Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering M.Sc

EMBEDDED SYSTEMS DESIGN
Dr. BELAL SABABHA

SPRING SEMESTER 2023/2024

FINAL PROJECT: HAND GESTURE CONTROLLED ROBOT CAR WITH OBSTACLE DETECTION

Jumana Abu Hamad	*ID: 20228188* *Email: jum20228188@std.psut.edu.jo* *Major: M.Sc Electrical Engineering*
Omar Khader	*ID: 20228186* *Email: oma20228186@std.psut.edu.jo* *Major: M.Sc Electrical Engineering*

08, June 2024

Table of Contents

Abstract

The goal of this project is to develop and create an automobile robot that can be controlled via hand gestures. The HCS12 microcontroller will be used to read hand gestures from the accelerometer attached on the Arduino Uno board. These hand gestures will be relayed to the HCS12 via Bluetooth communication, which will then control the car's movement and speed. The primary goal of the project is to be able to move the car in all directions using hand gestures. Furthermore, a cruise control feature will be available, which can be enabled with precise hand movements. The automobile will also include an obstacle detection capability that will use an ultrasonic sensor to detect these obstacles.

1. Introduction

1.1 Project Idea

This project aims to design and build a hand gesture-controlled car robot. The HCS12 microprocessor will be utilized to read the hand gestures from the accelerometer mounted to the Arduino Uno board, these hand gestures will be sent via Bluetooth communication to the HCS12 which will in turn accordingly control the motion of the car. The main aim of the project is to be able to move the car in all directions through hand gestures. Moreover, a cruise control feature will be available and is automatically activated through PID controller. The obstacle detection feature will also be implemented onto the car using an ultrasonic sensor.

1.2 Project Objectives

- Develop a gesture-controlled robot car capable of interpreting hand gestures via accelerometer and capable of moving in all directions.
- Incorporate cruise control functionality to maintain a constant speed irrespective of terrain.
- Implementing Obstacle Detection for the car.

2 Background Information

In the present day, there is an apparent increase the number of physically unfit people who are uncappable of driving themselves or need some assistance is driving due to certain disabilities. Therefore, the first application of this project would be that is going to help these unfit people to drive around by simply using their hand gestures to control the movement of their car. Furthermore, the ability to control a car using small hand gestures can also serve to help in navigating small or tight places that humans cannot access themselves. Simple hand gestures can control the car which is equipped with a camera that can record a love video of the inaccessible area. This application is extremely useful in navigating destructed areas resulting from war or natural catastrophes. The hand gesture-controlled robot car can easily make its way into these areas to for example, help rescue teams look injured people under the rubbles.

3 Design & Methodology

3.1 List of Used Components & Costs

❖ Software Components:

- CodeWarrior IDE
- Model-Based Programming Software, i.e. MATLAB Simulink

- Hardware Components:

No.	Component	Price/JD	Quantity	Total Cost (JD)
1	Robot Car Chassis Kit	30	1	30
2	DC Motors and wheels	3.5	4	14
3	HCS12 Microprocessor	15	1	15

4	Arduino UNO Board	9	1	9
5	Hall effect sensor	0.5	2	1
6	Small Magnets	0.5	2	1
7	Batteries (9V and 1865)	1.5	1 (9V) 3 (1865)	4.5
8	Jumper Wires Bundle	0.1	20	2
9	Bluetooth/Wireless modules for Communication between the glove and the car	7	2	14
10	Battery Holders	2	2	4
11	DC Jacks	1	2	2
12	Switch	0.5	1	0.5
13	Ultrasonic Module	4	1	4
14	Accelerometer	6.5	1	6.5
15	180 degree - Servo Motor	3	1	3
Final Expected Total Cost (JD)				110.5

3.2 Mechanical Design

The prototype used in this project is composed of a 4 wheels car with four DC motors that include gearboxes within them. A servo motor is used to steer the ultrasonic sensor in 180-degree motion to scan for obstacles and stop the car when there is an obstacle Infront if it (obstacle avoidance feature). In addition to the HSCS12 microcontroller on the car with its on-board H-bridge, the following components have been attached to the car body using screws and adhesive tape: ultrasonic sensor (HC-SR04), servo motor, two hall effect sensors (near the wheels of the car) and three 1865 batteries used to power up the HCS12. The basic prototype is shown in Fig. 1 and the complete prototype with all the components is shown in figures 2-3.

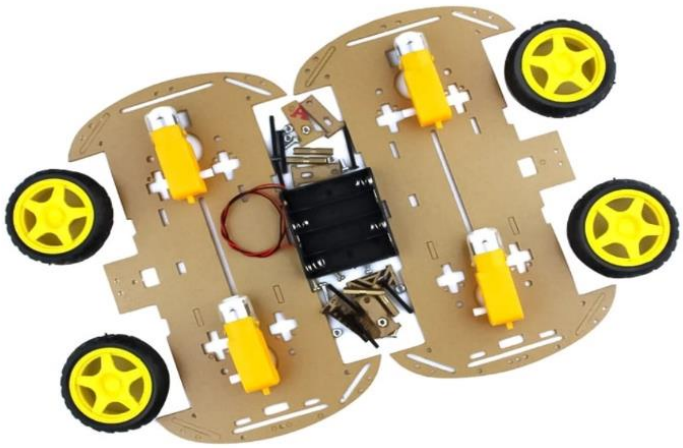

Figure 1: Basic Prototype of car without components

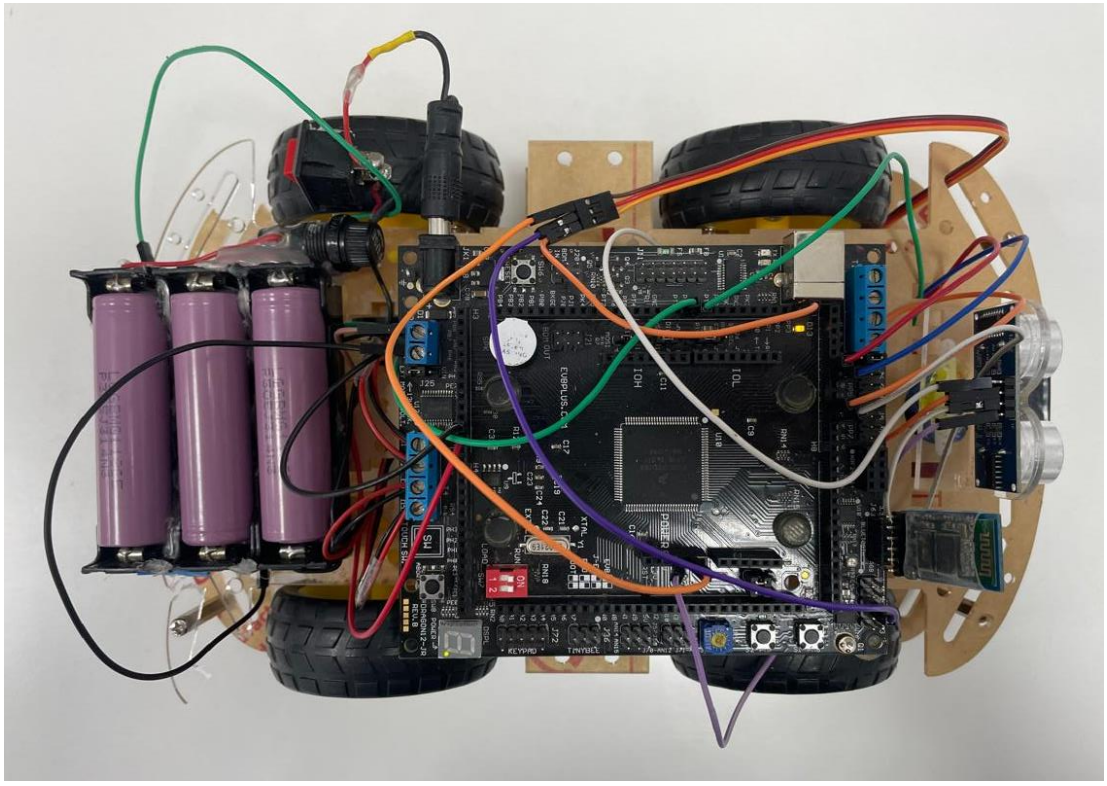

Figure 2: Complete prototype of car with all components.

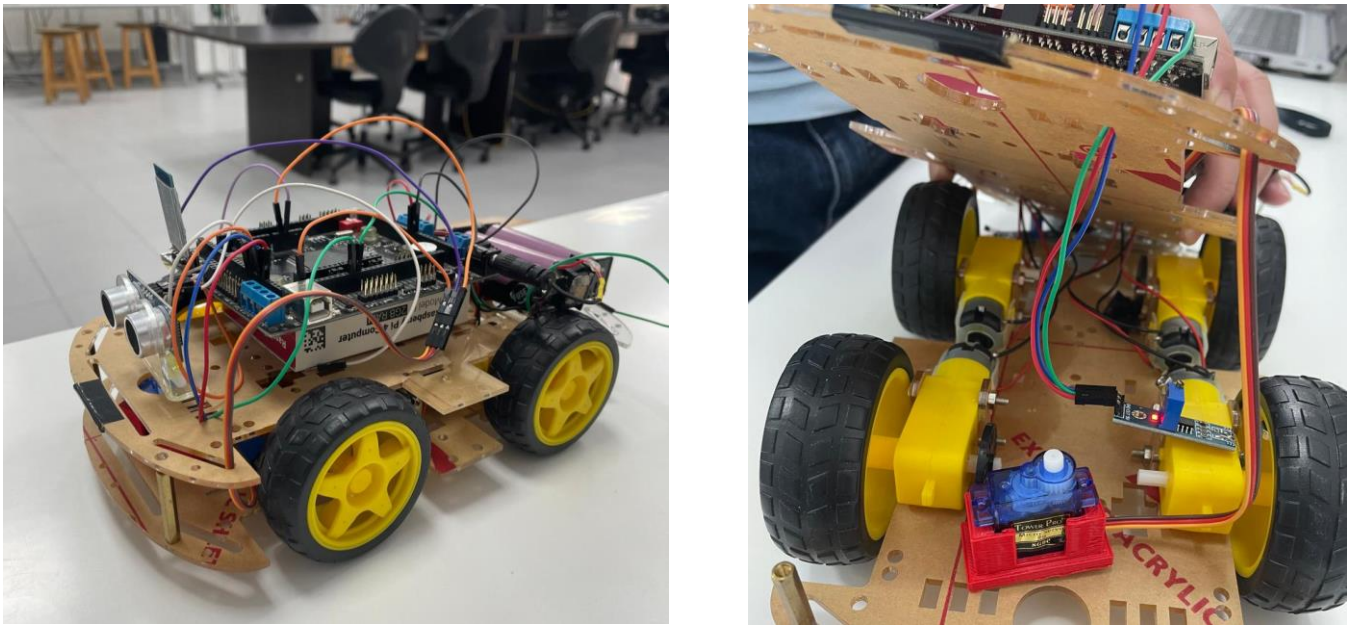

Figure 3: Complete prototype of car with all components (extra pictures).

3.3 Electrical Design

The electrical design for this project is displayed in figure 4. The main electrical connections are those of the HCS12 microcontroller and those of the Arduino Uno board. The main electrical components connected to the HCS12 are the following: H-bridge, ultrasonic sensor, servo motor, Bluetooth module for receiving, four DC motors, three lithium ion 1865 batteries and two hall effect sensors. On the other hand, the main electrical components connected to the Arduino Uno board are the following: 9v battery, accelerometer and a Bluetooth module for transmitting.

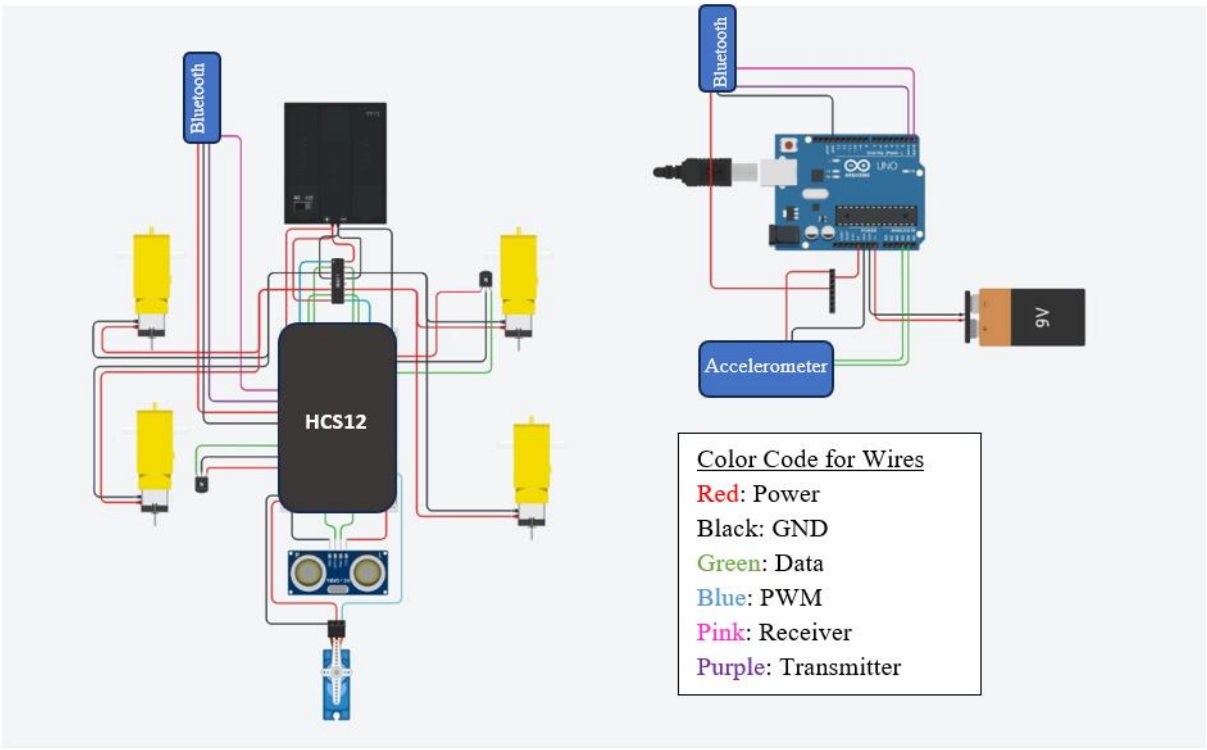

Figure 4: Electrical System Design.

3.4 Software Design

3.4.1 Code Explanation

The provided code (see appendix) is designed for an HCS12-based car system that can be controlled via hand gestures from a separate Arduino system with an Accelerometer to read those gestures and send them using Bluetooth. It includes functionalities for movement control, speed regulation using a PID controller, and obstacle detection using an ultrasonic sensor. The robot can perform various actions, such as moving forward, reversing, turning left, turning right, and stopping. It uses Pulse Width Modulation (PWM) to control the motors and a servo to direct the ultrasonic sensor.

Here is a detailed breakdown of the code:

1. System Initialization

- PLL Initialization (PLLinit()): Sets up the Phase-Locked Loop to stabilize and set the system clock speed for consistent operation.
- Motor Initialization (motorinit()): Configures the ports for motor control and calculates the top speed by counting initial pulses from the hall sensors.
- UART Initialization (initUART()): Sets up the UART for serial communication, enabling the robot to receive commands.
- PWM Initialization (initPWM()): Configures the Pulse Width Modulation channels for controlling motor speed and servo position, setting specific duty cycles and periods.
- Ultrasonic Sensor Initialization (initUltrasonic()): Configures the pins for the ultrasonic sensor to enable obstacle detection.

2. Movement Control

- PWM Duty Cycle Control (setPWMDutyCycle(char channel, int dutyCycle)): Adjusts the PWM duty cycle for a specified channel to control motor speed.
- Servo Position Control (setServoPosition(int angle)): Moves the servo to a specified angle, directing the ultrasonic sensor.
- Movement Functions (moveForward(int speed), moveReverse(int speed), turnLeft(int speed), turnRight(int speed), stop()): Control the robot's movement in various directions or stop it entirely.

3. Speed Regulation

- Speed Calculation (calculateSpeed()): Calculates the current speed of the wheels based on pulse counts from hall sensors.
- PID Control Update (updatePID()): Uses a PID controller to adjust the motor speed and maintain the target speed. It calculates the error between the target and current speeds and adjusts the motor output accordingly.

4. Obstacle Detection

- Distance Measurement (measureDistance()): Measures the distance to an obstacle using the ultrasonic sensor by sending a trigger pulse and measuring the echo duration.
- Obstacle Checking (checkObstacle()): Sweeps the ultrasonic sensor to check for obstacles within a specified distance threshold. If an obstacle is detected, the robot stops.

5. Command Handling via UART

- UART Receive ISR (interrupt 20 receive(void)): Interrupt service routine that handles commands received via UART. It processes the command to control the robot's movement, adjusts the target speed, and calls functions to update speed and check for obstacles.

6. Hall Sensor Interrupts

- Hall Sensor ISRs (interrupt 9 hallSensorISR1(void), interrupt 10 hallSensorISR2(void)): Count pulses from the hall sensors to measure the speed of each wheel. These counts are used in speed calculation and PID control.

7. Timing Control

- Timer Overflow ISR (interrupt 16 handler2()): Handles timer overflow interrupts for additional timing control, such as measuring durations or generating periodic events.

8. Utility Functions

- Delay Function (myDelay()): Provides a delay loop for timing purposes, such as waiting for the servo to move to a new position.

Summary of Functionality

- Movement Control: The robot can move forward, reverse, turn left, turn right, and stop based on commands received via UART.
- Speed Regulation: The robot maintains a target speed using a PID controller, adjusting the motor output to match the desired speed.
- Obstacle Detection: The robot uses an ultrasonic sensor to detect obstacles and stops if an obstacle is within a certain distance.
- Command Handling: The robot receives and processes commands via UART to control its movements and other functions.
- Speed Measurement: The robot uses hall sensors to measure the speed of its wheels for feedback in the PID control loop.

3.4.2 Software System Flowchart

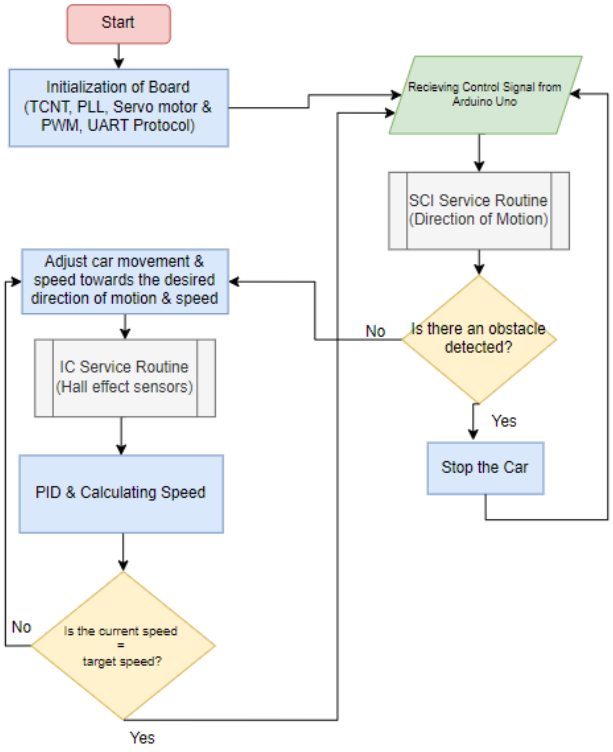

Figure 5: Software System Flowchart.

3.4.3 System Interrupts Flowchart

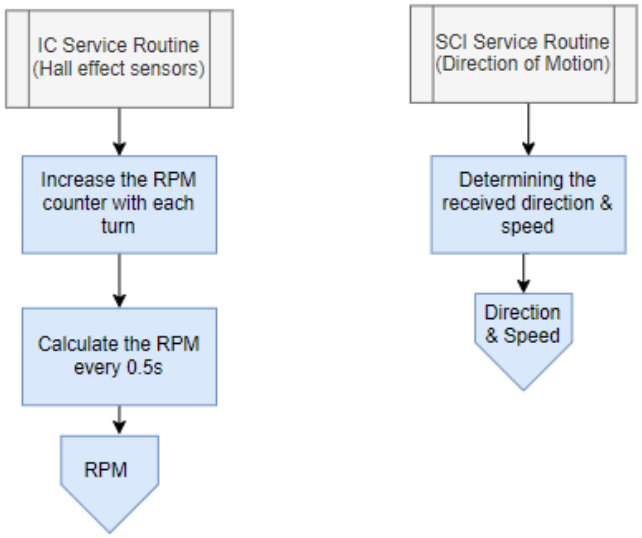

Figure 6: Interrupt Service Routines Flowchart.

3.4.4 Accelerometer MATLAB Simulink Model & Transmitted Data Output

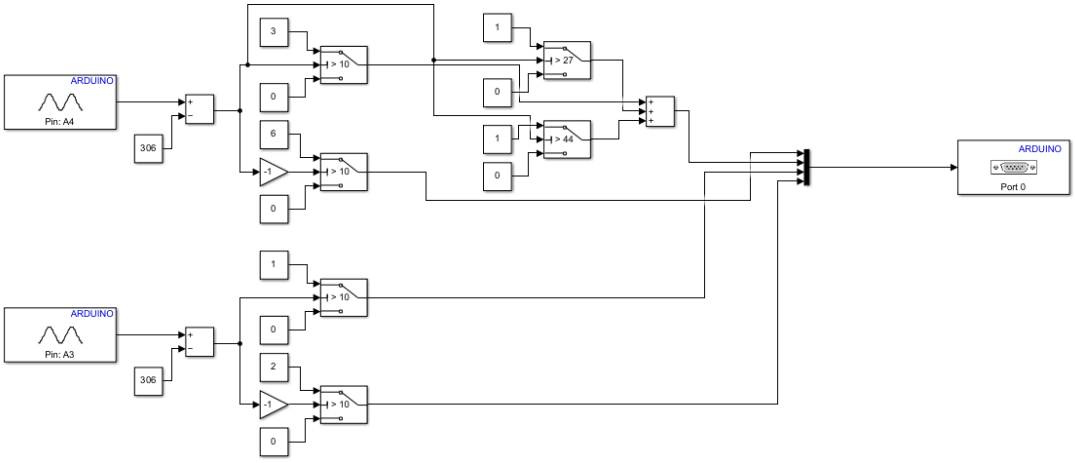

Figure 7: Accelerometer MATLAB Simulink Model.

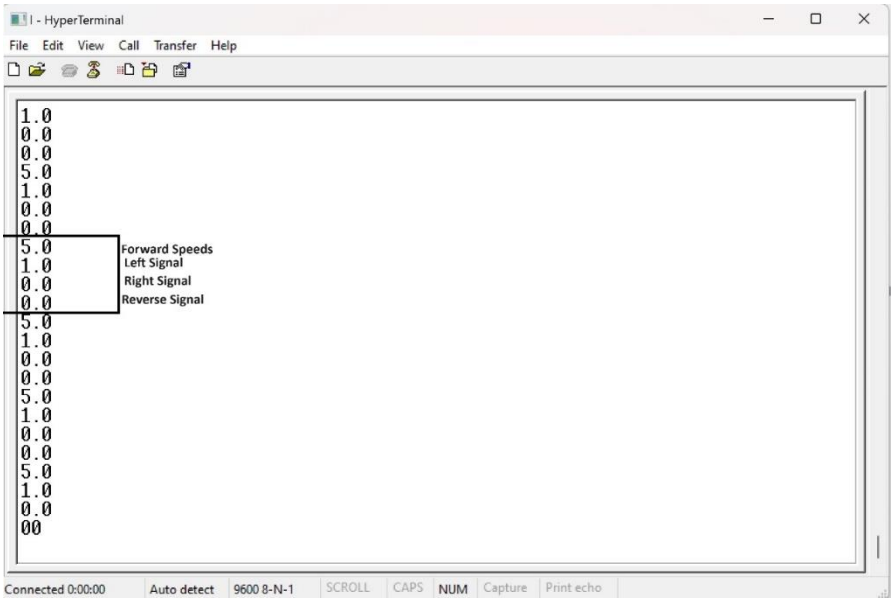

Figure 8: Accelerometer Transmitted Data Output on Terminal Port.

The code first collects analogue data from an accelerometer, measuring both the x and y axes to capture hand movements. These raw values are then processed to fit within a standardized range, typically between -62 and 62, ensuring uniformity across different movements. Next, through a series of conditional checks, the system determines the direction of the hand movement based on these processed values. For instance, if the x-axis reading suggests a tilt to the left and the y-axis indicates forward motion, the system infers a combined action of turning left while advancing. Each potential movement is associated with a specific control value, where 0 represents no action or a complete stop, 1 denotes a leftward turn, 2 signifies a rightward turn, and 3 through 6 corresponds to various speeds and directions of forward and backward motion. These control

signals are then serially sent through the Bluetooth module to the HCS12 MCU to be translated into actions on the car's movement.

3.4.5 Bluetooth Communication Pairing

To establish a secure and reliable point-to-point communication link using two HC-05 Bluetooth modules, one module was configured as the master and the other as the slave. The wiring involved connecting the VCC and GND of each module to the Arduino's 5V and GND, respectively, and connecting the modules' TX and RX pins to the Arduino's RX and TX pins. The Arduino's RESET pin was connected to GND to allow direct serial communication via the serial monitor without running any Arduino sketches.

For the master module, the AT commands set its role to master (AT+ROLE=1), renamed it to "Master" (AT+NAME=Master), set a password (AT+PSWD=1234), restricted connections to a specific address (AT+CMODE=0), and bound it to the slave's MAC address (AT+BIND=<Slave_MAC_Address>), with the baud rate configured to 9600 (AT+UART=9600,0,0). Similarly, the slave module was set to slave mode (AT+ROLE=0), renamed to "Slave" (AT+NAME=Slave), given the same password, and restricted to connections from the master only. After powering on both modules, the master automatically connected to the slave; this can be noticed when they exit the discovery mode and start blinking in sync immediately after powering up. After configuration, the slave module was connected to the receiving end (HCS12 MCU) and the master module on the transmission end of the project (Arduino Uno).

4 Problems & Recommendations

In this project, several problems were encountered, some of them were solved intime, however some remained unsolved due to time constraints & limits. The following problems remain in the project:

- The flex sensor was supposed to be used for interpreting the hand gestures, however, the two flex sensors that we intended to use were corrupted and damaged and were not displaying any correct readings. As a result, we switched to an external accelerometer attached to the Arduino Uno to interpret the hand gestures and control the movement in all directions.
- We faced difficulties in initiating and controlling the servo motor that was intended for moving the ultrasonic sensor to be able to detect and avoid obstacles. If there was more time available, a second servo motor could have been tested as the one used might be in a faulty condition.
- The signals interpreted from the hand gestures through accelerometer were correctly being transmitted from the Uno board to the HCS12. This was checked by using the terminal port to view the transmitted signals and they were in fact correctly being transmitted as required. However, these signals were not being received correctly by the HCS12 and thus the car did not move as intended. To ensure that the issue was purely a receiving signal issue, the car was tested independently to move in the four directions in all speeds and it did so successfully. Thus, the issue is purely a communication issue (receiving signal problem) between the Bluetooth modules.

5 Conclusion

The proposed gesture-controlled robot car project offers an exciting opportunity to explore embedded systems, microprocessor programming, and sensor technology. By using an accelerometer and advanced control algorithms, the project aims to develop a functional prototype capable of intuitive and efficient hand gesture control. Additional features such as cruise control and obstacle avoidance are included to further enhance the robot car's capabilities, making it a versatile and practical application of embedded systems technology. Overall, this project was successful at building the car porotype and ensuing that it moves and successfully transmitting the accelerometer hand gesture movement signals. Several problems where encountered, however due to time constraints some of these problems were solved in time and others were not. This project can be further expanded in the future to include a live camera recording feed where the car can be moved and can simultaneously record the scenery around it.

6 Appendix

❖ Main Code for the Project:

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */

#define FORWARD1  '3.0'
#define FORWARD2  '4.0'
#define FORWARD3  '5.0'
#define REVERSE  '6.0'
#define LEFT     '2.0'
#define Right    '1.0'
#define CRUISE   '7'
#define STOP     '0.0'

#define TRIGGER_PIN  PTM_PTM2   // Ultrasonic trigger pin
#define ECHO_PIN     PTM_PTM3   // Ultrasonic echo pin
#define SERVO_PIN    PTM_PTM4   // Servo control pin

#define TRIGGER_DELAY 10   // Trigger pulse width in microseconds
#define SOUND_SPEED 343   // Speed of sound in m/s
#define DIST_THRESHOLD 5 // Distance threshold in cm

// Function prototypes
void interrupt 20 receive(void);
void initUART(void);
void initPWM(void);
void initTimer(void);
void initUltrasonic(void);
void RTI_init(void);
void setPWMDutyCycle(char channel, int dutyCycle);
void setServoPosition(int angle);
void moveForward(int speed);
void moveReverse(int speed);
void turnLeft(int speed);
void turnRight(int speed);
void stop(void);
void cruiseControl(void);
void interrupt 7 calculateSpeed(void);
void updatePID(void);
```

```
unsigned int measureDistance(void);
void checkObstacle(void);
void myDelay2 (unsigned int val);

void interrupt 9 hallSensorISR1(void);
void interrupt 10 hallSensorISR2(void);
void motorinit(void);
void PLLinit(void);
void myDelay(void);

// Global variables for PID co0100;

volatile int currentSpeed1 = 0;
volatile int currentSpeed2 = 0;
volatile int pulseCount1 = 0;
volatile int pulseCount2 = 0;
volatile char receivedCommand = STOP;
int topspeed=0;
int targetSpeed=0;
int Kp = 1, Ki = 0, Kd = 0;
int error, previousError = 0, integral = 0, derivative;
int pidOutput;
int overflow;
int flag;
unsigned int j;
void interrupt 16 handler2(){     //interrupt service routine for the TCNT overflow interrupt
    if(flag==1){
      TFLG2=0x80; // clear the TOF flag
      overflow++;
    }

}
void main(void)
{
  PLLinit();
  motorinit();
  initUART();
  initPWM();
  RTI_init();
  initUltrasonic();
  EnableInterrupts;
```

```
  overflow=0;
  flag=0;
  TSCR1=0x90; // enable the TCNT counter and Fast Flag clear
  TSCR2=0x05; // disable the overflow interrupt and set the prescale to 32
  TIOS=TIOS & 0xF9;
  TCTL4=0x28; // latch the counter on the rising edge of IC1
  TFLG1=0x06; // clear the IC1 flag
  TIE=0x06; // enable IC1 & IC2 Interrupt
  for(;;) {

  }
}

void RTI_init(void){
RTICTL=0x7F;
CRGINT = CRGINT | 0x80;// enabled the RTI interrupt
}
void PLLinit(void){
   SYNR=0x02;
   REFDV=0x01;
   PLLCTL=0x60;
   while(!(CRGFLG & 0x08));
   CLKSEL=0x80;
}

void motorinit(void){
  DDRB=0xFF;
  PORTB=0x0A;
  PTP=0x03;
  myDelay2(500);
  topspeed = ((pulseCount1 + pulseCount2)/2)*120;
  PTP = 0x00;
  PORTB = 0x00;
}
void initUART(void) {

    SCI0CR1=0x00;
    SCI0CR2=0x24;
    SCI0BDL=156;//9600 baud
    SCI0BDH=0x00;      // Return received byte
}
```

```
void initPWM(void) {
  PWMCLK=0x03; // select SA
  PWMPOL=0x0F; // Start high (1 polirity)// we are using a nMOS
  PWMPRCLK=0x47; //prescale the E-clock by 128 --> 24M/128 = 187.5 Khz
  PWMSCLA=0x04;//  prescale A by 8 (4*2) to get SA --> 187.5KHz/8 = 23.43KHz
  PWMCTL=0x2C; // select 8-bit mode and disable PWM in wait and freeze modes
//i.e each count will take this amount of time: (1/24MHz)*128*8 = 42.666 microseconds

  PWMPER0=235; // this will get us 10.02 ms period (10.02 ms =235 counts * 42.666 micro Second
per count)
  PWMDTY0=235/2; // 50% duty cycle (initially)
  PWMPER1=235; // this will get us 10.02 ms period (10.02 ms =235 counts * 42.666 micro Second
per count)
  PWMDTY1=235/2; // 50% duty cycle (initially)
  PWMPER2=30000>>8; //highest 8 bit of the Period
  PWMPER3=30000 & 0x00FF; // lowest 8 bit of the period


  // start with the servo pointing 0 degrees
  PWMDTY2=2000>>8; // highest 8 bit of the duty
  PWMDTY3= 2000 & 0x00FF; //lowest 8 bit of the duty
  PWMCNT3=0x00;
  PWMCNT2=0x00;
  PWME=0x0F;
}

void initTimer(void) {
  TSCR1 = 0x80;      // Enable timer and fast flag clear
  TSCR2 = 0x06;      // Enable timer interrupt, prescale by 64
  TIOS |= 0x01;      // Output compare on channel 0
  TC0 = 37500;       // Set up timer compare value for 100ms (assuming 24MHz clock)
  TIE |= 0x01;       // Enable interrupt for channel 0
}



void initUltrasonic(void) {
  DDRM |= 0x04;    // Set PTM2 (trigger) as output
  DDRM &= ~0x08;   // Set PTM3 (echo) as input
}
```

```
void setPWMDutyCycle(char channel, int dutyCycle) {
  int scaledDutyCycle = (dutyCycle * 255) / topspeed; // Scale top speed to 0-255
  switch(channel) {
    case 0: PWMDTY0 = scaledDutyCycle; break;
    case 1: PWMDTY1 = scaledDutyCycle; break;
   // case 2: PWMDTY2 = scaledDutyCycle; break;
   // case 3: PWMDTY3 = scaledDutyCycle; break;

    default: break;
  }
}

void setServoPosition(int angle) {
  int pulseWidth = 1000 + (angle * 1000 / 180); // Pulse width in microseconds (1000us to 2000us)
  int dutyCycle = (pulseWidth * 100) / 20000; // Convert to duty cycle percentage (0-100%)
  PWMDTY2=dutyCycle>>8; // highest 8 bit of the duty
  PWMDTY3= dutyCycle & 0x00FF; //lowest 8 bit of the
}

void moveForward(int speed) {
  // Set PWM duty cycle to control motor speed
  PORTB =0x0A;
  setPWMDutyCycle(0, speed); // Assume channel 0 controls forward movement
  setPWMDutyCycle(1, speed);     // Set reverse channel to 0
}

void moveReverse(int speed) {
  // Set PWM duty cycle to control motor speed
  PORTB =0x05;
  setPWMDutyCycle(0, speed);     // Set forward channel to 0
  setPWMDutyCycle(1, speed); // Assume channel 1 controls reverse movement
}

void turnLeft(int speed) {
  // Implement motor control logic to turn left
  PORTB =0x0A;
  setPWMDutyCycle(0, speed); // Example: turn left at 50% speed
  setPWMDutyCycle(1, speed/2);
}

void turnRight(int speed) {
```

```
  // Implement motor control logic to turn right
  PORTB =0x0A;
  setPWMDutyCycle(0, speed/2); // Example: turn left at 50% speed
  setPWMDutyCycle(1, speed);
}

void stop(void) {
  // Implement motor control logic to stop

  setPWMDutyCycle(0, 0);
  setPWMDutyCycle(1, 0);



}

void cruiseControl(void) {
  // Implement logic to set the cruise control speed
  // For simplicity, toggle between a set speed and stop
  if (targetSpeed == 0) {
    targetSpeed = topspeed/2; // Set to desired cruise speed
  } else {
    targetSpeed = 0;  // Reset to 0 to stop
  }
}

void interrupt 7 calculateSpeed(void) {    // RTI that triggers each 22ms
  // Calculate the wheel speeds based on pulse counts
  // Reset pulse counts after calculation
  currentSpeed1 = pulseCount1*2728; // You can scale this as needed
  currentSpeed2 = pulseCount2*2728;
  pulseCount1 = 0;
  pulseCount2 = 0;
  j++;
}

void updatePID(void) {
  // Calculate the PID control value
  error = targetSpeed - ((currentSpeed1 + currentSpeed2) / 2);
  integral += error /10; // Integral term (0.1 represents the time interval in seconds)
  derivative = (error - previousError) *10; // Derivative term
  pidOutput = (int)(Kp * error + Ki * integral + Kd * derivative);
  previousError = error;
```

```
  // Ensure PID output is within valid range (0-100%)
  if (pidOutput > 100) pidOutput = 100;
  if (pidOutput < 0) pidOutput = 0;

  moveForward(pidOutput); // Adjust motor speed based on PID output
}

unsigned int measureDistance(void) {
  unsigned int duration, distance;
  // Send trigger pulse
  TRIGGER_PIN = 1;
  myDelay2(TRIGGER_DELAY);
  TRIGGER_PIN = 0;

  // Wait for echo start
  while (!ECHO_PIN);
  // Measure echo duration
  TFLG1 = 0x01; // Clear timer overflow flag
  TCNT = 0;
  while (ECHO_PIN && !TFLG1) duration = TCNT;

  // Calculate distance in cm
  distance = (duration * SOUND_SPEED) / (2 * 10000); // Convert to cm
  return distance;
}

void checkObstacle(void) {
  unsigned int distance;
  int angle;
  for (angle = 0; angle <= 180; angle += 10) {
    setServoPosition(angle);

    myDelay2(100);

    distance = measureDistance();
    if (distance < DIST_THRESHOLD) {
      stop();
      break;
    }
  }
}
```

```
void interrupt 20 receive(void){
    receivedCommand = SCI0DRL;// Read the command from UART

 // Process the received command
 switch(receivedCommand) {
  case FORWARD1:
   targetSpeed = topspeed/3;
   break;
  case FORWARD2:
   targetSpeed = (topspeed*2)/3;
   break;
  case FORWARD3:
   targetSpeed = topspeed;
   break;
  case REVERSE:
   moveReverse(topspeed/3); //
   break;
  case LEFT:
   turnLeft(topspeed/3);
   break;
  case Right:
   turnRight(topspeed/3);
   break;
  case CRUISE:
   cruiseControl();
   break;
  case STOP:
   stop();
   targetSpeed = 0;
   break;
  default:
   stop();
   targetSpeed = 0;
   break;
 }


 updatePID();
 checkObstacle(); // Check for obstacles
}
```

```
void interrupt 9 hallSensorISR1(void) {
  pulseCount1++; // Increment pulse count for first wheel
  TFLG1 = 0x02;  //clear flag
}

void interrupt 10 hallSensorISR2(void) {
  pulseCount2++; // Increment pulse count for second wheel
  TFLG1 = 0x04;  //clear flag
}

void myDelay(void){
    unsigned char i;
    unsigned int j;

    for(i=0;i<20;i++){
        for(j=0;j<20000;j++){

            i=i;
            j=j;
        }



    }

}

void myDelay2 (unsigned int val){
 j=0;
 while(j*22<val);
}
```