

Open Sky Planetarium

Summer Internship Project Report

Submitted by

Saloni Mundra

DA-IICT, Gandhinagar



FOSSEE Group

Indian Institute of Technology Bombay

July 2017

Mentors:

Ms. Inderpreet Arora

Mr. Rupak Rokade

Supervisor:

Prof. Kannan M. Moudgalya

IIT Bombay

CERTIFICATE

We hereby declare that this project report entitled, "Development of Open Sky Planetarium" is being submitted by us in fulfillment of the requirements for the completion of Summer Internship 2017 at Integrated Development Lab, FOSSEE Group, Indian Institute of Technology Bombay. This report is an authentic record of the aforesaid project carried out during the period of May 2017-June 2017.

Jaideep Mishra
NIT Durgapur

Ram Mohan Kumar
IIT BHU

Saloni Mundra
DA-IICT

This is to certify that Jaideep Mishra, Ram Mohan Kumar, and Saloni Mundra have worked sincerely for the above mentioned project under my supervision and guidance during their summer internship. Their performance and conduct during the project was satisfactory.



Mr. Rupak Rokade
Assistant Project Manager, IIT Bombay

Prof. Kannan M. Moudgalya
Professor, IIT Bombay

प्रोफेसर
रसायनिक अभियंत्रित्व विभाग
भारतीय प्रौद्योगिकी संस्थान-मुंबई, फर्स्ट, मुंबई-७६.
Professor
Department of Chemical Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai-400 076.

Acknowledgement

First of all, I would like to express my heartfelt gratitude to our project supervisor **Prof. Kannan M. Moudgalya** for offering and allowing me to work on this novel project to develop an affordable Open Sky Plane-tarium.

I further express our special thanks to my mentors, **Ms. Inderpreet Arora** and **Mr. Rupak Rokade** whose valuable contribution in stimulating suggestions and encouragement helped me in coming with the best solutions.

I would like to thank **Mr. Rajesh Kushalkar**, Senior Project Manager of Integrated Development Lab (IDL) for providing resources and constant support for the development of the project. I would also like to express my sincere thanks to the entire IDL team, who helped me with their knowledge and support.

I express my deepest gratitude to **Mr. Sudhakar Kumar**, who has invested his full efforts in guiding the team and achieving the goal.

Abstract

The project aims to provide low cost access to planetariums, especially to the schools of the country. The purpose is to educate and motivate the youth to explore and know about the night sky. Visit to a planetarium is expensive and seldom do we get a chance to see one. On the contrary, clear skies can still be observed away from the cities, but with the lack of proper knowledge base to rely on for stargazing, the experience becomes obsolete. Open Sky Planetarium is a project that seeks out schools to provide them with an educational tool to introduce kids to the night sky through a planetarium-like experience under an open sky. This is achieved by making the equipment low-cost, thus making it affordable and accessible to all schools to play a planetarium show in their own locality. It also aims to keep the operation of the equipment and the application as user-friendly as possible.

List of Figures

2.1	Stellarium software GUI	3
2.2	Qt Creator IDE.....	4
4.1	LASER intensity control.....	9
4.2	Code snippets for LASER intensity control.....	10
4.3	Reset functionality in the plug-in	10
4.4	Code snippets for reset functionality in the plugin.....	11
4.5	Current coordinates (in degrees).....	11
4.6	Code snippets for current coordinates (in degrees).....	12
4.7	Motor speed control	12
4.8	Code snippets for motor speed control	13
4.9	Instant help on mouse hover	13
4.10	Success message after setting first reference	15
4.11	Success message after setting second reference	15
5.1	Code snippet in SerialCom.cpp.....	16
5.2	Reference setting section.....	17
5.3	Message on successful reference set up.....	17
5.4	Code snippet for the move function.....	18
5.5	Increased timeout in plugin.....	18
5.6	Code snippets for releasing handle.....	19

Contents

Acknowledgement	i
Abstract	ii
List of Figures	iii
1 Introduction	1
2 Tools for Plug-in Development	2
2.1 Stellarium.....	2
2.1.1 Plugins.....	3
2.2 Qt.....	3
2.2.1 Qt Framework.....	3
2.2.2 Qt Creator.....	4
2.3 CMake.....	5
3 Qt based Software module	6
4 Functions added to the Plugin	9
5 Problems Resolved	16
6 Future prospects	20
Bibliography	21

Chapter 1

Introduction

Open Sky Planetarium (OSP) makes use of Stellarium- an open source software that simulates stars in real-time and can be used as an on-screen planetarium. The tool has a LASER pointing device that guides the user through the stars and is controlled by an application supported by Stellarium. The idea is to calibrate the device by setting at least two stars as references and then any star can be guided to by the means of Stellarium and the OSP plugin built for it. The OSP plugin also provides the user with a script engine that can be used to write scripts with background audio to be played during the planetarium show.

Chapter 2

Tools for Plug-in Development

2.1 Stellarium

Stellarium is an open source software that simulates the night sky and displays a number of stars in their position at a particular time. The night sky simulated is specific to various locations, hence anybody can view the sky in their location in Stellarium to refer to actual objects in the sky while stargazing. The database in Stellarium is comprehensive: it contains the equatorial coordinates of an object in the sky in reference year J2000, from which the data is calculated to simulate the object in its position at any particular point of time. The objects in the Stellarium sky move just as the objects in the real sky do: the simulation mirrors the actual sky. There are also various sky cultures so that people from all parts of the world can use the software hassle-free.

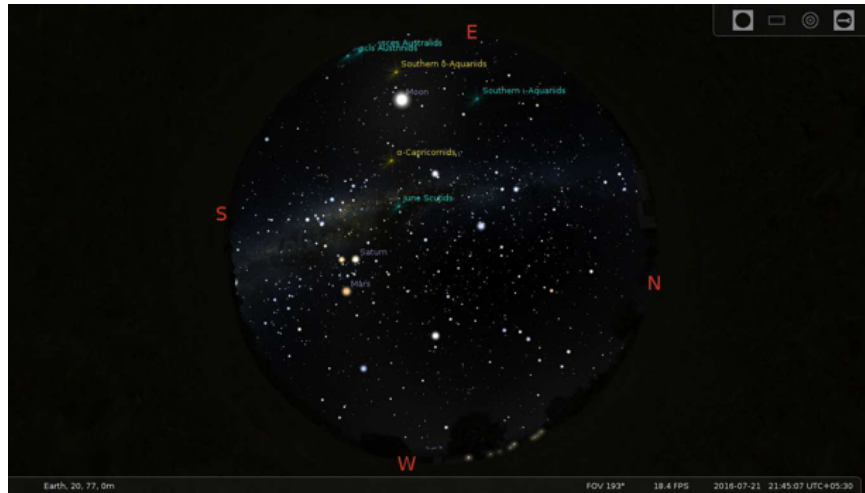


Figure 2.1: Stellarium software GUI

2.1.1 Plugins

Stellarium offers a wide range of functional flexibility which lets us use Stellarium for highly specific purposes through plugins, which are add-on functionalities that let perform a certain task within the Stellarium environment. Some plugins are installed along with Stellarium and just need to be activated when needed. These are the static plugins. Some other plugins will have to be downloaded separately and installed onto Stellarium: these are the dynamic plugins.

2.2 Qt

2.2.1 Qt Framework

Qt is a cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with the capabilities and speed

thereof. Qt is currently being developed both by the Qt Company, a subsidiary of Digia, and the Qt Project under open-source governance, involving individual developers and firms working to advance Qt.

2.2.2 Qt Creator

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which is part of the SDK for the Qt GUI Application development framework. It includes a visual debugger and an integrated GUI layout and forms designer. The editor's features include syntax highlighting and autocompletion, but purposely not tabs (although plug-ins are available). Qt Creator uses the C++ 11 compiler from the GNU Compiler Collection on Linux and FreeBSD. On Windows it can use MinGW or MSVC with the default install and can also use Microsoft Console Debugger when compiled from source code.

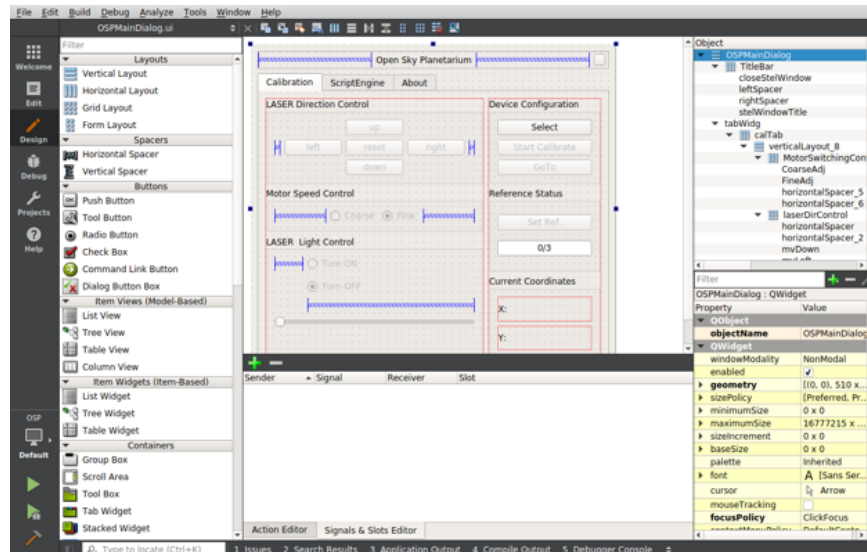


Figure 2.2: Qt Creator IDE

2.3 CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of one's choice.

Chapter 3

Qt based Software module

bt_osp_off.png/bt_osp_on.png

The png files for the icon displayed in Stellarium to start OpenSky-Planetarium Plugin. These icons are simple png images displaying osp text. When the icon is not clicked it is shown in the white colour else shown in grey colour in the toolbar in Stellarium. Hence two icons with same image and different colour was needed to display its on/off state.

OpenSkyPlanetarium.qrc

The Qt resource file for including resources such as png files in our plugin compilation.

CMakeLists.txt

CMakeLists.txt includes commands to build the project Open Sky Planetarium. This file is used to determine the external libraries and path variable for successfully building the plugin.

src/

This directory includes source files for the dynamic plugin.

Calibrate.hpp/Calibrate.cpp

All codes related to calibration are included in this class Calibrate. The transformation matrix is calculated in this class. The current build includes calculation using three references. This can be increased in future for more accurate calculation of the transformation matrix. It also includes functions for getting equatorial and horizontal coordinates using transformation matrix.

SerialCom.hpp/SerialCom.cpp

Serial Communication between the Arduino and the plugin is facilitated by SerialCom class. The examples in Qt docs served helpful for development of Serial Communication between Arduino and Stellarium.

LaserDev.hpp/LaserDev.cpp

Communication with devices such as sending request and receiving response is performed by LaserDev class. LaserDev class uses SerialCom class for sending and receiving data from Arduino. Different commands such as move, movx, movy, laon, loff, post etc. is sent to the Arduino, which in turn gives back response to these commands.

OpenSkyPlanetarium.hpp/OpenSkyPlanetarium.cpp

The main file of the plugin. Links the GUI file with Stellarium. This class has a predefined format and no changes are required to this format to integrate the plugin with Stellarium.

CMakeLists.txt

CMakeLists.txt includes commands to compile the classes. This file is used to set resources and ui files for compiling with the project.

gui/

The gui/ folder includes files for the plugin gui. This folder includes OSPMainDialog.ui and OSPMainDialog class.

OSPMainDialog.ui

The xml file of the gui is OSPMainDialog.ui. This file is created using Qt Creator IDE.

OSPMainDialog.hpp/OSPMainDialog.cpp

This is the main class of the plugin. It links GUI to the various classes and its functions. The GUI signals are connected to the respective functions in this class. The OSPMainDialog includes linking GUI and the Script Engine Functions like compile, open, save and execute. It also includes user defined signals that will be called during execution of the script.

Chapter 4

Functions added to the Plugin

1. Laser Intensity Control

The slider allows you to control the intensity of the laser by dragging it. It has a maximum value of 255 and a minimum of 50. The slider has a step size of 1. On dragging the slider, the `setIntensity()` function is called. The returned integer value is converted into a string and sent to the Arduino after “lase” command is sent to it. The Arduino writes the value sent by the plugin into the laser pin and sends back the command “done_lase.” Thus, hand-shake is established. The slider is only enabled when the laser is turned on. On turning off the laser, the slider comes back to its initial position.

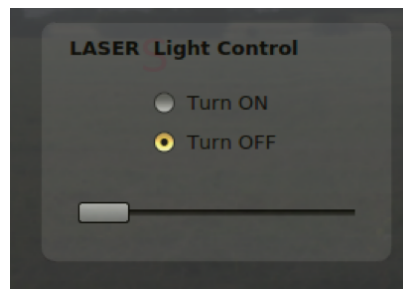


Figure 4.1: LASER intensity control

```

void OSPMainDialog :: setIntensity(int x){
    int l=ui->intensity->value();
    device.setIntensity(l);
    qDebug() << "[OpenSkyPlanetarium]:Changing intensity" << endl;
}

void LaserDev ::setIntensity(int x){
    QString s=QString::number(x);
    if(s.length()==2)
        s="00"+s;
    else if(s.length()==3)
        s="0"+s;
    intense=s;
    comm=QString("lase");
    emit debug_send(comm);
    thread.sendRequest(osp_serialPort,1000,QString(comm));
}

else if(recvd.compare("done_lase")==0){
    emit debug_send("intensity");
    comm=intense;
    thread.sendRequest(osp_serialPort,1000,QString(comm));
    emit debug_send(comm);
}

```

Figure 4.2: Code snippets for LASER intensity control

2. Reset

The reset button on being clicked takes the motor to the initial position from which the movement started, or in other words, takes it to the coordinates (0,0). This button is available all the time to the user except before selecting the device. The reset button issues the move(0,0) command to the Arduino, which then takes it to the initial coordinates. Calibration related data is not lost during the reset operation. It makes use of the original goto command code.



Figure 4.3: Reset functionality in the plug-in

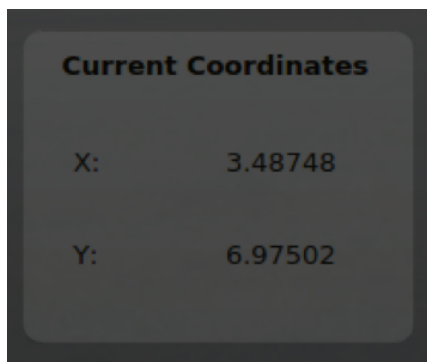

```
void OSPMainDialog :: reset(){
    ... //this->initDevice();
    ... device.resetAll();
    ... QString val="0.000000";
    ... ui->X->setText(val);
    ... ui->Y->setText(val);
}
```

```
void LaserDev :: resetAll(){
    ... move(0,0);
}
```

Figure 4.4: Code snippets for reset functionality in the plugin

3. Current Coordinates (in degrees)

This section gives the current position of the laser. It shows how many degrees the motor has moved in the X and Y direction. After every motor movement command, the `getPos()` function is called which sends the “post” string to the Arduino. The Arduino then sends the current position of the laser to the plugin, which is then displayed in the Current Coordinates section.



The image shows a dark grey rectangular box with rounded corners. At the top, the text "Current Coordinates" is displayed in a bold, white font. Below this, there are two lines of text. The first line shows "X:" followed by the value "3.48748". The second line shows "Y:" followed by the value "6.97502".

Current Coordinates
X: 3.48748
Y: 6.97502

Figure 4.5: Current coordinates (in degrees)

```
void OSPMainDialog :: pos_received(QString x,QString y){
    qDebug() << "Printing X and Y = ["<<x<<","<<y<<"]";
    .....
    acTemp = x;
    altTemp = y;
    .....
    double l=x.toDouble()*(180.0/3.14159);
    .....
    double m=y.toDouble()*(180.0/3.14159);
    .....
    QString s=QString::number(l);
    .....
    QString t=QString::number(m);
    .....
    ui->X->setText(s);
    .....
    ui->Y->setText(t);
    .....
}
```

Figure 4.6: Code snippets for current coordinates (in degrees)

4. Motor Speed Controls

It allows you to select whether you want to move the motor on high speed or low. The default action is fine adjustment (low speed). On selecting the required checkbox, “coad” is sent to theArduino for coarse adjustment (high speed) and “fiad” for fine adjustment (low speed).

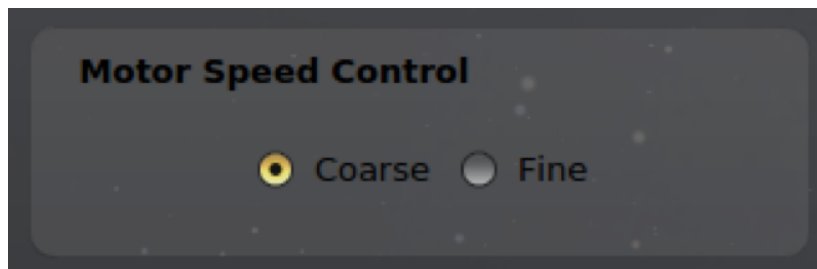


Figure 4.7: Motor speed control

```

void OSPMainDialog :: adjToggled(){
.....if (ui->CoarseAdj->isChecked()){
.....device.CoarseAdj();
.....qDebug() << "[OpenSkyPlanetarium]:High Speed ON" << endl;
.....}
.....else{
.....device.FineAdj();
.....qDebug() << "[OpenSkyPlanetarium]:Low Speed ON" << endl;
.....}
}

```

```

/*
CoarseAdj():
to let the motors go on high speed
*/
void LaserDev :: CoarseAdj(){
.....comm=QString("coad");
.....thread.sendRequest(osp_serialPort,1000,QString(comm));
}
/*
FineAdj():
to let the motors go on a low speed
*/
void LaserDev :: FineAdj(){
.....comm=QString("fiad");
.....thread.sendRequest(osp_serialPort,1000,QString(comm));
}

```

Figure 4.8: Code snippets for motor speed control

5. Help Over Mouse Hover

A brief instant help is displayed on mouse hover over different buttons and sections. This is done to make the plugin as user friendly as possible. This is implemented by using the tooltip feature of the QT Creator.

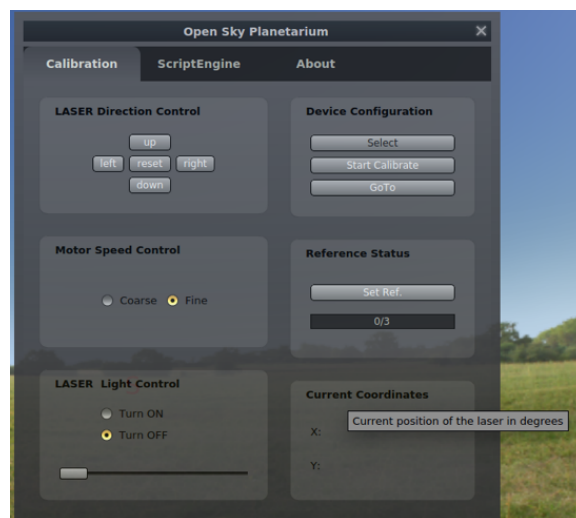


Figure 4.9: Instant help on mouse hover

6. **Enable/Disable Hierarchy**

For making the controls more user friendly and to avoid mistakes, the plugin has been designed such that the user is directed to follow a certain path that will take him to the correct output.

- Initially, on startup, only the Select button is enabled. Rest are all disabled.
- On Selecting the device, apart from Laser Direction Controls, GoTo and Set Ref, everything else is enabled.
- The Laser Direction Controls and Set Ref gets enabled on clicking Start Calibration.
- The GoTo button gets enabled when at least two references are correctly set.
- After three references have been set and calibration done, Set Ref is disabled along with Laser Direction Controls (except reset).
- Clicking on Select Device, the plugin reverts to its original state.

7. **Reference Message Pop ups**

To make the application more user friendly and check for its correctness, message pop ups have been added. On successful reference set up, a dialog box opens which displays the success message along with the current az and alt coordinates(in radians) of the reference star.

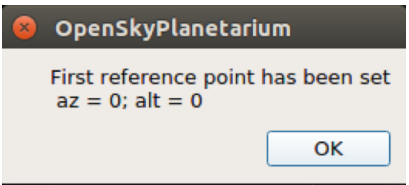


Figure 4.10: Success message after setting first reference

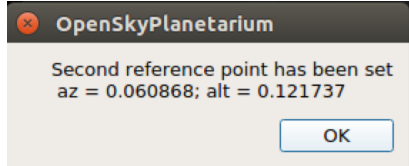


Figure 4.11: Success message after setting second reference

Chapter 5

Problems Resolved

1. The system of Laser Direction Control was designed in such a way that the motor moves when the button is clicked and stops on its release. But it did not work as it was expected because of the inclusion of mutex. Mutex are programming objects that help in synchronization. Earlier, if a new command was given when the motor was executing a command, the new command was entertained only after the old one was finished. If the old command did not get executed because of any reason, the program would hang. Removal of mutex code solved these undesired behaviours and solved some problems of Serial Communication.

In `SerialCom::run()`, the lines `mutex.lock()` and `mutex.unlock()` have been removed.

```
· mutex.lock();  
· QString myPortName;  
· if (myPortName != port) {  
·   · myPortName = port;  
·   · myPortNameChanged = true;  
· }  
  
· int myWaitTimeout = waitTime;  
· QString myRequest = request;  
· mutex.unlock();
```

Figure 5.1: Code snippet in `SerialCom.cpp`

2. The problem with the reference counter has been fixed and a message is displayed on successful reference set up with the current az and alt coordinates. Earlier, there was no upper bound to the reference counter and could be incremented indefinitely. Also, a message box is now displayed on successful reference set up specifying the coordinates of the reference star. This will help the user to ensure that he is going the right way.

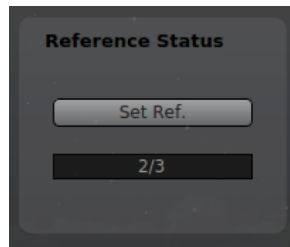


Figure 5.2: Reference setting section

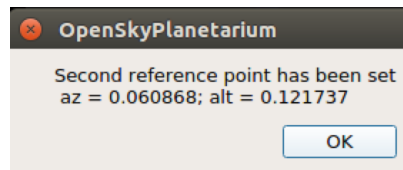


Figure 5.3: Message on successful reference set up

3. When two or more simultaneous GoTo commands were sent by the plugin, the Arduino used to hang. This has been solved by the usage of flags which now discard any new GoTo commands sent in mid of execution of one. The plugin accepts the GoTo request if the flag is true and further changes it to false. So, if any new request is sent in mid of execution of one, then the new GoTo request gets rejected. This solves the problem of Arduino hang which otherwise had to use the mechanism of hard reset.

```

void LaserDev::move(double x, double y){
    .....
    ..... dac=x;
    ..... dalt=y;
    if(flag){
    ..... comm=QString("move");
    ..... thread.sendRequest(osp_serialPort,1000,QString(comm));
    ..... flag=false;
    ..... }
}

```

Figure 5.4: Code snippet for the move function

4. Initially the Arduino used to send the string done_move before the completion of the goto step. This was done in order to protect it from the timeout code of 1s. We changed the timeout for the move command from 1s to 20s so that a proper handshake can be established.

```

else if(recvd.compare("float")==0){
    ..... emit debug_send("float");
    ..... bool ac_neg=false,alt_neg=false;
    ..... if(dac<0) ac_neg=true;
    ..... if(dalt<0) alt_neg=true;
    ..... ac=QString::number(dac,'f',4);
    ..... alt=QString::number(dalt,'f',4);
    ..... if(!ac_neg){
    ..... ac = QString("+"+ac);
    ..... }
    ..... if(!alt_neg){
    ..... alt = QString("+"+alt);
    ..... }
    ..... comm=QString("m_%1_%2").arg(ac).arg(alt);
    ..... thread.sendRequest(osp_serialPort,20000,QString(comm));
    ..... emit debug_send(comm);
    ..... }
}

```

Figure 5.5: Increased timeout in plugin

5. Earlier, closing the plugin wouldn't release the serial port or reset the hardware state. Now, an additional function has been made that is called on clicking the close button on the gui that releases the handle and sends the "clos" string to the Arduino which then resets the hardware.


```
void OSPMainDialog :: closeWin(){
    ... debug_received("closing window");
    ... device.closeWindow();
    ... close();
}
```

```
void LaserDev :: closeWindow(){
    ... comm=QString("clos");
    ... thread.sendRequest(osp_serialPort,5000,QString(comm));
}
```

```
void LaserDev :: releasePort(){
    ... thread.serial.close();
    ... qDebug()<<"Releasing Port";
}
```

Figure 5.6: Code snippets for releasing handle

Chapter 6

Future prospects

Most of the features in the plugin have been implemented and been made as user friendly as possible. But still some more tasks and enhancements are needed before making it available to the end user. The tasks remaining are:

- Implement Open Source Cross Platform Audio Playing.
- Solve the problem of multiple goto in the script engine.
- Improve calibration by increasing number of references from three to four.
- Write a script to automate installation of required packages for stellarium and the plugin.

Another improvement that can be made is to change the serial communication approach from synchronous to asynchronous. This can make it more user friendly and robust.

Bibliography

[1] <http://www.stellarium.org/>

[2] <https://www.qt.io/>