ENGINEERING
W Booth School of Engineering
Practice and Technology

McMaster University

SEP 780: ADVANCED ROBOTICS AND AUTOMATION

FALL 2024

GUIDED BY:

PROF. RICHARD MA

FINAL PROJECT REPORT

TITLE:

VISION-ENABLED FACIAL SEARCH VEHICLE

GROUP 2

STUDENT NAME	STUDENT ID
Leo Chen	400188406
Priyam Gandhi	400490099
Juliusz Gasior	400490100
Mostafa Tehrani	400490450

Table of Contents

Table of Figures

1 INTRODUCTION

1.1 BACKGROUND

The Freenove 4WD Smart Car Kit provides an intelligent robotics development platform that can be used either for amateur projects or can be taught academically. Together with the addition of sensors and actuators with a microcontroller, it will support the development of a fully functional robotic vehicle. This project works on developing such a system that shall be able to work in both manual and autonomous modes, letting the robot go through the space around it by using obstacle detection and face recognition methodologies. (Freenove, n.d.)

At its core lies the Freenove Control Board, compatible with Arduino; thus, advanced programmability and precision control of its hard elements are possible. An additional distance-measuring sensor and the HuskyLens for face recognition have been included in the robot. These, in turn, all combine to enable the robot to navigate through different terrains, and obstacles, and to perform elementary functions like stopping upon detection of a face. IR remote control is available for manual operations to enable a user to have flexibility in instructing the robot for various activities and also for switching to auto or manual mode.

The system devises a new way of obstacle avoidance whereby the robot auto-modulates its movement according to real-time information coming from the ultrasonic sensor. Thus, the autonomous navigational capabilities of the robot come into play, thereby illustrating the smart robotic functionalities.

1.2 PURPOSE

The purpose of this project is to build a smart autonomous robot using the Freenove 4WD Smart Car Kit, incorporating both autonomous and manual control features. The robot is designed to perform the following tasks.

1. **Obstacle Avoidance (Autonomous Mode)**
   In this mode, the ultrasonic sensor monitors any obstacle that may come in its way. Once an obstacle is detected, the robot immediately stops or turns to avoid a collision. This feature is very similar to real-world systems of autonomous navigation in self-driving cars or robotic vacuum cleaners.

2. **Facial Recognition and Automatic Termination**
   The HuskyLens works for face recognition the moment it detects a human face, the robot automatically stops working either through embedded safety mechanisms or because it has been taught to do so, and it will then be allowed to identify persons or objects for interaction.

3. **Manual Adjustment via Infrared Remote Controller**

The infrared remote-control device lets users control the movement of the robot, which includes moving forward, backward, rotation, and stoppage. Moreover, the remote helps in changing the mode from manual to automatic and vice-versa with great ease for diversified use.

4. **Buzzer Feedback**
   It will provide audible feedback through a buzzer in case of any significant events, like switching modes, obstacle detection, or face recognition.

2 HARDWARE

In this section, we will take a deeper look at the hardware utilized in the Freenove vehicle kit project, including the Freenove Control Board (Based on Arduino, used to manage inputs and outputs for sensors, motors, and LEDs) and Extension Board that Connects additional modules like ultrasonic sensors, motors, and Bluetooth, acting as an interface.

2.1 ACRYLIC PARTS

In the Freenove 4WD Smart Car Kit are lightweight structural components forming the car's chassis. They provide mounting points for motors, sensors, LEDs, and control boards. Key parts include the bottom board for motors and wheels and the top board for LEDs and control modules.

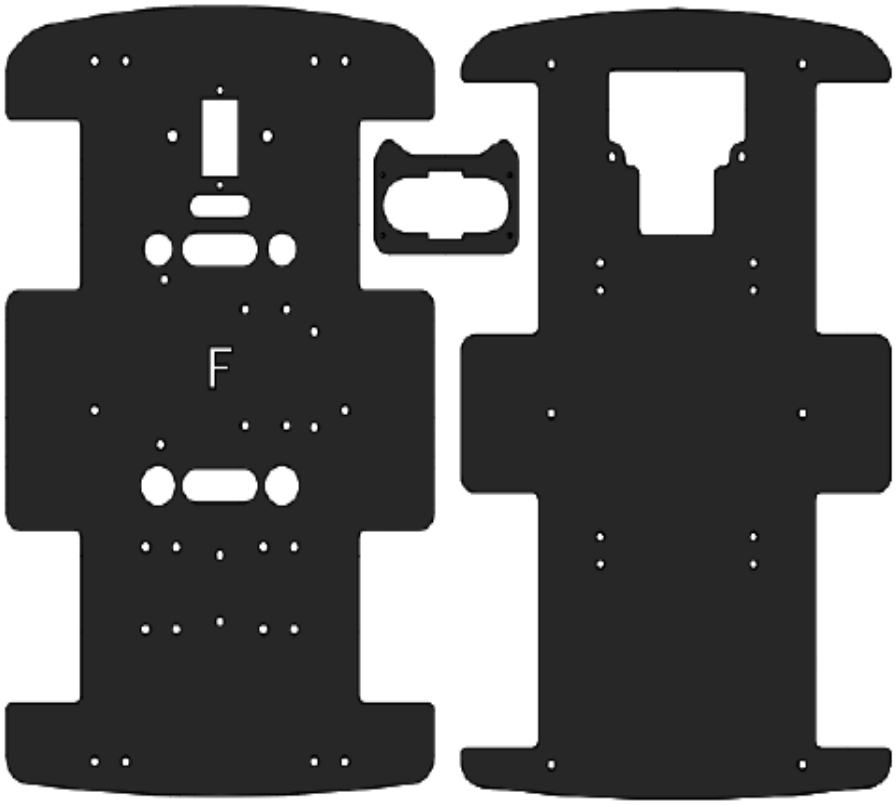

Figure 1: Car Chassis

2.2 MECHANICAL PARTS

The kit had the required Bolts, screws, and nuts of different sizes needed for the assembly.

Figure 2: Screws & Nuts

2.3 TRANSMISSION PARTS

This includes servo motors, DC motors, Drive Wheels, and Motor Bracket packages as shown below.

1. Motors (x4): These are the primary sources of power for the wheels, converting electrical energy into rotational motion.
2. Driver Wheels (x4): These attach to the motors, directly transmitting the rotational motion from the motors to the ground, enabling movement.
3. Servo (x1): Although not part of the main transmission, the servo is used for precise positioning tasks, such as rotating the ultrasonic sensor.
4. Motor Bracket Packages (x4)**:** These include brackets and screws to securely mount the motors to the car's chassis, ensuring stability.

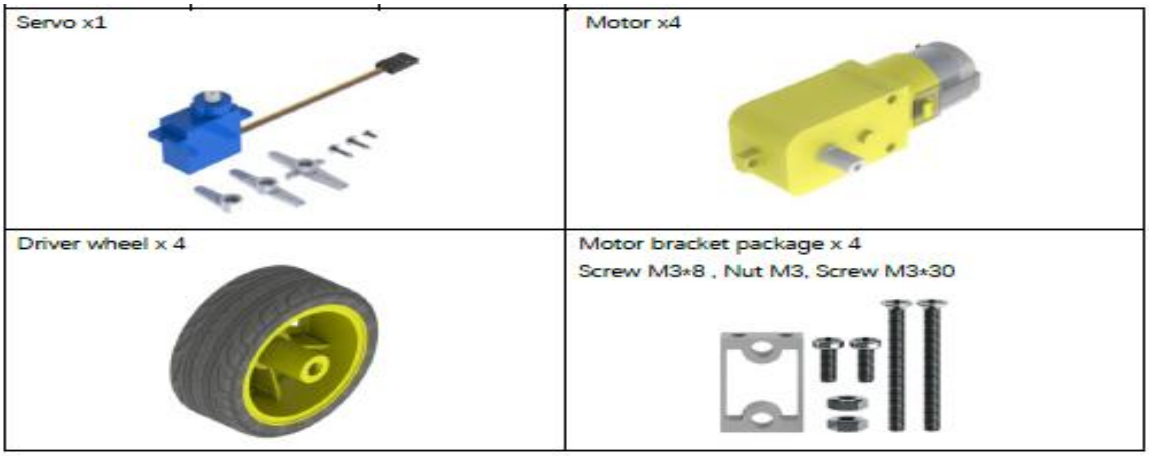

Figure 3: Transmission Parts

2.4 ELECTRONIC PARTS

2.4.1 FREENOVE CONTROL BOARD

The Freenove Control Board is the main Arduino-based controller for the 4WD Smart Car. It manages inputs from sensors and outputs to motors, LEDs, and servos. Key features include digital and analog I/O ports, PWM support, a USB interface for programming, onboard LEDs for power and communication, and a reset button. It powers and controls the car's components, integrating modules like Bluetooth or RF for remote functionality.

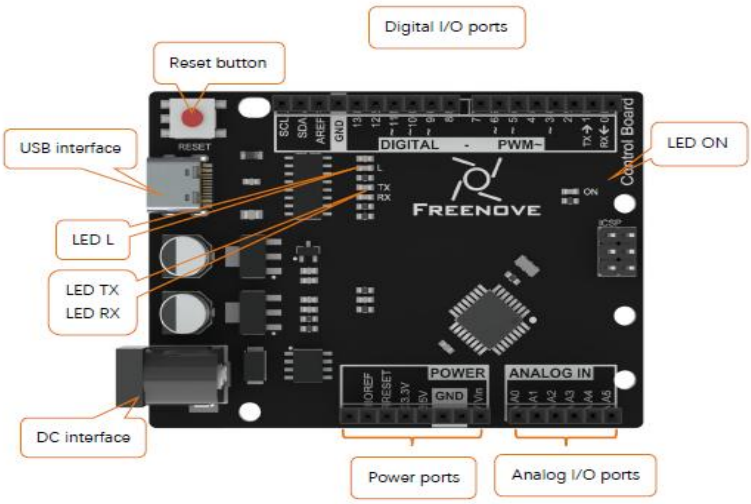

Figure 4: Arduino UNO Control Board

2.4.2 FREENOVE 4WD EXTENSION BOARD

The Freenove 4WD Extension Board extends the control board by adding dedicated ports and connections for the components of the smart car. It includes a motor port, sensor ports like ultrasonic or line-tracking-and module connections for Bluetooth, RF, and IR.

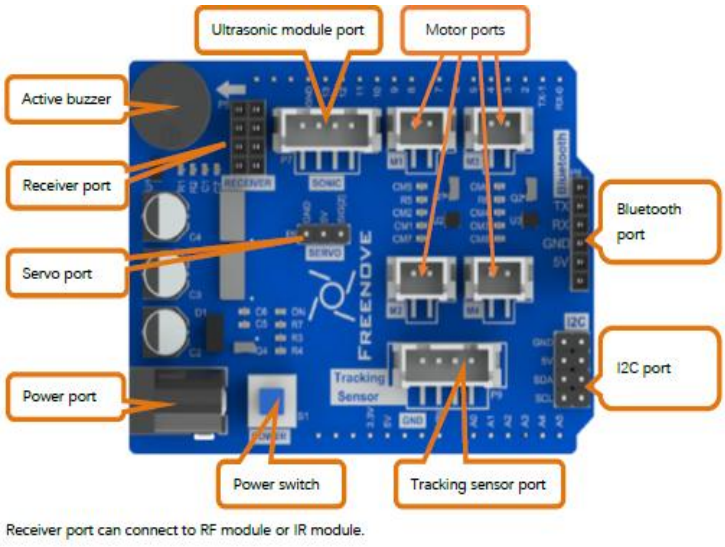

Figure 5: FreeNove Arduino UNO Extension Board

2.4.3 ULTRASONIC MODULE CONNECTOR

The Ultrasonic Module Connector on the Freenove Extension Board connects the ultrasonic sensor to detect obstacles. It has a pin each for Trig, Echo, VCC, and GND to allow the sensor to measure distances to support obstacle avoidance.

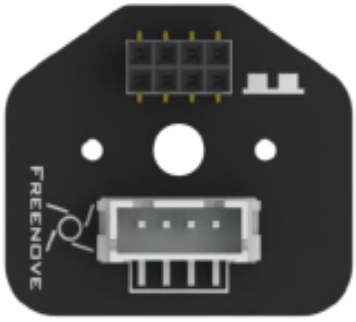

Figure 6: Ultrasonic Sensor Module Connector

2.4.4 ULTRASONIC MODULE

The ultrasonic module of the Freenove 4WD Smart Car Kit adopts Trig to send a pulse and Echo to receive the reflected signal and then uses the time taken to calculate the distances to detect obstacles. Connect the ultrasonic module to the ultrasonic module port on the extension board. This module is the main obstacle avoidance for the car.

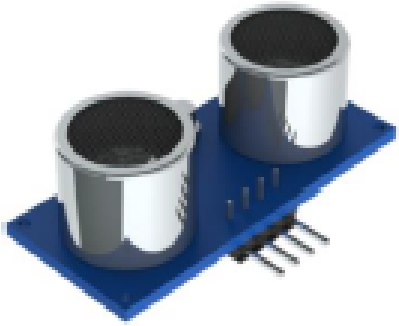

Figure 7: HC-SR04 Ultrasonic Sensor

2.4.5 IR RECEIVER

The IR Receiver in the Freenove 4WD Smart Car Kit detects infrared signals from a remote control.

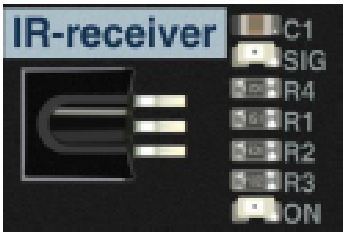

Figure 8: IR Receiver

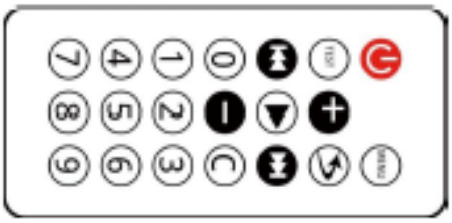

Figure 9: IR Remote

3 SOFTWARE

In this section, we will take a deeper look at the software utilized in the Freenove vehicle kit project, including Arduino IDE, which is used to program control boards, and Solidworks, which is used to construct the fixture for the HuskyLens camera.

3.1 ARDUINO IDE SOFTWARE

Arduino IDE Software is the software platform the team used to program the Freenove Control Board (which is based on Arduino UNO). The basic features of the the software allow for editing Arduino sketches and uploading them to the Freenove Control Board using a USB cable. The serial monitor function allows for debugging the program through real-time communication with the board. The software also supports multiple libraries provided by Freenove, making it easier to integrate various sensors and modules.

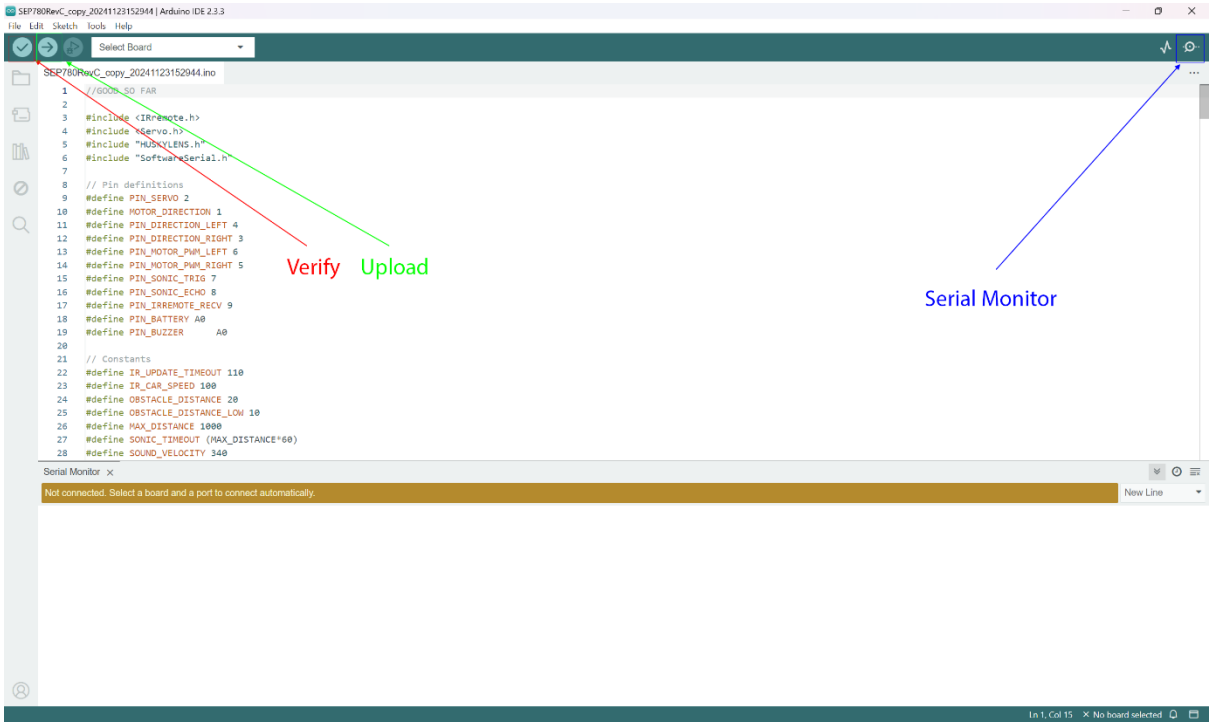

Figure 10: Screenshot of Arduino Uno IDE Software Interface

3.2 SOLIDWORKS

SolidWorks is a powerful CAD software the team used to design a custom fixture for the HuskyLens camera. The software allows for the designing and editing of 3D models according to functional and dimensional requirements. Once the design is finalized, the 3D model can be exported for 3D printing.

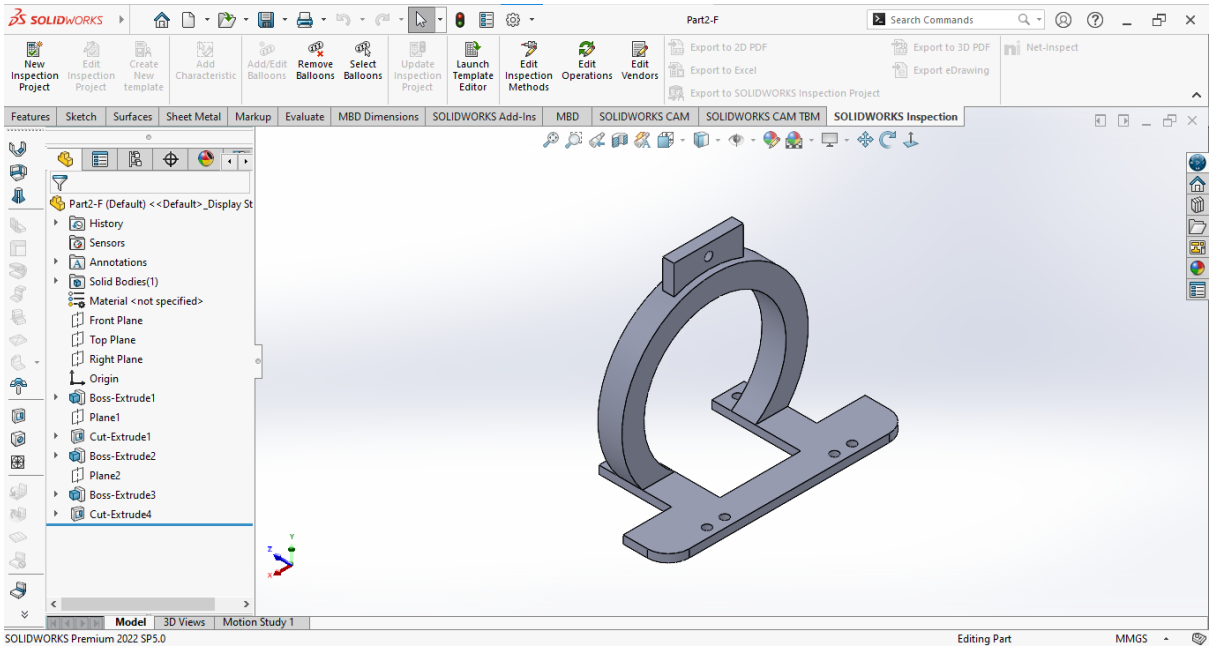

Figure 11: Screenshot of SolidWorks Software Interface

4 METHODOLOGY

4.1 CONCEPTS

Our project aims to expand the functionality of the basic robot by making the robot able to conduct tasks based on vision data with an integrated camera. One of those functions is facial detection, which we developed in the project as a proof-of-concept of the scalability of the robot.

In manual mode, the operator controls the robot using buttons on the IR remote controller. In autonomous mode, the robot moves by itself and automatically avoids obstacles detected by the ultrasonic sensor. The robot stops moving with a buzzer noise if the HuskyLens camera recognizes one of the faces it stores in its library. The buzzer also gives feedback when a critical action such as reset, or mode change occurs. Integrating the HuskyLens camera with the essential hardware components ensures the robot can process and respond to real-time vision data. This highly scalable design can perform many more functions besides facial detection.

4.2 MATERIALS

The following components were utilized to achieve the project goal of being the proof-of-concept of vision-based functionalities that can be integrated into the robot.

4.2.1 FREENOVE 4WD CAR KIT

The Freenove 4WD Car Kit comes with most of the required hardware components for our project, and all of them are mentioned in the hardware section above. The kit comes with an instruction manual for assembly, and the additional resources provided by the kit developer also include a set of pre-written libraries and sketches for Arduino that can be used to execute different operational modes or troubleshoot in case of suspected malfunctions.

4.2.2 HUSKYLENS CAMERA

HuskyLens Camera is powered by artificial intelligence and was used as a vision sensor for face detection in our project. It allows teaching and storing different types of objects, including human faces, in its library; if the camera recognizes any of the stored objects, it will send a unique ID output to the control board.

4.2.3 BREADBOARD

The breadboard is a board that allows for the construction of electronic circuits without the need for soldering the wires. In our project, it was a very useful tool at the early stage to test the functionalities of electronic components such as the servo motor and the ultrasonic sensor.

4.2.4 BATTERIES

We used two 3.7V 4000mAh rechargeable batteries from a Tokeyla flashlight to power the entire system, including the HuskyLens camera. The flashlight itself was also used to charge those two batteries, which makes the power supply simple and cost-effective.

4.2.5 MULTIMETER

A multimeter was used to check the battery voltage and troubleshoot components after we finished the assembly and entered the testing phase of the project. The multimeter's beep function was essential for checking the connectivity of the pins and wires to identify the issues with wires and sensors.

4.2.6 3D PRINTER

A 3D printer was used to print a custom cover and a custom fixture for the HuskyLens camera designed using SolidWorks. The 3D printed parts were used to attach the HuskyLens camera to the robot securely and prevent it from physical damage during the operation.

4.3 FUNCTIONALITY

The robotic system was designed to operate in both manual and autonomous modes. The manual mode gives the operator complete control over the movement of the robot using the IR remote controller, in which the IR receiver receives the IR signals and sends them to the Freenove Control Board, which is responsible for processing those signals to execute the actions. In autonomous mode, the ultrasonic sensor continuously updates the distance values, and the HuskyLens camera constantly searches for the stored objects from the real-time vision it captures; the Freenove Control Board in this mode is responsible for processing data inputs from both sensors and converts them into actions

according to the program uploaded. The operator is also allowed to switch between manual and autonomous modes using the IR remote. (Angelo, n.d.)

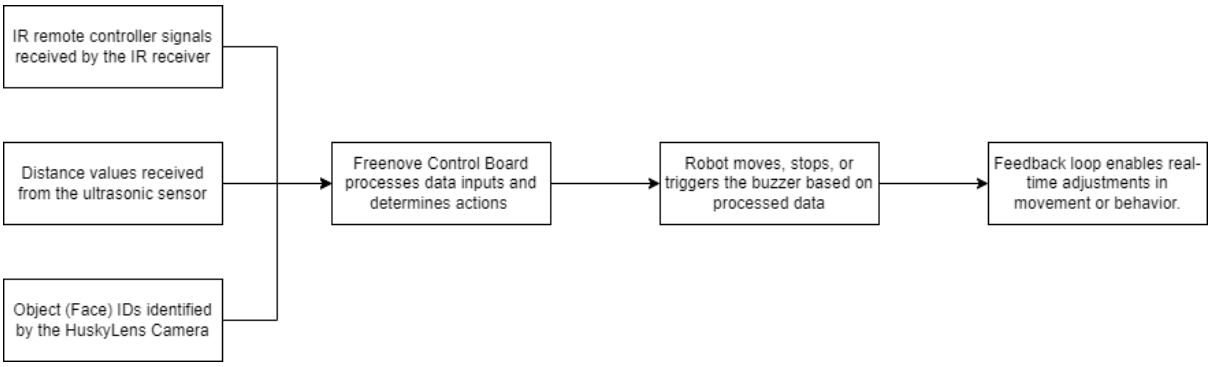

Figure 12: Workflow Diagram of Robot System Functions

5 IMPLEMENTATION

5.1 BLOCK DIAGRAM

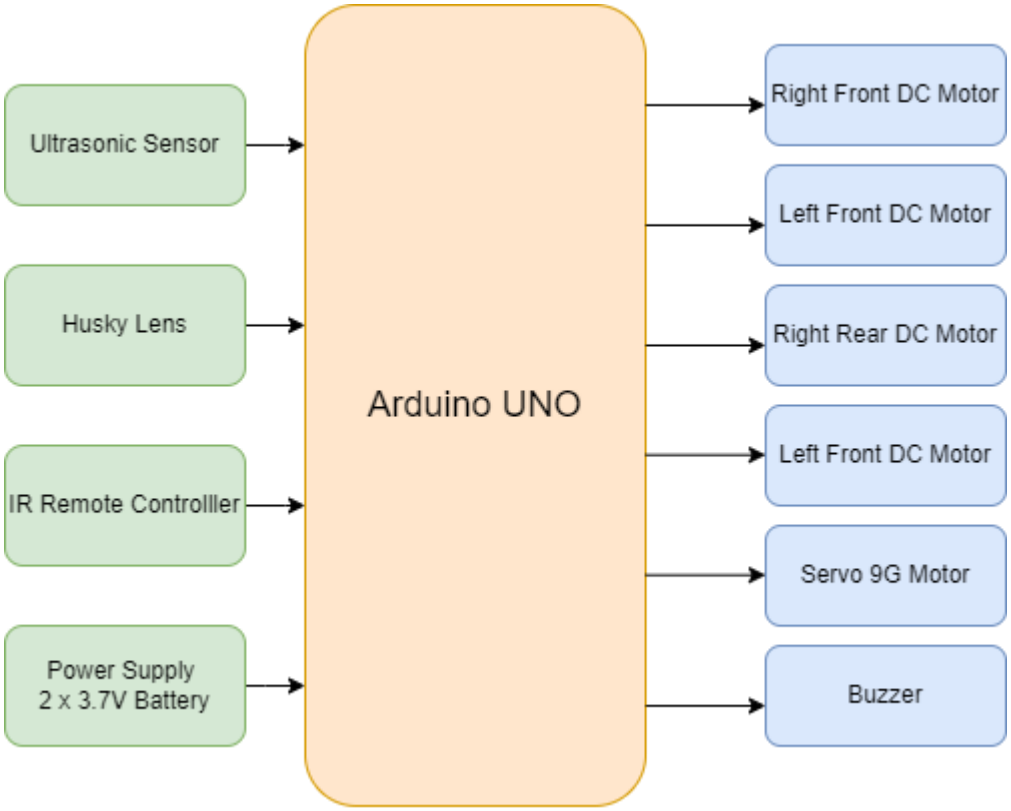

Figure 13: System Architecture (Block Diagram)

5.2 PROGRAMMING OF FREENOVE CONTROL BOARD

Arduino IDE software was used to program the Freenove Control Board. The program integrates different functionalities, including manual control, automatic obstacle avoidance, and sensor-based monitoring.

```
#include <IRremote.h>
#include <Servo.h>
#include "HUSKYLENS.h"
#include "SoftwareSerial.h"

// Pin definitions
#define PIN_SERVO 2
#define MOTOR_DIRECTION 1
#define PIN_DIRECTION_LEFT 4
#define PIN_DIRECTION_RIGHT 3
#define PIN_MOTOR_PWM_LEFT 6
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_SONIC_TRIG 7
#define PIN_SONIC_ECHO 8
#define PIN_IRREMOTE_RECV 9
#define PIN_BATTERY A0
#define PIN_BUZZER      A0

// Constants
#define IR_UPDATE_TIMEOUT 110
#define IR_CAR_SPEED 100
#define OBSTACLE_DISTANCE 20
#define OBSTACLE_DISTANCE_LOW 10
#define MAX_DISTANCE 1000
#define SONIC_TIMEOUT (MAX_DISTANCE*60)
#define SOUND_VELOCITY 340

// IR remote key codes
#define IR_REMOTE_KEYCODE_UP 0xFF02FD
#define IR_REMOTE_KEYCODE_DOWN 0xFF9867
#define IR_REMOTE_KEYCODE_LEFT 0xFFE01F
#define IR_REMOTE_KEYCODE_RIGHT 0xFF906F
#define IR_REMOTE_KEYCODE_CENTER 0xFFA857
#define IR_REMOTE_KEYCODE_1 0xFF30CF
#define IR_REMOTE_KEYCODE_9 0xFF52AD

IRrecv irrecv(PIN_IRREMOTE_RECV);
decode_results results;
Servo servo;
```

Figure 14: Libraries and Definitions

The program begins by including libraries for remote control, servo motor operation, HuskeLens detection, and serial communication, and then the program defines the pin numbers for hardware components, the constants for timeout values, speed limits and sensor parameters, and the hexadecimal values for signals received by the IR Receiver.

```
bool isAutoMode = false;
bool isCarStopped = false;
bool faceFound = false;
int speedOffset = -75;
unsigned long lastIRUpdateTime = 0;
bool autoOn = false;

// Define HUSKYLENS
HUSKYLENS huskylens;
SoftwareSerial mySerial(18, 19); // RX, TX
//HUSKYLENS green line >> Pin 10; blue line >> Pin 11

///Define variables
float batteryVoltage = 0;
bool isBuzzered = false;

void setup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
  pinMode(PIN_SONIC_TRIG, OUTPUT);
  pinMode(PIN_SONIC_ECHO, INPUT);
  servo.attach(PIN_SERVO);
  irrecv.enableIRIn();
  calculateVoltageCompensation();

    Serial.begin(9600);
    mySerial.begin(9600);
    while (!huskylens.begin(mySerial))
    {
        Serial.println(F("Begin failed!"));
        Serial.println(F("1.Please recheck the \"Protocol Type\" in HUSKYLENS (General Settings>>Protocol Type>>Serial 9600)"));
        Serial.println(F("2.Please recheck the connection."));
        delay(100);
    }
}
```

Figure 15: Global Variables and Function Setup

After the definitions and libraries, the program initializes the hardware components and their functions, including the servo motor, ultrasonic sensor, and IR receiver, and it establishes communication with HuskyLens using the software serial interface.

```
void loop() {
  //battery level check. if low, beep
  checkBatteryVoltage();

  if (irrecv.decode(&results)) {
    handleIRCommand(results.value);
    irrecv.resume();
    lastIRUpdateTime = millis();
  } else if (!isAutoMode && millis() - lastIRUpdateTime > IR_UPDATE_TIMEOUT) {
    motorRun(0, 0);
  }
  if (isAutoMode && !isCarStopped) {

    huskyLensRun();
    if (!faceFound) {
      updateAutomaticObstacleAvoidance();
    } else {
      isAutoMode = false;
    }
  }

}
```

Figure 16: Main Loop

```
104  void huskyLensOutput(HUSKYLENSResult result) {
105
106      Serial.println(result.ID);
107      if (result.ID == 1) {
108        faceFound = true;
109        resetCarAction();
110        alarm(4,1);
111      } else {
112        faceFound = false;
113      }
114  }
115
116  void huskyLensRun() {
117      if (!huskylens.request()) Serial.println(F("Fail to request data from HUSKYLENS, recheck the connection!"));
118      else if(!huskylens.isLearned()) Serial.println(F("Nothing learned, press learn button on HUSKYLENS to learn one!"));
119      else if(!huskylens.available()) Serial.println(F("No block or arrow appears on the screen!"));
120      else
121      {
122          Serial.println(F("###########"));
123          while (huskylens.available())
124          {
125              HUSKYLENSResult result = huskylens.read();
126              huskyLensOutput(result);
127              break;
128          }
129      }
130
131  }
```

Figure 17: HuskyLens Function Definitions

```
133  void resetCarAction() {
134    motorRun(0, 0);
135    setBuzzer(false);
136  }
137
138  void setBuzzer(bool flag) {
139    isBuzzered = flag;
140    pinMode(PIN_BUZZER, flag);
141    digitalWrite(PIN_BUZZER, flag);
142  }
143
144  void alarm(u8 beat, u8 repeat) {
145    beat = constrain(beat, 1, 9);
146    repeat = constrain(repeat, 1, 255);
147    for (int j = 0; j < repeat; j++) {
148      for (int i = 0; i < beat; i++) {
149        setBuzzer(true);
150        delay(100);
151        setBuzzer(false);
152        delay(100);
153      }
154      delay(500);
155    }
156  }
```

Figure 18: Helper Functions

```
158  void handleIRCommand(unsigned long command) {
159    switch (command) {
160      case IR_REMOTE_KEYCODE_UP:
161        if (!isAutoMode && !isCarStopped) motorRun(IR_CAR_SPEED, IR_CAR_SPEED);
162        break;
163      case IR_REMOTE_KEYCODE_DOWN:
164        if (!isAutoMode && !isCarStopped) motorRun(-IR_CAR_SPEED, -IR_CAR_SPEED);
165        break;
166      case IR_REMOTE_KEYCODE_LEFT:
167        if (!isAutoMode && !isCarStopped) motorRun(-IR_CAR_SPEED, IR_CAR_SPEED);
168        break;
169      case IR_REMOTE_KEYCODE_RIGHT:
170        if (!isAutoMode && !isCarStopped) motorRun(IR_CAR_SPEED, -IR_CAR_SPEED);
171        break;
172      case IR_REMOTE_KEYCODE_CENTER:
173        isCarStopped = !isCarStopped;
174        isAutoMode = false;
175        motorRun(0, 0);
176        break;
177      case IR_REMOTE_KEYCODE_1:
178        if (!isCarStopped) {
179          isAutoMode = !isAutoMode;
180          motorRun(0, 0);
181        }
182        break;
183      case IR_REMOTE_KEYCODE_9:
184        //reset car
185        isAutoMode = false;
186        isCarStopped = false;
187        faceFound = false;
188        autoOn = false;
189        alarm(2,1);
190    }
191  }
```

Figure 19: Handle IR Command

```
void updateAutomaticObstacleAvoidance() {
  int distance[3], tempDistance[3][5], sumDisntance;
  static u8 leftToRight = 0, servoAngle = 0, lastServoAngle = 0;
  const u8 scanAngle[2][3] = { {150, 90, 30}, {30, 90, 150} };
  int delayTime = 200;
  double speedRatio = 0.90;
  //move car forward until it starts seeing obstacles
  if (autoOn == false) {
    motorRun(90,90);
    autoOn = true;
  }

  for (int i = 0; i < 3; i++) {
    servoAngle = scanAngle[leftToRight][i];
    servo.write(servoAngle);
    if (lastServoAngle != servoAngle) {
      delay(130);
    }
    lastServoAngle = servoAngle;
    for (int j = 0; j < 5; j++) {
      tempDistance[i][j] = getSonar();
      delayMicroseconds(2 * SONIC_TIMEOUT);
      sumDisntance += tempDistance[i][j];
    }
    if (leftToRight == 0) {
      distance[i] = sumDisntance / 5;
    } else {
      distance[2 - i] = sumDisntance / 5;
    }
    sumDisntance = 0;
  }
  leftToRight = (leftToRight + 1) % 2;

  if (distance[1] < OBSTACLE_DISTANCE) {
    if (distance[0] > distance[2] && distance[0] > OBSTACLE_DISTANCE) {
      motorRun(-(speedRatio * 150), -(speedRatio * 150));
      delay(delayTime);
      motorRun(-(speedRatio * 150), (speedRatio * 150));
    } else if (distance[0] < distance[2] && distance[2] > OBSTACLE_DISTANCE) {
      motorRun(-(speedRatio * 150), -(speedRatio * 150));
      delay(delayTime);
      motorRun((speedRatio * 150), -(speedRatio * 150));
    } else {
      motorRun(-(speedRatio * 150), -(speedRatio * 150));
      delay(delayTime);
      motorRun(-(speedRatio * 150), (speedRatio * 150));
    }
  } else {
    if (distance[0] < OBSTACLE_DISTANCE_LOW) {
      motorRun(-(speedRatio * 150), -(speedRatio * 150));
      delay(delayTime);
      motorRun((speedRatio * 180), (speedRatio * 50));
    } else if (distance[2] < OBSTACLE_DISTANCE_LOW) {
      motorRun(-(speedRatio * 150), -(speedRatio * 150));
      delay(delayTime);
      motorRun((speedRatio * 50), (speedRatio * 180));
    } else {
      motorRun((speedRatio * 80), (speedRatio * 80));
    }
  }
}

float getSonar() {
  unsigned long pingTime;
  float distance;
  digitalWrite(PIN_SONIC_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_SONIC_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_SONIC_TRIG, LOW);
  pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT);
  if (pingTime != 0)
    distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000;
  else
    distance = MAX_DISTANCE;
  return distance;
}
```

Figure 20: Obstacle Avoidance Functions

```
void motorRun(double speedl, double speedr) {
  int dirL = (speedl > 0) ? (0 ^ MOTOR_DIRECTION) : (1 ^ MOTOR_DIRECTION);
  int dirR = (speedr > 0) ? (1 ^ MOTOR_DIRECTION) : (0 ^ MOTOR_DIRECTION);
  speedl = abs(speedl);
  speedr = abs(speedr);
  digitalWrite(PIN_DIRECTION_LEFT, dirL);
  digitalWrite(PIN_DIRECTION_RIGHT, dirR);
  analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
  analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
}
```

Figure 21: Motor Control

```
void calculateVoltageCompensation() {
  float voltageOffset = 8.4 - getBatteryVoltage();
  speedOffset = voltageOffset * 20;
}

float getBatteryVoltage() {
  pinMode(PIN_BATTERY, INPUT);
  int batteryADC = analogRead(PIN_BATTERY);
  float batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
  return batteryVoltage;
}

void checkBatteryVoltage() {
  float batteryVolt = getBatteryVoltage();
  if (batteryVolt > 0) {
      Serial.println(String("Battery Voltage: ") + String(batteryVolt,1));
  }
  //if
  if (batteryVolt < 6.4) {
      alarm(2,1);
  }

}
```

Figure 22: Battery Management

During operation, the main loop repeatedly executes tasks to check battery voltage and process IR remote signals; the program will handle the IR remote commands in manual mode and run automatic obstacle avoidance and HuskyLens detection in autonomous mode. If a face is detected, the robot will switch out of autonomous mode with buzzer feedback. If the reset button is pressed on the IR remote controller, the robot will immediately stop, and the program will reset.

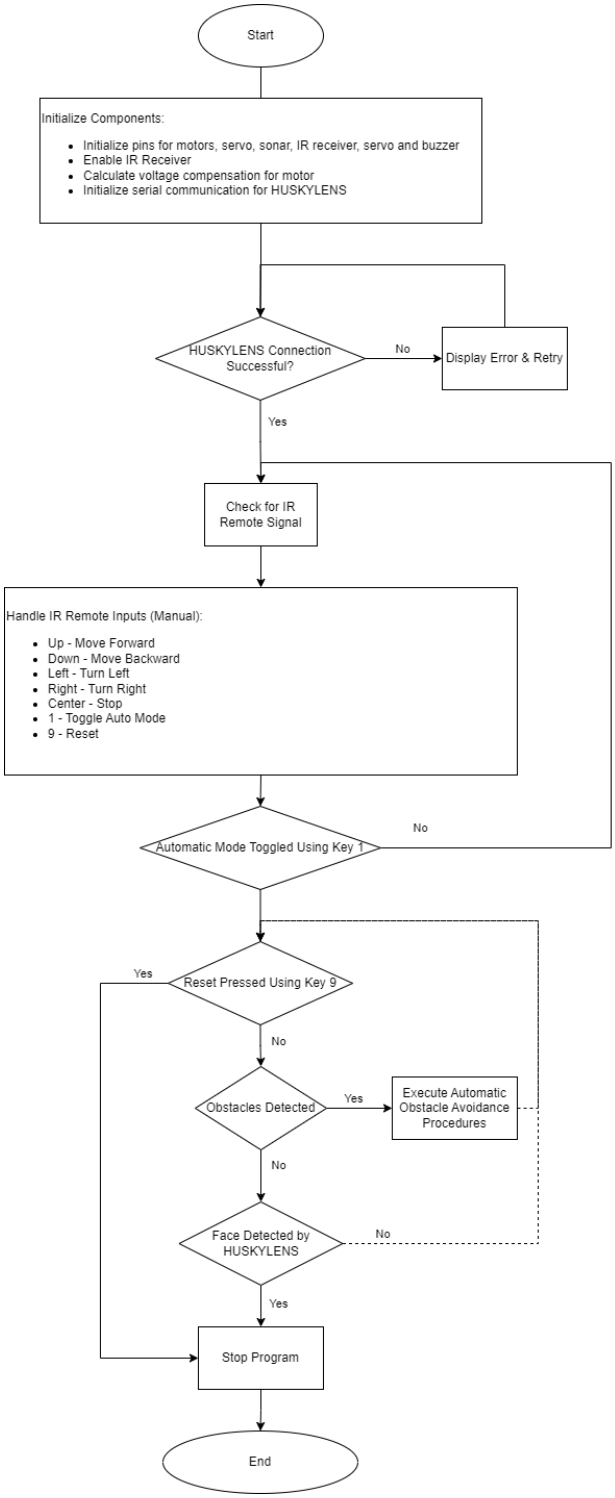

Figure 23: System Flow Chart

6 RESULTS

The robotic system functioned as it was designed to function, successfully meeting the design objectives. The robot could operate in both manual and autonomous modes, and the operator could switch between the two modes. In manual mode, the robot movements were precise, and the robot was responsive to the IR remote controller. In autonomous mode, the robot could effectively avoid obstacles detected by the ultrasonic sensor, while the HuskyLens successfully recognized the face and stopped the robot with a buzzer noise. Additionally, the batteries and fixtures were reliable during the operation.

7 CONCLUSION

In conclusion, the Freenove 4WD Smart Car Kit was upgraded with an ultrasonic sensor for obstacle detection and the use of HuskyLens for face recognition allowed showing both the manual and autonomous modes in the robotics frame. The Arduino UNO board, infrared remote controller, and several types of sensors were combined in the multifunctional platform of intelligent interaction with the outside world.

The dual functionality of the system permitting manual control through the infrared remote as well as facilitating automatic obstacle avoidance utilizing the ultrasonic sensor represents a significant advancement in the development of more autonomous systems capable of adapting to their environments. Furthermore, the incorporation of the face detection feature improves the robot's capacity to respond to individuals, rendering it appropriate for use in personal security, surveillance, or human-robot interaction contexts.

This robotics project demonstrates that a highly interactive and autonomous robot can indeed be achieved with relatively simple hardware and processing, thus convincing the potentiality of an intelligent robotic system operating in a natural environment. Its modularity offers ease in enhancements and modifications that can be made in the future, especially in enhancing the functionalities of the system for practical applications.

8 FUTURE SCOPE

Future improvements would be:

1. Mapping of the already traversed environment to allow for more efficient facial searching and avoid ‘seeing’ already known surfaces.
2. Once one face has been identified, move the robot such that it is perpendicular and centered with the face.
3. Optimize speed of facial reading to more quickly identify faces.
4. Upgraded battery system to allow for longer run times; and
5. Utilizing more position- and speed-accurate controllers when in manual mode.
6. Utilizing the husky lens camera for redundancy during the obstacle avoidance mode for better path planning.

To aid in the above, additional sensors would be required to track GPS coordinates. Due to the lack of free I/O on the existing controller, we would need to upgrade to a larger microcontroller.

9 GROUP CONTRIBUTION

Group Member	**Contribution**
Leo Chen	Hardware and Documentation
Priyam Gandhi	Software and Documentation
Juliusz Gasior	Hardware and Programming (HuskyLens & Car)
Mostafa Tehrani	3D Design & Hardware

10 REFERENCES

1. Angelo, R. (n.d.). *HuskyLens*. Retrieved from GitHub: https://github.com/HuskyLens/HUSKYLENSArduino

2. Freenove. (n.d.). Retrieved from Freenove: https://store.freenove.com/products/fnk0041