ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E
MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	7
EXPERIMENT DATE	:	04.01.2021
LAB SESSION	:	MONDAY - 13.30
GROUP NO	:	G10

GROUP MEMBERS:

150180103	:	EGEMEN GÜLSERLİLER
150180028	:	ABDULKADİR PAZAR
150180716	:	AHMET SELÇUK TUNÇER

AUTUMN 2020

Contents

1 INTRODUCTION

In this experiment, we discover how communication protocols such as UART, I2C, SPI and CAN work. At first part, we programmed a SensorMC that takes measurement data and transmit it via UART protocol. At second part, we implemented a program that gets commands via UART protocol and make relevant position changes on the servo engines. At last part, we implemented a program that takes data from flex sensor and control the servo engines by using this data via I2C protocol.

2 MATERIALS AND METHODS

2.1 Part 1: UART Transmit Data

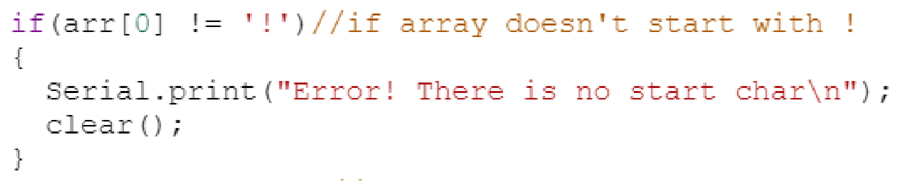
In this part, we used to sensor microcontroller to convert the value measured from flex sensors to packets in "!SensorIndex;AnalogValue;Resistance;Angle#" format. To accomplish this, we created a packet function that takes sensor index and value as arguments. Since value is between 236 and 634 for angles between 0 - 180, we used a linear function $angle = \frac{(value-236)}{2.21}$ to turn values to angles. We used a similar function $(resistor = (angle \times 0.44) + 45)$ to turn values to resistor values between $45k\Omega - 125k\Omega$. We multiplied obtained resistor value by 100 since in requested format resistor values had 2 digits after the decimal point. Finally we wrote the data in requested format to an array. The calculated values for angles were different from observed values since we assumed flex sensors work linearly.

Observed values	Calculated values
0	0
30	24
60	52
90	83
120	115
150	148
180	180

Table 1: Calculated values compared to observed values

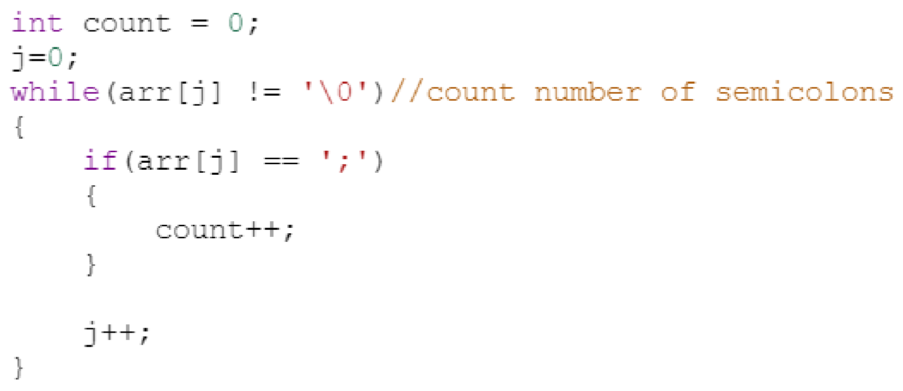
2.2 Part 2: UART Recieve Data

In this part we used the servo microcontroller to parse messages written to serial monitor and turn servos according to the angle given in message. The format of the message is ”!S0;S1;S2;S3#”. The following code was used to detect possible errors in the message:



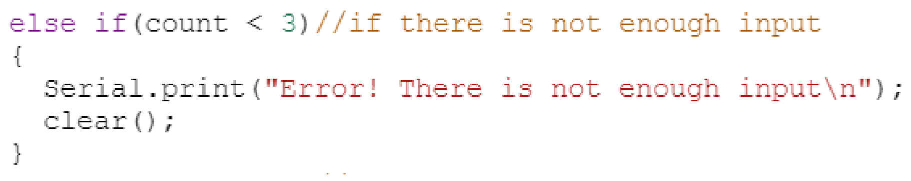
```
if(arr[0] != '!')//if array doesn't start with !
{
  Serial.print("Error! There is no start char\n");
  clear();
}
```

Figure 1: No start char error



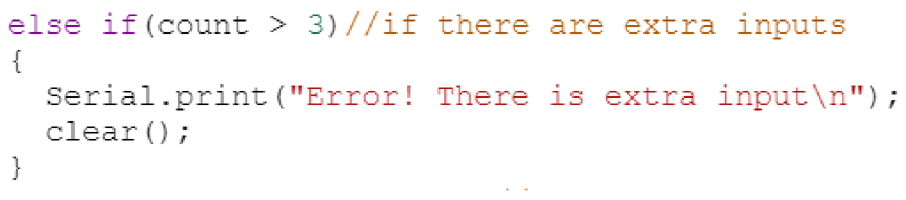
```
int count = 0;
j=0;
while(arr[j] != '\0')//count number of semicolons
{
    if(arr[j] == ';')
    {
        count++;
    }

    j++;
}
```

Figure 2: Count semicolons to determine input number



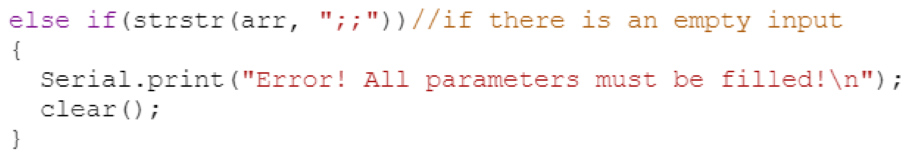
```
else if(count < 3)//if there is not enough input
{
  Serial.print("Error! There is not enough input\n");
  clear();
}
```

Figure 3: Not enough input error



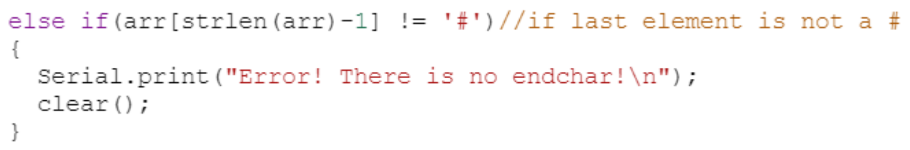
```
else if(count > 3)//if there are extra inputs
{
  Serial.print("Error! There is extra input\n");
  clear();
}
```

Figure 4: Extra input error



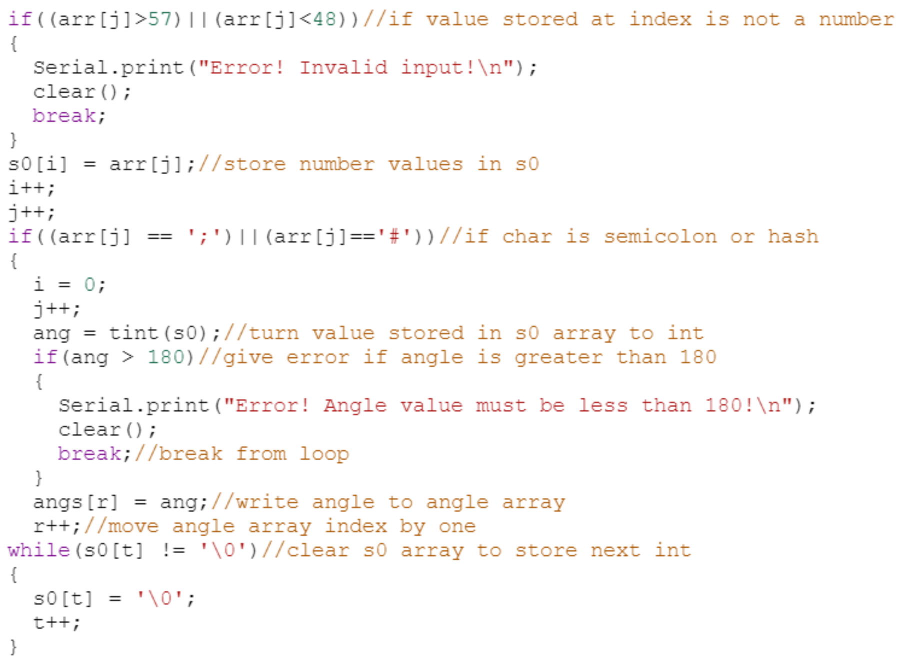
```
else if(strstr(arr, ";;"))//if there is an empty input
{
  Serial.print("Error! All parameters must be filled!\n");
  clear();
}
```

Figure 5: Empty input error



```
else if(arr[strlen(arr)-1] != '#')//if last element is not a #
{
  Serial.print("Error! There is no endchar!\n");
  clear();
}
```

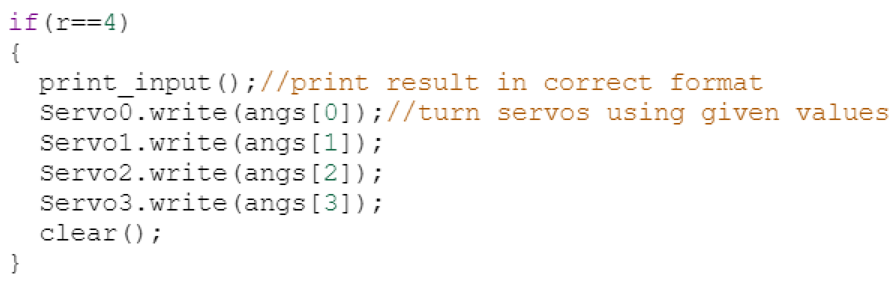
Figure 6: No end char error



```
if((arr[j]>57)||(arr[j]<48))//if value stored at index is not a number
{
  Serial.print("Error! Invalid input!\n");
  clear();
  break;
}
s0[i] = arr[j];//store number values in s0
i++;
j++;
if((arr[j] == ';')||(arr[j]=='#'))//if char is semicolon or hash
{
  i = 0;
  j++;
  ang = tint(s0);//turn value stored in s0 array to int
  if(ang > 180)//give error if angle is greater than 180
  {
    Serial.print("Error! Angle value must be less than 180!\n");
    clear();
    break;//break from loop
  }
  angs[r] = ang;//write angle to angle array
  r++;//move angle array index by one
while(s0[t] != '\0')//clear s0 array to store next int
{
  s0[t] = '\0';
  t++;
}
```

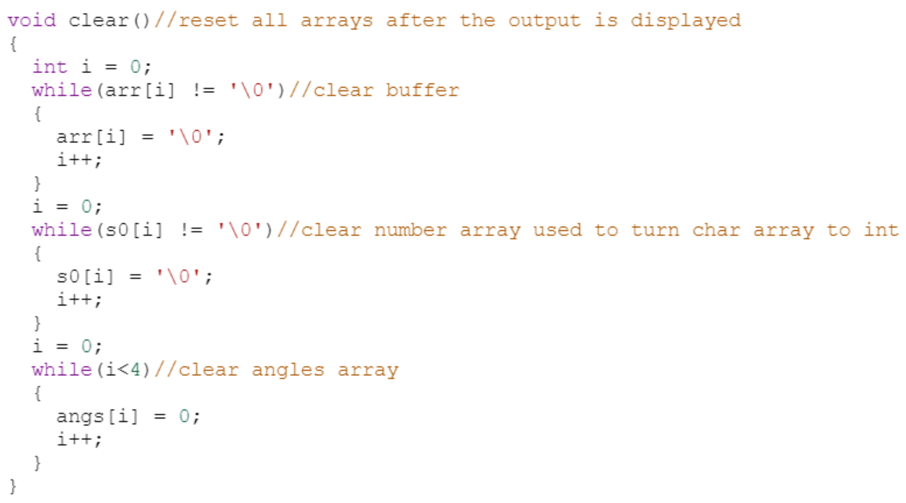
Figure 7: Remaining errors

In this code block, we first check if there are characters that are not numbers using the fact that digits have ASCII values between 48 to 57 to print invalid input error. We then store numbers in s0 array to later turn them to integer values. If ';' or '#' characters are encountered, we skip again and turn s0 array to integer. If result is greater than 180, we print angle greater than 180 error.



```
if(r==4)
{
  print_input();//print result in correct format
  Servo0.write(angs[0]);//turn servos using given values
  Servo1.write(angs[1]);
  Servo2.write(angs[2]);
  Servo3.write(angs[3]);
  clear();
}
```

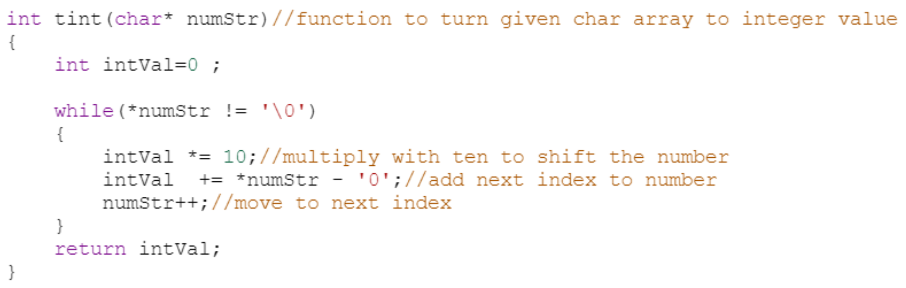
Figure 8: Input is in correct format



```
void clear()//reset all arrays after the output is displayed
{
  int i = 0;
  while(arr[i] != '\0')//clear buffer
  {
    arr[i] = '\0';
    i++;
  }
  i = 0;
  while(s0[i] != '\0')//clear number array used to turn char array to int
  {
    s0[i] = '\0';
    i++;
  }
  i = 0;
  while(i<4)//clear angles array
  {
    angs[i] = 0;
    i++;
  }
}
```

Figure 9: Clear arrays

We clear arrays after each message is parsed.



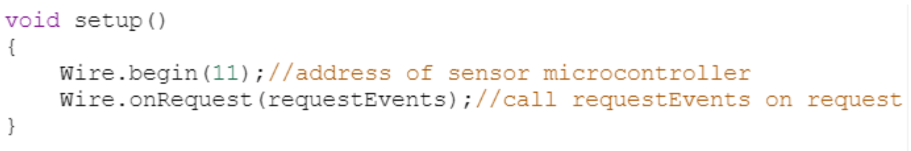
```
int tint(char* numStr)//function to turn given char array to integer value
{
    int intVal=0 ;

    while(*numStr != '\0')
    {
        intVal *= 10;//multiply with ten to shift the number
        intVal  += *numStr - '0';//add next index to number
        numStr++;//move to next index
    }
    return intVal;
}
```

Figure 10: Function to turn char arrays to integers

2.3 Part 3: I2C Communication

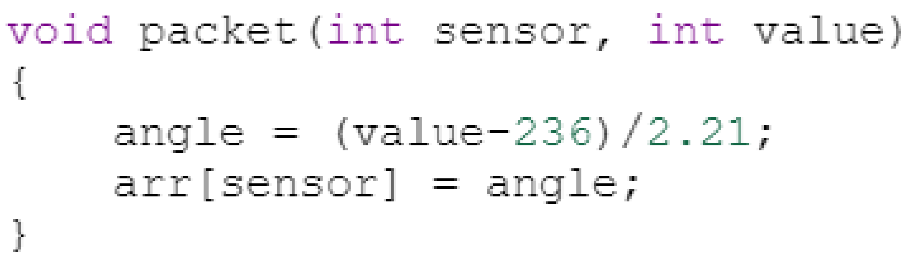
In this part we used all microcontrollers to implement a program that gets angle values from flex sensors and control the servo engines using these angle values. We used the Wire library to establish communication between 3 microcontrollers.

In the code of the Sensor microcontroller, we first appoint an address for this microcontroller we measured values using analogRead() and created packets of angles using packet() function. Using onRequest function of Wire library, we send the packets when the Master microcontroller makes a request using requestEvents function.



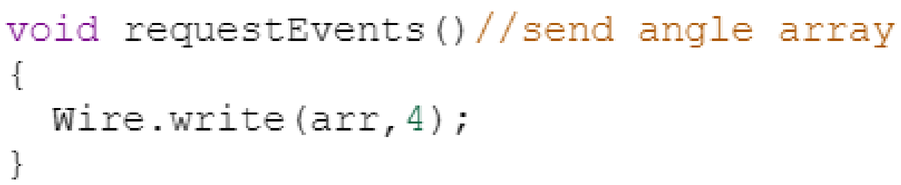
```
void setup()
{
    Wire.begin(11);//address of sensor microcontroller
    Wire.onRequest(requestEvents);//call requestEvents on request
}
```

Figure 11: Code in setup



```
void packet(int sensor, int value)
{
    angle = (value-236)/2.21;
    arr[sensor] = angle;
}
```

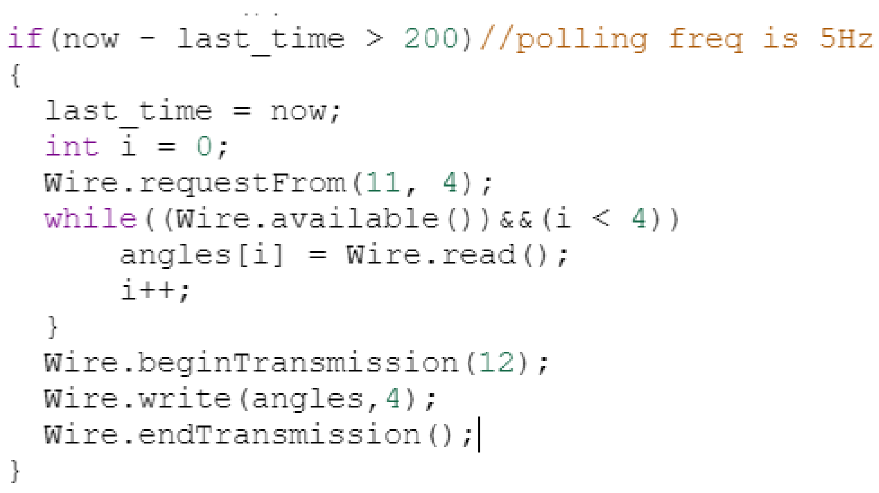
Figure 12: New packet() function



```
void requestEvents()//send angle array
{
  Wire.write(arr,4);
}
```

Figure 13: requestEvents function

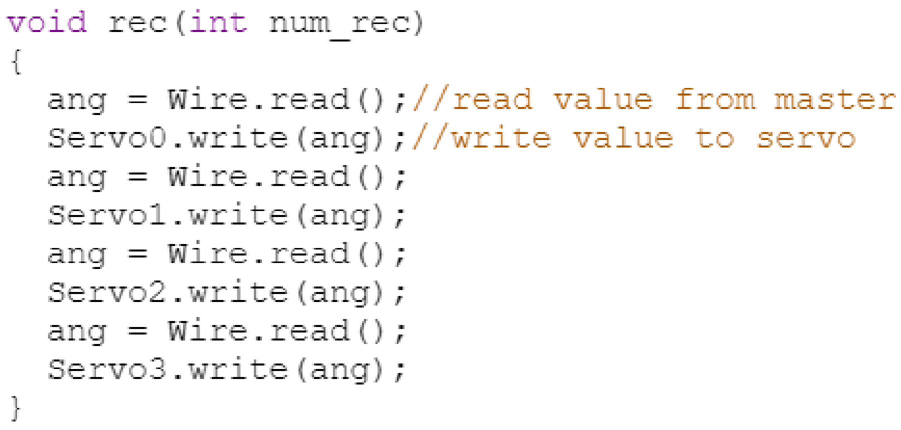
In the code of the Master microcontroller, since this controller is the master, we do not appoint an address. Then we request data from Sensor MC to send to Servo MC.



```
if(now - last_time > 200)//polling freq is 5Hz
{
  last_time = now;
  int i = 0;
  Wire.requestFrom(11, 4);
  while((Wire.available())&&(i < 4))
      angles[i] = Wire.read();
      i++;
  }
  Wire.beginTransmission(12);
  Wire.write(angles,4);
  Wire.endTransmission();
}
```

Figure 14: Master MC communication code

In the code of the Servo microcontroller, we first appoint an address for this microcontroller. We use onRecieve() function on Wire library to call rec function when we recieve information.



```
void rec(int num_rec)
{
  ang = Wire.read();//read value from master
  Servo0.write(ang);//write value to servo
  ang = Wire.read();
  Servo1.write(ang);
  ang = Wire.read();
  Servo2.write(ang);
  ang = Wire.read();
  Servo3.write(ang);
}
```

Figure 15: rec() function

3 RESULTS

3.1 Part 1: UART Transmit Data

In part 1, the packets in correct format are produced according to the angle values of the flex sensors.

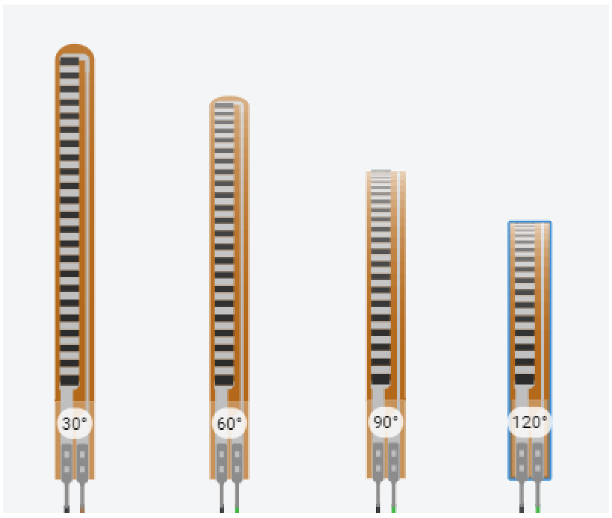
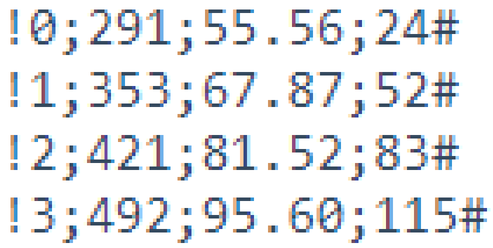


Figure 16: Angles of flex sensors



```
!0;291;55.56;24#
!1;353;67.87;52#
!2;421;81.52;83#
!3;492;95.60;115#
```

Figure 17: Produced packets

3.2 Part 2: UART Recieve Data

In part 2, the messages are parsed and if there are errors correct error messages are printed accordingly. If the message is in correct format, the servo engines are activated to turn by specified angle values.

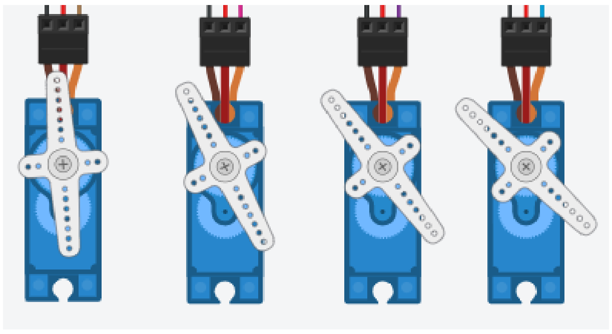
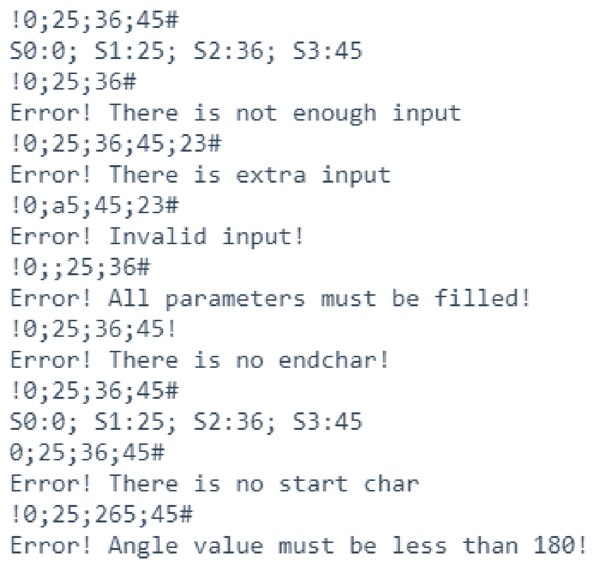


Figure 18: Positions of servo engines



```
!0;25;36;45#
S0:0; S1:25; S2:36; S3:45
!0;25;36#
Error! There is not enough input
!0;25;36;45;23#
Error! There is extra input
!0;a5;45;23#
Error! Invalid input!
!0;;25;36#
Error! All parameters must be filled!
!0;25;36;45!
Error! There is no endchar!
!0;25;36;45#
S0:0; S1:25; S2:36; S3:45
0;25;36;45#
Error! There is no start char
!0;25;265;45#
Error! Angle value must be less than 180!
```

Figure 19: State of the serial monitor after all example packets are entered.

3.3 Part 3: I2C Communication

In this part servo engines' positions change according to the values of flex sensors because of the communications between microcontrollers.

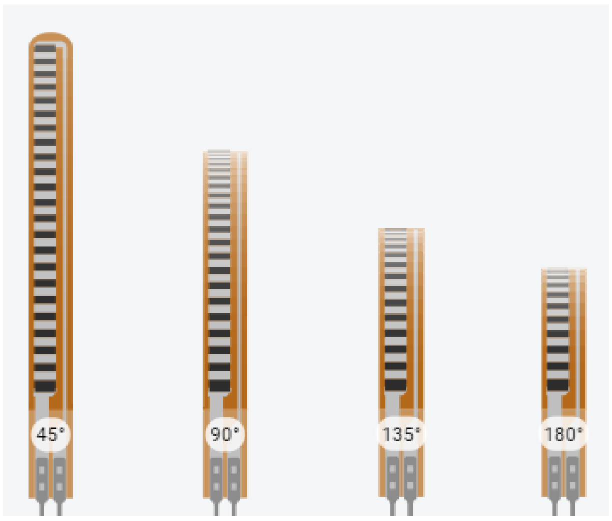


Figure 20: Values of flex sensors

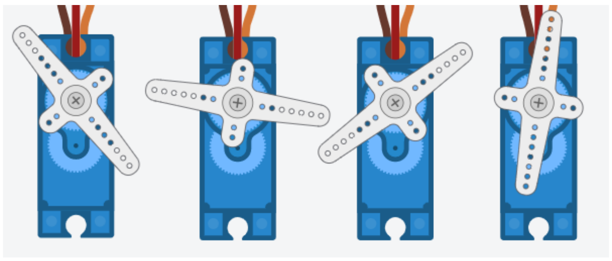


Figure 21: Positions of servo engines

4 DISCUSSION

There are 2 type of communication protocols that are used in embedded systems: Inter System Protocol and Intra System Protocol.

Inter System Protocols used for interaction between different systems. The communication is done through an inter bus system for example the communication between a computer and a peripheral device using a USB cable. UART, USART and USB are example of inter-system protocols.

Intra System Protocols allow the secured communication within the system, like the connection between the MCU and a sensor. The circuit complexity and power consumption will be increased by using intra-system protocol and it is very secure to accessing the data compared with inter system protocol. I2C, SPI and CAN are examples of intra-system protocols.

4.1 Difference Between Inter System Protocols

4.1.1 UART -Universal Asynchronous Transmitter and Receiver-

is a two wired protocol labeled as Rx and Tx. It is transferred and receives the data serially bit by bit without class pulses. It is a half-duplex communication and is slower compared with USART.

4.1.2 USART -Universal Synchronous and Asynchronous Transmitter and Receiver-

USART is also a two wired protocol labeled as Rx and Tx. It is used to transmitting and receiving the data byte by byte along with the clock pulses. It is a full-duplex protocol that means transmitting and receiving data simultaneously to different board rates. It is faster than UART and slower than USB.

4.1.3 USB -Universal Serial Bus-

USB is a serial communication of two-wire protocol labeled as D+ and D-. Sends and receives a data along with clock pulses. It is also full-duplex communication and faster than USART and UART.

4.2 Difference Between Intra System Protocols

4.2.1 I2C -Inter Integrated Circuit-

I2C requires two wires SDA and SCL to carry information between devices. It is a master to a slave communication protocol. It is a half-duplex and a multi master protocol. It transfer 8-bit data and receives 1 bit acknowledgment. It is within the circuit board. The biggest disadvantage of it is its limited speed.

4.2.2 SPI -Serial Peripheral Interface-

SPI requires 4 wires MOSI,MISO,SS and SCLK. It is a full duplex and a single master protocol. The master selects only one slave at a time. It is not limited to 8-bit words in the case of bit transferring. It is within the circuit board. It supports multiple slaves connectivity but it also leads to circuit complexity.

4.2.3 CAN -Controller Area Network-

CAN is a two wire CAN H+ and CAN H- protocol. It is a full duplex and a multi master protocol. It is within two circuit board. It shows robust performance.

5 CONCLUSION

In this experiment, we learned how IC2 and UART that are types of communication protocols on embedded systems work. We learned how to take analog inputs and then transmitting and receiving them via different communication protocols. We realize the principle of master and slave communication. Furthermore, thanks to using libraries Servo.h and Wire.h we improve our skills on Arduino environment. Moreover, searching on the internet for communication protocols make us more curious about the concept of embedded system communications. This experiment overall was an important experiment and a good practice and to understand the basic of communication protocols and their differences.