ILR-01| SENSORS AND MOTOR CONTROLS LAB

Submitted by: Mohak Bhardwaj

Andrew Id: mbhardwa

TEAM-I

Members: Shivam Gautam, Pranav Maheshwari, Dorothy Kirlew, Richa Varma

INDIVIDUAL PROGRESS

I undertook the following tasks for Task 7 - Sensors and Motors Control Lab:

1. Design and implement the Graphical User Interface (GUI)
2. Set up serial communication with the Arduino
3. Interface of sensors and motors code with the GUI.

GRAPHICAL USER INTERFACE

The goal for Task 7, as outlined in the assignment, was to build a system with two operating modes:

1. Manual control of actuators using a GUI.
2. Automatic control of actuators via sensor feedback.

After discussing the team objectives, following requirements were decided for the GUI:

1. Any sensor shall be able to control any actuator.
2. Only one sensor and actuator combination shall be active at a time.
3. In manual control mode, no sensor shall interfere with any actuator control.
4. Most importantly, the user shall be able to choose any control mode, actuator and sensor via an easy to use and reliable interface.

The GUI and serial interfacing were implemented using the Qt C++ framework. In order to realize the above requirements, it was essential to make efficient internal logic for the GUI in order to be able to operate in different, independent modes.

SALIENT FEATURES OF INTERFACE

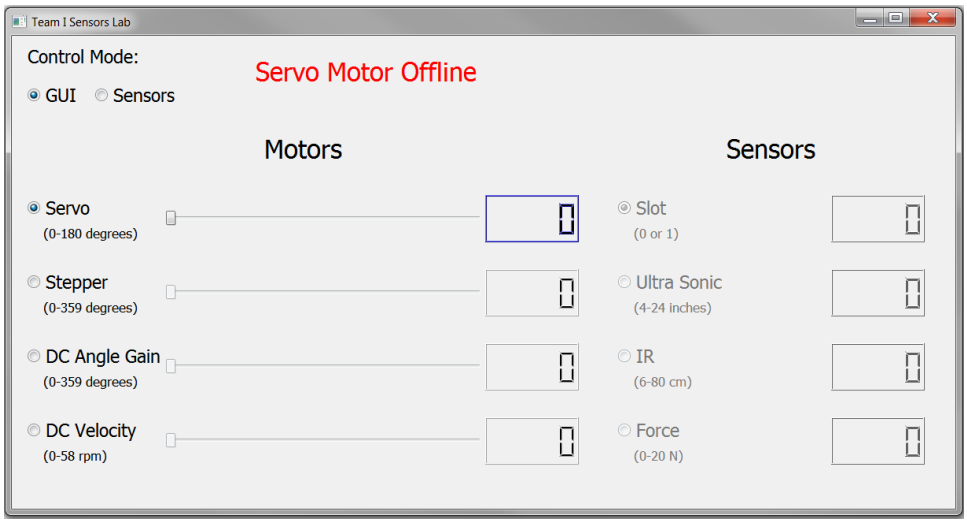

Fig 1: The GUI in its default setting with Servo Motor Offline

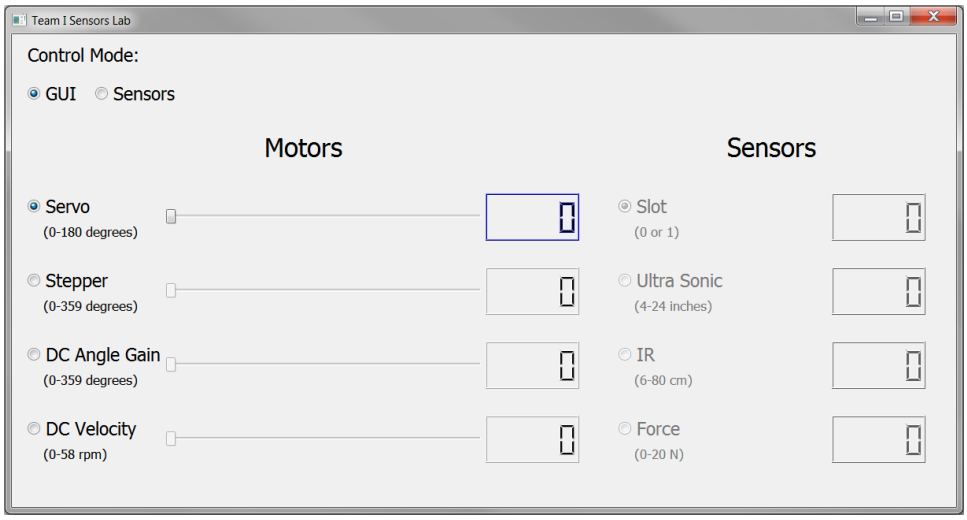

Fig 2: GUI in default setting with servo motor online

On the top-left of the GUI are the radio buttons for selecting the appropriate mode of operations i.e. manual control or sensor feedback. These options keep the two modes independent of each other.

The left pane of GUI contains Motor information, including radio buttons for selecting actuators, actuator control sliders, and LCD displays that show the current value of the actuator state. There are independent options for controlling the servo motor angle, stepper motor angle, and DC motor angle and velocity.

The right pane contains Sensor information, including radio buttons for selecting sensors and LDC displays that show the current value of the sensor. There are independent options for selecting the slot sensor, ultra-sonic sensor, infrared range finder and force sensor

The top right corner contains the connection status, which provides information regarding the connection to the Arduino via a serial COM port.

When the GUI is loaded, it defaults to the following options:

- Control Mode: Manual
- Servo Motor
- Slot Sensor (although this option is ignored while in Manual Control Mode)

The **'Servo Motor Offline'** message (see Fig. 1) is displayed when the switch connected the power line to the servo motor is in its off state.

The design of the GUI is minimalistic to keep it easy to use for all users. The motor control interface has sliders for sending appropriate position or velocity control commands to the Arduino. The motor angles vary from 0-180° for the servo motor, 0-359° for the DC motor (angle gain) and 0-359° for the stepper motor. The DC motor velocity has maximum value of 58rpm. The sensor interface displays appropriate values of the parameters measured by every sensor.

The GUI keeps track of three variables for internal logic: **activeMoter**, **activeSensor**, and **updateMode.** The variables are used to implement the feature that allows the GUI to disjoin the control tasks. When a particular set of controls are selected by the user, the values are updated so that other controls are appropriately disabled.

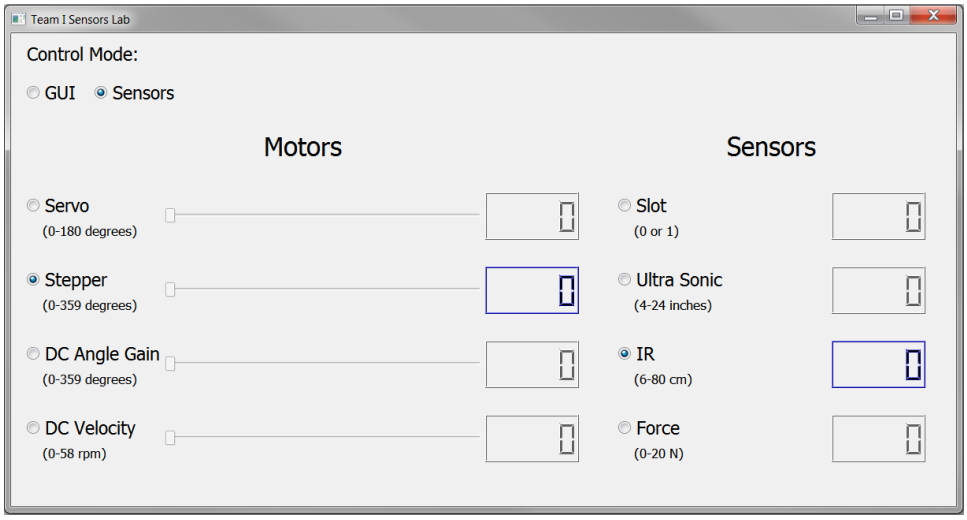

Fig 3: GUI in sensor control mode (Stepper motor via IR)

SERIAL INTERFACING

The serial interface with Arduino was implemented using the following Qt libraries:

- QtSerialPort/QSerialPort
- QtSerialPort/QSerialPortInfo
- QIODevice

1. ESTABLISHING A SERIAL CONNECTION:

a. The program searches all COM ports for Arduino and identifies a connected Arduino using its Vendor Id and Product Id.

b. On finding a connected Arduino, it is verified that the Arduino is indeed available for serial communication and in a writable state.

c. Depending on the above, a connection is established with the Arduino Uno over the appropriate COM port, using appropriate connection parameters:

Baud Rate: 9600
Parity: No Parity
Stop Bits: One
Data Bits: 8

2. PROTOCOL FOR SERIAL MESSAGES

Serial messages are exchanged between the Arduino and the GUI in the form of uniquely formatted strings. The following is the protocol for exchanging messages between the Arduino and the GUI:

Motor Control Messages:

- **Manual control Mode**

The standard motor control message is in the form: **C[0-3][0-1023]**, where **'C'** signifies a Control message, **0-3** is the number associated with the motor being controlled, and a value from **0-1023**, which is an 8-bit value used to control the appropriate motor. The motor control commands are only sent from the GUI to Arduino during Manual control mode. The reason for choosing a 0-1023 format and not actual values for each motor is that this allows for a common format for all motors and is easier for parsing the message. This message is sent from GUI to Arduino whenever appropriate options are selected.

- **Sensor Control Mode:**

During control of motors using sensor, the motor control message sent from the GUI to the Arduino is of the following the form: **S[0-3][0-3],** where **'S'** signifies a Sensor control, **0-3** is the number associated with the active motor and the second number **[0-3]** is associated with the active sensor. This message is only sent from the GUI to Arduino to identify the active components. The position values and signals to be sent to the motors are controlled by the Arduino internally.

Sensor Value Messages:

The standard sensor value message is of the following form: **S[0-3][0-1023]**, where **'S'** signifies a Sensor message, **0-3** is the number associated with the sensor being used for motor control and a value from **0-1023**, which is the value being returned by the active sensor. The value is mapped to the appropriate range of measurements by the GUI and then displayed. Though this format involves mapping the values twice, once on the Arduino end and once on the GUI end, it is preferred in order to keep things consistent.

3. COMMUNICATION

As per the state of the system specified by the GUI, the Arduino either sends sensor messages to the GUI or receives motor messages from the GUI. In order to do this, the Arduino relays sensor data continuously with a 100ms delay so that no messages are dropped. The main challenge was on the side of the GUI, where it is not possible to

freeze the main thread to wait for serial data. A Qt function called **ReadyRead()** in the QSerialPort library was used to avoid this. It acts as an interrupt and fires a signal every time data is present to be read from the serial port. The signal calls a slot function, which in our case is called **ReadSerial().** This function has been accepts the serial data values into a list which acts as a buffer to ensure that values are not dropped.

CHALLENGES

The major reason I chose to do the GUI was to learn something I had no experience with. The main challenges that I faced in the GUI development were:

- Having no prior background in GUI design and limited coding experience, I struggled initially with setting up the Qt environment and serial libraries on Ubuntu.
- Deciding on the GUI layout and logical flow of the program so that all different scenarios were taken care of while maintaining simplicity
- Setting up a serial connection with Arduino
- Exchanging messages with the Arduino reliably and at a fast rate.
- Making all elements of the system work cohesively

TEAMWORK

I had no prior experience with Qt or GUI design and limited coding experience, so Dorothy who has a lot of C++ experience helped me with the GUI design and implementation. A lot of focus was given to understanding the structure of a Qt program, serial communication and appropriate layout. I also worked with Pranav who had designed the overall structure of the Ardunio code. Together, we came up with the protocol for the string messages that would be passed between the GUI. Plenty of my effort was also directed in debugging serial communication related issues with Shivam and Dorothy. In the end, the entire team worked together to integrate the hardware and software systems to ensure that the system worked cohesively.

FUTURE PLANS

The team will be working on the tasks set in our Conceptual Design Review for the next Progress Review on 22nd October. I will be working actively on researching vision based techniques and other sensors for obstacle detection along with Pranav. I will also be working with Dorothy on researching appropriate tools for developing the smartphone interface and creating a Bluetooth connection between the smartphone and mobile platform.

CODE

File: main.cpp

Code:

```
#include "mainwindow.h"
#include <QApplication>
#include <QtSerialPort/QSerialPort>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    w.setWindowTitle("Team I Sensors Lab");
    w.setWindowIcon(QIcon("images.jpg"));

    w.setFixedSize(1200,600);

    return a.exec();
}
```

File: mainwindow.h

Code:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QtSerialPort/QSerialPort>
#include <QObject>
#include <QIODevice>
namespace Ui
{
    class MainWindow;
}
class MainWindow : public QMainWindow
```

```
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void readSerial();
    void parseInput(QString input);
    void updateSensor(int senseNum, int sensVal);
    void updateMotor(int motorNum, int motorVal);
    void enableMotor(int motorNum);
    void enableSensor(int sensorNum);
    void on_SlidersUpdate_clicked();
    void on_SensorUpdate_clicked();
    void on_ServoButton_clicked();
    void on_StepperButton_clicked();
    void on_DCPosButton_clicked();
    void on_DCVelButton_clicked();
    void on_ServoSlider_sliderReleased();
    void on_StepperSlider_sliderReleased();
    void on_DCPosSlider_sliderReleased();
    void on_DCVelSlider_sliderReleased();
    void on_SlotButton_clicked();
    void on_UltraButton_clicked();
    void on_IRButton_clicked();
    void on_ForceButton_clicked();
private:
    Ui::MainWindow *ui;
    QSerialPort *arduino;
    static const quint16 vendID = 9025;
    static const quint16 prodID = 67;
    QString arduinoPortName;
    bool arduinoAvailable;
    bool signalAvailable;
    bool arduinoWritable;
    QByteArray serialData;
    QString serialBuffer;
    int activeSensor;
    int activeMotor;
    int updateMode;
};
#endif // MAINWINDOW_H
```

File: mainwindow.cpp

Code:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
#include <QDebug>
#include <QtWidgets>
#include <QObject>
#include <QIODevice>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    arduino = new QSerialPort;
    arduinoPortName = "";
    arduinoAvailable = false;
    arduinoWritable=false;
    signalAvailable=false;
    serialBuffer = "";
    activeSensor = 0;
    activeMotor = 0;
    updateMode = 0;
    ui->ServoLabel->setPalette(Qt::blue);
    ui->SlotValue->setPalette(Qt::lightGray);
    ui->UltraValue->setPalette(Qt::lightGray);
    ui->IRValue->setPalette(Qt::lightGray);
    ui->ForceValue->setPalette(Qt::lightGray);

    // Check each COM port to find arduino
    foreach(const QSerialPortInfo &serialPortInfo, QSerialPortInfo::availablePorts())
    {
        if(serialPortInfo.hasVendorIdentifier() && serialPortInfo.hasProductIdentifier())
        {
            if(serialPortInfo.vendorIdentifier() == vendID)
            {
                if(serialPortInfo.productIdentifier() == prodID)
                {
                    arduinoPortName = serialPortInfo.portName() ;
                    arduinoAvailable = true;
                }
            }
        }
    }
    if(arduinoAvailable)
    {
```

```
        arduino->setPortName(arduinoPortName);
        arduino->open(QIODevice::ReadWrite);

        if(arduino->isWritable())
        {
            arduinoWritable=true;
            qDebug() << "Arduino is Writable";
        }
        else
        {
            qDebug() << "Arduino is NOT Writable";
        }
        arduino->setBaudRate(QSerialPort::Baud9600);
        arduino->setDataBits(QSerialPort::Data8);
        arduino->setFlowControl(QSerialPort::NoFlowControl);
        arduino->setParity(QSerialPort::NoParity);
        arduino->setStopBits(QSerialPort::OneStop);

        QObject::connect(arduino, SIGNAL(readyRead()), this, SLOT(readSerial()));

    }
    else if((arduinoAvailable) && (!arduinoWritable))
    {
        QMessageBox::warning(this, "Port Error", "Couldn't find the arduino port number!");
    }
}
MainWindow::~MainWindow()
{
    if(arduino->isOpen())
    {
        arduino->close();
    }
    delete ui;
}

// When user moves Servo slider, send message to arduino and update number on GUI
void MainWindow::on_ServoSlider_sliderReleased()
{
    int position = ui->ServoSlider->value();
    updateMotor(0, position);

    QString arduinoOutput = "C0" + QString("%1").arg(position);

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
    }
    else
```

```
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

// When user moves Stepper slider, send message to arduino and update number on GUI
void MainWindow::on_StepperSlider_sliderReleased()
{
    int position = ui->StepperSlider->value();
    updateMotor(1, position);

    QString arduinoOutput = "C1" + QString("%1").arg(position);

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
    }
    else
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

// When user moves DC Angle Position slider, send message to arduino and update number on GUI
void MainWindow::on_DCPosSlider_sliderReleased()
{
    int position = ui->DCPosSlider->value();
    updateMotor(2, position);

    QString arduinoOutput = "C2" + QString("%1").arg(position);

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
    }
    else
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

// When user moves DC Velocity slider, send message to arduino and update number on GUI
void MainWindow::on_DCVelSlider_sliderReleased()
{
    int position = ui->DCVelSlider->value();
    updateMotor(3, position);

    QString arduinoOutput = "C3" + QString("%1").arg(position);

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
```

```
    }
    else
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

// When something has been added to the serial port, read it
void MainWindow::readSerial()
{
    // arduino is seperating messages with commas
    QStringList bufferSplit = serialBuffer.split(",");

    // only read items if the length is less than three
    // this ensures that complete messages are being read
    if(bufferSplit.length() < 3)
    {
        serialData = arduino->readAll();
        serialBuffer += QString::fromStdString(serialData.toStdString());
    }
    else if (bufferSplit.length() >=3)
    {
        // print to debug what will be used
        // using second item in case first message sent is incomplete or in error
        qDebug() << bufferSplit[1];
        parseInput(bufferSplit[1]);
        serialBuffer = bufferSplit[1];

        // un-split serialBuffer, excluding the first item
        for(int i = 2; i < bufferSplit.length(); i++)
        {
            serialBuffer = serialBuffer + "," + bufferSplit[i];
        }
    }
}
// interprets input sent from arduino and responds accordingly
void MainWindow::parseInput(QString input)
{
    int sensVal;
    if(input.at(0) == "X") // servo motor off
    {
        // hide label
        ui->ArduinoPowerLabel->show();
    }
    else if(input.at(0) == "O") // servo motor on
    {
        // show label and send current sensor/motor combo to arduino if updating via sensors
        ui->ArduinoPowerLabel->hide();
        if(updateMode == 1)
        {
            QString arduinoOutput = "S" + QString("%1").arg(activeMotor)+ QString("%1").arg(activeSensor);
```

```
        if(arduino->isWritable())
        {
            arduino->write(arduinoOutput.toStdString().c_str());
            qDebug() << "Sending message to arduino: " << arduinoOutput;
        }
        else
            qDebug() << "Can't write following message to arduino: " << arduinoOutput;
    }
}
else if(input.at(0) == "S") // Slot
{
    // remove identifying char from string so only sensor output remains
    input.remove(0,1);
    // convert sensor output to int
    sensVal = input.toInt();
    // if updating motors via sensors and this is the correct active sensor, update GUI appropriately
    if(updateMode == 1 && activeSensor == 0)
    {
        updateSensor(0, sensVal);
        updateMotor(activeMotor, sensVal);
    }
}
else if(input.at(0) == "U") // Ultra
{
    // remove identifying char from string so only sensor output remains
    input.remove(0,1);
    // convert sensor output to int
    sensVal = input.toInt();
    // if updating motors via sensors and this is the correct active sensor, update GUI appropriately
    if(updateMode == 1 && activeSensor == 1)
    {
        updateSensor(1, sensVal);
        updateMotor(activeMotor, sensVal);
    }
}
else if(input.at(0) == "I") // IR
{
    // remove identifying char from string so only sensor output remains
    input.remove(0,1);
    // convert sensor output to int
    sensVal = input.toInt();
    // if updating motors via sensors and this is the correct active sensor, update GUI appropriately
    if(updateMode == 1 && activeSensor == 2)
    {
        updateSensor(2, sensVal);
        updateMotor(activeMotor, sensVal);
    }
}
```

```
    else if(input.at(0) == "T") // Force
    {
        // remove identifying char from string so only sensor output remains
        input.remove(0,1);
        // convert sensor output to int
        sensVal = input.toInt();
        // if updating motors via sensors and this is the correct active sensor, update GUI appropriately
        if(updateMode == 1 && activeSensor == 3)
        {
            updateSensor(3, sensVal);
            updateMotor(activeMotor, sensVal);
        }
    }
    else
    {
        qDebug() << "Serial message in incorrect format.";
    }
}
// changes sensor output to appropriate measurement and updates GUI
void MainWindow::updateSensor(int senseNum, int senseVal)
{
    qreal mappedVal;
    switch(senseNum)
    {
    case 0: // Slot sensor
        if(activeSensor == 0)
        {
            // Slot sensor is only on/off, so only display on/off
            if(senseVal > 0)
                ui->SlotValue->display(1);
            else
                ui->SlotValue->display(0);
        }
        break;
    case 1:// Ultrasonic
        if(activeSensor == 1)
        {
            // Converts value to 4-24 inches range
            mappedVal = senseVal/51.15+4;
            ui->UltraValue->display(qRound(mappedVal));
        }
        break;
    case 2:    // IR
        if(activeSensor == 2)
        {
            // Converts value to 6-80 cm range
            mappedVal = ((80.0-6)/1023)*senseVal+6;
            ui->IRValue->display(qRound(mappedVal));
        }
```

```
        break;
    case 3:    // Force
        if(activeSensor == 3)
        {
            // Converts value to 0-20 N range
            mappedVal = ((20.0)/1023)*senseVal;
            ui->ForceValue->display(qRound(mappedVal));
        }
        break;
    }
}

// changes motor output to appropriate measurement and updates GUI
void MainWindow::updateMotor(int motorNum, int motorVal)
{
    qreal mappedVal;
    switch(motorNum)
    {
    case 0: // Servo
        if(activeMotor == 0)
        {
            // converts value to 0-180 degree range
            mappedVal = ((180.0)/1023)*motorVal;
            ui->ServoLabel->display(qRound(mappedVal));
        }
        break;
    case 1: // Stepper
        if(activeMotor == 1)
        {
            // converts value to 0-359 degree range
            mappedVal = ((359.0)/1023)*motorVal;
            ui->StepperLabel->display(qRound(mappedVal));
        }
        break;
    case 2: // DC Pos
        if(activeMotor == 2)
        {
            // converts value to 0-359 degree range
            mappedVal = ((359.0)/1023)*motorVal;
            ui->DCPosLabel->display(qRound(mappedVal));
        }
        break;
    case 3: //DC Vel
        if(activeMotor == 3)
        {
            // converts value to 0-58 rpm range
            mappedVal = ((58.0)/1023)*motorVal;
            ui->DCVelLabel->display(qRound(mappedVal));
        }
```

```
    }
}

// if updating via GUI, disable all sensor information and enable selected motor
void MainWindow::on_SlidersUpdate_clicked()
{
    updateMode = 0;
    ui->SlotButton->setEnabled(false);
    ui->SlotUnits->setEnabled(false);
    ui->SlotValue->display(0);
    ui->SlotValue->setEnabled(false);
    ui->SlotValue->setPalette(Qt::lightGray);
    ui->UltraButton->setEnabled(false);
    ui->UltraUnits->setEnabled(false);
    ui->UltraValue->display(0);
    ui->UltraValue->setEnabled(false);
    ui->UltraValue->setPalette(Qt::lightGray);
    ui->IRButton->setEnabled(false);
    ui->IRUnits->setEnabled(false);
    ui->IRValue->display(0);
    ui->IRValue->setEnabled(false);
    ui->IRValue->setPalette(Qt::lightGray);
    ui->ForceButton->setEnabled(false);
    ui->ForceUnits->setEnabled(false);
    ui->ForceValue->display(0);
    ui->ForceValue->setEnabled(false);
    ui->ForceValue->setPalette(Qt::lightGray);

    enableMotor(activeMotor);
}

// if updating via sensors, disable motor sliders and enable selected sensor
void MainWindow::on_SensorUpdate_clicked()
{
    updateMode = 1;
    ui->SlotButton->setEnabled(true);
    ui->SlotUnits->setEnabled(true);
    ui->UltraButton->setEnabled(true);
    ui->UltraUnits->setEnabled(true);
    ui->IRButton->setEnabled(true);
    ui->IRUnits->setEnabled(true);
    ui->ForceButton->setEnabled(true);
    ui->ForceUnits->setEnabled(true);

    ui->StepperSlider->setEnabled(false);
    ui->ServoSlider->setEnabled(false);
    ui->DCPosSlider->setEnabled(false);

    ui->ServoLabel->display(0);
```

```
    ui->StepperLabel->display(0);
    ui->DCPosLabel->display(0);

    enableSensor(activeSensor);
}

// disable all motors but one selected
// send appropriate message to arduino
void MainWindow::enableMotor(int motorNum)
{
    ui->ServoSlider->setEnabled(false);
    ui->ServoSlider->setSliderPosition(0);
    ui->ServoLabel->setEnabled(false);
    ui->ServoLabel->display(0);
    ui->ServoLabel->setPalette(Qt::lightGray);

    ui->StepperSlider->setEnabled(false);
    ui->StepperSlider->setSliderPosition(0);
    ui->StepperLabel->setEnabled(false);
    ui->StepperLabel->display(0);
    ui->StepperLabel->setPalette(Qt::lightGray);

    ui->DCPosSlider->setEnabled(false);
    ui->DCPosSlider->setSliderPosition(0);
    ui->DCPosLabel->setEnabled(false);
    ui->DCPosLabel->display(0);
    ui->DCPosLabel->setPalette(Qt::lightGray);

    ui->DCVelSlider->setEnabled(false);
    ui->DCVelSlider->setSliderPosition(0);
    ui->DCVelLabel->setEnabled(false);
    ui->DCVelLabel->display(0);
    ui->DCVelLabel->setPalette(Qt::lightGray);

    switch(motorNum)
    {
    case 0: // Servo
        if(updateMode == 0)
            ui->ServoSlider->setEnabled(true);
        ui->ServoLabel->setEnabled(true);
        ui->ServoLabel->setPalette(Qt::blue);
        break;
    case 1: // Slider
        if(updateMode == 0)
            ui->StepperSlider->setEnabled(true);
        ui->StepperLabel->setEnabled(true);
        ui->StepperLabel->setPalette(Qt::blue);
        break;
    case 2: // DC Pos
```

```
        if(updateMode == 0)
            ui->DCPosSlider->setEnabled(true);
        ui->DCPosLabel->setEnabled(true);
        ui->DCPosLabel->setPalette(Qt::blue);
        break;
    case 3: // DC Vel
        if(updateMode == 0)
            ui->DCVelSlider->setEnabled(true);
        ui->DCVelLabel->setEnabled(true);
        ui->DCVelLabel->setPalette(Qt::blue);
        break;
    }

    QString arduinoOutput;
    if(updateMode == 0)
    {
        arduinoOutput = "C" + QString("%1").arg(activeMotor) + QString("0");
    }
    else
    {
        arduinoOutput = "S" + QString("%1").arg(activeMotor)+ QString("%1").arg(activeSensor);
    }

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
    }
    else
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

// disable all sensors but one selected
// send appropriate message to arduino
void MainWindow::enableSensor(int sensorNum)
{
    ui->SlotValue->setEnabled(false);
    ui->SlotValue->display(0);
    ui->SlotValue->setPalette(Qt::lightGray);
    ui->UltraValue->setEnabled(false);
    ui->UltraValue->display(0);
    ui->UltraValue->setPalette(Qt::lightGray);
    ui->IRValue->setEnabled(false);
    ui->IRValue->display(0);
    ui->IRValue->setPalette(Qt::lightGray);
    ui->ForceValue->setEnabled(false);
    ui->ForceValue->display(0);
    ui->ForceValue->setPalette(Qt::lightGray);
```

```
    QString arduinoOutput = "S" + QString("%1").arg(activeMotor);

    switch(sensorNum)
    {
    case 0: // Slot
        ui->SlotValue->setEnabled(true);
        ui->SlotValue->setPalette(Qt::blue);
        arduinoOutput = arduinoOutput + QString("0");
        break;
    case 1: // Ultra
        ui->UltraValue->setEnabled(true);
        ui->UltraValue->setPalette(Qt::blue);
        arduinoOutput = arduinoOutput + QString("1");
        break;
    case 2: // IR
        ui->IRValue->setEnabled(true);
        ui->IRValue->setPalette(Qt::blue);
        arduinoOutput = arduinoOutput + QString("2");
        break;
    case 3: // Force
        ui->ForceValue->setEnabled(true);
        ui->ForceValue->setPalette(Qt::blue);
        arduinoOutput = arduinoOutput + QString("3");
        break;
    }

    if(arduino->isWritable())
    {
        arduino->write(arduinoOutput.toStdString().c_str());
        qDebug() << "Sending message to arduino: " << arduinoOutput;
    }
    else
        qDebug() << "Can't write following message to arduino: " << arduinoOutput;
}

void MainWindow::on_ServoButton_clicked()
{
    activeMotor = 0;
    enableMotor(0);
}

void MainWindow::on_StepperButton_clicked()
{
    activeMotor = 1;
    enableMotor(1);
}

void MainWindow::on_DCPosButton_clicked()
{
```

```
    activeMotor = 2;
    enableMotor(2);
}

void MainWindow::on_DCVelButton_clicked()
{
    activeMotor = 3;
    enableMotor(3);
}
void MainWindow::on_SlotButton_clicked()
{
    activeSensor = 0;
    enableSensor(0);
}

void MainWindow::on_UltraButton_clicked()
{
    activeSensor = 1;
    enableSensor(1);
}

void MainWindow::on_IRButton_clicked()
{
    activeSensor = 2;
    enableSensor(2);
}

void MainWindow::on_ForceButton_clicked()
{
    activeSensor = 3;
    enableSensor(3);
}
```