Ankara University
Faculty of Engineering

Electrical and Electonics Enginering

EEE3330 Embedded Systems Project

Yusuf Emre BAYSAL
18290193

PROJECT NAME

Throttle Control and Motor RPM Sensor with Car Dial

ABSTRACT

In this project, operation of Controller Area Network System is demonstraded by using Arduino microcontrollers. There are 3 Arduino-Canbus Module system sets in the system, each set consists of 1 Arduino Microcontroller, 1 CANBUS-SPI Module and necessary sensors and/or modules.

First set is consists of a potantiometer, Arduino Nano Clone and MCP2515 CANBUS-SPI Module. The purpose of the system is to demonstrate car throttle system. The second set is consist of an Infrared RPM sensor, Arduino Nano Clone and MCP2515 CANBUS-SPI Module. The purpose of the system is to demonstrate motor RPM sensor. The third set is consist of an Arduino Mega 2560 R3 Clone, a DC motor, a servo motor and 2x16 LCD display. The purpose of systems is to demonstrate the car motor controller and a show speed and RPM in a car dial.

1. INTRODUCTION

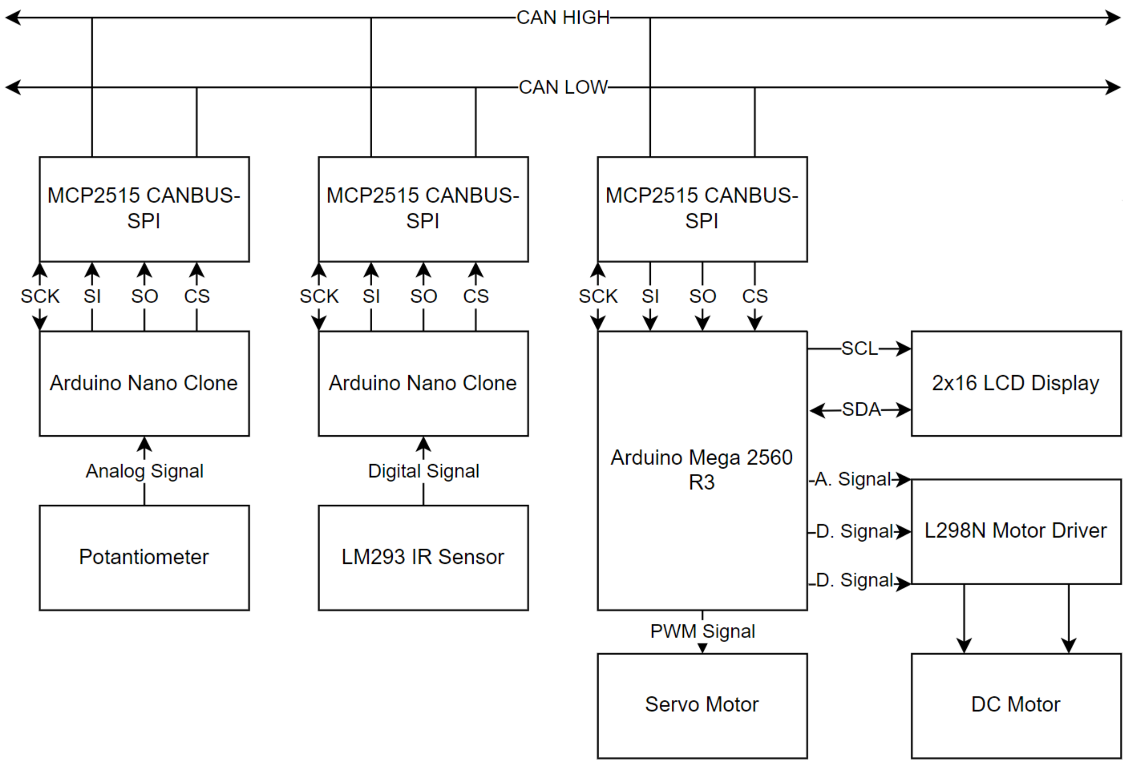
1.1. Purpose and Aim

In this project we have aimed to learn the use of the Controller Are Network system by designing a mini car Controller Area network system prototype, consisting o several sensors, modules, motors, microcontrollers, and CANBUS-SPI Communication Modules. The reason for choosing the car throttle, car RPM sensor, motor, and car dial system is that these parts are one of the most important parts of the car system.

1.2. Method

The method of the project is building an Arduino system with breadboards and jumper cables and writing the necessary codes to let the system works properly.

1.3. Schematic of Project



1.4. Time Planning Chart

	1 Unit = 1 Week				
Design Stage	1	2	3	4	5
Learn to use CANBUS	▨	▨			
Development of Sensors		▨			
Development of Car Dial and DC Motor System			▨		
Connecting Systems with CANBUS				▨	
Testing					▨

2. Theoretcial Backgrund

2.1. Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs and turn it into an output.[1]

2.2. CANBUS-SPI Module

The CAN module handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate message buf- fer and control registers. Transmission is initiated by using control register bits via the SPI interface or by using the transmit enable pins.

2.3. DC Motor

A DC motor is an electrical machine that converts electrical energy into mechanical energy. In a DC motor, the input electrical energy is the direct current which is transformed into the mechanical rotation.[2]

2.4. LCD Display

LCD (Liquid Crystal Display) is a type of flat panel display which uses liquid crystals in its primary form of operation.[3]

2.5. Servo Motor

A servomotor (or servo motor) is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity, and acceleration. It consists of a suitable motor coupled to a sensor for position feedback.[4]

2.6. Potentiometer

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.[1] If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.[5]

2.7. Infrared RPM Sensor

The IR sensor module consists of IR Transmitter & Receiver in a single pair that can work a Digital Tachometer for speed measurement of any rotating object. The Tachometer is an RPM counter which counts the no. of rotation per minute.[6]

3. DESIGN

3.1. General Informations

In this project, the purpose was to demonstrate the operation of the Controller Area Network. The system is consist of three sets, each containing an Arduino microcontroller, an MCP2515 CANBUS-SPI Module, and necessary sensors or modules. In the code, the “mcp2515.h” library is used. Arduino microcontroller can communicate with other devices with Serial Peripheral Interface, or simply SPI. So, the message is sent to MCP2515 CANBUS-SPI Module with SPI communication protocol, and the MCP2515 microchip inside the MCP2515 CANBUS-SPI Module transforms SPI communication protocol to CANBUS communication protocol. Then, the data package is sent via CAN HIGH and CAN LOW pins. In order to choose proper communication addresses, the J-1939 Fault Code Source Protocol is used.

The power is supplied to all microcontrollers and modules from a two 1500mAh 18650 Li-Ion rechargeable connected in series within a double battery bed. The output voltage of the battery bed is 7.4 volts and it is connected directly to an L298N Motor Driver Board and a voltage regulator in order to change the voltage level to 5 volts.

The first set is a transmitter system that consists of an Arduino Nano R3 clone, an MCP2515 CANBUS-SPI Module, and a potentiometer. In this design, the potentiometer was acting as a throttle system of a car. The microcontroller inside the Arduino nano processes the information coming from the potentiometer and transforms the data into 8-bit data packages. The message data structure consists of three adjustable variables. First is data ID. According to J-1939 Fault Code Source, the address “Accelerator Pedal Position” is decimal “91”, or hexadecimal “5B”, therefore; the address of the throttle set is chosen as 0x05B. The second adjustable variable is the data length code, it specifies how many bytes of data are sent with each CANBUS data package. For this set, the analog inputs of the Arduino microcontroller can read 10 bits of data, and these 10 bits of incoming data are transformed into 8 bits of data. Data loss during this process is not significant and can be easily ignored. So, 1 byte of data is enough for this process and the data length code is chosen as 1. The third adjustable variable is the message is being sent. So, the mapped value of the incoming potentiometer value is sent to MCP2515 CANBUS-SPI Module.

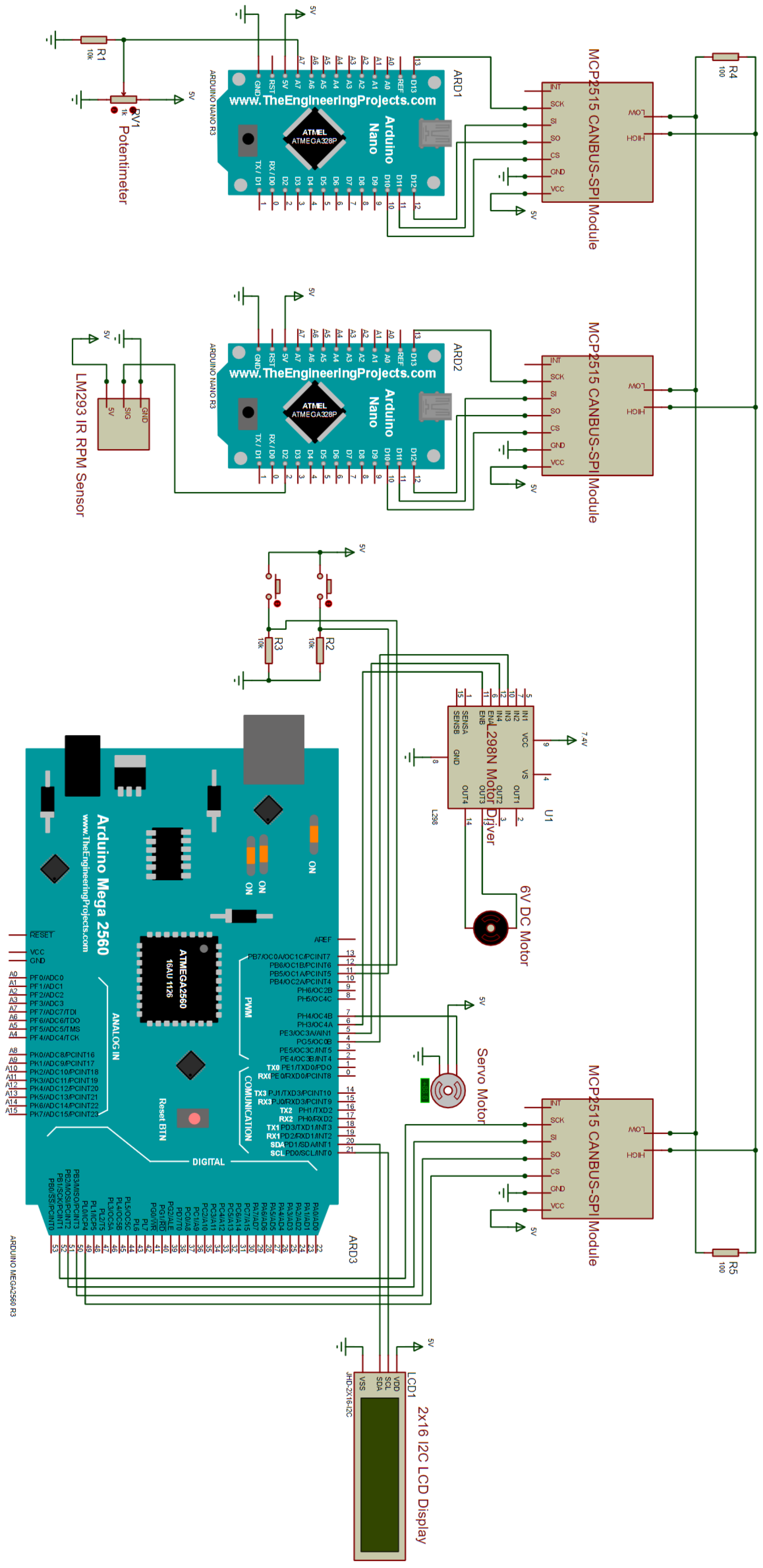
The second set again a transmitter system that consists of an Arduino Nano R3 clone, an MCP2515 CANBUS-SPI Module, and an LM293 IR RPM Sensor. In this design, the LM293 IR RPM Sensor was acting as a speedometer system of a car. The microcontroller inside the Arduino nano processes the information coming from the potentiometer and transforms the data into 8-bit data packages. Again, the message data structure consists of three adjustable variables. First is data ID. According to J-1939 Fault Code Source, the address "Wheel-Based Vehicle Speed" is decimal "84", or hexadecimal "54", therefore; the address of the RPM set is chosen as 0x054. The second adjustable variable is the data length code, it specifies how many bytes of data are sent with each CANBUS data package. For this set, the analog inputs of the Arduino microcontroller can read 10 bits of data, and these 10 bits of incoming data are transformed into 8 bits of data. Data loss during this process is not significant and can be easily ignored. So, 1 byte of data is enough for this process and the data length code is chosen as 1. The third adjustable variable is the message is being sent. So, the mapped value of the incoming LM293 IR RPM Sensor value is sent to MCP2515 CANBUS-SPI Module.

The third set is a receiver system that consists of an Arduino Mega 2560 R3 clone, an MCP2515 CANBUS-SPI Module, a 16x2 I2C LCD Display, a servo motor, an L298N Motor Driver Board, a 6V DC motor, and 2 buttons. In this set, the Arduino Mega works like 2 core microcontroller, with each core being given different tasks. In the first core, the 6V DC motor acts as a motor for the car, and this DC motor is controlled via the L298N Motor Driver Board. The Arduino microcontroller controls the L298N Motor Driver Board with PWM pins, and the L298N Motor Driver Board controls the supplied DC motor voltage. The speed of the motor is set according to data coming from the throttle system, which is a potentiometer inside set 1. In the second core, the 16x2 I2C LCD Display and the servo motor was acting as the car dial system of a car. Servo motor acts as analog RPM dashboard and LCD Display acts like a digital display of cars, displaying both RPM and speed of the car. The RPM value is obtained from the data coming from the speedometer system, which is an LM293 IR RPM Sensor inside the set 2. 2x16 LCD Display has an I2C connector and communicates with the Arduino with I2C protocol. The speed calculations are made according to the radius of the wheel to which the DC motor is connected. The remaining 2 buttons are for error detection. They are used to stop the engine from running and to enable the CANBUS system to start the engine and communicate with each other without making any noise. The microcontroller inside the Arduino Mega processes the information coming from MCP2515 CANBUS-SPI Module and runs a motor system, which is a 6V DC motor, and the car dial system, which is a 16x2 I2C LCD Display and a servo motor.

3.2. Materials Used in the Project and Prices:

Device	Quantity	Unit Price	Total Price
Arduino Mega 2560 R3 Clone	1	₺ 285,26	₺ 285,26
Arduino Nano R3 Clone	2	₺ 132,48	₺ 264,96
MCP2515 CANBUS-SPI Module	3	₺ 75,60	₺ 226,80
L298N Motor Driver Board	1	₺ 36,58	₺ 36,58
DC Motor and Tire Set	1	₺ 19,25	₺ 19,25
2x16 LCD Display with I2C module	1	₺ 61,21	₺ 61,21
Servo Motor	1	₺ 33,61	₺ 33,61
LM293 IR Sensor	1	₺ 15,93	₺ 15,93
Buzzer	1	₺ 7,00	₺ 7,00
Potentiometer	1	₺ 3,68	₺ 3,68
Disc Encoder	1	₺ 1,31	₺ 1,31
Double 18650 Battery Bed	1	₺ 9,44	₺ 9,44
18650 3,7V 1500mAh Li-ion Battery	2	₺ 25,85	₺ 51,70
Breadboard	1	₺ 21,00	₺ 21,00
Jumper Cable Set	3	₺ 19,08	₺ 57,24
Power Supply Module	1	₺ 18,30	₺ 18,30
Total			₺1113,27

3.3. Wiring Diagram of the System



3.4. Codes

3.4.1 Transmitter Throttle Code (Set 1)

```
// Transmitter Throttle (Potentiometer)

// define necessary libraries
#include <mcp2515.h>
#include <SPI.h>
#define SPI_CS_PIN 10

// define mcp 2515 object
MCP2515 mcp2515(SPI_CS_PIN);

// define variables
struct can_frame canMsg1;
int potVal;
int mappedPotVal;

void setup()
{
  // canbus message
  canMsg1.can_id  = 0x05B;
  canMsg1.can_dlc = 1;
  canMsg1.data[0] = 0x00;

  while (!Serial);
  Serial.begin(9600);

  // canbus settings
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS);
  mcp2515.setNormalMode();

  pinMode(A7, INPUT);
}

void loop()
{
  potVal = analogRead(A7);    // read potentiometer income value
  mappedPotVal = map(potVal, 0, 1030, 0, 255); // map incoming 10 bit value to 8 bit
value
  if (mappedPotVal > 255) {
    mappedPotVal = 255;
  }

  canMsg1.data[0] = mappedPotVal;
  mcp2515.sendMessage(&canMsg1);  // send data

  delay(10);
}
```

3.4.2 Transmitter RPM Sensor Code (Set 2)

```
// Transmitter 2 RPM

// define necessary libraries
#include <mcp2515.h>
#include <SPI.h>
#include <TimerOne.h>

// define necesary variables
#define SPI_CS_PIN 10
#define encoderPin 2

// define mcp 2515 object
MCP2515 mcp2515(SPI_CS_PIN);
struct can_frame canMsg1;

// RPM calculation variables
unsigned int rpm; // rpm variable
volatile byte pulses; // signal counter
unsigned long timeold;
unsigned int pulsesperturn = 1;

// define necesary variables
int normalizedRPM = 0;

int RPMarray[15] = {};
int averageRPM;

// pulse counter
void counter()
{
  //sayımı arttır
  pulses++;
}


void setup()
{
  // canbus message
  canMsg1.can_id  = 0x054;
  canMsg1.can_dlc = 1;
  canMsg1.data[0] = 0x00;

  while (!Serial);
  Serial.begin(96000);

  // canbus settings
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS);
  mcp2515.setNormalMode();

  // rpm sensor settings
  attachInterrupt(0, counter, FALLING);
  // start
  pulses = 0;
  rpm = 0;
  timeold = 0;
}

void loop()
{
  // calculate the rpm
  if (millis() - timeold >= 100) {
    detachInterrupt(0);
    rpm = ((60 * 100 / pulsesperturn ) / (millis() - timeold) * pulses);
    timeold = millis();
    pulses = 0;
    //Serial.print("RPM = ");
    //Serial.println(rpm, DEC);
    //restart
    attachInterrupt(0, counter, FALLING);
  }

  // map rpm value to 8 bit
  normalizedRPM = map(rpm, 0, 6000, 0, 255);
  if ( normalizedRPM > 255) {
    normalizedRPM = 255;
  }

  canMsg1.data[0] = normalizedRPM;
  mcp2515.sendMessage(&canMsg1);  // send data

  delay(10);
}
```

3.4.3 Receiver Car Dial Code (Set 3)

```
// Receiver Car Dial (Motor, LCD Display and Servo Motor)

// include necessary libraries
#include <SPI.h>
#include <mcp2515.h>

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

```
// LCD Libaries
#include<Wire.h>
#include<LiquidCrystal_I2C.h>

// define necessary variables
#define SPI_CS_PIN 49
#define servo

#define topButton 11
#define bottomButton 12

bool errorState = false;

int potValReceived = 0;   // incoming potentimeter value
int rpmValReceived = 0;   // incoming rpm sensor value

int Rpwm_pin = 6;     //pin of controlling speed---- ENB of motor driver board
int pinRB = 4;        //pin of controlling diversion----IN3 of motor driver board
int pinRF = 5;        //pin of controlling diversion----IN4 of motor driver board

unsigned char motorSpeed = 100;

// tru rom value variables
int RPMarray[25] = {};
int averageRPM = 0;
int correctedRPM = 0;

struct can_frame canMsg;

// create objects
MCP2515 mcp2515(SPI_CS_PIN);
Servo myservo;
LiquidCrystal_I2C lcd( 0x27, 16, 2);

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

  // canbus module settings
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS);
  mcp2515.setNormalMode();

  // define connected servo pin
  myservo.attach(7);

  pinMode(pinRB, OUTPUT);      // pin 7--IN3 of motor driver board
  pinMode(pinRF, OUTPUT);      // pin 8--IN4 of motor driver board
  pinMode(Rpwm_pin, OUTPUT);   // pin 10 (PWM) --ENB of motor driver board
  pinMode(topButton, INPUT);   // button pin
  pinMode(bottomButton, INPUT); // button 2 pin

  // lcd settings
  lcd.init();
  lcd.backlight();
}

void loop()
{

  // button controls for error
  if (digitalRead(topButton) == HIGH ) {
    errorState = false;
    potValReceived = 0;
    rpmValReceived = 0;
    lcd.clear();
    delay(50);
  }
  if (digitalRead(bottomButton) == HIGH) {
    errorState = true;
    potValReceived = 0;
    rpmValReceived = 0;
    lcd.clear();
    delay(50);
  }
```

```
// Potantiometer Income Value
canMsg.can_id == 0x00;  // set adress to zero

// control if the message is received
if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {

  // control incoming message adress
  if (canMsg.can_id == 0x05B) {

    // display the incoming adress values to Serial Monitor
    //      Serial.print("CAN ID: ");
    //      Serial.print(canMsg.can_id, HEX); // print ID
    //      Serial.print(" ");
    //      Serial.print("CAN DLC: ");
    //      Serial.print(canMsg.can_dlc, HEX); // print DLC
    //      Serial.println(" ");
    //      Serial.println(canMsg.data[0]);
    //      for (int i = 0; i < canMsg.can_dlc; i++)  { // print the data
    //        Serial.print(canMsg.data[i], HEX);
    //        Serial.print(" ");
    //      }
    //      Serial.println();

    // Code Segment
    potValReceived = canMsg.data[0]; // received data

    // control the dc motor via motor driver
    if (errorState == false) {
      analogWrite(Rpwm_pin, potValReceived);
      digitalWrite(pinRB, HIGH);
      digitalWrite(pinRF, LOW);
    }

    // display the incoming canbus value and shut the motor for error detection
    else if (errorState == true) {
      if (millis() % 5 == 0) {
        analogWrite(Rpwm_pin, 0);
        digitalWrite(pinRB, LOW);
        digitalWrite(pinRF, LOW);

        lcd.setCursor(0, 0);
        lcd.print("Pot In:          ");
        lcd.setCursor(0, 0);
        lcd.print("Pot In: ");
        lcd.print(potValReceived);
        lcd.print(" ");
        lcd.print(potValReceived, HEX);
      }
    }
  }

  // RPM Sensor Income
  // control the incoming message adress
  if (canMsg.can_id == 0x054) {

    //      Serial.print("CAN ID: ");
    //      Serial.print(canMsg.can_id, HEX); // print ID
    //      Serial.print(" ");
    //      Serial.print("CAN DLC: ");
    //      Serial.print(canMsg.can_dlc, HEX); // print DLC
    //      Serial.print(" ");
    //
    //      for (int i = 0; i < canMsg.can_dlc; i++)  { // print the data
    //        Serial.print(canMsg.data[i], HEX);
    //        Serial.print(" ");
    //      }
    //      Serial.println();

    // Code Segment
    rpmValReceived = canMsg.data[0]; // received data

    // take the average of last 25 values of rpm to let the system work more stable
    for (int j = 0; j < 24; j++) {
      RPMarray[j] = RPMarray[j + 1];
    }
```

```
        RPMarray[24] = rpmValReceived;

        for (int j = 0; j < 24; j++) {
          averageRPM = averageRPM + RPMarray[j];
        }
        averageRPM = averageRPM / 25;

        myservo.write(map(averageRPM, 0, 255, 180, 0));   // adjust the servo motor
position
        correctedRPM = map(averageRPM, 0, 255, 0, 6000);  // decode the rpm

        // display the values to LCD display
        if (errorState == false) {
          if (millis() % 10 == 0) {
            lcd.setCursor(0, 0);
            lcd.print("RPM:        ");
            lcd.setCursor(0, 0);
            lcd.print("RPM: ");
            lcd.print(correctedRPM);

            lcd.setCursor(0, 1);
            lcd.print("Speed: ");
            lcd.print(correctedRPM * 0.213 * 0.06);
            lcd.print("km/h  ");
          }
        }
      }
    }
}
```

4. REFERENCES

Web sayfası

[1]. https://www.arduino.cc/en/Guide/Introduction

[2]. https://byjus.com/physics/dc-motor/

[3]. https://www.techtarget.com/whatis/definition/LCD-liquid-crystal-display

[4]. https://www.engineeringchoice.com/servo-motor/

[5]. https://en.wikipedia.org/wiki/Potentiometer

[6]. https://how2electronics.com/digital-tachometer-ir-sensor-arduino

[7]. https://wiki.dfrobot.com/CAN-BUS_Shield_V2__SKU__DFR0370_

[8]. https://www.senodes.com/docs/sensor-node/introduction/get-started/

[9]. https://www.robimek.com/lm-393-kizilotesi-hiz-sensoru-kullanimi/

[10]. https://circuitdigest.com/microcontroller-projects/arduino-can-tutorial-interfacing-mcp2515-can-bus-module-with-arduino

[11]. https://docs.arduino.cc/learn/communication/spi

[12]. https://circuitdigest.com/microcontroller-projects/arduino-spi-communication-tutorial