- 1mm.

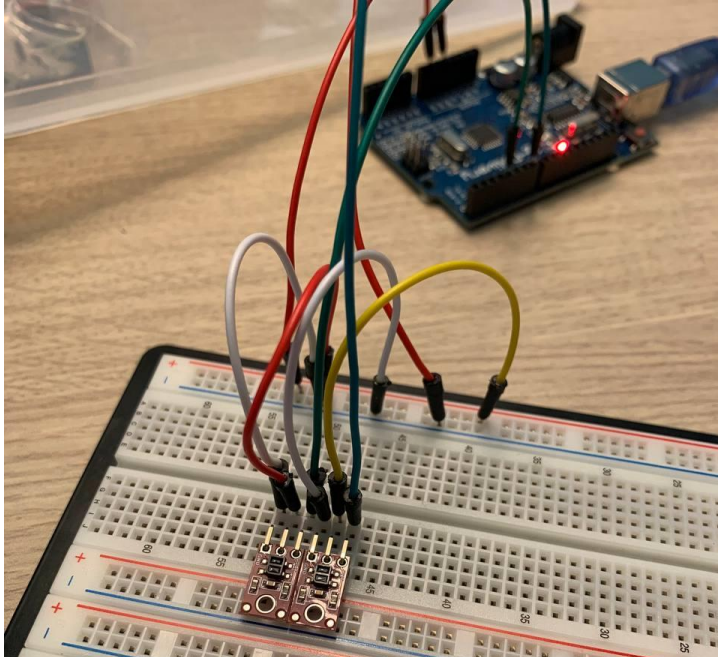


Figure 20 - QTRC unit testing 2

Code:

```
``C++
```

```
#include <QTRSensors.h>
```

```
// This example is designed for use with eight RC QTR sensors. These
// reflectance sensors should be connected to digital pins 3 to 10. The
// sensors' emitter control pin (CTRL or LEDON) can optionally be connected to
// digital pin 2, or you can leave it disconnected and remove the call to
// setEmitterPin().
//
// The setup phase of this example calibrates the sensors for ten seconds and
// turns on the Arduino's LED (usually on pin 13) while calibration is going
// on. During this phase, you should expose each reflectance sensor to the
// lightest and darkest readings they will encounter. For example, if you are
// making a line follower, you should slide the sensors across the line during
```

```
// the calibration phase so that each sensor can get a reading of how dark the
// line is and how light the ground is. Improper calibration will result in
// poor readings.
//
// The main loop of the example reads the calibrated sensor values and uses
// them to estimate the position of a line. You can test this by taping a piece
// of 3/4" black electrical tape to a piece of white paper and sliding the
// sensor across it. It prints the sensor values to the serial monitor as
// numbers from 0 (maximum reflectance) to 1000 (minimum reflectance) followed
// by the estimated location of the line as a number from 0 to 5000. 1000 means
// the line is directly under sensor 1, 2000 means directly under sensor 2,
// etc. 0 means the line is directly under sensor 0 or was last seen by sensor
// 0 before being lost. 5000 means the line is directly under sensor 5 or was
// last seen by sensor 5 before being lost.
```

```
QTRSensors qtr;
```

```
const uint8_t SensorCount = 8;
uint16_t sensorValues[SensorCount];
```

```
void setup()
{
    // configure the sensors
    qtr.setTypeRC();
    qtr.setSensorPins((const uint8_t[]){3, 4, 5, 6, 7, 8, 9, 10}, SensorCount);
    qtr.setEmitterPin(2);
```

```
    delay(500);

    pinMode(LED_BUILTIN, OUTPUT);

    digitalWrite(LED_BUILTIN, HIGH); // turn on Arduino's LED to indicate we are in
    calibration mode

    // 2.5 ms RC read timeout (default) * 10 reads per calibrate() call
    // = ~25 ms per calibrate() call.
    // Call calibrate() 400 times to make calibration take about 10 seconds.
    for (uint16_t i = 0; i < 400; i++)
    {
        qtr.calibrate();
    }

    digitalWrite(LED_BUILTIN, LOW); // turn off Arduino's LED to indicate we are
    through with calibration

    // print the calibration minimum values measured when emitters were on
    Serial.begin(9600);
    for (uint8_t i = 0; i < SensorCount; i++)
    {
        Serial.print(qtr.calibrationOn.minimum[i]);
        Serial.print(' ');
    }
    Serial.println();

    // print the calibration maximum values measured when emitters were on
    for (uint8_t i = 0; i < SensorCount; i++)
```

```
{  
    Serial.print(qtr.calibrationOn.maximum[i]);  
    Serial.print(' ');  
}  
Serial.println();  
Serial.println();  
delay(1000);  
}  
  
void loop()  
{  
    // read calibrated sensor values and obtain a measure of the line position  
    // from 0 to 5000 (for a white line, use readLineWhite() instead)  
    uint16_t position = qtr.readLineBlack(sensorValues);  
  
    // print the sensor values as numbers from 0 to 1000, where 0 means maximum  
    // reflectance and 1000 means minimum reflectance, followed by the line  
    // position  
    for (uint8_t i = 0; i < SensorCount; i++)  
    {  
        Serial.print(sensorValues[i]);  
        Serial.print('\t');  
    }  
    Serial.println(position);  
  
    delay(250);  
}
```

MPU-6050

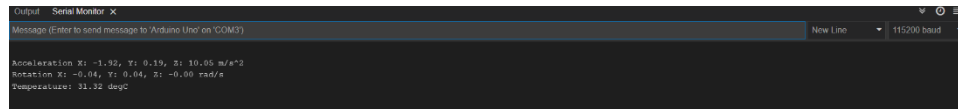


Figure 21 - MPU6050 unit testing

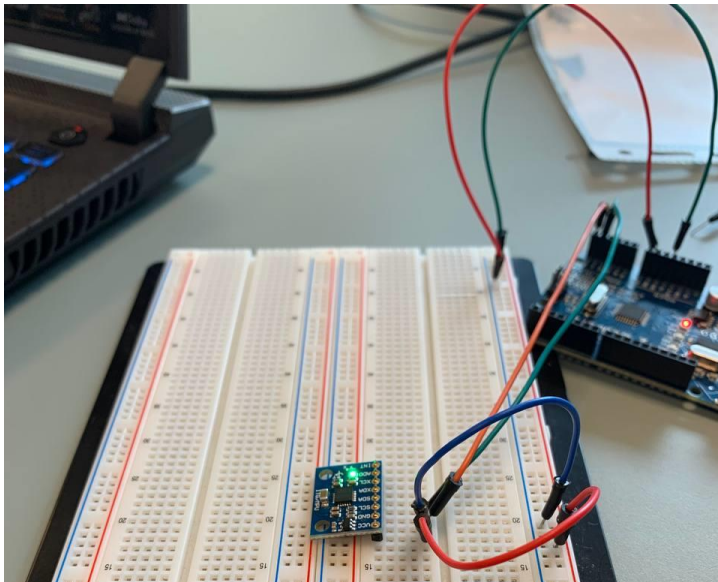


Figure 22- MPU6050 unit testing 2

Code:

```C++

```
// Basic demo for accelerometer readings from Adafruit MPU6050
```

```
#include <Adafruit_MPU6050.h>
```

```
#include <Adafruit_Sensor.h>
```

```
#include <Wire.h>
```

```
Adafruit_MPU6050 mpu;

void setup(void) {
 Serial.begin(115200);
 while (!Serial)
 delay(10); // will pause Zero, Leonardo, etc until serial console opens

 Serial.println("Adafruit MPU6050 test!");

 // Try to initialize!
 if (!mpu.begin()) {
 Serial.println("Failed to find MPU6050 chip");
 while (1) {
 delay(10);
 }
 }
 Serial.println("MPU6050 Found!");

 mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
 Serial.print("Accelerometer range set to: ");
 switch (mpu.getAccelerometerRange()) {
 case MPU6050_RANGE_2_G:
 Serial.println("+/-2G");
 break;
 case MPU6050_RANGE_4_G:
```

```
 Serial.println("+4G");
 break;
 case MPU6050_RANGE_8_G:
 Serial.println("+8G");
 break;
 case MPU6050_RANGE_16_G:
 Serial.println("+16G");
 break;
 }
 mpu.setGyroRange(MPU6050_RANGE_500_DEG);
 Serial.print("Gyro range set to: ");
 switch (mpu.getGyroRange()) {
 case MPU6050_RANGE_250_DEG:
 Serial.println("+ 250 deg/s");
 break;
 case MPU6050_RANGE_500_DEG:
 Serial.println("+ 500 deg/s");
 break;
 case MPU6050_RANGE_1000_DEG:
 Serial.println("+ 1000 deg/s");
 break;
 case MPU6050_RANGE_2000_DEG:
 Serial.println("+ 2000 deg/s");
 break;
 }

 mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
 Serial.print("Filter bandwidth set to: ");
```



```
switch (mpu.getFilterBandwidth()) {
 case MPU6050_BAND_260_HZ:
 Serial.println("260 Hz");
 break;
 case MPU6050_BAND_184_HZ:
 Serial.println("184 Hz");
 break;
 case MPU6050_BAND_94_HZ:
 Serial.println("94 Hz");
 break;
 case MPU6050_BAND_44_HZ:
 Serial.println("44 Hz");
 break;
 case MPU6050_BAND_21_HZ:
 Serial.println("21 Hz");
 break;
 case MPU6050_BAND_10_HZ:
 Serial.println("10 Hz");
 break;
 case MPU6050_BAND_5_HZ:
 Serial.println("5 Hz");
 break;
}

Serial.println("");
delay(100);
}
```

```
void loop() {

 /* Get new sensor events with the readings */
 sensors_event_t a, g, temp;
 mpu.getEvent(&a, &g, &temp);

 /* Print out the values */
 Serial.print("Acceleration X: ");
 Serial.print(a.acceleration.x);
 Serial.print(", Y: ");
 Serial.print(a.acceleration.y);
 Serial.print(", Z: ");
 Serial.print(a.acceleration.z);
 Serial.println(" m/s^2");

 Serial.print("Rotation X: ");
 Serial.print(g.gyro.x);
 Serial.print(", Y: ");
 Serial.print(g.gyro.y);
 Serial.print(", Z: ");
 Serial.print(g.gyro.z);
 Serial.println(" rad/s");

 Serial.print("Temperature: ");
 Serial.print(temp.temperature);
 Serial.println(" degC");
}
```

```
Serial.println("");
delay(500);
}
```

## IR Distance Sensor – GP2Y0A02YK

The GP2Y0A02YK IR sensors use infrared light to detect objects in their proximity. They consist of an infrared emitter and a photodiode receiver. The emitted infrared light reflects off objects and is detected by the receiver. By measuring the intensity of the reflected light, the sensors can calculate the distance to the object.

The distance calculation is based on a mathematical model that relates the sensor's analog voltage output to the distance. The sensors provide distance measurements in millimeters. To improve stability and accuracy, one 10uF capacitor was added for each sensor.

The sensors were controlled using Arduino programming and the average of multiple readings was taken to calculate a more stable result.

Also, the measurements from the datasheet were tested and the GP2Y0A02YK ir sensors have an output range of 0.4V to 2.4V.

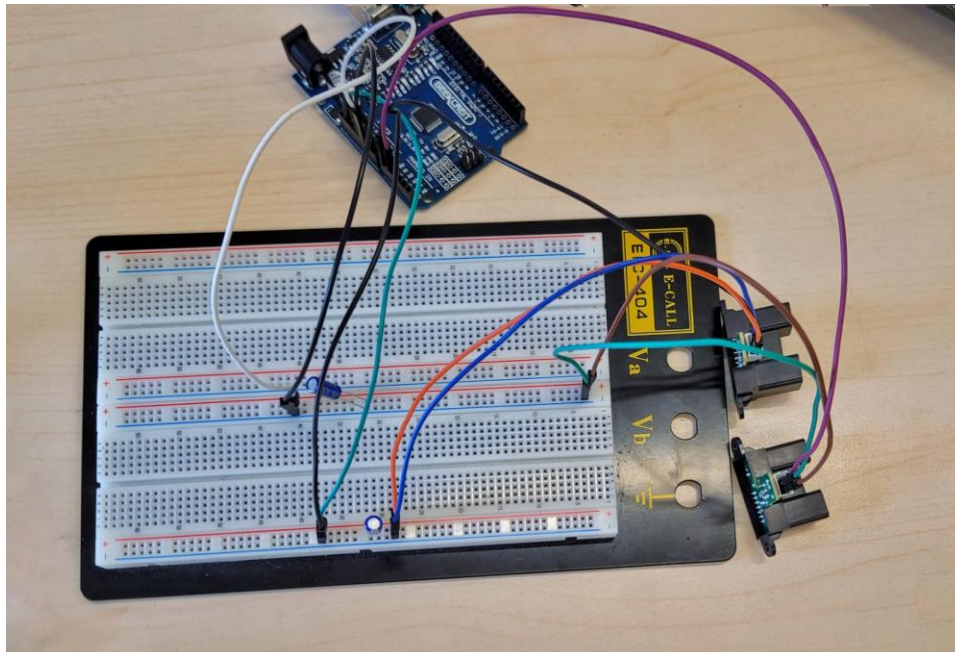


Figure 23 - IR unit testing

Code:

```
'''
```

```
void setup() {
 Serial.begin(9600);
}

float get_Sharp_GP2Y0A02YK_Distance(int Pinid) {
 float ADCValue = (float)analogRead(Pinid);

 //return distance in millimeters
 return 2583.711122992086
 - 20.197897855471 * ADCValue
 + 0.071746539329 * ADCValue * ADCValue
 - 0.000115854182 * ADCValue * ADCValue * ADCValue
 + 0.000000068590 * ADCValue * ADCValue * ADCValue * ADCValue;
}

void loop() {
 const int numReadings = 10; // average calculations
 int total1 = 0;
 int total2 = 0;

 for (int i = 0; i < numReadings; i++) {
 total1 += get_Sharp_GP2Y0A02YK_Distance(A0);
 total2 += get_Sharp_GP2Y0A02YK_Distance(A1);
 delay(10);
 }
}
```

```

 }

 int result1 = total1 / numReadings;

 int result2 = total2 / numReadings;

 Serial.print(result1 / 10);

 Serial.print(" ");

 Serial.print(result2 / 10);

 Serial.println();

 delay(500);
}

```

## H-bridge/Motor

Code:

```

int RPWM_Output = 5; // Arduino PWM output pin 5; connected to IBT-2 pin 1 (RPWM)

int LPWM_Output = 6; // Arduino PWM output pin 6; connected to IBT-2 pin 2 (LPWM)

void setup()
{
 pinMode(RPWM_Output, OUTPUT);
 pinMode(LPWM_Output, OUTPUT);
}

void loop()
{
 setSpeed(6);
}

void setSpeed(int speed){
 if(speed > 0){
 analogWrite(LPWM_Output, 0);
 analogWrite(RPWM_Output, map(speed, 0, 10, 0, 255));
 }else if(speed < 0){
 analogWrite(LPWM_Output, map(speed, 0, 10, 0, 255));
 }
}

```



```

 analogWrite(RPWM_Output, 0);
 }else{
 analogWrite(LPWM_Output, 0);
 analogWrite(RPWM_Output, 0);
 }

```

## 6 Conclusion

### 6.1 Hardware

#### 6.1.1 QTR-8RC sensor array (the incident)

Using the QTR-8RC that has 8 sensors (2 removable) which see the difference between reflectance, with black having the value 1000 (minimum reflectance) and white having the value 0 (maximum reflectance). Using all these 8 values it can determine the position of the line using the formula:

*Equation 1 - QTR-8RC formula*

$$\frac{(0 * v_0) + (1000 * v_1) + (2000 * v_2) + \dots}{(v_0 + v_1 + v_2 + \dots)}$$

With this formula it is possible to find out a precise position of the line, thus obtaining perfect line following algorithm.

#### 6.1.2 H-bridge

An H-bridge is an electronic circuit configuration used to control the direction and speed of a motor or actuator. It consists of four switching elements arranged in a bridge formation, enabling bidirectional current flow through the motor. The H-bridge is used to replace the absima driver.

#### 6.1.3 Long-distance IR sensors

The GP2Y0A02YK IR sensors use infrared light to detect objects in their proximity. The emitted infrared light reflects off objects and is detected by the receiver. By measuring the intensity of the reflected light, the sensors can calculate the distance to the object. Setting the limit when the sensors should signal the detection of an object makes it possible to have a nice, close and realistic avoidance.

By positioning the sensors one above each other it can make the difference between an obstacle and a ramp.

## 6.2 Software

6.2.1 Car class - In order to enhance the code's readability and ease-of-use, our team decided to create a dedicated class for our autonomous car. This class effectively serves as a library for the microcontroller, streamlining development and promoting ease of implementation.

Header file:

```
#ifndef Car_h
#define Car_h
#include <QTRSensors.h>
#include "Arduino.h"
#include <Servo.h>

class Car{
private:
 // External objects
 QTRSensors _qtrs;
 Servo _steering_servo;
 // Pins
 unsigned int _rpwm_pin;
 unsigned int _lpwm_pin;
 unsigned int _ir_pins[2];
 // Values
 uint16_t _qtr_values[6];

public:
 Car(unsigned int rpwm_pin, unsigned int lpwm_pin, unsigned int qtr_pins[6],
 unsigned int ir_pins[2], Servo &obj); // Constructor initializes everything
 // Movement logic
 void set_speed(int speed); // speed between -100 and 100
 void steer(int steering_angle); // steering_angle between -27 and 27 (negative
- left, positive - right)
 // Sensors logic
 void calibrate_qtrs(unsigned int min, unsigned int max);
 int get_line_position();
 float get_ir_distance(unsigned int ir_pin);
 bool is_obstacle();
};
#endif
```

Implementation:

```
#include "Car.h"
```

```

Car::Car(unsigned int rpwm_pin, unsigned int lpwm_pin, unsigned int qtr_pins[6],
unsigned int ir_pins[2], Servo &obj) {
 _rpwm_pin = rpwm_pin;
 _lpwm_pin = lpwm_pin;
 _steering_servo = obj;
 _ir_pins[0] = ir_pins[0];
 _ir_pins[1] = ir_pins[1];

 _qtrs.setTypeRC();
 _qtrs.setSensorPins((const uint8_t[])qtr_pins, 6);
 _qtrs.setEmitterPin(2);
}

float Car::get_ir_distance(unsigned int ir_pin)
{
 float ADCValue = (float)analogRead(ir_pin);
 return 2583.711122992086 - 20.197897855471 * ADCValue + 0.071746539329 *
ADCValue * ADCValue - 0.000115854182 * ADCValue * ADCValue * ADCValue +
0.000000068590 * ADCValue * ADCValue * ADCValue * ADCValue;
}

bool Car::detects_obstacle(){
 int x=0;
 int result2 = get_ir_distance(_ir_pins[0])/10;
 int result1 = get_ir_distance(_ir_pins[1])/10;

 for(int i=0; i < 15; i++){
 if(result2-result1>= 10 && result1>20 && result1<50)
 x++;
 else
 x=0;
 }
 if(x==15)
 return true;

 return false;
}

void Car::set_speed(int speed)
{
 if(speed > 0 && speed <= 100){
 analogWrite(_lpwm_pin, 0);
 analogWrite(_rpwm_pin, map(speed, 0, 100, 0, 255));
 }else if(speed < 0 && speed >= -100){

```

```

 analogWrite(_lpwm_pin, map(speed, 0, 100, 0, 255));
 analogWrite(_rpwm_pin, 0);
 }else{
 analogWrite(_lpwm_pin, 0);
 analogWrite(_rpwm_pin, 0);
 }
}

void Car::calibrate_qtrs(unsigned int min, unsigned int max) {
 _qtrs.calibrate();
 for (uint8_t i = 0; i < 6; i++)
 {
 _qtrs.calibrationOn.minimum[i]=min;
 _qtrs.calibrationOn.maximum[i]=max;
 }
 delay(100);
}

void Car::steer(int servo_angle){
 _steering_servo.write(90 + servo_angle);
}

int Car::get_line_position(){
 return _qtrs.readLineWhite(_qtr_values);
}

```

#### 6.2.2 Final.ino

```

#include "Car.h"

Servo steering_servo;
Car car(5, 6, (int[]){3, 4, 8, 9, 10, 11}, (int[]){14, 15}, steering_servo);
unsigned long delayDuration = 1200;
bool avoided = false;

void setup() {
 Serial.begin(9600);
 steering_servo.attach(7);
 car.calibrate_qtrs(1000, 2500);
}

void loop() {
 car.set_speed(12);
 if(car.detects_obstacle() && !avoided){

```



```
 for (unsigned long start = millis(); millis() - start < delayDuration;) {
 car.steer(17);
 }
 avoided = true;
}
else car.steer(map(car.get_line_position(), 0,5000, -27, 27));
}
```