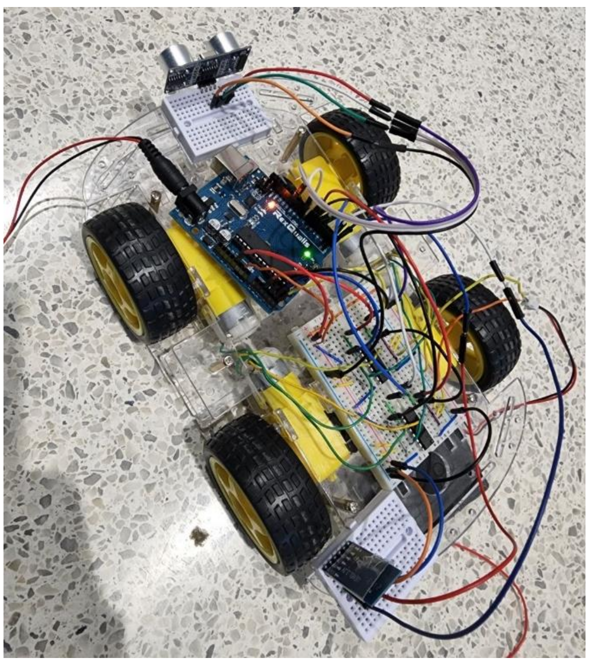
Name of Project: A Homemade Tesla from Africa
Joslin Some & Adnane Ezouhri, Group 7
April 29th, Embedded Systems ECE:3360
Professor R.Beichel

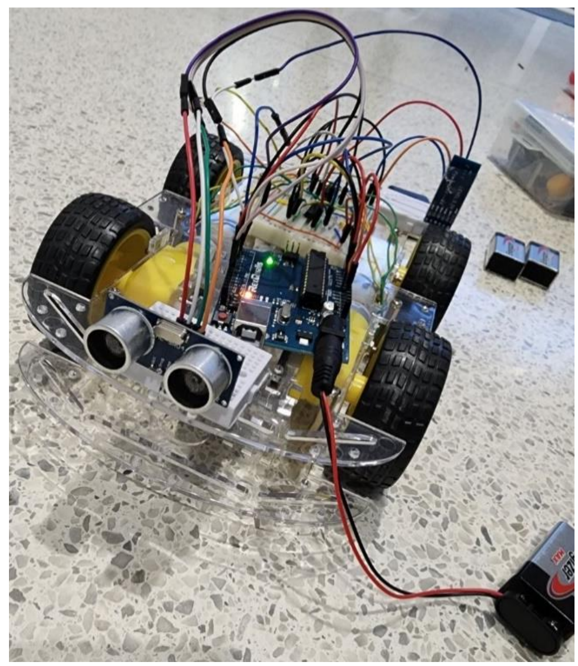
1. Introduction:

(1.a)

Ever since we enrolled in Embedded Systems and heard about the final Project, we have been pondering on what we should make. Our first couple thoughts were vary generic; everyone had the same idea. We wanted to think out of the box, bigger, smarter, and more embedded than ever. With the price of gas rising up and our world being more polluted than ever, in addition to our ambitious ideas:

We would like to present our very own Bluetooth controlled car, or what we like to call our "Tesla".





Now we know what you're thinking, "Group 7, this hardly sounds like a feasible project, and even if you did get it to work, what makes you think it could compete with the visionary cars of today?". Well, what if we told you, that our "Tesla" has everything a driver could ever ask for, and more.

(1.b)

Our goal was to successfully build a car and be able to control it using a Bluetooth module we would connect to use a pre-developed software. As much as we would have liked to work on our own piece of software for a speech-to-text application, with the constraints of resources and time we decided it was wiser to back on this idea.

In addition, another feature we thought would be interesting to add is object detection. As our “Tesla” will be driving down the halls of the Seamans Center, we wouldn’t want I to accidently bumped into someone (or something idk yet).

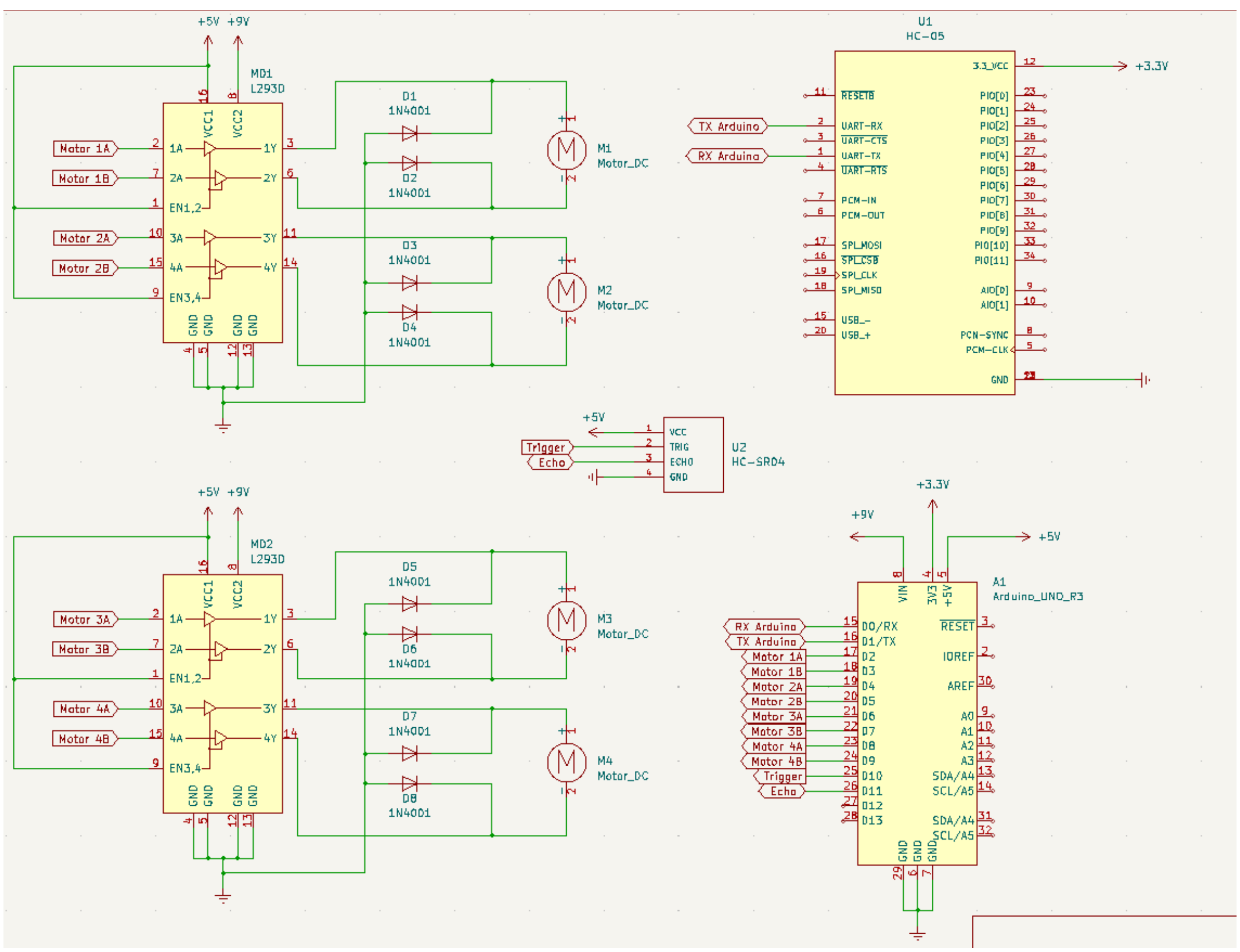
While we wanted to add extra features in our project, after multiple conversation with Pr. Beichel, we understood that with the amount of time given we’d only be able to implement our main features on top of the DC motors implementation and getting the required components.

2. Implementation:

(2.a) Overview: big picture of the project HW&SW components, outline interaction between components

Our project, being a car prototype will of course include 4 wheels, and in our case 4 motors. The motors will be connected in pairs to two L293D motor drivers. As the brain of this project, we will be using an Arduino UNO R3 powering and ATMega328p micro-controller. The multiple LEDs on our car used as front and back lights as well as indicators will be connected to a breadboard in a first instance before connecting some inputs to control them accordingly with our Arduino. In addition, our ultrasonic sensor will be connected directly to our Arduino to avoid any unnecessary extra delays and power consumption issue as the Arduino can get overwhelmed due to the LEDs and computation it has to do. Finally, the TAs for this course directed our attention to power consumption as the DC motors need a greater power to work efficiently. Hence, we have decided to opt for an external battery to accommodate for the needs of our DC motors.

(2.b) Schematic



(2.c) Software description

For the software, we created multiple functions to help us with our tasks.

First, we created a function to control the ultra-sonic sensor. This function allows us to initialize the sensor and measure the distance away from an object. Whenever an object is too close to the sensor, we set a command to make the car change direction.

```
double ultra_sonic(void)
{

    char string[10]={NULL};
    double count=0;
    double distance=0;


    PORTB |= (1 << PORTB3);                 /*Setting Trigger Pin high*/
    _delay_us(10);                          /*Wait 10 us to trigger the burst of an ultrasound
    PORTB &= (~(1 << PORTB3));              /*Setting Trigger Pin low*/


    TCNT1 = 0;                              /* Clear Timer counter */
    TCCR1B = 0x41;                          /* Capture on rising edge, No pre-scaler*/
    TIFR1 = 1<<ICF1;                        /* Clear ICP flag (Input Capture flag) */
    TIFR1 = 1<<TOV1;                        /* Clear Timer Overflow flag */
```

Next, we used the used the same methods as lab 5 to transfer serial data to the HC-05 Bluetooth module. However, we edited the USART_RECIEVE function such that the ultra-sonic sensor would always check for obstacles to see if it needs to make the car turn.

```
unsigned char USART_Receive( void )
{
    /* Wait for data to be received */
    while ( !(UCSR0A & (1<<RXC0)) )
    {
        double distance = ultra_sonic();

        //USART_Transmit('t');
        if(distance < 10.00 && distance > 0.93)
        {
            turnRight();
            distance=ultra_sonic();
            //_delay_ms(200);
            if(distance>=10.00)
            {
                stop();

            }
        }

    }
    /* Get and return received data from buffer */
    return UDR0;
}
```

We then got to do the fun part of making the wheels spin. That part was as easy as setting one of the motor wires to high and the other to low so the wheel can go forward, and vice versa for backwards. Doing this gave us these functions.

```
void frontLeftForward()
{
    PORTC &= ~ (1<<PORTC3);
    PORTC |=   (1<<PORTC2);
}
void frontLeftBackwards()
{
    PORTC |=   (1<<PORTC3);
    PORTC &= ~ (1<<PORTC2);
}
void backtLeftBackwards()
{
    PORTD |=   (1<<PORTD4);
    PORTD &= ~ (1<<PORTD5);
}
void backLeftForward()
{
    PORTD |=   (1<<PORTD5);
    PORTD &= ~ (1<<PORTD4);
}
```

After coding to control each individual wheel, it became easy to make general movement functions to allow the car to move in all directions. It was interesting to see that we could make the car turn by mixing up which wheels we turn forward and backwards. We also created a stop function that just sets all ports back to low in order to stop all movement. It's important that we call our stop function before any new movement since some wheels could just continue going otherwise.

```
void moveForward()
{
    stop();
    //_delay_ms(200);
    frontLeftForward();
    frontRightForward();
    backLeftForward();
    backRightForward();
}

void moveBackwards()
{
    stop();
    //_delay_ms(8000);
    frontLeftBackwards();
    frontRightBackwards();
    backRightBackwards();
    backtLeftBackwards();

}
void turnRight()
{
    stop();
    //_delay_ms(200);
    frontLeftForward();
    backLeftForward();
    backRightBackwards();
}
void turnLeft()
{
    stop();
    //_delay_ms(200);
    frontRightForward();
    backRightForward();
    backtLeftBackwards();
}
```

Finally, we set up our main function to receive commands via the serial monitor or in our case the HC-05 Bluetooth module. We downloaded an android app that allowed us to make voice commands that would send out a value to the Arduino. We made it so that saying “forward” would return ‘f’, ‘backwards’ ‘b’, ‘left’ ‘l’, ‘right’ ‘r’ and ‘stop’ ‘s’. Doing this allowed us to get our desired actions based on our given voice commands as shown below.

```
while (1) {

    command= USART_Receive();


        if(command=='f')
        {
            moveForward();
        }

        if(command=='b' )
        {
            moveBackwards();
        }

        if(command=='l' )
        {
            turnLeft();
        }

        if(command=='r' )
        {
            turnRight();
        }

        if(command=='s' )
        {
            stop();
        }


}
```

3. Experimental methods:

As we started our project, we decided to break down the different features, our group's mindset was to get each part to work independently from each other to then merge all of our work together and tune it to build dependencies between our features.

The first one we started with was the Bluetooth module:

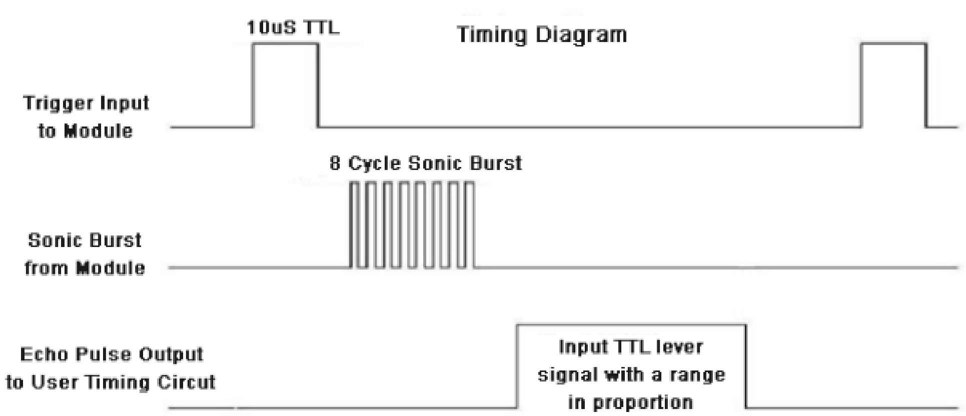
We initialized the module, similarly to what we did in lab5 by setting the correct baud rate, stop bit, and enabling the receiver and transmitter. When this was finished, it was easy enough to communicate with our device through the serial monitor.

```
void USART_Init( unsigned int ubrr)
{
    /*Set baud rate */
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char) ubrr;

    /* Enable receiver and transmitter */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);

    //Only 1 stop bit but 8 data No Parity
    UCSR0C = (0<<USBS0)|(3<<UCSZ00);
}
```

After correctly initializing the sensor, we tested whether we could send commands through it or not. We wired up a simple circuit with a red and green LED and created the commands "Red" and "Green" each turning on their respective colored LED and turning off the other. We needed to find a way to send the data via Bluetooth. For this we needed some sort of Bluetooth Arduino app which for some reason was not available on IOS devices. Luckily, we had an android phone and were able to find an app that sends Bluetooth text command wirelessly. After a few tests with that, we were able to control the light easily. From there, we just needed to find an app that sent values via Bluetooth using voice commands. After a few minutes of searching, we found the perfect app for us! We made it so the vocal "Green" command would send out a 'g' to the Arduino and so the "Red" command would send an 'r'.

We then focused on the ultrasonic sensor:

The key to understanding how the HC-SR04 works, was to look into the datasheet[1] available online from multiple resources. We learned that it works similarly to the DHT11 we used in a previous Laboratory assignment. In the sense that it requires to send and receive data during specific intervals. First, we need to enable the trigger pin as an output, and set high for 10µs, after getting this information, the module will send eight 40KHz signals automatically, at this point we start a timer and then detect whether or not the pulse has been detected using the echo pin as a way to communicate back to the Arduino.



When we detect a change in the echo pin, it triggers an interrupt, stopping the timer. The distance between the module and an object is then computed using the following formula:

distance = (travel time/2) x speed of sound

This formula varies depending on the frequency the Arduino is set to operate at, in our case 16MHz. In addition, we set up interrupts to work with this sensor. As depicted on our schematic, the echo pin of the HC-SR04 is connect to PB0 on our Arduino. This is a key connection between the Arduino and the sensor, as this specific pin is an Input Capture interrupt. The Timer/Counter we are using incorporates an Input Capture unit that can capture external events and give them a timestamp indicating time of occurrence. The external signal indicating an event, in our case the distance between an object and our car, is applied via the ICP1 pin. The timestamp is used to compute the distance relative to time and the speed of sound.

Experiment1: To make sure our project would work and to allow us some time to redirect in case of failure, we wanted to see if the Bluetooth device worked as expected. We connected the device on a single board with 2 LEDS and sent an on and off command through a pre-developed android app using voice recognition.

Experiment 2: Our first test was to check if the sensor could detect if an object is above or under a set distance from the sensor. To do so, we wired 2 LEDs a red and a green one, when an object was 10cm or more away from the module, the green LED would turn on. While on the other hand, when an object is detected at a distance under the set 10cm, the green light would turn off and the red one would then light up.

Experiment3: We then worked on the core part of our project, building the car, and wiring the motors to the motor drivers. In our laboratory kits, we each had an L293D motor driver. Each one of them could power 2 wheels. First, we had to get our hands on a datasheet to understand how the pin input and output works. The motor drivers being straightforward, we were able to wire it up very easily. To test our hardware, we manually set pins high and low simulating forward, backward, left, and right actions.

Experiment 4: Finally, we put all of our parts together to form an efficient “Tesla” model.

The main task of this experiment was to wire properly the different components and enable the appropriate pins as input or output. In addition, we also tested our poser source which was a 9v battery connected to the DC input of the Arduino.

4. Results:

Experiment 1: The Bluetooth device ended up working successfully with our application and we were able to turn lights on and off with voice commands.

Experiment 2: This experiment was a success as we were able to do exactly what we were reaching for. We set a threshold distance of 10 cm and slowly moved and object towards the sensor. The green LED was on until the threshold distance, once it was reached and any other distance below it, the red LED was on.

Experiment 3: This experiment was very successful, after getting all the parts, putting the different elements was like putting Legos together.

Experiment 4: We were able to correctly wire everything on our car. At first it was a mess, and the pin assignments were inconsistent. Re-wiring was necessary for us once we were done with this experiment

5. Discussion of Results:

Understanding the Bluetooth functionality with the lights allowed us to easily make the same process work with the car wheels. We set up similar commands with what we learned in our experiment in order to accurately control our car's movement.

Using what we learned from experiment 2, we were able to accurately work on getting our vehicle to detect objects in front of it. The same way we got the Arduino to turn on LEDs when an object was close, we also got it to make the car turn upon object detection.

At first, during our experiment we had our Arduino and the motors all connected to a 9V battery, this was very inefficient and made the batteries drain very fast. However, knowing this allowed us to fix things in the final version where we instead hooked up the Arduino alone to the 9V battery and connected the motors to an external battery pack. This let the car run for much longer than before.

6. Conclusion:

In conclusion, this was a very fun and fulfilling project that allowed us to use everything we've learned so far in embedded systems. Every step of the process was amazing. From deciding what project to make, to researching how to do it, to getting stuck on problems for hours, this was one of the few times in our college careers that we've felt like "true engineers" and we would gladly do it again. If we had more time, it would have been interesting to see what we could add to the car. Maybe LEDs for forward and back lights, a sound system while the car is in reverse or even find a way to implement other team's interesting projects into our own. One thing is for sure, this class and project have ensured that wherever we go whenever we see an embedded system we think "I have an idea how that might work" and in our eyes, that's the coolest thing possible!

Appendix A: Code:

```
/*

* FinalProject.c

*

* Created: 4/24/2022 8:03:06 PM

* Author : Adnane Ezouhri

*/




  #include <avr/io.h>

  #include <avr/interrupt.h>

  #include <util/delay.h>

  #include <string.h>

  #include <math.h>

  #include <stdio.h>

  #include <stdlib.h>

  #include <time.h>



  #define F_CPU 16000000UL

  #define FOSC 16000000 // Clock
Speed

  #define BAUD 9600

  #define MYUBRR FOSC/16/BAUD-1



  void USART_Init( unsigned int ubrr)
```

```
  void USART_Transmit( unsigned char
data )

{

   /* Wait for empty transmit buffer
*/

   while ( !( UCSR0A & (1<<UDRE0)));

   /* Put data into buffer, sends the
data */

   UDR0 = data;

}

  void WriteTerm(char data[] )

{

   int i=0;

   while(data[i]!=NULL)

   {

      USART_Transmit(data[i]);

      i=i+1;

   }

}



  void moveForward()
```

```
{

   /*Set baud rate */

   UBRR0H = (unsigned char)(ubrr>>8);

   UBRR0L = (unsigned char) ubrr;



   /* Enable receiver and transmitter
*/

   UCSR0B = (1<<RXEN0)|(1<<TXEN0);



   //Only 1 stop bit but 8 data No
Parity

   UCSR0C = (0<<USBS0)|(3<<UCSZ00);

}

  int TimerOverflow=0;



ISR(TIMER1_OVF_vect)

{

   TimerOverflow++;
/* Increment Timer Overflow count */

  }

  unsigned char garbage= 'g';



  double ultra_sonic(void)

{



   char string[10]={NULL};

   double count=0;

   double distance=0;
```

```
{

   stop();

   //_delay_ms(200);

   frontLeftForward();

   frontRightForward();

   backLeftForward();

   backRightForward();

}



  void moveBackwards()

{

   stop();

   //_delay_ms(8000);

   frontLeftBackwards();

   frontRightBackwards();

   backRightBackwards();

   backtLeftBackwards();



}



  void frontLeftForward()

{

   PORTC &= ~ (1<<PORTC3);

   PORTC |=   (1<<PORTC2);

}

  void frontLeftBackwards()

{

   PORTC |=   (1<<PORTC3);

   PORTC &= ~ (1<<PORTC2);
```

```
   PORTB |= (1 << PORTB3);
/*Setting Trigger Pin high*/

   _delay_us(10);
/*Wait 10 us to trigger the burst of
an ultrasound*/

   PORTB &= (~(1 << PORTB3));
/*Setting Trigger Pin low*/




   TCNT1 = 0;                       /*
Clear Timer counter */

   TCCR1B = 0x41;
/* Capture on rising edge, No pre-
scaler*/

   TIFR1 = 1<<ICF1;
/* Clear ICP flag (Input Capture
flag) */

   TIFR1 = 1<<TOV1;
/* Clear Timer Overflow flag */



   /*Calculate width of Echo by Input
Capture (ICP) */



   while ((TIFR1 & (1 << ICF1)) ==
0);        /* Wait for rising edge */

   TCNT1 = 0;                       /*
Clear Timer counter */

   TCCR1B = 0x01;
/* Capture on falling edge, No pre-
scaler */

   TIFR1 = 1<<ICF1;
/* Clear ICP flag (Input Capture
flag) */

   TIFR1 = 1<<TOV1;
/* Clear Timer Overflow flag */



   TimerOverflow = 0;
/* Clear Timer overflow count */

   while ((TIFR1 & (1 << ICF1)) ==
```

```
}

  void backtLeftBackwards()

{

   PORTD |=   (1<<PORTD4);

   PORTD &= ~ (1<<PORTD5);

}

  void backLeftForward()

{

   PORTD |=   (1<<PORTD5);

   PORTD &= ~ (1<<PORTD4);

}

  void frontRightForward()

{

   PORTC |=   (1<<PORTC4);

   PORTC &= ~ (1<<PORTC5);

}

  void frontRightBackwards()

{

   PORTC |=   (1<<PORTC5);

   PORTC &= ~ (1<<PORTC4);

}


  void backRightBackwards()

{

   PORTD &= ~ (1<<PORTD7);

   PORTD |=   (1<<PORTD6);

}
```

```c
0);        /* Wait for falling edge */

   count = ICR1 + (255 *
TimerOverflow);  /* Take count */

   /* 16MHz Timer freq, sound speed
=343 m/s */

   distance = ((double)count * 343) /
320000;

   dtostrf(count,2,2,string);

   strcat(string," cm");

   if(count>30000)

   {

      distance=50;

   }

   return distance;

}

  unsigned char USART_Receive( void )

{

   /* Wait for data to be received */

   while ( !(UCSR0A & (1<<RXC0)) )

   {

      double distance =
ultra_sonic();

      //USART_Transmit('t');

      if(distance < 10.00 && distance
> 0.93)

      {
```

```c
  void backRightForward()

{

   PORTD &= ~ (1<<PORTD6);

   PORTD |=   (1<<PORTD7);

}

  void turnRight()

{

   stop();

   //_delay_ms(200);

   frontLeftForward();

   backLeftForward();

   backRightBackwards();

}

  void turnLeft()

{

   stop();

   //_delay_ms(200);

   frontRightForward();

   backRightForward();

   backtLeftBackwards();

}

  void stop()

{

   PORTC &= ~  0b111100;

   PORTD &= ~  0b11110000;

}
```

```
        turnRight();

        distance=ultra_sonic();

        //_delay_ms(200);

        if(distance>=10.00)

        {

           stop();



        }

     }



  }

  /* Get and return received data
from buffer */

  return UDR0;

}




 char* ReceivedTerm(void)

{

  char data[20];

  int i=0;

  while(i<5)

  {

     data[i]=USART_Receive();

     i+=1;

  }

  return data;

}
```

```
  //Pb1 input 1

//PD6 - input 2

//Pd5 - input +

//Pb2,Pb4

//Pb0

// yellow high

// green low

  int main()

{

   USART_Init(MYUBRR);

   char text[4]={'0'};

   DDRB |= 0B01000;

   DDRD |= 0B111100000;

   DDRC |= 0B111100;



   //front right Wheel Green- PC4 ,
Yellow- PC5

   //front left Wheel  Green- PC2 ,
Yellow- PC3



   //back right Wheel, Green- PD6 ,
Yellow- PD5

   //back left wheel, Green-  PD4 ,
Yellow- PD5



   PORTC &= ~  0b00000000;

   PORTD &= ~  0b00000000;



   sei();                          /*
Enable global interrupt */

   TIMSK1 = (1 << TOIE1);
```

	`` /* Enable Timer1 overflow interrupts */ TCCR1A = 0; /* Set all bit to zero Normal operation */ char command=NULL; char temp=NULL; while (1) { command= USART_Receive(); if(command=='f') { moveForward(); } if(command=='b') { moveBackwards(); } if(command=='l') { turnLeft(); } if(command=='r') ```

```
          {
             turnRight();
          }


          if(command=='s' )
          {
             stop();
          }



    }


}
```

Appendix B: References:

-HC-05 Bluetooth module datasheet:

http://www.electronica60norte.com/mwfls/pdf/newBluetooth.pdf

-L293D IO pin mappings: https://components101.com/ics/l293d-pinout-features-datasheet

- Beichel, Reinhard. *Lab5_ES_S22_f*, April 7. University of Iowa.

- Beichel, Reinhard. *ES22_Lab05Notes*, March 22. University of Iowa.

- Beichel, Reinhard. *ES22_CProgramming*, March 29. University of Iowa.

- Beichel, Reinhard. *ATmega328P_Datasheet*, January 18. University of Iowa.

- Beichel, Reinhard. *ARDUINO_V2*, January 18. University of Iowa.

- L293D IO pin mapping: https://components101.com/ics/l293d-pinout-features-datasheet

- HC-05 Bluetooth module datasheet: http://www.electronica60norte.com/mwfls/pdf/newBluetooth.pdf