Sensors and Motor Control Lab

Tushar Agrawal

Team A - Avengers
Teammates: Adam Yabroudi, Pratik Chatrath, Sean Bryan

ILR #1

October 15, 2015

1. Individual Progress

For the Sensors and Motors lab, I was responsible for implementing the DC motor and ultrasonic sensor.

The subsystem used the Cytron SPG30-60K DC motor (12VDC, 60:1 geared), Solarbotics L298 Compact Motor Driver and the Maxbotix EZ0 Ultrasonic sensor. A quadrature rotary Hall-Effect encoder is attached with the DC motor for movement feedback. A SPDT slide switch was used to choose whether the sensor controlled the position or the velocity of the DC motor.

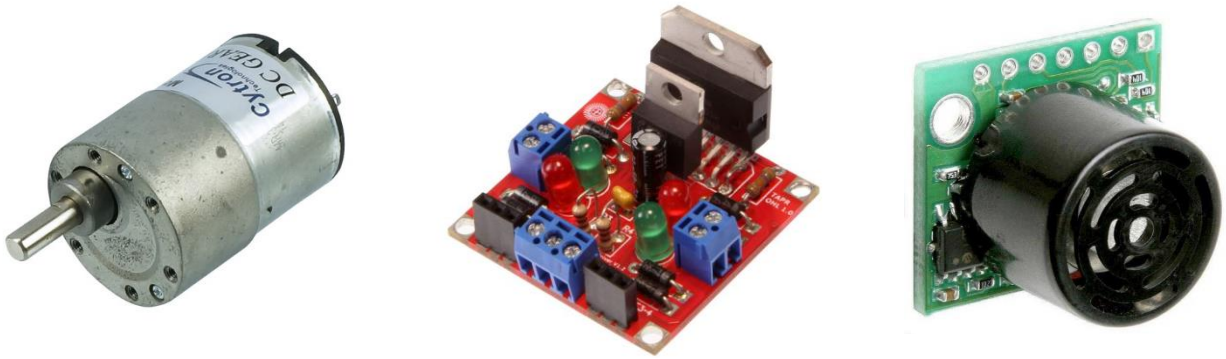

Figure 1. a. DC motor. b. Solarbotics compact motor driver. c. Maxbotix ultrasonic sensor

The following circuit was used to connect the motor, encoder and the ultrasonic sensor.

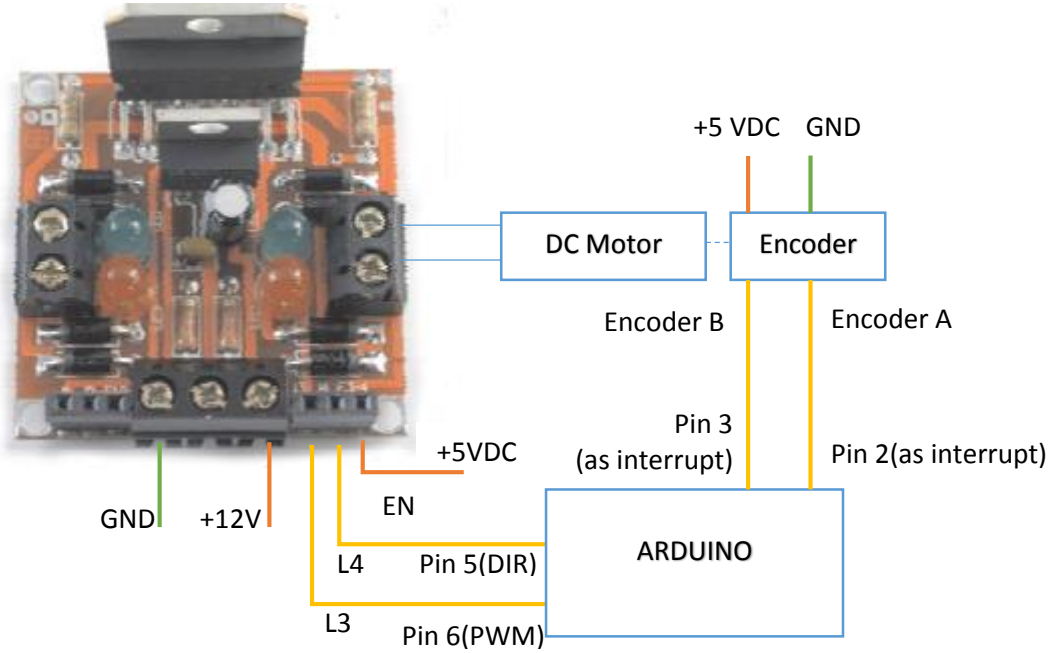

Figure 2. Motor and Encoder schematic

I used the battery eliminator to provide both the 5V DC line for the components (except the Arduino, which was powered by the USB) and the 12V line for the DC motor.

Ultrasonic:

I was excited by the application of Ultrasonic sensors as the Obstacle detection system for our project, so, I first connected the Ultrasonic sensor into the Arduino by the analog output to test the raw data received. The raw data has small spikes due to some noise. I implemented a basic median filter to reduce the noise. Feeling satisfied to a certain extent, I tested this against different sized obstacles at different orientations and distances from the sensor and realized that there was more noise in cases of small obstacles as opposed to bigger ones. A good response was seen from the sensor in the range of 5 to 200 inches. For ease of demonstration, 5 to 30 inches range was used.

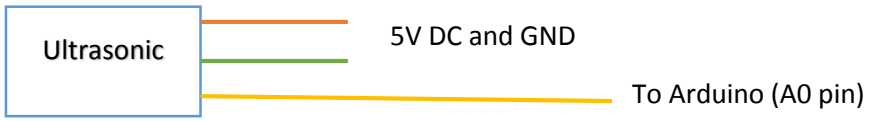

Figure 3. Ultrasonic schematic

Motor and Encoder:

To start working with the DC motor, I plugged in the motor to 12V to check that it works. After verifying that, I connected it using the motor driver to the Arduino, and tested basic open loop velocity control using PWM in both directions. Then, I made the encoder connections. I used the interrupt pins on the Arduino after confirming that no one else in the team required them. I plugged in the encoder code for reading interrupts and updating the encoder ticks. To test the encoder, I verified that the encoder ticks increased as the motor rotated in one direction and reduced in the other direction. For position control, an angle taken from the user was converted into required encoder ticks (encoder had 2 ticks/degree) and controlled. I used a PID library (PID_v1) for implementing PID control on the required encoder ticks. A switch was used to shift between position and velocity control.

Ultrasonic Integration:

To control the position and velocity based on the output of the sensor, the code was made generic to have an input end, which set the control parameters like position or velocity, and an output end, where depending on the mode, the input parameters were implemented. For instance, when the mode was sensor control and button was set on position control, the input position parameter would be mapped based on the sensor value and pre-programmed ranges. This position would then be implemented using individual pins and PWM with PID by the output system.

PID Tweaking

Tuning the PID setting took considerable time. Trying with a recommended set of Ks was not giving a satisfactory response. With the help of Adam, we increased Kp, keeping the rest at zero until we started getting fast response (with some oscillations), then we increased the Kd value to try and damn the oscillations. A very low Kd was working well as the static and kinetic friction was quite different for the motor and hence the value was damped mostly automatically. Very minute Ki was used to slowly degrade the steady state error seen. Finally the values were set to Kp =4, Ki=0.001, Kd=0.1.

At this stage, I had a working position and velocity control by entering values or using the motor.

GUI and Protocol:

Worked with Pratik to resolve a few serial port issues in Qt (permission problem), and designed the protocol for sending sensor updates to the GUI and position/velocity updates from it.

Integration:

Took Sean and Adam's implementation of the Stepper and the Servo motor and integrated into the main code framework, which separated the input/output and selected the sensor value to send.

Then, I integrated the serial protocol to change the required parameters in the Arduino. And tested the same with the GUI. After minor fixes, full functionality was available.

Project – Setting up the Beagleboard xM

I set up the beagleboard with power from adapter, DVI-D to screen and internet from Ethernet (not in the lab). The default image (Ubuntu 13.04) booted up and was well configured for the network with a static local IP. It had ROS but not OpenCV. Also Ubuntu 13.04 was not our choice of the operating system as it was not supported long term and its package sources were deprecated. So, I tried to replace the default image with a 14.04 version of the image. 2 such images failed to display anything on the DVI-D screen. Later, by attaching the beagleboard to the serial port using the RS-232 to USB cable, I was able to see the image booting up. But, it detected a fault in the image halfway through the boot. I am planning to debug that this week before setting it up with ROS and OpenCV.

2. Challenges

While doing velocity control on the DC motor, I faced some issues to understanding how the PWM data needs to be changed when direction changes. After reading in the data sheet, I realized that the motor works on the difference in the signal, so a High direction with 0 PWM means a high speed in one direction and to obtain the same speed in the other direction, direction pin should be Low with a 255 PWM.
Another challenge was faced when tuning PIDs, as the response was not responding well to changes in Kd. After discussion with Adam, I realized that I needed to use a higher Kp and A very low Kd, as friction had a major role to play.

Also, I faced some issues in the beagleboard setup. The Ubuntu 14.04 images for armv7l architecture were not giving DVID output. Also, based on their boot-up sequence seen over the RS232 serial, there is some problem in the image which stops the booting procedure midway.

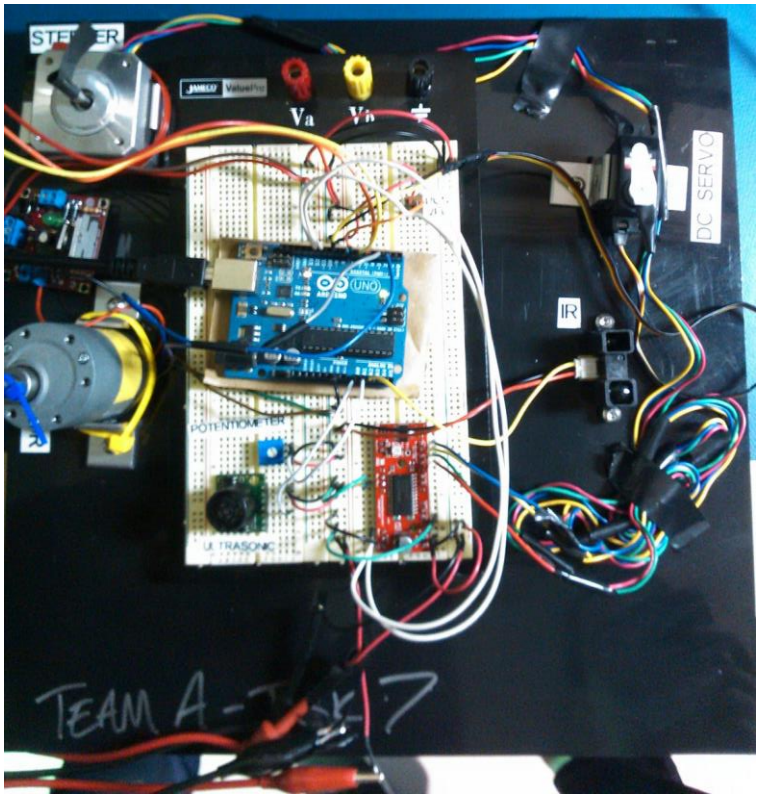

Figure 4. Final system

3. Teamwork

As recommended, we split the work for the motor lab amongst ourselves. I was responsible for the Ultrasonic and DC motor subsystem. Pratik was responsible for the GUI and testing. Adam was responsible for the IR and Servo subsystem in addition to the final electronic setup. Sean was responsible for the Potentiometer and Stepper subsystem in addition to mechanical system for holding motors and bread board. In addition to the subsystem, as the software lead, I took up the task to integrate the subsystems and interface serially with the GUI. Adam helped me in tuning the PID parameters, and helped in shifting the electronics from the Polulu stepper motor driver to the sparkfun one when it stopped working after integrating electronics. I helped Pratik with certain components of Qt and Serial communication, as I had previous experience in it. Overall, it was a good team effort and everyone contributed to the system.

4. Future Plans

Over the next week, I plan to work on setting up the BeagleBoard xM with an Ubuntu 14.04 image with internet access, SSH, ROS and hopefully OpenCV. I have already tried two images on the beagle board but I believe I am missing some configurations for display. After that I shall be interfacing a USB camera to it, and verify that the drivers detect it and it is ready to be used with the system. As a team, we are planning to move forward on three fronts. Firstly, we hope to get further details about ranges, and interferences in combinations of IR and Ultrasonic sensors for obstacle detection. Adam will be working on setting up the Pixhawk board as a flight controller and have it interfacing with some motors. Sean will be working on making a model of the firefly6 to run trial and error tests for finding locations of IR and Ultrasonic sensors. We will also work on defining the risk mitigation strategies and a detailed work breakdown structure for the Systems Presentation.

References

[1] Ultrasonic sensor https://www.sparkfun.com/products/8502

[2] Rotary Encoders http://playground.arduino.cc/Main/RotaryEncoders

[3] Solarbotics Motor Driver https://solarbotics.com/download.php?file=43

Appendix – Code:

Full code of the system is attached. My contribution includes all of the Arduino part except the Stepper and the Servo subsystems. It contains 2 parts:

Arduino side

```
#include <SharpIR.h>
#include <AccelStepper.h>
#include <PID_v1.h>
#include <RunningMedian.h>
#include <Servo.h>

// System Level Variables

int subsystem = 0;        // Chooses sensor-motor subsystem.
int mode = 0;             // Chooses GUI/Sensor control. GUI:0 Sensor:1
int isVel = 0;            // Chooses Velocity or Position control FROM GUI
int countLoop = 0;
int senseRate = 200;      // Send sensor reading once every senseRate loops

// Subsystem 0 ---------------------DC with Ultrasonic----------------------------
    //PINS
    int DC_pwm = 6;
    int DC_dir = 5;
    int Ultrasonic_pin = A0;
    int encoderPinA = 2;
    int encoderPinB = 3;
    int buttonPin = 7;
    int DCpos = 0;
    int DCvel = 0;
    int lastBval = 0;
    int lastVel = 0;
    // Is sensor controlling velocity (els position)
    int DCsensorIsVel = 0;

    RunningMedian USdistanceBuffer = RunningMedian(5);

    // Encoder Stuff
    volatile int encoderPos = 0;
    boolean A_set = false;
    boolean B_set = false;
    float ticksPerDeg = 2;

    // PID variables
    double PidSetpoint, PidInput, PidOutput;
```

```
    //Define the aggressive and conservative Tuning Parameters
    double aggKp=4, aggKi=0.001, aggKd=0.1;
    double consKp=1.1, consKi=0.001, consKd=0.03;

    //Specify the links and initial tuning parameters
    PID myPID(&PidInput, &PidOutput, &PidSetpoint, consKp, consKi, consKd, DIRECT);
// Subsystem 0 ------------------------------------------------------------------
Ends

// Subsystem 1 --------------------- Servo with IR -------------------------------

    int IR_Pin = A2; //analog
    int IR_Model = 20150; // for our specific sensor
    int num_points_avg = 50; // number of points to average/filter
    int percent_similar = 95; // +/- 7% of values are considered -- higher number
means they have to be more similar
    SharpIR sharp(IR_Pin, num_points_avg, percent_similar, IR_Model);

    Servo servo;
    int Servo_Pin = 9;

    int IR_out=0;
    int Servo_outVal=0;

    int IR_max = 70;
    int IR_min = 20;

// Subsystem 1 ------------------------------------------------------------------
Ends

// Subsystem 1 --------------------- Stepper with Potentiometer ------------------

    int potPin = A1;
    int potVal = 0;
    int newPos = 0;

    int stepPin = 11;
    int dirPin = 12;
    AccelStepper stepper(1, stepPin, dirPin);

// Subsystem 1 ------------------------------------------------------------------
Ends

int getDistance() {
  int distanceAnalog = (analogRead(Ultrasonic_pin)/2);
  return distanceAnalog;
}

// Interrupt on A changing state
void doEncoderA(){
  A_set = digitalRead(encoderPinA) == HIGH;
  // and adjust counter + if A leads B
  encoderPos += (A_set != B_set) ? +1 : -1;
}

// Interrupt on B changing state
void doEncoderB(){
  B_set = digitalRead(encoderPinB) == HIGH;
  // and adjust counter + if B follows A
  encoderPos += (A_set == B_set) ? +1 : -1;
}
```

```
void populateVariables(String str) {
    //Serial.print("Full String"); Serial.print('\t'); Serial.println(str);
    int firstIndex = str.indexOf(',');
    int secondIndex = str.indexOf(',', firstIndex+1);
    int thirdIndex = str.indexOf(',', secondIndex+1);
    int slData[4] = {0,0,0,0};
    slData[0] = (str.substring(0, firstIndex)).toInt();
    slData[1] = (str.substring(firstIndex+1, secondIndex)).toInt();
    slData[2] = (str.substring(secondIndex+1, thirdIndex)).toInt();
    slData[3] = (str.substring(thirdIndex+1)).toInt(); //To the end of the string
    subsystem = slData[0];
    stepper.setMaxSpeed(400);
    stepper.setAcceleration(20);
    mode = slData[1];
    if(mode!=0)
    {
        digitalWrite(DC_dir, LOW);
        analogWrite(DC_pwm, 0);
    }
    lastVel = isVel;
    isVel = slData[2];
    // GUI Control
    if(mode == 0) {
        if(isVel == 0) DCpos = slData[3];
        if(isVel == 1) DCvel = slData[3];
    }
}

void sendToGui(int subSys, float sensorVal)
{
    int sensorInt = 0;
    if(subSys == 1 || subSys == 0) sensorInt = sensorVal * 10;
    else if(subSys == 2) sensorInt = sensorVal;
    Serial.println(String(subSys) + "," + String(sensorInt));
}


// the setup routine runs once when you press reset:
void setup() {

    Serial.begin(115200);

    // DC + Encoder
    pinMode(DC_dir, OUTPUT);
    pinMode(encoderPinA, INPUT);
    pinMode(encoderPinB, INPUT);
    digitalWrite(encoderPinA, HIGH);  // turn on pullup resistor
    digitalWrite(encoderPinB, HIGH);  // turn on pullup resistor
    attachInterrupt(0, doEncoderA, CHANGE);  // encoder pin on interrupt 0 (pin 2)
    attachInterrupt(1, doEncoderB, CHANGE);  // encoder pin on interrupt 1 (pin 3)
    pinMode(buttonPin, INPUT);
    encoderPos = 0;

    // PID Settings
    myPID.SetMode(AUTOMATIC);
    myPID.SetTunings(aggKp, aggKi, aggKd);
    myPID.SetOutputLimits(-255,255);

    // Debug Values
    DCvel = 100;
    DCpos = 180;

    // Stepper + Potentiometer
```

```
    stepper.setMaxSpeed(400);
    stepper.setAcceleration(20);

    // Servo + IR
    servo.attach(Servo_Pin);
}


void loop() {
    if(Serial.available()) {
        populateVariables(Serial.readString());
    }
    float sensorVal;
    switch(subsystem) {
        // DC Motor Using Ultrasonic Sensor
        case 0:
            {
              float DCFinalPos;
              int DCFinalVel;
              if(mode == 0) {
                  if(isVel)
                  {
                     DCFinalVel = DCvel;
                     encoderPos = 0;
                  }
                  else DCFinalPos = DCpos;
              }
              else {
                  // Populate Button State
                  int bVal = digitalRead(buttonPin);
                  if(bVal == HIGH)
                      DCsensorIsVel = 1;
                  else
                  {
                      if(lastBval == HIGH)
                      {
                          encoderPos = 0;
                          //Serial.println("setting encoder sero");
                      }
                      DCsensorIsVel = 0;
                  }
                  lastBval = bVal;
                  //Serial.print("Button: "); Serial.println(DCsensorIsVel);
                  int USdistanceRaw = getDistance();
                  USdistanceBuffer.add(USdistanceRaw);
                  float USdistanceMedian = USdistanceBuffer.getAverage();
                  float lowInch = 5;
                  float highInch = 30;
                  sensorVal = USdistanceMedian;
                  if(DCsensorIsVel == 1) { // Velocity Control
                      DCFinalVel = ((USdistanceMedian - lowInch)/(highInch-lowInch) *
255) + 50;
                  }
                  else if(DCsensorIsVel == 0) { // Position Control
                      digitalWrite(DC_dir, LOW);
                      analogWrite(DC_pwm, 0);
                      float USposMotor = (USdistanceMedian - lowInch)/(highInch-
lowInch) * 200;
                      if(USposMotor> 200) USposMotor=200;
                      if(USposMotor < 0) USposMotor=0;
                      DCFinalPos = USposMotor;
                  }
              }
```

```
            if((mode == 0 && isVel == 1) || (mode == 1 && DCsensorIsVel == 1)) // DC
velocity control by sensor/GUI
            {
                //Serial.print('Final Velocity: '); Serial.print('\t');
Serial.println(DCFinalVel);
                digitalWrite(DC_dir, LOW);
                if(DCFinalVel < 256 && DCFinalVel >= 0)
                    analogWrite(DC_pwm, DCFinalVel);
                else
                    analogWrite(DC_pwm, 255);
            }
            else if((mode == 0 && isVel == 0) || (mode == 1 && DCsensorIsVel == 0))
// DC position control by sensor/GUI
            {
                int reqdEncoderCount = DCFinalPos * ticksPerDeg;
                PidSetpoint = reqdEncoderCount;
                PidInput = encoderPos;
                int error = PidSetpoint - PidInput;
                myPID.Compute();
//              Serial.print(USposMotor); Serial.print('\t');
Serial.print(PidSetpoint); Serial.print('\t'); Serial.print(PidInput);
Serial.print('\t'); Serial.print(error); Serial.print('\t'); Serial.print(PidOutput);
//              Serial.println();
                int dir= PidOutput>0 ? 1: -1;
                if(PidOutput > 0) {
                   digitalWrite(DC_dir, HIGH);
                   analogWrite(DC_pwm, 255-(abs(PidOutput)));
                }
                else {
                   digitalWrite(DC_dir, LOW);
                   analogWrite(DC_pwm, abs(PidOutput));
                }
            }
            break;
        }
    // IR and DC Servo
    case 1:
        {
            digitalWrite(DC_dir, LOW);
            analogWrite(DC_pwm, 0);
            if(mode == 0) {
                Servo_outVal = map(DCpos, 0, 255, 0, 180);
            }
            else {
                int IR_val=sharp.distance();
                if (IR_val >= IR_min && IR_val < IR_max)
                {
                    IR_out=IR_val;
                    //Serial.println(IR_out);
                }
                Servo_outVal=map(IR_out, IR_min, IR_max, 0, 180);
                //Serial.println(Servo_outVal);
                sensorVal = IR_out;
            }
            servo.write(Servo_outVal);
        }
        break;
    // Pot and Stepper control
    case 2:
        {

            if(mode == 0) {
                potVal = map(DCpos, 0, 255, 0, 1023);
```

```
                }
                else {
                    potVal = analogRead(potPin);

                    //  stepper.setSpeed(potVal/4);
                    //  stepper.runSpeed();
                    //  Serial.println(newPos);
                    //  Serial.println(stepper.currentPosition());
                    sensorVal = potVal;
                }
                newPos = potVal/5.12;
                stepper.moveTo(newPos);
                //  stepper.setSpeed(80);
                stepper.run();
            }
            break;
        default:
            break;
    }
    if(subsystem == 1) senseRate = 10;
    if(mode == 1 && countLoop%senseRate == 0)
    {
        sendToGui(subsystem, sensorVal);
        countLoop = 1;
    }
  countLoop++;
  delay(1);
}
```

GUI Side

dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QSerialPort>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private slots:
    void on_position_slider_valueChanged(int value);
    void on_velocity_slider_valueChanged(int value);
    void on_radioButton_DCUltrasonic_clicked();
    void on_radioButton_ServoIR_clicked();
    void on_radioButton_StepperPot_clicked();
    void on_radioButton_Gui_clicked();
    void on_radioButton_Sensor_clicked();
    void on_position_control_radio_clicked();
    void on_velocity_control_radio_clicked();
```

```
    void updateArduino(QString);
    void readSerial();

    void updateLCD_ultrasonic(const QString);
    void updateLCD_ir(const QString);
    void updateLCD_potentiometer(const QString);
    void on_position_slider_sliderReleased();
    void on_velocity_slider_sliderReleased();
    void on_radioButton_Sensor_pressed();
    void on_update_pushButton_clicked();

private:
    Ui::Dialog *ui;
    QSerialPort *arduino;
    static const quint16 arduino_uno_vendor_id = 9025;
    static const quint16 arduino_uno_product_id = 67;
    quint8 subsystem;
    quint8 input_method;
    quint8 control;
    QString arduino_port_name;
    bool arduino_is_available;
    QByteArray serialData;
    QString serialBuffer;
    QString sen_val;
    QString subsystem_value;
};

#endif // DIALOG_H
```

dialog.cpp

```
#include "dialog.h"
#include "ui_dialog.h"
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QtWidgets>

Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
    ui->ultrasonic_lcdNumber->display("-----");
    ui->ir_lcdNumber->display("-----");
    ui->potentiometer_lcdNumber->display("-----");

    arduino_is_available = false;
    arduino_port_name = "";

    arduino = new QSerialPort(this);
    serialBuffer = "";
    subsystem = 0;
    input_method = 0;
    control = 0;

    foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts()){
        if(serialPortInfo.hasVendorIdentifier() &&
serialPortInfo.hasProductIdentifier()){
```

```
            if(serialPortInfo.vendorIdentifier() == arduino_uno_vendor_id){
                if(serialPortInfo.productIdentifier() == arduino_uno_product_id){
                   arduino_port_name = serialPortInfo.portName();
                    arduino_is_available = true;
                }
            }
        }
    }

    if(arduino_is_available){
        //open and configure the serialport
        arduino->setPortName(arduino_port_name);
        qDebug() << arduino_port_name;
        if(!arduino->open(QIODevice::ReadWrite)) {
            qDebug() << "Could not open port!" << arduino->error();
         }
        arduino->setBaudRate(QSerialPort::Baud115200);
        arduino->setDataBits(QSerialPort::Data8);
        arduino->setParity(QSerialPort::NoParity);
        arduino->setStopBits(QSerialPort::OneStop);
        arduino->setFlowControl(QSerialPort::NoFlowControl);
        qDebug() << "Hi";
        qDebug() << arduino->isOpen();
        QObject::connect(arduino, SIGNAL(readyRead()), this, SLOT(readSerial()));
        Dialog::updateArduino(QString("%1").arg(0));


    }else{
        //give error message if not available
        QMessageBox::warning(this, "Port error", "Couldn't find the Arduino!");
    }

}

Dialog::~Dialog()
{
    if(arduino->isOpen()){
        arduino->close();
    }
    delete ui;
}

void Dialog::on_position_slider_valueChanged(int value)
{
    ui->position_slider_label->setText(QString("%1").arg(value));
}

void Dialog::on_velocity_slider_valueChanged(int value)
{
    ui->velocity_slider_label->setText(QString("%1").arg(value));
}

void Dialog::on_radioButton_DCUltrasonic_clicked()
{
    subsystem = 0;
    if(ui->radioButton_Gui->isChecked()){
        ui->position_slider_label->setEnabled(true);
        ui->velocity_slider_label->setEnabled(true);
        ui->position_slider->setEnabled(true);
        ui->velocity_slider->setEnabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->velocity_control_radio->setEnabled(true);
        ui->potentiometer_lcdNumber->setDisabled(true);
```

```
        ui->potentiometer_label->setDisabled(true);
        ui->ultrasonic_lcdNumber->setDisabled(true);
        ui->ultrasonic_label->setDisabled(true);
        ui->ir_lcdNumber->setDisabled(true);
        ui->IR_label->setDisabled(true);
        ui->CM_label->setDisabled(true);
        ui->Inch_label->setDisabled(true);
    }
    else if(ui->radioButton_Sensor->isChecked()){
        ui->position_slider_label->setDisabled(true);
        ui->velocity_slider_label->setDisabled(true);
        ui->position_slider->setDisabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setDisabled(true);
        ui->velocity_control_radio->setDisabled(true);
        ui->potentiometer_lcdNumber->setDisabled(true);
        ui->potentiometer_label->setDisabled(true);
        ui->ultrasonic_lcdNumber->setEnabled(true);
        ui->ultrasonic_label->setEnabled(true);
        ui->ir_lcdNumber->setDisabled(true);
        ui->IR_label->setDisabled(true);
        ui->CM_label->setDisabled(true);
        ui->Inch_label->setEnabled(true);
    }
}

void Dialog::on_radioButton_ServoIR_clicked()
{
    subsystem = 1;

    if(ui->radioButton_Gui->isChecked()){
        ui->position_slider_label->setEnabled(true);
        ui->velocity_slider_label->setDisabled(true);
        ui->position_slider->setEnabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->velocity_control_radio->setDisabled(true);
        ui->potentiometer_lcdNumber->setDisabled(true);
        ui->potentiometer_label->setDisabled(true);
        ui->ultrasonic_lcdNumber->setDisabled(true);
        ui->ultrasonic_label->setDisabled(true);
        ui->ir_lcdNumber->setDisabled(true);
        ui->IR_label->setDisabled(true);
        ui->CM_label->setDisabled(true);
        ui->Inch_label->setDisabled(true);
    }
    else if(ui->radioButton_Sensor->isChecked()){
        ui->position_slider_label->setDisabled(true);
        ui->velocity_slider_label->setDisabled(true);
        ui->position_slider->setDisabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setDisabled(true);
        ui->velocity_control_radio->setDisabled(true);
        ui->potentiometer_lcdNumber->setDisabled(true);
        ui->potentiometer_label->setDisabled(true);
        ui->ultrasonic_lcdNumber->setDisabled(true);
        ui->ultrasonic_label->setDisabled(true);
        ui->ir_lcdNumber->setEnabled(true);
        ui->IR_label->setEnabled(true);
        ui->CM_label->setEnabled(true);
        ui->Inch_label->setDisabled(true);
    }
```

```
}

void Dialog::on_radioButton_StepperPot_clicked()
{
    subsystem = 2;

    if(ui->radioButton_Gui->isChecked()){
        ui->position_slider_label->setEnabled(true);
        ui->velocity_slider_label->setDisabled(true);
        ui->position_slider->setDisabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->velocity_control_radio->setDisabled(true);
        ui->potentiometer_lcdNumber->setDisabled(true);
        ui->potentiometer_label->setDisabled(true);
        ui->ultrasonic_lcdNumber->setDisabled(true);
        ui->ultrasonic_label->setDisabled(true);
        ui->ir_lcdNumber->setDisabled(true);
        ui->IR_label->setDisabled(true);
        ui->CM_label->setDisabled(true);
        ui->Inch_label->setDisabled(true);
    }
    else if(ui->radioButton_Sensor->isChecked()){
        ui->position_slider_label->setDisabled(true);
        ui->velocity_slider_label->setDisabled(true);
        ui->position_slider->setDisabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setDisabled(true);
        ui->velocity_control_radio->setDisabled(true);
        ui->potentiometer_lcdNumber->setEnabled(true);
        ui->potentiometer_label->setEnabled(true);
        ui->ultrasonic_lcdNumber->setDisabled(true);
        ui->ultrasonic_label->setDisabled(true);
        ui->ir_lcdNumber->setDisabled(true);
        ui->IR_label->setDisabled(true);
        ui->CM_label->setDisabled(true);
        ui->Inch_label->setDisabled(true);
    }
}

void Dialog::on_radioButton_Gui_clicked()
{
    input_method = 0;
    ui->CM_label->setDisabled(true);
    ui->Inch_label->setDisabled(true);
    ui->position_slider->setEnabled(true);
    //ui->velocity_slider->setEnabled(true);
    ui->position_control_radio->setEnabled(true);
    //ui->velocity_control_radio->setEnabled(true);
    ui->potentiometer_lcdNumber->setDisabled(true);
    ui->potentiometer_label->setDisabled(true);
    ui->ultrasonic_lcdNumber->setDisabled(true);
    ui->ultrasonic_label->setDisabled(true);
```

```
    ui->ir_lcdNumber->setDisabled(true);
    ui->IR_label->setDisabled(true);
    if(ui->radioButton_DCUltrasonic->isChecked()){
        ui->velocity_control_radio->setEnabled(true);
        ui->velocity_slider->setEnabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->position_slider->setEnabled(true);
    }
    else if(ui->radioButton_ServoIR->isChecked()){
        ui->velocity_control_radio->setDisabled(true);
```

```
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->position_slider->setEnabled(true);

    }
    else if(ui->radioButton_StepperPot->isChecked()){
        ui->velocity_control_radio->setDisabled(true);
        ui->velocity_slider->setDisabled(true);
        ui->position_control_radio->setEnabled(true);
        ui->position_slider->setEnabled(true);
    }

}

void Dialog::on_radioButton_Sensor_clicked()
{

}

void Dialog::on_position_control_radio_clicked()
{
    control = 0;
    ui->position_slider->setEnabled(true);
    ui->velocity_slider->setDisabled(true);

}

void Dialog::on_velocity_control_radio_clicked()
{
    control = 1;
    ui->position_slider->setDisabled(true);
    ui->velocity_slider->setEnabled(true);

}

void Dialog::updateArduino(QString command)
{
    if(arduino->isWritable()){
        QString str;
        str.append(QString("%1,").arg(subsystem));
        str.append(QString("%1,").arg(input_method));
        str.append(QString("%1,").arg(control));
        //qDebug() << str;

        str.append(QString("%1\n").arg(command));
        qDebug() << str;
```

```
        arduino->write(str.toStdString().c_str());
    }else{
        qDebug() << "Couldn't write to serial";
    }
}

void Dialog::readSerial()
{
    serialData = arduino->readAll();
    serialBuffer += QString::fromStdString(serialData.toStdString());
    QStringList bufferSplit = serialBuffer.split("\r\n");
    if(bufferSplit.length() < 2){
        serialData = arduino->readAll();
         serialBuffer += QString::fromStdString(serialData.toStdString());

     }else{
```

```
        //bufferSplit(1) is a good value
        qDebug() << bufferSplit;
        QString tmp = bufferSplit[0];
        QStringList DataReceived = tmp.split(",");
        subsystem_value = DataReceived[0];
        sen_val = DataReceived[1];
        int valtmp;
        if(QString::compare(subsystem_value, "0") == 0 ||
QString::compare(subsystem_value, "1") == 0){
            valtmp = sen_val.toInt()/10;
        }
        else if(QString::compare(subsystem_value, "2") == 0){
            valtmp = ((sen_val.toInt()*360)/1023);

        }
        sen_val = QString::number(valtmp);
        qDebug() << sen_val << "subsystem" << subsystem ;
        //Dialog::updateLCD(bufferSplit[1]);
        serialBuffer = "";
        if(QString::compare(subsystem_value, "0") == 0){
            ui->ultrasonic_lcdNumber->setEnabled(true);
            ui->ultrasonic_label->setEnabled(true);
            ui->ir_lcdNumber->setDisabled(true);
            ui->IR_label->setDisabled(true);
            ui->potentiometer_lcdNumber->setDisabled(true);
            ui->potentiometer_label->setDisabled(true);
            //QString updated_sen_val
            //Dialog::updateLCD(QString("%1").arg(sen_val));
            Dialog::updateLCD_ultrasonic(sen_val);
            //Debug() << "In 0";

        }
        else if(QString::compare(subsystem_value, "1") == 0){
            ui->ir_lcdNumber->setEnabled(true);
            ui->IR_label->setEnabled(true);
            ui->potentiometer_lcdNumber->setDisabled(true);
            ui->potentiometer_label->setDisabled(true);
            ui->ultrasonic_lcdNumber->setDisabled(true);
            ui->ultrasonic_label->setDisabled(true);
             Dialog::updateLCD_ir(sen_val);
        }
        else if(QString::compare(subsystem_value, "2") == 0){
            ui->potentiometer_lcdNumber->setEnabled(true);
            ui->potentiometer_label->setEnabled(true);
            ui->ultrasonic_lcdNumber->setDisabled(true);
            ui->ultrasonic_label->setDisabled(true);
            ui->ir_lcdNumber->setDisabled(true);
            ui->IR_label->setDisabled(true);
             Dialog::updateLCD_potentiometer(sen_val);

        }

    }

}

void Dialog::updateLCD_ultrasonic(const QString sensor_reading)
{
    ui->ultrasonic_lcdNumber->display(sensor_reading);
}


void Dialog::updateLCD_ir(const QString sensor_reading)
```

```
{
    ui->ir_lcdNumber->display(sensor_reading);
}


void Dialog::updateLCD_potentiometer(const QString sensor_reading)
{
    ui->potentiometer_lcdNumber->display(sensor_reading);
}

void Dialog::on_position_slider_sliderReleased()
{
    int value = ui->position_slider->sliderPosition();
    Dialog::updateArduino(QString("%1").arg(value));
    //qDebug() << value;
}

void Dialog::on_velocity_slider_sliderReleased()
{
    int value = ui->velocity_slider->sliderPosition();
    Dialog::updateArduino(QString("%1").arg(value));

}

void Dialog::on_radioButton_Sensor_pressed()
{
    input_method = 1;
    ui->position_slider->setDisabled(true);
    ui->velocity_slider->setDisabled(true);
    ui->position_control_radio->setDisabled(true);
    ui->velocity_control_radio->setDisabled(true);
    ui->potentiometer_lcdNumber->setEnabled(true);
    ui->potentiometer_label->setEnabled(true);
    ui->ultrasonic_lcdNumber->setEnabled(true);
    ui->ultrasonic_label->setEnabled(true);
    ui->ir_lcdNumber->setEnabled(true);
    ui->IR_label->setEnabled(true);
    ui->CM_label->setEnabled(true);
    ui->Inch_label->setEnabled(true);
    ui->position_slider_label->setDisabled(true);
    ui->velocity_slider_label->setDisabled(true);

}

void Dialog::on_update_pushButton_clicked()
{
    Dialog::updateArduino(QString("%1").arg(0));
}
```

Main.cpp

```
#include "dialog.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.show();
    w.setWindowTitle("Sensor & Motor Lab GUI");
    return a.exec();
}
```

dialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1029</width>
    <height>396</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <widget class="QLabel" name="position_slider_label">
   <property name="geometry">
    <rect>
     <x>520</x>
     <y>230</y>
     <width>67</width>
     <height>17</height>
    </rect>
   </property>
   <property name="text">
    <string>0</string>
   </property>
  </widget>
  <widget class="QLabel" name="velocity_slider_label">
   <property name="geometry">
    <rect>
     <x>520</x>
     <y>270</y>
     <width>67</width>
     <height>17</height>
    </rect>
   </property>
   <property name="text">
    <string>0</string>
   </property>
  </widget>
  <widget class="QWidget" name="layoutWidget">
   <property name="geometry">
    <rect>
     <x>140</x>
     <y>40</y>
     <width>881</width>
     <height>71</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QRadioButton" name="radioButton_DCUltrasonic">
        <property name="text">
         <string>DC_Ultrasonic</string>
        </property>
```

```
        <property name="checked">
         <bool>true</bool>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">Sensor_Motor</string>
        </attribute>
       </widget>
      </item>
      <item>
       <widget class="QRadioButton" name="radioButton_ServoIR">
        <property name="text">
         <string>Servo_IR</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">Sensor_Motor</string>
        </attribute>
       </widget>
      </item>
      <item>
       <widget class="QRadioButton" name="radioButton_StepperPot">
        <property name="text">
         <string>Stepper_Potentiometer</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">Sensor_Motor</string>
        </attribute>
       </widget>
      </item>
     </layout>
    </item>
    <item>
     <layout class="QHBoxLayout" name="horizontalLayout_2">
      <item>
       <widget class="QRadioButton" name="radioButton_Gui">
        <property name="text">
         <string>GUI</string>
        </property>
        <property name="checked">
         <bool>true</bool>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">GUIorSensor</string>
        </attribute>
       </widget>
      </item>
      <item>
       <widget class="QRadioButton" name="radioButton_Sensor">
        <property name="text">
         <string>Sensor</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">GUIorSensor</string>
        </attribute>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <widget class="QWidget" name="layoutWidget">
   <property name="geometry">
    <rect>
     <x>320</x>
```

```
    <y>210</y>
    <width>191</width>
    <height>91</height>
   </rect>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_3">
   <item>
    <widget class="QSlider" name="position_slider">
     <property name="maximum">
      <number>255</number>
     </property>
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QSlider" name="velocity_slider">
     <property name="maximum">
      <number>255</number>
     </property>
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <widget class="QWidget" name="layoutWidget">
  <property name="geometry">
   <rect>
    <x>600</x>
    <y>210</y>
    <width>103</width>
    <height>101</height>
   </rect>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_4">
   <item>
    <widget class="QLabel" name="ultrasonic_label">
     <property name="text">
      <string>Ultrasonic</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QLabel" name="IR_label">
     <property name="text">
      <string>IR</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QLabel" name="potentiometer_label">
     <property name="text">
      <string>Potentiometer</string>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <widget class="QWidget" name="layoutWidget">
  <property name="geometry">
```

```
    <rect>
     <x>140</x>
     <y>210</y>
     <width>161</width>
     <height>91</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout_2">
    <item>
     <widget class="QRadioButton" name="position_control_radio">
      <property name="text">
       <string>Position Control</string>
      </property>
      <property name="checked">
       <bool>true</bool>
      </property>
      <attribute name="buttonGroup">
       <string notr="true">positioorvelocity</string>
      </attribute>
     </widget>
    </item>
    <item>
     <widget class="QRadioButton" name="velocity_control_radio">
      <property name="text">
       <string>Velocity Control</string>
      </property>
      <attribute name="buttonGroup">
       <string notr="true">positioorvelocity</string>
      </attribute>
     </widget>
    </item>
   </layout>
  </widget>
  <widget class="QWidget" name="layoutWidget">
   <property name="geometry">
    <rect>
     <x>700</x>
     <y>210</y>
     <width>111</width>
     <height>101</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout_5">
    <item>
     <widget class="QLCDNumber" name="ultrasonic_lcdNumber"/>
    </item>
    <item>
     <widget class="QLCDNumber" name="ir_lcdNumber"/>
    </item>
    <item>
     <widget class="QLCDNumber" name="potentiometer_lcdNumber"/>
    </item>
   </layout>
  </widget>
  <widget class="QPushButton" name="update_pushButton">
   <property name="geometry">
    <rect>
     <x>140</x>
     <y>160</y>
     <width>80</width>
     <height>23</height>
    </rect>
   </property>
```

```
  <property name="text">
   <string>Update</string>
  </property>
 </widget>
 <widget class="QWidget" name="layoutWidget">
  <property name="geometry">
   <rect>
    <x>810</x>
    <y>210</y>
    <width>81</width>
    <height>71</height>
   </rect>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_6">
   <item>
    <widget class="QLabel" name="Inch_label">
     <property name="text">
      <string>Inch</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QLabel" name="CM_label">
     <property name="text">
      <string>cm</string>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <widget class="QLabel" name="label">
  <property name="geometry">
   <rect>
    <x>40</x>
    <y>50</y>
    <width>91</width>
    <height>21</height>
   </rect>
  </property>
  <property name="text">
   <string>Subsystem</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_2">
  <property name="geometry">
   <rect>
    <x>40</x>
    <y>80</y>
    <width>91</width>
    <height>21</height>
   </rect>
  </property>
  <property name="text">
   <string>Control Method</string>
  </property>
 </widget>
 <zorder>layoutWidget</zorder>
 <zorder>layoutWidget</zorder>
 <zorder>layoutWidget</zorder>
 <zorder>layoutWidget</zorder>
 <zorder>layoutWidget</zorder>
 <zorder>layoutWidget</zorder>
```

```
  <zorder>position_slider_label</zorder>
  <zorder>velocity_slider_label</zorder>
  <zorder>update_pushButton</zorder>
  <zorder>label</zorder>
  <zorder>label_2</zorder>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
 <buttongroups>
  <buttongroup name="positioorvelocity"/>
  <buttongroup name="Sensor_Motor"/>
  <buttongroup name="GUIorSensor"/>
 </buttongroups>
</ui>
```