Individual Lab Report 2

Guillermo Manuel Cidre

Team G – Bob's builders

Teammates: Eric Newhall, Michael O’Connor, Christian Heaney-Secord

IRL02

2/12/14

Individual Work:

In the sensor lab, I was assigned to make a fancy GUI program. The GUI was supposed to read from the serial connection to the arduino, be able to write back data, and be able to overide the sensor inputs with inputs given on the GUI. In addition after implementing it, I was supposed to modify the code of the arduino so that the arduino can understand whether the inputs to overide the sensor values were given. This, in some sense, attributed to a large chunk of the lab itself. In addition, I ran to many bugs and issues. So ultimately, this is my one and only contribution for this week. The picture of the GUI along with the code is displayed below on Figure 1 and Figure 2 respectively.

Challenges / Issues:

For the sensor lab, there were two main challenges/issues that I faced. The first issue was the lack of help on using Qt. I had prior experience in making GUI programs, and Qt programing turned out to be very similar to the programming I'm accustomed to. But even though many of the mechanics where the same, they used different function names for their stuff. In addition, Qt 5.2 have major changes to its programming structure compared to older versions. So what ended up happening was that everything that I looked up online ended up being outdated and not applicable. One such example was that installation instructions online asked the user to install Visual Studios and compile it manually. Installing Visual Studios takes up to 3 hours of installation and it turned out that one could just download the binary directly. Also, there are many websites that claim that Qt does not have serial support and a user would have to install a seperate library while it in fact does. Another example of this issue can be seen from the help docs provided by the Qt community. The help docs only tell what the functions are supposed to do but never show how to implement it with example. So basically, I would just write a use of the function and get error "Invalid reference: __imp_S3E1RI9L..." These problems sadly took me 9 hours to work around it, but it is only a temporary problem that may be fixed once all the tutorial/documents catches up with the current version. But the good news was that this GUI program can also interface with many other serial connections as well.

Another challenge was modifying the arduino code so that the messages could be received from GUI. It was simple writing the code. The problem was where we would have to write the code. The encoder implementation consisted of triggers and for loops that would needed to be modified all together. While looking through the code, I found that one of the pin that would be needed to control the velocity of one of our motors was in fact outputed as a digital output instead of an analog output. In our case, this meant that we would have to rewire a major part of the circuit we assembled. We were really pressed for time at the time. So I didn't actually fix that problem so that I could deal with other problems.

Team Work:

Eric Newhal and I mainly helped create the lab. Eric assembled and connected the sharp sensor, the potentiameter, the flex sensor, the servo motor, the stepper motor, and the DC motor into the circuit for the motor lab. Eric also implemented the debouncing. Michael and Christian helped in some of the wiring and probably in some of the coding for the motors/sensors. And Eric helped me see some of the problems our circuit board was emiting near its final stages when implementing the GUI code unto the arduino.

Eric, Michael, and Christian helped precut some of the components made in the mockup in order to be able to build the correct framing for our final project. In addition, we got 24 metal hinges for support for the railing in the project. Michael and Christian familiarizing themselves witht the 3D printer so we could do the hopper prototyping

Future Plans:

We didn't get much of my previous future plans. So we have the same goals as last week. Such plans include assessing the strength of the electromagnets, prototyping different hopper models for best result, and ultimately assemble part of the arms of the project for the flux dispenser and place orientator. Since there are no more extra labs for the end of this semester, I plan to devote more time unto the project itself and the website. This include but not limited to sequence diagrams and updating pictures/info.

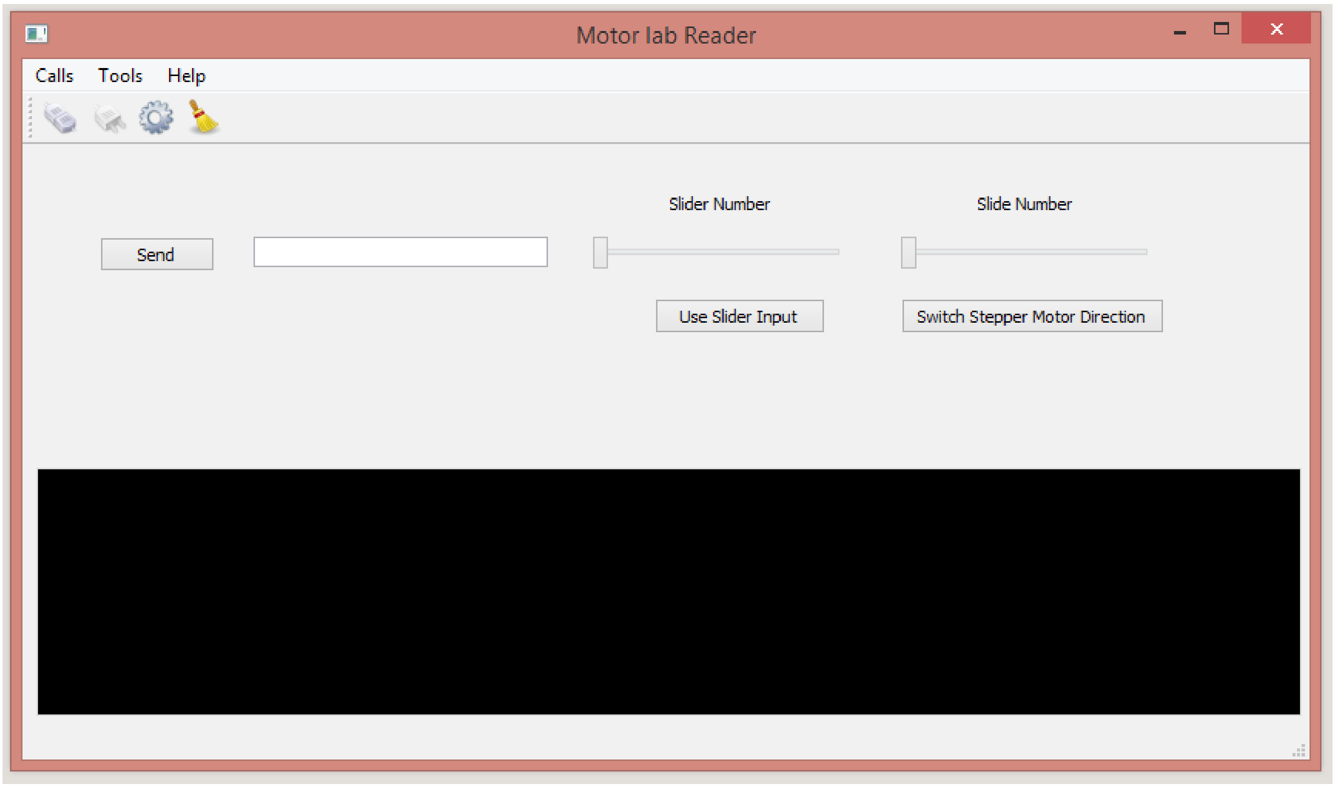

Figure 1: The GUI program

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "console.h"
#include "settingsdialog.h"

#include <QMessageBox>
#include <QtSerialPort/QSerialPort>

//! [0]
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
//! [0]
    ui->setupUi(this);
    console = new Console;
    console->setEnabled(false);
    //setCentralWidget(console);
    ui->widget->addWidget(console);
    ui->widget->setCurrentWidget(console);

//! [1]
    serial = new QSerialPort(this);
//! [1]
    settings = new SettingsDialog;

    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionQuit->setEnabled(true);
    ui->actionConfigure->setEnabled(true);

    initActionsConnections();

    connect(serial, SIGNAL(error(QSerialPort::SerialPortError)), this,
            SLOT(handleError(QSerialPort::SerialPortError)));

//! [2]
    connect(serial, SIGNAL(readyRead()), this, SLOT(readData()));
//! [2]
    connect(console, SIGNAL(getData(QByteArray)), this,
SLOT(writeData(QByteArray)));
//! [3]
}
//! [3]

MainWindow::~MainWindow()
{
    delete settings;
    delete ui;
}
//! [4]
void MainWindow::openSerialPort()
{
    SettingsDialog::Settings p = settings->settings();
    serial->setPortName(p.name);
    serial->setBaudRate(p.baudRate);
    serial->setDataBits(p.dataBits);
    serial->setParity(p.parity);
    serial->setStopBits(p.stopBits);
    serial->setFlowControl(p.flowControl);
    if (serial->open(QIODevice::ReadWrite)) {
            console->setEnabled(true);
            console->setLocalEchoEnabled(p.localEchoEnabled);
            ui->actionConnect->setEnabled(false);
            ui->actionDisconnect->setEnabled(true);
            ui->actionConfigure->setEnabled(false);
            ui->statusBar->showMessage(tr("Connected to %1 : %2, %3, %4, %5, %6")

.arg(p.name).arg(p.stringBaudRate).arg(p.stringDataBits)

.arg(p.stringParity).arg(p.stringStopBits).arg(p.stringFlowControl));
    } else {
        QMessageBox::critical(this, tr("Error"), serial->errorString());

        ui->statusBar->showMessage(tr("Open error"));
    }
}
//! [4]

//! [5]
void MainWindow::closeSerialPort()
{
    serial->close();
    console->setEnabled(false);
    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionConfigure->setEnabled(true);
    ui->statusBar->showMessage(tr("Disconnected"));
}
//! [5]

void MainWindow::about()
{
    QMessageBox::about(this, tr("About Simple Terminal"),
                       tr("The <b>Simple Terminal</b> example demonstrates how to "
                          "use the Qt Serial Port module in modern GUI applications
"
                          "using Qt, with a menu bar, toolbars, and a status
bar."));
}

//! [6]
void MainWindow::writeData(const QByteArray &data)
{
    serial->write(data);
}
//! [6]

//! [7]
void MainWindow::readData()
{
    QByteArray data = serial->readAll();
    console->putData(data);
}
//! [7]
//! [8]
void MainWindow::handleError(QSerialPort::SerialPortError error)
{
    if (error == QSerialPort::ResourceError) {
        QMessageBox::critical(this, tr("Critical Error"), serial->errorString());
        closeSerialPort();
    }
}
//! [8]

void MainWindow::initActionsConnections()
{
    connect(ui->actionConnect, SIGNAL(triggered()), this, SLOT(openSerialPort()));
    connect(ui->actionDisconnect, SIGNAL(triggered()), this,
SLOT(closeSerialPort()));
    connect(ui->actionQuit, SIGNAL(triggered()), this, SLOT(close()));
    connect(ui->actionConfigure, SIGNAL(triggered()), settings, SLOT(show()));
    connect(ui->actionClear, SIGNAL(triggered()), console, SLOT(clear()));
    connect(ui->actionAbout, SIGNAL(triggered()), this, SLOT(about()));
    connect(ui->actionAboutQt, SIGNAL(triggered()), qApp, SLOT(aboutQt()));
}

void MainWindow::on_pushButton_clicked()
{
    QByteArray mess;
    mess.insert(0,ui->lineEdit->text());
    writeData(mess);
}

void MainWindow::on_horizontalSlider_sliderMoved(int position)
{
    ui->label->setText(QString::number(position));
}

void MainWindow::on_pushButton_2_clicked()
{
    static bool state = false;
    QByteArray A = QByteArray("a");
    if(state)
    {
        ui->pushButton_2->setText("Use Slider Input");
        ui->horizontalSlider->setEnabled(false);
        ui->horizontalSlider_2->setEnabled(false);
        state = false;
        A[0] = 100;
    }
    else
    {
        ui->pushButton_2->setText("Use Sensor Input");
        ui->horizontalSlider->setEnabled(true);
        ui->horizontalSlider_2->setEnabled(true);
        state = true;
        A[0] = 101;
    }

    writeData(A);
    //ui->lineEdit->setText(QString::number(A[0]));
}
```

```
void MainWindow::on_horizontalSlider_2_sliderMoved(int position)
{
    ui->label_2->setText(QString::number(position));
}

void MainWindow::on_horizontalSlider_sliderReleased()
{
    //update primary variable
    QByteArray A = QByteArray("ab");
    A[0] = 30;
    A[1] = ui->horizontalSlider->value();
    writeData(A);
    //ui->lineEdit->setText(QString::number(A[1]));
}

void MainWindow::on_pushButton_3_clicked()
{
    //switch servo direction(Not actually needed)

}

void MainWindow::on_horizontalSlider_2_sliderReleased()
{
    //update secondary variable
    QByteArray A = QByteArray("ab");
    A[0] = 80;
    A[1] = ui->horizontalSlider_2->value();
    writeData(A);
    //ui->lineEdit->setText(QString::number(A[1]));
}
```

Figure 2: mainWindow.cpp code which is the main code for this program.
The rest is generated from their designer program and example codes