

FOSSEE Summer Internship Programme - 2016

Open Sky Planetarium



Vasudha Varadarajan
Dhiraj Salian
Yudhisther Bhargava

22 July, 2016

Guided by:

Prof. Kannan. M. Moudgalya
Prof. Dipankar
Prof. D.B. Phatak

Mentor:

Ms. Inderpreet Arora

Contents

Acknowledgements	1
Declarations	2
Abstract	3
1 Introduction	5
1.1 Aim of the project	5
1.2 Definitions to some terms used	5
2 Tools for Software	9
2.1 Stellarium	9
2.1.1 Plugins	10
2.1.2 Scripting API	10
2.2 Qt	11
2.2.1 Qt Framework	11
2.2.2 Qt Creator	11
2.3 PyQt	12
3 Modules	13
3.1 Hardware	13
3.1.1 Stepper motor with Dobsonian mount	13
3.1.2 Servo motor with Pan-tilt	13
3.1.3 Shifting Calibration code to Software	14
3.2 Software	15
3.2.1 Python Based Software Module	17
3.2.2 Qt/C++ based software module	21
4 Calibration	27
5 ScriptEngine	29
5.1 Goto	30
5.2 Play	31
5.3 Wait	32
5.4 Laser On/Off	33
6 Future Prospects	37
Bibliography	39

Acknowledgements

I would like to express our heartfelt gratitude towards **Prof. Kannan M. Moudgalya** and **Prof. Dipankar** for offering and allowing us to carry out this project on OPEN SKY PLANETARIUM which aims at providing low-cost planetarium experience targeted at schools. I express my special thanks to our project guide **Ms. Inderpreet Arora**, and team members **Mr. Rupak Rokade**, and **Ms. Nivedita** were a great help in formulation and execution of this project. I would also like to express my sincere thanks to the entire IDL (Integrated Development Lab) team, who helped us with their knowledge and support. I also thank the lab in-charges for their help without which I would not have been able to concentrate on this project. I thank my parents without whose support this entire venture would not have seen the light of success.

Friday 22 July 2016

Declarations

I declare that this written submission represents my team's ideas along with mine. I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Vasudha Varadarajan
BITS Pilani, Hyderabad

Dhiraj Salian
MIT, Manipal

Yudhisther Bhargava
LNMIT, Jaipur

Abstract

Our project aims to support a number of schools that can encourage and motivate the students to look up and know their skies. The knowledge of night sky is shockingly low among most adults, attributed to the lack of stargazing experiences back in childhood. Planetariums in cities are mostly once-in-a-lifetime visit, unless one is very motivated. 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.

1 Introduction

Open Sky Planetarium is an open source educational tool that uses Stellarium, a software that simulates stars in real-time and that can be used as on-screen planetarium. The tool would consist of a laser pointer to point at a specific object in the sky controlled by an application supported by Stellarium.

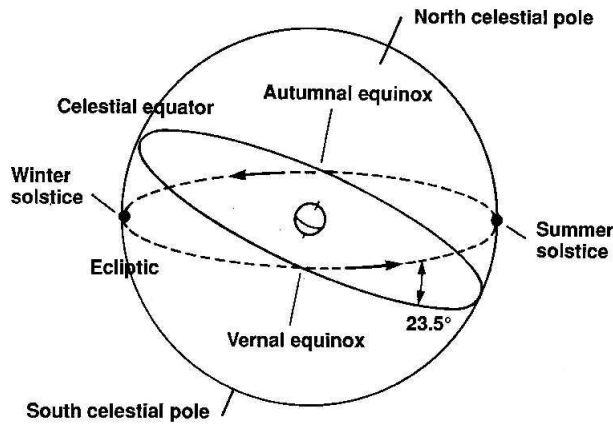
The user needs to place the pointer in an open field/ground with minimal obstruction on a clear night. After calibrating, the pointer is made to point at various objects and audio clippings, which would play in the background, describe the objects being pointed at.

1.1 Aim of the project

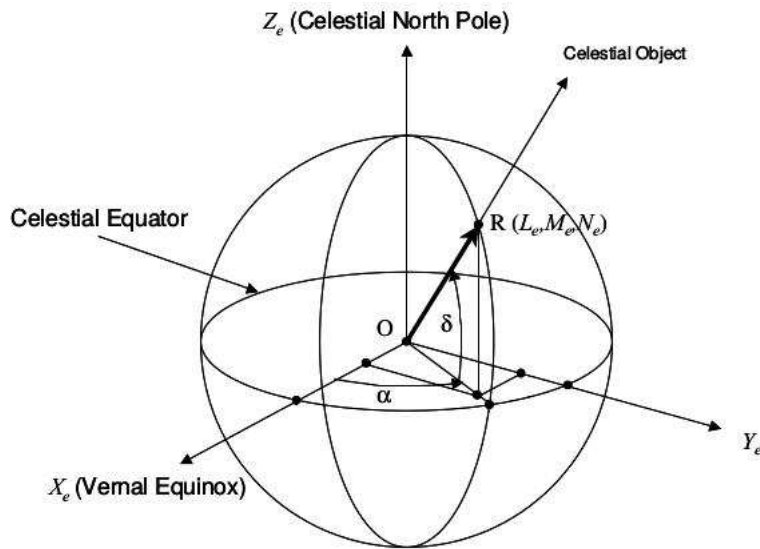
The primary aim of the project is to bring the planetarium experience home to every kid who looks up at the sky in wide-eyed wonder. This is achieved by making the equipment low-cost, thus making it affordable and accessible to all schools to play a planetarium show in their own locality. We also aim to keep the operation of the equipment and the application as user-friendly as possible.

1.2 Definitions to some terms used

celestial equator is the equator around the hypothetical celestial sphere around the sphere of the earth, parallel to the equator defined on the earth.

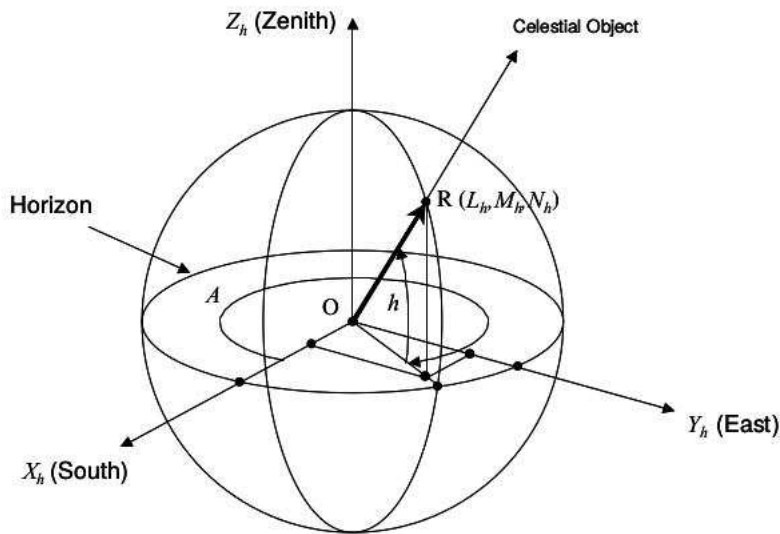


equatorial/ra-dec coordinates is a system of coordinates to describe the position of stars with respect to the centre of the earth, imagining the entire set of visible stars to be placed on the celestial sphere on the earth, with the coordinate $ra(\alpha)$ measuring the angle subtended by the north-south line on a point (analogous to longitude) and $dec(\delta)$ measuring the angle of the horizontal plane through the centre of the earth and the celestial equator (analogous to latitude). These are global as the coordinates of a point in the night sky doesn't change with a change in the position of observation on the earth.



horizontal/alt-az coordinates is a system of coordinates that uses the observer's location as the centre and plane with respect to which the altitude angle(h) above the plane, and the azimuth angle(A),

parallel to the plane but away from the reference line, is measured.



Dobsonian telescope mount is one that employs horizontal system of coordinates to point the telescope to an object specified by its horizontal coordinates.

equinox is a moment in time at which the vernal point, celestial equator, and other such elements are taken to be used in the definition of a celestial coordinate system.

J2000 is the system where equinox is taken to be 12:00 midnight, 1st January, 2000.

DDA or digital differential analyzer is an algorithm that moves between two points along at least two orthogonal directions (if 2D) incrementally, so as to move linearly along the two directions. This is done by finding the lowest common multiple between the distances to be traversed along the two directions. The steps are obtained by dividing the distance into smaller distances and these are traversed iteratively.

2 Tools for Software

2.1 Stellarium

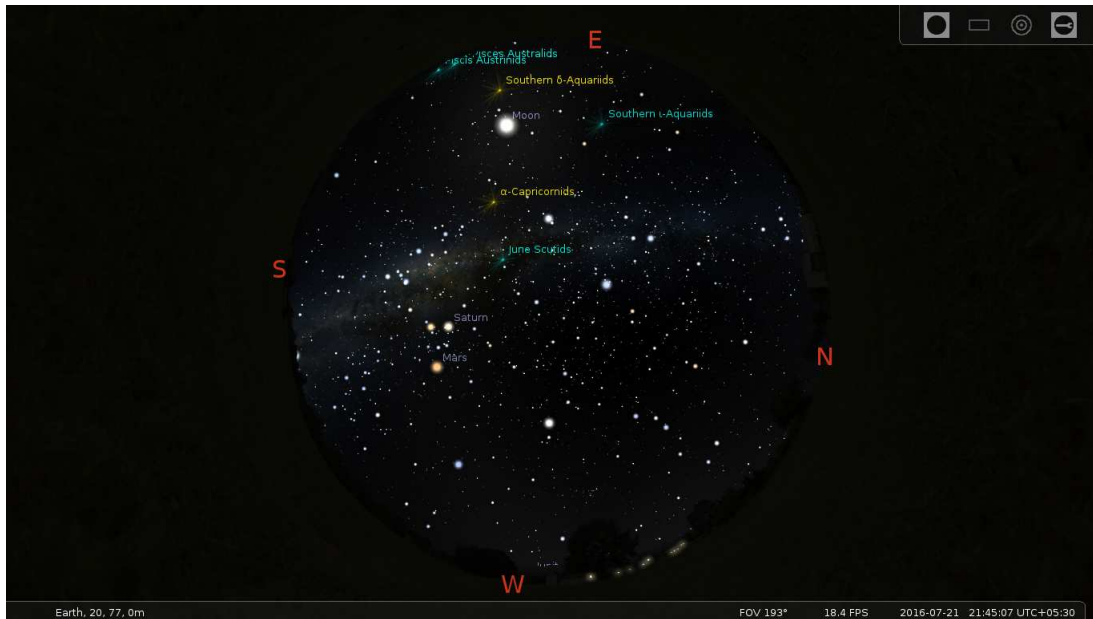


Figure 2.1: Stellarium software GUI-1

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.



Figure 2.2: Stellarium software GUI-2

There are also various sky cultures so that people from all parts of the world can use the software hassle-free. [2]

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*.

2.1.2 Scripting API

Stellarium also offers a Scripting API called StelMainScriptAPI makes it possible to write small programs within Stellarium to produce presentations, set up custom configurations, and to automate repetitive tasks. This feature is based on Qt Scripting Engine. The core scripting language is ECMAScript, giving users access to all basic ECMAScript language features such as flow control, variables string manipulation and so on. Interaction with Stellarium-specific features is done via a collection of objects which represent components of Stellarium itself.

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.[10][11][12]

2.2.2 Qt Creator

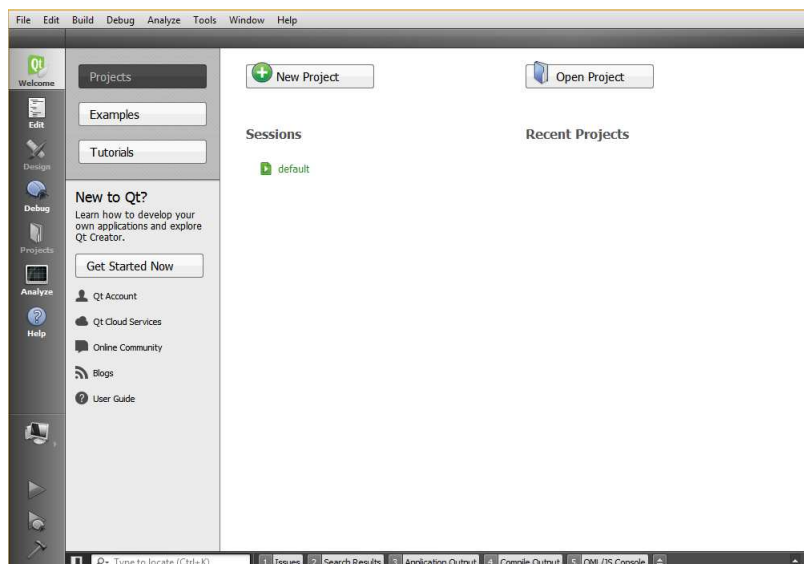


Figure 2.3: Qt Creator IDE

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.[3] It includes a visual debugger and an integrated GUI layout and forms designer. The editor's features include syntax highlighting and autocompletion, but purposely[4] not tabs (although plug-ins are available[5][6]). Qt Creator uses the C++

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.

2.3 PyQt

PyQt is a python wrapper around graphic library for C++. PyQt is a very powerful tool as it contains various libraries for a variety of GUI functionalities. It is cross-platform.[3]

PyQt has been used to create a sufficient GUI in the stand alone application. The GUI initially starts up when the python file `laser_control_main.py` (see below) is run. In the configuration mode, the user has to set references for creating a transformation matrix to extract horizontal coordinates and time (needed for a Dobsonian mount) from the equatorial coordinates (obtained from Stellarium). In tracking mode, a feedback is sent at regular intervals.

3 Modules

Formulating a working model of a moving laser pointer is a collaboration of hardware and software modules. Taking cues from an attempt to computerise the control of a telescope mount with Arduino through Stellarium[1], we made slight changes to the code to test with a crude two stepper-motor system prepared by the hardware team.

3.1 Hardware

The hardware underwent change in its structure, components and code as the project gradually progressed. In total there were three significant changes which are mentioned below.

3.1.1 Stepper motor with Dobsonian mount

The 1st phase of the hardware used a stepper motor system to drive a green laser pointer on a Dobsonian mount controlled through an Arduino board connected to the USB port of the system with Stellarium installed. Since the mount requires two independent coordinates given as altitude and azimuth in the horizontal system, two stepper motors provide the movement along the two axes. The transformation between the two coordinate systems viz equatorial and horizontal, is burnt into the hardware, as the input coordinates for controlling the movement of the laser pointer is extracted from Stellarium as equatorial coordinates.

3.1.2 Servo motor with Pan-tilt

The 2nd phase of the hardware is the current build of the hardware in terms of mechanical structure and component. It uses a Servo Motor

system to drive a green laser pointer on a Pan-tilt controlled through an Arduino board connected to the USB port of the system with Stellarium installed. Similar to the 1st Phase of the hardware, it requires two coordinates for the two axes which are provided by Pan-Tilt and Servo Motor along the two axes. The transformation between the two coordinate systems viz equatorial and horizontal, is burnt into the hardware, as the input coordinates for controlling the movement of the laser pointer is extracted from Stellarium as equatorial coordinates.

3.1.3 Shifting Calibration code to Software

The 3rd phase of the hardware was mainly a change in its code. Initially the calculations for calibration were done in the hardware. The Calibration Code was shifted from hardware to software. This change was brought into effect to better utilize the Arduino memory in its coordinate calculation. Since CPU has more power and speed, all mathematical calculation such as matrix transformation was shifted to CPU in view that it can be easily improved in future if the calculations become lengthy.

3.2 Software

This section explains the code used to implement the Open Sky Planetarium application along with Stellarium and the connected pointer device. The flow of control between modules of the Open Sky Planetarium is elucidated below:

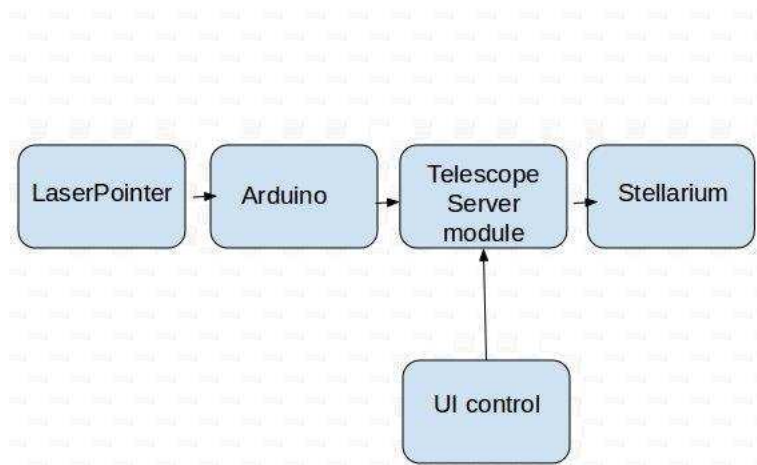


Figure 3.1: Flow of the Software Code

The software module also underwent changes as the project gradually progressed. From a stand-alone application to dynamic plugin integrated into Stellarium, the software evolved into more stable and complete software as foreseen before the start of the project.

The stand-alone application was inspired from the GitHub repository Arduino-Telescope-Control which is a python based application utilizing PyQt library for its GUI. This repository acted as the base for our project for the duration of the software phase where it was being developed as a stand-alone application.

After successful implementation of the python based version of the software and few tests, it was decided to integrate it into Stellarium. Since Stellarium is based on Qt/C++, the python based version was slowly being converted to Qt/C++ version. Qt Framework with its wide variety of API's under QtCore was used for such conversions. Serial port communication, Regular Expressions, TCP server, etc are some of the API's provided by Qt Framework under QtCore which are extensively used in the Qt/C++ based implementation of the software.

The python based software module used to handle the instantiation of a telescope server, and it's communication with the other modules: Arduino, user interface, and Stellarium. A telescope server module was being instantiated at localhost:port and for serial communication, serial port was specified by the user in GUI as the corresponding USB port at which the Arduino board is connected. The server module communicates with Stellarium through the Telescope Control Plugin in Stellarium. A user interface is created using PyQt for the users to control the telescope server and thus the movement of the laser pointer.

The Qt/C++ based software module doesn't require telescope server. Since the telescope server was only used to locate a star in stellarium and send the coordinates to the stand-alone application (python based software module) , including it in dynamic plugin (qt/c++ based software module) is no longer a necessity. The position of stars and other planetary bodies could be directly returned by Stellarium classes and modules. Also the coordinates transformation functions are directly available in Stellarium, therefore a separate class was no longer needed for coordinate transformations.

3.2.1 Python Based Software Module

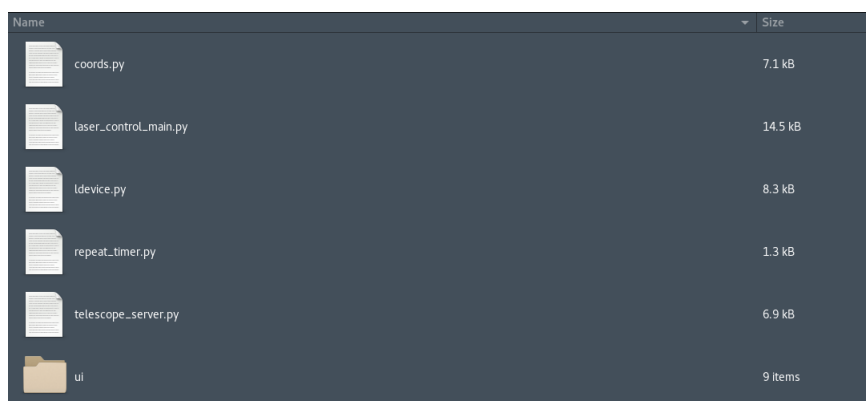


Figure 3.2: Python based software module gui

Python code written provides the user interface, instantiates a telescope server instance, and handles communication via the USB serial ports. The code has been split into several modules that have specific functionalities as elucidated below.

3.2.1.1 Software Code

File Structure

A screenshot of a file explorer window showing the file structure of the software code. The window has a dark theme and displays a list of files and folders. The files are listed with their names and sizes, and a folder named 'ui' is also shown.

Name	Size
coords.py	7.1 kB
laser_control_main.py	14.5 kB
ldevice.py	8.3 kB
repeat_timer.py	1.3 kB
telescope_server.py	6.9 kB
ui	9 Items

Figure 3.3: File Structure of Software Code for Python Based Software Module

coords.py

This module has a lot of functions to aid in conversions in the format used by Stellarium and the device, such as conversion of angle in hours-minutes-seconds to degrees-minutes-seconds or radians, or vice versa. The format of right ascension as received from stellarium is hours-minutes-seconds, while declination is degrees-minutes-seconds, with a precision of two points after decimal.

ldevice.py

Uses serial package from Python to communicate with the device attached through the serial port. This is achieved through a GUI implemented using QtCore and QThread. Asyncore and socket modules are also used to implement a socket handler to handle the input/output operations with the device. A class called Laser_Dev is derived from QThread class which contains a set of read/write functions that have different functionalities, such as initialization of the device, obtaining/sending coordinates for moving the laser to a particular point, to switch on/off the laser etc.

repeat_timer.py

Uses logging and threading packages of Python. It periodically executes a function, with a particular interval period, a certain number of times. It defines a class Repeat_Timer that derives from Thread class and is responsible for periodic execution of functions using run() for a specific interval and cancel() functions.

telescope_server.py

Uses logging, asyncore, socket modules and Thread class to implement a Telescope Server. It has two classes: Telescope_Channel and Telescope_Server. This uses the modules listed above to begin a telescope server that can be connected through the telescope plugin in Stellarium.

A terminal window with a black background and green text. The text shows a user navigating to a directory and running a Python script. The output of the script is displayed on the next line.

```
vasudha@vasudha-Inspiron-5558:~$ cd Downloads/OpenSkyProject/Arduino-Telescope-Cont
rol-master/main/python/
vasudha@vasudha-Inspiron-5558:~/Downloads/OpenSkyProject/Arduino-Telescope-Contr
ol-master/main/python$ python telescope_server.py
telescope_server.py: run - INFO: Telescope_Server is running...
```

Figure 3.4: telescope_server.py instantiates a telescope server and runs it. This can be connected to Stellarium via the telescope control plugin.

laser_control_main.py

This module implements a GUI to bring together all the modules

mentioned above controlled by user input. It defines a single class called LaserControlMain which derives from QMainWindow class of QtGui. This starts GUI window along with a Telescope server.

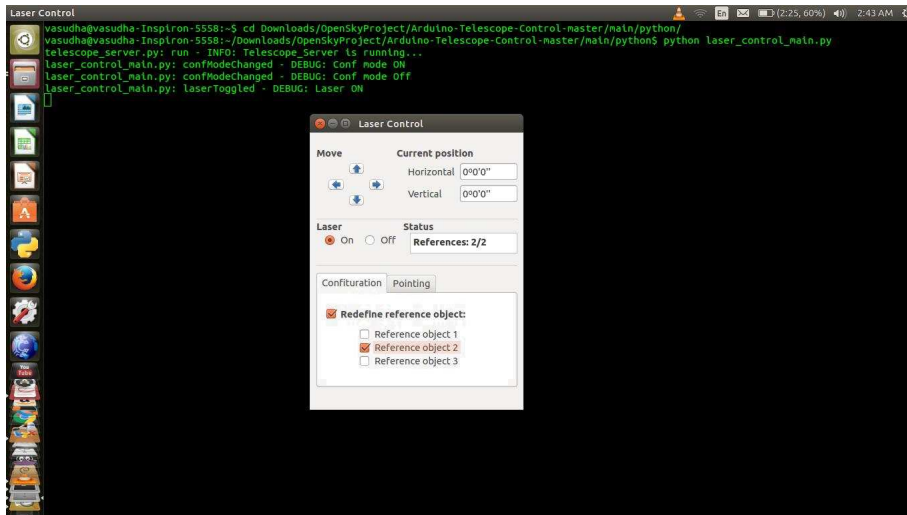


Figure 3.5: Connecting the device by choosing from the list of ports

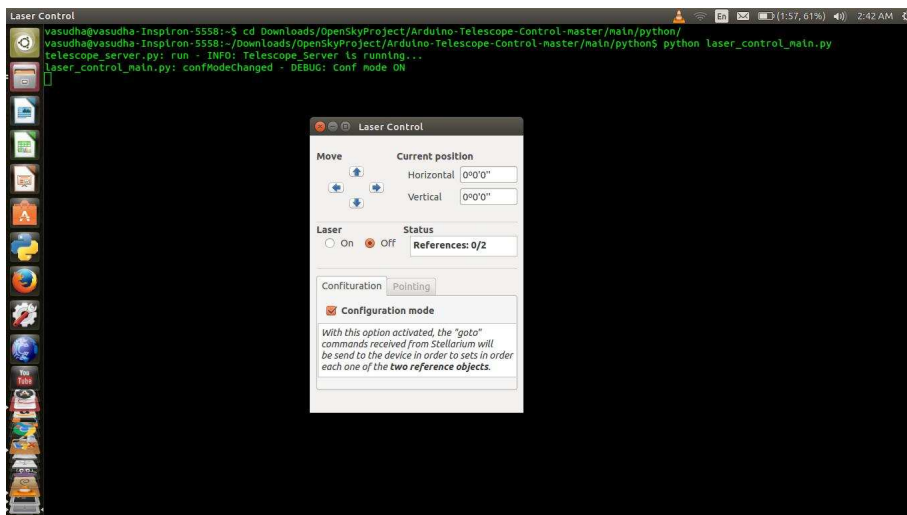


Figure 3.6: In configuration mode

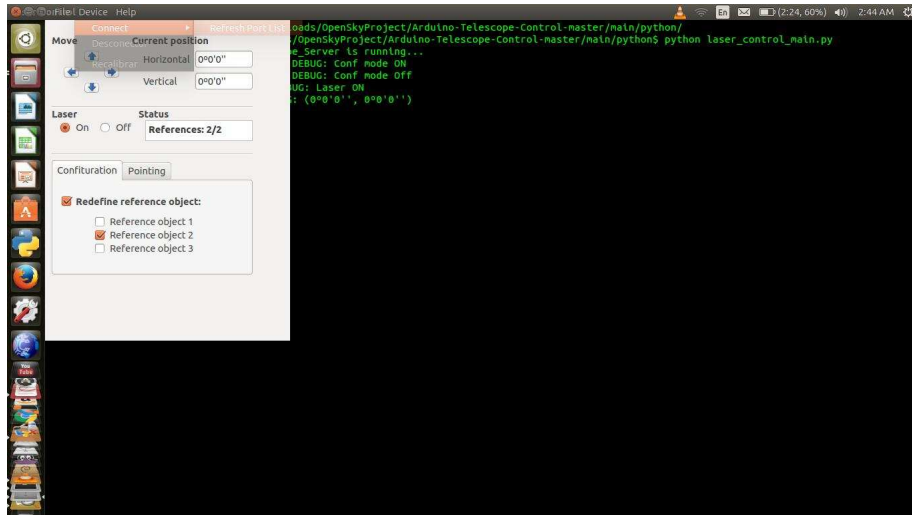


Figure 3.7: When calibration is done once again, we choose each reference on Stellarium and move the laser pointer accordingly.

3.2.1.2 Arduino code

AxesLib

AxesLib.h defines a class called AxesLib which will set the axes for the laser pointer to move about, and control the on/off state of the laser. The movements are based on the Digital Differential Algorithm (DDA). AxesLib.cpp defines the methods declared for the class AxesLib and uses AxesLib.h header file.

CoordsLib

A class called CoordsLib is defined which contains various attributes for storing various values essential in calculating the transformation matrix for calculating the horizontal coordinates from the equatorial coordinates. The calibration is explained in greater detail below.

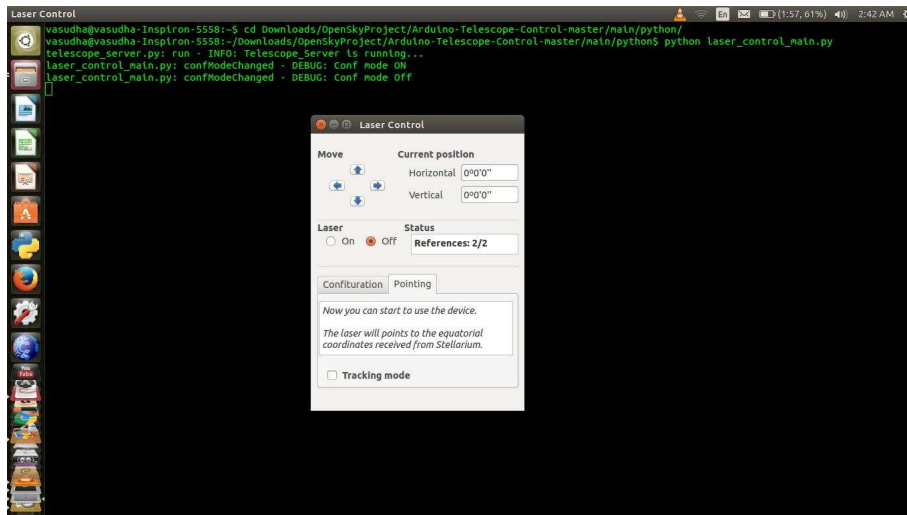


Figure 3.8: When configuration mode is unchecked, references are assumed to be set and pointing is activated.

3.2.2 Qt/C++ based software module

3.2.2.1 Plugin Code

File Structure

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. The files

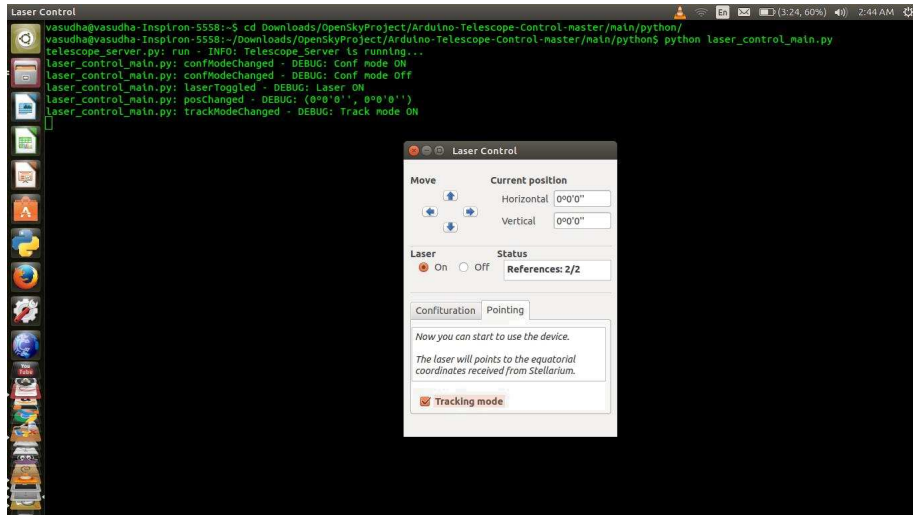


Figure 3.9: When tracking mode is checked in pointing, there is a feedback from the device pointer coordinates at regular intervals.

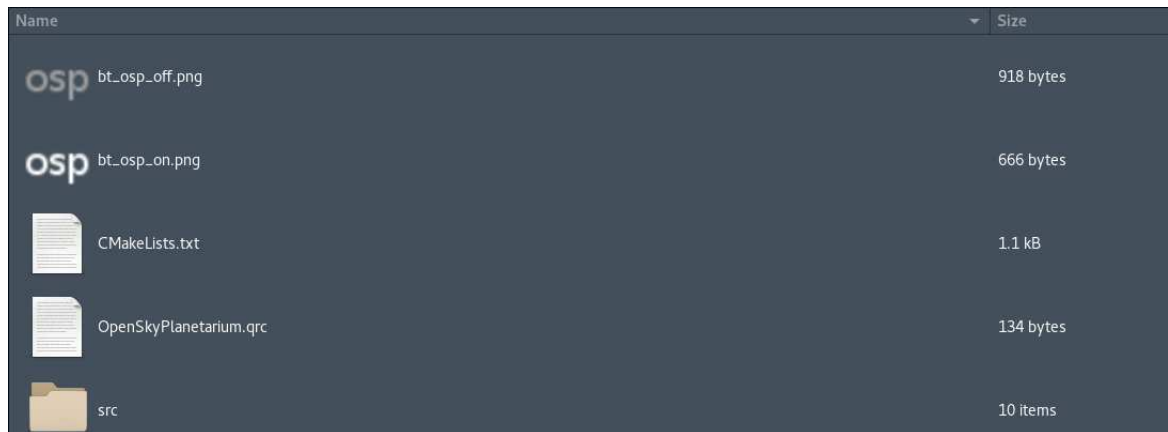


Figure 3.10: Open Sky Planetarium Root Directory

and directories included in src/ are represented in the Fig.3.10.

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.

Name	Size
Calibrate.cpp	6.2 kB
Calibrate.hpp	6.6 kB
CMakeLists.txt	1.0 kB
gui	3 Items
LaserDev.cpp	4.9 kB
LaserDev.hpp	1.7 kB
OpenSkyPlanetarium.cpp	4.2 kB
OpenSkyPlanetarium.hpp	2.0 kB
SerialCom.cpp	3.2 kB
SerialCom.hpp	1.4 kB

Figure 3.11: Open Sky Planetarium src/ Directory

SerialCom.hpp/SerialCom.cpp

Serial Communication between the arduino and the plugin is facilitated by SerialCom class. The Serial Communication approach used for communication is a blocking master approach. 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 and post is send to the arduino, which in turn gives back response to these commands.

OpenSkyPlanetarium.hpp/OpenSkyPlanetarium.cpp

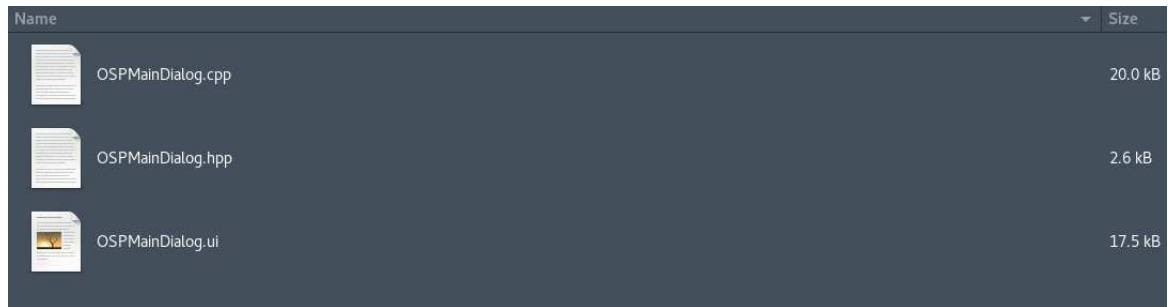
The main file of the plugin. Links the Gui file with Stellarium. This class has a predefined format and no changes are required to this format to integrate the plugin with Stellarium.

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.



Name	Size
OSPMainDialog.cpp	20.0 kB
OSPMainDialog.hpp	2.6 kB
OSPMainDialog.ui	17.5 kB

Figure 3.12: Open Sky Planetarium src/gui/ Directory

OSPMainDialog.ui

The xml file of the gui is OSPMainDialog.ui. This file is created using Qt Creator IDE. The ScreenShots of the GUI are attached below:

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.

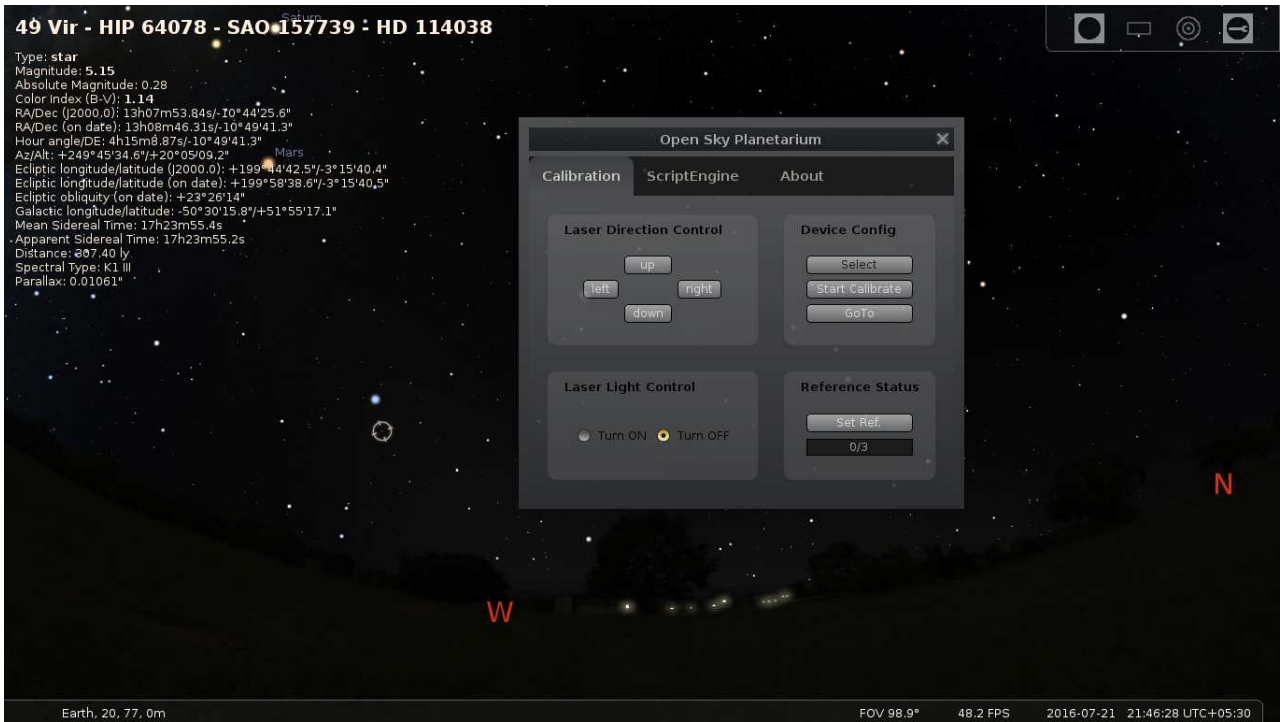


Figure 3.13: OpenSkyPlanetarium Calibrate Window



Figure 3.14: OpenSkyPlanetarium ScriptEngine Window

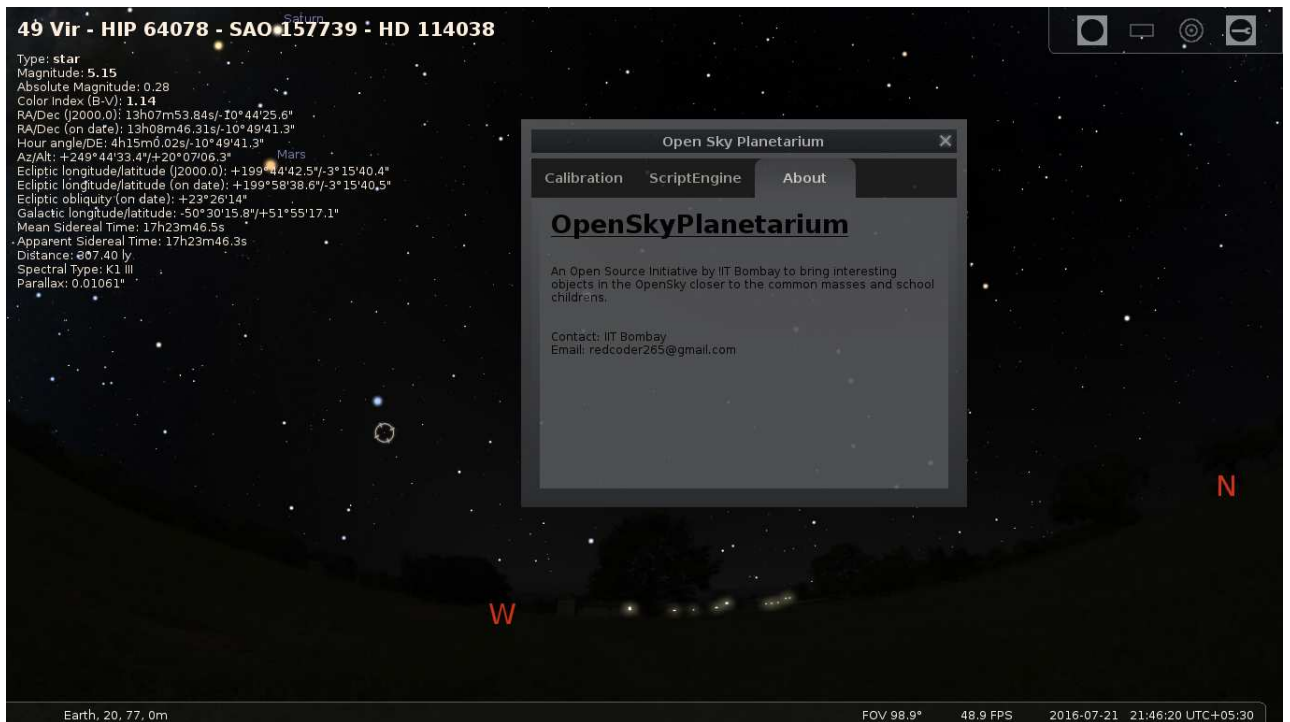


Figure 3.15: OpenSkyPlanetarium About Window

4 Calibration

Initially calibration was being done as a part of the Arduino code, i.e. the transformation matrix was calculated and stored in the hardware module, as opposed to obtaining the transformation matrix in the software, transforming the coordinates, and deploying to the hardware for just pointing.

But with updates to the hardware and the software code, later it was decided to go with obtaining transformation matrix in the software. This increases the chances of including more accuracy to the calibration by including more calculations and this would provide more memory to arduino which can be used to obtain better accuracy in its mechanical positioning calculations.

So now the software only sends horizontal coordinates to the arduino. These horizontal coordinates are calculated using equatorial coordinates of the star and transformation matrix obtained after calibration.

For calibrating, the position of the telescope is needed. The telescope positions are calculated in arduino and sent to plugin for setting reference when the command “post” is sent to arduino. The telescope position is obtained in the following format “t_[x]_[y]” where [x] is azimuth and [y] is altitude of telescope.

Mathematical Explanation for Calibration:

Let us assume that, for a particular point, ra, dec, alt and az are given by α, δ, h and A respectively. Time is also taken into consideration since the objects in the night sky are in motion. Since we have 3 variables, we would require to measure the coordinates at least three orthogonally independent points, or two points and their cross product.

In Cartesian space, let $[L, M, N]$ be the equatorial coordinates, and $[l, m, n]$ be the horizontal coordinates. They are thus given by:

$$\begin{pmatrix} L \\ M \\ N \end{pmatrix} = \begin{pmatrix} \sin\delta \cos(\alpha - k(t - t_0)) \\ \sin\delta \sin(\alpha - k(t - t_0)) \\ \cos\delta \end{pmatrix}$$

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \begin{pmatrix} \sin(h) \cos A \\ \sin(h) \sin A \\ \cos A \end{pmatrix}$$

We need a transformation matrix T such that:

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = T \times \begin{pmatrix} L \\ M \\ N \end{pmatrix} \text{ where } T \text{ is } 3 \times 3 \text{ matrix.}$$

$$\begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{pmatrix} = T \times \begin{pmatrix} L_1 & L_2 & L_3 \\ M_1 & M_2 & M_3 \\ N_1 & N_2 & N_3 \end{pmatrix}$$

$$[T] = \begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{pmatrix} \begin{pmatrix} L_1 & L_2 & L_3 \\ M_1 & M_2 & M_3 \\ N_1 & N_2 & N_3 \end{pmatrix}^{-1}$$

Thus we calculate the transformation matrix. The inverse transformation matrix is obtained by inverting matrix T, which when multiplied by a set of horizontal coordinates in Cartesian space gives rise to equatorial coordinates. [4]

During initial build of the hardware ,the Arduino code CoordsLib.h and CoordsLib.cpp defined various functions in the class CoordsLib to facilitate the calculation of a transformation matrix with minimal error.

Instead of three references chosen, the code made transformation matrix with just two by using the third independent point as the vector product of the first two:

$$P_3 = P_1 \times P_2$$

$$P_3 = \begin{pmatrix} l_3 \\ m_3 \\ n_3 \end{pmatrix} = \begin{pmatrix} m_1 n_2 - n_1 m_2 \\ n_1 l_2 - l_1 n_2 \\ l_1 m_2 - m_1 l_2 \end{pmatrix}$$

The current build of the project includes calibration in software. The class Calibrate is used for calibration. It includes three reference calculation rather than two reference calculation as in CoordsLib. The more references used increases the accuracy of the transformation matrix.

The various matrix functions defined in Calibrate to enable calculation of transformation matrix are:

- inverse a matrix: `__inv()`
- set references (for each of the two references) : `__setR1()`, `__setR2()`, `__setR3()`
- set equatorial coordinates and horizontal coordinates in Cartesian coordinates: `__setEVC()` and `__set HVC()`
- set the transformation matrix : `__setT()`
- get horizontal coordinates (multiplying the transformation matrix by equatorial coordinates): `__getHCoords()`
- get equatorial coordinates (multiplying the transformation matrix by horizontal coordinates): `__getECoords()`

These functions help in deriving and storing the transformation matrix for subsequent pointing.

5 ScriptEngine

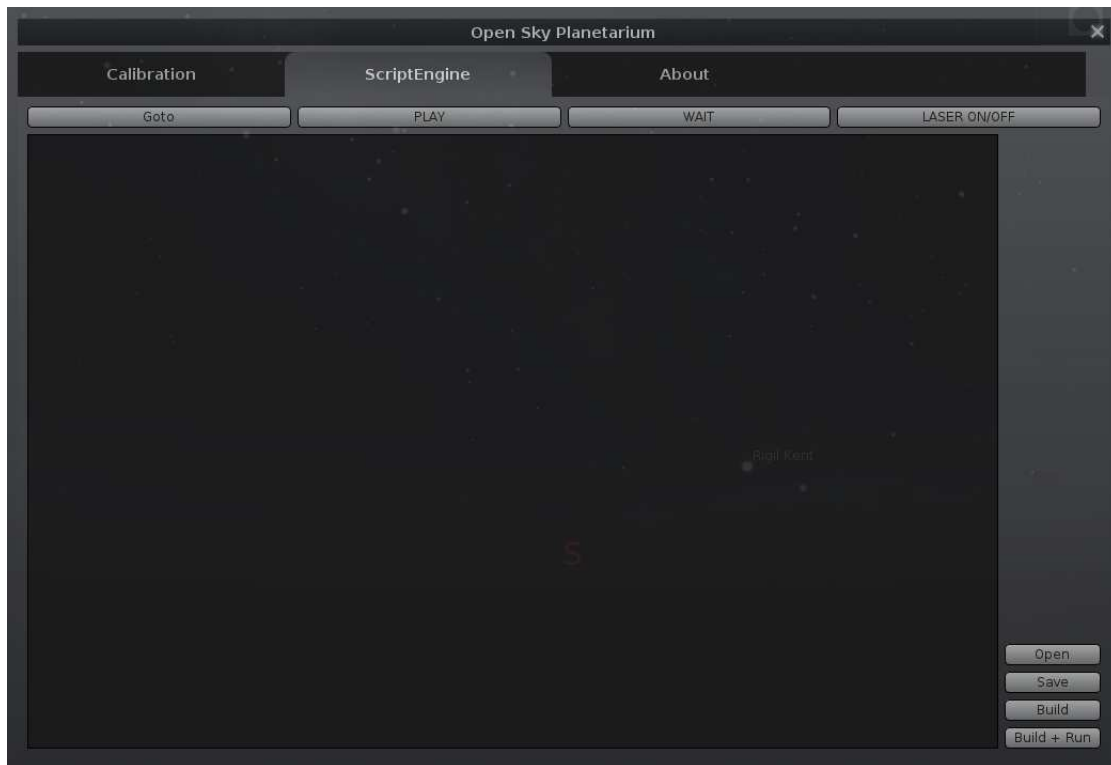


Figure 5.1: ScriptEngine

The Script Engine in Open Sky Planetarium is used to write and execute scripts in Stellarium. Since Open Sky Planetarium is being developed to reach to common masses and students, the scripting part of the ScriptEngine is kept very simple. It includes only four commands :

- Goto
- Play
- Wait
- Laser on/off

It is recommended to use GUI buttons to write scripts rather than typing the scripts manually. The GUI buttons for the above mentioned four commands have inbuilt input dialog which takes input in the required format of the script.

5.1 Goto

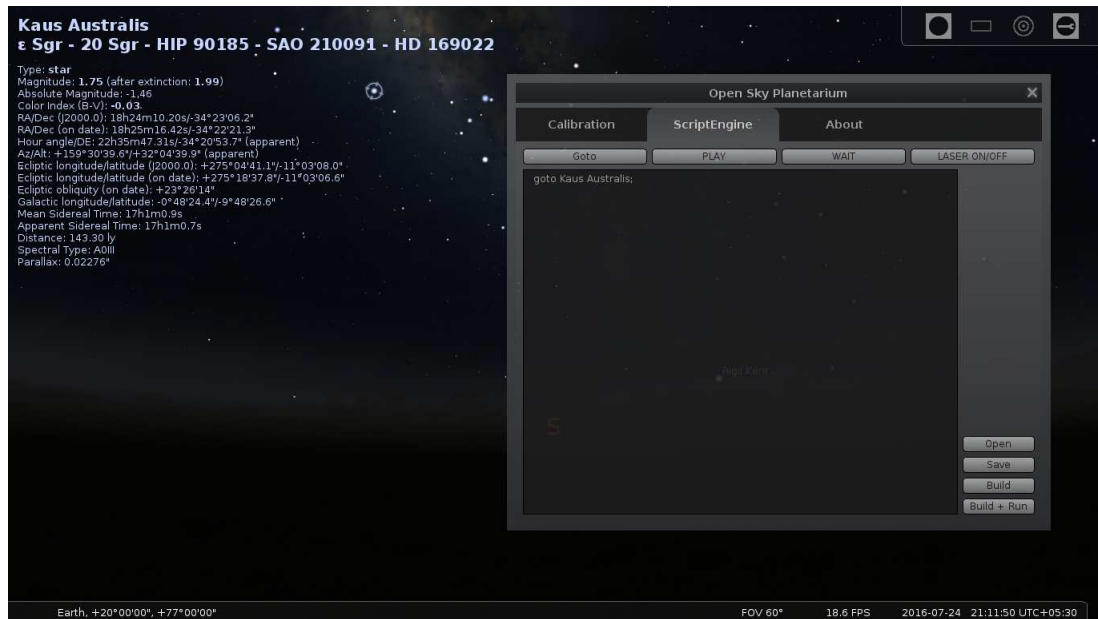


Figure 5.2: Goto Command Success

The goto command requires an object in stellarium sky to be clicked before clicking on goto button. A error message is shown if goto is directly clicked. The goto command syntax is as follows:

```
goto [star-name];
```

During compilation when goto command is encountered while parsing, the plugin finds the coordinates of the star from its name and stores it in a list which is accessed while executing the script.

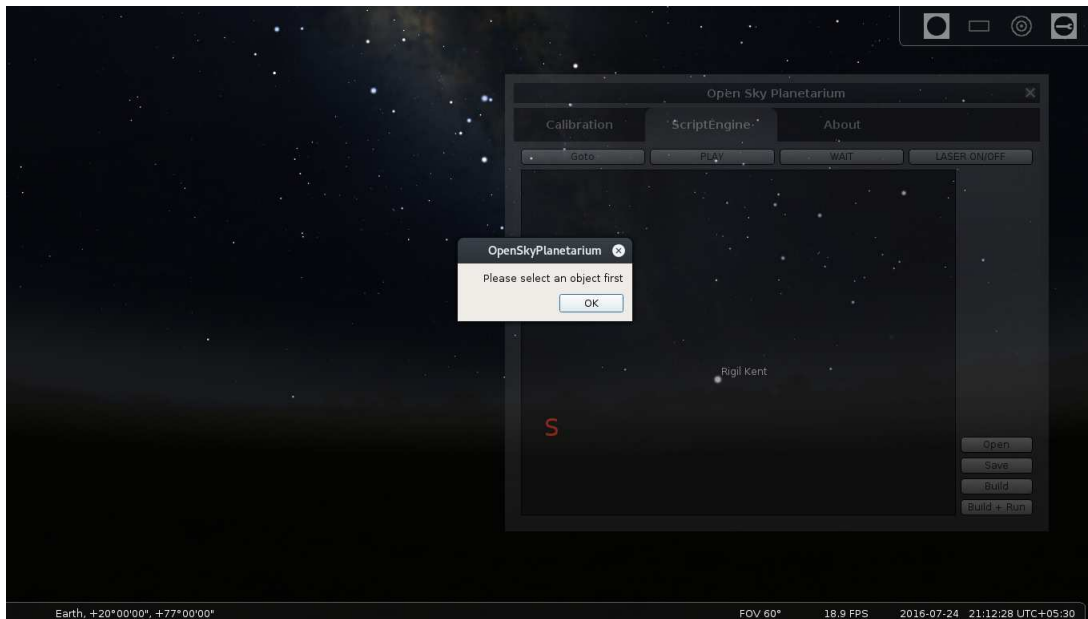


Figure 5.3: Goto Command Failure

5.2 Play

The play command is used to select the audio file. The audio file must be selected from the {User-Directory}/modules/OpenSkyPlanetarium/audio folder. The current build of the plugin uses QMediaPlayer to play audio. QMediaPlayer requires libgstreamer for playing audio files. Also stellarium is not enabled by default to use Media. Hence another opensource cross platform class should be found out for better portability. The play command syntax is as follows:

```
play [script-audio].mp3;
```

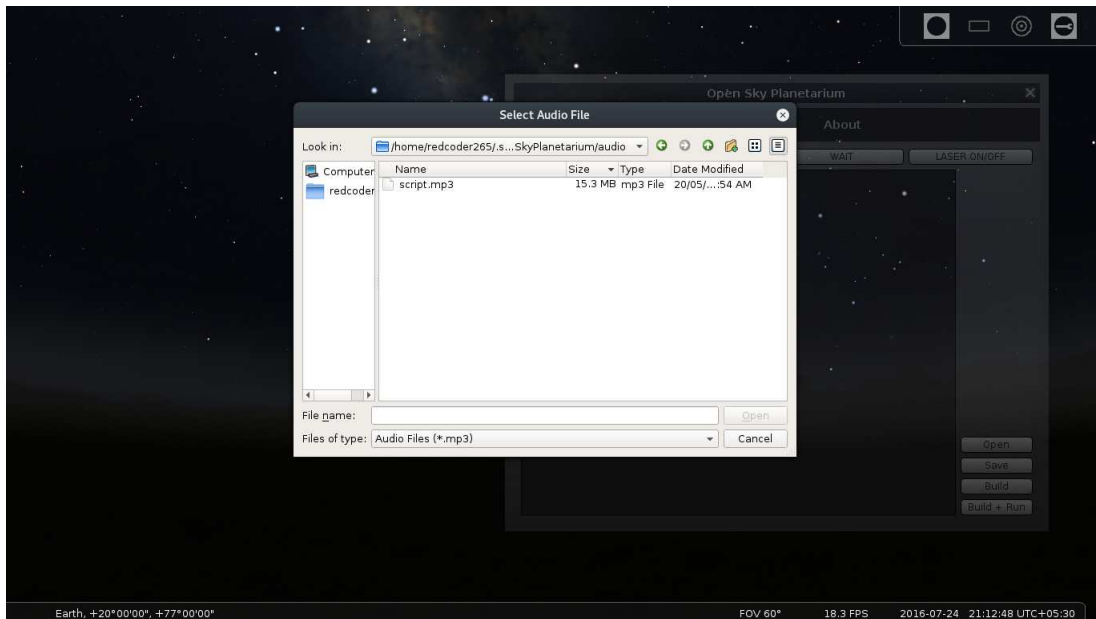


Figure 5.4: Play Command Success

5.3 Wait

The wait command is used to insert wait time in script. This command is required to synchronise the laser timing with the script audio. When clicked on wait button, input dialog pops up which asks for X minutes and Y seconds for wait time. The wait syntax is as follows:

```
wait [X]m[Y]s;
```

Here X is in minutes and Y is in seconds.

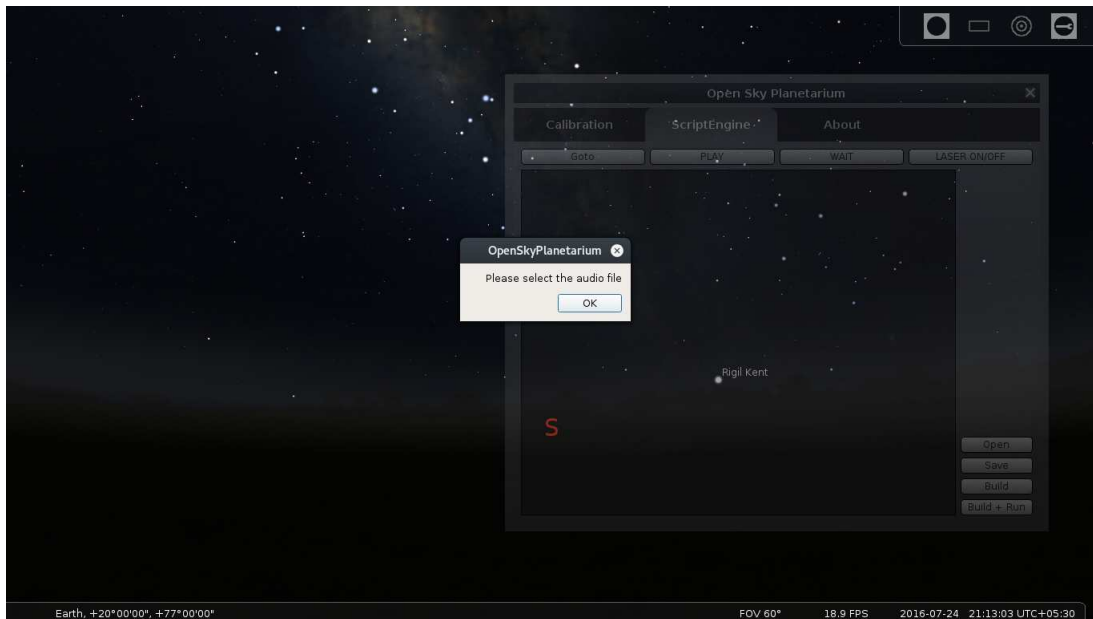


Figure 5.5: Play Command Failure

5.4 Laser On/Off

The Laser On/Off command button click opens an input dialog asking for “on” and “off”. The Laser command syntax is :

```
turn [ on / off ] ;
```

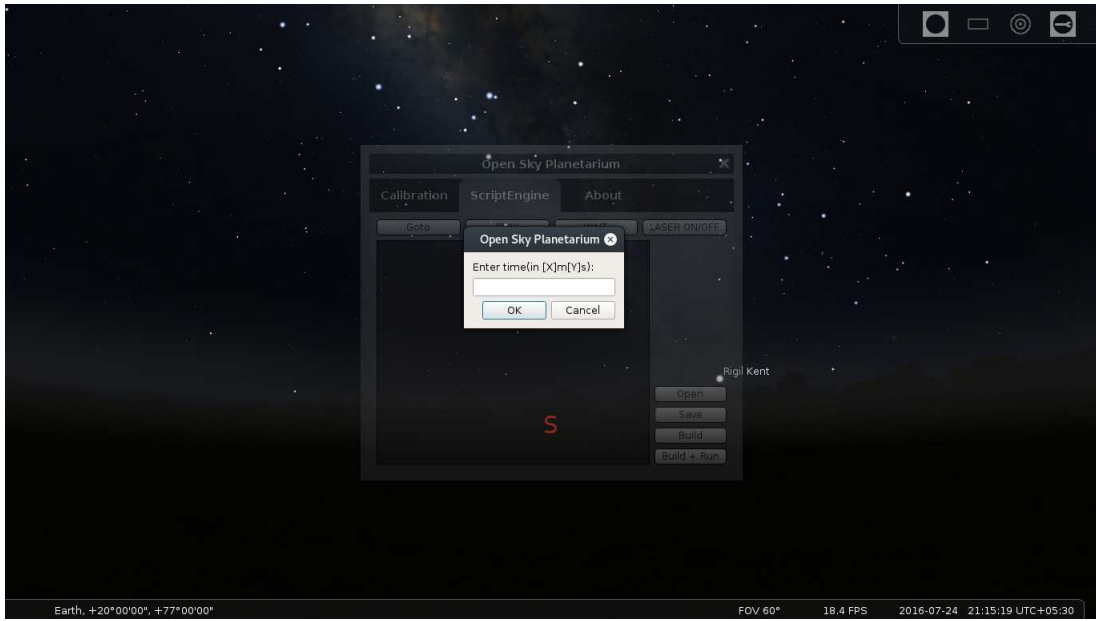


Figure 5.6: Wait Command Success

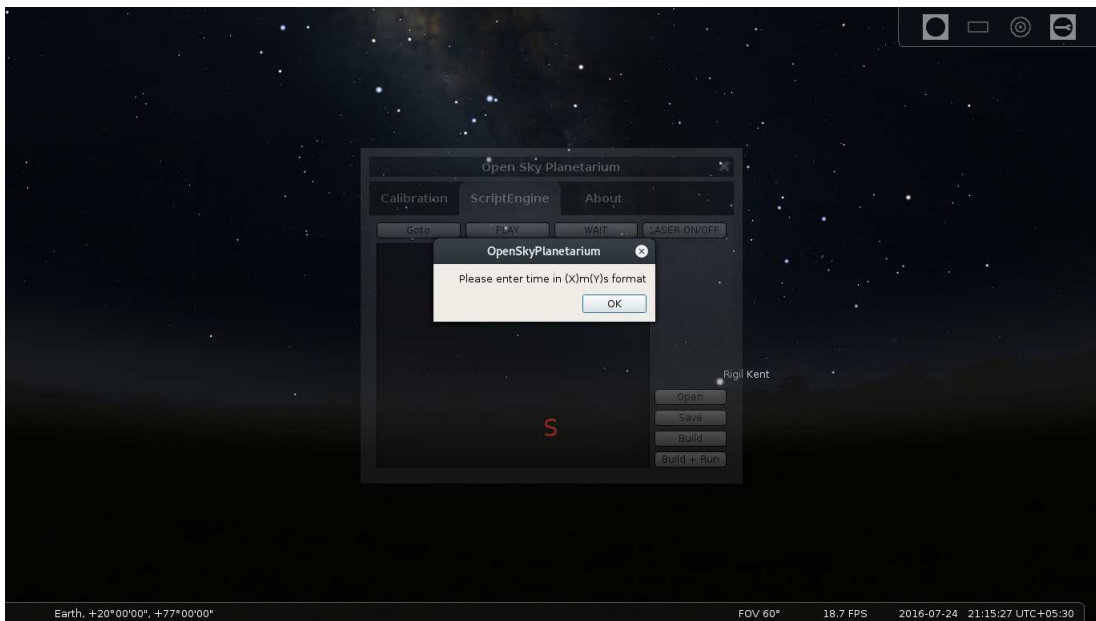


Figure 5.7: Wait Command Failure

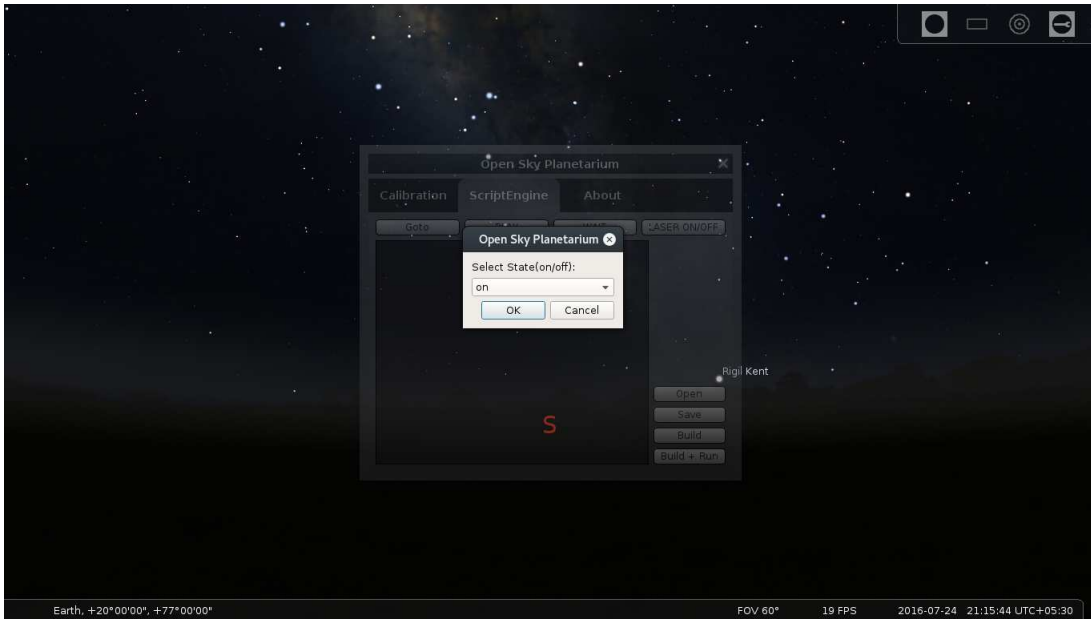


Figure 5.8: Laser Command Success

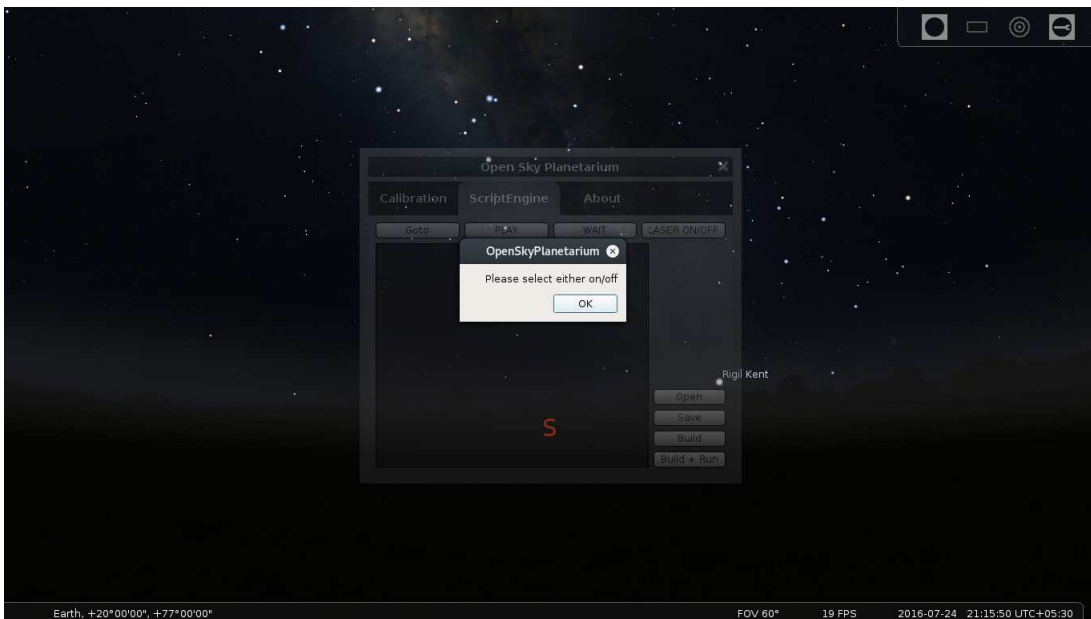


Figure 5.9: Laser Command Failure

6 Future Prospects

During the course of this project it was realized that the project is not as user-friendly as we would like in its current form, which is rather crude. To enable smoother user experience, calibration should be included as a part of Stellarium itself, as a dynamic plugin which could be downloaded by anybody at will and used. The stand alone application will have to be distributed separately along with the pointer equipment, which could cause some hassle. Also, the calibration takes a lot of time as the user would not know which objects to point at, or whether they are pointing at the right object. This could be cross-checked if the application was integrated into Stellarium.

So the project was slowly evolved to cater all the needs and a dynamic plugin was created as a result. Most of the features such as calibration, serial communication, script engine have been implemented in the plugin. But still some more tasks and enhancements are needed before making it available to the end user. The tasks remaining are :

- Implement OpenSource Cross Platform Audio Playing.
- Improve calibration by increasing number of references from three to four.
- Improve Serial Communication to make it more reliable.

Yet another improvement is regarding the movement of the laser pointer which has been restricted to 0° to 360° and back. This means that, in a planetarium show, the laser beam would trace out the distance between two stars to let the audience grasp the proximity of the objects in the night sky, but this need not be the smallest possible path as the pointer can move either clockwise or anticlockwise, once. A better algorithm to trace the smallest path could be used.

Time is a minor factor in astronomical calculations, especially since references are usually taken one after another in a short time span, and thus can be eliminated. In that case, instead of converting the horizontal/equatorial coordinates to Cartesian system and back, we could directly use the polar system and store the transformation matrix as a 2x2 matrix. This improves calculation speed and lowers the burden on the Arduino microcontroller which also needs to individually calculate each of the coordinates using a long series of arithmetic operations.

Bibliography

- [1] Arduino-Telescope control: A blogpost on a Spanish blog
- [2] Stellarium wiki
- [3] PyQt reference and installation
- [4] Toshimi Taki's monograph on matrix transformations
- [5] Stellarium Classes
- [6] Qt Docs