

EE 335

Advanced Microcontrollers Lab Projects

Austin Hankins | Michael Hernandez

Table of Contents

1. LAB #1: BLUETOOTH WIRELESS COMMUNICATION	
i. Part 1: Connecting the Bluetooth Modem.....	3
ii. Part 2: Testing Bluetooth Connection.....	4
iii. Part 3: Modify Sketch.....	6
iv. Conclusion.....	12
2. ENCODERS AND MOTORS	
i. Part1: Wheel Encoders.....	13
ii. Part2: Motors and the Motor Shield.....	18
iii. Conclusion.....	20
3. SPEED FEEDBACK	
i. Part1: Gripper & Servo	23
ii. Part2: Line Sensor	26
iii. Part3: Ultrasonic Range Finder	29
iv. Conclusion	33
4. MOTOR SPEED CONTROL AND LINE FOLLOWING	
i. Part1: PID Motor Speed Control Class.....	34
ii. Part2: Line Follower Control Class.....	36
iii. Conclusion.....	38
5. ROBOT OBSTACLE COURSE (Final Lab Write Up)	
i. Part:1 Testing PID Speed Control on the Obstacle Course.....	43
ii. Testing the Line Following Function on the Obstacle Course.....	45
iii. Testing the Object Collection Function on the Obstacle Course.....	50
iv. Part4: Testing Maze Navigation on the Obstacle Course.....	52
v. UART / TIMER / INTERRUPT Resources.....	53
vi. Bill of Material List.....	54

Lab #1: Bluetooth Wireless Communication

Objective:

Interface with an Atmega2560 Microcontroller to Bluetooth, using an HC-05 Bluetooth Modem. A default sketch is provided by instructor as a starting point for setup, this will verify Bluetooth connectivity and the ability to decode ASCII characters transmitted by a cell-phone application.

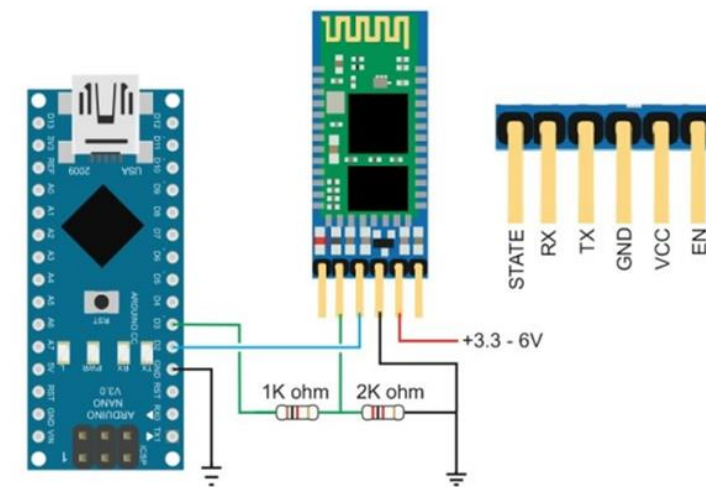
Required Materials:

Breadboard | AtMega2560 Microcontroller and Data Sheet | HC-05 Bluetooth Modem | Jumper Wires | Oscilloscope | Arduino ISE Software | Cell-phone or Tablet

Part 1: Connecting the Bluetooth Modem

Layout of connection between Arduino Mega and HC-05 Bluetooth Modem, the connections made properly will allow communication through serial pins RX and TX.

Figure 1: Diagram of Wiring Connections



Part #2: Connecting Via Bluetooth

Provided Sketch “Bluetooth_Testing_HC_05.ino” is downloaded from Blackboard and uploaded to Arduino-Mega, and an Application from NEXT PROTOTYPES “BlueTooth Serial Controller” is installed on a Cell-Phone. Only an Android type device will work for this application, our Apple cell-phones with OS will not work due to the Bluetooth hardware not being compatible with HC-05. Because of this we decided to use a Motorola Moto e5 PLAY phone running on stock Android.

Bluetooth_Testing_HC_05.ino

```
#include <f>

const byte rxPin = 0;
const byte txPin = 1;
SoftwareSerial BTSerial(rxPin, txPin);

void setup() {
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  Serial.begin(38400);
  // Usually 9600 for BT mode, although it is sometimes 38400:
  BTSerial.begin(9600);
  //38400 for command mode:
  //BTSerial.begin(38400);
}

void loop() {
  if(BTSerial.available())
  {
    Serial.print((char)BTSerial.read());
  }
  if(Serial.available())
  {
    BTSerial.print((char)Serial.read());
  }
}
```

In the Android app BlueTooth Serial Controller, the buttons 1 through 9 were programmed to send the corresponding signals in ASCII (49 – 57). When the signal was sent on the serial monitor the following was observed.

Figure 2: Serial monitor output when buttons are pressed

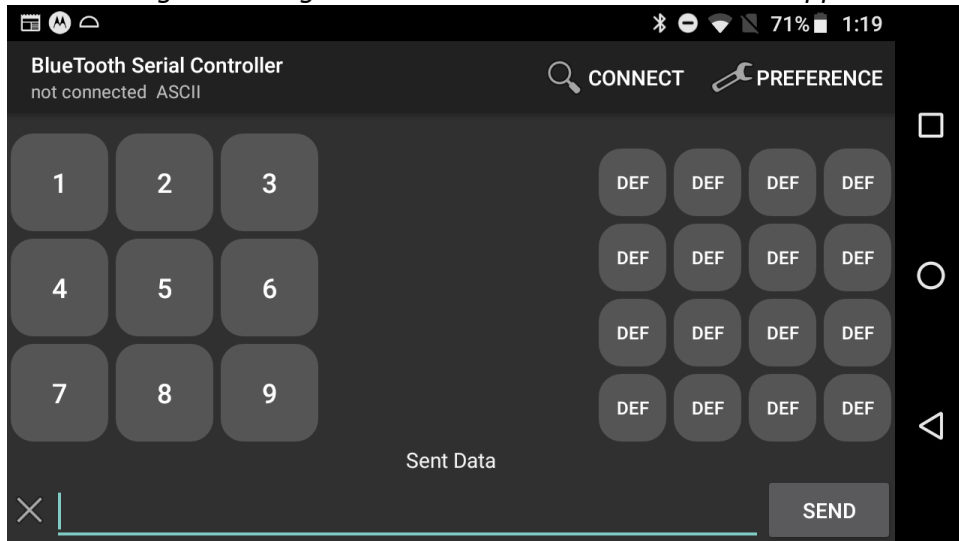
```

COM10 (Arduino/Genuino Mega or Mega 2560)

19:12:33.315 -> 2
19:12:35.060 -> 3
19:12:38.326 -> 1
19:12:39.491 -> 4
19:12:40.488 -> 5
19:12:40.900 -> 6
19:12:41.759 -> 7
19:12:42.036 -> 8
19:12:42.380 -> 9
19:12:52.680 -> 8
19:12:55.055 -> 7
19:12:55.433 -> 8
19:12:55.673 -> 9

```

Figure 3: Programmed BlueTooth Serial Controller App



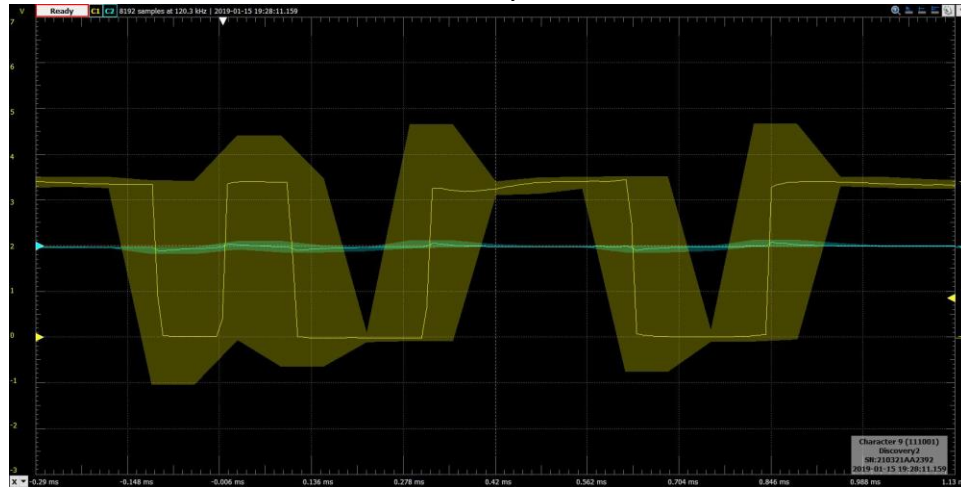
When each ASCII signal for 1 through 9 was sent the transmission is in binary. The decimal to ASCII to binary can be observed in table 1 shown below.

Table 1: Decimal numbers are displayed in the far left hand column. The center column shows the ASCII equivalent, and the right hand column shows the transmitted binary signal for each ASCII number.

Decimal	ASCII	Transmitted Binary Signal
1	49	110001
2	50	110010
3	51	110011
4	52	110100
5	53	110101

6	54	110110
7	55	110111
8	56	111000
9	57	111001

Figure 4: Oscilloscope waveform of a transmitted 9 signal; 57 in ASCII transmitted as 111001 in binary.



Part #3: Write a Custom Sketch

The next part of the lab calls for writing a custom sketch which allows for the program to determine when a button is pressed and when it is released. In order to accomplish this, a switch statement must be written. It was decided that in order to differentiate between when a button is pressed and when it is released the transmitted signal must be modified slightly. When a button is pressed it is to send a '?' after the transmitted bit, example when a '1' is pressed it will transmit '1?'. When a button is released another signal is to be transmitted which sends a '\$' after the number, therefore upon release a signal of '1\$' is to be sent. The goal of the switch statement is to evaluate the transmitted signal after the number to differentiate between a pressed and released button.

Modified_Bluetooth_Testing_HC_05.ino

```
const byte rxPin = 18;
const byte txPin = 19;
char message[256];
int index = 0;

void setup() {
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  Serial.begin(38400);
  // Usually 9600 for BT mode, although it is sometimes 38400:
```

```

Serial1.begin(9600);
}

void loop() {
  if(Serial1.available())
  {
    message[index] = (char)Serial1.read();
    if (message[index] == 0x3F) //Checks for a ? character
    {
      switch(message[0]){
        case '1':
          Serial.print("Button 1 Pressed");
          break;
        case '2':
          Serial.print("Button 2 Pressed");
          break;
        case '3':
          Serial.print("Button 3 Pressed");
          break;
        case '4':
          Serial.print("Button 4 Pressed");
          break;
        case '5':
          Serial.print("Button 5 Pressed");
          break;
        case '6':
          Serial.print("Button 6 Pressed");
          break;
        case '7':
          Serial.print("Button 7 Pressed");
          break;
        case '8':
          Serial.print("Button 8 Pressed");
          break;
        case '9':
          Serial.print("Button 9 Pressed");
          break;
        default :
          Serial.print("BREAK");
          break;
      }
    }
    index = 0;
    Serial.println();
  }
}

```

```

}
else if (message[index] == 0x24) { //Checks for a $ character
    switch(message[0]){
        case '1':
            Serial.print("Button 1 Released");
            break;
        case '2':
            Serial.print("Button 2 Released");
            break;
        case '3':
            Serial.print("Button 3 Released");
            break;
        case '4':
            Serial.print("Button 4 Released");
            break;
        case '5':
            Serial.print("Button 5 Released");
            break;
        case '6':
            Serial.print("Button 6 Released");
            break;
        case '7':
            Serial.print("Button 7 Released");
            break;
        case '8':
            Serial.print("Button 8 Released");
            break;
        case '9':
            Serial.print("Button 9 Released");
            break;
        default:
            Serial.print("BREAK");
            break;
    }
    index = 0;
    Serial.println();
}
else{
    index++; //Increases the index counter by one if neither a ? or a $ was encountered.
}
}
}
}

```


This code enables the differentiation of a pressed button from a released button. When a button is pressed it will send a signal with a number and a '?'. The switch statement is within a loop so that it can be executed repeatedly. After a signal has been transmitted it the switch statement is index to zero, it checks the second signal to see if the transmitted character is a '?' Or a '\$'. If the program does not first detect a '?' Then it will check for a '\$'. If it does not detect either of those characters, then the program knows that it has encountered a number and will increase the index by one. When it encounters either a '?' or a '\$' the program will then check the preceding character to determine which number was transmitted, then it prints the corresponding message to the serial monitor.

Figure 5: Microcontroller connected to Bluetooth device via breadboard.

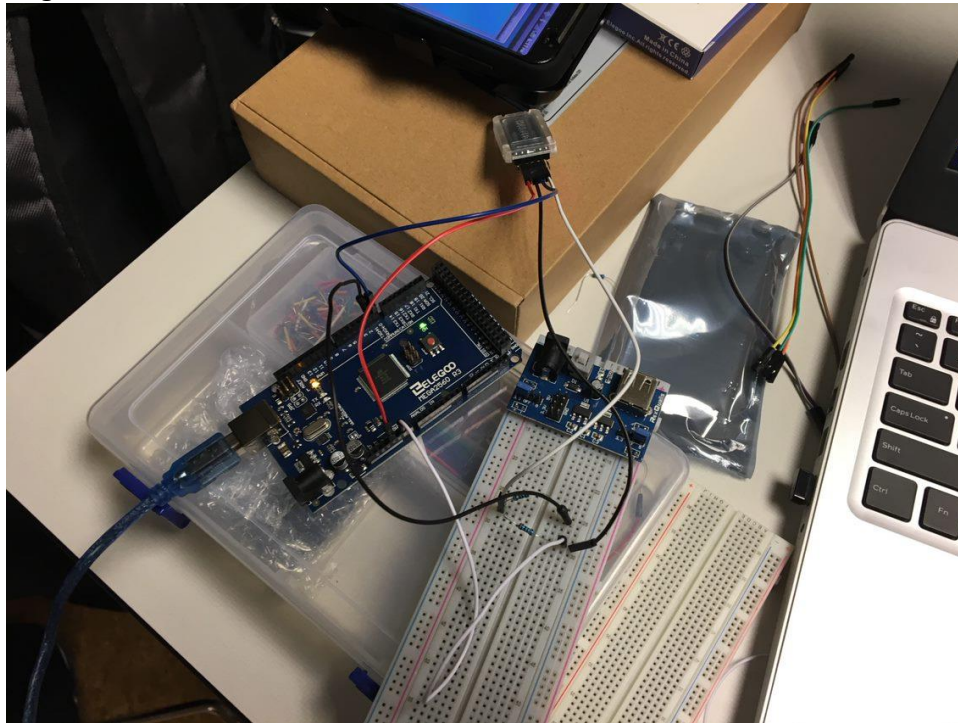


Figure 6: Oscilloscope waveform of a transmitted 1? character. The following binary waveform can be observed 10|10011|00|10|11111|00

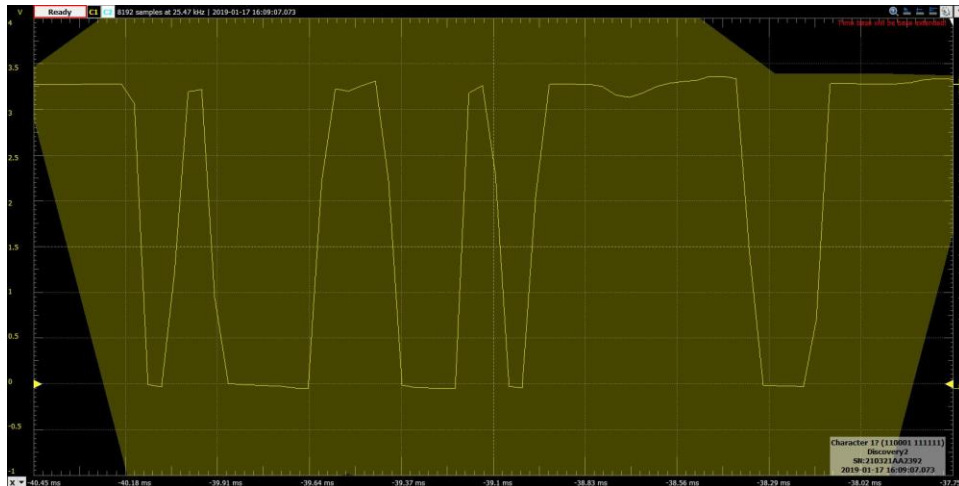
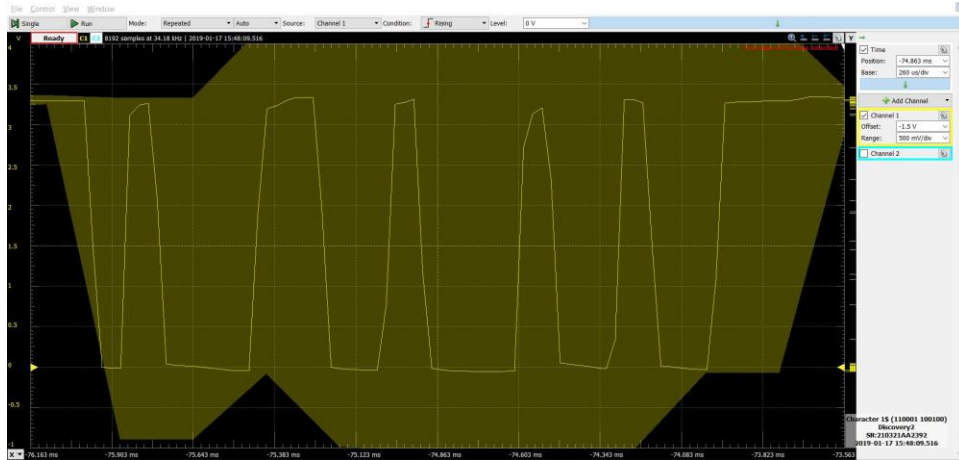


Figure 7: Oscilloscope waveform of a transmitted 1\$ character. The following binary waveform can be observed 10|10011|00|10|001001|00



In figures 5 and 6 the transmitted waveforms can be observed. Each transmitted character has a start bit which is 10, and an end bit which is 00, the bits in the middle are the actual ASCII character (in binary), although they are transmitted backwards due to the nature of serial communication.

Table 2: Transmitted signals for pushed and released buttons.

Decimal	ASCII	Transmitted Binary Signal
1?	49 63	10100011001000100100
2?	50 63	10010011001000100100
3?	51 63	10110011001000100100
4?	52 63	10001011001000100100
5?	53 63	10101011001000100100
6?	54 63	10011011001000100100
7?	55 63	10111011001000100100
8?	56 63	10000111001000100100

9?	57 63	10100111001000100100
1\$	49 36	10100011001011111100
2\$	50 36	10010011001011111100
3\$	51 36	10110011001011111100
4\$	52 36	10001011001011111100
5\$	53 36	10101011001011111100
6\$	54 36	10011011001011111100
7\$	55 36	10111011001011111100
8\$	56 36	10000111001011111100
9\$	57 36	10100111001011111100

Figure 8: Pressed button, note that the transmitted character is 1? as seen in the bottom center.

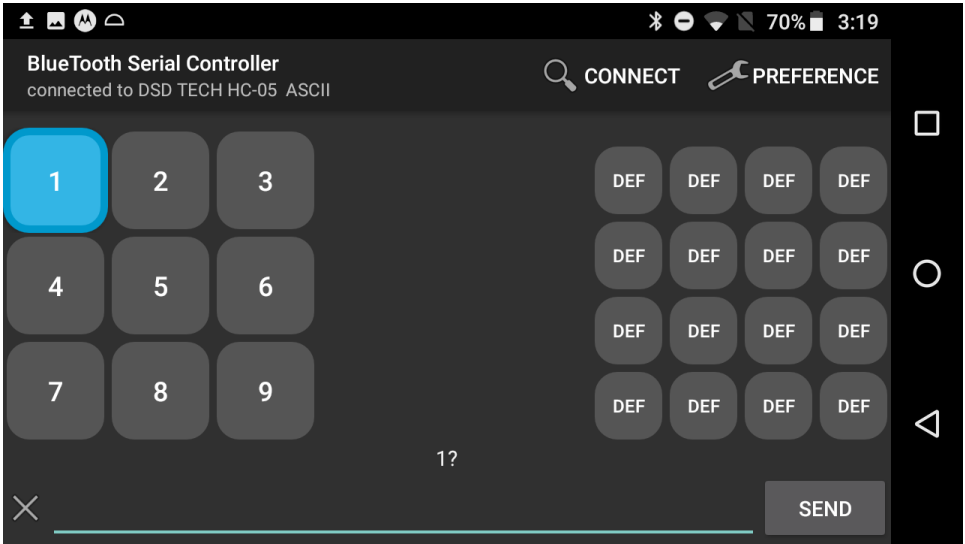


Figure 9: Released button, note that the transmitted character is 1\$ as seen in the bottom center.

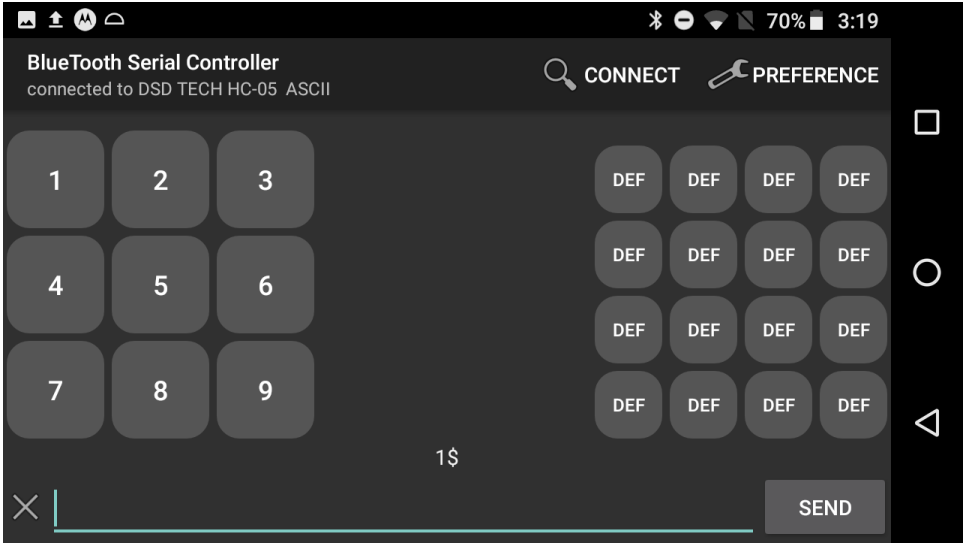
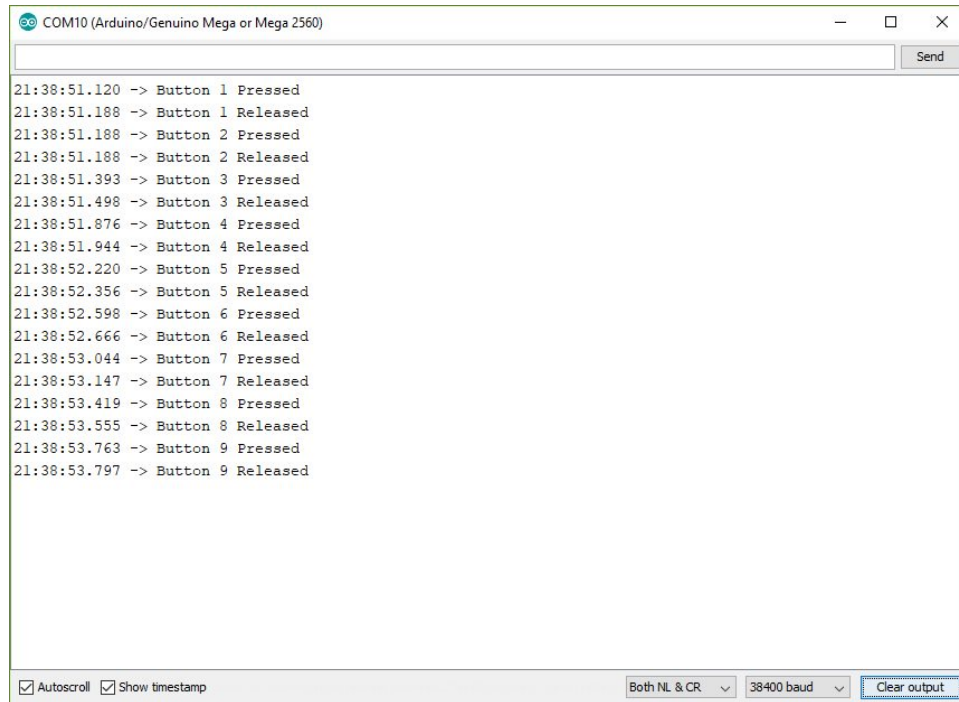


Figure 10: Serial monitor demonstrating the modified code detecting when a button is pressed or released.



The screenshot shows a serial monitor window titled "COM10 (Arduino/Genuino Mega or Mega 2560)". The window contains a list of messages indicating button presses and releases for 9 buttons. The messages are as follows:

```
21:38:51.120 -> Button 1 Pressed
21:38:51.188 -> Button 1 Released
21:38:51.188 -> Button 2 Pressed
21:38:51.188 -> Button 2 Released
21:38:51.393 -> Button 3 Pressed
21:38:51.498 -> Button 3 Released
21:38:51.876 -> Button 4 Pressed
21:38:51.944 -> Button 4 Released
21:38:52.220 -> Button 5 Pressed
21:38:52.356 -> Button 5 Released
21:38:52.598 -> Button 6 Pressed
21:38:52.666 -> Button 6 Released
21:38:53.044 -> Button 7 Pressed
21:38:53.147 -> Button 7 Released
21:38:53.419 -> Button 8 Pressed
21:38:53.555 -> Button 8 Released
21:38:53.763 -> Button 9 Pressed
21:38:53.797 -> Button 9 Released
```

At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", both of which are checked. To the right of these checkboxes are dropdown menus for "Both NL & CR" and "38400 baud", and a "Clear output" button.

Conclusion

Bluetooth communication works by sending serial ASCII signals back and forth between the master and slave device. These signals can be observed on an oscilloscope represented as high and low binary signals. Later a button press will need to be differentiated from when a button is released for purposes of controlling the robot. In order to accomplish this it was decided to send a unique character after the number. A '?' character denotes that a button is pressed, while a '\$' character denotes that the button has been released. The written program then checks the second bit to determine if it is a pressed or released button. In order to ensure that the program is working properly, the results were displayed both on the serial monitor and as a waveform on the oscilloscope.

Lab #2: Encoders and Motors

Objective:

Interface with an Atmega2560 Microcontroller to a pair of Wheel-Encoders and 4 DC-Motors, by instantiating a C++ encoder and motor classes to test the new devices.

Pins will be added to Adafruit Motor_Shield, and connected to top of AtMega2560 Microcontroller for control of DC-Motors and Wheel Encoders. The motor shield will allow connection of separate power and control between Microcontroller and Motor Shield

Required Materials:

Breadboard | AtMega2560 Microcontroller and Data Sheet | TinySine Wheel Encoder Kit for Robot Car | Jumper Wires | Oscilloscope | Arduino ISE Software | Markerfire 4-Wheel Robot Smart Car Chassis Kit] Adafruit Motor/Stepper/Servo Shield for Arduino v2.3 Kit

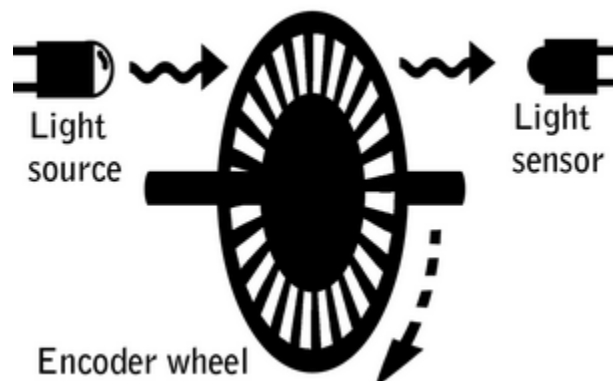
Part 1 – Wheel Encoders

First Step, One wheel encoder connected to power and ground to observe output of an wheel spin on oscilloscope. The TinySine Wheel Encoder has an 100kHz maximum PWM Frequency and 20mA consumption at 5V. Parameters to consider are it's 24mm diameter and 20 slots, will expect 20 rising or falling edges per a complete rotation.

- $360^\circ/20 = 18^\circ$ per edge
- Time between rising edges = 101ms
- Rotational Speed = $18^\circ/101\text{ms} = 178^\circ/\text{sec}$

Calculating the Robot Speed

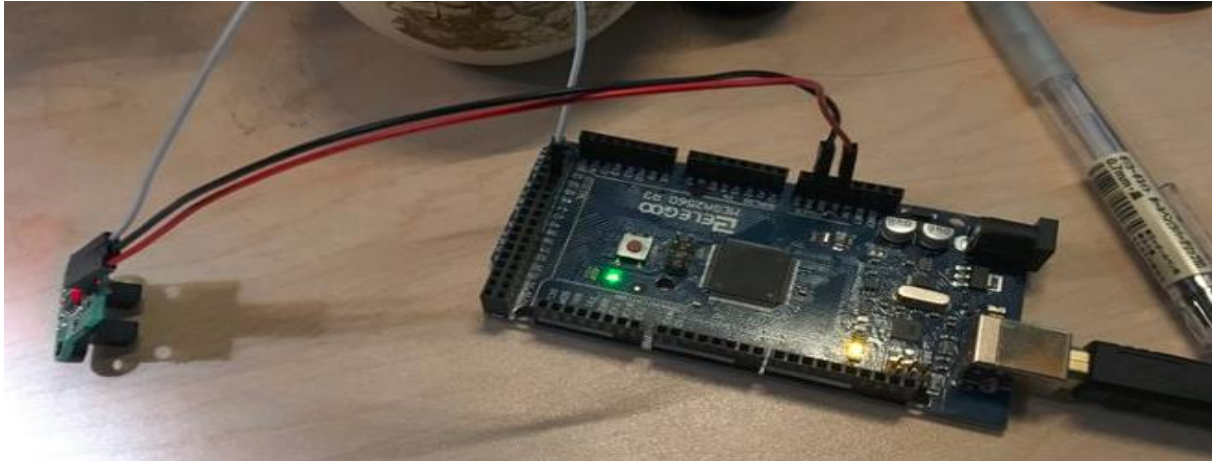
- Wheel Diameter = 66mm
- Circumference = $\pi d = 207.335\text{mm}$
- Wheel Travel over 360° turn = 207.35
- Linear Speed = $(207.35/360^\circ) \times (178^\circ)$
= 102.5mm/sec



The Encoder specs and calculations done on robot wheel speed can be used in the programming, the code will contain process to capture speed of wheel with the encoder rising or falling edge changes.

The Mega2560 Controller has multiple timers available, the encoder will send signal to an input pin on controller and an internal timer of controller to count the duration of the pulses.

Image: Wheel Encoder Connected to Mega2560 Controller



Encoder Class (encoder.cpp)

```

#include "Encoder.h"
#include "Arduino.h"

Encoder::Encoder(unsigned int pin) {
    //PI = 3.14159;
    // SLOTS = 20;
    // WHEEL_DIAMETER = 66.0;
    m_speed = 0.0;
    m_currentTime = 0;
    m_prevTime = 0;
    m_deltaTime = 0;
    m_edge = 0.0;
    pinMode(pin, INPUT);
}

void Encoder::updateTime(unsigned int time) {
    unsigned int degrees = 360;
    unsigned int SLOTS = 20;
    float WHEEL_DIAMETER = 66.0;
    float circumference = 0;
    m_currentTime = (float)time*.000016;
    m_prevTime = m_currentTime;
    m_edge = degrees / SLOTS;
    m_rSpeed = ((float)m_edge/m_currentTime);
    circumference = PI * WHEEL_DIAMETER;
    m_speed = (circumference / degrees);

    m_currentTime = 0;
    // Compute m_speed and handle Timer Overflow
}

double Encoder::getSpeed() {
    return(m_speed);
}

void Encoder::zeroSpeed() {
    m_speed = 0.0;
}

```

Encoder Header File (encoder.h)

```

#ifndef Encoder_h
#define Encoder_h

class Encoder {
private:
    double m_speed;
    double m_rSpeed;
    unsigned int m_currentTime;
    unsigned int m_prevTime;
    unsigned int m_deltaTime;
    float m_edge;
    // const float PI;
    // const unsigned int SLOTS;
    // const float WHEEL_DIAMETER;

public:
    Encoder(unsigned int pin);
    void updateTime(unsigned int time);
    double getSpeed();
    void zeroSpeed();
};

#endif

```

Encoder test sketch written to help capture a working Tiny Wheel encoder, to verify an Oscilloscope is used to show rising and falling edge of encoder output signal.

Image: Connected encoder to oscilloscope showing working output signal

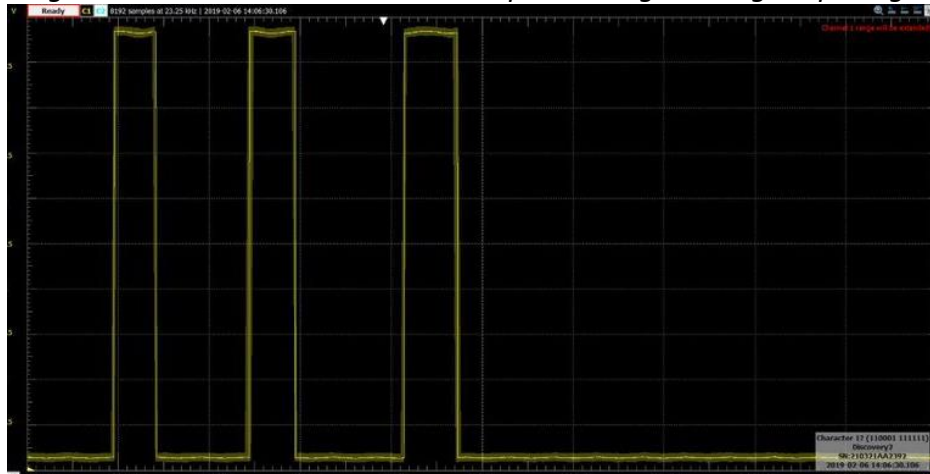
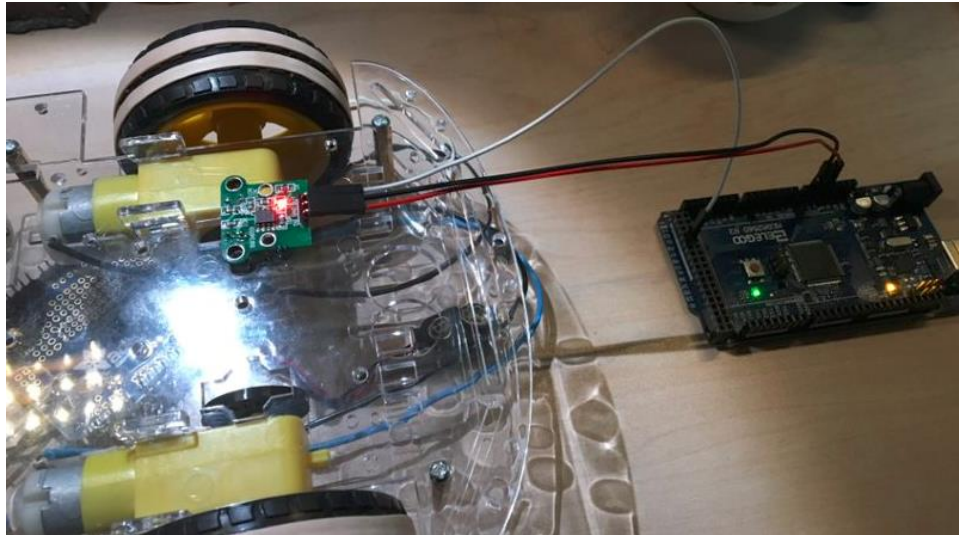


Image: Encoder connected to robot wheel, and test sketch simulated



Encoder Test Sketch (endoder_test.ino)

```
test_sketch
#include <encoder.h>

int val;
int encoder0PinA = 48;
int encoder0PinB = 49;
int encoder0Pos = 0;
int encoder0PinALast = LOW;
int n = LOW;

void setup() {
  pinMode (encoder0PinA, INPUT);
  pinMode (encoder0PinB, INPUT);
  Serial.begin (9600);
}

void loop() {
  n = digitalRead(encoder0PinA);
  if ((encoder0PinALast == LOW) && (n == HIGH)) {
    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos--;
    } else {
      encoder0Pos++;
    }
  }

  Serial.print (encoder0Pos);
  Serial.println ("/");
  encoder0PinALast = n;
}
```

Image: Serial Monitor, Test Sketch simulated with an input signal to Mega2560 Controller



Part 2 - Motor Class

A Motor class and header file written for 4-wheel motor control, each motor is wired to the Adafruit motor shield with an 6-9V DC power supply independent of Mega2560. First the motor shield is modified with pins soldered to board, this will allow the Motor-shield to attach right to Mega2560 board. A test sketch is written to allow a functional powering of each motor, the test will run motors forward and reverse with brief stop (release) inbetween.

Image: Motor-Shield soldered with Pins and attached to Meag2560

DC_MotorTest.ino: Drive all four motors; forward and reverse with release.

/*

This is a test sketch for Adafruit assembled Motor Shield for Arduino modified from sample test sketch for one motor. This sketch will run all for motors with operation in two directions and stop in between; Forward and Reverse.

For use with the Adafruit Motor Shield v2

*/

```
#include <Wire.h>
```

```
#include <Adafruit_MotorShield.h>
```

```
// Create the motor shield object with the default I2C address
```

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

```
// Selecting each 'port' M1, M2, M3 and M4
```

```
// front_right motor on port M1
```

```
Adafruit_DCMotor *MotorFR_RH = AFMS.getMotor(1);
```

```

// front left motor on port M2
Adafruit_DCMotor *MotorFR_LH = AFMS.getMotor(2);

// rear motors
// rear_left motor on port M3
Adafruit_DCMotor *MotorRR_LH = AFMS.getMotor(3);
// rear_right motor on port M4
Adafruit_DCMotor *MotorRR_RH = AFMS.getMotor(4);

void setup() {
  Serial.begin(9600);      // set up Serial library at 9600 bps
  Serial.println("Adafruit Motorshield v2 - DC Motor test!");

  AFMS.begin(); // create with the default frequency 1.6KHz
  //AFMS.begin(1000); // OR with a different frequency, say 1KHz

  // Set the speed to start, from 0 (off) to 255 (max speed)
  MotorFR_RH->setSpeed(150);
  MotorFR_RH->run(BACKWARD);
  MotorFR_LH->setSpeed(150);
  MotorFR_LH->run(FORWARD);
  // rear motors
  MotorRR_LH->setSpeed(150);
  MotorRR_LH->run(FORWARD);
  MotorRR_RH->setSpeed(150);
  MotorRR_RH->run(BACKWARD);
  // turn on motor
  MotorFR_RH->run(RELEASE);
  MotorFR_LH->run(RELEASE);
  // rear motors
  MotorRR_LH->run(RELEASE);
  MotorRR_RH->run(RELEASE);
}

void loop() {
  uint8_t i;
  Serial.print("Forward");

  MotorFR_RH->run(BACKWARD);
  MotorFR_LH->run(FORWARD);
  // rear motors
  MotorRR_LH->run(FORWARD);
  MotorRR_RH->run(BACKWARD);
  for (i=0; i<255; i++) {
    MotorFR_RH->setSpeed(i);

```

```

    MotorFR_LH->setSpeed(i);
    // rear motors
    MotorRR_LH->setSpeed(i);
    MotorRR_RH->setSpeed(i);
    delay(10);
}
for (i=255; i!=0; i--) {
    MotorFR_RH->setSpeed(i);
    MotorFR_LH->setSpeed(i);
    // rear motors
    MotorRR_LH->setSpeed(i);
    MotorRR_RH->setSpeed(i);
    delay(10);
}
Serial.print("Reverse");

MotorFR_RH->run(FORWARD);
MotorFR_LH->run(BACKWARD);
// rear motors
MotorRR_LH->run(BACKWARD);
MotorRR_RH->run(FORWARD);
for (i=0; i<255; i++) {
    MotorFR_RH->setSpeed(i);
    MotorFR_LH->setSpeed(i);

    // rear motors
    MotorRR_LH->setSpeed(i);
    MotorRR_RH->setSpeed(i);
    delay(10);
}
for (i=255; i!=0; i--) {
    MotorFR_RH->setSpeed(i);
    MotorFR_LH->setSpeed(i);
    // rear motors
    MotorRR_LH->setSpeed(i);
    MotorRR_RH->setSpeed(i);
    delay(10);
}
Serial.print("Release");
// front motors
MotorFR_RH->run(RELEASE);
MotorFR_LH->run(RELEASE);
// rear motors
MotorRR_LH->run(RELEASE);

```

```
MotorRR_RH->run(RELEASE);  
delay(1000);  
}
```

Conclusion

Rotary encoders are used on the wheels to determine the speed at which the vehicle is traveling. They work by causing periodic breaks in a light sensor which. When the sensor detects light it sends a high voltage signal to the microcontroller, and when the light is blocked by the encoder the device sends no voltage. The high and low signals are detected by the microcontroller and used to calculate the speed at which the car is traveling. The way the class is currently written relies on analog rather than digital (register level) encoding. This feature will need to be changed at a later , as digital encoding allows for a greater level of precision in performing time calculations rather than reading an analog signal.

The motor class uses pointers to the Adafruit_Motorshield which is a separate class that was written for the Adafruit motorshield. The .ino file creates four instances of the motor class so that four motors can be used. It then goes through the motorshield to interface with the motors. This is a necessary step because the microcontroller only has enough pins to support two motors. The class is written in a way to modify the polarity of the motors to work either forwards or backwards. This is convenient because it enables the motors to be wired in any way that is convenient. If the motors are accidentally wired in backwards it can be easily corrected in the .ino file. Furthermore, this will be a necessary feature when the car is programmed to drive autonomously so that it can reverse direction or steer left or right.

Lab #3: Servo, Line-Sensor and Range-Finder

Objectives

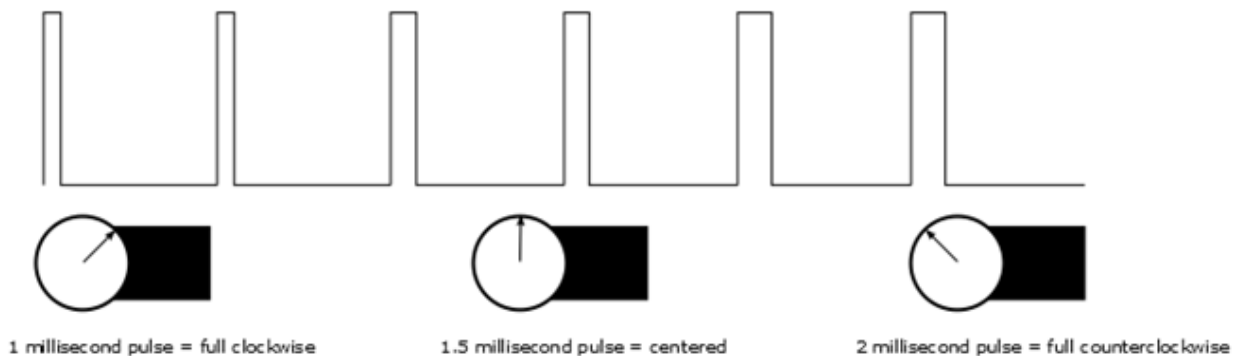
The purpose of this lab is to create individual classes for a servo, infrared light sensors, and an ultrasonic range finder. The servo will control a gripper on the front of the robot to pick up objects when commanded to do so. The range detector will send out an ultrasonic soundwave and record the amount of time it takes for the sound to bounce back. The maximum distance of the range detector can be adjusted by changing the length of time between sound impulses.

Required Materials:

Breadboard | AtMega2560 Microcontroller and Data Sheet | Horizontal Standard Gripper Kit A | Hitec 31311S HS-311 Servo Standard Universal Package | Jumper Wires | Oscilloscope | Arduino IDE Software | 5V Infrared Line Track Tracking Tracker Sensor Module | Mihappy 2pcs HC-SR04 Ultrasonic Distance Measuring Sensor Module

Part 1 – Gripper and Servo.

A servo will be used to control the gripper. Servos work by pulse width modulation. The longer the pulse, the further the servo rotates. When an object is connected to the rotating portion of the motor, it will move when the pulse width is modified. This will be used to move a gripper claw to grasp small balls and move them around.



Gripper class .h file

```
#ifndef Gripper_h // #ifndef means if not defined, so if Gripper_h isn't defined, start a loop
```

```

#define Gripper_h // Gripper_h is defined here.

#include <MegaServo.h> // include the MegaServo class in the header so it can be
referenced later.

class Gripper{ // Name the class that will be written.

private: // variables in the private section are only accessible within the class and
// can't be modified outside of it.
    bool m_position; // 0 is completely closed, 1 is completely open
    MegaServo *m_gripper; // create a pointer to the MegaServo class.

public: // variables in the public section are accessible outside of the class and
// can be modified outside of it.
    Gripper(unsigned int pin);
    void open();
    void close();
    bool isOpen();
};

#endif // close the statement started with the #ifndef command.

```

Gripper class .cpp file

```

#include "Gripper.h" // reference the Gripper header file.
#include "arduino.h" // reference the arduino header file
#define OPEN_ANGLE 170 // Define OPEN_ANGLE as a constant at 170 degrees.
#define CLOSED_ANGLE 0 // Define CLOSED_ANGLE as a constant at 0 degrees.

Gripper::Gripper(unsigned int pin) {
    m_gripper = new MegaServo; // instantiate a new instance of the MegaServo class
    m_gripper->attach(pin,1,.5); // pointing to the attach function, & declaring the pin,
// min and max values in microseconds; all within
// the MegaServo class.
    m_gripper->write(OPEN_ANGLE); // point to the write function in MegaServo,
// set the OPEN_ANGLE
    m_position = 1; // open position is on (Boolean value).
}

void Gripper::open() { // the new function in the Gripper class is called open; it does
// no return a value
    if(m_position) // if m_position = 1 (i.e. if it is true) then return the value of TRUE
        return;
    else { // if m_position is false (closed) then the value of m_gripper will be forced
// to be OPEN_ANGLE

```

```

    m_gripper->write(OPEN_ANGLE);
    m_position = 1;
}
}

void Gripper::close() { // this function is called close and does not return a value
    if(!m_position) // if not m_position (i.e. if m_position is false) then return that value
        return;
    else { // if m_position is not false (i.e. if it is true)
        m_gripper->write(CLOSED_ANGLE); // force m_gripper to be set to CLOSED_ANGLE
    }
    m_position = 0;
}

bool Gripper::isOpen() { // When the function isOpen() is called, return
// whatever value m_position is currently set to.
    return(m_position);
}

```

Gripper .ino file

```

#include <Gripper.h> // include the Gripper header so the program knows
// to use the Gripper class
#include <MegaServo.h> // include the MegaServo header so that the class can be used.
#define DELAY_TIME 1000 // Set a constant value for DELAY_TIME to be used in
// later functions

Gripper *myGripper; // create a pointer to the Gripper class that we can use later

void setup() {
    myGripper = new Gripper(10); // instantiate a new class of Gripper, set to pin 10
    Gripper(22);
    void open(); // call the open() function to change the PWM of the servo and
// open the gripper.
    delay(1000); // delay for 1 second before moving on
    void close(); // call the close() function from the .cpp file to close the gripper.
}

void loop() {
    if(myGripper->isOpen()){ // recall that myGripper is a pointer to the Gripper class, if
// the conditions for isOpen() are met then we will perform
// the conditions in the if statement.
        myGripper->close(); // close the gripper
        delay(DELAY_TIME); // Wait DELAY_TIME in milliseconds (recall that it was
// set to 1000 which is equivalent to 1 second)
    }
}

```



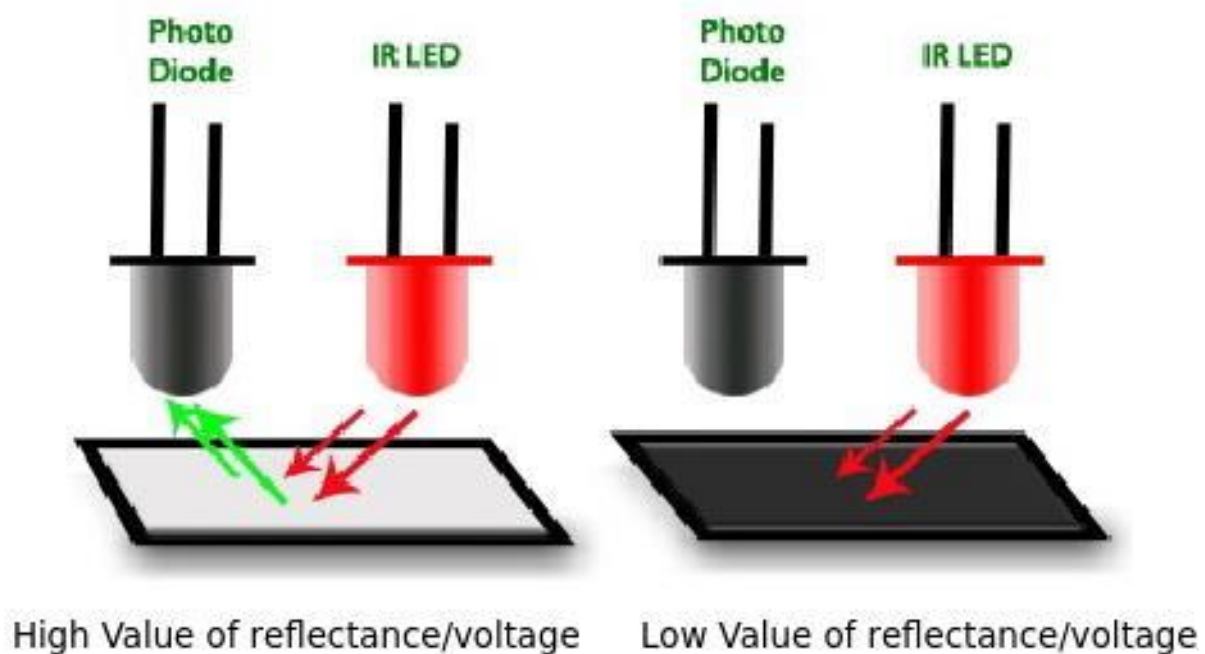
```

    }
    else {
        myGripper->open();
        delay(DELAY_TIME);
    }
}

```

Part 2 – Line Sensor

The line sensor uses infrared light to determine if it is positioned over a light or dark surface. If it is positioned over a light surface, light bounces back to the light sensor. If it is over a dark surface light does not bounce back. The light sensor is connected to a 5volt power source to power the light and the receiver. When the sensor detects light is being bounced back, it sends a high (5volt) signal back to the microcontroller. When light is not reflected back, it sends 0volts back to the microcontroller.



Line detector class .h file

```

#ifndef lineSensor_h // #ifndef denotes not defined, lineSensor_h isn't defined so start a
loop
#define lineSensor_h // lineSensor_h is defined here.

class lineSensor { // Name the class that will be written.

```

```

private:
    bool m_white; // for detecting if the sensor is over white (1) or black (0)
    int m_pin; // stores the value of the pin that connects the microcontroller to the sensor

public:
    lineSensor(unsigned int pin);
    bool whiteTest(); // this returns the value to the .ino file if the sensor detects black or
white
};

#endif // close the statement started with the #ifndef command.

```

Line detector .cpp file

```

#include "lineSensor.h" // reference the header file
#include "arduino.h" // reference arduino's header file

lineSensor::lineSensor(unsigned int pin) { // create a new function which takes a pin as input
    pinMode(pin,INPUT); // declare the pin that will be used, and set it as an input
    m_white = 1; // m_white is a boolean value, by default it will be set to TRUE.
    m_pin = pin; // Save the input pin as an integer value for later use
}

bool lineSensor::whiteTest() { // create a new function to return the value from the sensor
    m_white = digitalRead(m_pin); // m_white is set to whatever value digitalRead gives it.
    return(m_white); // the value of m_white is returned to the microcontroller.
}

```

Line detector .ino file

```

#include <lineSensor.h>

bool result = 0; // initialize a boolean value to be used later as a global variable

lineSensor *rightSensor; // create a pointer to be used later.

void setup() {
    rightSensor = new lineSensor(12); //initialize a new instance of lineSensor class set to pin 12
    Serial.begin(9600); // start the serial monitor tool
}

void loop() {

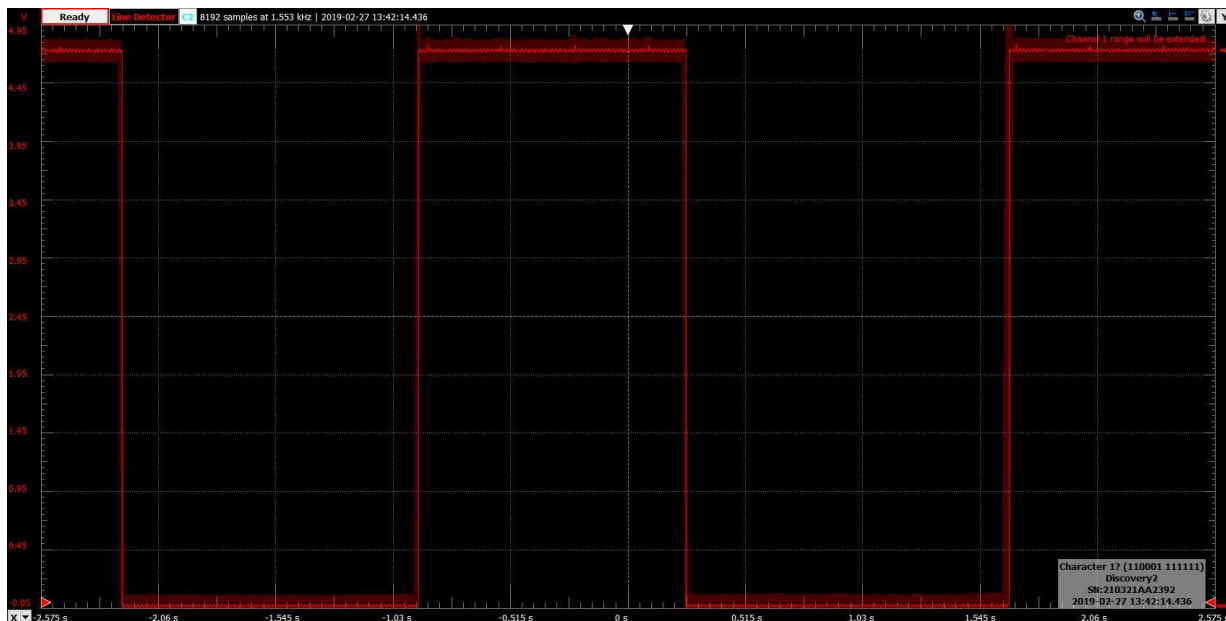
```

```

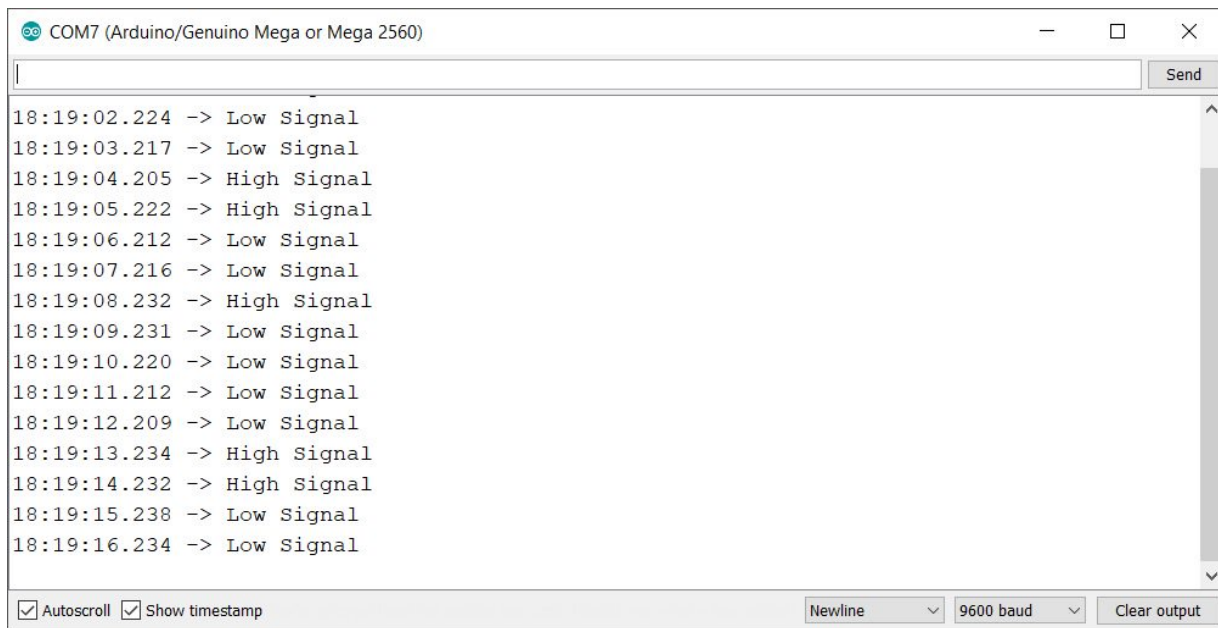
result = rightSensor->whiteTest(); // point to the whiteTest() function, and assign the value
// to the global variable to be the result.
if(result == 1) { // if the result is TRUE
  Serial.println("High Signal"); // print "High Signal" to the serial monitor.
  delay(1000); // wait 1 second
}
else { // if the result is not TRUE (I.e. if the result is FALSE)
  Serial.println("Low Signal"); // print "Low Signal" to the serial monitor.
  delay(1000); // wait 1 second.
}
}

```

Waveform of a high signal when light is reflected back, and a low signal when light is not reflected back.



Serial monitor output demonstrating a high and low signal.



```
COM7 (Arduino/Genuino Mega or Mega 2560)
18:19:02.224 -> Low Signal
18:19:03.217 -> Low Signal
18:19:04.205 -> High Signal
18:19:05.222 -> High Signal
18:19:06.212 -> Low Signal
18:19:07.216 -> Low Signal
18:19:08.232 -> High Signal
18:19:09.231 -> Low Signal
18:19:10.220 -> Low Signal
18:19:11.212 -> Low Signal
18:19:12.209 -> Low Signal
18:19:13.234 -> High Signal
18:19:14.232 -> High Signal
18:19:15.238 -> Low Signal
18:19:16.234 -> Low Signal
[Autoscroll] [Show timestamp] [Newline] [9600 baud] [Clear output]
```

Part 3 – Ultrasonic Range Finder

The ultrasonic range finder sends out an ultrasonic signal and uses the amount of time it takes for the soundwave to bounce back to calculate the distance between the sensor and the object. The maximum distance that the range finder can detect can be adjusted by modifying the pulse width. If the length of time it takes for a sound wave to bounce back exceeds the pulse width, then by default the microcontroller will assume that there is an object at the maximum distance away, unless otherwise specified.

Range Finder .h file

```
#ifndef Range_Echo_h // if Range_Echo_h is not defined...
#define Range_Echo_h // define Range_Echo_h

#if ARDUINO >= 100 // this block of code dictates which .h file to include
#include "Arduino.h" // given certain conditions
#else
#include "WProgram.h"
#endif

#define CM 1 // instantiate CM as a constant equal to 1
#define INC 0 // instantiate INC as a constant equal to 0

class Range_Echo{
public:
Range_Echo(int TP, int EP);
```

```

Range_Echo(int TP, int EP, long TO);
long Timing();
long Ranging(int sys);

private:
int Trig_pin;
int Echo_pin;
long Time_out;
long duration,distance_cm,distance_inc;
};

#endif // close the #ifndef statement

```

Range Finder .cpp file

```

#if ARDUINO >= 100 // decide which header file to include in the .cpp file
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include "Range_Echo.h" // also include the Range_Echo header file

Range_Echo::Range_Echo(int TP, int EP){
pinMode(TP,OUTPUT); // Declare the output pin
pinMode(EP,INPUT); // Declare the input pin
Trig_pin=TP; // save the output pin as an integer
Echo_pin=EP; // save the input pin as an integer
Time_out=3000; // sets the range that the device can detect (3000 μs = 50cm;
// 27000 μs = 450cm)
}

Range_Echo::Range_Echo(int TP, int EP, long TO){
pinMode(TP,OUTPUT);
pinMode(EP,INPUT);
Trig_pin=TP;
Echo_pin=EP;
Time_out=TO;
}

long Range_Echo::Timing(){ // create a new function called Timing()
digitalWrite(Trig_pin, LOW); // Write a low signal to the trig pin
delayMicroseconds(2); // wait 2 microseconds
digitalWrite(Trig_pin, HIGH); // have Trig_pin write a high signal

```

```

delayMicroseconds(10); // wait 10 microseconds
digitalWrite(Trig_pin, LOW); // have Trig_pin write low again
duration = pulseIn(Echo_pin, HIGH, Time_out);
if ( duration == 0 ) { // if duration is equal to 0, return that value
    duration = Time_out; }
return duration;
}

long Range_Echo::Ranging(int sys){
Timing();
if (sys) { // if the input parameter is true, do this block of code
    distance_cm = duration // 29 ÷ 2 = 14.5;
    return distance_cm;
}
else { // if the input parameter is not true, do this block of code
    distance_inc = duration // 74 ÷ 2 = 37;
    return distance_inc;
}
}

```

Ultrasonic range finder .ino file

```

#include <Range_Echo.h> // Include the Range_Echo header so the program knows to use
                        // the Range_Echo.cpp file

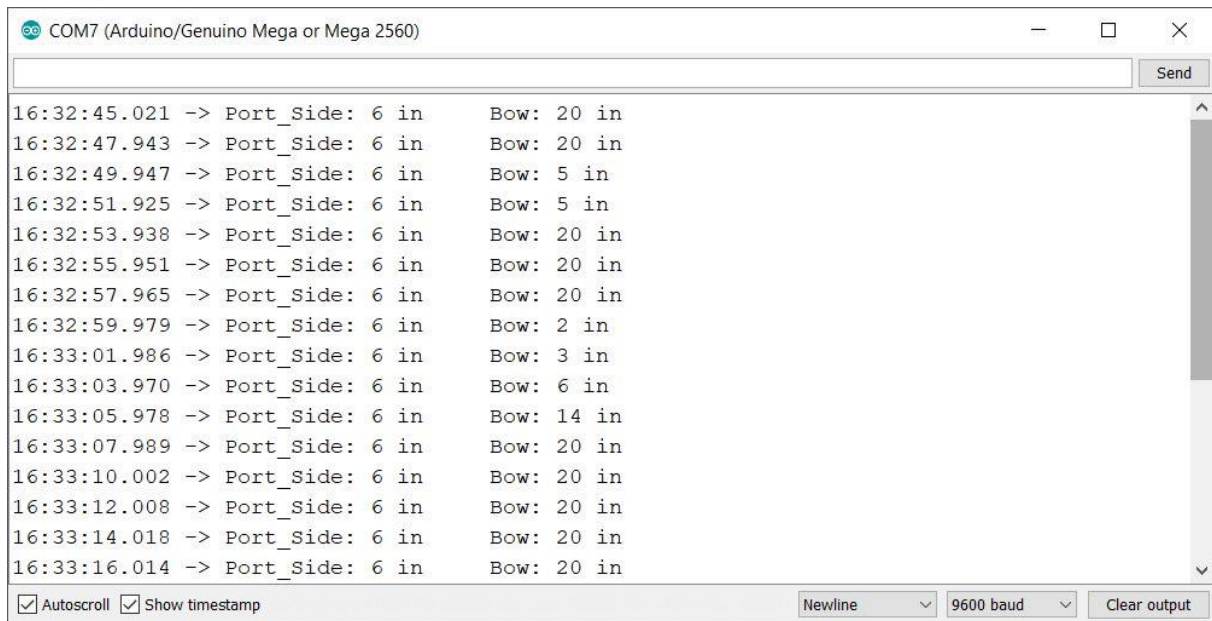
Range_Echo echoleft(8,9); // (Trig PIN,Echo PIN)
Range_Echo echofront(10,11); // (Trig PIN,Echo PIN)

void setup() {
    Serial.begin(9600); // Initialize the serial monitor tool
}

void loop()
{
    Serial.print("Port_Side: "); // Print Port_Side (left) regardless of what is detected
    Serial.print(echoleft.Ranging(INC)); // print the distance the range finder detects
    Serial.print(" in  "); // Print the units, in this case inches
    delay(1000); // wait 1 second
    Serial.print("Bow: "); // print Bow: (front) regardless of what is detected.
    Serial.print(echofront.Ranging(INC)); // print the distance the range finder detects
    Serial.println(" in" ); // Print the units, in this case inches
    delay(1000); // wait 1 second
}

```

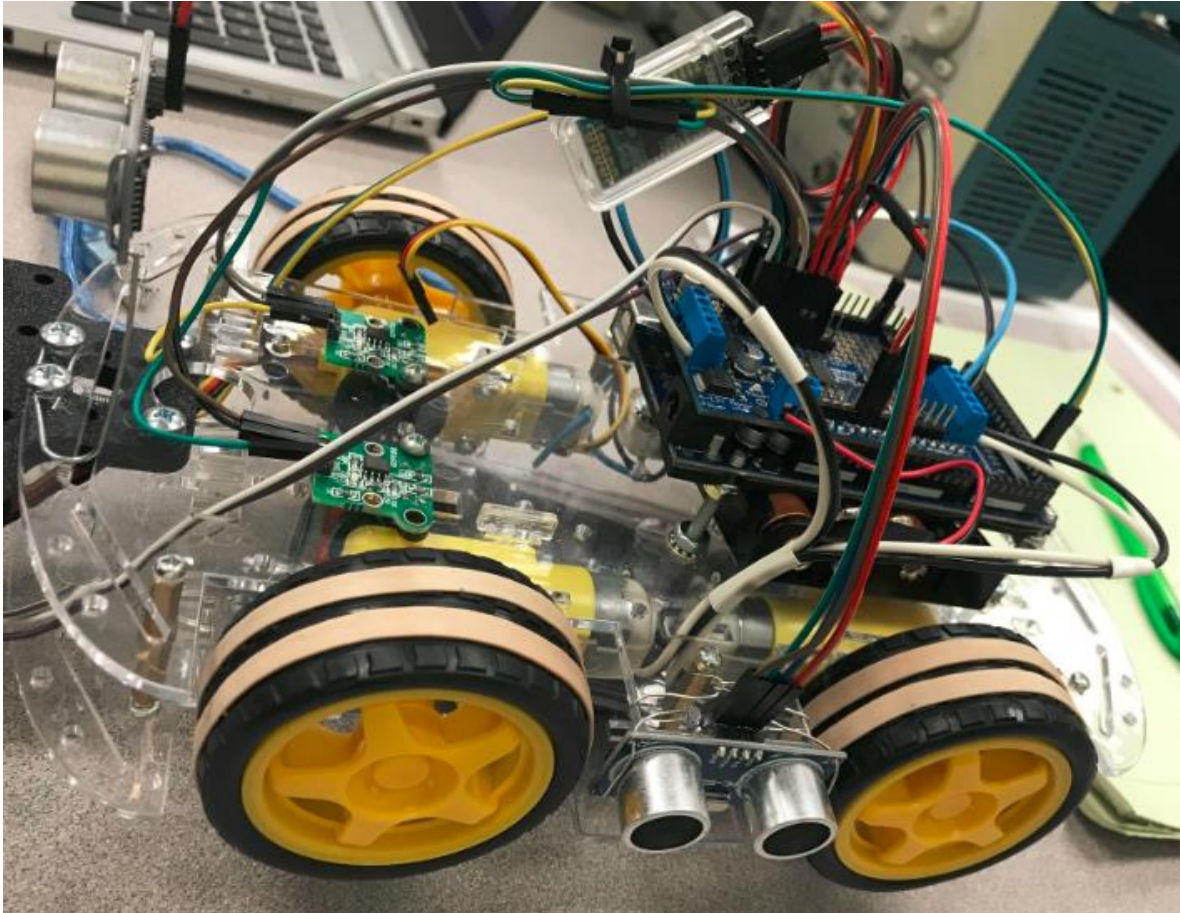
Serial monitor output for the Range_Echo sketch demonstrating the ability for the sketch to receive input from two devices independently at various distances.



The screenshot shows a serial monitor window titled "COM7 (Arduino/Genuino Mega or Mega 2560)". The window contains a list of 15 lines of data. Each line represents a timestamp followed by two sensor readings: "Port_Side" and "Bow". The "Port_Side" readings are consistently "6 in". The "Bow" readings vary, including "20 in", "5 in", "2 in", "3 in", "6 in", and "14 in". At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", both of which are checked. To the right of these checkboxes are dropdown menus for "Newline" and "9600 baud", and a "Clear output" button.

```
16:32:45.021 -> Port_Side: 6 in      Bow: 20 in
16:32:47.943 -> Port_Side: 6 in      Bow: 20 in
16:32:49.947 -> Port_Side: 6 in      Bow: 5 in
16:32:51.925 -> Port_Side: 6 in      Bow: 5 in
16:32:53.938 -> Port_Side: 6 in      Bow: 20 in
16:32:55.951 -> Port_Side: 6 in      Bow: 20 in
16:32:57.965 -> Port_Side: 6 in      Bow: 20 in
16:32:59.979 -> Port_Side: 6 in      Bow: 2 in
16:33:01.986 -> Port_Side: 6 in      Bow: 3 in
16:33:03.970 -> Port_Side: 6 in      Bow: 6 in
16:33:05.978 -> Port_Side: 6 in      Bow: 14 in
16:33:07.989 -> Port_Side: 6 in      Bow: 20 in
16:33:10.002 -> Port_Side: 6 in      Bow: 20 in
16:33:12.008 -> Port_Side: 6 in      Bow: 20 in
16:33:14.018 -> Port_Side: 6 in      Bow: 20 in
16:33:16.014 -> Port_Side: 6 in      Bow: 20 in
```

An image of the robot featuring the ultrasonic range finder featured on the left-hand side.



Conclusion

Servos work by modifying the input pulse width to adjust the amount that the servo has rotated. If something is attached to the servo, it will naturally move along with it. The gripper will be adjusted using a predefined program controlled by a Bluetooth signal. It will open and close based on predefined parameters to adjust the amount that the servo moves.

The infrared line sensor emits a constant amount of infrared light. When light is reflected back, the sensor will return a high signal to the microcontroller, when light is not reflected it will send a 0 volt signal. The microcontroller will be programmed to use feedback from the light sensors to make the adjustments in its direction while driving.

The ultrasonic range finder emits pulses of noise in the inaudible ultrasonic frequencies. It then uses the length of time it takes for a sound wave to bounce back to calculate the distance at which an object is in front of the device. The distance at which the device can detect an object can be modified based on the frequency of the high signals fed into the device, which happen at regular intervals. The more time between pulses, the further the range that the device can

detect to a maximum of 450 cm (177.2 inches). If the device times out before a soundwave is returned to the device, it assumes a maximum distance.

Lab # 4: Motor Speed Control and Line Following

Objective

To create two classes which enable the robot to drive autonomously by interacting with the motor and line sensor classes. As well as to implement the classes in a .ino file to get the robot up to a functional level. Both classes will be implemented into the same .ino file.

Required Materials:

Breadboard | AtMega2560 Microcontroller and Data Sheet | Jumper Wires | Oscilloscope | Arduino ISE Software | 5V Infrared Line Track Tracking Tracker Sensor Module | Robot Chassis with Motors and Encoders | Milhappy 2pcs HC-SR04 Ultrasonic Distance Measuring Sensor Module.

Part 1 - PID Motor Speed Control Class

The speed control class works by receiving input from the wheel encoders to calculate the speed at which the wheels are turning and comparing it against a predefined speed setting then adjusting the PWM of the motors accordingly. If the wheels slow down, such as when going uphill, the class will increase the PWM of the motors automatically. If the wheels speed up, such as when going downhill, the class will decrease the PWM accordingly.

speedcontrol.h

```
#ifndef speedcontrol_h
#define speedcontrol_h
#include "Arduino.h"
#include <encoder.h>
#include <motor.h>

class speedcontrol{
private:
    double m_setPoint;
    double m_currentSpeed;
    double m_prevSpeed;
    double m_sampleTime;
    double m_error;
    double m_errorIntegral;
    double m_gainProportional;
```

```

double m_gainIntegral;
double m_gainDifferential;
double m_integratorLimit;
bool m_enable;
encoder *m_encoder;
motor *m_motor1;
motor *m_motor2;
HardwareSerial *m_serialPort;

public:
    speedcontrol(HardwareSerial *port, encoder *encoderPtr, motor *motorPtr1, motor
                *motorPtr2, double sampleTime);
    void setDesiredSpeed(double speed);
    double getDesiredSpeed();
    void enable();
    void disable();
    void update();
    void setPGain(double gain);
    void setIGain(double gain);
    void setDGain(double gain);
    void setILimit(double limit);
};

#endif

```

speedcontrol.cpp

```

#include <speedcontrol.h>

speedcontrol::speedcontrol(HardwareSerial *port, encoder *encoderPtr, motor
                *motorPtr1, motor *motorPtr2, double sampleTime) {
    m_setPoint = 0.0;
    m_enable = 0;
    m_error = 0.0;
    m_currentSpeed = 0.0;
    m_prevSpeed = 0.0;
    m_errorIntegral = 0.0;
    m_sampleTime = sampleTime;
    m_motor1 = motorPtr1;
    m_motor2 = motorPtr2;
    m_encoder = encoderPtr;
    m_gainIntegral = 100.0;
    m_gainProportional = 10.0;
    m_gainDifferential = 25.0;
    m_integratorLimit = 50.0;
    m_serialPort = port;
}

```

```

    }
void speedcontrol::setDesiredSpeed(double speed) {
    m_setPoint = speed;
}
double speedcontrol::getDesiredSpeed() {
    return(m_setPoint);
}
void speedcontrol::enable() {
    m_enable = 1;
}
void speedcontrol::disable() {
    m_enable = 0;
}

void speedcontrol::update() {
    m_prevSpeed = m_currentSpeed;
    m_error = (m_currentSpeed - m_setPoint)/100;
    m_errorIntegral += m_error*m_sampleTime;

    if(m_errorIntegral > m_integratorLimit) {
        m_errorIntegral = m_integratorLimit;
    }
    else if(m_errorIntegral < -m_integratorLimit) {
        m_errorIntegral = -m_integratorLimit;
    }
    double pTerm = m_gainProportional*m_error;
    double iTerm = m_gainIntegral*m_errorIntegral;
    double dTerm = m_gainDifferential*(m_currentSpeed-m_prevSpeed)/m_sampleTime;
    double newDrive = -(iTerm + pTerm - dTerm);
    newDrive = 255.0*newDrive/30.0;
    if(newDrive < 0.0)
        newDrive = 0.0;
    else if(newDrive > 255.0)
        newDrive = 255.0;
    if(m_enable) {
        if(newDrive == 0.0){
            m_motor1 -> setDrive(0, RELEASE);
            m_motor2 -> setDrive(0, RELEASE);
        }

    else {
        m_motor1 -> setDrive((int)newDrive, FORWARD);
        m_motor2 -> setDrive((int)newDrive, FORWARD);
    }
}

```

```

    m_serialPort -> print(m_error);
    m_serialPort -> print(", ");
    m_serialPort -> print(m_errorIntegral);
    m_serialPort -> print(", ");
    m_serialPort -> print(newDrive/255.0);// Normalize
    m_serialPort -> print("\r");
  }
}

```

speedcontrol.ino

```

#include <encoder.h>
#include <motor.h>
#include <speedcontrol.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <HardwareSerial.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();

volatile int toggle_motor_speed = 1;
volatile double speedSetting1 = 70.0;
volatile double speedSetting2 = 50.0;
volatile double speedSetting3 = 70.0;
volatile double speedSetting4 = 50.0;
volatile double samplingRate = 0.02;
volatile int update_speed = 0;
volatile double setspeed = 0;
volatile double setspeed1 = 0;

encoder *rightEncoder = new encoder(20, 66, 48);
encoder *leftEncoder = new encoder(20, 66, 49);
motor *LRMotor = new motor(&AFMS, 3, 1); //left rear
motor *RRMotor = new motor(&AFMS, 4, 0); //right rear
motor *RFMotor = new motor(&AFMS, 1, 0); //right front
motor *LFMotor = new motor(&AFMS, 2, 1); //left front
speedcontrol *rightSpeed = new speedcontrol(&Serial, rightEncoder, RFMotor, RRMotor,
    samplingRate);
speedcontrol *leftSpeed = new speedcontrol(&Serial, leftEncoder, LFMotor, LRMotor,
    samplingRate);

void setup() {
  Serial.begin(115200);
  noInterrupts(); // Disable the ISR's
  TCCR3A = 0;

```

```

TCCR3B = 0x0A; // Prescale and CIC mode
OCR3A = 0x9C40; // set the compare A Register
TIMSK3 = 0x02; // enable the output compare A Match Interrupt
TCNT3 = 0x00;

// Enable input capture 4 (ICNC4 enable noise canceler, ICES4 trigger on rising edge, CS42/0
no prescaler)
TCCR4A = 0x00; // Initialize register to 0
TCCR4B = 0x00; // Initialize register to 0
TCCR4B = (1 << ICNC4) | (1 << ICES4) | (0 << WGM42) | (1 << CS42) | (0 << CS41) | (1 <<
    CS40);
TCCR4C = 0x00; // Initialize register to 0
TIMSK4 = (1 << ICIE4); // Input capture interrupt enabled
TCNT4 = 0x00; // Sets the Timer to 0

// Enable input capture 5 (ICNC5 enable noise canceler, ICES5 trigger on rising edge, CS52/0
no prescaler)
TCCR5A = 0x00; // Initialize register to 0
TCCR5B = 0x00; // Initialize register to 0
TCCR5B = (1 << ICNC5) | (1 << ICES5) | (0 << WGM52) | (1 << CS52) | (0 << CS51) | (1 <<
    CS50);
TCCR5C = 0x00; // Initialize register to 0
TIMSK5 = (1 << ICIE5); // Input capture interrupt enabled
TCNT5 = 0x00; // Sets the Timer to 0
interrupts(); // enable interrupts.
rightSpeed -> enable();
leftSpeed -> enable();
AFMS.begin();
}

void loop() {
    setSpeed = rightEncoder -> getSpeed();
    setSpeed1 = leftEncoder -> getSpeed();
    if (setSpeed < 5 && setSpeed1 < 5)
        setSpeed = 0;
    if (update_speed == 1) {
        rightSpeed -> update(setSpeed);
        leftSpeed -> update(setSpeed);
        update_speed = 0;
    }

    if (toggle_motor_speed == 1) {
        rightSpeed -> setDesiredSpeed(speedSetting1);
    }
}

```

```

    leftSpeed -> setDesiredSpeed(speedSetting3);
}
else {
    rightSpeed -> setDesiredSpeed(speedSetting2);
    leftSpeed -> setDesiredSpeed(speedSetting4);
}
}

ISR(TIMER3_COMPA_vect) {
    update_speed = 1;
}
ISR(TIMER5_OVF_vect) {
    if (toggle_motor_speed == 1)
        toggle_motor_speed = 0;
    else
        toggle_motor_speed = 1;
    rightEncoder -> zeroSpeed();
}
ISR(TIMER4_OVF_vect) {
    if (toggle_motor_speed == 1)
        toggle_motor_speed = 0;
    else
        toggle_motor_speed = 1;
    leftEncoder -> zeroSpeed();
}
ISR(TIMER5_CAPT_vect) {
    rightEncoder -> updateTime(ICR5);
    TCNT5 = 0x00;
}
ISR(TIMER4_CAPT_vect) {
    leftEncoder -> updateTime(ICR4);
    TCNT4 = 0x00;
}

```

Part 2 - Line Follower Control Class

The line follower class will receive input from two infrared light sensors mounted on the sides of the robot. Using the input from both devices it will adjust the course of the robot accordingly. Since the robot will be following a black line on white paper, whenever the light detectors do not sense any light reflected back from the emitted light it will send a signal to the robot to adjust its course. If the light sensor detects a black line on the left, it will turn the robot left. If it senses a black line on the right it will direct the robot to turn right to correct its course.

linecontrol.h

```
#ifndef lineControl_h
#define lineControl_h

#include "linesensor.h"
#include "speedcontrol.h"

class lineControl{
private:
    speedcontrol *m_leftSpeed;
    speedcontrol *m_rightSpeed;
    linesensor *m_leftFlag;
    linesensor *m_rightFlag;

public:
    lineControl(linesensor *leftPTR, linesensor *rightPTR, speedcontrol *leftSpeed,
                speedcontrol *rightSpeed);
    void detectNone();
    void updateSpeed();
    void detectLeft();
    void detectRight();
};

#endif
```

linecontrol.cpp

```
#include "lineControl.h"
#include "linesensor.h"
#include "speedcontrol.h"

lineControl::lineControl(linesensor *leftPTR, linesensor *rightPTR, speedcontrol *leftSpeed,
                          speedcontrol *rightSpeed){
    m_leftFlag = leftPTR;
    m_rightFlag = rightPTR;
    m_leftSpeed = leftSpeed;
    m_rightSpeed = rightSpeed;
}

void lineControl::detectNone(){
    m_leftFlag -> LineStatus();
    m_rightFlag -> LineStatus();
    m_leftSpeed -> enable();
    m_rightSpeed -> enable();
}
```

```

void lineControl::updateSpeed(){
    if(m_leftFlag == 0 && m_rightFlag == 0){
        m_leftSpeed -> update();
        m_rightSpeed -> update();
    }
}

void lineControl::detectLeft(){
    if(m_leftFlag == 0){
        m_rightSpeed -> disable();
    }
}

void lineControl::detectRight(){
    if(m_rightFlag == 0){
        m_leftSpeed -> disable();
    }
}

```

linecontrol.ino

```

#include <Adafruit_MotorShield.h>
#include <lineControl.h>
#include <linesensor.h>
#include <motor.h>
#include <Range_Echo.h>
#include <Wire.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();

const float UNIT_CONVERSION = 0.003937;

volatile float frontSensorDistance = 0.0;
bool rightLineSensor = 1;
bool leftLineSensor = 1;

linesensor *leftSensor;
linesensor *rightSensor;

motor *RFMotor;
motor *LFMotor;
motor *LRMotor;
motor *RRMotor;

```



```

Range_Echo *frontRange;

void setup() {
  leftSensor = new linesensor(13);
  rightSensor = new linesensor(12);
  frontRange = new Range_Echo(10, 11);
  RFMotor = new motor(&AFMS, 1, 0); //right front
  LFMotor = new motor(&AFMS, 2, 1); //left front
  LRMotor = new motor(&AFMS, 3, 1); //left rear
  RRMotor = new motor(&AFMS, 4, 0); //right rear

  frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;

  rightLineSensor = rightSensor;
  leftLineSensor = leftSensor;
}

void loop() {
  Serial.begin(115200);
  AFMS.begin();

  frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;

  while ((frontSensorDistance > 1.0) && (frontSensorDistance < 7.5)) {
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;
  }

  if (frontSensorDistance > 8.0) {

    if (digitalRead(13) == HIGH) {
      LFMotor->setDrive(0.0, FORWARD);
      LRMotor->setDrive(0.0, FORWARD);
      RFMotor->setDrive(230.0, FORWARD);
      RRMotor->setDrive(230.0, FORWARD);
    }

    else {
      LFMotor->setDrive(230.0, FORWARD);
      LRMotor->setDrive(230.0, FORWARD);
      RFMotor->setDrive(0.0, FORWARD);
    }
  }
}

```

```
RRMotor->setDrive(0.0, FORWARD);
}

if (digitalRead(12) == HIGH) {
  LFMotor->setDrive(230.0, FORWARD);
  LRMotor->setDrive(230.0, FORWARD);
  RFMotor->setDrive(0.0, FORWARD);
  RRMotor->setDrive(0.0, FORWARD);
}
else {
  LFMotor->setDrive(0.0, FORWARD);
  LRMotor->setDrive(0.0, FORWARD);
  RFMotor->setDrive(230.0, FORWARD);
  RRMotor->setDrive(230.0, FORWARD);
}
}
```

Conclusion

The speed control works by directly reading and manipulating the registers associated with the wheel encoders to calculate the amount of time that has passed between two events. The event is started when the prescaler is initialized to zero and then another register measures the number of high signal events (i.e. the encoder slits pass by the light sensor) since the event was initialized. The time is compared against a preset value, and the PWM sent to the motor is adjusted accordingly. Two encoders are used, one for the front left wheel and one for the front right wheel. The TCCR4 registers are used for the right wheel, while the TCCR5 registers are used for the left wheel.

The line follower class works by using pointers to reference other classes both to receive and feed information to other classes and direct information accordingly. It interacts with both the speed control and the speed control and line sensor classes. It receives input from the line sensor class regarding positional information about the robot, then sends information to the speed control classes which then in turn manipulates the motors to adjust their speed to redirect the movement of the robot. The logic behind the practical implementation of the line control is that the robot will always be correcting its trajectory so if a line sensor reads high the other will necessarily be off. Therefore, the motors should propel the robot in the opposite direction of the line sensor in use.

Now that all classes have been created, each of these different classes can now be used to interact with each other to make a functioning robot to solve real world problems.

Lab 5: Robot Obstacle Course

Objective

The purpose of this lab is to implement all classes into a functional robot to complete four challenges: a line follower, a maze, a gripper with directional steering controlled by Bluetooth, and PID control to maintain a constant speed when the robot goes up a treadmill.

Required Materials

Completed robot, laptop

Part 1: Line Follower

The line follower is designed to have the robot follow a black line on a white background. Whenever the robot gets off track it automatically corrects its trajectory to stay on course. It accomplishes this by receiving feedback from the infrared line sensors and using that information to make adjustments to the DC motors accordingly. The line sensors were placed about a finger's width apart - just far enough so that the robot was always moving toward the line but not so far apart as to require major adjustments back to the course.

The .ino file used to manipulate the robot back on course

Linefollower.ino

```
#include <Adafruit_MotorShield.h>
#include <lineControl.h>
#include <linesensor.h>
#include <motor.h>
#include <Range_Echo.h>
#include <Wire.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();

const float UNIT_CONVERSION = 0.003937;

volatile float frontSensorDistance = 0.0;
bool rightLineSensor = 1;
bool leftLineSensor = 1;

linesensor *leftSensor;
linesensor *rightSensor;

motor *RFMotor;
motor *LFMotor;
motor *LRMotor;
```

```

motor *RRMotor;

Range_Echo *frontRange;

void setup() {
    leftSensor = new linesensor(13);
    rightSensor = new linesensor(12);
    frontRange = new Range_Echo(10, 11);
    RFMotor = new motor(&AFMS, 1, 0); //right front
    LFMotor = new motor(&AFMS, 2, 1); //left front
    LRMotor = new motor(&AFMS, 3, 1); //left rear
    RRMotor = new motor(&AFMS, 4, 0); //right rear

    frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;

    rightLineSensor = rightSensor;
    leftLineSensor = leftSensor;
}

void loop() {
    Serial.begin(115200);
    AFMS.begin();

    frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;

    while ((frontSensorDistance > 1.0) && (frontSensorDistance < 7.5)) {
        LFMotor->stop();
        LRMotor->stop();
        RFMotor->stop();
        RRMotor->stop();
        frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;
    }

    if (frontSensorDistance > 8.0) {

        if (digitalRead(13) == HIGH) {
            LFMotor->setDrive(0.0, FORWARD);
            LRMotor->setDrive(0.0, FORWARD);
            RFMotor->setDrive(230.0, FORWARD);
            RRMotor->setDrive(230.0, FORWARD);
        }

        else {
            LFMotor->setDrive(230.0, FORWARD);
            LRMotor->setDrive(230.0, FORWARD);

```

```
RFMotor->setDrive(0.0, FORWARD);
RRMotor->setDrive(0.0, FORWARD);
}

if (digitalRead(12) == HIGH) {
  LFMotor->setDrive(230.0, FORWARD);
  LRMotor->setDrive(230.0, FORWARD);
  RFMotor->setDrive(0.0, FORWARD);
  RRMotor->setDrive(0.0, FORWARD);
}
else {
  LFMotor->setDrive(0.0, FORWARD);
  LRMotor->setDrive(0.0, FORWARD);
  RFMotor->setDrive(230.0, FORWARD);
  RRMotor->setDrive(230.0, FORWARD);
}
}
}
```

Image: the robot following the line



Part 2: Gripper and Bluetooth Navigation

The robot implemented Bluetooth functionality to manually control the robot. One button was used to open and close the gripper, while eight other buttons were used to move the robot in the four primary directions, and four diagonal directions.

When a button is pressed on the phone, it sends out a predetermined signal. When the button is pressed, it will send a number 1-9 followed by a ? In binary. When the button is released it will send another number 1-9 with a \$ in binary. The Bluetooth module on the microcontroller is able to interface with the phone to receive input. To translate the input signals to usable functionality, a switch statement is used within the loop to be on the lookout for a button press. When it detects the appropriate button is pressed it will execute the corresponding programmed action. Within the switch statement, it converts the binary signal from the input serial port to a hexadecimal value. The value is then used to loop through the switch statement and take the appropriate action. If the program does not encounter the corresponding case on the first loop it increases a counter by one and tries again. For example, if the received a 2? signal it would first check for a 1? by default. Since it would not encounter this the counter will increase by one and it will try again. Now that the corresponding case has matched with the input signal, it will execute the preprogrammed action.

bluetooth.ino

```
//*****LIBRARIES THAT ARE TO BE INCLUDED*****  
#include <Adafruit_MotorShield.h>  
#include <Gripper.h>  
#include <MegaServo.h>  
#include <motor.h>  
#include <SoftwareSerial.h>  
  
//*****CREATE THE MOTOR SHIELD OBJECT WITH THE DEFAULT I2C  
ADDRESS*****  
Adafruit_MotorShield AFMS = Adafruit_MotorShield();  
  
//*****GLOBAL CONSTANTS ARE DECLARED HERE*****  
const int DELAY_TIME = 1000;  
const float UNIT_CONVERSION = 0.003937; // Converts whatever unit pops out for the range  
// finder to inches  
  
const byte rxPin = 18;  
const byte txPin = 19;  
  
//*****GLOBAL VARIABLES ARE DECLARED HERE*****  
char message[256];  
int index = 0;  
  
//*****POINTERS ARE DECLARED HERE*****  
Gripper *openGrip;
```

```

MegaServo myGripper;

motor *RFMotor = new motor(&AFMS, 1, 0); //right front
motor *LFMotor = new motor(&AFMS, 2, 1); //left front
motor *LRMotor = new motor(&AFMS, 3, 1); //left rear
motor *RRMotor = new motor(&AFMS, 4, 0); //right rear

SoftwareSerial BTSerial(rxPin, txPin);

//=====
=
//=====SETUP STARTS HERE=====
//=====
void setup() {

    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);

    Serial.begin(38400); // Usually 9600 for BT mode, although it is sometimes 38400:
    Serial1.begin(115200);
    AFMS.begin();
    Serial.println("test");

    openGrip = new Gripper(36);
    Gripper(22);
    delay(1000); // delay for 1 second before moving on
    openGrip -> close(); // call the close() function from the .cpp file to close the gripper.
}

//=====
=
//=====LOOP STARTS HERE=====
//=====
void loop() {
    if (Serial1.available()) {
        message[index] = (char)Serial1.read();
        if (message[index] == 0x3F) //Checks for a ? character
        {
            switch (message[0]) { // was previously message[0]
                case '1':
                    Serial.print("Button 1 Pressed");
                    LFMotor->setDrive(60.0, FORWARD);
                    LRMotor->setDrive(60.0, FORWARD);

```

```

RFMotor->setDrive(150.0, FORWARD);
RRMotor->setDrive(150.0, FORWARD);
break;
case '2':
    Serial.print("Button 2 Pressed");
    LFMotor->setDrive(125.0, FORWARD);
    LRMotor->setDrive(125.0, FORWARD);
    RFMotor->setDrive(132.0, FORWARD);
    RRMotor->setDrive(132.0, FORWARD);
    break;
case '3':
    Serial.print("Button 3 Pressed");
    LFMotor->setDrive(130.0, FORWARD);
    LRMotor->setDrive(130.0, FORWARD);
    RFMotor->setDrive(65.0, FORWARD);
    RRMotor->setDrive(65.0, FORWARD);
    break;
case '4':
    Serial.print("Button 4 Pressed");
    LFMotor->setDrive(60.0, BACKWARD);
    LRMotor->setDrive(60.0, BACKWARD);
    RFMotor->setDrive(60.0, FORWARD);
    RRMotor->setDrive(60.0, FORWARD);
    break;
case '5':
    Serial.print("Button 5 Pressed");
    if (openGrip->isOpen()) {
        openGrip->close();
    }
    else {
        openGrip->open();
    }
    break;
case '6':
    Serial.print("Button 6 Pressed");
    LFMotor->setDrive(60.0, FORWARD);
    LRMotor->setDrive(60.0, FORWARD);
    RFMotor->setDrive(60.0, BACKWARD);
    RRMotor->setDrive(60.0, BACKWARD);
    break;
case '7':
    Serial.print("Button 7 Pressed");
    LFMotor->setDrive(60.0, BACKWARD);
    LRMotor->setDrive(60.0, BACKWARD);

```



```

RFMotor->setDrive(120.0, BACKWARD);
RRMotor->setDrive(120.0, BACKWARD);
break;
case '8':
    Serial.print("Button 8 Pressed");
    LFMotor->setDrive(130.0, BACKWARD);
    LRMotor->setDrive(130.0, BACKWARD);
    RFMotor->setDrive(120.0, BACKWARD);
    RRMotor->setDrive(120.0, BACKWARD);
    break;
case '9':
    Serial.print("Button 9 Pressed");
    LFMotor->setDrive(150.0, BACKWARD);
    LRMotor->setDrive(150.0, BACKWARD);
    RFMotor->setDrive(60.0, BACKWARD);
    RRMotor->setDrive(60.0, BACKWARD);
    break;
default :
    Serial.print("BREAK");
    break;
}
index = 0;
Serial.println();
}
else if (message[index] == 0x24) { //Checks for a $ character
    switch (message[0]) {
        case '1':
            Serial.print("Button 1 Released");
            LFMotor->stop();
            LRMotor->stop();
            RFMotor->stop();
            RRMotor->stop();
            break;
        case '2':
            Serial.print("Button 2 Released");
            LFMotor->stop();
            LRMotor->stop();
            RFMotor->stop();
            RRMotor->stop();
            break;
        case '3':
            Serial.print("Button 3 Released");
            LFMotor->stop();
            LRMotor->stop();
    }
}

```

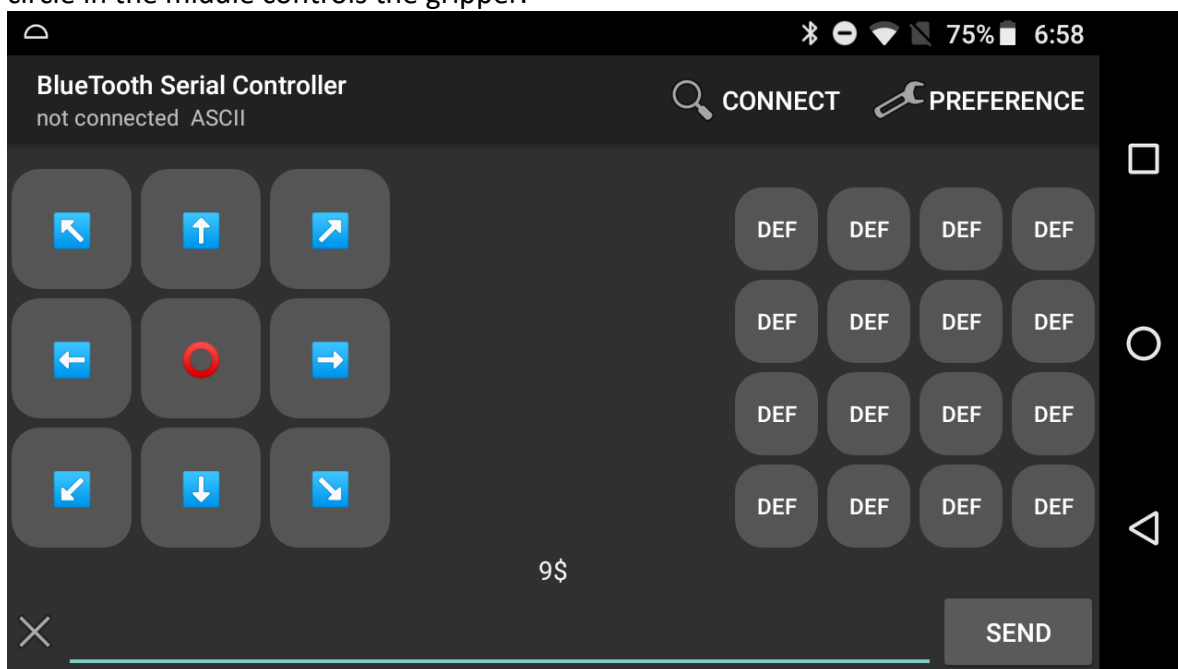
```
RFMotor->stop();
RRMotor->stop();
break;
case '4':
    Serial.print("Button 4 Released");
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    break;
case '5':
    Serial.print("Button 5 Released");
    break;
case '6':
    Serial.print("Button 6 Released");
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    break;
case '7':
    Serial.print("Button 7 Released");
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    break;
case '8':
    Serial.print("Button 8 Released");
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    break;
case '9':
    Serial.print("Button 9 Released");
    LFMotor->stop();
    LRMotor->stop();
    RFMotor->stop();
    RRMotor->stop();
    break;
default:
    Serial.print("BREAK");
    break;
```

```

    }
    index = 0;
    Serial.println();
  }
  else {
    index++; //Increases the index counter by one if neither a ? or a $ was encountered.
  }
}
}
}

```

Image: screenshot of the Bluetooth controller used on the phone to control the robot. Eight directions can be seen, which are programmed to transmit a character when pressed. The red circle in the middle controls the gripper.



Part 3: Maze Navigation

The robot is equipped with two ultrasonic range detectors. They are used to calculate the distance from the robot an object is by measuring the time it takes for sound to be reflected back to the robot. One range detector is located in front of the robot and another is off to the left side. Depending on the input from the two range detectors, the program decides whether to turn left, right, or go straight. If the range detectors sense that something is close by both in front and to the left of the robot, it turns right. If it detects that something is near the front but not near the left then it will turn left. If it detects that something is near the left but not near the front it will continue straight. In order to make a turn, tank turns are used, where the motors on the opposing side of the turn moves in the forward direction, and the motors on the side of the robot closer to the turn moves in the reverse direction.

mazenavigator.ino

```
//*****LIBRARIES THAT ARE TO BE INCLUDED*****  
#include <Adafruit_MotorShield.h>  
#include <motor.h>  
#include <Range_Echo.h>  
#include <Wire.h>  
  
//*****CREATE THE MOTOR SHIELD OBJECT WITH THE DEFAULT I2C  
ADDRESS*****  
Adafruit_MotorShield AFMS = Adafruit_MotorShield();  
  
//*****GLOBAL CONSTANTS ARE DECLARED HERE*****  
const float UNIT_CONVERSION = 0.003937;  
  
//*****GLOBAL VARIABLES ARE DECLARED HERE*****  
volatile float frontSensorDistance = 0.0;  
volatile float leftSensorDistance = 0.0;  
  
//*****POINTERS ARE DECLARED HERE*****  
motor *RFMotor;  
motor *LFMotor;  
motor *LRMotor;  
motor *RRMotor;  
  
Range_Echo *frontRange;  
Range_Echo *leftRange;  
  
//=====  
=  
//=====SETUP STARTS HERE=====  
//=====  
=  
void setup() {  
  AFMS.begin();  
  
  frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;  
  leftSensorDistance = (leftRange -> Timing()) * UNIT_CONVERSION;  
  
  RFMotor = new motor(&AFMS, 1, 0); //right front  
  LFMotor = new motor(&AFMS, 2, 1); //left front  
  LRMotor = new motor(&AFMS, 3, 1); //left rear  
  RRMotor = new motor(&AFMS, 4, 0); //right rear  
  
  frontRange = new Range_Echo(10, 11);  
  leftRange = new Range_Echo(8, 9);
```

```

}

//=====
=
//=====LOOP STARTS HERE=====
//=====
=
void loop() {
  frontSensorDistance = (frontRange -> Timing()) * UNIT_CONVERSION;
  leftSensorDistance = (leftRange -> Timing()) * UNIT_CONVERSION;

  if ((frontSensorDistance > 8.0) && (leftSensorDistance > 17.0)) {
    LFMotor->setDrive(90.0, FORWARD);
    LRMotor->setDrive(90.0, FORWARD);
    RFMotor->setDrive(94.0, FORWARD);
    RRMotor->setDrive(94.0, FORWARD);
  }
  else if ((frontSensorDistance <= 7.0) && (frontSensorDistance > 1.0) && (leftSensorDistance <= 10.0))
  { //RIGHT turn
    for (int x = 0; x < 48; x++) {
      LFMotor->setDrive(200.0, FORWARD);
      LRMotor->setDrive(200.0, FORWARD);
      RFMotor->setDrive(198.0, BACKWARD);
      RRMotor->setDrive(198.0, BACKWARD);
    }

  }
  else if ((frontSensorDistance <= 7.0) && (frontSensorDistance > 1.0) && (leftSensorDistance > 10.0)) {
    // LEFT turn
    for (int x = 0; x < 48; x++) {
      LFMotor->setDrive(200.0, BACKWARD);
      LRMotor->setDrive(200.0, BACKWARD);
      RFMotor->setDrive(200.0, FORWARD);
      RRMotor->setDrive(200.0, FORWARD);
    }
  }
  else {
    LFMotor->setDrive(90.0, FORWARD);
    LRMotor->setDrive(90.0, FORWARD);
    RFMotor->setDrive(94.0, FORWARD);
    RRMotor->setDrive(94.0, FORWARD);
  }
}

```

Image: The robot autonomously navigating through the maze.



Part 4: PID Control

The robot receives feedback from wheel encoders to calculate the speed at which the robot is traveling. Robot encoders work with a light emitter/detector with a disc that has slits cut out of it. As the wheel rotates light is allowed through the disc at regular intervals. The high impulse is sent to the microcontroller which then uses the information to manipulate internal registers that are then used to calculate the speed. As the speed of the robot adjusts, such as when it is going up a hill, the microcontroller detects a slower rotation from the encoders and adjusts the pulse width modulation to the motors accordingly to adjust the speed back to a predetermined rate.

The solution for this lab required that the robot maintain a constant speed as the incline was raised and lowered. This was almost achieved; the robot was able to maintain an almost constant speed, slowing down just slightly when the incline was raised. The speed of the treadmill needed to be adjusted from 1.3 while flat to 1.0 while inclined at a 15° angle. Otherwise the robot was able to maintain a constant speed as the angle of the incline was adjusted slowly.

PIDcontrol.ino

```
//*****LIBRARIES THAT ARE TO BE INCLUDED*****
#include <encoder.h>
#include <motor.h>
#include <speedcontrol.h>
```

```

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <HardwareSerial.h>

//*****CREATE THE MOTOR SHIELD OBJECT WITH THE DEFAULT I2C
ADDRESS*****
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

//*****GLOBAL VARIABLES ARE DECLARED HERE*****
volatile int toggle_motor_speed = 1;
volatile double speedSetting1 = 70.0;
volatile double speedSetting2 = 50.0;
volatile double speedSetting3 = 70.0;
volatile double speedSetting4 = 50.0;
volatile double samplingRate = 0.02;
volatile int update_speed = 0;
volatile double setspeed = 0;
volatile double setspeed1 = 0;

//*****POINTERS ARE DECLARED HERE*****
encoder *rightEncoder = new encoder(20, 66, 48);
encoder *leftEncoder = new encoder(20, 66, 49);

motor *RFMotor = new motor(&AFMS, 1, 0); //right front
motor *LFMotor = new motor(&AFMS, 2, 1); //left front
motor *LRMotor = new motor(&AFMS, 3, 1); //left rear
motor *RRMotor = new motor(&AFMS, 4, 0); //right rear

speedcontrol *rightSpeed = new speedcontrol(&Serial, rightEncoder, RFMotor,
      RRMotor, samplingRate);
speedcontrol *leftSpeed = new speedcontrol(&Serial, leftEncoder, LFMotor, LRMotor, samplingRate);

//=====
=
//=====SETUP STARTS HERE=====
//=====
=
void setup() {
  Serial.begin(115200);
  noInterrupts(); // Disable the ISR's
  TCCR3A = 0;
  TCCR3B = 0x0A; // Prescale and CIC mode
  OCR3A = 0x9C40; // set the compare A Register
  TIMSK3 = 0x02; // enable the output compare A Match Interrupt
  TCNT3 = 0x00;
  // Enable input capture 4 (ICNC4 enable noise canceler, ICES4 trigger on rising edge, CS42/0 no

```

```

// prescaler)
TCCR4A = 0x00; // Initilize register to 0
TCCR4B = 0x00; // Initilize register to 0
TCCR4B = (1<<ICNC4) | (1<<ICES4) | (0<<WGM42) | (1<<CS42) | (0<<CS41) | (1<<CS40);
TCCR4C = 0x00; // Initilize register to 0
TIMSK4 = (1<<ICIE4); // Input capture interrupt enabled
TCNT4 = 0x00; // Sets the Timer to 0
// Enable input capture 5 (ICNC5 enable noise canceler, ICES5 trigger on rising edge, CS52/0 no
// prescaler)
TCCR5A = 0x00; // Initilize register to 0
TCCR5B = 0x00; // Initilize register to 0
TCCR5B = (1<<ICNC5) | (1<<ICES5) | (0<<WGM52) | (1<<CS52) | (0<<CS51) | (1<<CS50);
TCCR5C = 0x00; // Initilize register to 0
TIMSK5 = (1<<ICIE5); // Input capture interrupt enabled
TCNT5 = 0x00; // Sets the Timer to 0
interrupts(); // enable interrupts.
rightSpeed -> enable();
leftSpeed -> enable();
AFMS.begin();
}

//=====
=
//=====LOOP STARTS HERE=====
//=====
=
void loop() {
  setspeed = rightEncoder -> getSpeed();
  setspeed1 = leftEncoder -> getSpeed();
  // Serial.println(setspeed);
  if(setspeed < 5 && setspeed1 < 5)
    setspeed = 0;
  if(update_speed == 1){
    rightSpeed -> update(setspeed);
    leftSpeed -> update(setspeed1);
    update_speed = 0;
  }
  if(toggle_motor_speed == 1){
    rightSpeed -> setDesiredSpeed(speedSetting1);
    leftSpeed -> setDesiredSpeed(speedSetting1);
  }
  else {
    rightSpeed -> setDesiredSpeed(speedSetting2);
    leftSpeed -> setDesiredSpeed(speedSetting2);
  }
}
}

```

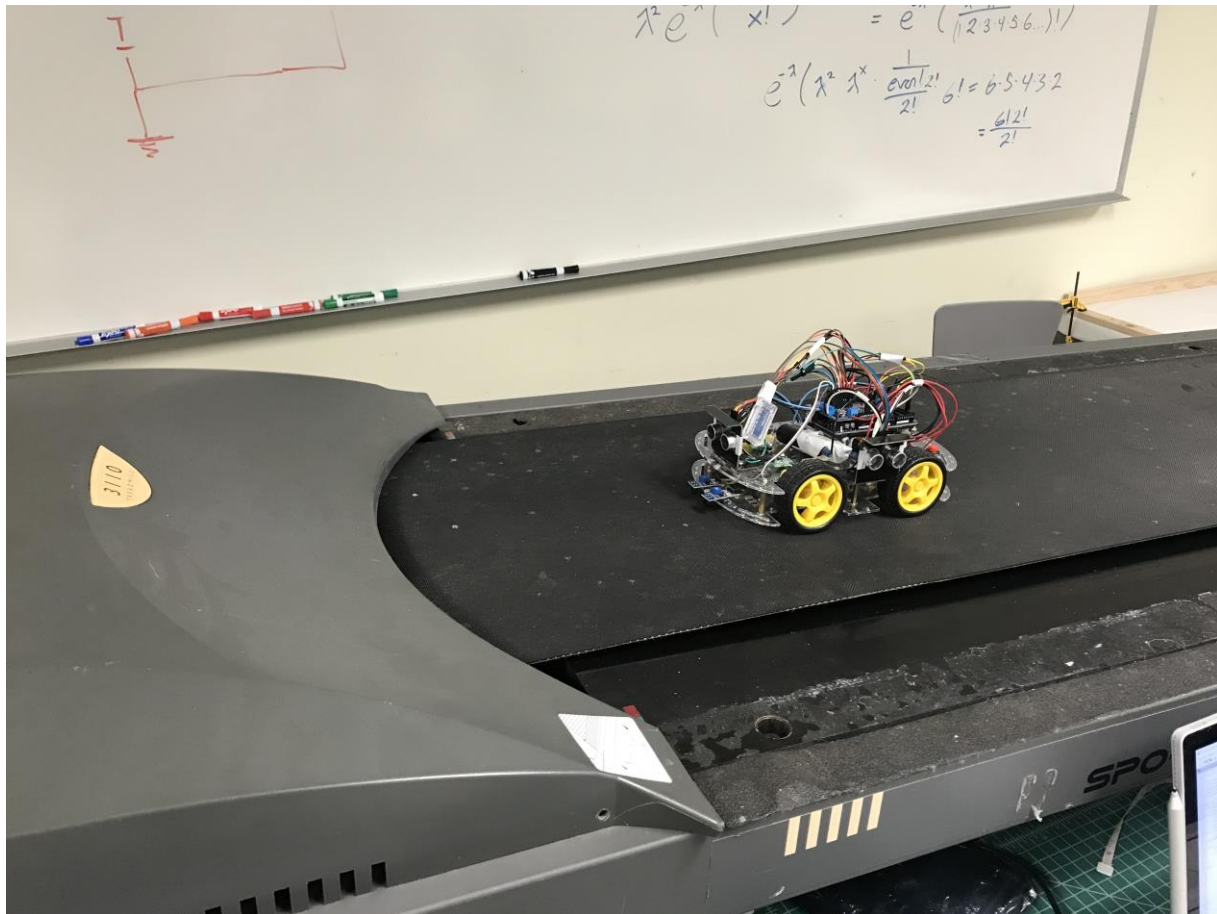


```

//*****INTERRUPT SERVICE ROUTINES*****
ISR(TIMER3_COMPA_vect){
update_speed = 1;
}
ISR(TIMER5_OVF_vect){
if(toggle_motor_speed == 1)
toggle_motor_speed == 0;
else
toggle_motor_speed;
}
ISR(TIMER4_OVF_vect){
if(toggle_motor_speed == 1)
toggle_motor_speed = 0;
else
toggle_motor_speed = 1;
}
ISR(TIMER5_CAPT_vect){
rightEncoder -> updateTime(ICR5);
TCNT5 = 0x00;
}
ISR(TIMER4_CAPT_vect){
leftEncoder -> updateTime(ICR4);
TCNT4 = 0x00;
}

```

Image: Robot on a treadmill maintaining a constant speed using the PID control.



UART / TIMER / INTERRUPT

Resources

Function	Resource	Notes
Serial Monitor	Hardware Serial (Serial)	0(Rx) and 1(Tx) connected to USB chip
Bluetooth Module	Hardware Serial (Serial)	19(Rx) and 18(Tx) connect BT Module
PID Tuning Serial Output	Hardware Serial (Serial)	17(Rx) and 16(Tx) connected to FTDI Module
Gripper	Timer1	Overlap
Periodic Timer	Timer3 - CTC Mode	Compare A Match Interrupt (CTC Mode)
Left Wheel Encoder	Timer4 - ICP4 (Pin, 48)	input Capture Mode (Capture IRO, Overflow (RO))
Right Wheel Encoder	Timer5 - ICP5 (Pin, 49)	input Capture Mode (Capture IRO, Overflow (RO))
LF Motor	Motor Shield (I2C)	Address = 0x60
RF Motor	Motor Shield (I2C)	
LR Motor	Motor Shield (I2C)	

RR Motor	Motor Shield (I2C)	
Ultrasonic Rangefinder (Front)	Digital IO (Trig=10, ECHO=11)	
Ultrasonic Rangefinder (Left)	Digital IO (Trig=8, ECHO=9)	
Left Line Sensor	Digital IO (13)	
Right Line Sensor	Digital IO (12)	

Bill of Materials

Description	Source	Quantity		Cost (USD)
Arduino Kit (IEEE)	IEEE Group (OIT)	1	\$115.00	\$115.00
Arduino MEGA 2560 R3	included	1		
Adafruit Motor Shield V2.3 Kit	included	1		
Horizontal Gripper Kit (standard)	included	1		
Hitec 31311S HS-311 Servo	included	1		
Wheel Encoder Kit (2)	included	1		
Line Sensor Module 5V Infrared	included	2		
Markerfire 4-Wheel Robot Chassis Kit with Motors	included	1		
HC-05 Bluetooth Modem	included	1		
HC-SR04 Ultrasonic Distance Sensor Module	included	2		
Misc. Hardware (Device Mounting)	Scrap/Parts			
Red Pushbutton (Plastic) SPST	Scrap/Parts			
AA Battery (1.5v)	Spare	4	\$0.50	\$2.00
AA Battery Holder (4-Slot)	Frys Electronics	1	\$2.95	\$2.95
18650 Battery (3.7v) 24..mAh	Amazon	2	\$3.17	\$6.34
			Total:	\$126.29