Automatic Wiper Control with Acoustic Rainfall Detection Sensor

Göktuğ Gökmen

Ankara University Department of Electrical and Electronics Engineering

1. Introduction

This report describes the designing process of a wiper control system with a manual control interface and a in-house developed rainfall detector. The design principles are affordability and integrability so cheapest possible solution are preferred in every step of the process and the whole project is developed by using CAN 2.0 protocol to achieve combability with commercial vehicles.

The aim of the project is to develop an affordable rainfall detection mechanism. The rainfall sensors used commercially are commonly use precise optoelectronic devices to determine water on a surface. Even though the rainfall detection is viable this way, these technologies are costly. This project approaches this problem in a different way. The detection of the rain achieved by the acoustic signals generated by the windshield when a droplet hits its surface.

2. System Overview

The system is divided into three ECU's mimicking the ECU structure found in vehicles. The communication between the ECU's is achieved by CAN 2.0 protocol with SAE J1939 addressing standards. The three ECU's are named as follows:

- Console ECU: The user interface.
- Sensor ECU: The part where the sensor data is evaluated.
- Wiper ECU: The control mechanism for the wiper motor.

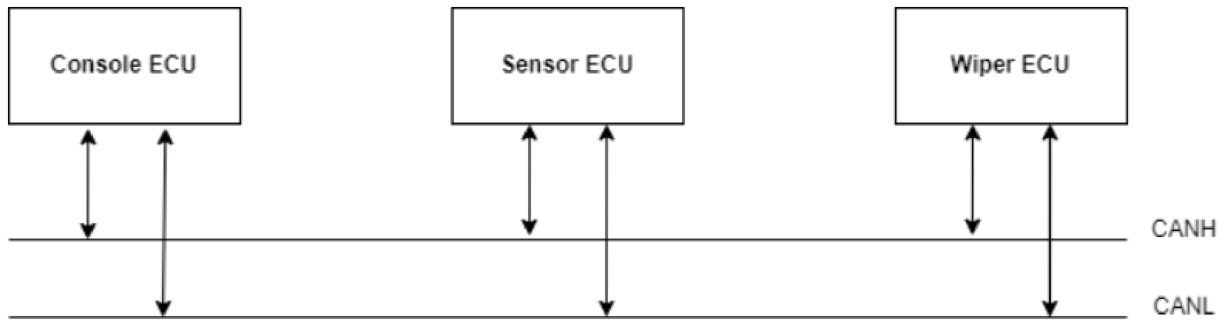


Figure 1 Flowchart of the Whole System

3. Console ECU

The console is designed to register the user inputs to the system. These inputs could be the wiper motor speed and the auto wiper activation.

3.1 Hardware

The user interference is designed to be a button array to select the desired output and a LED array to show which mode is selected. There are three buttons in the button array and three LEDs in the LED array. These inputs and outputs are marked as L, M, and H with respect to the desired wiper speeds of low, medium, and high. The I/O interface is connected to a microcontroller to determine which mode is selected. Then the microcontroller sends the desired frames to the other ECU's with the CAN controller and CAN transceiver modules.

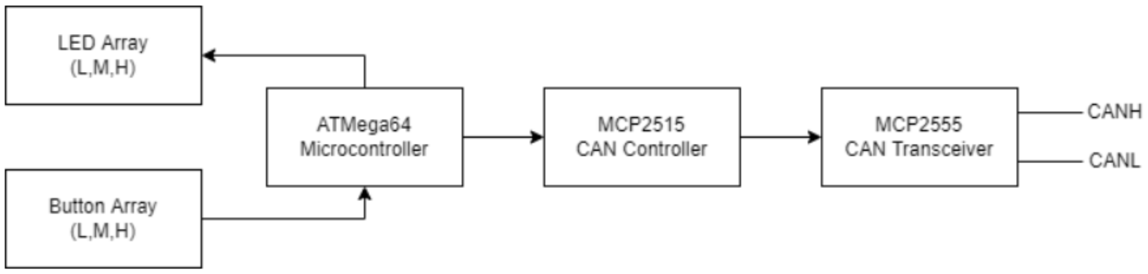


Figure 2 Flowchart of the Console ECU Hardware

3.2 Firmware

The three inputs are designed to be mutually exclusive, meaning that two outputs cannot occur simultaneously. Also, to set all outputs to zero, a mechanism where if a button is pushed twice all outputs default to zero is also introduced. The desired outputs are explained in the following flowchart.

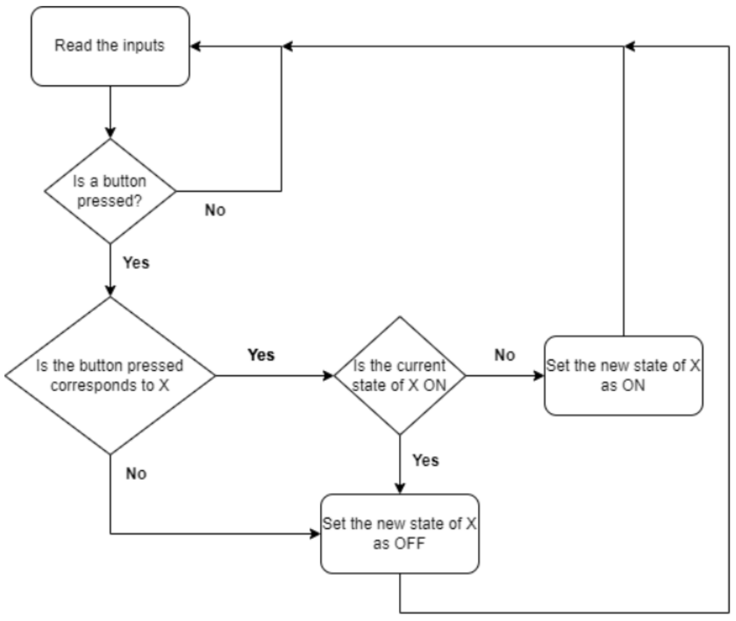

Read the inputs
Is a button pressed?
No
Yes
Is the button pressed corresponds to X
Yes
Is the current state of X ON
No
Set the new state of X as ON
No
Yes
Set the new state of X as OFF

Figure 3 Flowchart of the Button Logic

To integrate the desired feature into the software a combinational logic system is introduced rather than a sequential memory solution for increased processing speeds. The inputs and the outputs of the logic system is described as follows:

- **A:** Current state.
- **B:** Whether the intended button is pressed.
- **C:** Whether any of the three inputs is on. It can also be defined as $\mathbf{C = A_L + A_M + A_H}$
- **D:** The new state.

The logic table of the inputs and output is as shown:

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	X
1	0	1	0
1	1	0	X
1	1	1	0

Figure 4 Truth table of the button logic

The easiest solution to this logic system is found using a Karnaugh map and integrated into the embedded software.

AC **B**	00	01	11	10
0	0	0	0	X
1	1	1	0	X

Figure 5 K-map of the Button Logic

The easiest solution according to the K-map is following:

D = A'B

```
if ((state == !flag) && (state == HIGH)) {
  L_out = (!(L_out)) && L_state;
  M_out = (!(M_out)) && M_state;
  H_out = (!(H_out)) && H_state;
  flag = HIGH;
}
```

Figure 6 Implementation of the Button Logic

The selected outputs are sent to the other ECUs in CAN 2.0 protocol with the following data fields:

- 99 00 00 00 00 00 00 00: Set the wiper speed to low
- AA 00 00 00 00 00 00 00: Set the wiper speed to medium
- BB 00 00 00 00 00 00 00: Set the wiper speed to high

4.Sensor ECU

This part of the system receives acoustic signals with a microphone, processes the signals, and determines if a rainfall is present.

4.1 Hardware

The ECU structure is similar to the previous where there is an input (a microphone in this case), a output LED to show if rainfall is detected, a microprocessor determine the response and CAN modules to communicate the data to other ECU's

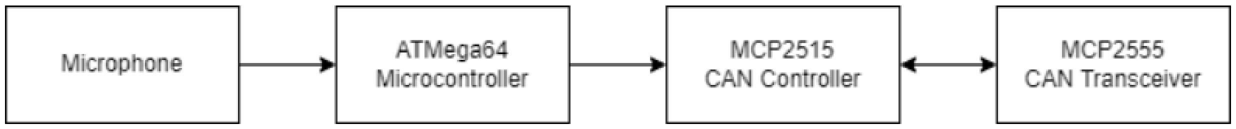


Figure 7 Flowchart of the Sensor ECU Hardware

4.2 Firmware

The received raw vibrations from the glass is not enough to determine whether there is rainfall. To make a conclusion, we must process the data in various steps.

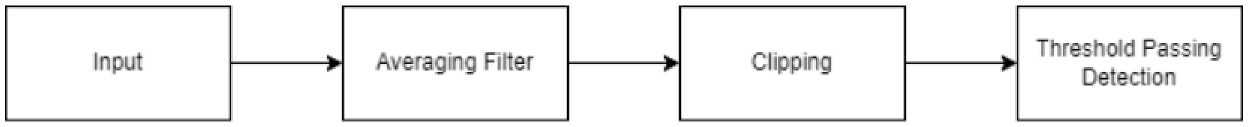


Figure 8 Flowchart of the Signal Process Steps

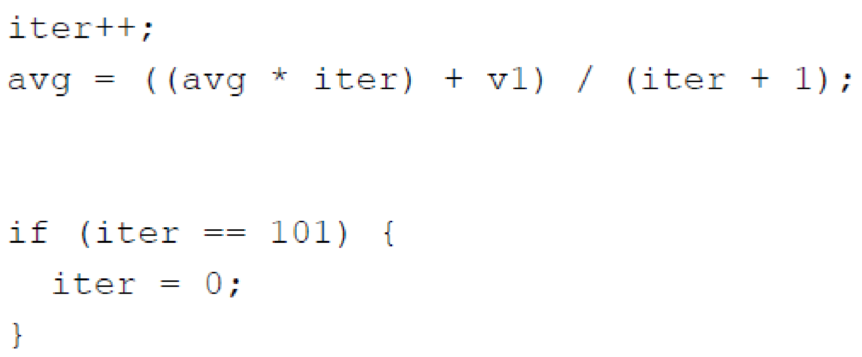
4.2.1 Averaging Filter

Due to the car being in different environments, the background sounds will differ, and the detection of the raindrops will become harder. To counteract this issue the moving average of the input is calculated, and the next input is compared against the moving average.

The moving average is defined as:

$$\hat{x}[n] = \frac{1}{N}\sum_{k=0}^{N-1} x[n-k]$$

N is chosen as N=100 for the implementation of the filter.

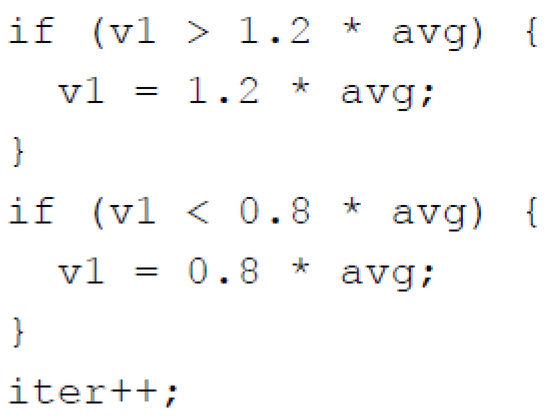


```
iter++;
avg = ((avg * iter) + v1) / (iter + 1);



if (iter == 101) {
  iter = 0;
}
```

Figure 9 Implementation of the Averaging Filter

4.2.2 Clipping

Clipping is needed in case of sudden spikes caused by loud noises or measuring errors. These types of errors affect the average value too much so any signal higher of lower than the average value by 20% gets clipped.



```
if (v1 > 1.2 * avg) {
  v1 = 1.2 * avg;
}
if (v1 < 0.8 * avg) {
  v1 = 0.8 * avg;
}
iter++;
```

Figure 10 Implementation of the Clipping

4.2.3 Threshold Passing Detection

A lower and a higher threshold are introduced as the 95% and 105% mark of the average. If an input surpasses the higher threshold or gets lower than the lower threshold, a timer starts where the number of times the threshold is surpassed is counted to determine if the noise increase is caused by raindrops.

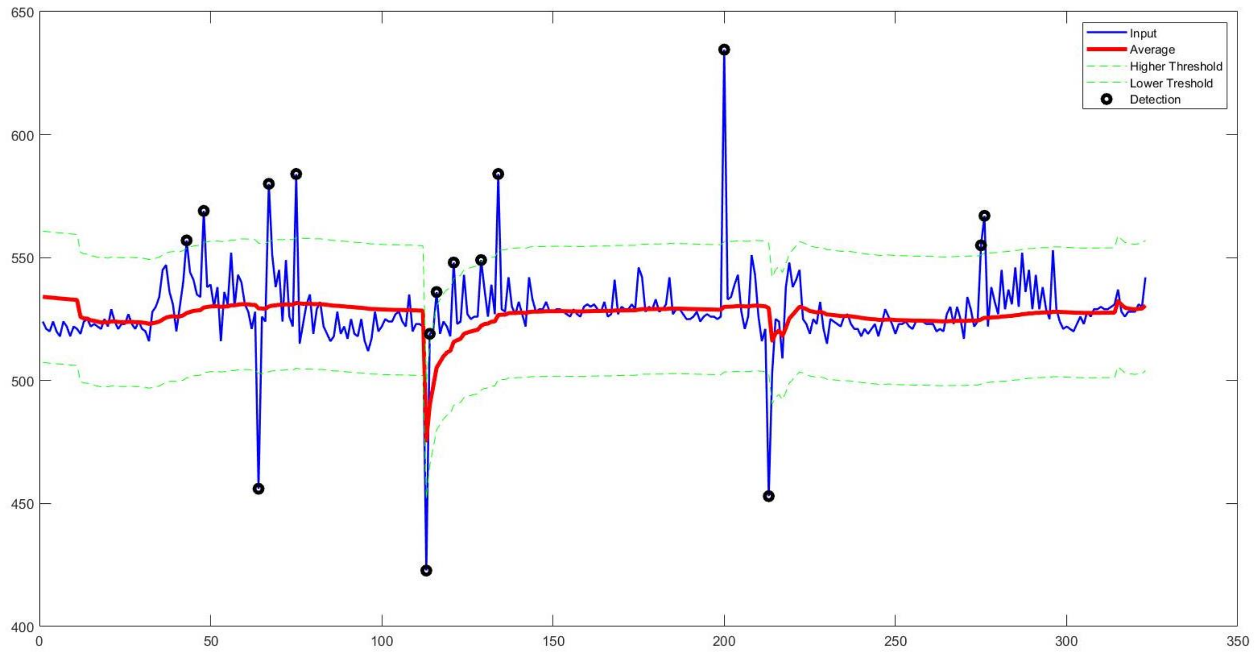


Figure 11 Threshold Passing Detection

The only can message sent from the Sensor ECU is as shown:

- AA 00 00 00 00 00 00 00: Set the wiper speed to medium

5.Wiper ECU

Wiper ECU is the part where the wiper motor is controlled according to the data received from the CAN bus.

5.1 Hardware

The hardware consists of an servo motor as a wiper, a microcontroller to set the motor speed and CAN modules to ensure communication with other ECU’s.

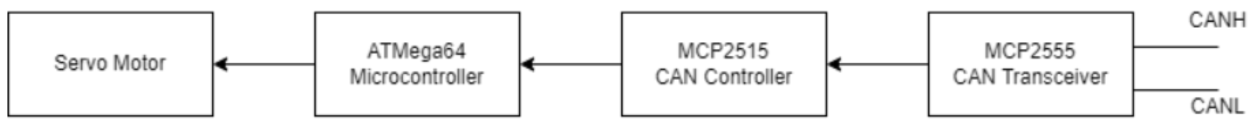
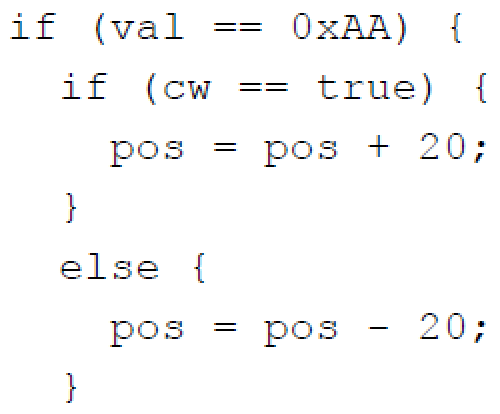


Figure 12 Flowchart of Wiper ECU Hardware

5.2 Firmware

The wiper sets the speed by deciding how much degrees should the servo change in a loop iteration. The range of motion is from 0 degrees to 180 degrees.



```
if (val == 0xAA) {
  if (cw == true) {
    pos = pos + 20;
  }
  else {
    pos = pos - 20;
  }
```

Figure 13 Implementation of the Wiper Speed Control

6.Appendix

6.1 Power

A 5V-2A power supply is used to power this project. Average current draw is measured as 400mA. To integrate this system to a vehicle a 12 to 5 volt voltage conversion is needed.

6.2 Cost

Name	Quantity	Unit Price	Total Price
Arduino Nano	3	159,28₺	468,84₺
MCP2515 CAN Controller Module	3	75,61₺	226,83₺
Microphone Module	1	15,93₺	15,93₺
SG90 Micro Servo Motor	1	33,61₺	33,61₺

*Peripherals are not added to the price calculation.

Total Cost:745,21₺

6.3 Console ECU Source Code

```
#include <SPI.h>
#include <mcp2515.h>
const int L_switch = 8;
const int M_switch = 3;
const int H_switch = 4;

const int L_lamp = 5;
const int M_lamp = 6;
const int H_lamp = 7;

int L_state = 0;
int M_state = 0;
int H_state = 0;

int L_out = 0;
int M_out = 0;
int H_out = 0;

int state = 0;
int flag = 0;

struct can_frame canMan;
struct can_frame canAuto;
MCP2515 mcp2515(10);

void setup() {
  Serial.begin(9600);
  pinMode(L_switch, INPUT);
  pinMode(M_switch, INPUT);
  pinMode(H_switch, INPUT);

  pinMode(L_lamp, OUTPUT);
  pinMode(M_lamp, OUTPUT);
  pinMode(H_lamp, OUTPUT);

  canMan.can_id  = 0x09A;
  canMan.can_dlc = 8;
  canMan.data[0] = 0x99;
  canMan.data[1] = 0x00;
  canMan.data[2] = 0x00;
  canMan.data[3] = 0x00;
  canMan.data[4] = 0x00;
  canMan.data[5] = 0x00;
  canMan.data[6] = 0x00;
  canMan.data[7] = 0x00;

  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();
}

void loop() {
  L_state = digitalRead(L_switch);
  M_state = digitalRead(M_switch);
  H_state = digitalRead(H_switch);

  state = (L_state || M_state || H_state);
  if (state == LOW) {
    flag = LOW;
  }

  if ((state == !flag) && (state == HIGH)) {
    L_out = (!(L_out)) && L_state;
    M_out = (!(M_out)) && M_state;
    H_out = (!(H_out)) && H_state;
    flag = HIGH;
  }

  digitalWrite(L_lamp, L_out);
  digitalWrite(M_lamp, M_out);
  digitalWrite(H_lamp, H_out);

  if (L_out == 1) {
    canMan.data[0] = 0x99;
    mcp2515.sendMessage(&canMan);
  }

  if (M_out == 1) {
    canMan.data[0] = 0xAA;
    mcp2515.sendMessage(&canMan);
  }

  if (H_out == 1) {
    canMan.data[0] = 0xBB;
    mcp2515.sendMessage(&canMan);
  }

  delay(100);
}
```

6.4 Sensor ECU Source Code

```
#include <SPI.h>
#include <mcp2515.h>

float avg = 0;
float v1 = 0;

unsigned long c1 = 0;
unsigned long c2 = 0;
unsigned long c3 = 0;
unsigned long c4 = 0;

int counter = 0;
int flag = 0;
bool sensor = false;
```

```
int iter = 0;
int sum = 0;

struct can_frame canAuto;
MCP2515 mcp2515(10);

void setup() {
  pinMode(A0, INPUT);
  pinMode(3, OUTPUT);

  Serial.begin(9600);

  canAuto.can_id  = 0x0AA;
  canAuto.can_dlc = 8;
  canAuto.data[0] = 0xAA;
  canAuto.data[1] = 0x00;
  canAuto.data[2] = 0x00;
  canAuto.data[3] = 0x00;
  canAuto.data[4] = 0x00;
  canAuto.data[5] = 0x00;
  canAuto.data[6] = 0x00;
  canAuto.data[7] = 0x00;

  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();
  v1 = analogRead(A0);
  avg = v1;
}

void loop() {
  v1 = analogRead(A0);

  if (v1 > 1.2 * avg) {
    v1 = 1.2 * avg;
  }
  if (v1 < 0.8 * avg) {
    v1 = 0.8 * avg;
  }
  iter++;
  avg = ((avg * iter) + v1) / (iter + 1);


  if (iter == 101) {
    iter = 0;
  }

  if ((v1 > 1.05 * avg) || (v1 < 0.95 * avg)) {
    if (sensor == false) {

      counter = (counter + 1) * flag;
      if (flag == 0) {
        c2 = millis();
        flag = 1;
        counter++ ;
      }
    }
  }
  if (flag == 1) {
    c1 = millis();
  }

  if (c1 - c2 > 3000) {
    flag = 0;
    c1 = 0;
    c2 = 0;
    if (counter > 5 && counter < 50) {
      sensor = true;
      c3 = millis();
    }
    counter = 0;
  }


  if (sensor == true) {
    c4 = millis();
    digitalWrite(3, HIGH);
    mcp2515.sendMessage(&canAuto);
    if (c4 - c3 > 10000) {
      digitalWrite(3, LOW);
      c3 = 0;
      c4 = 0;
      sensor = false;
    }
  }

  Serial.print(v1);
  Serial.print(" ");
  Serial.print(avg);
  Serial.print(" ");
  Serial.print(1.05*avg);
  Serial.print(" ");
  Serial.println(0.95*avg);

  delay(100);

}
```

6.5 Wiper ECU Source Code

```
#include <SPI.h>
#include <mcp2515.h>
#include <Servo.h>

struct can_frame canMsg;
MCP2515 mcp2515(10);
Servo sr;
int pos = 0;
bool cw = true;
int val = 0x00;

void setup() {
  sr.attach(5);
  Serial.begin(9600);
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();
}

void loop() {
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    val = canMsg.data[0];
  }
  else {
    val = 0x00;
  }
  if (val == 0x00 && pos > 0) {
    pos = pos - 20;
    sr.write(pos);
  }

  if (val == 0x99) {
    if (cw == true) {
      pos = pos + 10;
    }
    else {
      pos = pos - 10;
    }

    sr.write(pos);
  }
  if (val == 0xAA) {

    if (cw == true) {
      pos = pos + 20;
    }
    else {
      pos = pos - 20;
    }
    sr.write(pos);
  }

  if (val == 0xBB) {

    if (cw == true) {
      pos = pos + 30;
    }
    else {
      pos = pos - 30;
    }
    sr.write(pos);
  }

  if (pos >= 180) {
    cw = false;
  }
  if (pos <= 0) {
    cw = true;
  }
  delay(100);
```