Individual Lab Report - 1

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Gunjan Sethi

Team C:

Kaushik Balasundar | Parker Hill | Anthony Kyu
Sundaram Seivur | Gunjan Sethi

February 10 2022

Carnegie Mellon University
The Robotics Institute

Contents

1 Individual Progress **1**
- 1.1 Sensor and Motors Lab . . . 1
  - 1.1.1 Graphical User Interface . . . 1
- 1.2 MRSD Project . . . 3
  - 1.2.1 Camera/ROS Integration . . . 3
  - 1.2.2 Research on Registration Algorithms . . . 3

2 Challenges **3**
- 2.1 Sensors and Motors Lab . . . 3
- 2.2 MRSD Project . . . 4

3 Team Work **4**
- 3.1 Sensors and Motors Lab . . . 4
- 3.2 MRSD Project . . . 5

4 Plans **6**
- 4.1 Sensors and Motors Lab . . . 6
- 4.2 MRSD Project . . . 6

5 Sensors and Motors Lab Quiz **7**
- 5.1 Reading a Datasheet . . . 7
- 5.2 Signal Conditioning . . . 7
- 5.3 Control . . . 8
- 5.4 OpAmp- Handwritten Solution . . . 9

6 Code **11**

1 Individual Progress

1.1 Sensor and Motors Lab

1.1.1 Graphical User Interface

The graphical user interface (GUI) is a standalone application that runs on a laptop. It interfaces with the embedded controller- Arduino, in this case via the serial port. It dynamically reports the state of motors and sensor inputs using plots and virtual LCD displays. Further, if enabled, the GUI allows manipulating the states of motors via a suite of UI components like sliders, radio button and progress bars. For debugging, one can use the terminal feature that sends serial commands to the Arduino.

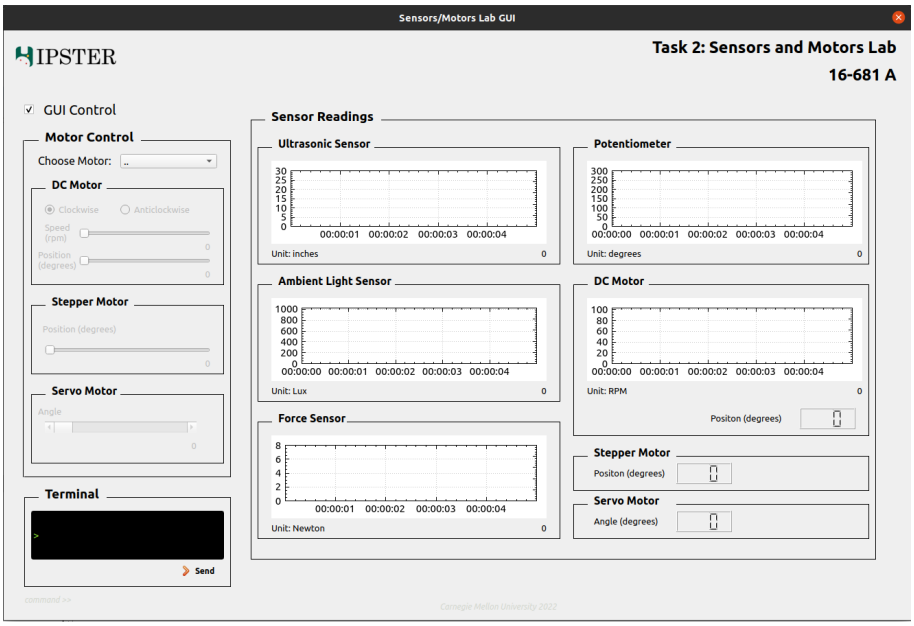

Figure 1: Graphical User Interface

Specifications – The GUI has been developed using ***Qt 6.2.0***, a free, open-source, cross-platform toolkit in ***C++***. ***Qt Creator 6.0.2*** was used as the primary integrated development environment (IDE). The GUI designer of Qt Creator is very easy to learn and use, with an abundance of support from online forums and documentation. Finally, the application was developed on ***Ubuntu 20.04***.

Features –

- ***GUI Control Toggle*** The GUI allows for the user to disable GUI control via a checkbox. GUI Control is enabled by default. When disabled, the sensor values begin to control the motors based on the logic written on the Arduino.
- ***Motor Control*** This section on the GUI can control the three motors asynchronously. Through a drop-down menu, the user can pick the motor to be controlled. Once selected, the corresponding subsection is enabled. For the DC Motor, the user can control velocity, position

and direction of rotation. For stepper motor, users can control the position and for servo motor, the angle can be controlled.

- ***Terminal*** The terminal section emulates a serial monitor. It allows users to type and hit return to serially write commands to the Arduino. It can be used for debugging and testing.

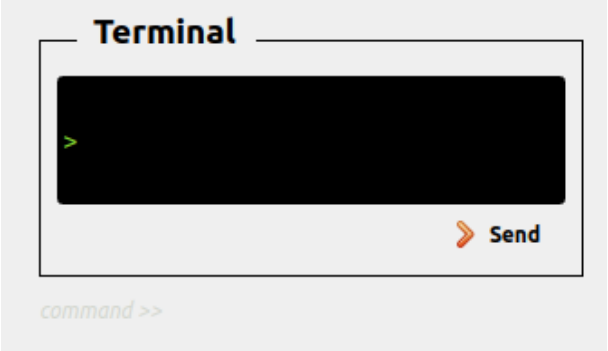

Figure 2: Terminal Section: GUI

- ***Sensor Readings*** The sensor readings section dynamically displays incoming sensor values at all times. Any sensor data processing is performed on the Arduino before it is serially received at the GUI end. The stepper motor and servo motor angles are displayed using an LCD label. The sensor readings are plotted using QCustomPlot. QCustomPlot is a Qt C++ widget for plotting and data visualization. It has no further dependencies and is well documented. This plotting library focuses on making good-looking, high-quality 2D plots, graphs and charts, as well as offering high performance for real-time visualization applications.

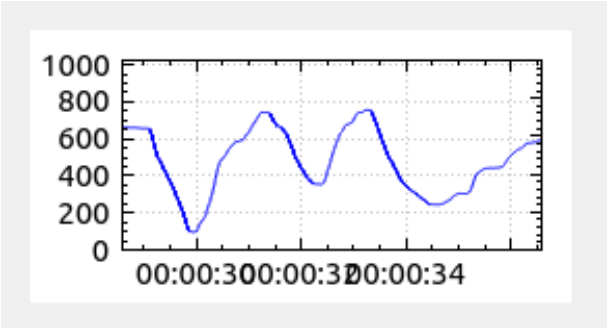

Figure 3: Sample of Plot using QCustomPlot

Integration With Hardware – The Arduino is expected to send a string that was agreed upon by all team members. This string is parsed on the GUI end to decipher sensor and motor readings. Similarly, the GUI sends a command string in a pre-defined format that is parsed by the Arduino to send signals to connected actuators (motors). Following are the commands expected to be received from the Arduino.

```
u,<ultrasonic_sensor_reading>

f,<force_sensor_reading>
```

```
p,<potentiometer_reading>

a,<ambient_light_sensor_reading>

u,<ultrasonic_sensor_reading>

m,<motor_number>,<motor_value>
```

Following is the command expected at the Arduino's end. "1" indicates that the motor is controlled by the GUI.

```
1,<motor_num>,<value>
```

Documentation The process and setup has been documented in detail here.

1.2 MRSD Project

1.2.1 Camera/ROS Integration

The majority of efforts so far this semester have been steered towards the integration of the Atracys Sprytrack camera with Robot Operating System (ROS). The camera ships with an SDK that contains functions to get fiducial marker poses given its geometry. The goal is to externally link the Atracys SDK with ROS using CMake in order to import the relevant libraries and acquire the marker poses. After the linking is successful, a topic is created and marker poses from the Atracys SDK are typecasted to the ROS standard messages type and published. For this, a deep understanding of the CMake system is required. Therefore, a considerable amount of time was spent watching and learning CMake from online tutorials. Further, attempts to link the SDK were made. As a proof of concept, a basic ROS package was created and manipulating the CMakeLists.txt to link the SDK was attempted. The screenshot attached below shows that a ROS node can now discover or connect with the Sprytrack camera.

Documentation The process and setup has been documented in detail here.

1.2.2 Research on Registration Algorithms

The perception module performs pointcloud registration on the pelvis model given the landmark fiducial markers. To perform the registration process, an evaluation of the present, state-of-the-art registration algorithms is required. Currently, only brief research on these algorithms has been performed. Documentation and proof-of-concept testing will be performed in the upcoming weeks.

2 Challenges

2.1 Sensors and Motors Lab

One major challenge in the Sensors and Motors labs was integrating the QCustomPlot library due to version differences. Downloading a patch from the support forum fixed the issue.

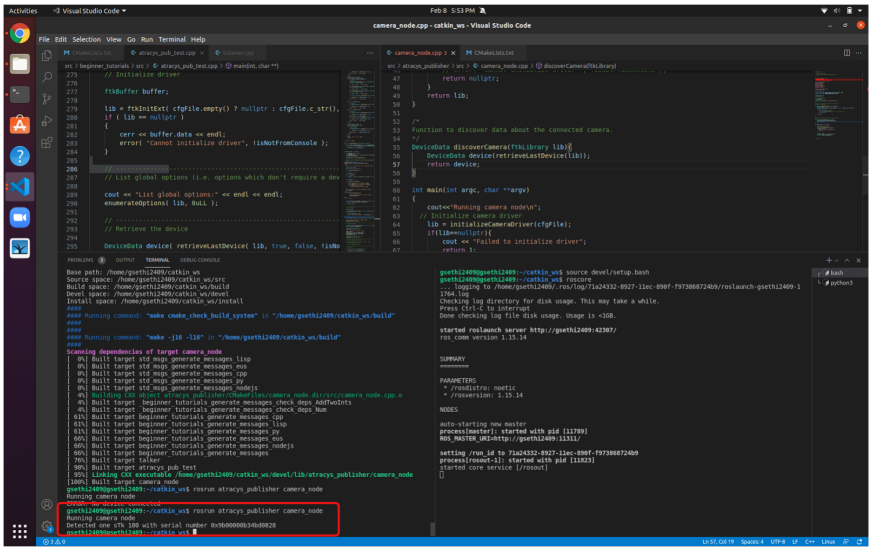

Figure 4: ROS Node Connects with Camera

2.2 MRSD Project

The biggest challenge was to understand how ROS links external libraries. Most common libraries have ROS packages that can be integrated with a custom ROS package using rosdep. However, the Atracys Sprytrack camera is only shipped with its SDK. Due to the unavailability of a ROS package, an external library linking with a ROS package can be challenging. To tackle the issues, it is important to understand the nitty-gritty of CMake and exactly how it links libraries.

Documentation The errors and bugs have been documented in detail here.

3 Team Work

3.1 Sensors and Motors Lab

Whilst working on the graphical user interface for the Sensors and Motors Lab, there was active collaboration between teammates. The initial GUI wire-frame was developed after discussion with the team.

Since the GUI integrates with each component of the circuit, it was required to spend a considerable amount of time with each teammate to understand the incoming sensor data and the respective trigger commands. In collaboration with Kaushik Balasundar, the GUI was interfaced, debugged and tested for the DC Motor position control and potentiometer plots. The ultrasonic sensor plots and stepper motor controls were interfaced, debugged and tested in collaboration with Parker Hill. The state machine, force sensor resistor (FSR) and servo motor were interfaced, debugged and tested in collaboration with Anthony Kyu. Finally, in collaboration with Sundaram Seivur, the DC Motor velocity control and ambient light sensor were interfaced, debugged and tested.

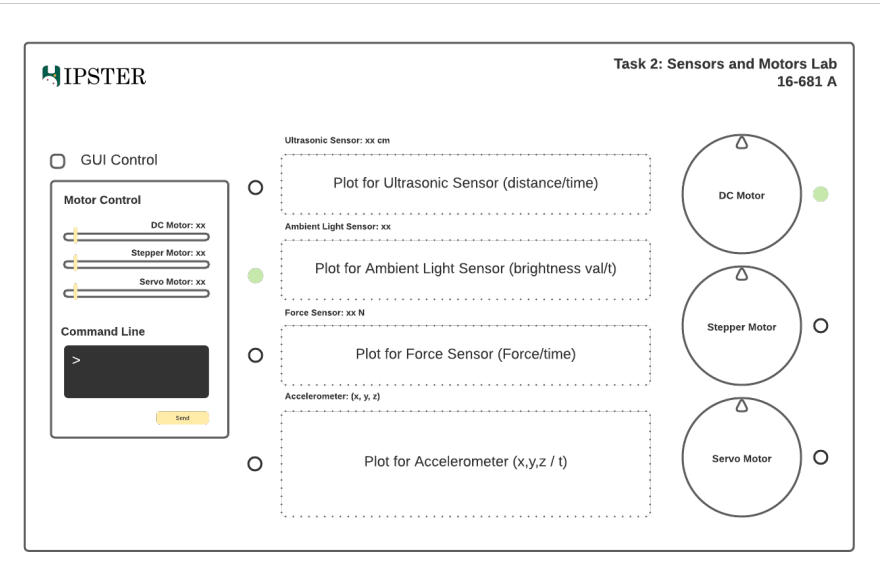

Figure 5: Initial Wireframe- Graphical User Interface

Following was the division of responsibilities between team members.

- ***Kaushik Balasundar*** DC Motor Position Control, Potentiometer, System Integration.
- ***Parker Hill*** Stepper Motor Integration, Ultrasonic Force Sensor Integration, Hardware/System Integration
- ***Anthony Kyu*** Servo Motor Control, Force Sensitive Resistor (Transfer Function Measurements, Analog Filtering), State Machine, Communication with GUI, System Integration
- ***Sundaram Seivur*** DC Motor Velocity Control, Ambient Light Sensor, Hardware/System Integration
- ***Gunjan Sethi*** GUI Development, Hardware/Software Integration

3.2 MRSD Project

Tasks in the last few weeks involved extensive collaboration with Kaushik Balasundar in perception and software engineering. Major collaborative tasks included in-depth discussions about the perception sub-tasks breakdown, eliciting strategies to tackle issues in ROS/Camera integration and trade studies on various pointcloud registration algorithms. Along with the entire team, an onboarding session was conducted by Kinnova that involved understanding the initial setup process and visual interface of the robot arm.

Following was the division of responsibilities between team members.

- ***Kaushik Balasundar*** ROS Docker Setup, Code Repository Setup, Cleaning-up URDF and Simulation Setup, Open3D ROS Integration, Basic moveit Package Configuration
- ***Parker Hill*** Camera Mount Fabrication, Researching/Designing/Collecting Marker Mounts, Picking up Robot Arm, Learning ROS

- ***Anthony Kyu*** MRSD Website, Camera Mount Design and Fabrication, Controls Research and Selection
- ***Sundaram Seivur*** Camera Mount Fabrication, Controller Integration in Simulation, moveit Integration
- ***Gunjan Sethi*** Atracys SDK Setup/Running Sample Scripts, Gaining Deeper Understanding of CMake, Linked Atracys SDK with ROS, Developed Proof-of-Concept ROS Package to Discover Camera through ROS Node, Documentation, Jira Management

4 Plans

4.1 Sensors and Motors Lab

For future work, the following tasks (individual) have been planned.

- ***Software Engineering Practices*** The current codebase can certainly benefit from some sophisticated software engineering processes to boost modularity and maintainability.
- ***UI/UX Revisions*** The current UI does not incorporate colors and graphics. To improve the design, common UI/UX reference materials will be studied and implemented.
- ***Export as Windows Application*** To leverage the cross-platform nature of Qt, a Windows application will be build and deployed.

4.2 MRSD Project

For future work, the following tasks (individual) have been planned.

- ***Camera/ROS Integration Debugging*** The task involves performing more extensive tests in linking the SDK. Further guidance on the task will be attained from discussions with project sponsors.
- ***Publish Fiducial Marker Poses*** Create a ROS Node that acquires fiducial marker poses with the Atracys SDK that typecastes the poses and publishes onto a topic.
- ***Registration Algorithms: Mini Proof-of-Concepts*** The task involves creating proof of concepts for the chosen registration algorithms and analyzing their performance. This performance will be documented. The final goal is to select one algorithm that works best with sufficient number of landmark points. This task will be performed jointly with Kaushik Balasundar.

5 Sensors and Motors Lab Quiz

5.1 Reading a Datasheet

1. The product measures acceleration with a minimum full-scale range of ± 3g.
2. The product measures acceleration dynamic range is 6g.
3. For most applications, a single 0.1 F capacitor, C_{DC}, placed close to the ADXL335 supply pins adequately decouples the accelerometer from noise on the power supply. A decoupling capacitor acts as a local electrical energy reservoir. Capacitors, like batteries, need time to charge and discharge. When used as decoupling capacitors, they oppose quick changes of voltage. If the input voltage suddenly drops, the capacitor provides the energy to keep the voltage stable. Similarly, if there is a voltage spike, the capacitor absorbs the excess energy.
4. V_{out} = 1.5V + (300 mV/g) * a where a = acceleration.
5. 7.2g * 0.3/100 = 0.0216 g
6. 0.5 Hz to 1600 Hz for the X and Y axes.
7. 150 * sqrt(25) = 750 (mu)g rms
8. Start by placing the sensor in an environment with ideal power supply, no noise and constant room temperature. This sort of static environment will ensure that sensor readings are isolated from any other external factors. The sensor must be placed on a flat surface due cancel out any xyz accelerations. Then, record several readings for a sufficient amount time. Finally, use the RMS equation to estimate the RMS noise.

5.2 Signal Conditioning

1. Filtering
    - **Moving Average Filter** – First, the moving average filter has a delayed response of the true signal value. It does not reflect changes in sensor measurement instantly. This latency is proportional to the size of the filter window. Second, it results in distortion in the frequency domain encoded signals. This is because they are incapable of separating out frequency bands.
    - **Median Filter** – First, the implementation of the median filter has a high computational complexity. This is because sorting of measurements is often required for median calculation. Second, similar to the moving average filter, it has a high latency and therefore does not instantly reflect changes in the sensor measurement.
2. Opamps
    - **Range of -1.5 to 1.0V** – Handwritten solution attached in Section 5.4. Conclusion: For the case where $V_1 = V_{in}$ and $V_2 = V_{ref}$, Rf/Ri ¡ 0. This is not possible. Therefore, the solution will be the case where $V_2 = V_{in}$ and $V_1 = V_{ref}$ and Rf/Ri = 1 with V_{ref} = -3V.

- **Range of -2.5 to 2.5V** – Solving the equations similar to the above question, we conclude that calibration cannot be done for this circuit since Rf/Ri = 0 and V1 is undefined. These are impossible gain values.

5.3 Control

1. First, sample some sensor measurement data for the control loop. Now determine the digital inputs for the P,I and D terms. The digital input for proportional can be calculated in terms of the error between the current and desired position of the DC motor. The digital input for integral is the accumulative sum of the error between desired and feedback position, multiplied with the time difference. The derivative's digital input can be calculated by calculating the error between the desired and actual position value, divided by the time difference.
2. Applying an proportional term can tackle the sluggishness.
3. Applying an integral term can reduce steady state error.
4. Applying a derivative term to introduce a dampening effect can reduce overshoot.

5.4 OpAmp- Handwritten Solution

$$V_{out} = V_{ref}\left(1 + \frac{R_f}{R_i}\right) - V_{in}\left(\frac{R_f}{R_i}\right)$$

$$\frac{R_f}{R_i} = \frac{V_{out} - V_{ref}}{V_{ref} - V_{in}}$$

where, $V_1 = V_{in}$

$V_2 = V_{ref}$

substituting range and output

$$\frac{R_f}{R_i} = \frac{-V_{ref}}{V_{ref} + 1.5} \quad \text{——— (1)}$$

$$\frac{R_f}{R_i} = \frac{5 - V_{ref}}{V_{ref} - 1} \quad \text{——— (2)}$$

Equating (1) and (2):

$$\frac{-V_{ref}}{V_{ref} + 1.5} = \frac{5 - V_{ref}}{V_{ref} - 1}$$

$$V_{ref} = -3V$$

$$\frac{R_f}{R_i} = -2$$

$$V_{out} = V_{in}\left(1 + \frac{R_f}{R_i}\right) - V_{ref}\left(\frac{R_f}{R_i}\right)$$

$$\frac{R_f}{R_i} = \frac{V_{out} - V_{in}}{V_{in} - V_{ref}}$$

where, $V_1 = V_{ref}$

$V_2 = V_{in}$

Substituting the range and output:

$$\frac{R_f}{R_i} = \left(\frac{0 + 1.5}{-1.5 - V_{ref}}\right) = \frac{1.5}{-1.5 - V_{ref}} \quad \text{——— (1)}$$

$$\frac{R_f}{R_i} = \left(\frac{5 - 1}{1 - V_{ref}}\right) = \frac{4}{1 - V_{ref}} \quad \text{——— (2)}$$

Equating (1) and (2):

$$\frac{1.5}{-1.5 - V_{ref}} = \frac{4}{1 - V_{ref}} \Rightarrow \boxed{V_{ref} = -3V}$$

$$\boxed{\frac{R_f}{R_i} = 1}$$

6 Code

Following is the complete code for the GUI.

main.cpp

```
#include "dialog.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
    w.show();
    return a.exec();
}
```

dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QSerialPort>
#include <QElapsedTimer>
#include <QDebug>
#include "qcustomplot.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Dialog; }
QT_END_NAMESPACE

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = nullptr);
    ~Dialog();
```

```
    int t = 0;

    void writeToArduino(QString command);

    QString command_format = "1,%1,%2";

private slots:
    void readSerial();
    // DC Motor
    void on_dc_clockwise_radio_toggled(bool checked);
    void on_dc_motor_slider_valueChanged(int value);
    void on_dc_anticlockwise_radio_toggled(bool checked);
    void on_comboBox_currentIndexChanged(int index);

    void on_terminal_returnPressed();

    void on_dc_motor_pos_slider_valueChanged(int value);

    void on_gui_control_checkbox_stateChanged(int arg1);

    void on_servo_slider_valueChanged(int value);

    void on_stepper_slider_valueChanged(int value);

private:
    Ui::Dialog *ui;
    QSerialPort *arduino;
    QByteArray buff;
    QString serialBuffer;
    QString parsedData;
    static const quint16 ard_uno_vID = 9025;
    static const quint16 ard_uno_pID = 67;
    QElapsedTimer timer;
    QString ard_portName;
    bool ard_isAvailable;
```

```
    // TEST
    void test();

    // Motor Functions
    enum MotorState {SERVO, DC_MOTOR_POS, DC_MOTOR_VEL, STEPPER};
    int updateDCMotorControlUI(bool enable = false);
    int updateStepperMotorControlUI(bool enable = false);
    int updateServoMotorControlUI(bool enable = false);

    // Sensor Functions
    QVector<double> ultrasonicVals;
    QVector<double> forceSensorVals;
    QVector<double> motorVelocityVals;
    QVector<double> ambiantLightVals;
    QVector<double> potentiometerVals;
    void processSensorValue(QString buff);
    void plotUltrasonicVals(double valueToPlot);
    void plotMotorVelocityVals(double valueToPlot);
    void plotForceVals(double valueToPlot);
    void plotAmbientVals(double valueToPlot);
    void plotPotentiometerVal(double ValueToPlot);

    void guiControlModeUpdate(bool enable);

};
#endif // DIALOG_H
```

dialog.cpp

```
#include "dialog.h"
#include "ui_dialog.h"
#include <string>
#include <QtSerialPort>
#include <QSerialPortInfo>
#include <QDebug>
#include <QtWidgets>
#include <QElapsedTimer>
```

```
bool DEBUG = true;

Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::Dialog)
{
    ui->setupUi(this);

    arduino = new QSerialPort(this);
    serialBuffer = "";
    ard_isAvailable = false;
    ard_portName = "";
    QSharedPointer<QCPAxisTickerTime>
        timeTicker(new QCPAxisTickerTime);
    timeTicker->setTimeFormat("%h:%m:%s");

    ui->ultrasonicPlot->addGraph();
    ui->ultrasonicPlot->xAxis->setTicker(timeTicker);
    ui->ultrasonicPlot->axisRect()->setupFullAxesBox();
    ui->ultrasonicPlot->yAxis->setRange(0, 30);
    connect(ui->ultrasonicPlot->xAxis,
            SIGNAL(rangeChanged(QCPRange)),
            ui->ultrasonicPlot->xAxis2,
            SLOT(setRange(QCPRange)));
    connect(ui->ultrasonicPlot->yAxis,
            SIGNAL(rangeChanged(QCPRange)),
            ui->ultrasonicPlot->yAxis2,
            SLOT(setRange(QCPRange)));

    ui->forceSensorPlot->addGraph();
    ui->forceSensorPlot->xAxis->setTicker(timeTicker);
    ui->forceSensorPlot->axisRect()->setupFullAxesBox();
    ui->forceSensorPlot->yAxis->setRange(0, 8);
    connect(ui->forceSensorPlot->xAxis,
            SIGNAL(rangeChanged(QCPRange)),
```

```
        ui->forceSensorPlot->xAxis2,
        SLOT(setRange(QCPRange)));
connect(ui->forceSensorPlot->yAxis,
        SIGNAL(rangeChanged(QCPRange)),
        ui->forceSensorPlot->yAxis2,
        SLOT(setRange(QCPRange)));

ui->ambientLightPlot->addGraph();
ui->ambientLightPlot->xAxis->setTicker(timeTicker);
ui->ambientLightPlot->axisRect()->setupFullAxesBox();
ui->ambientLightPlot->yAxis->setRange(0, 1024);
connect(ui->ambientLightPlot->xAxis,
    SIGNAL(rangeChanged(QCPRange)),
    ui->ambientLightPlot->xAxis2, SLOT(setRange(QCPRange)));
connect(ui->ambientLightPlot->yAxis,
    SIGNAL(rangeChanged(QCPRange)),
    ui->ambientLightPlot->yAxis2, SLOT(setRange(QCPRange)));

ui->potentiometerPlot->addGraph();
ui->potentiometerPlot->graph(0)->setPen(QPen(Qt::blue));
ui->potentiometerPlot->xAxis->setTicker(timeTicker);
ui->potentiometerPlot->axisRect()->setupFullAxesBox();
ui->potentiometerPlot->yAxis->setRange(0, 300);
connect(ui->potentiometerPlot->xAxis, S
        IGNAL(rangeChanged(QCPRange)),
        ui->potentiometerPlot->xAxis2,
        SLOT(setRange(QCPRange)));
connect(ui->potentiometerPlot->yAxis,
        SIGNAL(rangeChanged(QCPRange)),
        ui->potentiometerPlot->yAxis2,
        SLOT(setRange(QCPRange)));

ui->dcMotorVelocityPlot->addGraph();
ui->dcMotorVelocityPlot->xAxis->setTicker(timeTicker);
ui->dcMotorVelocityPlot->axisRect()->setupFullAxesBox();
ui->dcMotorVelocityPlot->yAxis->setRange(0, 103);
```

```
connect(ui->dcMotorVelocityPlot->xAxis,
        SIGNAL(rangeChanged(QCPRange)),
        ui->dcMotorVelocityPlot->xAxis2,
        SLOT(setRange(QCPRange)));
connect(ui->dcMotorVelocityPlot->yAxis,
        SIGNAL(rangeChanged(QCPRange)),
        ui->dcMotorVelocityPlot->yAxis2,
        SLOT(setRange(QCPRange)));

timer.start();

foreach(const QSerialPortInfo &serialPortInfo,
    QSerialPortInfo::availablePorts()){
    if(serialPortInfo.hasVendorIdentifier() &&
            serialPortInfo.hasProductIdentifier()){
        if(serialPortInfo.vendorIdentifier()
                            == ard_uno_vID){
            if(serialPortInfo.productIdentifier()
                            == ard_uno_pID){
                ard_portName = serialPortInfo.portName();
                ard_isAvailable = true;
            }
        }
    }
}

if(ard_isAvailable){
    // open + configure port
    arduino->setPortName(ard_portName);
    arduino->open(QSerialPort::ReadWrite);
    arduino->setBaudRate(QSerialPort::Baud9600);
    arduino->setDataBits(QSerialPort::Data8);
    arduino->setParity(QSerialPort::NoParity);
    arduino->setStopBits(QSerialPort::OneStop);
    arduino->setFlowControl(QSerialPort::NoFlowControl);
    QObject::connect(arduino,
```

```
                        SIGNAL(readyRead()),
                        this, SLOT(readSerial()));

    }else{
        // display message on error
        QMessageBox::warning(this, "Port Error",
                             "Can't find Arduino!");
    }
}

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

// ------------------------------ SERIAL COMMUNICATION FUNCTIONS

void Dialog::writeToArduino(QString command) {
    ui -> command_label->setText(QString
        ("<span style=\" font-size:9pt;
                        font-style:italic;
                        color:#d3d7cf;\">%1</span>"
                        ).arg(command));
    qDebug() << command;
    if(arduino->isWritable()){
        arduino->write(command.toStdString().c_str());
    }else{
        qDebug() << "Could not write to Serial!";
    }
}

void Dialog::readSerial(){
//    qDebug() << "Serial works";
```

```
    QStringList buffer_split = serialBuffer.split("\n");
    if(buffer_split.length() < 3){
            buff = arduino->readAll();
            serialBuffer = serialBuffer +
                           QString::fromStdString(
                           buff.toStdString()
                           );
            buff.clear();
        }else{
            temp_lcdNumber
            serialBuffer = "";
            qDebug() << "buffer_split" << buffer_split << "\n";
            parsedData = buffer_split[1];
            processSensorValue(parsedData);
     }

}

// ------------------------------- SLOT FUNCTIONS FROM GUI

// Gui Control Enable/Disable
void Dialog::on_gui_control_checkbox_stateChanged(int arg1)
{
    qDebug() << "arg for gui: " << arg1;
    if(arg1){
        guiControlModeUpdate(true);
    }
    else{
       guiControlModeUpdate(false);
       Dialog::writeToArduino("0,0,0");
       updateDCMotorControlUI(false);
       updateStepperMotorControlUI(false);
       updateServoMotorControlUI(false);
    }

}
```

```
void Dialog::guiControlModeUpdate(bool enable){
    ui->comboBox->setEnabled(enable);
    ui->label_2->setEnabled(enable);
    ui->motor_control_label->setEnabled(enable);
    ui->dc_motor_label->setEnabled(enable);
    ui->stepper_motor_label->setEnabled(enable);
    ui->servo_motor_label->setEnabled(enable);

    ui->terminal->setEnabled(enable);
    ui->terminalButton->setEnabled(enable);
    ui->terminal_label->setEnabled(enable);
    if(enable)
        ui->terminal->setText("> ");
    else
        ui->terminal->setText("> gui control disabled");
}

// Motor Control Enable/Disable
void Dialog::on_comboBox_currentIndexChanged(int index)
{
    if(index == 1){
        // DC Motor UI
        updateDCMotorControlUI(true);
        // Stepper Motor UI
        updateStepperMotorControlUI(false);
        // Servo Motor UI
        updateServoMotorControlUI(false);
    }

    else if(index == 2){
        // DC Motor UI
        updateDCMotorControlUI(false);
        // Stepper Motor UI
        updateStepperMotorControlUI(true);
        // Servo Motor UI
```

```
        updateServoMotorControlUI(false);
    }

    else if(index == 3){
        // DC Motor UI
        updateDCMotorControlUI(false);
        // Stepper Motor UI
        updateStepperMotorControlUI(false);
        // Servo Motor UI
        updateServoMotorControlUI(true);
    }

    else{
        // DC Motor UI
        updateDCMotorControlUI(false);
        // Stepper Motor UI
        updateStepperMotorControlUI(false);
        // Servo Motor UI
        updateServoMotorControlUI(false);
    }
}

// DC Motor
void Dialog::on_dc_clockwise_radio_toggled(bool checked)
{
    if(checked){
        // send motor control command
        qDebug() << "DC Motor radio checked!";
        if(checked){
            // send motor control command
            qDebug() << "DC Motor radio checked!";
            if(ui->dc_motor_pos_slider->isEnabled()){
                int position_value =
                    ui->dc_motor_pos_slider->value();
                Dialog::writeToArduino(command_format.
                    arg(DC_MOTOR_POS).arg(position_value));
```

```
            }
            else if(ui->dc_motor_slider->isEnabled()){
                int velocity_value =
                    ui->dc_motor_slider->value();
                Dialog::writeToArduino(command_format.
                    arg(DC_MOTOR_VEL).arg(velocity_value));
            }
        }
    }
}

void Dialog::on_dc_anticlockwise_radio_toggled(bool checked)
{
    if(checked){
        // send motor control command
        qDebug() << "DC Motor radio checked!";
        if(ui->dc_motor_pos_slider->isEnabled()){
            int position_value = -1 *
                        ui->dc_motor_pos_slider->value();
            Dialog::writeToArduino(command_format.
                arg(DC_MOTOR_POS).arg(position_value));
        }
        else if(ui->dc_motor_slider->isEnabled()){
            int velocity_value = -1 *
                        ui->dc_motor_slider->value();
            Dialog::writeToArduino(command_format.
                arg(DC_MOTOR_VEL).arg(velocity_value));
        }
    }
}

void Dialog::on_dc_motor_slider_valueChanged(int value)
{
    // update UI
    qDebug() << "DC Motor Speed Changed to" << value;
    ui->dc_motor_speedVal_label->setText(QString
```

```
        ("<p align=\"right\">%1</p>").arg(value));
    if(value!=0){
        ui->dc_motor_pos_slider->setEnabled(false);
    }
    else{
        ui->dc_motor_pos_slider->setEnabled(true);
    }
    Dialog::writeToArduino(command_format.arg(
                           DC_MOTOR_VEL).arg(value)
                           );
//    Dialog::writeToArduino(QString("%1").arg(value));

}

void Dialog::on_dc_motor_pos_slider_valueChanged(int value)
{
    ui->dc_motor_posVal_label->setText(QString
        ("<p align=\"right\">%1</p>").arg(value));
    if(value!=0){
        ui->dc_motor_slider->setEnabled(false);
    }
    else{
        ui->dc_motor_slider->setEnabled(true);
    }
    Dialog::writeToArduino(command_format.arg(
                           DC_MOTOR_POS).arg(value)
                           );
//    Dialog::writeToArduino(QString("%1").arg(value));

}

// Servo Motor
void Dialog::on_servo_slider_valueChanged(int value)
{
    ui->servo_slider->setTracking(false);
    ui->servo_motor_stepVal_label->setText(QString
```

```
        ("<p align=\"right\">%1</p>").arg(value));
    Dialog::writeToArduino(command_format.arg(
                                SERVO).arg(value)
                                );
}

// Stepper
void Dialog::on_stepper_slider_valueChanged(int value)
{
    ui->stepper_slider->setTracking(false);
    ui->stepper_speedVal_label->setText(QString
        ("<p align=\"right\">%1</p>").arg(value));
    Dialog::writeToArduino(command_format.arg(
                                STEPPER).arg(value)
                                );
}

// ------------------------------- MOTOR CONTROL FUNCTIONS

int Dialog::updateDCMotorControlUI(bool enable){
    ui->dc_motor_pos_slider->setTracking(false);
    ui->dc_motor_slider->setTracking(false);
    ui->dc_clockwise_radio->setEnabled(enable);
    ui->dc_anticlockwise_radio->setEnabled(enable);
    ui->dc_motor_slider->setEnabled(enable);
    ui->dc_motor_speedVal_label->setEnabled(enable);
    ui->dc_motor_text->setEnabled(enable);
    ui->dc_motor_posVal_label->setEnabled(enable);
    ui->dc_motor_pos_slider->setEnabled(enable);
    ui->dc_motor_text_2->setEnabled(enable);
    writeToArduino(command_format.arg(
                MotorState::DC_MOTOR_POS).arg(0)
                );
    writeToArduino(command_format.arg(
                MotorState::DC_MOTOR_VEL).arg(0)
                );
```

```
    return 0;

}

int Dialog::updateStepperMotorControlUI(bool enable){
//    ui->stepper_clockwise_radio->setEnabled(enable);
//    ui->stepper_anticlockwise_radio->setEnabled(enable);
    ui->stepper_slider->setEnabled(enable);
    ui->stepper_speedVal_label->setEnabled(enable);
    ui->stepper_text->setEnabled(enable);
    writeToArduino(command_format.arg(
                MotorState::STEPPER).arg(0)
                );
    return 0;
}

int Dialog::updateServoMotorControlUI(bool enable){
    ui->servo_motor_stepVal_label->setEnabled(enable);
    ui->servo_slider->setEnabled(enable);
    ui->servo_text->setEnabled(enable);
    writeToArduino(command_format.arg(
                MotorState::SERVO).arg(0)
                );
    return 0;
}

// ------------------------------- SENSOR FUNCTIONS

void Dialog::processSensorValue(QString buff){
    qDebug() << "buff " << buff;
    double valueToPlot;
//    QString buff = "u.210";
    QStringList l = buff.split(",");
    qDebug() << "l.length()" << l.length();
    if(l.length()!=2 or l.length()!=6){
        qDebug() << "returning!!";
```

```
//        return;
    }

    qDebug() << "l: " << l;
    qDebug() << "l[0]" << l[0];

    if(l[0] == 'u'){
        // u,255
        valueToPlot = l[1].toDouble();
        ultrasonicVals.push_back(valueToPlot);
        plotUltrasonicVals(valueToPlot);
    }
    else if(l[0] == 'f'){
        // f,200
        valueToPlot = l[1].toDouble();
        forceSensorVals.push_back(valueToPlot);
        plotForceVals(valueToPlot);
    }
    else if(l[0] == 'a'){
        // a,255
        valueToPlot = l[1].toDouble();
        ambiantLightVals.push_back(valueToPlot);
        plotAmbientVals(valueToPlot);
    }
    else if(l[0] == 'm'){
        // m,255
        int motor = l[1].toInt();
        switch (motor) {
        case 0:
            //Servo Motor
            valueToPlot = l[2].toDouble();
            ui->servo_motor_lcd->intValue();
            ui->servo_motor_lcd->display(valueToPlot);
            // show servo value
            break;
        case 1:
```

```
            // DC Motor Position
            valueToPlot = l[3].toDouble();
            ui->dc_motor_pos_lcd->intValue();
            ui->dc_motor_pos_lcd->display(valueToPlot);
            break;
        case 2:
            // DC Motor Velocity
            valueToPlot = l[4].toDouble();
            motorVelocityVals.push_back(valueToPlot);
            plotMotorVelocityVals(valueToPlot);
            break;
        case 3:
            // Stepper
            valueToPlot = l[5].toDouble();
            ui->stepper_motor_lcd->intValue();
            ui->stepper_motor_lcd->display(valueToPlot);
            break;
        default:
            break;
        }
        valueToPlot = l[1].toDouble();
        motorVelocityVals.push_back(valueToPlot);
        plotMotorVelocityVals(valueToPlot);
    }
    else if(l[0] == 'p'){
        double valueToPlot = (l[1].split('\r'))[0].toDouble();
        potentiometerVals.push_back(valueToPlot);
        plotPotentiometerVal(valueToPlot);
    }
    else
        return;
}

void Dialog::plotUltrasonicVals(double valueToPlot){
                double key = timer.elapsed()/1000.0;
                static double lastPointKey = 0;
```

```
                if (key-lastPointKey > 0.002)
                {
                  ui->ultrasonicPlot->
                            graph(0)->
                            addData(key, valueToPlot);
                  lastPointKey = key;
                }
                ui->ultrasonicPlot->xAxis->setRange(
                               key,
                               8,
                               Qt::AlignRight
                               );
                ui->ultrasonicPlot->replot();

                ui->ultrasonicLabel->setText(
                        QString (
                               "<p align=\"right\">%1</p>").
                               arg(valueToPlot));
}

void Dialog::plotForceVals(double valueToPlot){
                double key = timer.elapsed()/1000.0;
                static double lastPointKey = 0;
                if (key-lastPointKey > 0.002)
                {
                  // add data to lines:
                  ui->forceSensorPlot->
                            graph(0)->
                            addData(key, valueToPlot);
                  lastPointKey = key;
                }
                ui->forceSensorPlot->xAxis->
                                  setRange(
                                      key,
                                      8,
                                      Qt::AlignRight);
```

```
                ui->forceSensorPlot->replot();

                ui->forceLabel->setText(QString
                    ("<p align=\"right\">%1</p>").
                                    arg(valueToPlot));
}

void Dialog::plotAmbientVals(double valueToPlot){
                double key = timer.elapsed()/1000.0;
                static double lastPointKey = 0;
                if (key-lastPointKey > 0.002)
                {
                  // add data to lines:
                  ui->ambientLightPlot->graph(0)->
                                addData(key,
                                valueToPlot);
                  lastPointKey = key;
                }
                ui->ambientLightPlot->xAxis->
                                setRange(key,
                                        8,
                                        Qt::AlignRight);
                ui->ambientLightPlot->replot();

                ui->ambientLightLabel->setText(QString("
                    <p align=\"right\">%1</p>").
                    arg(valueToPlot));
}

void Dialog::plotPotentiometerVal(double ValueToPlot){
    double key = timer.elapsed()/1000.0;
    static double lastPointKey = 0;
    if (key-lastPointKey > 0.002)
    {
      // add data to lines:
      ui->potentiometerPlot->graph(0)->
```

```
                              addData(key, ValueToPlot);
      lastPointKey = key;
    }
    ui->potentiometerPlot->xAxis->setRange(key,
                                    8,
                                    Qt::AlignRight);
    ui->potentiometerPlot->replot();
}

void Dialog::plotMotorVelocityVals(double valueToPlot){
    double key = timer.elapsed()/1000.0;
    static double lastPointKey = 0;
    if (key-lastPointKey > 0.002)
    {
      // add data to lines:
      ui->dcMotorVelocityPlot->graph(0)->
                                addData(key, valueToPlot);
      lastPointKey = key;
    }
    ui->dcMotorVelocityPlot->xAxis->setRange(key,
                                    8,
                                    Qt::AlignRight);
    ui->dcMotorVelocityPlot->replot();

    ui->dcMotorVelocityLabel->setText(QString(
        "<p align=\"right\">%1 rpm</p>").arg(valueToPlot));
}

void Dialog::on_terminal_returnPressed()
{
    // send serial command typed
//    qDebug() << ui->terminal->text();
    QStringList typed = (ui->terminal->text()).split(" ");
    QString command = "";
    if(typed.length()<2 and typed[0][0]!='>'){
        command = typed[0];
```

```
    }
    else if(typed.length()<2 and typed[0][0]=='>'){
        command = (typed[0].split(">"))[1];
    }
    else{
        command = typed[1];
    }
//    qDebug() << command;
    writeToArduino(command);
    ui->terminal->setText("> ");

}
```