Self-autonomous moving car
NexusTech
Final Report

Version: 1.1

Table of contents:

1. Introduction

Background

This project is part of a larger initiative aimed at developing an autonomous real-life car. This project is simply a miniature version of the car, designed to test hypotheses and theories in a controlled manner to assess the feasibility of creating a real-life model through a scaled-down prototype. NexusTech has created a working model with all the necessary requirements that a car would need to be self-sufficient. In order to achieve this milestone, it had to overcome countless tribulations, both from a practical standpoint when implementing the technical design, and internal matters such as the restructuring of team members, the addition of new personnel, and the departure of individuals due to personal reasons. These issues complicated communications, but eventually NexusTech managed to regroup and successfully complete the project

The assignment

The task assigned to Nexus Tech involved developing a miniature self-driving car capable of following a white line, avoiding obstacles, and navigating ramps. First, NexusTech created a project plan outlining all their goals, the costs they expected, and possible issues that may arise and how to fix them. Then they created a functional design outlining all the things they wanted to make, but it was mostly an overview, which involved functional blocks and high-level flowcharts. Then they made a more technical design with all the components they chose and more detailed flowcharts, not to mention the 3D design they had created for printing and covering the car.

Approach and methodology

The task assigned to Nexus Tech involved developing a miniature self-driving car capable of following a white line, avoiding obstacles, and navigating ramps. First, NexusTech created a project plan outlining all their goals, the costs they expected, and possible issues that may arise and how to fix them. Then they created a functional design outlining all the things they wanted to make, but it was mostly an overview, which involved functional blocks and high-level flowcharts. Then they made a more technical design with all the components they chose and more detailed flowcharts, not to mention the 3D design they had created for printing and covering the car.

2. Execution of assignment

Project Objectives

The main objective of this project was to create a self-driving miniature car. It consisted of four sub-objectives:

1. Following the line: The car should be able to follow the white line even when it curves.

2. Avoiding obstacles: The car should be able to maneuver around any obstacles in its path.
3. Going up a ramp: If the car encounters a ramp, it should recognize this and accelerate to maintain sufficient speed to go over it. It should also be able to decelerate when going downhill.
4. Stopping at the end: The car should stop at the end of the line.

Project activities Develop

a project plan: Introduction

Project Objectives

Project Boundaries

Results

Quality Assurance

Project Organization

Planning

Costs and Benefits

Risk Analysis

Create a functional design document:

Functional Design

Functional Blocks

Three Functional Design Concepts

Block Diagram

High-Level Flow Chart

Testing Requirements

Draw up a technical design document:

Low-Level Design (Flow Chart)

Test Cases

Protocols

Standards

List of chosen sensors, power supply, and other components for the car

Technical Drawings

Technical Schematics

3-D Model View

Start the implementation phase:

Create the code for the car

Build the PCB

Verify functionality using previously designed tests

Assemble all sensors, power supply, servo motors, PCB, shields, and 3-D design cover to construct the actual car

Create a final report:

Detailed information about NexusTech's activities in creating the functional design

Detailed information about NexusTech's activities in creating the technical design

Detailed information about NexusTech's activities during the implementation phase

Project boundaries

The project was required to be completed within 16 weeks (approximately 3 and a half months) from the start of the module, which aligns with the end of quarter 4. The final deadline for submission was Friday, 2nd June at 23:59, as indicated in Chapter 3.

The total budget allocated for the project was a maximum of 50€. The specific terms and conditions related to the budget were subject to discussion.

The group established certain deadlines to manage time effectively and meet project requirements. While these deadlines served as guidelines, they were not strictly enforced. Their purpose was to ensure a balanced workload for group members and minimize stress.

The product's scope included the following functionalities:

- Traversing a track marked on the ground by lines.
- Avoid obstacles.
- Returning to the track if it deviated from it.
- Stopping after completing the course.
- Operating independently on its own batteries for a minimum of 30 minutes before requiring recharging.
- Accounting for changes in the slope of the track.

It's important to note that the product was not designed to withstand a wide range of operating conditions such as extreme weather or temperature variations. It was specifically designed for indoor

use, at temperatures close to room temperature, with moderate humidity and no direct contact with liquids. Additionally, the product was not built to withstand impacts greater than those resulting from collisions at its maximum speed during level travel in a straight line.

Quality assurance

To ensure the quality of their work, the team implemented several measures aimed at identifying and correcting any errors or defects in their processes or deliverables. NexusTech established clear quality standards and procedures, defining and documenting them for each stage of their project, including planning, execution, monitoring, and evaluation. The company used the morphological methodology to systematically explore and combine different solutions or options, creating a comprehensive solution that met quality and safety standards. They broke down potential quality and safety issues into their component parts and explored different solutions to address them. This approach allowed them to ensure that their work met the highest quality and safety standards and addressed any potential issues in a systematic and comprehensive manner.

Planning

The project was divided into 4 parts the project plan, functional design, technical design, implementation, and tests. The time and dates of each part and their own sub-objectives was decided down to the day, and was put in a Gantt plan you can see in Appendix A.

Costs and benefits THE COST TABLE:

NAME	DESCRIPTION	QUANTITY	PRICE/UNIT (€)
SS-1010200	Grove - Ultrasonic Ranger	2.00	4.50

20310	Infrared reflection sensor Line follower sensor	3.00	1.45
2136256 - 8j	Joy-it MPU6050 Acceleration sensor	2.00	5.49
184543 -8j	Vossloh Schwabe WU8-53HD LED wired Red Circular	6.00	0.10832
791 - 6454	MIKROE-512, 150mm Insulated Breadboard Jumper Wire in Black, Blue, Brown, Green, Grey, Orange	2.00	4.27

TOTAL Excl. BTW (€): 33.52

Benefits:

The car has a lot of benefits such as it being a working design of a car that can drive by itself regardless if there is an obstacle or a ramp. As this proof that it is possible to achieve a full sized autonomous car.

Risk analysis

Risk	Risk level	Response
Insufficient components	**HIGH**	Need to check frequently with the components and the supplier to assure components accuracy Order the new components using the save up money or find replacement. There will also be a delay in delivery, so it is better to order in advance
Accidentally break a components	**HIGH**	Order multiple components of each type (preferably 2 of each) so that if it is broken, we can still work without delay

Testing fail	MEDIUM	To prevent this from happening, the team should test each part of the car (white line, ramp detection,) multiple times before assembling the car
Not enough distinction between our project and other team project (company risks)	LOW	The project will have some latter integration upgrade that will differentiate our product than the others

3. Functional Design

1. Introduction

For this project NexusTech is creating a miniature self-driving car that follows a white line while avoiding obstacles that also stops at the end of the line. In this document NexusTech's will detail all the steps they took to reach a working functional design. Which will include analyzing client requirements and identifying the biggest problems. Then produce possible concept principles for said problems by creating sketches, equations, block diagrams. After that we look at our concepts and at the system requirements at the same time and determine if the concepts really fulfil them and drop those that do not. The next step is having some basic hardware tests for concepts that need them. Furthermore, is the evaluation of the concept principle and the generation of several the most promising concept designs, in the form of

sketches, block diagrams of servo control, electr(on)ical hardware and/or software. And finally analyzing and comparing each concept with the system requirements to choose the most promising one. Thus, the document will show the best functional design NexusTech has come up with among the ones they come up with and how they arrived at that conclusion.

2. **Analysis of requirements**

Code	*Function*	*Requirements*	*Relation*	*Value*	*Unit*
RQ001	*Starting the car*	Turn on at the beginning of the track	Relation to RQ002, RQ006 and RQ009. At the beginning of the track, of a ramp or while avoiding an obstacle, the car must turn on.	NONE	NONE
RQ002	*Detecting the line*	Detect the line it has to follow.	Relation to RQ003, RQ005 and RQ008. The car must be able to detect the beginning of the line that it must follow; it must also be able to detect the line after avoiding an obstacle.	NONE	NONE
RQ003	*Following the line*	Following it when it curves or turns.	Relation to RQ005, RQ007, RQ008. If there is an obstacle, avoiding it takes priority over following the line. When going over the hill the car should still follow the line.	True or False	NONE
RQ004	*Controlling the steering*	Being able to change direction.	Relation to RQ003. The car must steer whenever it reaches a curve so that it can continue following the line.	Angle (0 –180)	Angle Degrees
RQ005	*Detecting the obstacle*	Detecting the obstacle and stopping the car before collision.	Relation to RQ003 and RQ006. The process of avoiding the obstacle can only start once an obstacle has been detected;	True or False	cm
			and the car would only manage to find the obstacles by following the line.		
RQ006	*Avoiding the obstacle*	Be able to maneuver around the obstacle and return to the line.	Relation to RQ002. The car must be able to turn around and move past the obstacle; then it should return and follow the line.	NONE	NONE

RQ007	*Controlling the speed*	Being able to control the acceleration of the car.	Relation to RQ009. To avoid accidents, the speed the car must reach is calculated once it detects a ramp. It must be fast enough to go up the ramp, but also slow enough to avoid gaining momentum that could lead to it crashing.	Velocity	m/s
RQ008	*Detecting the ramp*	Being able to detect the ramp.	Relation to RQ009. The car must be able to detect a ramp and stop before it can go any further.	X/Y/Z gyro axes should read 0	rotation distance over time
RQ009	Ascending/Descending the ramp	Being able to go up and down a ramp	Relation to RQ002 and RQ007. It must still follow the line on the ramp, while simultaneously having a speed that allows it to go up the ramp.	Angle of the Car	Angle Degrees And m/s
RQ010	Stopping the car	The car has to stop at the end of the track	Relation to RQ003, RQ005 and RQ008. When facing an obstacle or reaching the end of the track, the car must stop.	True or False	NONE

Component	*Technical requirement*
Battery	Must stay on for 30 minute minimum
Car Body	Must fit in a box of 400mm x 250mm x200mm
Obstacle Detection Sensor	Detect obstacles 5 cm away
Line Detection Sensor	Follow the 20mm white line which is the track and stop at the end of the track

To analyse the feasibility of the functional and technical requirements for a self-moving car as documented in the SR-document, we need to consider various factors such as motion, accuracy, mass, vibrations, power consumption, and others. Here are some details on each of these aspects:

1.Motion:

The motion requirements for a self-moving car include velocities, accelerations, motion profiles, and cycle time. The car should be able to move at various speeds and accelerations, and the motion profiles should be smooth and controlled. The cycle time, which is the time required for the car to

complete a task, should be optimized to ensure time restrictions. The feasibility of these requirements can be analysed using simulations and testing to ensure that the car can perform the necessary motions safely and effectively. The functions affiliated with motion are steering, acceleration and hill detection. These systems will be responsible for taking care of the vehicle's velocity, acceleration, and deceleration.

2.Accuracy:

The accuracy of a self-moving car is crucial to ensure that it can navigate according to the line correctly and without collisions to obstacles. The feasibility of these requirements can be tested by measuring the car's position accuracy and on different speeds and terrain types. The functions in charge of the vehicle's precision are the line detection function and the obstacle detection function, which prevent the automobile from straying off track or striking any obstructions. The automobile must be precise in order to avoid battery drain and have superior mobility.

3.Mass:

The mass of a self-moving car can impact its ability to move and manoeuvre, as well as its power consumption. Decomposition of the mass into individual components and analysis of their weights can help optimize the car's design to ensure that it meets the necessary requirements. The feasibility of these requirements can be tested using physical prototypes or simulations to ensure that the car can move and operate as required.

4.Vibrations:

Vibrations can affect the performance and accuracy of a self-moving car, so it's important to consider sensitivity, transfer functions, and response spectra when designing the car. This includes analysing the car's response to various types of vibrations, such as those from uneven terrain and high speed. The feasibility of these requirements can be tested using simulations or physical testing to ensure that the car can operate safely and effectively under different conditions.

5.Power consumption:

The power consumption of a self-moving car is important to ensure that it can operate efficiently and for extended periods of time. Decomposition of the power consumption into individual components can help identify areas for optimization, such as reducing internal losses. The feasibility of these requirements can be tested using simulations or physical testing to ensure that the car can operate for the necessary duration on its available power source.

In conclusion, analysing the feasibility of the functional and technical requirements for a selfmoving car involves considering various factors such as motion, accuracy, mass, vibrations, power consumption, and others. Testing and simulations can be used to ensure that the car can perform the necessary tasks safely and effectively, and to identify areas for optimization and improvement.

3. **Concept principles**

To ensure the selection of an appropriate concept for the vehicle, NexusTech must prioritize the quality assurances:

Safety is the most critical quality objective for a self-driving car project. The car must be designed and tested to ensure that it is safe.
A self-driving car must be reliable to ensure that it operates as intended in all its tests. This means that the car should be able to operate in different conditions such as avoiding obstacles, going up on a ramp or stop at the end of the track.

Durability, that means the car should be made of high-quality materials that can withstand being dropped or played with roughly.

A self-driving car should also be cost-effective to produce and sell. This means that it should be designed with materials and manufacturing processes that are efficient and cost-effective.

3.1 Description of alternative concepts

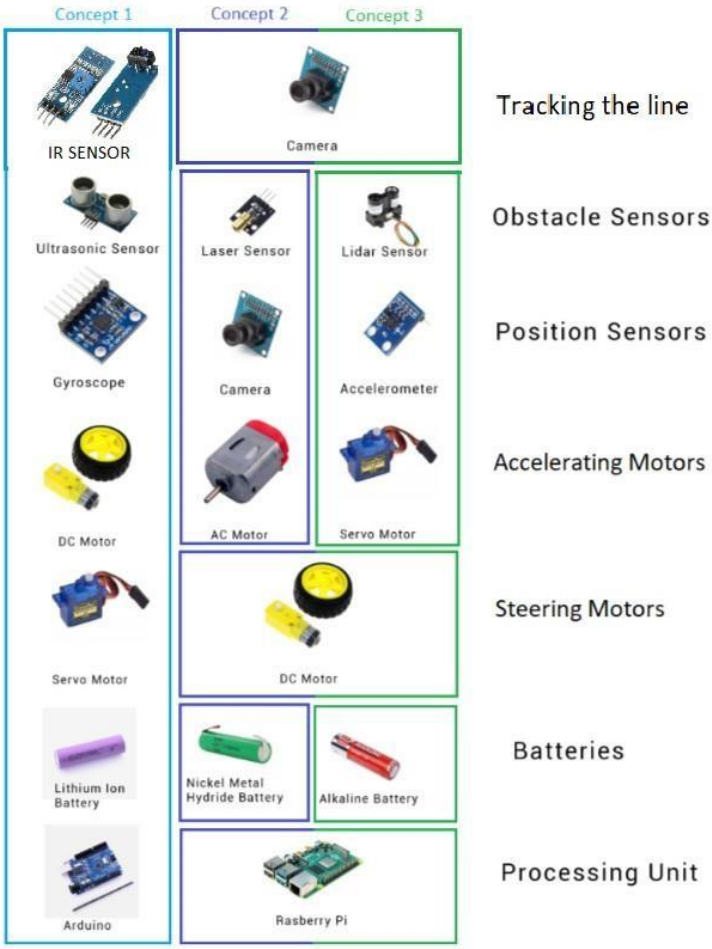


Fig 3.1 The choices in components

Picture is of all the type of components that were considered.

NexusTech have considered 2 or more components for each functionality of the system.

1. To detect the white line:

a) Infrared Sensor (Concept 1):

- which is an electronic device that measures and detects infrared radiation;
- no contact with the line is required for the sensor to detect it;
- easier to program;
- measuring accuracy affected by dust, smoke, background; - field of view and spot size may restrict sensor application.

b) Camera Module (Concept 2 and Concept 3):

- the image array is capable of operating at up to 30 frames per second;
- it is hard to interface because it has a large number of pins and jumbled wiring to carry out.

2. To detect the obstacles:

a) Ultrasonic Sensor (Concept 1):

- which are classified as proximity sensors. It measures the distance of a target object by sending soundwaves (which are converted into electrical signals);
- it has higher sensing distance compared to inductive/capacitive proximity sensor types;
- it provides good readings in sensing large sized objects with hard surfaces;
- it has more difficulties in reading reflections from soft, curved, thin and small objects.

b) Laser Sensor (Concept 2):

- which emits a laser beam and receives the reflection from it;
- can range up to several kilometers;
- easy to install;
- the laser sensor's measurement is very accurate;
- detects a wide range of materials;
- are more expensive than analog measuring devices; - can damage eyesight.

c) Lidar Sensor (Concept 3):

- which targets an object with a laser to determine the ranges and measures the time for the reflected light to return to the receiver;
- the sensor sends out laser pulses and receives them back in nanoseconds, making it possible to scan large areas in a fairly short period of time;
- offers incredibly accurate, consistent results;
- low cost;
- can be used day and night;
- may affect the human eye in cases where the beam is powerful;
- high operating costs in some applications (can be expensive when applied in smaller areas when collecting data).

3. To detect the hills:

a) Gyroscope (Concept 1):

- which is used for angular velocity measurements and navigation;
- it measures relative orientation on all three axes and all types of rotation;
- it is fast in operation;
- it doesn't measure linear motion in any direction or any static angle of orientation; - it is a more expensive alternative to navigation and tilt sensing applications.

b) Accelerometer (Concept 3):

- which is used to measure the vibration or acceleration of motion of a structure;
- it has high frequency response;
- it is available at lower cost due to advancement in MEMS (Micro-electromechanical systems); - it also can't measure around its own axis of movement, which is why the gyroscope is also involved to measure angular velocity.

c) Camera Module (Concept 2).

4. Accelerating Motor:

a) DC Motor (Concept 1):

- a rotary electrical motor that converts DC electrical energy into mechanical energy;
- they have high startup power;
- they have fast response times to starting, stopping, and acceleration;
- DC motors are typically more efficient and make better use of their input energy;
- high initial cost;

b) AC Motor (Concept 2):

- an electric motor with an alternating current to generate mechanical energy through magnetism; - design is easy & simple; - good power to weight ratio; - at low speeds, it won't operate;

c) Servo (Concept 3):

- an actuator that allows for precise control of angular or linear position, velocity and acceleration.
- high output power relative to motor size and weight;
- high efficiency;
- poor motor cooling;
- high speed operation is possible;

5. For processing:

a) Arduino board (Concept 1):

- which can read inputs and turn them into outputs;
- a multitude of libraries;
- inexpensive hardware; - not optimized for performance; - less memory storage capacity.

b) Raspberry Pi (Concept 2 and Concept 3):

- a minicomputer that is the size of a credit card.
- low power consumption;

- great for small tasks and limited goals;
- faster processor;
- not ideal for multitasking;

6. For supplying power:

a) Lithium (Concept 1):

- a type of rechargeable battery which uses lithium-ions to store energy;
- low maintenance;
- higher battery life;
- performance declines with time;
- highly fragile;

b) Nickel (Concept 2):

- another type of rechargeable battery;
- easy to recharge;
- available in a wide range of sizes and performance options;
- lower power density;

c) Alkaline (Concept 3):

- a popular type of disposable battery in the market.
- high energy density;
- cost less than other types of batteries;
- they store well - even after 2 years they hold up to 90% of their energy;
- high internal resistance reduces output; - they can explode when charging with faulty charger;

3.2 Comparison of concept principles

Morphologic diagram

#		Concept 1	Concept 2	Concept 3
1	Turn On/Off	By Computer	Switch	Switch
2	Power Source	Lithium	Nickel	Alkaline
3	Track the line	Infrared Sensor	Camera	Camera
4	Detect obstacles	Ultrasonic Sensor	Laser Sensor	Lidar Sensor
5	Detect hill	Gyroscope	Camera	Accelerometer
6	Accelerating Motor	DC Motor	AC Motor	Servo
7	Steering Motor	Servo	DC Motor	DC Motor

8	*Processing Platform*	*Arduino*	*Raspberry Pi*	*Raspberry Pi*

Table is of all the different parts of all the concepts that were considered.

Functionality	Ideas (from concepts)	Costeffectiveness	Lifespan	Performance	Accuracy
Supplying power to the system	1.Lithium	More expensive than Alkaline batteries and Nickel batteries	Lithium batteries last longer (life cycle of 5 years) compared to Nickel and Alkaline batteries	Good charging performance at cool temperature; Stands up well to repeated charging and discharging. Overall outperforms the Nickel battery.	-
	2.Nickel	Cheaper than Lithium batteries, more expensive than Alkaline.	Nickel batteries don't last as much as Lithium batteries do, but they last 24 times longer than Alkaline batteries	Better capability than Alkaline batteries. Easy to recharge. Rapid recharge capability.	-
	3.Alkaline	Cheaper than Lithium batteries and Nickel batteries	Alkaline don't last as much as Nickel or Lithium batteries	Can operate at very low temperatures. It provides good, longterm power.	-

Following the line	1. Infrared Sensor	Cheap	7-8 years	Practically no speed limit for IR detection	±1°C Also, an infrared camera (for example) can catch what a normal camera can't.
	2/3. Camera	The Arduino camera module is usually cheap	Generally, the lifespan of a camera sensor is 7-10 years	-	-

Detecting the obstacles	1.Ultrasonic Sensor	(Specifically for Arduino) Can cost around 4-5 EUR or less	- (could not find info)	Fast The ultrasonic sensor will measure the distance of the object every 1 second. It sends an ultrasonic pulse out at 40kHz	Most ultrasonic sensors can achieve between 1%3% accuracy. The Arduino Ultrasonic Sensor works at about 2cm to 400 cm.
	2.Laser Sensor	Can cost around 10 EUR or less	- (could not find info)	Speeds up to 50.000 Hz	Great accuracy
	3.Lidar Sensor	Much more expensive	Nearly 10 years	Over 670 miles per hour	Very accurate 15 cm vertically 40 cm horizontally
Detecting the hill	1.Gyroscope	Around 5-7 EUR or less	- (could not find info)	The gyro wheel rotates at 15000-3000 revolutions per minute	The angular velocity measurement is rounded to the nearest degree per second

	2.Camera	The Arduino camera module is usually cheap	Generally, the lifespan of a camera sensor is 7-10 years	-	-
	3.Accelerometer	Range of 3-10 EUR	- (could not find info)	Operation at higher temperatures	Good resolution
Accelerating	1.DC Motor	Cheaper than a Servo motor and more expensive than the AC motor	About 20000 to 50000 hours (about 5 and a half years) for general DC motors	A DC motor's speed is directly proportional to the input voltage. The higher the voltage is, the faster the motor is. The lower the	Control accuracy can reach 0.1 mm (the moving parts can stop almost anywhere that is desired)

				voltage is, the slower the motor is.	
	2.AC Motor	Cheaper than a DC motor and the Servo motor	AC motors have a long service life	AC motors are generally considered to be more powerful than DC motors (although DC motors can use their input energy are more efficient)	-

	3.Servo	More expensive than the DC motor or the AC motor	It is estimated that under ideal conditions, it can last 20+ years. Under extreme conditions, it can last less than 1 year. However, based on the bearings, it's estimated that it can last around 20000 or 30000 hours (about 3 and a half years).	It can provide 2-3 times the speed of a typical stepper motor (which is also known as a brushless DC motor).	to about +- 0.02 degrees It's the most precise motor.
Steering	1.Servo	The Servo Motor is more expensive than the DC motor	It is estimated that under ideal conditions, it can last 20+ years. Under extreme conditions, it can last less than 1 year.	It can provide 2-3 times the speed of a typical stepper motor (which is also known as a brushless DC motor).	to about +- 0.02 degrees
			However, based on its bearings, it's estimated that it can last around 20000 or 30000 hours (about 3 and a half years).		

	2/3. DC Motor	The DC motor is less expensive than the Servo motor	About 20000 to 50000 hours (about 5 and a half years) for general DC motors	A DC motor's speed is directly proportional to the input voltage. The higher the voltage is, the faster the motor is. The lower the voltage is, the slower the motor is.	Control accuracy can reach 0.1 mm (the moving parts can stop almost anywhere you want)
Processing	1.Arduino	Arduino is cheaper than Raspberry Pi (costs around 10-20 EUR)	2-3 years (hot environment) 10-15 years (cold environment)	Its clock speed is 16 MHz.	-
	2/3. Raspberry Pi	Raspberry Pi is more expensive than Arduino (costs around 35-40 EUR)	Around 7-10 years	Its clock speed is around 1.2 GHz	-

NexusTech has decided to make comparisons between the components based on their lifespan, cost, performance and accuracy; although there was a problem with information gathering in certain places.

For Supplying power to the system, lithium batteries would be the best choice. They are rechargeable batteries that don't sustain much damage from repeated charges/discharges and have longer lifespans than Alkaline or Nickel batteries. The main disadvantage is that it is also the most expensive option.

For Following the line, the IR sensor would be the best choice. They can last around 8 years and can catch wavelengths that are invisible to a normal camera and to the human eye.

For Detecting the obstacle, the Ultrasonic sensor would be the more suitable choice. It has the capability to measure the distance of the object every 1 second. The measurements are precise, and the sensor doesn't have to touch the respective object. It is also cheaper compared to laser sensors, and especially cheaper compared to Lidar sensors (which cost around 30 EUR and beyond).

For Detecting the hill, the gyroscope would be a more suitable choice. It is relatively cheap, it is fast in operation, it can measure relative orientation on all the 3 axes, all types of rotation (not movement) and

angular velocity. It can be used in combination with the accelerometer, which measures linear acceleration.

For Accelerating, the DC motor would be the preferable choice. It is cheaper than a Servo motor, but more expensive than the AC motor. It can last up to 5 years and the motor speed depends on the amount of voltage. The speed of the device can also be controlled in a smooth manner for an extended range.

For Steering, the Servo motor would be the better choice. It's more expensive than the DC motor, but it can function for much longer, has a higher speed and can be used for the precise control of the steering position.

For the Processing part, it would be a better choice to use Arduino, since NexusTech has worked with it before, so prior knowledge can be used; it is also a significantly cheaper option, can last longer under the best conditions and it is generally considered good for interfacing sensors and controlling LEDs and motors. Raspberry Pi is more suitable for developing software applications.

3. 3 Choice of most promising concept principle

Each conceptual idea has undergone comprehensive research, and the weighted criteria have been carefully evaluated based on their precision, speed, and cost-effectiveness.

- Given that the task at hand involves guiding a car to follow a white line, avoid obstacles, and stopping at the right moment, accuracy is of utmost importance, and thus it represents 50% of the project's criteria for decision-making.
- While speed may not be the top criterion on the list, it still holds significant importance for NexusTech's goal of winning the final race, and therefore represents 30% of the project's criteria.
- Although cost is not as critical as the other two criteria, it still carries significant weight in the decision-making process as NexusTech operates within a limited budget. Therefore, cost represents 20% of the project's criteria, and must be taken into careful consideration.

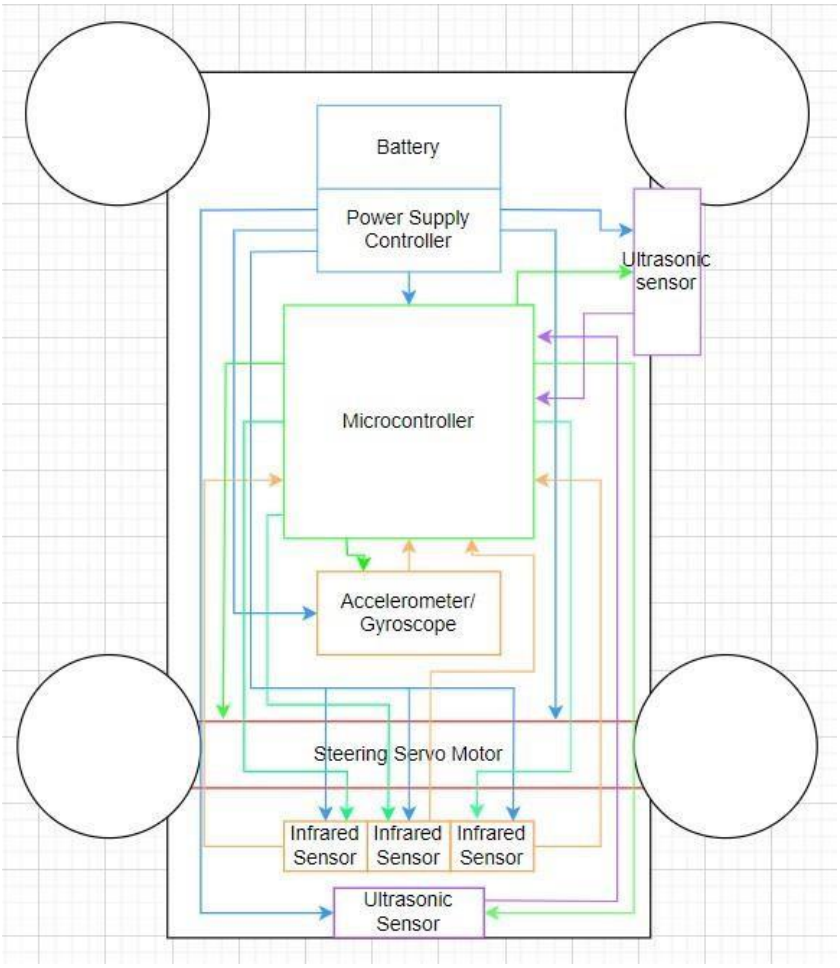
Using these criteria, NexusTech have evaluated all the concepts, and these are the findings:

- Concept one was found to be the most accurate and fastest, although not the most costefficient.
- Concept two, on the other hand, was identified as the most cost-efficient, although less accurate and slower than Concept one.
- Finally, concept three was also highly accurate, but slower than Concept one, and was identified as the most cost-efficient option.

4. **Elaboration of chosen principle to functional design**

4.1 Overview of Functional Design

The picture bellow is an easy to understate diagram that is meant to represent a general outline of how the final product it going to be arranged and connected. The lines represent the input and output, from sensors to microcontroller to microcontroller to servo motor, and even all the places the power from the battery goes. Further along the document is the design but divided into multiple parts each demonstrating a different aspect of the design.



Picture is a functional design that shows all the connection both electric and electronic

Fig 4.1 Car representation (electric and electronic connections)

4.2 Elaboration of functional design

4.2.1 Mechanical

Creating a mechanical layout of a self-moving car, Nexus tech starts by identifying the major components of the vehicle, such as the frame, suspension system, steering system and wheels. The company would then position these components relative to each other and define their interconnections, considering factors such as the vehicle's centre of gravity, weight distribution, and desired performance characteristics.

Once the company has a basic layout then they need to evaluate the mass, strength, and stiffness of each component, as well as their eigenfrequencies (the natural frequencies at which the components vibrate). This motion can help the company optimize the design for safety, durability, and performance, and ensure that the vehicle operates smoothly and efficiently.

After that, the company would consider the mechanisms and guidance systems that control the vehicle's motion, such as the steering and suspension systems. Nexus tech would evaluate the stiffness of these components and calculate the disturbance forces that they are likely to encounter during operation by putting the car design on trial and test. This information can help them refine the design to ensure that the vehicle handles well and is stable under a variety of driving conditions.

Finally, Nexus tech would dimension the drives and transmissions that power the vehicle and evaluate the flexibilities of these components. This information can help them ensure that the vehicle has the necessary power and torque to move efficiently.

Throughout the design process, Nexus tech would use a combination of analytical calculations, computer simulations, and physical testing to refine the design and ensure that it meets the required specifications.

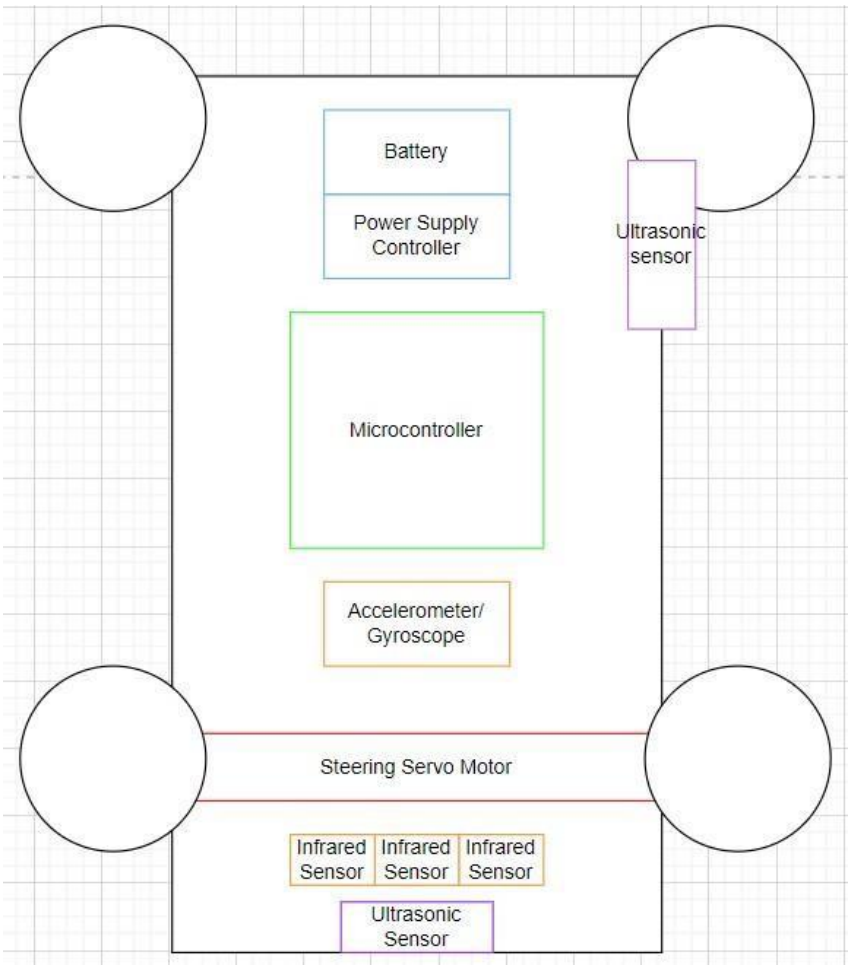


Fig 4.2.1 Car representation (mechanical aspect)

It is the functional design when looking only at the mechanical part.

4.2.2 Electrical

The electrical diagram shows all the electric connection of the automated car. It starts at the 12V power supply then it is managed by a power manager to ensure that every part gets the exact amount of energy it need and it not being too low for it not to work or too high for the device that will probably fry. After that it goes to the microcontroller, sensors, motors, and the steering motor.

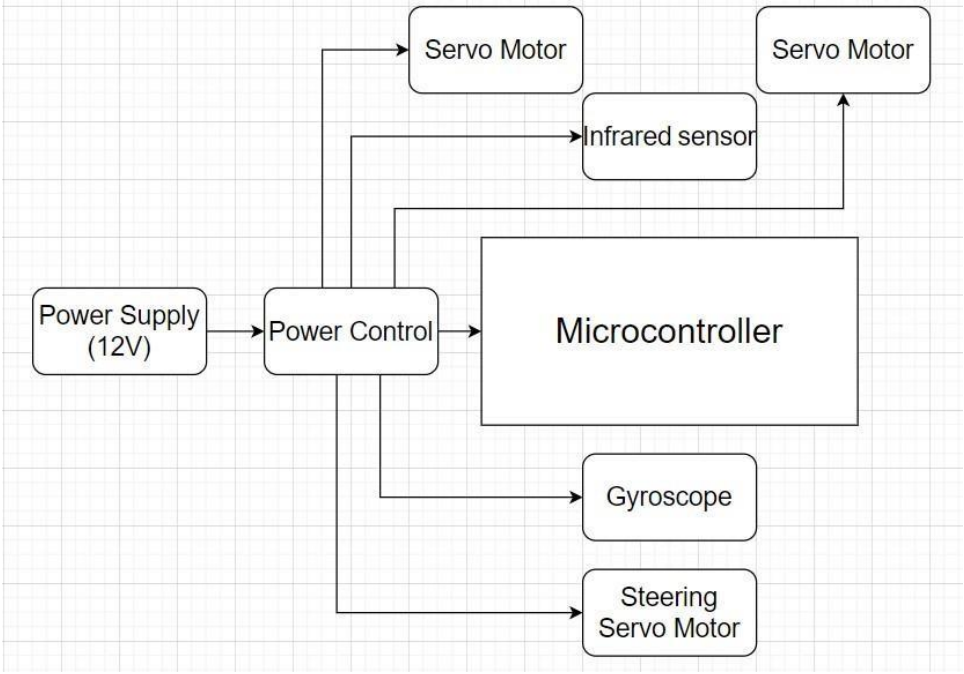


Fig 4.2.2 Car representation (only electrical part)

It is the functional design when looking only at the electrical part.

4.2.3 Electronic

The electronic diagram provides a comprehensive map of all the signal connections within the system. While most signals are initiated by the microcontroller to control the vehicle's operations, the sensors on the car send signals to the microcontroller to process environmental data.

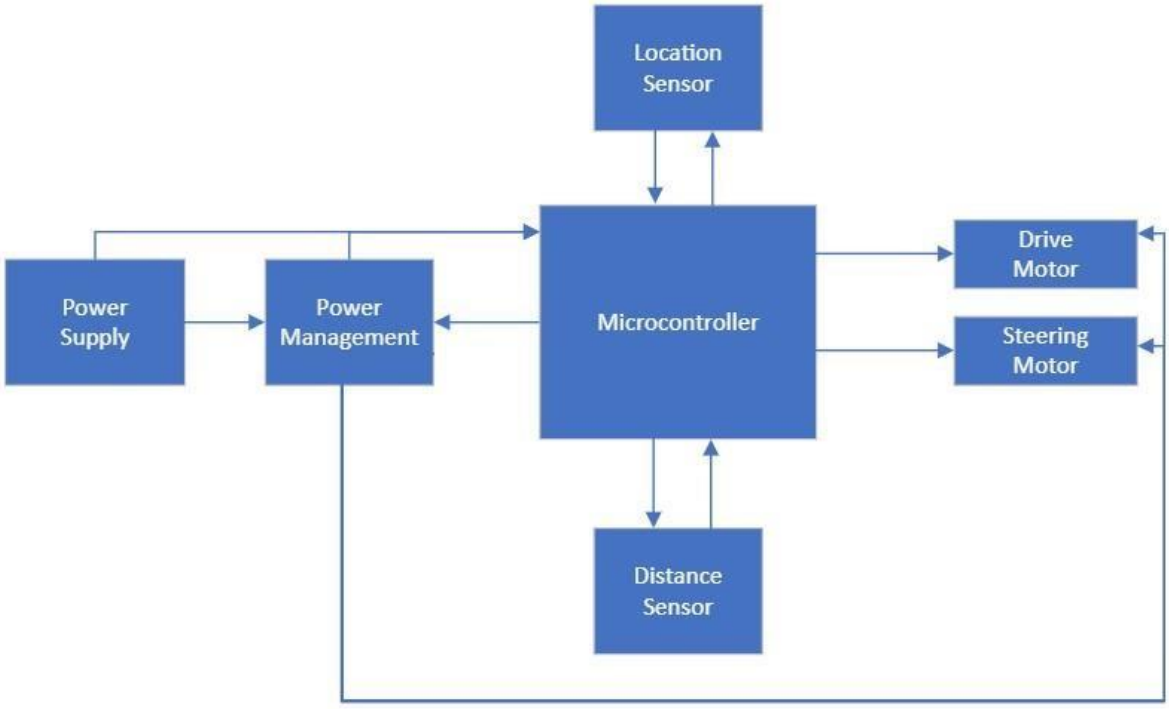


Fig 4.2.3 Car representation (only electronic aspect)

It is the functional design when looking only at the electronic part.

4.2.4 Software

The flowchart below describes the code for the car's functions, from turning it on, to finishing the race.

The code will start by checking to ensure that all sensors are working correctly, before moving on to the car's main function loop. In the main loop, it will consistently check to see if it is on the line and if there are any obstacles, then react accordingly.

This will then loop until the program reaches the end, or a warning is returned to the computer

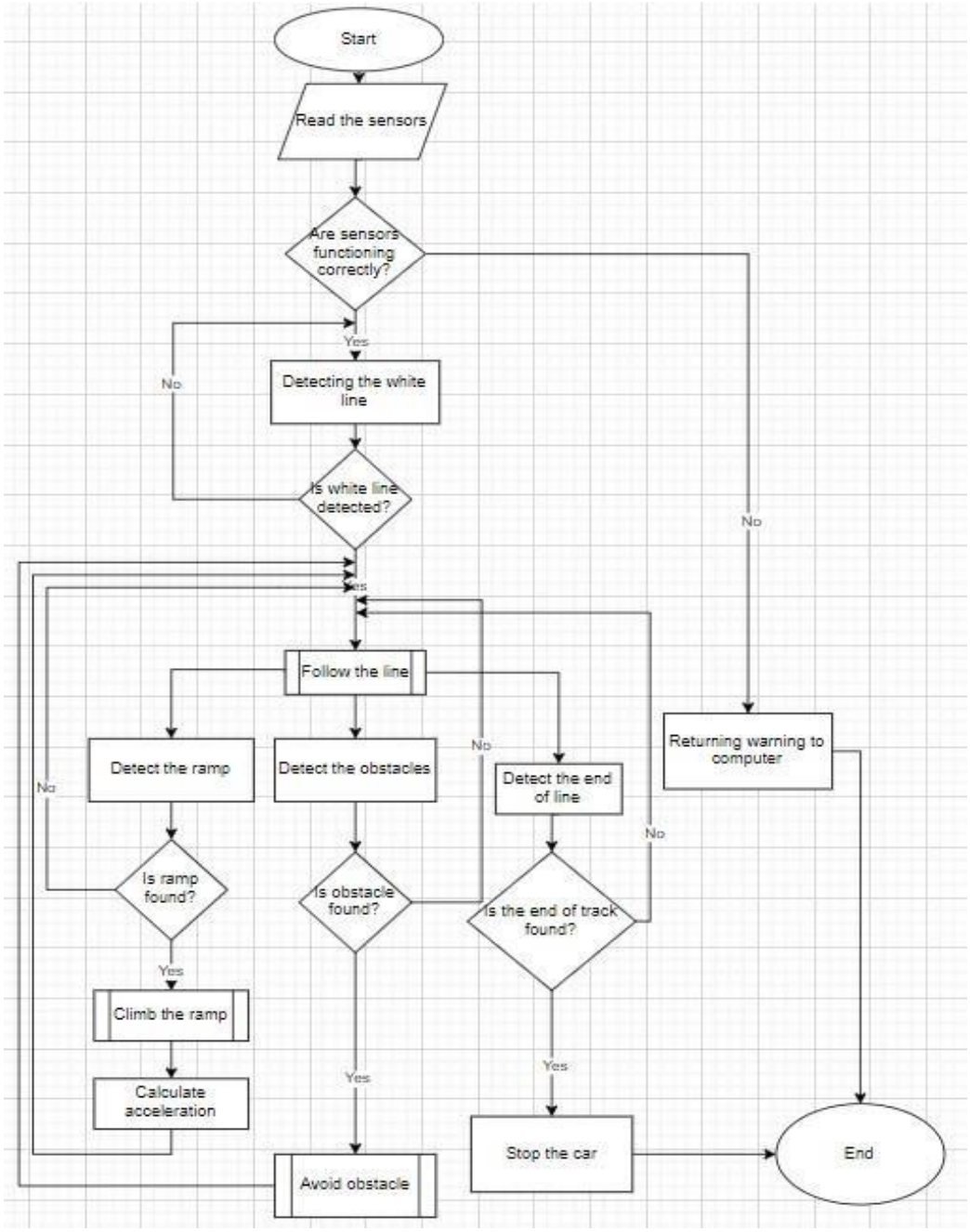


Fig 4.2.4 High-Level Flowchart

It is the functional design when looking only at the software part.

4.3 Functional Design intergration

Based on the evaluation criteria, the chosen concept for the vehicle is concept one, which was found to be the most accurate and fastest among the options. While it may not be the most costefficient,

the prioritization of precision and speed aligns with the primary goal of guiding the car to follow a white line, avoid obstacles, and stop at the right moment.

Considering the importance of safety, speed, accuracy, cost, and maintenance, concept one was deemed the best fit for NexusTech's requirements and goals. The team will continue to monitor and optimize the chosen concept to ensure it meets all necessary standards and performs at its best potential.

5. **Functional Design Tests**

Within the scope of the Functional Design of this report, only the Unit Test cases and results will be provided for the following functions:

1. Detect the line (sensor);
2. Detect an obstacle (sensor); 3. Detect the ramp (sensor);
   4. Control the speed.

Test Case No.	Func. Req. ID	Func. Req. Being tested	Components	Description	Expected result	Result obtained
IN0101	RQ002	Detecting the line	IR sensor	Testing the IR sensor with the Serial Monitor, making it detect black and white surfaces.	The IR sensor should be able to give different outputs depending on the surface it detects.	The test was successful. The sensor detected both black and white surfaces.
IN0102	*RQ005*	Detecting the obstacle	Ultrasonic sensor	Testing the Ultrasonic Sensor with the Serial Monitor, to determine whether it can determine an object at a small distance. A phone was used for giving the distance between the sensor and the	The Ultrasonic Sensor should be able to detect the object from a small distance at all times.	The test was semisuccessful, since the component wasn't operating as efficiently as expected (constantly having an output of 0 multiple times even

				back of a breadboard.		while facing the object).

IN0103	*RQ005*	Detecting the obstacle	Ultrasonic sensor	Conducting the same experiment as in IN0102; except a different Ultrasonic sensor is used.	The Ultrasonic Sensor should be able to detect the object from a small distance at all times.	The test was fully successful, the Serial Monitor displaying the distance between the object and the sensor at all times.
IN0104	*RQ007*	Controlling the speed	Servo	Testing the efficiency of the Servo by constantly changing the speed of the car.	The car should be able to accelerate or decelerate depending on the given input.	The test was successful, the car being able to attain different speeds depending on the given inputs.
IN0105	RQ008	Detecting the ramp	GyroscopeAccelerometer	It's part of the process of finding a way to detect the ramp, but it's only about finding the x, y, z coordinates.	The axis values should be calculated at any given point with good accuracy.	The test was successful, receiving different values for each axis at any given point.

IN0106	*RQ005*	Detecting the obstacle	Ultrasonic sensor	Conducting the same experiment as in IN0102 and IN0103; except a different Ultrasonic sensor and a different code is used. This represents a more definitive version of testing the Ultrasonic Sensor's capability of	The same expectations are to be met, except a more efficient algorithm is being used.	The test was successful; not only that, but also the distance between the obstacle and the sensor, at which the sensor is supposed to give a different output, can be
				detecting the obstacle.		decreased or increased.

For the snippets of code, the Appendix can be checked.

4. Technical design

4.1 Introduction

The technical design is thorough documentation consisting of detailed schematics and low-level flowcharts design. It will discuss the Electrical and Mechanical aspects of the project, as well as all the software in the form of flowcharts. The team will create mechanical components for the NexusTech automobile shell, as well as electrical and electronic schematics and a low-level flowchart for software development. They will also supply information on the pieces that will be employed in this project, as well as the necessary calculations.

4.2 Mechanical Design

The motion system is already integrated in the car's chassis for mechanical design, so NexusTech didn't need to make any adjustments to the steering system, which is controlled by the servo motor and the dc motor at the back.

However, the engineers devised a design for the body of the car:

Cover Design

The design inspiration for NexusTech's self-driving car cover is the iconic Mustang from 1965.

Fig 4.2 Design inspiration

Using the software Blender, NexusTech's team have crafted a custom cover for the self-driving car that takes inspiration from classic car designs. Drawing from our reference materials and the dimensions of the given Ishima Booster electro buggy RTR chassis, the company have created a cover that perfectly integrates with the car's body, while also providing enough room for the necessary components required for the car's advanced functionality

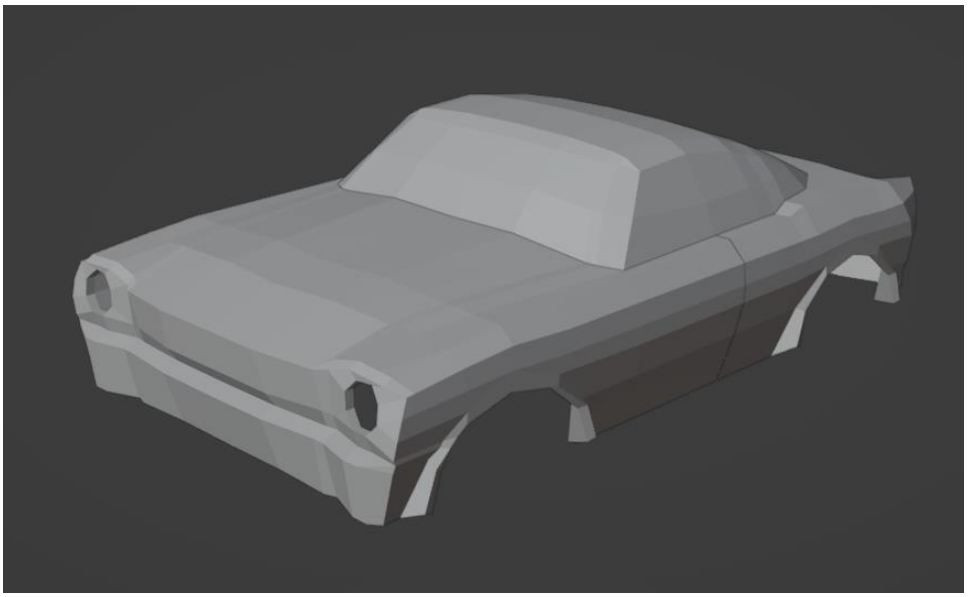


Fig 4.2.1 Solid view

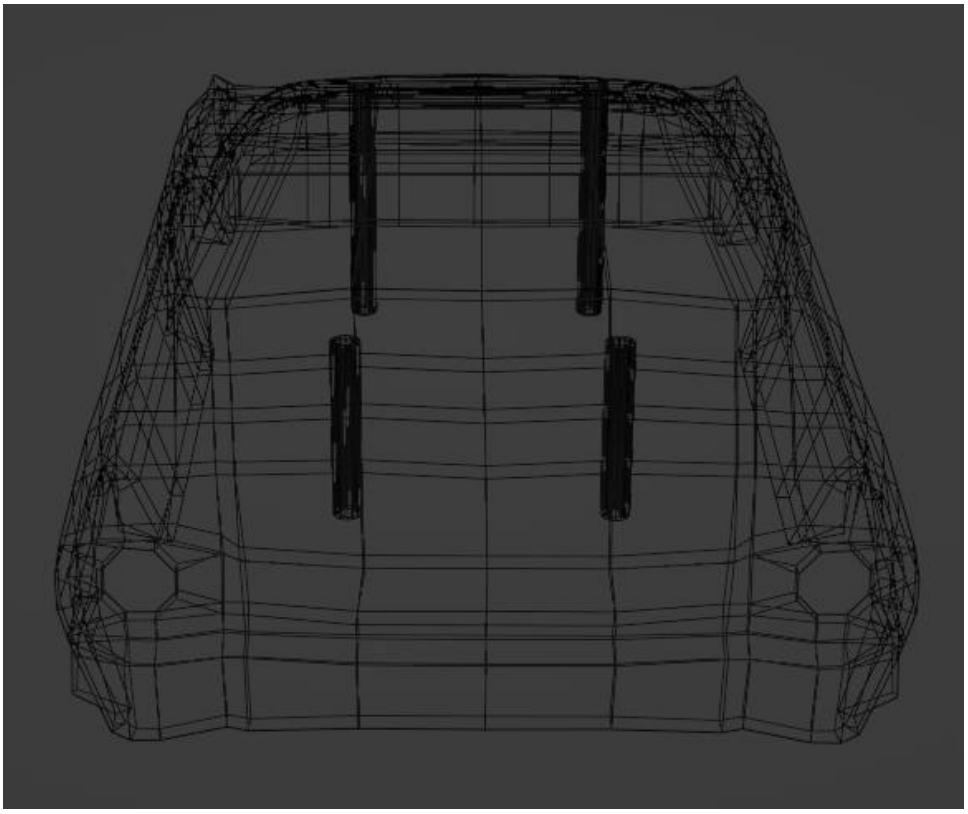


Fig 4.2.2 Wireframe View

Integration

The company uses advanced techniques to ensure the car cover fits perfectly with the Ishima Booster electro buggy RTR chassis, improving the car's performance and safety. All necessary components will be carefully positioned within the cover to ensure the car's advanced functions operate

efficiently. NexusTech's goal is to create a self-driving car that looks great, performs exceptionally, and prioritizes safety.

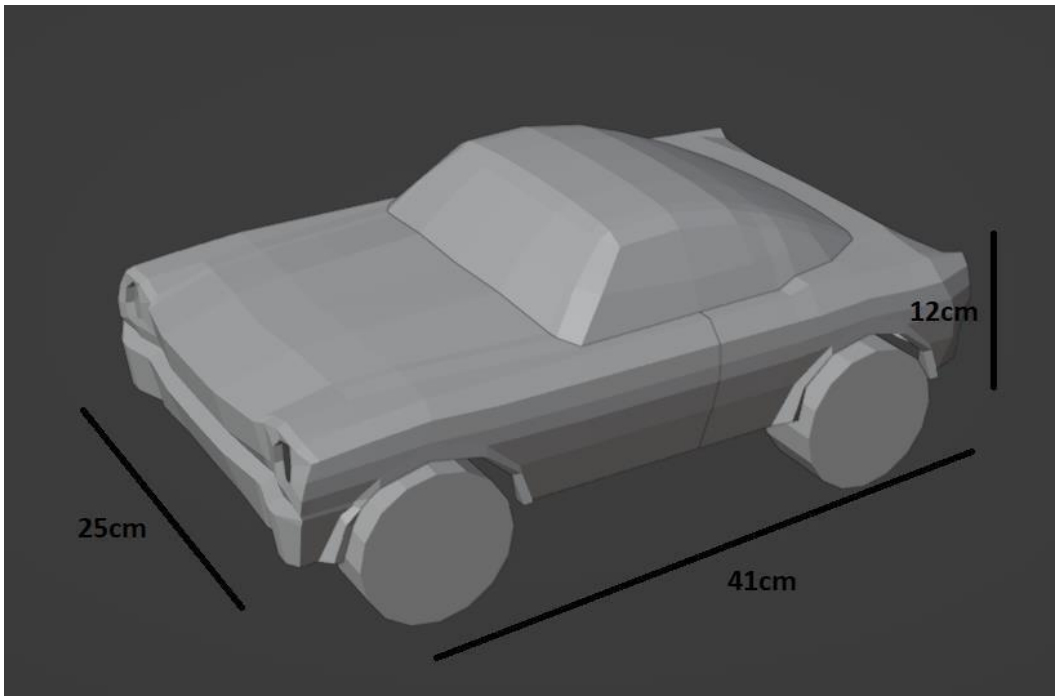


Fig 4.2.3 Measurements

NexusTech will use extended cylinders to attach the cover to the chassis with small screws. This method ensures a secure and stable connection, while also keeping the cover at a safe distance from the chassis to prevent damage.

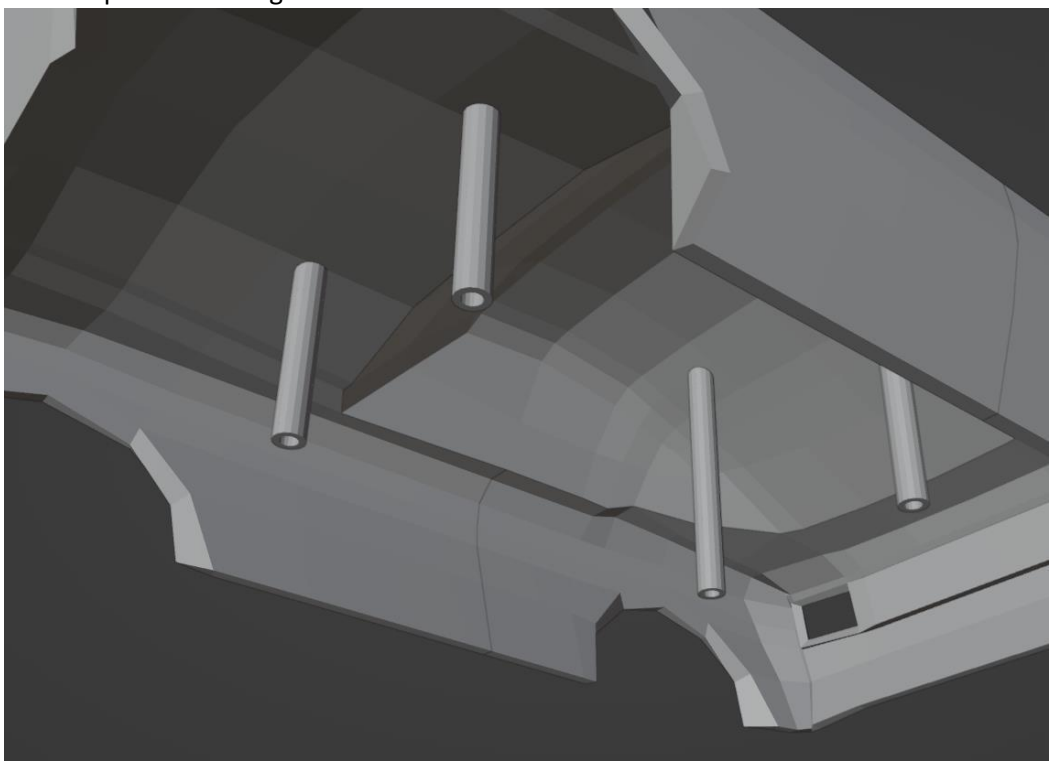
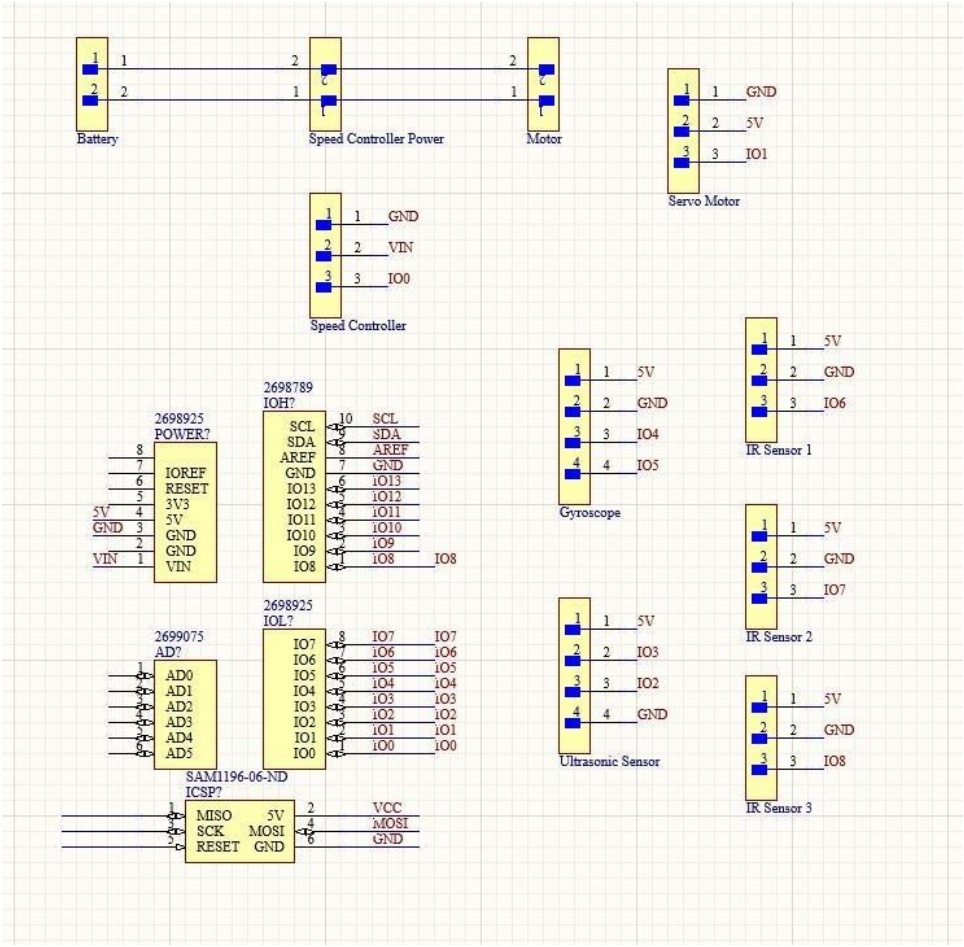


Fig 4.2.4

4.3 Electrical Design

4.3.1 Circuit Schematic



4.3.2 Component List

Speed controller:

H-bridege (HW-039) (Fig 4.3.1)

The Speed Controller H-bridge (HW-039) regulates motor speed and direction. It makes use of an H-bridge circuit with four switches for flexible control. It works with DC motors and is frequently used in robotics and automation projects. Microcontroller or similar device input signals are converted into motor control operations. It provides flexibility and precision control, but it must be matched to the voltage and current needs of the motor to avoid damage. The stakeholders had previously selected these components for us.

Motor:

Brushed Motor 15T (Fig 4.3.2)

The most prevalent are permanent magnet brushed DC motors. The stator field in these motors is generated by permanent magnets. They are commonly used in applications that need fractional horsepower. The stakeholders had previously selected these components for us.

Battery:

NiMH Battery(nickel-metal hydride) 7.2V (Fig 4.3.3)

The 7.2V NiMH Battery (nickel-metal hydride) is a rechargeable battery that is extensively used in electrical devices. It works at 7.2V and provides a dependable power supply. When compared to standard Ni-Cd batteries, NiMH batteries have a better energy density. They are less harmful to the environment and have a lesser risk of memory impact. They are suited for applications such as remotecontrolled cars, cordless power tools, and portable gadgets because to their 7.2V rating. The stakeholders had previously selected these components for us.

IR-Sensor:

Tcrt5000 (Fig 4.3.4)

The TCRT5000 IR sensor is a small and multifunctional sensor module that can detect obstacles and detect proximity. It is made up of an infrared transmitter and receiver combined into a single package. NexusTech uses three of these components for White Line Detection. The sensor works by producing infrared light and measuring the intensity of the reflected light. It is frequently utilized in applications such as line-following robots, obstacle detection systems, and item counting. The TCRT5000 sensor has a fast reaction time and a tiny form size, making it ideal for use in electrical applications. It is frequently used in conjunction with microcontrollers or Arduino boards for precise control and automation.

Ultrasonic Sensor:

HC-SR04 (Fig 4.3.5)

The HC-SR04 Ultrasonic Sensor is a commonly used and popular distance measurement module. It measures the distance between the sensor and an item using ultrasonic waves. It is made up of an ultrasonic transmitter and receiver. The sensor generates ultrasonic pulses and monitors the time it takes for the pulses to return after colliding with an item. The HC-SR04 sensor is widely utilized in robotics, security systems, and automation applications because to its excellent precision and dependability. It is Arduino and other microcontroller compatible, making it simple to integrate into a variety of applications. NexusTech uses this component for Obstacle Evasion.

Gyroscope/Accelerometer:

MPU-6050 (Fig 4.3.6)

The MPU-6050 is a gyroscope and accelerometer combination module that is often used for motion detection and orientation tracking. It combines a three-axis gyroscope and a three-axis accelerometer into a single chip. The accelerometer measures linear acceleration while the gyroscope measures angular velocity. The MPU-6050 has great precision and sensitivity, making it suitable for robots, drones, and motion-controlled systems. It may offer real-time rotational and linear motion data,

enabling precise motion monitoring and control. To provide motion-based functionality, the module is frequently used in conjunction with microcontrollers such as Arduino. NexusTech uses this component for Ramp Detection and Acceleration.

Servo Motor:

SG90 (Fig 4.3.7)

The SG90 servo motor is a small and light motor that is widely utilized in robotics and automation projects. It is a little motor that can rotate between 0 and 180 degrees. The SG90 servo motor runs at a low voltage and uses little current. It provides accurate position control, making it ideal for robotic arms, remote-controlled vehicles, and camera gimbals. The SG90 servo motor is compatible with most microcontrollers and is simple to operate using PWM signals. NexusTech uses this component for Wheel Steering.

Microcontroller:

Arduino Uno (Fig 4.3.8)

The Arduino Uno includes several digital and analog input/output pins for connecting sensors, actuators, and other electrical components. NexusTech uses this component for applying code to the components mentioned previously.

Fig 4.3.1  Fig 4.3.2   Fig 4.3.3

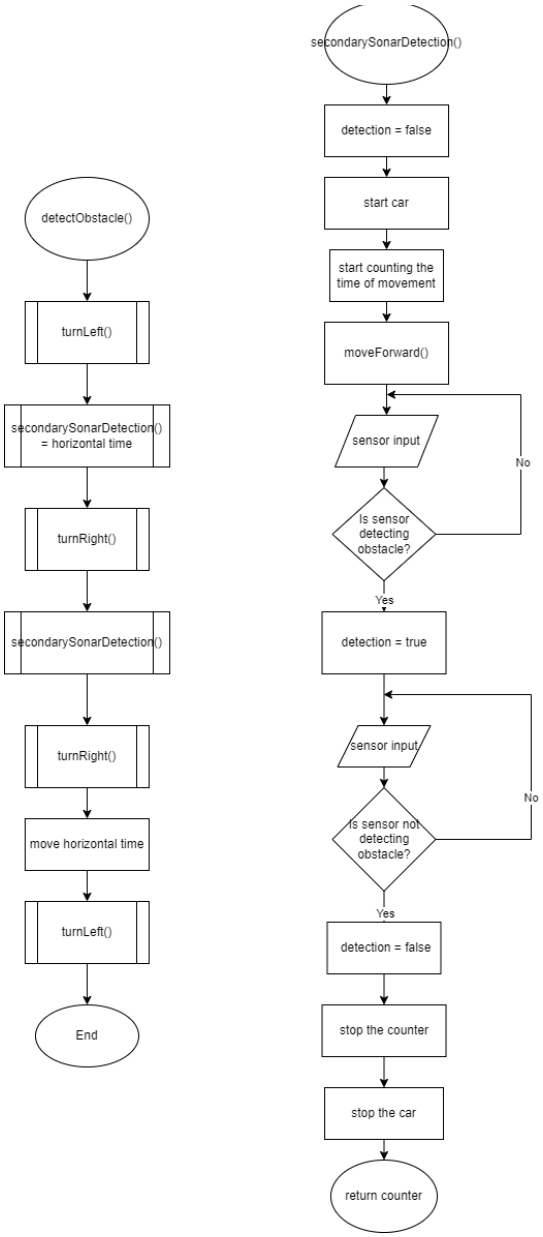
Fig    4.3.4Fig 4.3.5Fig 4.3.6

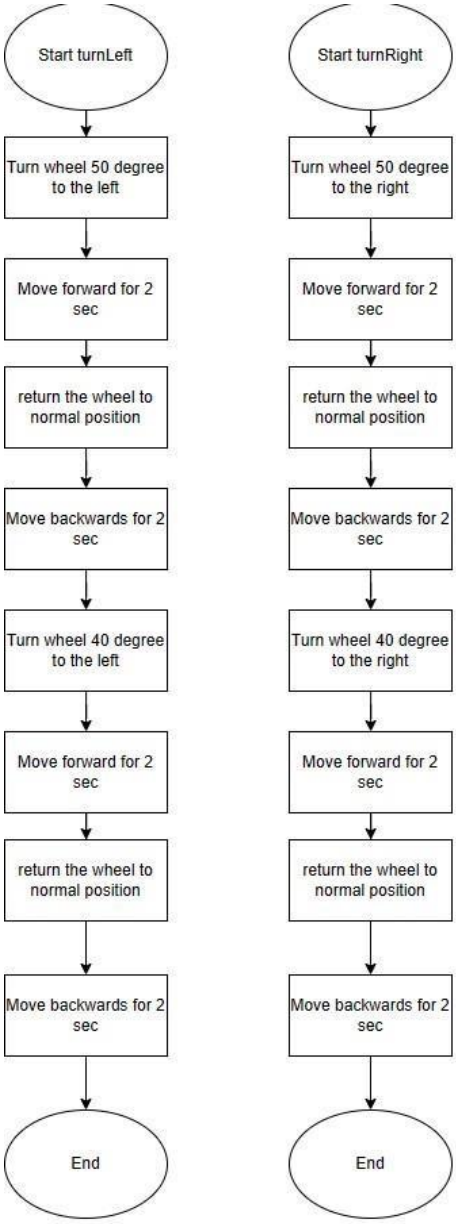
 

Fig 4.3.7Fig 4.3.8)

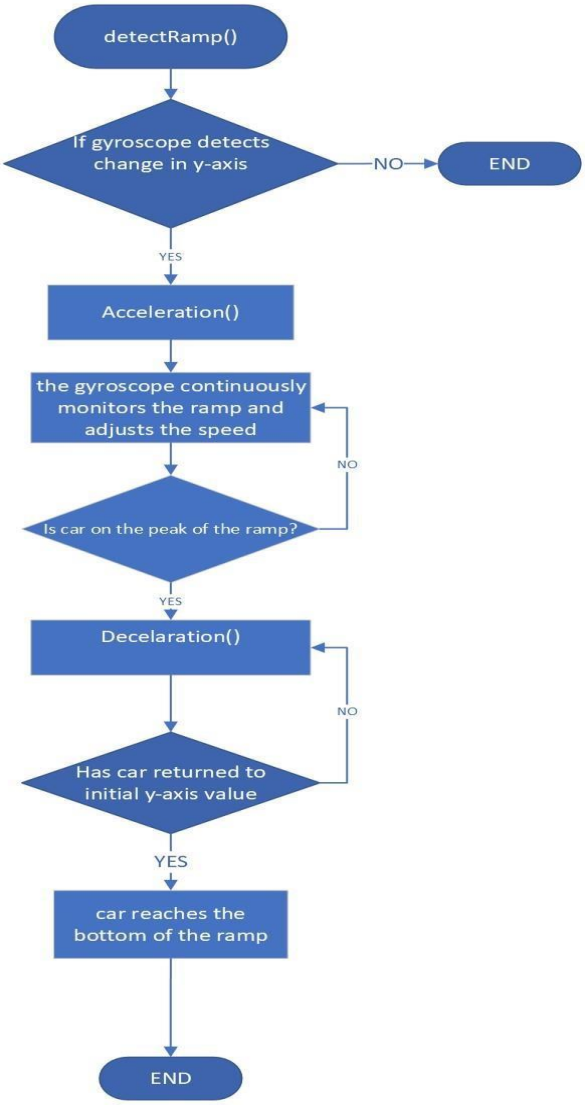
4.4 Software (Low-Level Flowcharts) 4.4.1

Obstacle Evasion

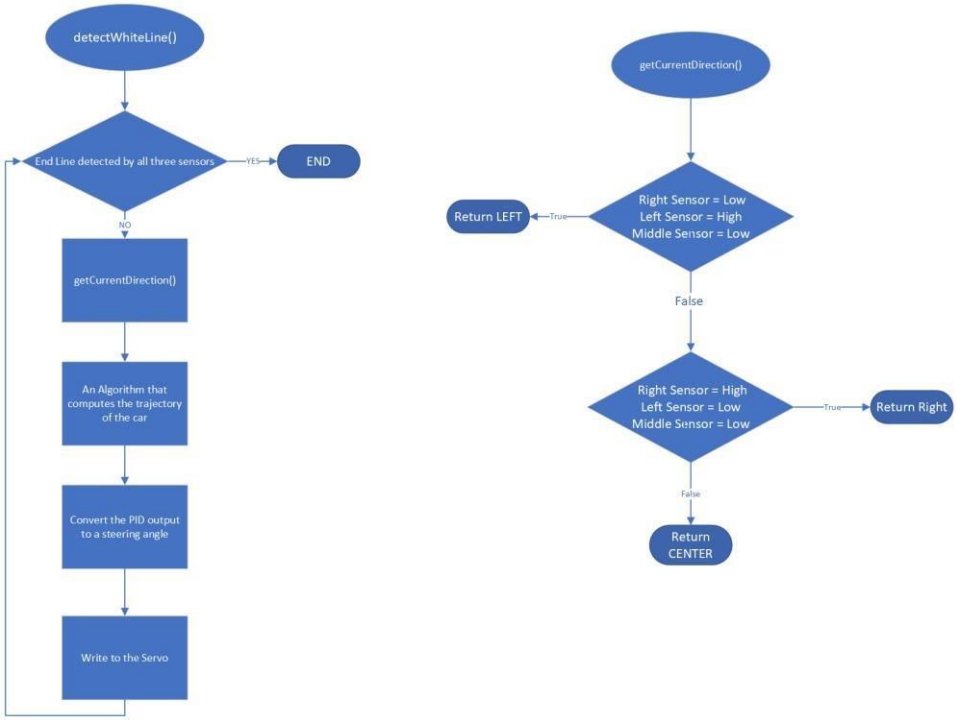

detectObstacle()
turnLeft()
secondarySonarDetection() = horizontal time
turnRight()
secondarySonarDetection()
turnRight()
move horizontal time
turnLeft()
End
secondarySonarDetection()
detection = false
start car
start counting the time of movement
moveForward()
sensor input
No
Is sensor detecting obstacle?
Yes
detection = true
sensor input
No
Is sensor not detecting obstacle?
Yes
detection = false
stop the counter
stop the car
return counter



4.4.2 Ramp Detection



4.4.3 White Line Detection



The codes for the respective flowcharts can be found in the appendix.

5. Products

NexusTech has created a small self-driving car that can do some amazing things. It can follow a white line, avoid obstacles, and stop when it's supposed to. To make all of this possible, the car has different parts that work together.

In order to detect the white line, NexusTech has implemented infrared sensors. These sensors are capable of measuring and detecting infrared radiation without requiring physical contact with the line. This technology allows for accurate detection and tracking of the line's position.

For obstacle detection, NexusTech has utilized ultrasonic sensors. These sensors emit soundwaves to measure the distance between the car and target objects, providing reliable readings for large-sized objects with hard surfaces. This enables the car to navigate its surroundings effectively and avoid collisions.

To detect ramps, NexusTech has incorporated gyroscope sensors into the car's design. These sensors are designed to measure orientation and angular velocity on all three axes. By leveraging this data, the car can effectively navigate ramps and adjust its speed accordingly.

For the car's movement, a DC motor has been chosen. This type of motor converts electrical energy into mechanical energy, delivering high startup power and rapid response times. The DC motor facilitates the car's acceleration and ensures efficient propulsion.

To handle the processing tasks and act as the car's central control unit, NexusTech has opted for the Arduino Uno. This microcontroller board enables the interpretation of inputs and the generation of

appropriate outputs. It offers a wide range of libraries and a cost-effective solution for efficient processing.

To power the entire system, NexusTech has selected lithium batteries as the primary power source. These rechargeable batteries are renowned for their extended lifespan and resilience against damage caused by repeated charging and discharging. By employing lithium batteries, the car can operate reliably and sustainably.

Overall, NexusTech's miniature self-driving car demonstrates the successful integration of these components to achieve remarkable functionality. The careful selection and implementation of these technologies contribute to the car's ability to follow lines, avoid obstacles, and come to a controlled stop.

6. Conclusion and Recommendations

Conclusion

In conclusion, the NexusTech team has successfully designed and built a self-driving car that can navigate autonomously, avoiding obstacles and following a white line. The project demonstrates the team's dedication and problem-solving abilities. Throughout the project, NexusTech carefully analyzed client requirements and identified the main challenges. They developed multiple concept ideas and selected the best components based on safety, reliability, durability, and cost-effectiveness. By integrating infrared sensors for line detection, ultrasonic sensors for obstacle avoidance, and gyroscope/accelerometer for hill detection, NexusTech created a car that performs with impressive accuracy. The use of DC motors for acceleration and servo motors for steering ensures smooth and controlled movements. The selection of Arduino for processing and lithium batteries as the power source further contributes to the car's efficiency and reliability. These choices demonstrate NexusTech's expertise in selecting the right technologies for optimal performance.

About power consumption

The HW-039 motor driver board is a compact module that supports a wide input voltage range of 6 to 27V DC. It utilizes a dual BTS7960 H Bridge configuration for motor control and can handle a peak current of 43 Amps. With PWM capability of up to 25 kHz, it offers precise speed control and supports control input levels ranging from 3.3V to 5V. The board can be operated in either PWM or level control mode, with a working duty cycle of 0% to 100%. Additionally, it incorporates safety features such as overvoltage lockout and under-voltage shutdown. Brushed Motor 15T is connected directly to the HW039. Sensors:

- 3 IR sensors: these sensors typically consume very low power, usually in the range of a few mA each.
- Ultra Sonic sensor: consume very low power
- Gyroscope sensor: consume very low power, usually in the range of a few mA

The Arduino Uno itself consumes around 50-100 mA when active and running a program.

To calculate the overall capacity of the battery, the voltage, 7.4V, has to be multiplied by the total capacity of a single battery, which is 2000 mAh. This results in a total capacity of 14400 mAh(milliampere/hour) (14.4 ampere-hour)

Recommendations

Continuous monitoring and evaluation of the project's progress should be carried out to identify any potential issues or areas for improvement, regular feedback.

Enhanced Line Tracking: Consider incorporating additional infrared (IR) sensors or exploring alternative technologies, such as computer vision systems. These advancements would improve the car's ability to track the white line with higher accuracy and reliability.

Integration of Cameras and AI: Explore the inclusion of cameras in future iterations of the selfdriving car. Combined with artificial intelligence (AI) algorithms, the system can leverage image processing and pattern recognition to track the line more effectively. This would enable the car to adapt to various lighting conditions and handle complex line patterns.

Collaboration and Knowledge Sharing: Encourage collaboration and knowledge sharing within the team and with external partners, universities, and research institutions. Emphasize the importance of staying up-to-date with the latest advancements in autonomous vehicle technology and leveraging collective expertise for continuous improvement.

References

Blackboard. (2023). Project system – Educational Material - Example Racing Car and Pictures

Bring a Trailer. (2023). 1965 Ford Mustang Fastback [Photograph]. Retrieved April 2, 2023, from https://bringatrailer.com/listing/1965-ford-mustang-fastback-29/

Reichelt. (n.d.). LED 5MM 2MA GE LED, 5 mm, gekleurd, 3,2 mcd, laagvermogen, geel. https://www.reichelt.nl/nl/nl/led-5-mm-gekleurd-3-2-mcd-laagvermogen-geel-led-5mm-2ma-gep21629.html?PROVID=2809&gclid=Cj0KCQjwla-hBhD7ARIsAM9tQKsrcNvfWuaDkUPlIg2Wmmzlkvb5Omy3SE9T9wi_TtdDerQZ-YG6q0YaAj4hEALw_wcB

Ben's electronics.(n.d.) .Led 5mm diffused red. https://www.benselectronics.nl/led-5mmdiffuusrood.html?source=googlebase&gclid=Cj0KCQjwla-hBhD7ARIsAM9tQKvgoIS2-8SRx3DzfXFxisKEw8iPUDhvuyOT-2BO_LDkIqPjqimRwaAl4gEALw_wcB

Makerlab-electronics.(n.d.). Grove Ultrasonic Distance Sensor. https://www.makerlabelectronics.com/product/grove-ultrasonic-distance-sensor/

Speed Studio. (n.d.). Grove - Ultrasonic Distance Sensor. https://www.seeedstudio.com/GroveUltrasonicDistance-Sensor.html

Electronicscomp. (n.d.). TCRT5000 Dual Channel Line Tracking Sensor Module. https://www.electronicscomp.com/tcrt-5000-infrared-tracking-sensor-module

instructables. (n.d.). How to Use TCRT5000 IR Sensor Module With Arduino UNO. https://www.instructables.com/How-to-Use-TCRT5000-IR-Sensor-Module-With-Arduino-/

VISHAY. (2009). Reflective Optical Sensor with Transistor Output. https://www.vishay.com/docs/83760/tcrt5000.pdf

Conrad. (n.d.). Joy-it MPU6050 Versnellingssensor Versnellingssensor Geschikt voor serie: micro:bit, Arduino, Raspberry Pi, Rock Pi, Ban. https://www.conrad.nl/nl/p/joy-it-mpu6050-versnellingssensorversnellingssensor-geschikt-voor-serie-micro-bit-arduino-raspberry-pi-rock-pi-ban2136256.html

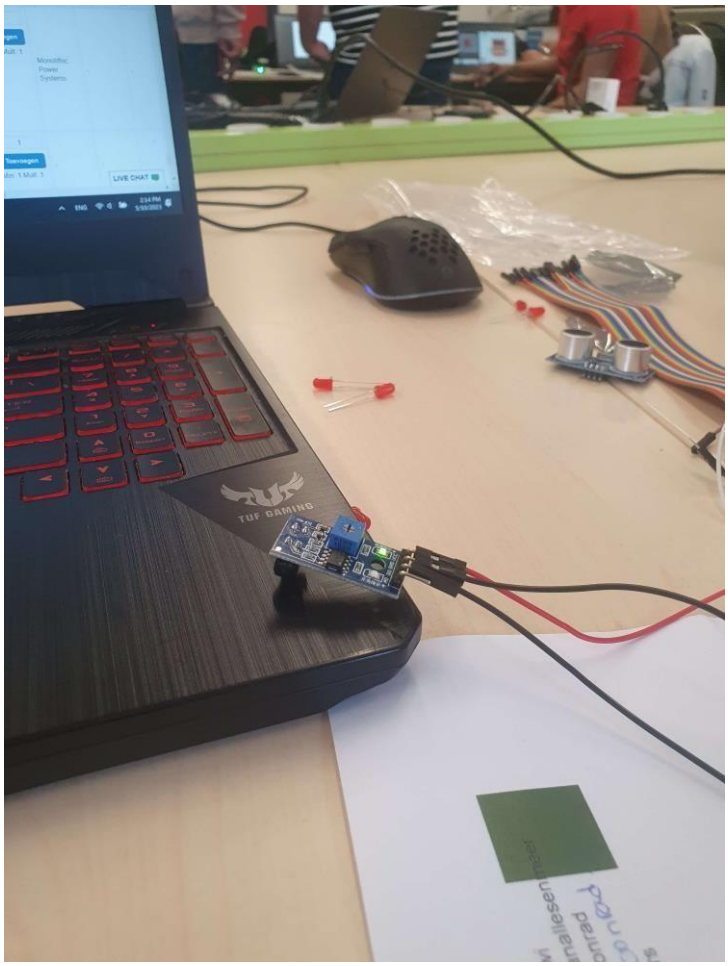
JOY-IT. (n.d.). SEN-MPU6050. https://asset.conrad.com/media10/add/160267/c1/-/en/002136256DS01/datablad-2136256-joy-it-mpu6050-versnellingssensor-versnellingssensorgeschiktvoor-serie-microbit-arduino-raspberry-pi-rock-pi-ban.pdf

Appendix

A) Gant plan

B) Unit testing:

IN0101 (IR sensor testing):


TUF GAMING

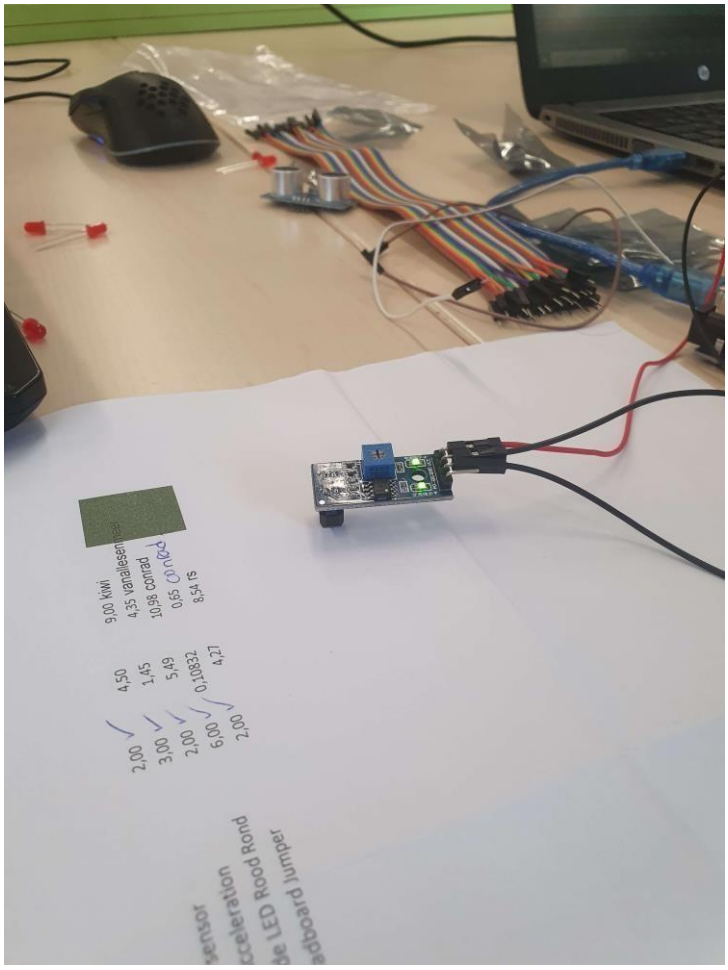
IR_Sensor_Testing

```
const int trackingPin = 2; //the tracking module attach to pin 2
const int ledPin = 13; //pin13 built-in led
void setup()
{
  Serial.begin(9600);
  pinMode(trackingPin, INPUT); // set trackingPin as INPUT
  pinMode(ledPin, OUTPUT); //set ledPin as OUTPUT
}
void loop()
{
  boolean val = digitalRead(trackingPin); // read the value of tracking module
  if(val == HIGH) //if it is HiGH
  {
    digitalWrite(ledPin, LOW); //turn off the led
    Serial.println("Detect: Black!");
  }
  else
  {
    digitalWrite(ledPin, HIGH); //turn on the led
    Serial.println("Detect: White!");
  }
}
```

COM4

```
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
Detect: Black!
```

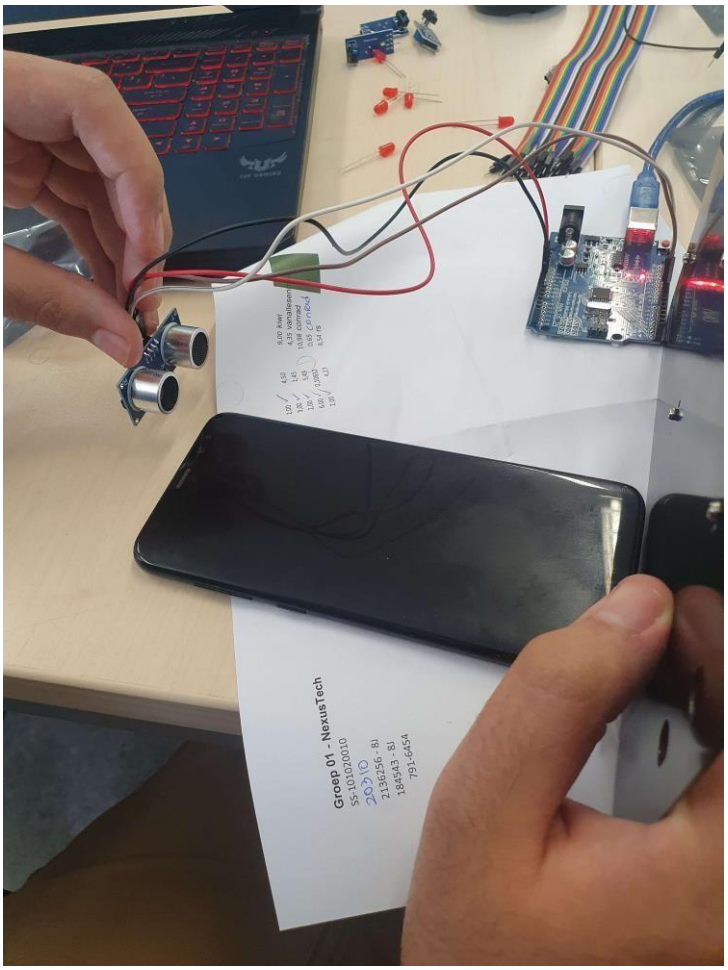
Autoscroll Show timestamp



```ino
const int trackingPin = 2; //the tracking module attach to pin 2
const int ledPin = 13; //pin13 built-in led
void setup()
{
  Serial.begin(9600);
  pinMode(trackingPin, INPUT); // set trackingPin as INPUT
  pinMode(ledPin, OUTPUT); //set ledPin as OUTPUT
}
void loop()
{
  boolean val = digitalRead(trackingPin); // read the value of tracking module
  if(val == HIGH) //if it is HiGH
  {
    digitalWrite(ledPin, LOW); //turn off the led
    Serial.println("Detect: Black!");
  }
  else
  {
    digitalWrite(ledPin, HIGH); //turn on the led
    Serial.println("Detect: White!");
  }
}
```

```
COM4
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Detect: White!
Autoscroll  Show timestamp
```

IN0102 and IN0103 (Ultrasonic sensor):



(First Sensor)

Ultrasonic_Sensor_Testing

```
*/
// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
```

COM4

```
Distance: 16
Distance: 0
Distance: 15
Distance: 0
Distance: 15
Distance: 0
Distance: 15
Distance: 0
Distance: 16
Distance: 0
Distance: 16
Distance: 0
Distance: 15
Distance: 0
Distance: 16
```

Autoscroll Show timestamp

(Second sensor)

```
*/
// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as a
  pinMode(echoPin, INPUT); // Sets the echoPin as a
  Serial.begin(9600); // Starts the serial communic
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro se
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave trav
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
```

COM4

```
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance: 16
Distance:
```

Autoscroll Show timestamp

IN0104 (Servo):

```
#include <Servo.h>
Servo esc;
int escPin = 11;

void setup()
{
  esc.attach(escPin);
  esc.writeMicroseconds(1500);
  delay(2000);

}

void loop()
{

  esc.writeMicroseconds(2000);
  delay(1000);
  esc.writeMicroseconds(1700);
  delay(1000);
  esc.writeMicroseconds(1500);
  delay(2000);
  esc.writeMicroseconds(1000);
  delay(1000);
  esc.writeMicroseconds(1200);
  delay(1000);


}
```

IN0105 (Gyroscope-Accelerometer):

```ino
#include <Wire.h>
const int MPU_ADDR = 0x68; // MPU-9250/MPU-6500 I2C address

void setup() {
  Wire.begin();
  Serial.begin(9600);
  initializeMPU();
}

void loop() {
  readGyroscopeData();
  delay(1000); // Delay for 1 second
}

void initializeMPU() {
  // Wake up the MPU-9250/MPU-6500
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0x00); // Wake up the MPU-9250/MPU-6500
  Wire.endTransmission(true);

  // Configure the gyroscope range
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x1B); // GYRO_CONFIG register
  Wire.write(0x08); // Set the full scale range to +/- 500 degrees per second
  Wire.endTransmission(true);
}

void readGyroscopeData() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x43); // Starting register for gyroscope data
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 6, true); // Request 6 bytes of data

  int16_t gyroX = Wire.read() << 8 | Wire.read();
  int16_t gyroY = Wire.read() << 8 | Wire.read();
  int16_t gyroZ = Wire.read() << 8 | Wire.read();

  Serial.print("Gyroscope X: ");
  Serial.print(gyroX);
  Serial.print(" Y: ");
  Serial.print(gyroY);
  Serial.print(" Z: ");
  Serial.println(gyroZ);
}
```

IN0106 (Ultrasonic Sensor):

```
#include <NewPing.h>
#define ULTRASONIC_TRIG_PIN 3
#define ULTRASONIC_ECHO_PIN 2
#define MAX_DISTANCE 70

// NewPing setup of pins and maximum distance.
NewPing sonar(ULTRASONIC_TRIG_PIN, ULTRASONIC_ECHO_PIN, MAX_DISTANCE);

void setup() {
  // Set the pins for the ultrasonic and IR sensors as inputs
  pinMode(ULTRASONIC_TRIG_PIN, OUTPUT);
  pinMode(ULTRASONIC_ECHO_PIN, INPUT);

  Serial.begin(9600);
}


void loop() {
  delay(50);
  unsigned int duration = sonar.ping();
  if((duration / US_ROUNDTRIP_CM) == 0) {
    Serial.println("No obstacle");
  }
  else {
    Serial.println("Obstacle");
  }
}
```

C) Implementation codes:

1) Line following

```
#include <Servo.h>
#include <NewPing.h>
#include <PID_v1.h>

// Pins
#define IR_MAX_LEFT 2
#define ECHO_PIN 3
#define TRIG_PIN 4
#define RPWM 5
#define LPWM 6
#define IR_MAX_RIGHT 7
#define STEERING 8
#define IR_LEFT 9
#define IR_MID 10
#define IR_RIGHT 11
#define LED_PIN 13

// Constants
#define MAX_DISTANCE 70
#define MAX_LEFT -3
#define LEFT -1
#define CENTER 0
#define RIGHT 1
#define MAX_RIGHT 3

// Inititalize steering servo
Servo steering;

unsigned long currentTime, previousTime = 0;

// Setup variables to use with PID
double setPoint, input, output;
double prop = 15, integ = 4, deriv = 1;

// Initialize PID
PID steeringPID(&input, &output, &setPoint, prop, integ, deriv, DIRECT);

// Speed of car
long speed = 10;
long reverseSpeed = 0;
```

```
void setup() {

  // Serial.begin(9600);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);

  steering.attach(STEERING);
  setPoint = 90;
  input = 90;
  steering.write(setPoint);
  steeringPID.SetMode(AUTOMATIC);
  steeringPID.SetOutputLimits(50, 130);

  pinMode(IR_MAX_LEFT, INPUT);
  pinMode(IR_LEFT, INPUT);
  pinMode(IR_MID, INPUT);
  pinMode(IR_RIGHT, INPUT);
  pinMode(IR_MAX_RIGHT, INPUT);
  delay(2000);
}
```

```
void loop() {

  if(endOfLine()) {
    speed = 0;
   reverseSpeed = 0;
    analogWrite(LPWM, map(speed, 0, 100, 0, 255));
  analogWrite(RPWM, map(reverseSpeed, 0 ,100, 0, 255));
  while (1){}
  }
  analogWrite(LPWM, map(speed, 0, 100, 0, 255));
  analogWrite(RPWM, map(reverseSpeed, 0 ,100, 0, 255));

  int temp = getCurrentDirection();

  if((temp < 0) && input > 50) {
    input += temp;
  } else if((temp > 0) && input < 130) {
    input += temp;
  }
  // Serial.print("Input: ");
  // Serial.println(input);

  // Compute the PID output
  steeringPID.Compute();

  //  Serial.print("Output: ");
  //  Serial.println(output);

  // Convert the PID output to a steering angle
  //int steeringAngle = constrain(map(output, -1, 1, 70, 110), 70, 110);

  //Serial.print("Steering Angle: ");
  //Serial.println(steeringAngle);

  // Control the steering servo
   steering.write(output);
}
```

```
int getCurrentDirection() {
  // Read the values from the IR sensors
  int ir1 = digitalRead(IR_MAX_LEFT);
  int ir2 = digitalRead(IR_LEFT);
  int ir3 = digitalRead(IR_MID);
  int ir4 = digitalRead(IR_RIGHT);
  int ir5 = digitalRead(IR_MAX_RIGHT);

  //Serial.println("Output: ");
  //Serial.println(ir1);
  //Serial.println(ir2);
  //Serial.println(ir3);
  return (ir1 * MAX_RIGHT) + (ir2 * RIGHT) + (ir4 * LEFT) + (ir5 * MAX_LEFT);
}

bool endOfLine() {
  int ir1 = digitalRead(IR_MAX_LEFT);
  int ir2 = digitalRead(IR_LEFT);
  int ir3 = digitalRead(IR_MID);
  int ir4 = digitalRead(IR_RIGHT);
  int ir5 = digitalRead(IR_MAX_RIGHT);

if (ir1 == 0 && ir2 == 0 && ir3 == 0 && ir4 == 0 && ir5 == 0) return true;
return false;
}
```

2) Obstacle evasion

The code isn't working, but it represents some of the ideas used for avoiding the obstacle.

```
#include <NewPing.h>
#include <Servo.h>

#define ULTRASONIC_TRIG_PIN1 4
#define ULTRASONIC_ECHO_PIN1 3
#define ULTRASONIC_TRIG_PIN2 13
#define ULTRASONIC_ECHO_PIN2 12
#define MAX_DISTANCE 35
#define RPWM 5
#define LPWM 6
#define STEERING 8
#define IR_MID 10

// NewPing setup of pins and maximum distance.
NewPing sonar1(ULTRASONIC_TRIG_PIN1, ULTRASONIC_ECHO_PIN1, MAX_DISTANCE);
NewPing sonar2(ULTRASONIC_TRIG_PIN2, ULTRASONIC_ECHO_PIN2, 20);
bool detection = false;
Servo steering;
double setPoint, input, output;
long forwardSpeed = 10;
long backwardsSpeed = 0;
void setup() {
  // Set the pins for the ultrasonic and IR sensors as inputs
  pinMode(ULTRASONIC_TRIG_PIN1, OUTPUT);
  pinMode(ULTRASONIC_ECHO_PIN1, INPUT);
  pinMode(ULTRASONIC_TRIG_PIN2, OUTPUT);
  pinMode(ULTRASONIC_ECHO_PIN2, INPUT);
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);

  steering.attach(STEERING);
  setPoint = 90;
  input = 90;
  steering.write(setPoint);

  pinMode(IR_MID, INPUT);

  delay(2000);

  Serial.begin(9600);
}
```

```
void loop() {
  analogWrite(LPWM, map(10, 0, 100, 0, 255));
  analogWrite(RPWM, map(0, 0, 100, 0, 255));

  unsigned int duration = sonar1.ping();
  if((duration / US_ROUNDTRIP_CM) != 0) {
      forwardSpeed = 0;
      analogWrite(LPWM, forwardSpeed);
       turnLeft();
       secondarySonarDetection();
       turnRight();
       secondarySonarDetection();
       turnRight();
       IRdetection();
       turnLeft();
  }
}
```

```
void turnLeft(){
    steering.write(135);
    if(checkMovingForwards() == true) {
      forwardSpeed = 10;
      analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
    }
    delay(4000);
    forwardSpeed = 0;
    analogWrite(LPWM, forwardSpeed);
    steering.write(90);
    if(checkMovingBackwards() == true) {
      backwardsSpeed = 10;
      analogWrite(RPWM, map(backwardsSpeed, 0, 100, 0, 255));
    }
    delay(4000);
    backwardsSpeed = 0;
    analogWrite(RPWM, backwardsSpeed);
//    steering.write(90);
//    if(checkMovingForwards() == true) {
//      forwardSpeed = 10;
//      analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
//    }
//    delay(1000);
//    forwardSpeed = 0;
//    analogWrite(LPWM, forwardSpeed);
//    steering.write(90);
//    if(checkMovingBackwards() == true) {
//      backwardsSpeed = 10;
//      analogWrite(RPWM, map(backwardsSpeed, 0, 100, 0, 255));
//    }
//    delay(1000);
//    backwardsSpeed = 0;
//    analogWrite(RPWM, backwardsSpeed);
}
```

```
void turnRight(){
    steering.write(45);
    if(checkMovingForwards() == true) {
      forwardSpeed = 10;
      analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
    }
    delay(4000);
    forwardSpeed = 0;
    analogWrite(LPWM, forwardSpeed);
    steering.write(90);
    if(checkMovingBackwards() == true) {
      backwardsSpeed = 10;
      analogWrite(RPWM, map(backwardsSpeed, 0, 100, 0, 255));
    }
    delay(4000);
    backwardsSpeed = 0;
//    analogWrite(RPWM, backwardsSpeed);
//    steering.write(50);
//    if(checkMovingForwards() == true) {
//      forwardSpeed = 10;
//      analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
//    }
//    delay(1000);
//    forwardSpeed = 0;
//    analogWrite(LPWM, forwardSpeed);
//    steering.write(90);
//    if(checkMovingBackwards() == true) {
//      backwardsSpeed = 10;
//      analogWrite(RPWM, map(backwardsSpeed, 0, 100, 0, 255));
//    }
//    delay(2000);
//    backwardsSpeed = 0;
//    analogWrite(RPWM, backwardsSpeed);
}
```

```
void IRdetection(){
  analogWrite(RPWM, map(0, 0, 100, 0, 255));
  if(checkMovingForwards() == true){
    forwardSpeed = 10;
    analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
  }
  while (IR_MID==HIGH){
    forwardSpeed = 0;
    analogWrite(LPWM, forwardSpeed);
  }
  delay(1000);
}

bool checkMovingForwards(){
  if(backwardsSpeed == 0){
    return true;
  }
  return false;
}

bool checkMovingBackwards(){
  if(forwardSpeed == 0){
    return true;
  }
  return false;
}
```

```
void secondarySonarDetection(){
  if(checkMovingForwards() == true){
    forwardSpeed = 10;
    analogWrite(LPWM, map(forwardSpeed, 0, 100, 0, 255));
  }
  while(detection == false){
    unsigned int duration_secondary = sonar2.ping();
    if((duration_secondary / US_ROUNDTRIP_CM) != 0) {
        detection = true;
    }
    if (detection == true) {
      break;
    }
  }
  while (detection == true) {
    unsigned int duration_secondary2 = sonar2.ping();
    if((duration_secondary2 / US_ROUNDTRIP_CM) == 0) {
        detection = false;
    }
    if (detection == false) {
      break;
    }
  }
  delay(500);
  forwardSpeed = 0;
  analogWrite(LPWM, forwardSpeed);
}
```

3) Climbing the ramp

The code isn't fully working, but it represents some of the ideas used for the ramp.

```
#include <Servo.h>
#include <NewPing.h>
#include <PID_v1.h>
#include <Wire.h>

// MPU-6050 I2C address
const int MPU6050_ADDR = 0x68;

// MPU-6050 register addresses
const int MPU6050_REG_ACCEL_X = 0x3B;
const int MPU6050_REG_ACCEL_Y = 0x3D;
const int MPU6050_REG_ACCEL_Z = 0x3F;
const int MPU6050_REG_TEMP = 0x41;
const int MPU6050_REG_GYRO_X = 0x43;
const int MPU6050_REG_GYRO_Y = 0x45;
const int MPU6050_REG_GYRO_Z = 0x47;

// MPU-6050 sensitivities
const float MPU6050_ACCEL_SENSITIVITY = 16384.0;
const float MPU6050_GYRO_SENSITIVITY = 131.0;

// Calculate pitch angle
float calculatePitch(float accelX, float accelZ) {
  return atan2(accelX, accelZ) * 180.0 / PI;
}
```

```
// Pins
#define IR_MAX_LEFT 2
#define ECHO_PIN 3
#define TRIG_PIN 4
#define RPWM 5
#define LPWM 6
#define IR_MAX_RIGHT 7
#define STEERING 8
#define IR_LEFT 9
#define IR_MID 10
#define IR_RIGHT 11
#define LED_PIN 13

// Constants
#define MAX_DISTANCE 70
#define MAX_LEFT -3
#define LEFT -1
#define CENTER 0
#define RIGHT 1
#define MAX_RIGHT 3

// Inititalize steering servo
Servo steering;

unsigned long currentTime, previousTime = 0;

// Setup variables to use with PID
double setPoint, input, output;
double prop = 15, integ = 4, deriv = 1;

long speed = 10;
long reverseSpeed = 0;

// Initialize PID
PID steeringPID(&input, &output, &setPoint, prop, integ, deriv, DIRECT);

// Speed of car
```

```
void setup() {
  // Serial.begin(9600);
  Wire.begin();
    // Initialize MPU-6050
  Wire.beginTransmission(MPU6050_ADDR);
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0);    // Wake up MPU-6050
  Wire.endTransmission();

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);

  steering.attach(STEERING);
  setPoint = 90;
  input = 90;
  steering.write(setPoint);
  steeringPID.SetMode(AUTOMATIC);
  steeringPID.SetOutputLimits(50, 130);

  pinMode(IR_MAX_LEFT, INPUT);
  pinMode(IR_LEFT, INPUT);
  pinMode(IR_MID, INPUT);
  pinMode(IR_RIGHT, INPUT);
  pinMode(IR_MAX_RIGHT, INPUT);
  delay(2000);
}
```

```
void loop() {


  if(endOfLine()) {
    speed = 0;
   reverseSpeed = 0;
    analogWrite(LPWM, map(speed, 0, 100, 0, 255));
  analogWrite(RPWM, map(reverseSpeed, 0 ,100, 0, 255));
  while (1){}
  }
  analogWrite(LPWM, map(speed, 0, 100, 0, 255));
  analogWrite(RPWM, map(reverseSpeed, 0 ,100, 0, 255));

  int temp = getCurrentDirection();

  if((temp < 0) && input > 50) {
    input += temp;
  } else if((temp > 0) && input < 130) {
    input += temp;
  }
  // Serial.print("Input: ");
  // Serial.println(input);

  // Compute the PID output
  steeringPID.Compute();
```

```
//  Serial.print("Output: ");
//  Serial.println(output);

// Convert the PID output to a steering angle
//int steeringAngle = constrain(map(output, -1, 1, 70, 110), 70, 110);

//Serial.print("Steering Angle: ");
//Serial.println(steeringAngle);

// Control the steering servo
 steering.write(output);

  Wire.beginTransmission(MPU6050_ADDR);
Wire.write(MPU6050_REG_ACCEL_X);
Wire.endTransmission(false);
Wire.requestFrom(MPU6050_ADDR, 6, true);

int16_t accelX = Wire.read() << 8 | Wire.read();
int16_t accelY = Wire.read() << 8 | Wire.read();
int16_t accelZ = Wire.read() << 8 | Wire.read();

// Convert raw accelerometer data to G's
float accelX_G = accelX / MPU6050_ACCEL_SENSITIVITY;
float accelY_G = accelY / MPU6050_ACCEL_SENSITIVITY;
float accelZ_G = accelZ / MPU6050_ACCEL_SENSITIVITY;

 // Calculate pitch angle
 float pitchAngle = calculatePitch(accelX_G, accelZ_G);


 static float prevPitch = 0.0;
  float acceleration = 0.0;
  float difference  = pitchAngle - prevPitch;

  if(difference < 0){
   difference = difference*(-1);
  }

 if ((pitchAngle > prevPitch) && (difference > 2)) {
   difference = pitchAngle - prevPitch;
   speed = (pitchAngle / 45.0)*40;
   reverseSpeed = 0;
 } else if ((pitchAngle < prevPitch) && (difference > 15)) {
   difference = pitchAngle - prevPitch;
   speed = 0;
   reverseSpeed = 0;
 }
}
```

```
int getCurrentDirection() {
  // Read the values from the IR sensors
  int ir1 = digitalRead(IR_MAX_LEFT);
  int ir2 = digitalRead(IR_LEFT);
  int ir3 = digitalRead(IR_MID);
  int ir4 = digitalRead(IR_RIGHT);
  int ir5 = digitalRead(IR_MAX_RIGHT);

  //Serial.println("Output: ");
  //Serial.println(ir1);
  //Serial.println(ir2);
  //Serial.println(ir3);
  return (ir1 * MAX_RIGHT) + (ir2 * RIGHT) + (ir4 * LEFT) + (ir5 * MAX_LEFT);
}
```

```
bool endOfLine() {
  int ir1 = digitalRead(IR_MAX_LEFT);
  int ir2 = digitalRead(IR_LEFT);
  int ir3 = digitalRead(IR_MID);
  int ir4 = digitalRead(IR_RIGHT);
  int ir5 = digitalRead(IR_MAX_RIGHT);
  if(ir1 == 0 && ir2 == 0 && ir3 == 0 && ir4 == 0 && ir5 == 0) {
    return true;
  }
  return false;
}
```