Technical Design (TD)

Project: X

Company: REVO

Place, date: Enschede, 22nd March 2024

Drawn up by: Illia Guzerya

Anton Bucataru

Obidkhon Akhmadkhonov

Oleh Bielous

Version: 1.0

[Title page]

The same as the front page, but more info.

Place, date:	Enschede, 22nd March 2024		
Made by:	Project group 5		
	Illia Guzerya	547753	547753@student.saxion.nl
	Oleh Bielous	558059	558059@student.saxion.nl
	Anton Bucataru	538412	538412@student.saxion.nl
	Obidkhon Akhmadkhonov	553160	553160@student.saxion.nl
Version number:	[1.1]		
Clients:			
Waseem Elsayed			
Asif Khan			

Table of Contents

1 Introduction

The project aims to create a vehicle capable of autonomously following a line on a given base frame **(Model: ISH-010).** This is achieved through the integration of hardware and software, resulting in a car that moves independently along a predetermined path. The core mechanism involves a sensor detecting light reflections off the line, which, after processing, guides the car forward and steers it along any turns in the path by adjusting the wheels. This document outlines the **Technical design** of the project, elaborating on fundamental principles that were shown earlier, along with vehicle's conceptual framework. It includes detailed descriptions of both the software and hardware components involved. Additionally, we scope in details of functionality of our project and its peculiarities and specify information/models about parts used.

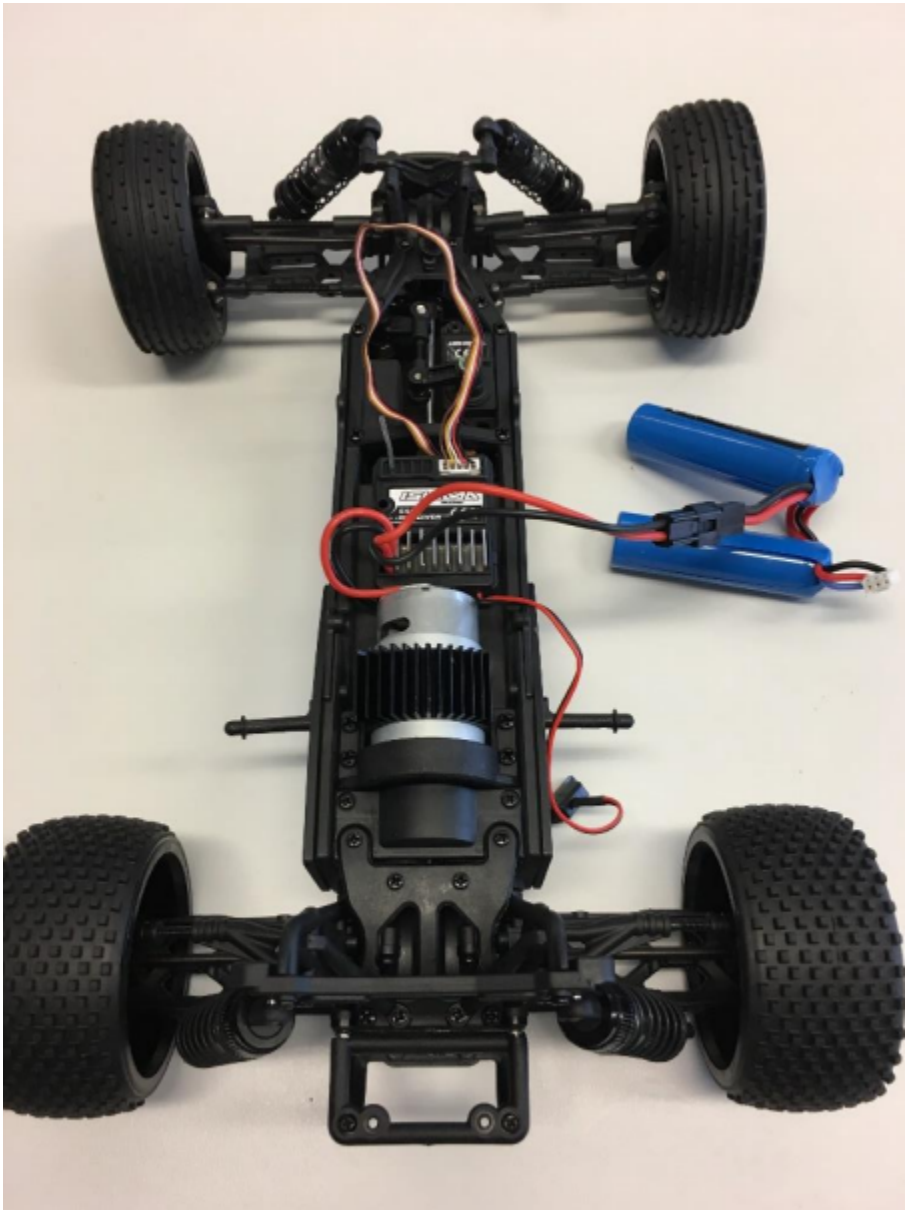


Figure 1.1 – IHS-010 frame.

1.1 Main requirements from previous phase(s)

Requirements:

- Complete autonomy of our car.
- Obstacle avoidance, line tracking, ramp detection algorithms.
- Completion of the given track.
- Stop at the end of the track.

See appendix A 1.1 – Flow chart describing main algorithm of our program.
See appendix A 1.2 – Main block diagram of the autonomous car.

Object detection	Ultrasonic sensors	Laser Range Finders	InfraRed distance sensors	
Processing Platform	Arduino Uno	FPGA	Raspberry Pi	Pc
Ramp Detection	Gyroscope	Magnetometer	Computer Vision	Tilt Switches
Wheel Steering	Rotating DC motor	Servo	Rotating AC Motor	Linear motor
Line Detection	IR sensors	Computer Vision		
Programming Language	python	C	C++	

Figure 1.1 – Enhanced Morphological diagram referenced from REVO functional design.

2 Analysis of requirements

The purpose of this section is to detail the selection criteria for each subsystem/function based on the main requirements from the previous phase(s). These criteria are technical characteristics relevant to each subsystem/function.

Given the goal to create a vehicle capable of autonomously following a line, avoiding obstacles, detecting ramps, and stopping at the track's end, let's outline the criteria for key components:

1. Object Detection

Sensitivity and Range: Ability to detect objects at varying distances (Weight: 5)

Accuracy: Precision in object identification and differentiation (Weight: 4)

Response Time: Speed of detection and processing (Weight: 5)

Reliability: Consistent performance under different environmental conditions (Weight: 4)

Cost: Affordability within project budget (Weight: 3)

Technology Maturity: Proven effectiveness and support (Weight: 3)

2. Processing Platform

Processing Power: Capability to handle complex computations efficiently (Weight: 5)

Connectivity: Options for updating software, debugging, and potential integration with additional modules (Weight: 4)

Compatibility: With chosen sensors, actuators, and other hardware components (Weight: 5)

Ease of Programming: Support for programming languages and development environments (Weight: 4)

Power Consumption: Efficiency in energy use (Weight: 3)

Size and Weight: Compactness and lightness for integration into vehicle chassis (Weight: 2)

3. Ramp Detection

Sensitivity: Ability to detect small changes in inclination (Weight: 4)

Accuracy: Precision in measuring angles of inclination (Weight: 4)

Reliability: Stable performance across different surfaces and conditions (Weight: 3)

Integration: Compatibility with the processing platform and other sensors (Weight: 3)

Cost: Keeping within the allocated budget for the project (Weight: 2)

4. Wheel Steering

Response Time: Quickness in steering action response to control commands (Weight: 5)

Precision: Accuracy in achieving desired steering angles (Weight: 5)

Range of Motion: Ability to achieve a wide range of steering angles for maneuverability (Weight: 4)

Durability: Resistance to wear and tear from regular use (Weight: 3)

Cost: Affordability and value for money (Weight: 2)

5. Line Detection

Sensitivity: Ability to detect line under various light conditions (Weight: 5)

Detection Range: Distance from which the line can be detected (Weight: 4)

Response Time: Speed in processing line detection to action (Weight: 5)

Reliability: Consistent line detection across different track materials and colors (Weight: 4)

Integration: Ease of integration with processing platform and steering mechanism (Weight: 3)

6. Programming Language

Library Support: Availability of libraries for interfacing with hardware, image processing, and other necessary functions (Weight: 4)

Ease of Use: Learning curve and developer productivity (Weight: 3)

Community Support: Access to forums, documentation, and troubleshooting resources (Weight: 3)

Performance: Efficiency in executing written programs (Weight: 2)

Compatibility: With selected processing platform and sensors (Weight: 4)

3 Component/Unit/Module selection

3.1 Options

Using our **morphological diagram (Figure 1.1)** we are now able to create a set of options to choose from. There will be total of 3 options from which we are going to conclude our final set of main pieces of equipment.

1st Option:

- *Object Detection: Capacitive Sensors*
- *Processing Platform: FPGA*
- *Ramp Detection: Tilt Switches*
- *Wheel Steering: Servo*
- *Line Detection: IR sensors*
- *Programming Language: C*

Pros:

- Capacitive Sensors provide versatile object detection.
- FPGA offers high-speed processing suitable for real-time control.
- Tilt Switches offer simplicity, not requiring complex algorithm to operate.
- Servo-based steering provides simple yet effective direction management.
- IR sensors is generally accepted as a good option for line tracking due to its simplicity and effectiveness.

Cons:

- FPGA requires more knowledge to use that MC like Arduino.
- Capacitive sensors may require calibration, adjustments.
- Tilt switches may have limited sensitivity and accuracy compared to other ramp detection methods.
- Computer vision will probably be an overkill for this task, it is too complex to handle video processing for a task of line tracking.

Resulting Analysis:

This option combines effective technologies like FPGA or computer vision and although its pros promise a lot, the flaws might take over. Finally, we do think that there might be something more powerful/sophisticated than this and will look for more interesting solution.

2nd *Option:*

- *Object Detection: computer vision*
- *Processing Platform: Raspberry Pi*
- *Ramp Detection: Magnetometer*
- *Wheel Steering: Rotating dc motor*
- *Line Detection: Computer Vision*
- *Programming Language: python*

Pros:

- Computer Vision for object and line detection offers flexibility and complete accuracy in navigation.
- Raspberry Pi provides rich processing power and connectivity options for advanced functionalities (e.g. Wi-Fi connection).
- Magnetometer detection can provide a non-contact reliable solution for quick detection of changes in orientation, ensuring effective completion of ramp.
- Programming in Python provides us with simplicity and a broad range of libraries and online references, which will be useful for development and prototyping with Raspberry Pi.

Cons:

- Raspberry Pi usage will result in additional complexity and potential portability issues.
- Computer Vision algorithms may require significant computational resources(as well as complexity of development), impacting real-time performance.
- Magnetometer detection may be really sensitive to interference from external magnetic fields which will result in loss of precision.
- Computer vision will probably be an overkill for this task, it is too complex to handle video processing for a task of line tracking.

Resulting Analysis:

In search of something powerful, our team came to a very thin edge. This option seems really interesting, but quite frankly hard. It might be an overkill, autonomous car implementation can be achieved with more laconic, efficient way. In the last option we will be more pragmatic.

3rd Option:

- *Object Detection: Ultrasonic sensors*
- *Processing Platform: Arduino UNO*
- *Ramp Detection: Gyroscope*
- *Wheel Steering: Rotating dc motor*
- *Line Detection: IR sensors*
- *Programming Language: C/C++*

Pros:

- Ultrasonic sensors usage is generally used in a task of object detection. It provides effectiveness and a good range capabilities.
- Arduino UNO is a good Micro Controller and has enough computing power for this task. Also, our team is familiar with this MC, so operating it will be easier.
- Gyroscope-based ramp detection has one main benefit – it is inbuilt inside the Arduino, so we can combine these two options, leaving us without a need in extra hardware piece.
- Rotating DC motor for wheel steering provides simplicity and affordability.
- IR sensors is generally accepted as a good option for line tracking due to its simplicity and effectiveness.

Cons:

- Arduino Uno's limited processing power may limit the complexity of algorithms and functionality.
- Ultrasonic sensors may have limitations in accuracy compared to other detection methods.
- Gyroscope-based ramp detection may require calibration for optimal performance.

Resulting Analysis:

Finally, given set of options is a good balance between functionality and effectiveness, all of the components are providing great purpose and promise to be a good combination. Our team will stick to this prototype of technical design for our autonomous car.

4 Technical design overview

4.1 Overview of initial hardware design

The design includes the following main components:

- Object Detection System (Ultrasonic sensors): Detects obstacles in the vehicle's path.
- Processing Platform (Arduino UNO): Serves as the brain of the vehicle, processing input from sensors and controlling actuators.
- Ramp Detection System (Gyroscope): Detects inclines and declines on the path.
- Wheel Steering Mechanism (Rotating DC motor): Controls the steering of the vehicle.
- Line Detection System (IR sensors): Identifies and follows the line on the track.
- Power Supply (Li-ion Battery): Provides power to the system.

Interfaces

When integrating these components, certain interfaces may require additional components for compatibility:

- Voltage Level Matching: Since the Arduino UNO operates at 5V, and if any component (like certain ultrasonic sensors) requires a different voltage, a step-up or step-down voltage converter will be necessary.
- Signal Conversion for Gyroscope and Ultrasonic Sensors: If the output of the gyroscope or ultrasonic sensors is not directly compatible with the Arduino's input, signal conditioning circuits or interface modules may be needed.
- Motor Driver for DC Motor: To interface the DC motor with the Arduino, a motor driver that can handle the motor's voltage and current requirements is essential. This ensures that the Arduino can control the motor speed and direction safely.
- Pull-Up/Pull-Down Resistors for IR Sensors: If the IR sensors require stabilization of input signals to the Arduino, pull-up or pull-down resistors might be necessary to ensure reliable readings.

4.2 Interfaces

The revised design would incorporate these additional components as follows:

- *The Arduino UNO is centrally connected to all sensors and the motor driver, orchestrating the vehicle's operations.*
- *Ultrasonic sensors are connected directly to the Arduino, with signal conditioning if necessary.*
- *The Gyroscope is connected to the Arduino, potentially through an interface module for signal compatibility.*
- *The Rotating DC motor is connected via a Motor Driver, which is controlled by the Arduino.*
- *IR sensors for line detection are directly connected to the Arduino, with pull-up/pull-down resistors added to their connections for signal stability.*

- *A Voltage Converter is used where necessary to match the power supply voltage with component requirements.*
- *This setup ensures that all components can communicate and function together effectively, with the Arduino UNO acting as the central processing unit. Adjustments and additions of interface components like voltage converters, motor drivers, and resistors ensure that the electrical and signal requirements of each component are met, allowing for seamless integration and operation of the autonomous vehicle.*

5 Technical requirements

5.1.1 Component #1 (e.g. Li-ion Battery)

#	Requirement	Relation	Value	Unit
TR001	Maximum voltage level	≤	7.4	Volt
TR002	Number of cycles	>	1000	Cycles
TR003	Complete discharge time for single charge at maximum power	>	30	minutes

- If **TR001** ensures compatibility with the vehicle's electrical system, preventing overvoltage issues.
- **TR002** ensures the battery's longevity, reducing the need for frequent replacements.
- **TR003** guarantees that the vehicle can operate sufficiently long on a single charge, crucial for completing tasks without mid-operation recharging.

5.1.2 Component #2 (RC 540 motor)

The RC 540 motor, selected for propelling the vehicle, needs to match the electrical system's output and not exceed current limits to maintain efficiency and prevent overheating.

#	Requirement	Relation	Value	Unit
TR011	Nominal/operating voltage level	=	7.4	Volt
TR012	Maximum current level	≤	17	Amps

- **TR011** is chosen to match the motor with the system's power supply ensuring efficient operation.
- **TR012** limits the motor's current draw to prevent overheating and ensure the system's energy efficiency.

5.1.3 Software function #1 (e.g. RampControl)

#	Requirement
TR101	When ValueSensor > 300 , then PowerMotorIncrease()
TR102	When ValueSensor < 0 , then PowerMotorDecrease()

- **TR101** increases motor power when an upward incline is detected, ensuring the vehicle can climb effectively.
- **TR102** decreases power in a downward incline or flat terrain to conserve energy and control speed.

5.1.4 Software function #2 (e.g. ObstacleDetection)

ObstacleDetection software function enables the vehicle to navigate around obstacles by steering accordingly based on sensor input, crucial for avoiding collisions and ensuring smooth operation.

#	Requirement
TR111	When ValueSensor > 15 , then SteeringRight()
TR112	When ValueSensor < 0 , then SteeringLeft()
TR113	When ObstacleDetection=TRUE, then PowerMotorDecrease()

- **TR111** and **TR112** guide the vehicle to steer away from obstacles, ensuring it remains on its intended path.
- **TR113** initiates a slowdown or stop when an obstacle is detected too close, enhancing safety.

5.1.5 Mechanical Requirements for Vehicle Chassis

#	Requirement	Relation	Value	Unit
TR021	Length	=	380	mm
TR022	Width	=	255	mm
TR022	Height	=	155	mm
TR023	Weight	=	1410	g

- **TR021** and **TR022** ensure the vehicle's size is suitable for the track, providing enough surface area for component installation while enabling maneuverability.
- **TR023** limits the vehicle's height to ensure stability by maintaining a low center of gravity.
- **TR024** targets a lightweight design for the chassis to enhance efficiency and reduce power consumption.

5.1.6 Mechanical Requirements for Wheel and Steering System

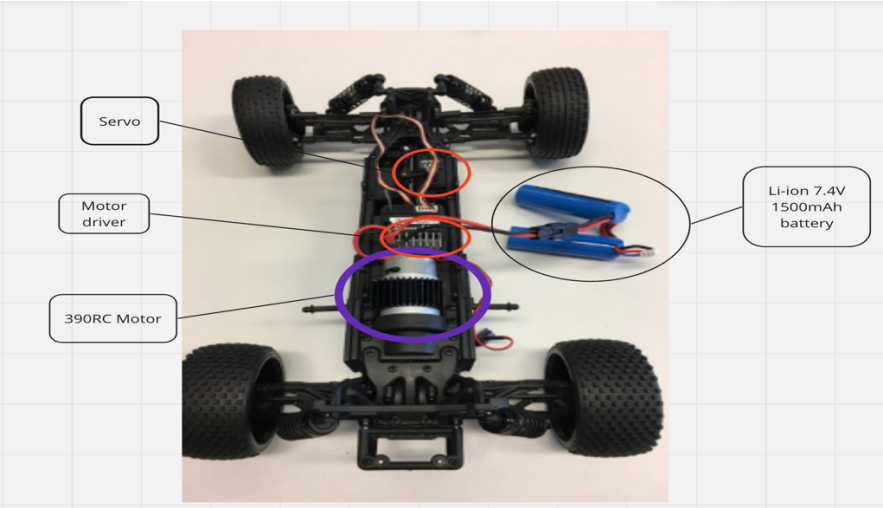
The wheel and steering system directly influence the vehicle's ability to navigate the track accurately.

#	Requirement	Relation	Value	Unit
ID	Requirement	Relation	Value	Unit
TR031	Wheel Diameter	=	88	mm
TR032	Wheel Width	=	42	Mm (front)
TR033	Wheel Width	=	42	Mm (rear)
TR034	Steering Range	≥	30	Degrees
TR035	Material	-	Rubber	-

- **TR031, TR032** and **TR033** define the dimensions of the wheels to ensure they provide adequate traction and are capable of handling the track's terrain.
- **TR033** specifies the minimum steering range to ensure the vehicle can navigate turns effectively.
- **TR034** indicates the choice of material for the wheels, emphasizing traction and durability.

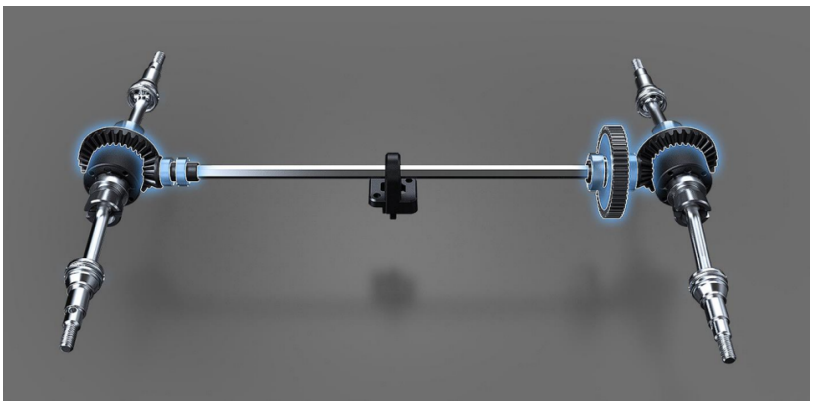
6 Detailed technical design

6.1 Mechanical

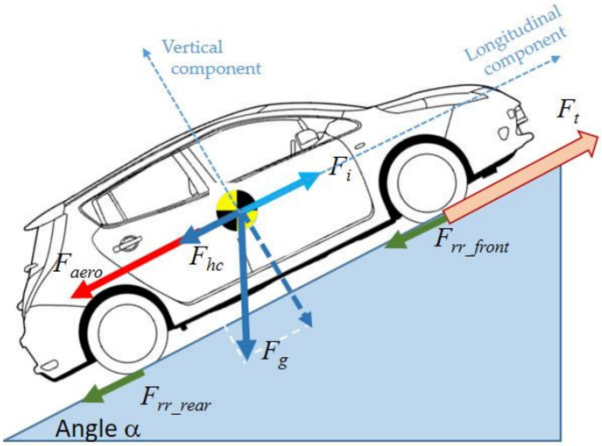


Car components fig.6.1.1

In figure nr.6.1.1 all the basic car components are shown inside the car base, excluding the Arduino and sensors.



transmission firg.6.1.2

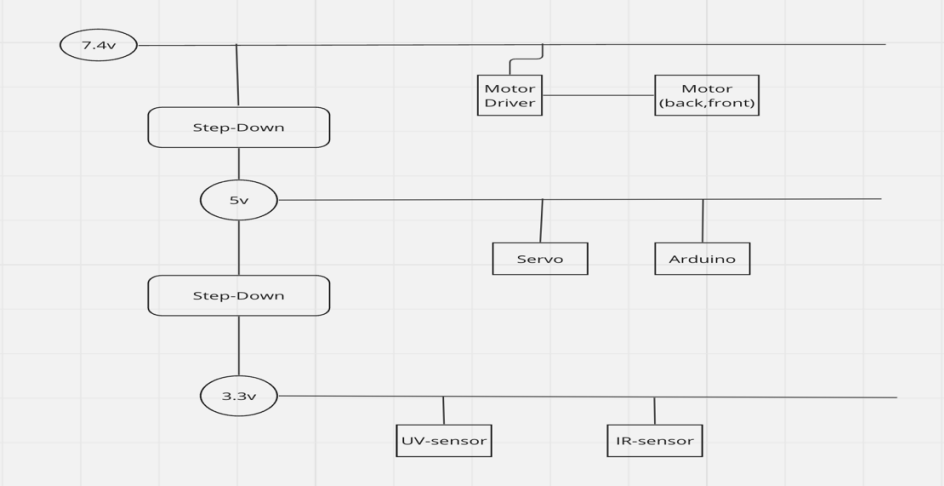


body diagram fig.6.1.3

Figure 6.1.2 Show the chassis of our car project which has rear wheel power drive.
On the other hand the figure 6.1.3 shows how different physical forces are distributed on the car while it is on the ramp.

6.2 Electrical

li-ion battery fig.6.2.1



One line diagram fig6.2.2

The one-line diagram above describes how our energy is being distributed through all of the components, here you can see that raw power of 7.4 goes directly to the motor drove and main motor, after a step-down convertor the load becomes 5 volts which is distributed to the servo motor (is used for turnings) and to the Arduino itself, after the last step-down convertor which is inbuilt in Arduino power becomes 3.3 volts and is distributed to the sensor.

Battery: Li-ion 2s 1p 7.4v 1500mAh

Safe discharge: reducing the capacity at which a Li-ion battery (also known as lithium-ion) is regarded as fully discharged for protection purposes,

however, no matter how low it might be, lithium ion batteries cannot be discharged to below two point five volts while Li-Po batteries can't go below three volts.

(Do not discharge below 3 volts or battery damage will occur!)

30 minutes of power can be obtained by discharging the battery at 80 % of its capacity without risking over discharge (while continuously pulling a current of 2.5 amps).

Voltage conversions:

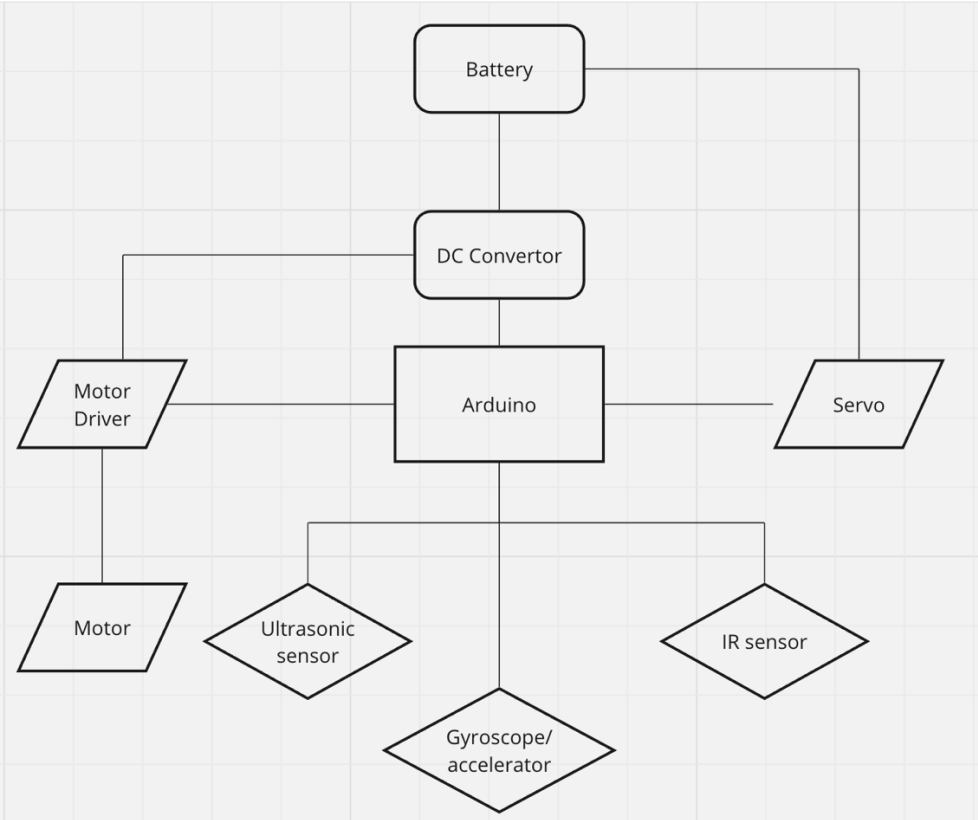
- Charger: Li-Po 7.4V

- 7.4v Li-ion-> step-down converter (5v,3.3v etc)

- 7.4v Li-ion-> Motor Driver-> Motor(s)

- Step-down converter (5V)-> servo motor, Arduino.

- Down converter 3.3v-> Sensors



Electrical flow-chart fig.6.2.3

The electrical flow chart represented above shows to the reader how the power load is flowing to the components, starting with Battery which is on the top going to the Servo and Motor driver, after that is flowing through DC convertor to the Arduino and after that it is distributed to the sensors.

6.3 Power Consumption

Given following energy sources:
Reely NiMh accupack 7.2V 2000mAh 15C

Energy consumers:
S60PH Sport Servo motor (average 1-2 Amps)

HC-SR04 Ultrasonic sensor (average 2mA)
Pololulu qtr-8rc InfraRed sensor (average 90-100 mA)
Brushed DC motor 540 (average maximum consumption (22,79 Amps)

Arduino UNO (average 500mA)

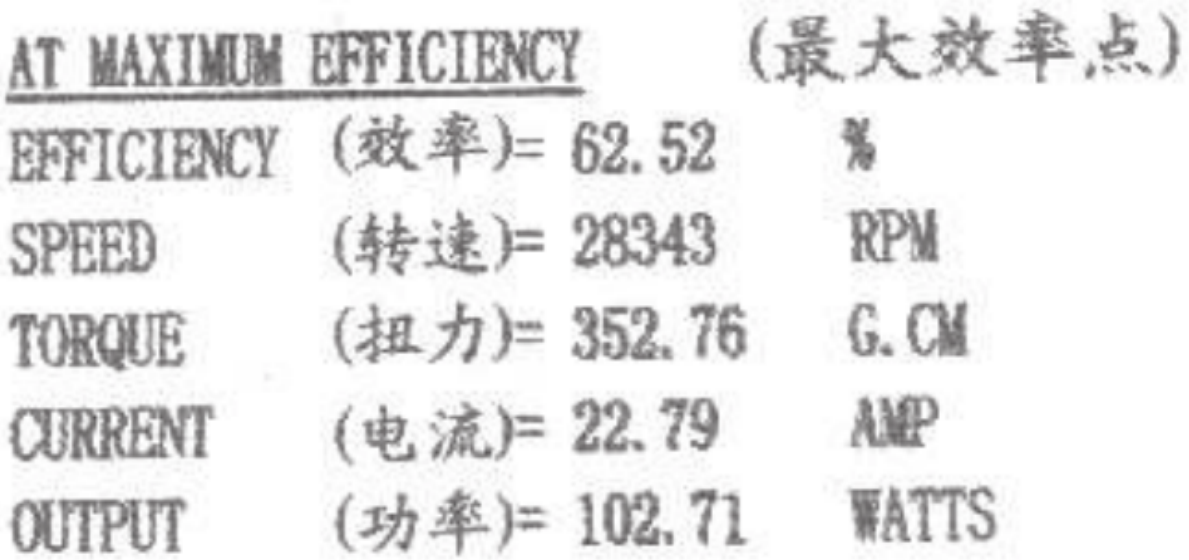


Figure 6.3.1 Brushed Dc motor 540 power consumption at maximum efficiency datasheet screenshot

Average maximum consumption = 22.79A (motor) + 0.095A (IR sensor) + 0.002A (UltraSonic sensor) + 2A (servo motor) + 0,5A (Arduino) = 25,387A

Using batter life formula we can calculate approximated time of work with maximum consumption:

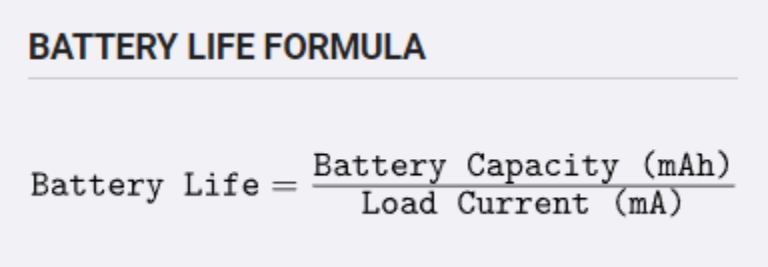


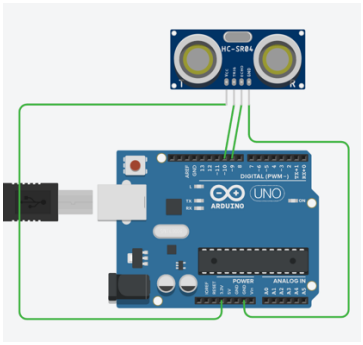
Figure 6.3.2 Battery Life formula.

Accounting for battery discharge safety (at 20%) we say that our capacity is 2000 – 400(20%) = 1600

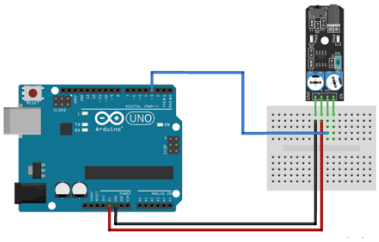
Battery life = 1600mAh / 2530 mA = 0.63 Hours = 38 minutes.

Theoretically speaking our car will be able to function 38 minutes in a row.

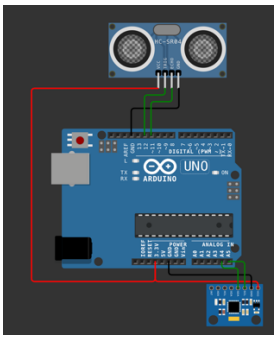
6.3 Electronic



Ultrasonic connection fig.6.3.1



Infrared connection fig.6.3.2



Gyroscope connection fig.6.3.3

In these three pictures above which unfortunately can't be combined because of lack of the components that each CAD persist of, but all the connections will be mentioned in this description.

The Ultrasonic sensor HC-SR04 has 4 pins: ground, VCC beside that two Echo and Trigger also which are connecting to the PWM pins of Arduino.

The next sensor that will be used is Pololu QTR-8RC which is not represented here because it is not existing in the digital way, the difference of this one comparing it to the simple IR that I represented is that Pololu is an array that has 8 IR sensors build-in so it is easier to operate an it will be connected also to same ground and power as Ultrasonic and to the other 8 pins of Arduino.

The last sensor represented is MPU 6050 Gyroscope and Accelerometer is a Micro Electro-Mechanical Systems (**MEMS**) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motions related parameter of a system or object. Vcc Provides power for the module, can be +3V to +5V. Typically +5V is used, Ground Connected to Ground of system, Serial Clock (SCL) Used for providing clock pulse for I2C Communication, Serial Data (SDA) Used for transferring Data through I2C communication, Auxiliary Serial Data (XDA) Can be used to interface other I2C modules with MPU6050. It is optional, Auxiliary Serial Clock (XCL) Can be used to interface other I2C modules with MPU6050. It is optional, AD0 If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address and Interrupt (INT) to indicate that data is available for MCU to read. From this bid diversity of pin, we will use only two of them beside the power and ground is SCL (serial clock) and SDA (serial data) both are connecting to the Analog pins of Arduino.

To sum up the information that we described above, after connecting all the components 13 pins of Arduino will be used.

6.4 Software

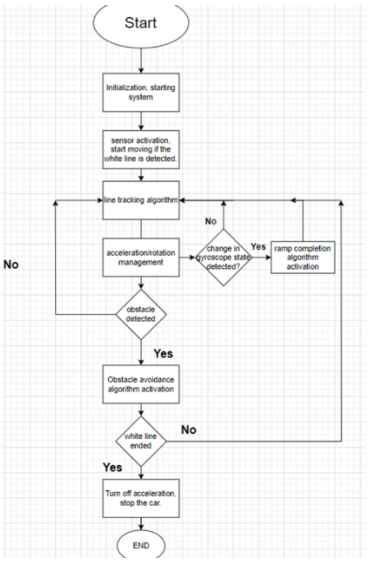


Fig.6.6.1 Overview code flow chart

References

Number	Author(s)	Title
[1]	101 components	HC-SR04 Ultrasonic Sensor
[2]	101 Components	MPU6050 Accelerometer and Gyroscope Module

Appendix A:

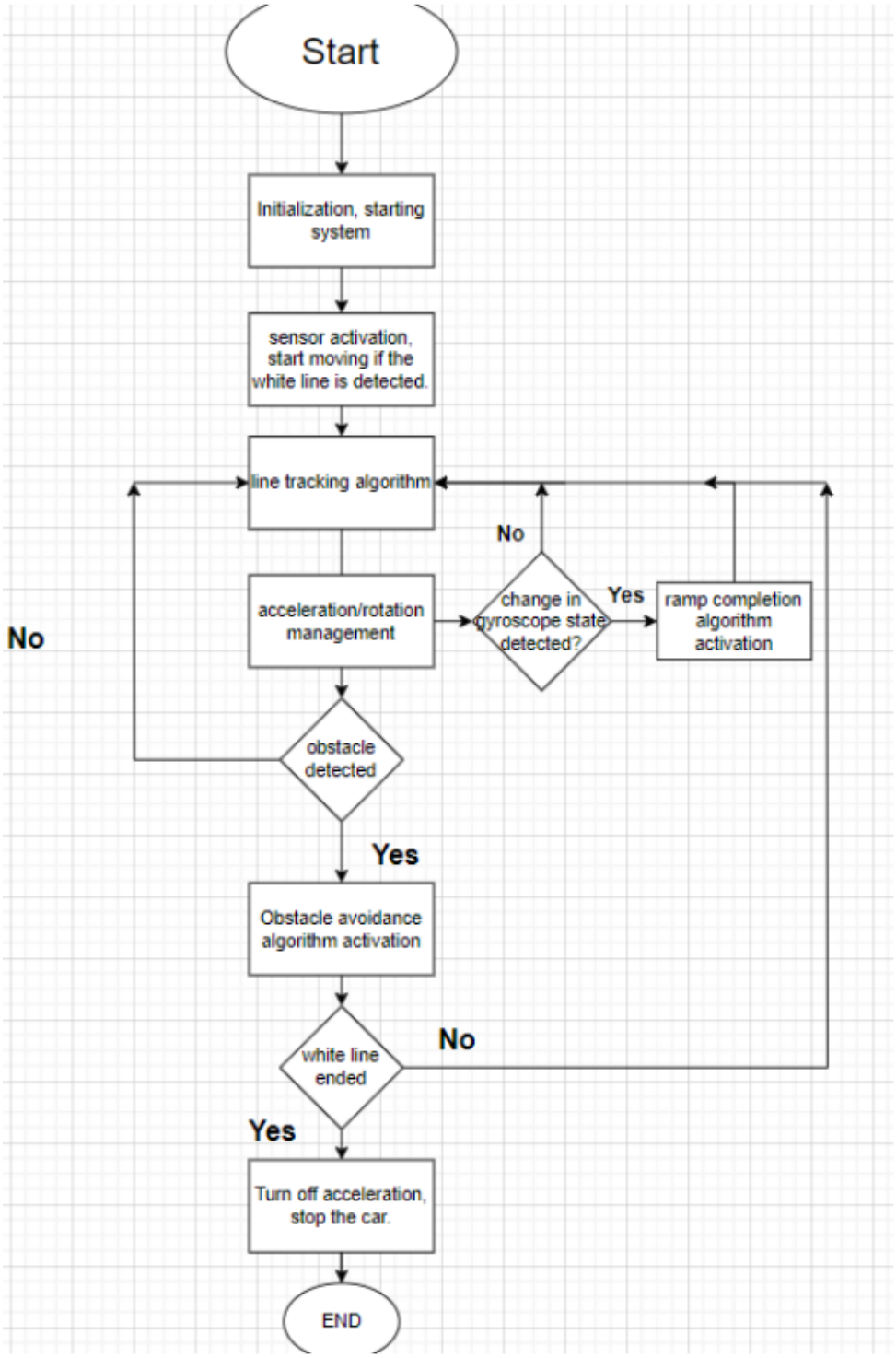


Figure 1.2 – Flowchart describing main algorithm of our project.

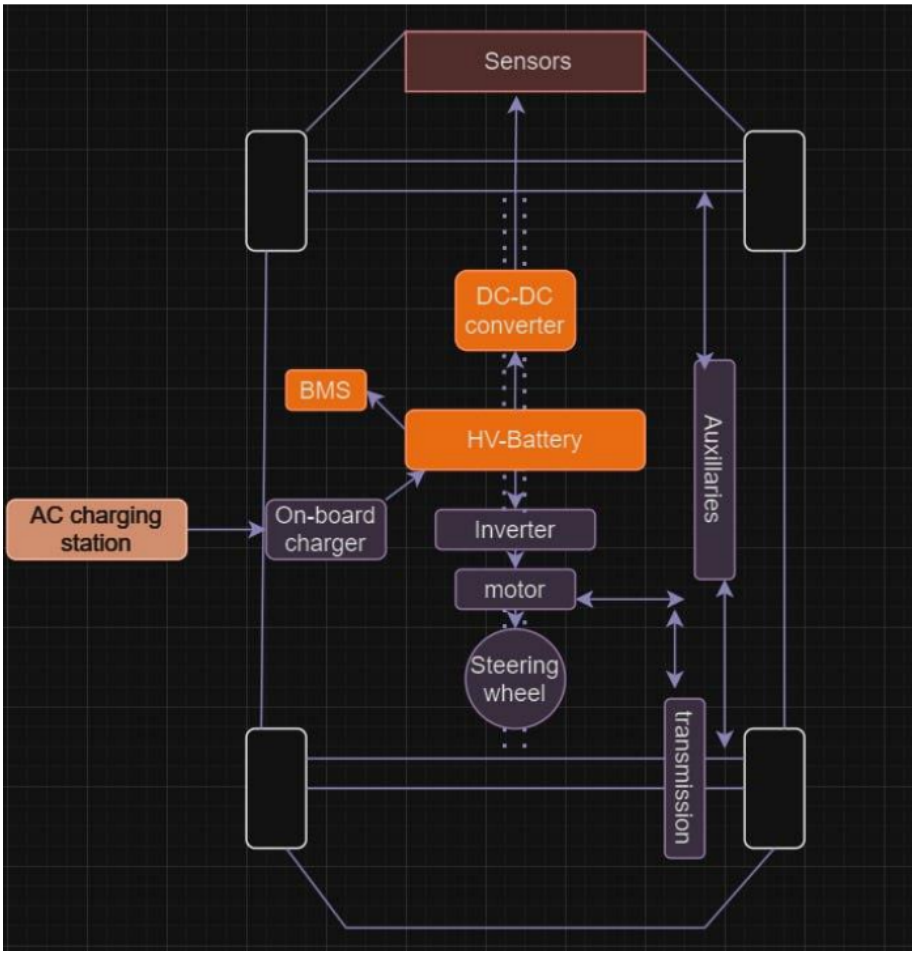


Figure 1.3 – Main block diagram of the autonomous car.

Sample code for Ultrasonic sensor:

```
void loop() {
  digitalWrite(10, LOW);
  delayMicroseconds(2);

  digitalWrite(10, HIGH);
  delayMicroseconds(10);
  digitalWrite(10, LOW);

  // Measure the duration of the pulse on the echoPin
  int duration = pulseIn( 9, HIGH);

  // Calculate distance in centimeters
  int distance = duration * 0.034 / 2;

  // Print the distance to the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Wait a short time before taking the next measurement
  delay(1000);
}
```

Sample code for gyroscope:

```
const int IR_PIN = 2;  // Define the pin to which the IR sensor

void setup() {
  Serial.begin(9600);  // Start serial communication
  pinMode(IR_PIN, INPUT);  // Set IR pin as input
}

void loop() {
  int irValue = digitalRead(IR_PIN);  // Read the value from the IR

  if (irValue == LOW) {  // If obstacle detected
    Serial.println("Obstacle Detected!");
  } else {  // If no obstacle detected
    Serial.println("No Obstacle");
  }

  delay(500);  // Delay for stability
}
```

Sample code for Infrared sensor:

```
#include <Wire.h>

const int MPU_addr = 0x68; // MPU-6050 I2C address

int16_t gyro_x, gyro_y, gyro_z; // Raw gyro values

void setup() {
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop() {
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x43); // Start with register 0x43 (GYRO_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 6, true); // Request 6 bytes for gyro data

  gyro_x = Wire.read()<<8 | Wire.read(); // Combine high and low bytes
  gyro_y = Wire.read()<<8 | Wire.read(); // Combine high and low bytes
  gyro_z = Wire.read()<<8 | Wire.read(); // Combine high and low bytes

  Serial.print("X: "); Serial.print(gyro_x);
  Serial.print(" Y: "); Serial.print(gyro_y);
  Serial.print(" Z: "); Serial.println(gyro_z);

  delay(100);
}
```