ILR 01 – Sensors and Motor Lab

Dorothy Kirlew

Team I Members: Pranav Maheshwari, Richa Varma, Mohak Bhardwaj, and Shivam Gautam

October 10, 2015

1. Individual Progress

a. Sonar Range Finder

I created the Arduino code for receiving the pulse width information from the MaxBotix Sonar Range Finder, pictured in Figure 1:. The Range Finder is also referred to as the Ultra Sonic Range Finder throughout the ILR.

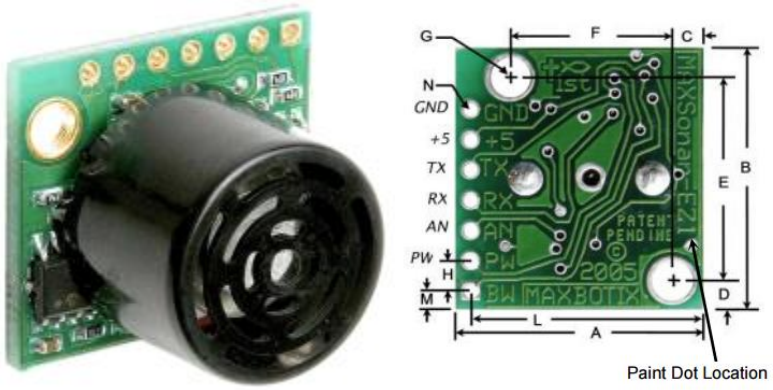

Figure 1: MB1010 LV-MaxSonar® -EZ™ Series High Performance Sonar Range Finder.

The bottom three pins (BW, PW, and AN) can be used to receive data from the Range Finder. I chose to use the pulse width (PW), as it is very reliable for distances between 5 and 24 inches. The PW output has a minimum accurate output of 820 units, so I first subtracted 818 from the output to scale the reading down. I then multiplied by 0.62 to map the scaled output to an input range for a motor (0-1023 units). This is represented in the formula below. The 0-1023 units range is the standard range used in this system to communicate all inputs and outputs within the Arduino, GUI, and between the Arduino and the GUI.

$$sensor\_value = (reading - 818) * 0.62$$

In order to filter out any noise and inaccurate values, I ignored any readings that were negative, differed from the previous reading by more than 50 units, and changed any values that were above 1023 to just 1023, as this is the maximum value possible. Additionally, I read and averaged 10 sequential readings to create the final output value for the sensor. This ensured that any slight discrepancies between readings were softened. The code for reading and averaging output from the Ultra Sonic sensor can be found in Figure 2**: Ultra Sonic Sensor Code** below.

```
//Dorothy
void ultrasonic_sensor()
{
  // Get base reading.  Subtract 818 to get values in correct range.
  // Multiply by 0.017 to change to inches.
  // Multiply by 36.54 to convert from inches to 0-1023 range
  sensor_val = 0;

  int lastRead = (pulseIn(ultra, HIGH) - 818) * 0.62;
  int temp;
  // sum 10 good readings that will be averaged to filter out noise
  for (int i = 0; i < 10; i++)
  {
    temp = (pulseIn(ultra, HIGH) - 818) * 0.62;
    if (temp > 1100)
    {
      sensor_val += 1023;
    }
    // if out of range or too different from last reading, get a new reading
    else if (temp < 0 || abs(temp - lastRead) > 50)
    {
      i--;
    }
    // if reading is acceptable, sum it
    else
    {
      sensor_val += temp;
    }
    lastRead = temp;
  }
  // if average is too high, just return 1023.  otherwise return average.
  if ((sensor_val / 10) > 1023)
  {
    sensor_val = 1023;
  }
  else
  {
    sensor_val = sensor_val / 10;
  }
}
```

Figure 2: Ultra Sonic Sensor Code

a. GUI

In addition to writing the code for the Ultra Sonic Sensor, I created the GUI seen in Figure 3**: GUI upon Start**. We used Qt to do so, which was new for both of us. Qt uses C++, which I am familiar with. However, as Mohak is not as comfortable with it, I supported Mohak in creating the GUI, as he would like to improve his programming skills. The complete code can be found in Section 5b: GUI Code.

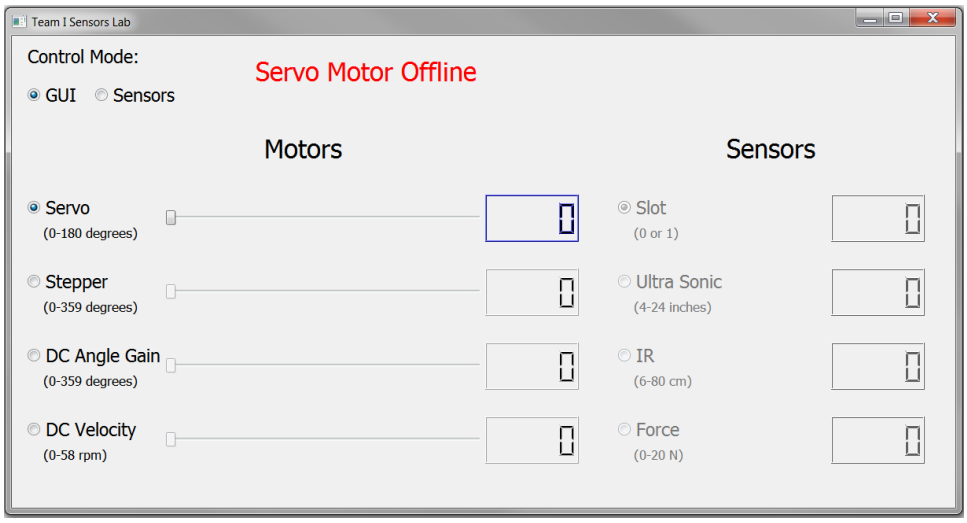

Figure 3: GUI upon Start

i. Control Mode: GUI

The first step to use the GUI is to select whether the motors will be controlled by the sliders on the GUI or the sensors. This can be done by selecting the corresponding radio button. If the user selects "GUI", which is the default mode, they can use the sliders to control the active motor. The active motor is indicated in two ways: the radio button is selected and the display number to the right of the active motor's slider is blue. The slider has a range of 0 to 1023, which is converted to the appropriate range for the active motor and displayed to the right of the corresponding slider. The user can change what motor they are using by selecting the appropriate radio button.

When the "GUI" option is selected, the user cannot select a sensor and any sensor information sent from the Arduino is ignored. See Figure 4**: Controlling DC Motor via GUI** for an example of the user moving the DC Motor by 180 degrees using the GUI.

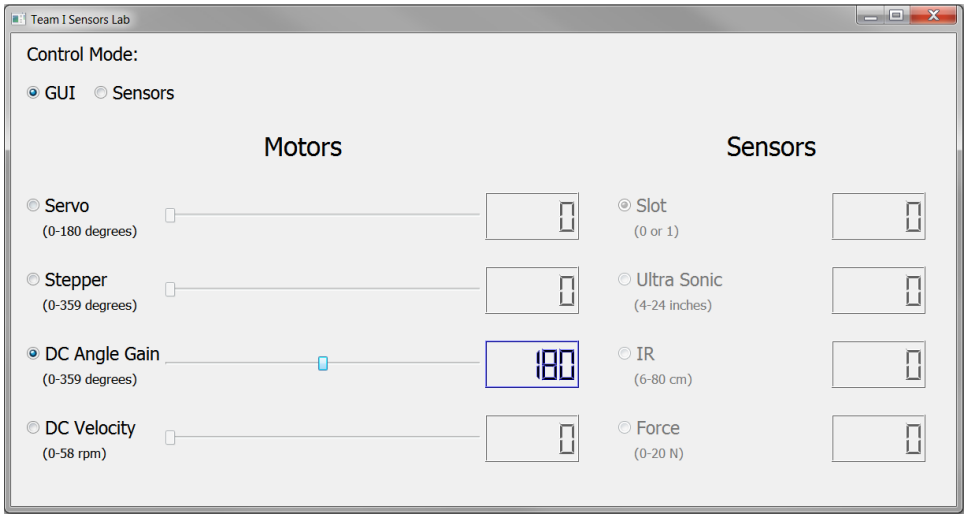

Figure 4: Controlling DC Motor via GUI

ii. Control Mode: Sensors

When the user selects “Sensors”, the radio buttons for the sensors are enabled and the sliders are disabled. The user can choose sensor to activate. The active sensor is indicated in two ways: the radio button is selected and the number to the right of the active sensor is blue. The number displayed for the active sensor is specific to that sensor. For example, the Slot sensor will only show “0” or “1”, while the Ultra Sonic sensor will show values between 4 and 24 inches. While using the sensor, the user can change which motor they want to be controlled by the sensor. The active motor will show the number in the range specific to the motor, not the value of the sensor. See Figure 5**: Stepper Motor controlled via IR Sensor** for an example of the IR sensor controlling the Stepper motor.

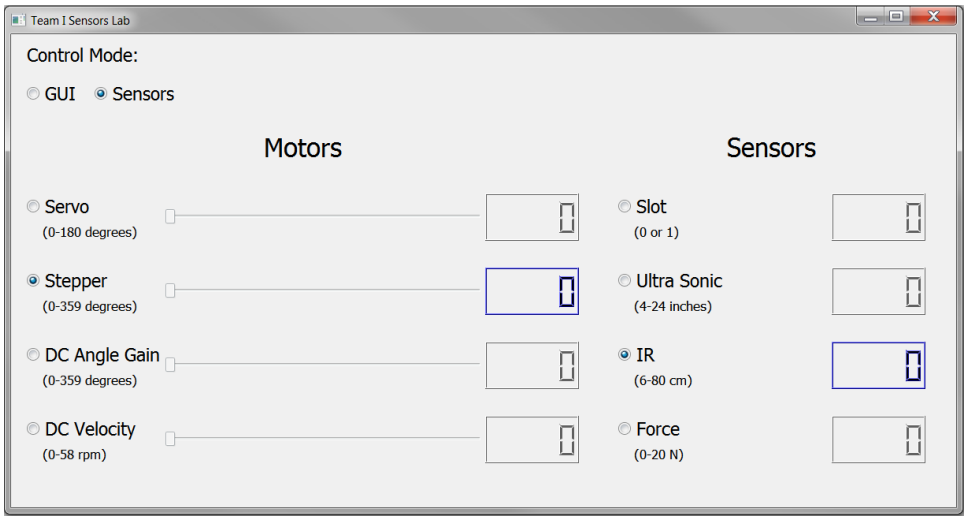

Figure 5: Stepper Motor controlled via IR Sensor

iii. Messages from Arduino

Behind the scenes, the GUI receives two types of messages from the Arduino; servo on/off and sensor data. All data sent to then GUI is separated by a comma for ease of parsing. The user can turn the servo motor on and off by flipping a switch connected to the Arduino. The Arduino simply sends “X” if the motor is turned off and “O” if it is turned on. A message is displayed on the GUI if the servo motor is off (as seen in Figure 3**: GUI upon Start**).

The second type of message received is sensor data and has two parts. The first part is a single letter that indicates which sensor is active (“S” for Slot, “U” for Ultra Sonic, “I” for IR, and “T” for Tension/Force). The second part of the message is a number between 0 and 1023 that indicates the value read from the sensor and sent to the motor. The GUI updates the active motor to display the value converted to the appropriate motor range (for example, between 0 and 58 rpm for the DC motor’s velocity). It also converts this number to the appropriate value for the sensor and displays it (for example, between 0 and 20 N for the Force sensor). If the Arduino sends a message for an inactive sensor (due to possible Arduino error or lag in communication), the data is ignored. If the Arduino sends data in an unrecognizable format, the data is ignored and an error is sent to the Qt output.

iv. Messages to Arduino

Two types of messages are sent from the GUI to the Arduino; motor control and sensor/motor change. When the user selects a motor to control or changes the value for a motor, a message is sent to the Arduino in three parts. The first part is "C" to indicate a control message. The second part is a number from 0-3, which corresponds to the active motor (0 for Servo, 1 for Stepper, 2 for DC Angular Position, and 3 for DC Velocity). The final part of the message is a value between 0 and 1023 that indicates the value to be sent to the motor.

The second type of message sent to the Arduino indicates the active sensor and active motor. There are three parts to the message. The first is an "S" to indicate a sensor message. The second part is a number from 0-3, which corresponds to the active motor, and the final part is a number 0-3, which corresponds to the active sensor (0 for Slot, 1 for Ultra Sonic, 2 for IR, and 3 for Force).

2. Challenges

This is my second time working with an Arduino. Although I'm a strong coder, I'm still getting used to the Arduino environment and capabilities. Consequently, it took me some time to become comfortable interfacing with the Range Finder. Initially, I tried using the analog output of the sensor. Not only were the analog outputs erratic, it required me to average several readings, insert a delay between readings, and ignore readings that were out of range (those that were negative or significantly different from the previous reading). After struggling to get suitable Analog output, I switched to PW output, which was much easier to handle and more reliable.

Another challenge was interfacing the Qt GUI with the Arduino. Never having done this before, Mohak and I took a lot of time learning how to send data from the GUI to the serial port, as well as how to send properly formatted data from the Arduino to the GUI, and then parsing that information. This a while to code, and even more time to test and debug.

Additionally, working with a GUI presents its own challenges. Formatting needs to be consistent to present an easy-to-use interface, and Qt can be incredibly stubborn when it comes to changing the sizes of the objects in the layout.

To assist with tuning the DC motor, Mohak and I spent several hours trying to create graphs with Qt. Unfortunately, we were unable to do so and determined that our time could be better spent enhancing the GUI. The DC motor had to be tuned via serial output, which is a timely and inaccurate process.

3. Teamwork

We divided the work amongst our five team members as evenly as possible, with respect to our skills. I created the code for the Sonar Range Finder and worked with Mohak to create the GUI. Shivam created the Arduino code for the DC motors and the force sensor. Richa created the Arduino code for the slot sensor and the stepper motor. She also worked on hardware integration. Pranav created the code for

the servo motor and IR sensor, as well as created the structure for the entirety of the Arduino code to be integrated together.

4. Plans

The team has four basic goals to complete by Progress Review 1.

a. Communication/Network

Come up with 1-3 possible methods of creating a scalable and reliable network for the vehicles. The solution must allow for vehicle collaboration and dynamic reconfiguration as vehicles enter and exit the parking lot. Shivam will be conducting a literature survey to find these solutions and gather data regarding their implementation and feasibility.

b. Obstacle Detection

Create a system capable of detecting obstacles in close range, i.e. 0-150cms. We will conduct a literature survey of existing vision algorithms and come up with 1-3 possible cameras and 1-3 possible sensors that are capable of achieving this goal. The cameras and sensors must easily integrate into the system. Additionally, we will define the obstacles allowed in the environment. Pranav and Mohak will be actively working on this.

c. Mobile Platform

Successfully integrate the Arduino Mega 2560 with the DFRobot 4WD Platform. There is a certain amount of risk associated with completing this goal on time, as the platform won't arrive until Monday, October 19. Upon arrival, Richa will integrate the two devices, test the actuators, and determine their accuracy via encoder readings.

d. App Development

Create the basic user interface design and use-case for the mobile app used by the user to communicate with their vehicle. Mohak and I will select the language and tools needed to create an Android app, as well as research what is needed to create a Bluetooth connection between the Android device and the mobile platform.

5. Code

All code used for our project can be found in the following two sections: Arduino and GUI.

a. Arduino

```
#include<SharpIR.h>
#include<PID_v1.h>
#include<Servo.h>
#include<Encoder.h>
#define m1 5 //motor pin 1 (Grey Wire from Motor Board)
#define m2 6 //motor pin 2 (Yellow Wire)

//Declare you sensor and actuator related constants
Servo myservo;
int ir = A0;
int slot = A1;
int flex = A2;
int ultra = 7;
SharpIR sharp(ir, 30, 95, 1080);
int stepper_signal = 13;
int stepper_direction = 12;
int serv;
Encoder myEnc(2, 3);
void velocityControl( int m_speed, int m_direction);

int flag = 0, prevState=100;

//PID stuff
double Kpv = 2, Kiv = 0.5, Kdv = 0.03;
double Kp = 3.4, Ki = 0.22, Kd = 0.14;
double Setpoint=0, Input=0, Output=0;
double Setpoint_v = 0, Input_v = 0, Output_v = 0;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
PID myPID_v(&Input_v, &Output_v, &Setpoint_v, Kpv, Kiv, Kdv, DIRECT);


//Declare the pins here. 2,3,5,6,9,12 and 13 are occupied by DC, Servo and Stepper
//int control = 11;
int button = A4;

//Declare your sensor related constants here. Use sensor_val for your sensor.
int sensor_val = 0;
int stepper_last = 0;
int sensor_control = -1;
int actuator_control = -1;
long debounceDelay = 50;
long lastDebounceTime = 0;
boolean state = LOW;
boolean last = LOW;
int oldactuator;
int val;

int prev_sensor_val = 0;
long previousMillis = 0;
long previousMillis2 = 0;
long interval = 100;
unsigned long currentMillis;
int desiredAngle = 0, currentPosition = 0, desiredPosition = 0, error = 0;
float setpoint = 0;
int m_speed = 0, m_direction = 1;
double desiredSpeed = 0, currentSpeed = 0;
int ultra_fix = 0;
int current = LOW;

//Declare your serial communication related constants here.
String incoming = "";
char test;
String temp;

//Setting up Serial and pin modes.
void setup()
{
  Serial.begin(9600);
```

```
  pinMode(m1, OUTPUT);
  pinMode(m2, OUTPUT);
  pinMode(slot, INPUT);
  pinMode(flex, INPUT);
  pinMode(stepper_signal, OUTPUT);
  pinMode(stepper_direction, OUTPUT);
  pinMode(ir, INPUT);
  pinMode(button, INPUT);
  //pinMode(control, OUTPUT);
  pinMode(ultra, INPUT);
  myservo.attach(9);
  myPID.SetMode(AUTOMATIC);
  currentPosition = myEnc.read();
  myPID.SetOutputLimits(-255, 255);
}

//Serial Communication: Two formats of inputs can be processed. C-A-XXXX where C is control of Actuator A {0,1,2} by Value XXXX {0-1023}.
// The other format is S-A-T where S is sense transducer T {0,1,2,3} to control Actuator A {0,1,2}.
void comm()
{
  if (Serial.available() > 0)
  {
    String incoming = "";
    while (Serial.available() > 0)
    {

      test  = char(Serial.read());
      incoming  += test;
      delay(100);
    }
    temp = incoming.substring(0, 1);
    if (temp == "C")
    {
      actuator_control = incoming.substring(1, 2).toInt();
      sensor_control = 4;
      sensor_val = incoming.substring(2, incoming.length()).toInt();
      setpoint = map(sensor_val, 0, 1023, 0, 720);
      currentPosition = myEnc.read();
      Setpoint = currentPosition + setpoint;
      Setpoint_v = sensor_val * 0.0008211;
     // curState=prevState+1;
    }
    else if (temp == "S")
    {
      actuator_control = incoming.substring(1, 2).toInt();
      sensor_control = incoming.substring(2, 3).toInt();
    }
  }
}

//One button to control them all, not really!
void debounce(int value_button){
  if (value_button>1000)
  {
    current = HIGH;
  }
  else
  {
    current = LOW;
  }
  if (current == last)
  {
    lastDebounceTime = millis();
  }
  else if (current != last && (millis() - lastDebounceTime) > debounceDelay)
  {
    last = current;
    state = !state;
    if (state == HIGH)
    {
      Serial.print("0,0,0,");
    }
```

```
    else
    {
      Serial.print("X,X,X,");
    }
  }
}

void velocityControl( int m_speed, int m_direction)
{
  if (m_direction == -1)
  {
    analogWrite(m1, m_speed);
    digitalWrite(m2, LOW);
  }
  else if (m_direction == +1)
  {
    analogWrite(m2, m_speed);
    digitalWrite(m1, LOW);
  }
  else
  {
    digitalWrite(m1, LOW);
    digitalWrite(m2, LOW);
  }
}

//Pranav
void sharp_sensor()
{
  sensor_val = sharp.distance();
  if ((sensor_val > 5) && (sensor_val < 81))
  {
    sensor_val = (sensor_val - 6) * 13;
  }
  else
  {
    sensor_val = 0;
  }


}

//Richa
void slot_sensor()
{
  sensor_val = 0;
```

```
  if (analogRead(slot) > 1000)
  {
    sensor_val = 1023;
  }

}

//Dorothy
void ultrasonic_sensor()
{
  // Get base reading.  Subtract 818 to get values in correct range.
  // Multiply by 0.017 to change to inches.
  // Multiply by 36.54 to convert from inches to 0-1023 range
  sensor_val = 0;

  int lastRead = ((pulseIn(ultra, HIGH) - 818) * 0.017) * 36.54;
  int temp;
  // sum 10 good readings that will be averaged to filter out noise
  for (int i = 0; i < 10; i++)
  {
    temp = ((pulseIn(ultra, HIGH) - 818) * 0.017) * 36.54;
    if (temp > 1100)
    {
      sensor_val += 1023;
    }
    // if out of range or too different from last reading, get a new reading
    else if (temp < 0 || abs(temp - lastRead) > 50)
    {
      i--;
    }
    // if reading is acceptable, sum it
```

```
    else
    {
      sensor_val += temp;
    }
    lastRead = temp;
  }
  // if average is too high, just return 1023.  otherwise return average.
  if ((sensor_val / 10) > 1023)
  {
    sensor_val = 1023;
  }
  else
  {
    sensor_val = sensor_val / 10;
  }

}

//Shivam
void flex_sensor()
{
  int frc = analogRead(flex);
  float voltage = frc * (5.0 / 1023.0);
  float frcRes = ((5 - voltage) * 10000) / voltage; //frc resistance= ((Vcc-Va)*R)/Va
  float force = 0;
  if (frcRes > 80000)
  {
    force = 0;
  }
  else
  {
    force = 1 + ((-frcRes + 80000) / 77500);
  }
  int force_map = force * 100;
  sensor_val = map(force_map, 0, 200, 0, 1023);

}

void serial_handler(String X)
{
   setpoint = map(sensor_val, 0, 1023, 0, 720);
 if (abs(prevState-setpoint)>40)
    {

  currentPosition = myEnc.read();
  Setpoint = currentPosition + setpoint;
  Setpoint_v = sensor_val * 0.0008211;
  flag = 1;
  prevState=setpoint;
    }

  if (currentMillis - previousMillis > interval)
  {
    previousMillis = currentMillis;
    Serial.print(X);
    Serial.print(sensor_val);
    Serial.print(",");
  }
}

void loop()
{
```

```
currentMillis = millis();
comm();
debounce(analogRead(button));
switch (sensor_control)
{
  case 0:
    slot_sensor();
    serial_handler("S");
    break;
  case 1:
    ultrasonic_sensor();
    serial_handler("U");
    break;
  case 2:
    sharp_sensor();
    serial_handler("I");
    break;
  case 3:
    flex_sensor();
    serial_handler("T");
    break;
  default:
    //check for some stuff, example sensor_val value, if received through Serial.
    break;
}

switch (actuator_control)
{
  case 0:
    serv = map(sensor_val, 0, 1023, 0, 180);
    myservo.write(serv);
    delay(20);
    break;
  case 1:
    if (prev_sensor_val != sensor_val)
    {
      val = (sensor_val - prev_sensor_val) / 5.12; //Sensor_val to steps conversion with difference between present and desired state
      if (val > 0)
      {
        digitalWrite(stepper_direction, HIGH);
        while (val > 0)
        {
          val--;
          digitalWrite(stepper_signal, HIGH);
          delay(5);
          digitalWrite(stepper_signal, LOW);
```

```
            delay(5);
          }
        }
        if (val < 0)
        {
          digitalWrite(stepper_direction, LOW);
          while (val < 0)
          {
            val++;
            digitalWrite(stepper_signal, HIGH);
            delay(5);
            digitalWrite(stepper_signal, LOW);
            delay(5);
          }
        }
        prev_sensor_val = sensor_val;
      }
      break;

    case 2:


      for (int i = 0; i < 300; i++)
      {
        Input = myEnc.read();
        myPID.Compute();
        m_speed = abs(Output);
        m_direction = Output > 0 ? 1 : -1;
        velocityControl( m_speed, m_direction);
        delay(5);
      }
      velocityControl( 0, m_direction);



      break;

    case 3:

      float initialCount = myEnc.read(), finalCount = 0;
      delay(50);
      finalCount = myEnc.read();
      currentSpeed = (finalCount - initialCount) / 50;
      Input_v = currentSpeed;
      myPID_v.Compute();
      m_direction = 1;

      float spMap = (Setpoint_v * 255) / 0.84 ;
      m_speed = Output_v + spMap;
      velocityControl( m_speed,  m_direction);

      break;
  }
  oldactuator = actuator_control;
}
```

b. GUI

i. TeamISensors.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-10-10T17:04:02
#
#-------------------------------------------------

QT       += core gui serialport
```

```
8.
9.  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport
10.
11. TARGET = TeamISensors
12. TEMPLATE = app
13.
14.
15. SOURCES += main.cpp \
16.     mainwindow.cpp
17.
18. HEADERS  += \
19.     mainwindow.h
20.
21. FORMS    += \
22.      mainwindow.ui
```

i. mainwindow.h

```
1.  #ifndef MAINWINDOW_H
2.  #define MAINWINDOW_H
3.
4.  #include <QMainWindow>
5.  #include <QtSerialPort/QSerialPort>
6.  #include <QObject>
7.  #include <QIODevice>
8.
9.  namespace Ui
10. {
11.     class MainWindow;
12. }
13.
14. class MainWindow : public QMainWindow
15. {
16.     Q_OBJECT
17.
18. public:
19.     explicit MainWindow(QWidget *parent = 0);
20.     ~MainWindow();
21.
22. private slots:
23.     void readSerial();
24.     void parseInput(QString input);
25.
26.     void updateSensor(int senseNum, int sensVal);
27.     void updateMotor(int motorNum, int motorVal);
28.
29.     void enableMotor(int motorNum);
30.     void enableSensor(int sensorNum);
31.
32.     void on_SlidersUpdate_clicked();
33.     void on_SensorUpdate_clicked();
34.
35.     void on_ServoButton_clicked();
36.     void on_StepperButton_clicked();
37.     void on_DCPosButton_clicked();
38.     void on_DCVelButton_clicked();
39.
40.
41.     void on_ServoSlider_sliderReleased();
42.     void on_StepperSlider_sliderReleased();
43.     void on_DCPosSlider_sliderReleased();
44.     void on_DCVelSlider_sliderReleased();
45.
46.     void on_SlotButton_clicked();
47.     void on_UltraButton_clicked();
48.     void on_IRButton_clicked();
49.     void on_ForceButton_clicked();
50.
51. private:
```

```
52.     Ui::MainWindow *ui;
53.     QSerialPort *arduino;
54.     static const quint16 vendID = 9025;
55.     static const quint16 prodID = 67;
56.     QString arduinoPortName;
57.     bool arduinoAvailable;
58.     bool signalAvailable;
59.     bool arduinoWritable;
60.     QByteArray serialData;
61.     QString serialBuffer;
62.     int activeSensor;
63.     int activeMotor;
64.     int updateMode;
65. };
66.
67. #endif // MAINWINDOW_H
```

i. main.cpp

```
1.  #include "mainwindow.h"
2.  #include <QApplication>
3.  #include <QtSerialPort/QSerialPort>
4.
5.  int main(int argc, char *argv[])
6.  {
7.      QApplication a(argc, argv);
8.      MainWindow w;
9.      w.show();
10.     w.setWindowTitle("Team I Sensors Lab");
11.     w.setWindowIcon(QIcon("images.jpg"));
12.
13.     w.setFixedSize(1200,600);
14.
15.     return a.exec();
16. }
```

i. mainwindow.cpp

```
1.  #include "mainwindow.h"
2.  #include "ui_mainwindow.h"
3.  #include <QtSerialPort/QSerialPort>
4.  #include <QtSerialPort/QSerialPortInfo>
5.  #include <QDebug>
6.  #include <QtWidgets>
7.  #include <QObject>
8.  #include <QIODevice>
9.
10. MainWindow::MainWindow(QWidget *parent) :
11.     QMainWindow(parent),
12.     ui(new Ui::MainWindow)
13. {
14.     ui->setupUi(this);
15.
16.     arduino = new QSerialPort;
17.     arduinoPortName = "";
18.     arduinoAvailable = false;
19.     arduinoWritable=false;
20.     signalAvailable=false;
21.     serialBuffer = "";
22.     activeSensor = 0;
23.     activeMotor = 0;
24.     updateMode = 0;
25.
26.     ui->ServoLabel->setPalette(Qt::blue);
27.     ui->SlotValue->setPalette(Qt::lightGray);
28.     ui->UltraValue->setPalette(Qt::lightGray);
29.     ui->IRValue->setPalette(Qt::lightGray);
30.     ui->ForceValue->setPalette(Qt::lightGray);
```

```
31.
32.     // Check each comm port to find arduino
33.     foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts())
34.     {
35.         if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.hasProductIdentifier())
36.         {
37.             if(serialPortInfo.vendorIdentifier() == vendID)
38.             {
39.                 if(serialPortInfo.productIdentifier() == prodID)
40.                 {
41.                     arduinoPortName = serialPortInfo.portName() ;
42.                     arduinoAvailable = true;
43.                 }
44.             }
45.         }
46.     }
47.
48.     if(arduinoAvailable)
49.     {
50.         arduino->setPortName(arduinoPortName);
51.         arduino->open(QIODevice::ReadWrite);
52.
53.         if(arduino->isWritable())
54.         {
55.             arduinoWritable=true;
56.             qDebug() << "Arduino is Writable";
57.         }
58.         else
59.         {
60.             qDebug() << "Arduino is NOT Writable";
61.         }
62.         arduino->setBaudRate(QSerialPort::Baud9600);
63.         arduino->setDataBits(QSerialPort::Data8);
64.         arduino->setFlowControl(QSerialPort::NoFlowControl);
65.         arduino->setParity(QSerialPort::NoParity);
66.         arduino->setStopBits(QSerialPort::OneStop);
67.
68.         QObject::connect(arduino, SIGNAL(readyRead()), this, SLOT(readSerial()));
69.
70.     }
71.     else if((arduinoAvailable) && (!arduinoWritable))
72.     {
73.         QMessageBox::warning(this, "Port Error", "Couldn't find the arduino port
    number!");
74.     }
75. }
76.
77. MainWindow::~MainWindow()
78. {
79.     if(arduino->isOpen())
80.     {
81.         arduino->close();
82.     }
83.
84.     delete ui;
85. }
86.
87. // When user moves Servo slider, send message to arduino and update number on GUI
88. void MainWindow::on_ServoSlider_sliderReleased()
89. {
90.     int position = ui->ServoSlider->value();
91.     updateMotor(0, position);
92.
93.     QString arduinoOutput = "C0" + QString("%1").arg(position);
94.
95.     if(arduino->isWritable())
96.     {
97.         arduino->write(arduinoOutput.toStdString().c_str());
98.         qDebug() << "Sending message to arduino: " << arduinoOutput;
99.     }
100.      else
```

```
101.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
102.  }
103.
104.  // When user moves Stepper slider, send message to arduino and update number on GUI
105.  void MainWindow::on_StepperSlider_sliderReleased()
106.  {
107.      int position = ui->StepperSlider->value();
108.      updateMotor(1, position);
109.
110.      QString arduinoOutput = "C1" + QString("%1").arg(position);
111.
112.      if(arduino->isWritable())
113.      {
114.          arduino->write(arduinoOutput.toStdString().c_str());
115.          qDebug() << "Sending message to arduino: " << arduinoOutput;
116.      }
117.      else
118.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
119.  }
120.
121.  // When user moves DC Angle Position slider, send message to arduino and update number
    on GUI
122.  void MainWindow::on_DCPosSlider_sliderReleased()
123.  {
124.      int position = ui->DCPosSlider->value();
125.      updateMotor(2, position);
126.
127.      QString arduinoOutput = "C2" + QString("%1").arg(position);
128.
129.      if(arduino->isWritable())
130.      {
131.          arduino->write(arduinoOutput.toStdString().c_str());
132.          qDebug() << "Sending message to arduino: " << arduinoOutput;
133.      }
134.      else
135.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
136.  }
137.
138.  // When user moves DC Velocity slider, send message to arduino and update number on GUI
139.  void MainWindow::on_DCVelSlider_sliderReleased()
140.  {
141.      int position = ui->DCVelSlider->value();
142.      updateMotor(3, position);
143.
144.      QString arduinoOutput = "C3" + QString("%1").arg(position);
145.
146.      if(arduino->isWritable())
147.      {
148.          arduino->write(arduinoOutput.toStdString().c_str());
149.          qDebug() << "Sending message to arduino: " << arduinoOutput;
150.      }
151.      else
152.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
153.  }
154.
155.  // When something has been added to the serial port, read it
156.  void MainWindow::readSerial()
157.  {
158.      // arduino is seperating messages with commas
159.      QStringList bufferSplit = serialBuffer.split(",");
160.
161.      // only read items if the length is less than three
162.      // this ensures that complete messages are being read
163.      if(bufferSplit.length() < 3)
164.      {
165.          serialData = arduino->readAll();
166.          serialBuffer += QString::fromStdString(serialData.toStdString());
167.      }
168.      else if (bufferSplit.length() >=3)
169.      {
170.          // print to debug what will be used
```

```
171.          // using second item in case first message sent is incomplete or in error
172.          qDebug() << bufferSplit[1];
173.          parseInput(bufferSplit[1]);
174.          serialBuffer = bufferSplit[1];
175.
176.          // un-split serialBuffer, excluding the first item
177.          for(int i = 2; i < bufferSplit.length(); i++)
178.          {
179.              serialBuffer = serialBuffer + "," + bufferSplit[i];
180.          }
181.      }
182.  }
183.
184.  // interprets input sent from arduino and responds accordingly
185.  void MainWindow::parseInput(QString input)
186.  {
187.      int sensVal;
188.      if(input.at(0) == "X") // servo motor off
189.      {
190.          // hide label
191.          ui->ArduinoPowerLabel->show();
192.      }
193.      else if(input.at(0) == "O") // servo motor on
194.      {
195.          // show label and send current sensor/motor combo to arduino if updating via
    sensors
196.          ui->ArduinoPowerLabel->hide();
197.          if(updateMode == 1)
198.          {
199.              QString arduinoOutput = "S" + QString("%1").arg(activeMotor)+
    QString("%1").arg(activeSensor);
200.
201.              if(arduino->isWritable())
202.              {
203.                  arduino->write(arduinoOutput.toStdString().c_str());
204.                  qDebug() << "Sending message to arduino: " << arduinoOutput;
205.              }
206.              else
207.                  qDebug() << "Can't write following message to arduino: " <<
    arduinoOutput;
208.          }
209.      }
210.      else if(input.at(0) == "S") // Slot
211.      {
212.          // remove identifying char from string so only sensor output remains
213.          input.remove(0,1);
214.          // convert sensor output to int
215.          sensVal = input.toInt();
216.          // if updating motors via sensors and this is the correct active sensor, update
    GUI appropriately
217.          if(updateMode == 1 && activeSensor == 0)
218.          {
219.              updateSensor(0, sensVal);
220.              updateMotor(activeMotor, sensVal);
221.          }
222.      }
223.      else if(input.at(0) == "U") // Ultra
224.      {
225.          // remove identifying char from string so only sensor output remains
226.          input.remove(0,1);
227.          // convert sensor output to int
228.          sensVal = input.toInt();
229.          // if updating motors via sensors and this is the correct active sensor, update
    GUI appropriately
230.          if(updateMode == 1 && activeSensor == 1)
231.          {
232.              updateSensor(1, sensVal);
233.              updateMotor(activeMotor, sensVal);
234.          }
235.      }
236.      else if(input.at(0) == "I") // IR
```

```
237.      {
238.          // remove identifying char from string so only sensor output remains
239.          input.remove(0,1);
240.          // convert sensor output to int
241.          sensVal = input.toInt();
242.          // if updating motors via sensors and this is the correct active sensor, update
    GUI appropriately
243.          if(updateMode == 1 && activeSensor == 2)
244.          {
245.              updateSensor(2, sensVal);
246.              updateMotor(activeMotor, sensVal);
247.          }
248.      }
249.      else if(input.at(0) == "T") // Force
250.      {
251.          // remove identifying char from string so only sensor output remains
252.          input.remove(0,1);
253.          // convert sensor output to int
254.          sensVal = input.toInt();
255.          // if updating motors via sensors and this is the correct active sensor, update
    GUI appropriately
256.          if(updateMode == 1 && activeSensor == 3)
257.          {
258.              updateSensor(3, sensVal);
259.              updateMotor(activeMotor, sensVal);
260.          }
261.      }
262.      else
263.      {
264.          qDebug() << "Serial message in incorrect format.";
265.      }
266.  }
267.
268.  // changes sensor output to appropriate measurement and updates GUI
269.  void MainWindow::updateSensor(int senseNum, int senseVal)
270.  {
271.      qreal mappedVal;
272.      switch(senseNum)
273.      {
274.      case 0: // Slot sensor
275.          if(activeSensor == 0)
276.          {
277.              // Slot sensor is only on/off, so only display on/off
278.              if(senseVal > 0)
279.                  ui->SlotValue->display(1);
280.              else
281.                  ui->SlotValue->display(0);
282.          }
283.          break;
284.      case 1:// Ultrasonic
285.          if(activeSensor == 1)
286.          {
287.              // Converts value to 4-24 inches range
288.              mappedVal = senseVal/51.15+4;
289.              ui->UltraValue->display(qRound(mappedVal));
290.          }
291.          break;
292.      case 2:    // IR
293.          if(activeSensor == 2)
294.          {
295.              // Converts value to 6-80 cm range
296.              mappedVal = ((80.0-6)/1023)*senseVal+6;
297.              ui->IRValue->display(qRound(mappedVal));
298.          }
299.          break;
300.      case 3:    // Force
301.          if(activeSensor == 3)
302.          {
303.              // Converts value to 0-20 N range
304.              mappedVal = ((20.0)/1023)*senseVal;
305.              ui->ForceValue->display(qRound(mappedVal));
```

```
306.          }
307.          break;
308.      }
309.  }
310.
311.  // changes motor output to appropriate measurement and updates GUI
312.  void MainWindow::updateMotor(int motorNum, int motorVal)
313.  {
314.      qreal mappedVal;
315.      switch(motorNum)
316.      {
317.      case 0: // Servo
318.          if(activeMotor == 0)
319.          {
320.              // converts value to 0-180 degree range
321.              mappedVal = ((180.0)/1023)*motorVal;
322.              ui->ServoLabel->display(qRound(mappedVal));
323.          }
324.          break;
325.      case 1: // Stepper
326.          if(activeMotor == 1)
327.          {
328.              // converts value to 0-359 degree range
329.              mappedVal = ((359.0)/1023)*motorVal;
330.              ui->StepperLabel->display(qRound(mappedVal));
331.          }
332.          break;
333.      case 2: // DC Pos
334.          if(activeMotor == 2)
335.          {
336.              // converts value to 0-359 degree range
337.              mappedVal = ((359.0)/1023)*motorVal;
338.              ui->DCPosLabel->display(qRound(mappedVal));
339.          }
340.          break;
341.      case 3: //DC Vel
342.          if(activeMotor == 3)
343.          {
344.              // converts value to 0-58 rpm range
345.              mappedVal = ((58.0)/1023)*motorVal;
346.              ui->DCVelLabel->display(qRound(mappedVal));
347.          }
348.      }
349.  }
350.
351.  // if updating via GUI, disable all sensor information and enable selected motor
352.  void MainWindow::on_SlidersUpdate_clicked()
353.  {
354.      updateMode = 0;
355.      ui->SlotButton->setEnabled(false);
356.      ui->SlotUnits->setEnabled(false);
357.      ui->SlotValue->display(0);
358.      ui->SlotValue->setEnabled(false);
359.      ui->SlotValue->setPalette(Qt::lightGray);
360.      ui->UltraButton->setEnabled(false);
361.      ui->UltraUnits->setEnabled(false);
362.      ui->UltraValue->display(0);
363.      ui->UltraValue->setEnabled(false);
364.      ui->UltraValue->setPalette(Qt::lightGray);
365.      ui->IRButton->setEnabled(false);
366.      ui->IRUnits->setEnabled(false);
367.      ui->IRValue->display(0);
368.      ui->IRValue->setEnabled(false);
369.      ui->IRValue->setPalette(Qt::lightGray);
370.      ui->ForceButton->setEnabled(false);
371.      ui->ForceUnits->setEnabled(false);
372.      ui->ForceValue->display(0);
373.      ui->ForceValue->setEnabled(false);
374.      ui->ForceValue->setPalette(Qt::lightGray);
375.
376.      enableMotor(activeMotor);
```

```
377.  }
378.
379.  // if updating via sensors, disable motor sliders and enable selected sensor
380.  void MainWindow::on_SensorUpdate_clicked()
381.  {
382.      updateMode = 1;
383.      ui->SlotButton->setEnabled(true);
384.      ui->SlotUnits->setEnabled(true);
385.      ui->UltraButton->setEnabled(true);
386.      ui->UltraUnits->setEnabled(true);
387.      ui->IRButton->setEnabled(true);
388.      ui->IRUnits->setEnabled(true);
389.      ui->ForceButton->setEnabled(true);
390.      ui->ForceUnits->setEnabled(true);
391.
392.      ui->StepperSlider->setEnabled(false);
393.      ui->ServoSlider->setEnabled(false);
394.      ui->DCPosSlider->setEnabled(false);
395.
396.      ui->ServoLabel->display(0);
397.      ui->StepperLabel->display(0);
398.      ui->DCPosLabel->display(0);
399.
400.      enableSensor(activeSensor);
401.  }
402.
403.  // disable all motors but one selected
404.  // send appropriate message to arduino
405.  void MainWindow::enableMotor(int motorNum)
406.  {
407.      ui->ServoSlider->setEnabled(false);
408.      ui->ServoSlider->setSliderPosition(0);
409.      ui->ServoLabel->setEnabled(false);
410.      ui->ServoLabel->display(0);
411.      ui->ServoLabel->setPalette(Qt::lightGray);
412.
413.      ui->StepperSlider->setEnabled(false);
414.      ui->StepperSlider->setSliderPosition(0);
415.      ui->StepperLabel->setEnabled(false);
416.      ui->StepperLabel->display(0);
417.      ui->StepperLabel->setPalette(Qt::lightGray);
418.
419.      ui->DCPosSlider->setEnabled(false);
420.      ui->DCPosSlider->setSliderPosition(0);
421.      ui->DCPosLabel->setEnabled(false);
422.      ui->DCPosLabel->display(0);
423.      ui->DCPosLabel->setPalette(Qt::lightGray);
424.
425.      ui->DCVelSlider->setEnabled(false);
426.      ui->DCVelSlider->setSliderPosition(0);
427.      ui->DCVelLabel->setEnabled(false);
428.      ui->DCVelLabel->display(0);
429.      ui->DCVelLabel->setPalette(Qt::lightGray);
430.
431.      switch(motorNum)
432.      {
433.      case 0: // Servo
434.          if(updateMode == 0)
435.              ui->ServoSlider->setEnabled(true);
436.          ui->ServoLabel->setEnabled(true);
437.          ui->ServoLabel->setPalette(Qt::blue);
438.          break;
439.      case 1: // Slider
440.          if(updateMode == 0)
441.              ui->StepperSlider->setEnabled(true);
442.          ui->StepperLabel->setEnabled(true);
443.          ui->StepperLabel->setPalette(Qt::blue);
444.          break;
445.      case 2: // DC Pos
446.          if(updateMode == 0)
447.              ui->DCPosSlider->setEnabled(true);
```

```
448.          ui->DCPosLabel->setEnabled(true);
449.          ui->DCPosLabel->setPalette(Qt::blue);
450.          break;
451.      case 3: // DC Vel
452.          if(updateMode == 0)
453.              ui->DCVelSlider->setEnabled(true);
454.          ui->DCVelLabel->setEnabled(true);
455.          ui->DCVelLabel->setPalette(Qt::blue);
456.          break;
457.      }
458.
459.      QString arduinoOutput;
460.      if(updateMode == 0)
461.      {
462.           arduinoOutput = "C" + QString("%1").arg(activeMotor) + QString("0");
463.      }
464.      else
465.      {
466.           arduinoOutput = "S" + QString("%1").arg(activeMotor)+
    QString("%1").arg(activeSensor);
467.      }
468.
469.      if(arduino->isWritable())
470.      {
471.          arduino->write(arduinoOutput.toStdString().c_str());
472.          qDebug() << "Sending message to arduino: " << arduinoOutput;
473.      }
474.      else
475.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
476.  }
477.
478.  // disable all sensors but one selected
479.  // send appropriate message to arduino
480.  void MainWindow::enableSensor(int sensorNum)
481.  {
482.      ui->SlotValue->setEnabled(false);
483.      ui->SlotValue->display(0);
484.      ui->SlotValue->setPalette(Qt::lightGray);
485.      ui->UltraValue->setEnabled(false);
486.      ui->UltraValue->display(0);
487.      ui->UltraValue->setPalette(Qt::lightGray);
488.      ui->IRValue->setEnabled(false);
489.      ui->IRValue->display(0);
490.      ui->IRValue->setPalette(Qt::lightGray);
491.      ui->ForceValue->setEnabled(false);
492.      ui->ForceValue->display(0);
493.      ui->ForceValue->setPalette(Qt::lightGray);
494.
495.      QString arduinoOutput = "S" + QString("%1").arg(activeMotor);
496.
497.      switch(sensorNum)
498.      {
499.      case 0: // Slot
500.          ui->SlotValue->setEnabled(true);
501.          ui->SlotValue->setPalette(Qt::blue);
502.          arduinoOutput = arduinoOutput + QString("0");
503.          break;
504.      case 1: // Ultra
505.          ui->UltraValue->setEnabled(true);
506.          ui->UltraValue->setPalette(Qt::blue);
507.          arduinoOutput = arduinoOutput + QString("1");
508.          break;
509.      case 2: // IR
510.          ui->IRValue->setEnabled(true);
511.          ui->IRValue->setPalette(Qt::blue);
512.          arduinoOutput = arduinoOutput + QString("2");
513.          break;
514.      case 3: // Force
515.          ui->ForceValue->setEnabled(true);
516.          ui->ForceValue->setPalette(Qt::blue);
517.          arduinoOutput = arduinoOutput + QString("3");
```

```
518.          break;
519.      }
520.
521.      if(arduino->isWritable())
522.      {
523.          arduino->write(arduinoOutput.toStdString().c_str());
524.          qDebug() << "Sending message to arduino: " << arduinoOutput;
525.      }
526.      else
527.          qDebug() << "Can't write following message to arduino: " << arduinoOutput;
528.  }
529.
530.  void MainWindow::on_ServoButton_clicked()
531.  {
532.      activeMotor = 0;
533.      enableMotor(0);
534.  }
535.
536.  void MainWindow::on_StepperButton_clicked()
537.  {
538.      activeMotor = 1;
539.      enableMotor(1);
540.  }
541.
542.  void MainWindow::on_DCPosButton_clicked()
543.  {
544.
545.      activeMotor = 2;
546.      enableMotor(2);
547.  }
548.
549.  void MainWindow::on_DCVelButton_clicked()
550.  {
551.      activeMotor = 3;
552.      enableMotor(3);
553.  }
554.  void MainWindow::on_SlotButton_clicked()
555.  {
556.      activeSensor = 0;
557.      enableSensor(0);
558.  }
559.
560.  void MainWindow::on_UltraButton_clicked()
561.  {
562.      activeSensor = 1;
563.      enableSensor(1);
564.  }
565.
566.  void MainWindow::on_IRButton_clicked()
567.  {
568.      activeSensor = 2;
569.      enableSensor(2);
570.  }
571.
572.  void MainWindow::on_ForceButton_clicked()
573.  {
574.      activeSensor = 3;
575.      enableSensor(3);
576.  }
```