



# INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR

SSTP - 2019

---

## :TelescoGalvans: Telescope Automation

---

Aditya Tripathi	18110010
Dave Hari Manish	18110044
Joshi Devvrat Shailesh	18110076
Kumar Ayush Paramhans	18110089
Prof. Shanmuganathan Raman	Faculty Mentor

## Contents

<b>1</b>	<b>Preface and Introduction</b>	<b>2</b>
1.1	Abstract . . . . .	2
1.2	Keywords . . . . .	2
1.3	Introduction . . . . .	2
<b>2</b>	<b>Software Assembly</b>	<b>3</b>
2.1	Desktop Application . . . . .	3
2.2	Extracting Data From Stellarium . . . . .	8
2.3	Conversion of the extracted Data . . . . .	11
2.4	Setting up Raspberry Pi . . . . .	13
2.5	Raspberry Pi application . . . . .	13
<b>3</b>	<b>Electrical Assembly</b>	<b>18</b>
3.1	Motors and Drivers . . . . .	19
3.2	Connecting Motors to Raspberry Pi . . . . .	19
<b>4</b>	<b>Mechanical Assembly</b>	<b>20</b>
4.1	Gears and Base . . . . .	21
4.2	Motor Holders . . . . .	22
4.3	Rotation Mechanism . . . . .	23
<b>5</b>	<b>The Complete Assembly</b>	<b>25</b>
<b>6</b>	<b>Procedure to Use The System</b>	<b>25</b>
<b>7</b>	<b>Challenges Faced</b>	<b>31</b>
7.1	Precision-Conversion . . . . .	32
7.2	Precision-Mechanical . . . . .	32
7.3	Strength of gears . . . . .	32
<b>8</b>	<b>Budget and timeline</b>	<b>32</b>
<b>9</b>	<b>Pros and Cons of the model</b>	<b>32</b>
9.1	Pros of the model . . . . .	33
9.2	Cons of the model . . . . .	33
<b>10</b>	<b>Conclusion</b>	<b>33</b>
<b>11</b>	<b>Vote of thanks</b>	<b>33</b>
<b>12</b>	<b>Credits and References</b>	<b>34</b>

# 1 Preface and Introduction

## 1.1 Abstract

The discovery of 'telescope' was an important event in the history of mankind. It enabled humans to understand the universe in a better way. With the development in technology, using the telescopes like 'Hubble' and 'Kepler', we have found thousands of galaxies, celestial bodies like stars, planets and even black holes and also witnessed some great events like a supernova.

Because of the commercialization of telescopes, many people observe the night sky at a lower level. As the celestial bodies are continuously moving, it is difficult to set the telescope in an appropriate position and also to trace the object. The system developed by us truncates the tedious work to set the telescope and also tracks the celestial object, letting the observer enjoy night gazing without any interruptions.

## 1.2 Keywords

Software system, Raspberry-pi, Planetary gear stepper motors, Driver for the motors, DC regulated power supply, Telescope, Azimuth plane, Earth's Mean Equator (J2000).

## 1.3 Introduction

We have developed an "Automated Telescope System". A telescopic system is qualified as an automated system when it makes observations without the intervention of humans. Human intervention only comes at the initial stage in a bid to initiate the process.

The project was done under the mentorship of **Prof. Shanmuganthan Raman**.

We have developed a desktop application for our system. The software extracts data from the 'Stellarium', an open-source planetarium software. The data is extracted in the form of coordinates of the celestial body. The coordinates extracted are with respect to Earth's Mean Equator (J2000). These coordinates are sent from our desktop application to Raspberry pi. The code running in raspberry pi converts the coordinates from RA, Dec to Alt, Az. On receiving input from the software, the drivers send the command to the motors to turn accordingly.

We have given two degrees of freedom to our telescopic system, one across the 'azimuth plane' and the other across the 'elevation plane'.

## 2 Software Assembly

This section will explain the software aspects of the project.

### 2.1 Desktop Application

We have made a desktop application on electron platform. The purpose of the application is to maintain a user friendly interface and provide a seamless connectivity across different layers of the software and hardware part.

Here are some screen-shots of the software.

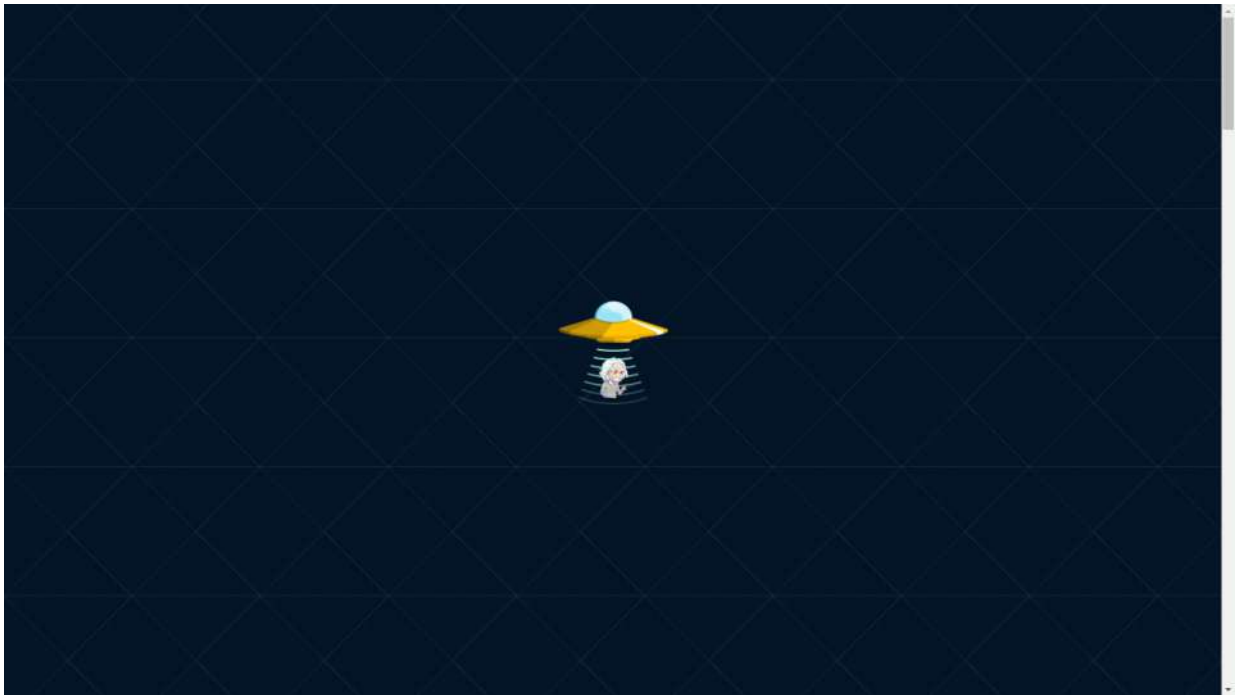


Fig. 2.1



Fig. 2.2

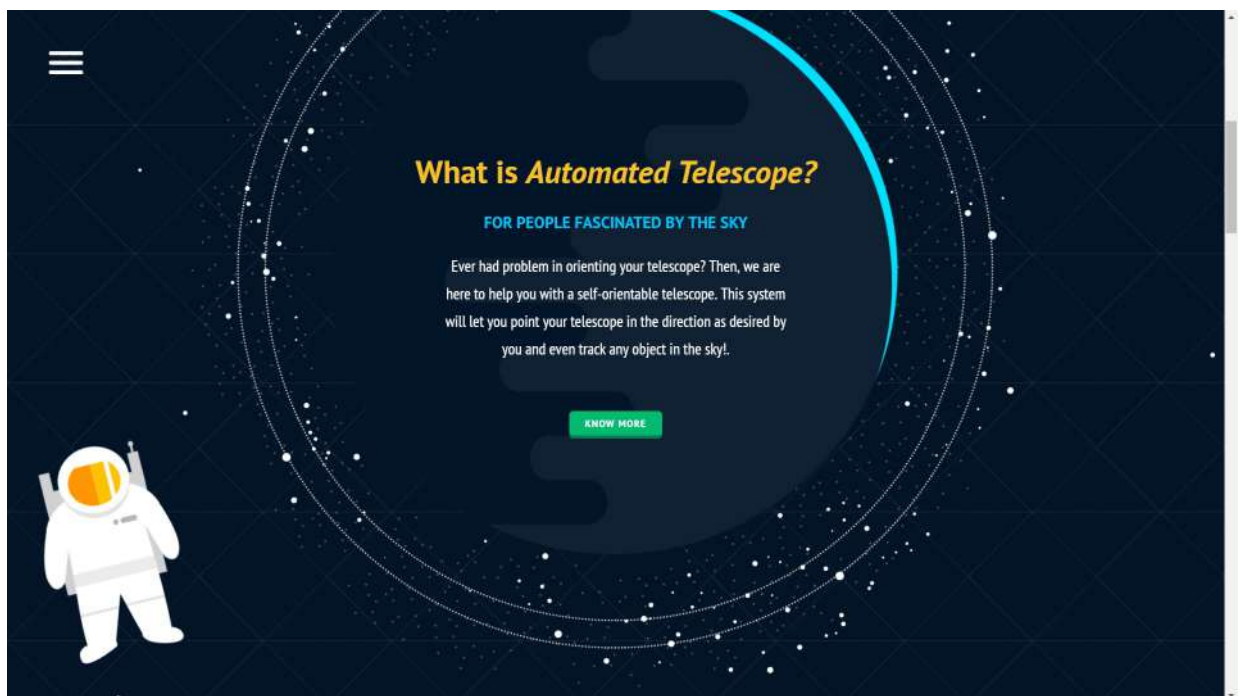


Fig. 2.3

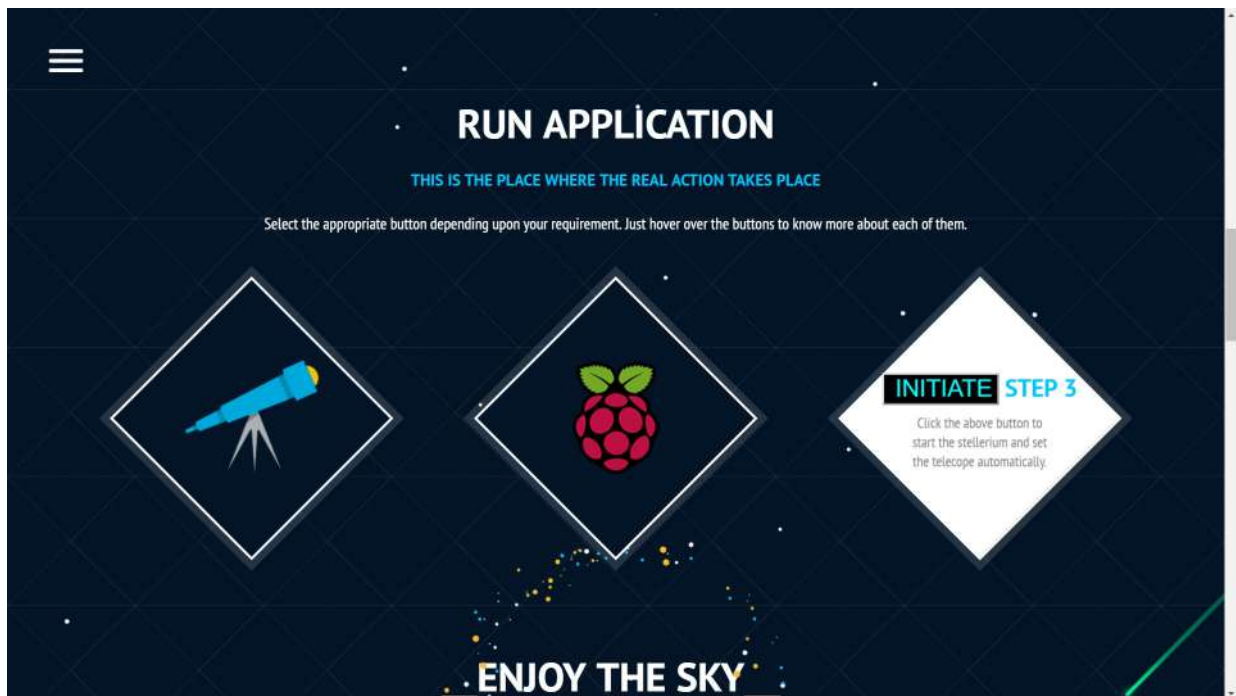


Fig. 2.4

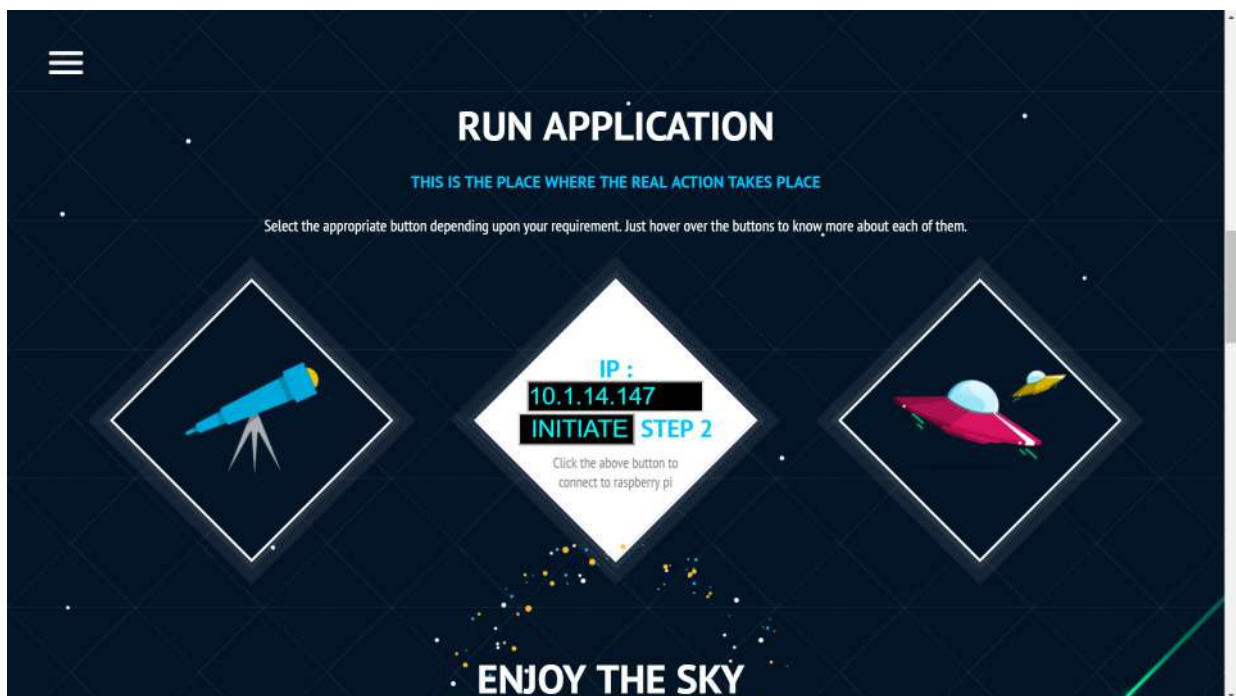


Fig. 2.5



Fig. 2.6

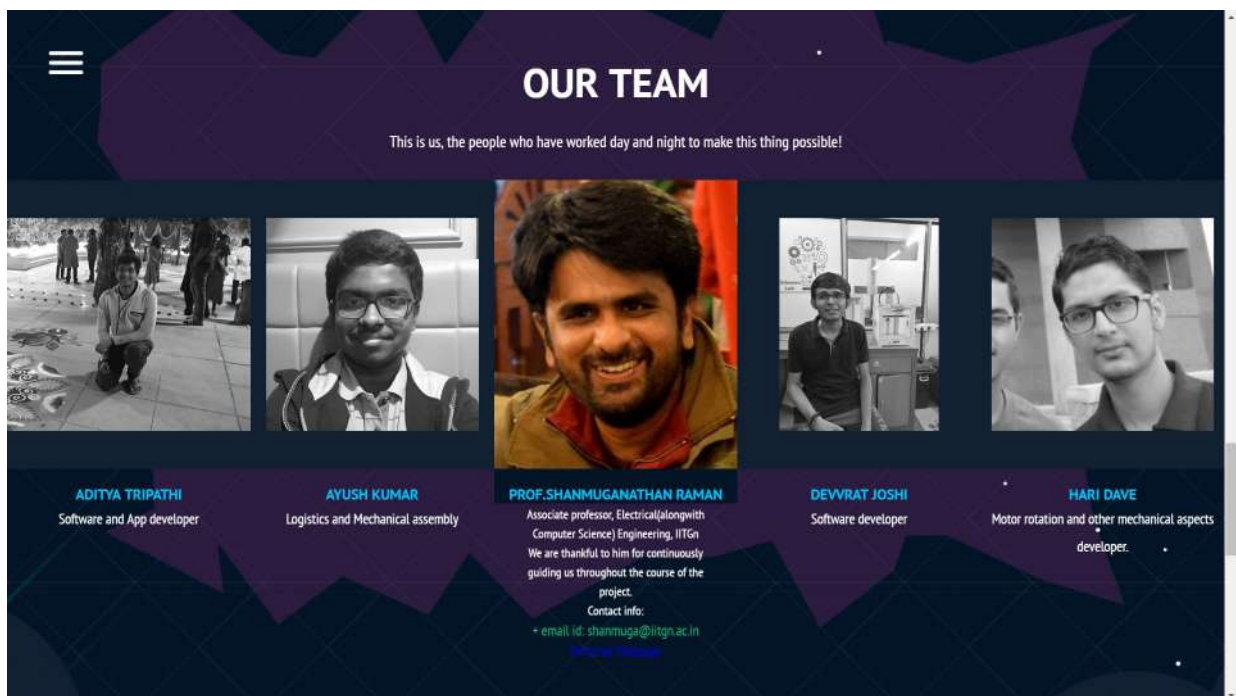


Fig. 2.7

It can be seen in fig 2.3 that there are three buttons present. Each button has a specific purpose.

- Button one will start the telescope server which will help in connecting the telescope to stellarium software.
- Button two will connect us to Rpi which will allow us to send the data extracted

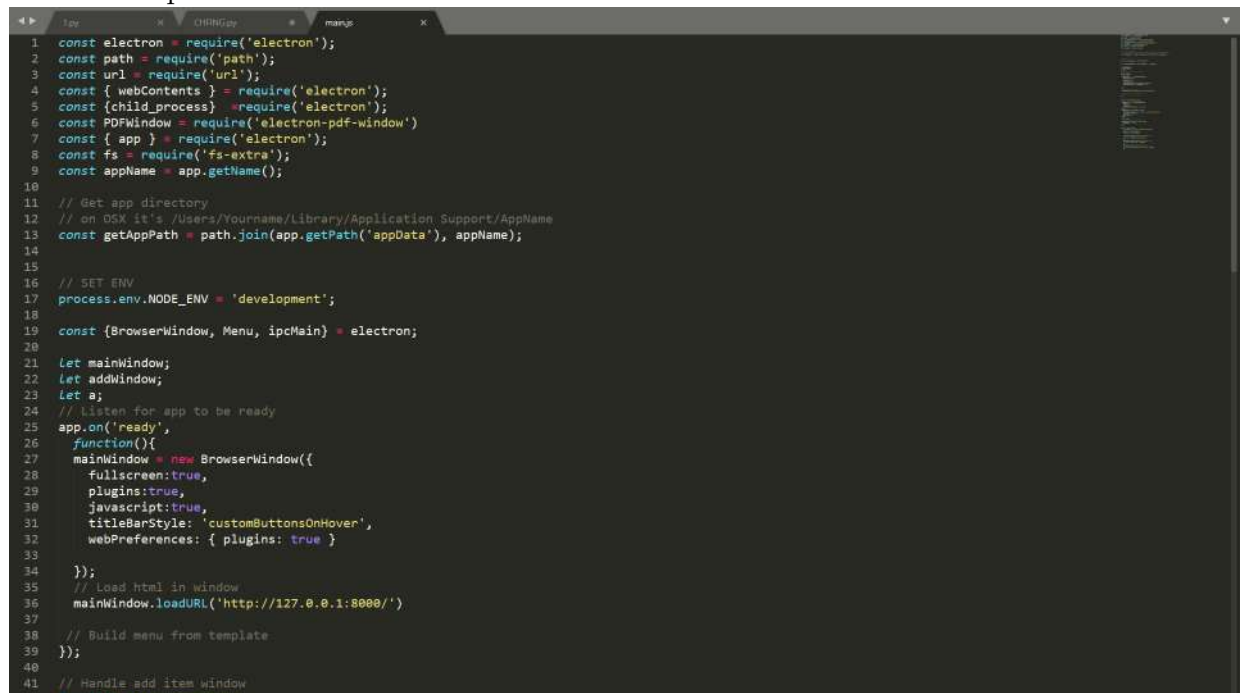


from stellarium to Rpi for conversion.

- Button three will start the stellarium software and will automatically start sending the coordinates of the selected heavenly body.

One of the biggest feature of the application is that it is completely independent from any other application and can be easily installed and run in any computer. We have designed the application so that even if some script libraries are missing, the software will automatically install them.

Here is a small snippet of the desktop application code, which has been made on electron-Js platform:

A screenshot of a code editor window with a dark theme. The editor shows a JavaScript file with Electron API usage. The code includes imports for 'electron', 'path', 'url', 'electron-pdf-window', and 'fs-extra'. It defines a main window with 'fullscreen: true', 'plugins: true', and 'javascript: true'. The window loads a URL 'http://127.0.0.1:8000/'. The code is numbered from 1 to 42.

```
1 const electron = require('electron');
2 const path = require('path');
3 const url = require('url');
4 const { webContents } = require('electron');
5 const { child_process } = require('electron');
6 const PDFWindow = require('electron-pdf-window');
7 const { app } = require('electron');
8 const fs = require('fs-extra');
9 const appName = app.getName();
10
11 // Get app directory
12 // on OSX it's /Users/Yourname/Library/Application Support/AppName
13 const getAppPath = path.join(app.getPath('userData'), appName);
14
15
16 // SET ENV
17 process.env.NODE_ENV = 'development';
18
19 const { BrowserWindow, Menu, ipcMain } = electron;
20
21 let mainWindow;
22 let addWindow;
23 let a;
24 // Listen for app to be ready
25 app.on('ready',
26   function(){
27     mainWindow = new BrowserWindow({
28       fullscreen: true,
29       plugins: true,
30       javascript: true,
31       titleBarStyle: 'customButtonsOnHover',
32       webPreferences: { plugins: true }
33     });
34   });
35 // Load html in window
36 mainWindow.loadURL('http://127.0.0.1:8000/')
37
38 // Build menu from template
39 });
40
41 // Handle add item window
42 function createAddWindow() {
```

Fig. 2.8



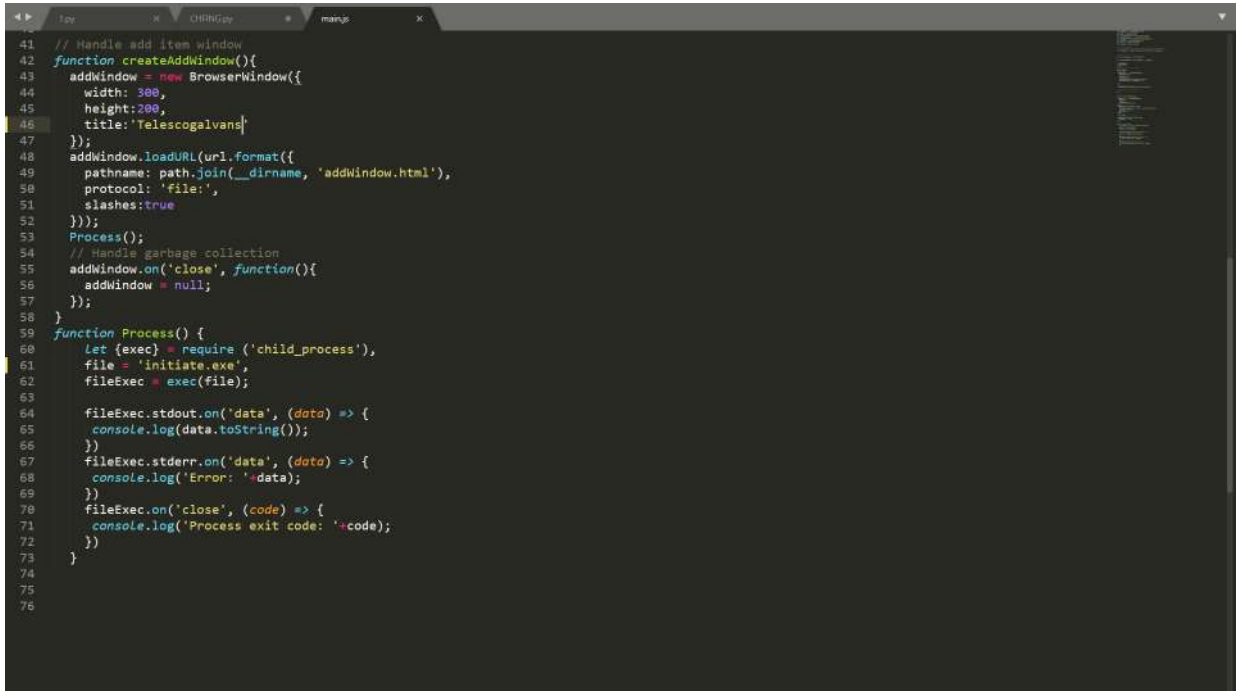


Fig. 2.9

## 2.2 Extracting Data From Stellarium

### Stellarium:

Stellarium is a free open source planetarium for your computer. It shows a realistic sky in 3D, just like what you see with the naked eye, binoculars or a telescope. It is an n-body simulation of our observable universe.

One can install Stellarium by going to the official Stellarium website:-  
<https://stellarium.org/en/>

### Why Stellarium:

Stellarium shows the coordinates of any celestial body up to 6 decimal places. The coordinates that we extract from the Stellarium are the Right Ascension angle and the declination angle.



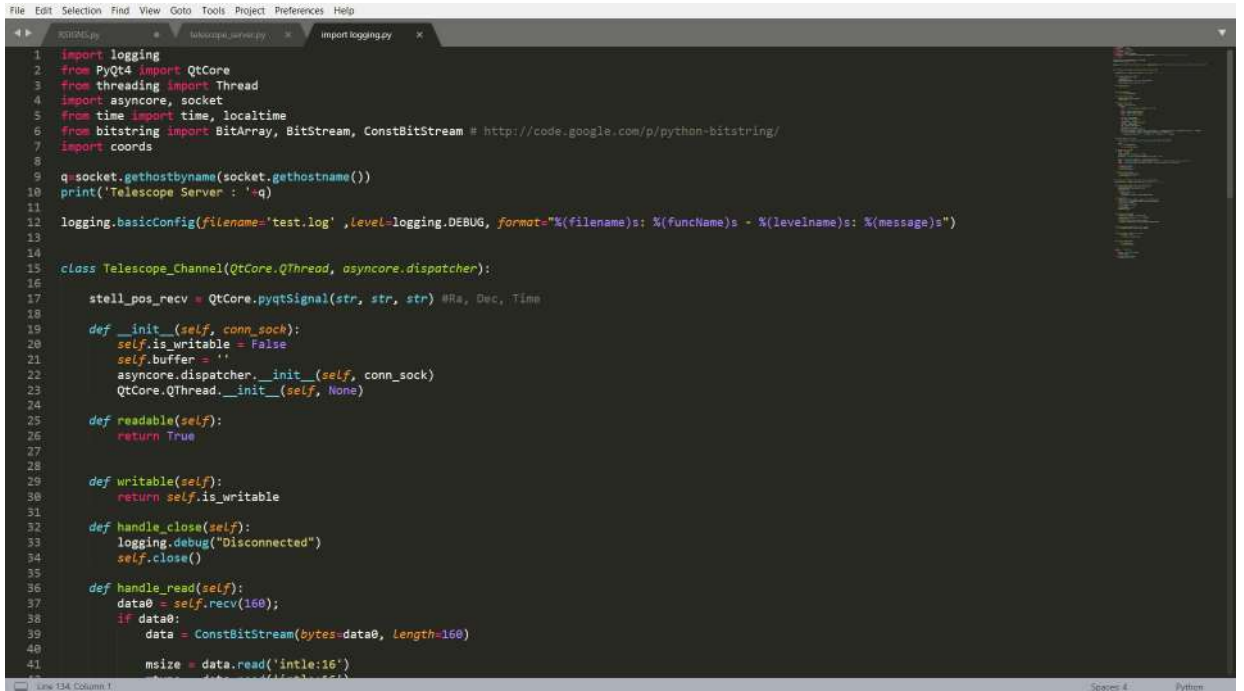
Fig. 2.10

### Extracting data

- We created a virtual telescope server on our PC.
- Stellarium has its in-built plug-in for a telescope control.
- Stellarium can send coordinates to some of the selected telescopes.
- The telescopes that are in the list of Stellarium are Celestron Nextar series.
- But in order to get the coordinates from Stellarium without buying such costly telescope, our virtual server can be connected to Stellarium and can be used to get the coordinates.
- In order to initiate the Stellarium telescope virtual server, click on the first button to initiate it from our desktop app.
- To connect Stellarium to telescope server, open the plug-ins of Stellarium and configure telescope control.
- In order to do the above step for the first time, one needs to follow the steps shown in the procedure section. The code for the virtual telescope server is given below:

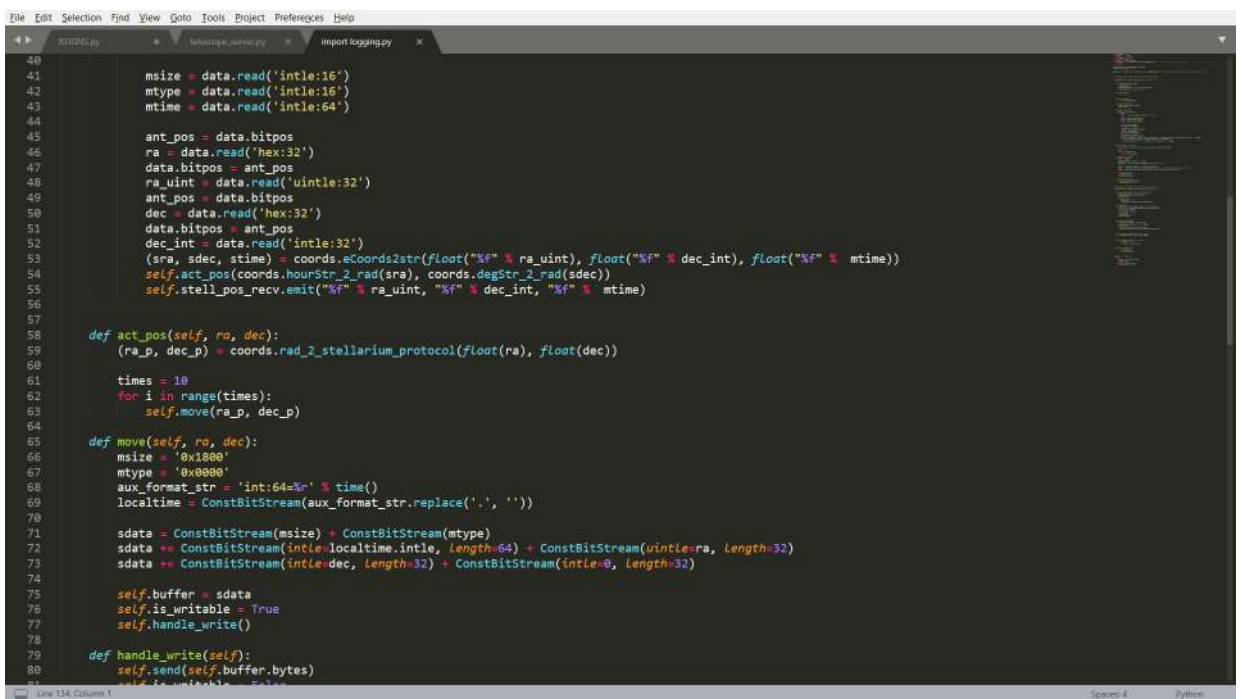
## :TelescoGalvans:

### Telescope Automation



```
1 import logging
2 from PyQt4 import QtCore
3 from threading import Thread
4 import asyncore, socket
5 from time import time, localtime
6 from bitstring import BitArray, BitStream, ConstBitStream # http://code.google.com/p/python-bitstring/
7 import coords
8
9 q_socket.gethostname(socket.gethostname())
10 print('Telescope Server : ' + q)
11
12 logging.basicConfig(filename='test.log', level=logging.DEBUG, format='%(filename)s: %(funcName)s - %(levelname)s: %(message)s')
13
14
15 class Telescope_Channel(QtCore.QThread, asyncore.dispatcher):
16
17     stell_pos_rcv = QtCore.pyqtSignal(str, str, str) #Ra, Dec, Time
18
19     def __init__(self, conn_sock):
20         self.is_writable = False
21         self.buffer = ''
22         asyncore.dispatcher.__init__(self, conn_sock)
23         QtCore.QThread.__init__(self, None)
24
25     def readable(self):
26         return True
27
28     def writable(self):
29         return self.is_writable
30
31     def handle_close(self):
32         logging.debug("Disconnected")
33         self.close()
34
35     def handle_read(self):
36         data0 = self.recv(160);
37         if data0:
38             data = ConstBitStream(bytes=data0, length=160)
39
40             msize = data.read('intle:16')
```

Fig. 2.11



```
40         msize = data.read('intle:16')
41         mtype = data.read('intle:16')
42         mtime = data.read('intle:64')
43
44         ant_pos = data.bitpos
45         ra = data.read('hex:32')
46         data.bitpos = ant_pos
47         ra_uint = data.read('uintle:32')
48         ant_pos = data.bitpos
49         dec = data.read('hex:32')
50         data.bitpos = ant_pos
51         dec_int = data.read('intle:32')
52         (sra, sdec, stime) = coords.eCoords2str(float("%f" % ra_uint), float("%f" % dec_int), float("%f" % mtime))
53         self.act_pos(coords.hourStr_2_rad(sra), coords.degStr_2_rad(sdec))
54         self.stell_pos_rcv.emit("%f" % ra_uint, "%f" % dec_int, "%f" % mtime)
55
56     def act_pos(self, ra, dec):
57         (ra_p, dec_p) = coords.rad_2_stellarium_protocol(float(ra), float(dec))
58
59         times = 10
60         for i in range(times):
61             self.move(ra_p, dec_p)
62
63     def move(self, ra, dec):
64         msize = '0x1800'
65         mtype = '0x0000'
66         aux_format_str = 'int:64=%r' % time()
67         localtime = ConstBitStream(aux_format_str.replace('.', ''))
68
69         sdata = ConstBitStream(msize) + ConstBitStream(mtype)
70         sdata += ConstBitStream(intle=localtime.intle, length=64) + ConstBitStream(uintle=ra, length=32)
71         sdata += ConstBitStream(intle=dec, length=32) + ConstBitStream(intle=0, length=32)
72
73         self.buffer = sdata
74         self.is_writable = True
75         self.handle_write()
76
77     def handle_write(self):
78         self.send(self.buffer.bytes)
79         self.is_writable = False
```

Fig. 2.12

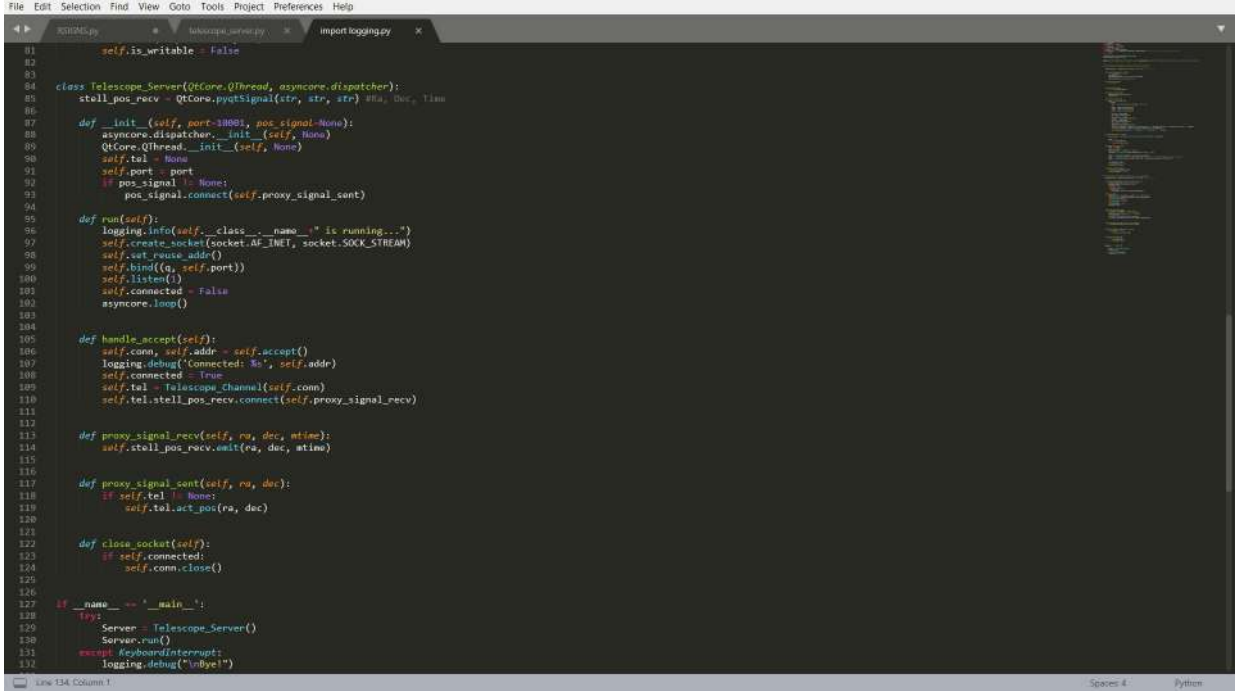


Fig. 2.13

- The code given above connects with Stellarium. Connection to Stellarium is done by using low-level networking, i.e. socket.
- The Telescope-Channel class does the receiving of the coordinates in hexadecimal. It does the decoding of the coordinates received in hexadecimal. The coordinates received are in j2000 system, i.e. Right Ascension and declination angles.
- The class Telescope-Server does the connection of the Server to Stellarium by using socket. It binds to the IP of the PC on which code is running.

The coordinates received are written in the file named test.log, by the logging module.

## 2.3 Conversion of the extracted Data

The socket provides the 'Right Ascension' and 'Declination' angle of the selected heavenly body.

- Right Ascension: *"the distance of a point east of the First Point of Aries, measured along the celestial equator and expressed in hours, minutes, and seconds."*<sup>[1]</sup>
- Declination: *"the angular distance of a point north or south of the celestial equator."*<sup>[1]</sup>

These angles are defined globally. We require an azimuth angle and elevation angle of the celestial body at the particular time for the given place.

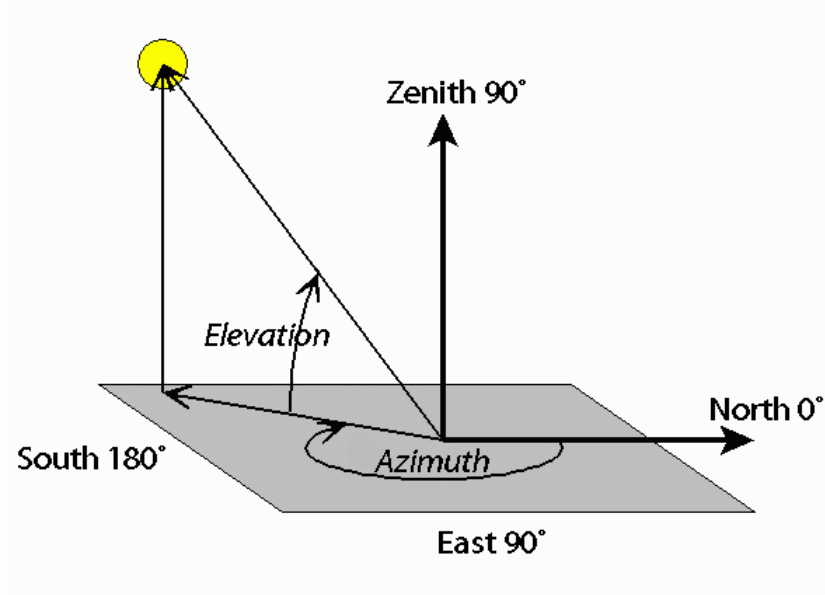


Fig. 2.14

Courtesy: WHAT ARE THE "AZIMUTH AND ELEVATION" OF A SATELLITE?

The conversion requires current Universal time Coordinated (UTC), number of days after January 01 2000 and longitude and latitude value of the place.

```
def RA_Dec_converter(ra,dec):
    long=geocoder.ip("me").lng
    LAT=geocoder.ip("me").lat
    DegRA=ra*15
    U=UtcNow()
    d1=get_days_between(date(2000,1,1),date(U.year,U.month,U.day))
    d2=timedelta(days=d1, hours=U.hour ,minutes=U.minute ,seconds=U.second)
    s=24*60*60
    d=d2.total_seconds()/s
    UT=timedelta(hours=U.hour ,minutes=U.minute ,seconds=U.second)
    UT=UT.total_seconds()/3600
    LST = 100.46 + 0.985647 * (d) + long + 15*UT
    HA=LST-DegRA
    Degree1 = math.asin(math.sin(radianconvert(dec))*math.sin(radianconvert(LAT))+math.cos(radianconvert(dec))
        *math.cos(radianconvert(LAT))*math.cos(radianconvert(HA)))
    A = math.acos((math.sin(radianconvert(dec)) - math.sin(Degree1)*math.sin(radianconvert(LAT)))/
        |math.cos(Degree1)*math.cos(radianconvert(LAT))|)
    Degree1=math.degrees(Degree1)
    A=math.degrees(A)
    if (math.sin(math.radians(HA))<0):
        Degree=A
    else:
        Degree=360-A
    l=[Degree,Degree1]
    return l
```

Fig. 2.15

We calculate Local Sidereal Time. Local Sidereal Time is the actual time at the place, calculated referred to noon when the shadow of the stick is the shortest.

The measurement of Hour angle is necessary to calculate Azimuth angle and Elevation angle.

In the code Degree and Degree1 provides azimuth and elevation angle, respectively.

## 2.4 Setting up Raspberry Pi

We downloaded 'NOOBS' from the Raspberry Pi website on Raspberry Pi 3 B+ model. After setting up the operating system, we downloaded the geocoder library by running the following command in 'terminal'.

```
sudo pip3 install geocoder
```

## 2.5 Raspberry Pi application

To create a better platform for the users, we created an interactive 'turtle window' through python code in Raspberry Pi. After running the code and once the connection is made, the window will open automatically and allow the user to interact using the four keys as shown in the instructions of the page.

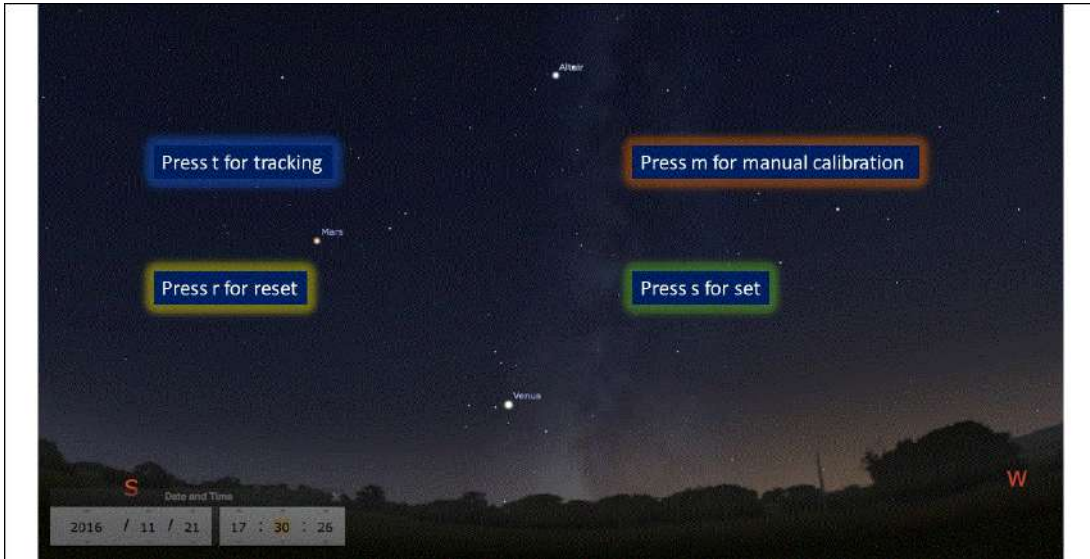


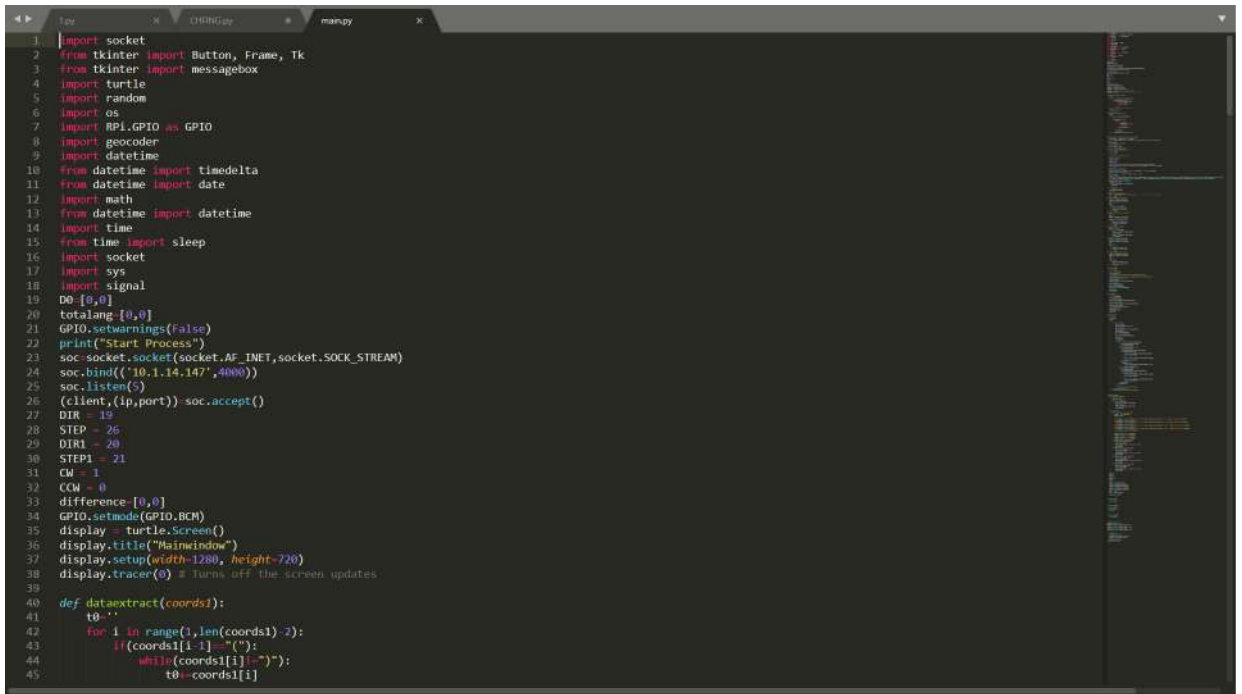
Fig. 2.16

- Pressing 's': After setting the telescope towards a visible celestial body, pressing 's' will ask for the name of the heavenly object. After selecting the same object in 'Stellarium' press 'Ctrl+1'. The code will take those coordinates as reference for future actions.
- Pressing 't': The telescope will start tracking the celestial body. The code is written such that after the change of 0.3 Degrees in any of the elevation and azimuth direction the motor for the motion in that direction will rotate. Pressing 'Ctrl+c' in the console exits this mode. Note: Pressing 'Ctrl+1' is required from sender side after pressing 'Ctrl+c'.
- Pressing 'm': After pressing 'm', a new 'Tkinter' window opens. Pressing 'w', 's', 'a' and 'd' will rotate telescope in up, down, left and right direction respectively by 50 steps. This option helps in centering the heavenly body in the eyepiece.



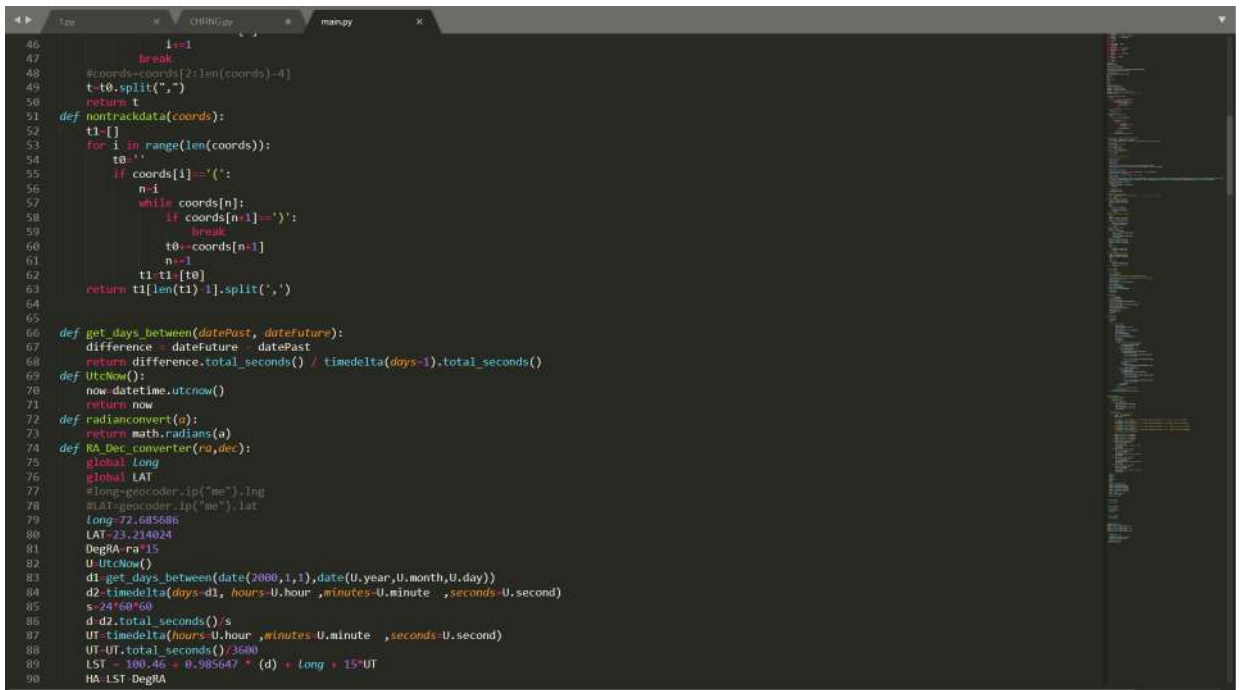
- Pressing 'r': After pressing 'r', the telescope moves to its initial position where it was set.

The code for this is attached below.



```
1 import socket
2 from tkinter import Button, Frame, Tk
3 from tkinter import messagebox
4 import turtle
5 import random
6 import os
7 import RPi.GPIO as GPIO
8 import geocoder
9 import datetime
10 from datetime import timedelta
11 from datetime import date
12 import math
13 from datetime import datetime
14 import time
15 from time import sleep
16 import socket
17 import sys
18 import signal
19 DO=[0,0]
20 totalang=[0,0]
21 GPIO.setwarnings(False)
22 print("Start Process")
23 soc=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
24 soc.bind(('10.1.14.147',4000))
25 soc.listen(5)
26 (client,(ip,port))=soc.accept()
27 DIR = 19
28 STEP = 26
29 DIR1 = 20
30 STEP1 = 21
31 CW = 1
32 CCW = 0
33 difference=[0,0]
34 GPIO.setmode(GPIO.BCM)
35 display = turtle.Screen()
36 display.title("Mainwindow")
37 display.setup(width=1280, height=720)
38 display.tracer(0) # turns off the screen updates
39
40 def dataextract(coords1):
41     t0=''
42     for i in range(1,len(coords1)-2):
43         if(coords1[i-1]!='('):
44             while(coords1[i]!=')'):
45                 t0+=coords1[i]
```

Fig. 2.17



```
46         i+=1
47         break
48     #coords=coords[2:len(coords)-4]
49     t=t0.split(',')
50     return t
51 def nontrackdata(coords):
52     t1=[]
53     for i in range(len(coords)):
54         t0=''
55         if coords[i]!='(':
56             n=i
57             while coords[n]:
58                 if coords[n+1]!=')':
59                     break
60                 t0+=coords[n+1]
61                 n+=1
62             t1=t1+[t0]
63     return t1[len(t1)-1].split(',')
64
65 def get_days_between(datePast, dateFuture):
66     difference = dateFuture - datePast
67     return difference.total_seconds() / timedelta(days=1).total_seconds()
68 def UtcNow():
69     now=datetime.utcnow()
70     return now
71 def radianconvert(a):
72     return math.radians(a)
73 def RA_Dec_converter(ra,dec):
74     global Long
75     global LAT
76     #long=geocoder.ip("me").lng
77     #LAT=geocoder.ip("me").lat
78     Long=22.685686
79     LAT=23.214024
80     DegRA=ra*15
81     U=UtcNow()
82     d1=get_days_between(date(2000,1,1),date(U.year,U.month,U.day))
83     d2=timedelta(days=d1, hours=U.hour, minutes=U.minute, seconds=U.second)
84     s=24*60*60
85     d=d2.total_seconds()/s
86     UT=timedelta(hours=U.hour, minutes=U.minute, seconds=U.second)
87     UT=UT.total_seconds()/3600
88     LST=100.45+0.985647*(d)+(d)*Long+15*UT
89     HA=LST-DegRA
```

Fig. 2.18



```

91 Degree1 = math.sin(math.sin(radianconvert(dec)) * math.sin(radianconvert(LAT)) * math.cos(radianconvert(dec)) * math.cos(radianconvert(LAT)) * math.cos(radianconvert(HA)))
92 A = math.acos((math.sin(radianconvert(dec)) * math.sin(Degree1) * math.sin(radianconvert(LAT))) / (math.cos(Degree1) * math.cos(radianconvert(LAT))))
93 Degree1 = math.degrees(Degree1)
94 A = math.degrees(A)
95 if (math.sin(math.radians(HA)) < 0):
96     Degree = A
97 else:
98     Degree = 360 - A
99 l = [Degree, Degree1]
100 return l
101 def SPRcalculate(degree):
102     SPR = int((degree/0.45) * 14 * (5.16)) # Steps per revolution (360/5.9=200)
103     return SPR
104 def Motorspin(STEP, DIR, d):
105     GPIO.setup(DIR, GPIO.OUT)
106     GPIO.setup(STEP, GPIO.OUT)
107     CW = 1
108     CCM = 0
109     if (d > 0 and d <= 360):
110         GPIO.output(DIR, CW)
111         SPR = SPRcalculate(d)
112     return SPR
113 def ReverseMotor(STEP, DIR, d):
114     CW = 1
115     CCM = 0
116     GPIO.setup(DIR, GPIO.OUT)
117     GPIO.setup(STEP, GPIO.OUT)
118     if (d > 0 and d <= 360):
119         GPIO.output(DIR, CCM)
120         SPR = SPRcalculate(d)
121     return SPR
122 def runmotor(STEP, STEP):
123     delay = 0.001 #
124     for x in range(SPR):
125         GPIO.output(STEP, GPIO.HIGH)
126         sleep(delay)
127         GPIO.output(STEP, GPIO.LOW)
128         sleep(delay)
129 def negativeazi(DIR, STEP, d):
130     GPIO.setup(DIR, GPIO.OUT)
131     GPIO.setup(STEP, GPIO.OUT)
132     CW = 1
133     CCM = 0
134     if (d < 0):
135         GPIO.output(DIR, CW)
136         SPR = SPRcalculate(-d)
137     return SPR
138 def negativeele(DIR, STEP, d):
139     GPIO.setup(DIR, GPIO.OUT)
140     GPIO.setup(STEP, GPIO.OUT)
141     CW = 1
142     CCM = 0
143     if (d < 0):
144         GPIO.output(DIR, CCM)
145         SPR = SPRcalculate(-d)

```

Fig. 2.19

```

146 return SPR
147
148 def initiate():
149     global D0
150     global totalang
151     print('start')
152     a = input("Enter the name of the celestial body:")
153     print("Press Ctrl + 1 after selecting", a, "in Stellarium.")
154     coords = client.recv(2048)
155     print(coords)
156     angles = nontrackdata(str(coords))
157     print(angles)
158     RA = float(angles[0])
159     Dec = float(angles[1])
160     D0 = RA_Dec_converter(RA, Dec)
161     print(D0)
162     totalang = D0
163
164 def reset():
165     global totalang
166     print(totalang)
167     print("Resetting")
168     s1 = Motorspin(STEP, DIR, totalang[0])
169     runmotor(s1, STEP)
170     s2 = ReverseMotor(STEP1, DIR1, totalang[1])
171     runmotor(s2, STEP1)
172     print("Resetting complete")
173     difference = [0, 0]
174     totalang = [0, 0]
175     D0 = [0, 0]
176
177 def tracing():
178     c = True
179     print(D0)
180     x0 = D0
181     try:
182         while(c == True):
183             print('start')
184             coords = client.recv(2048)
185             print(coords)
186             angles = dataextract(str(coords))
187             print(angles)
188             RA = float(angles[0])

```

Fig. 2.20

```

183 Dec=float(angles[i])
184 D=RA_Dec_converter(RA,Dec)
185 print(D)
186 print(D[1])
187 if(D[1]!=60):
188     for i in range(len(difference)):
189         difference[i]=D[i]-x0[i]
190         print(difference)
191         if(abs(difference[0])>0.3):
192             print(D[0],D[1])
193             print(difference[0],difference[1])
194             if(difference[0]>0):
195                 s1=negativeazi(DIR,STEP,difference[0])
196                 runmotor(s1,STEP)
197             else:
198                 s1=ReverseMotor(STEP, DIR,difference[0])
199                 runmotor(s1,STEP)
200             x0[0]=D[0]
201             if(x0[0]==360):
202                 x=Motorspin(STEP,DIR,(totalang[0]-360))
203                 runmotor(x,STEP)
204             if(abs(difference[1])>0.3):
205                 print(D[0],D[1])
206                 print(difference[0],difference[1])
207                 if(difference[1]>0):
208                     s2=negativeazi(DIR1,STEP1,difference[1])
209                     runmotor(s2,STEP1)
210                 else:
211                     s2=Motorspin(STEP1, DIR1,difference[1])
212                     runmotor(s2,STEP1)
213             x0[1]=D[1]
214             totalang=x0
215         else:
216             print("Sky is the limit")
217 except KeyboardInterrupt:
218     print("Successfully exited Tracing")
219
220 def newwindow():
221     def runmotor1(STEP):
222         delay=0.001/8
223         print("rotating")
224         for i in range(50):

```

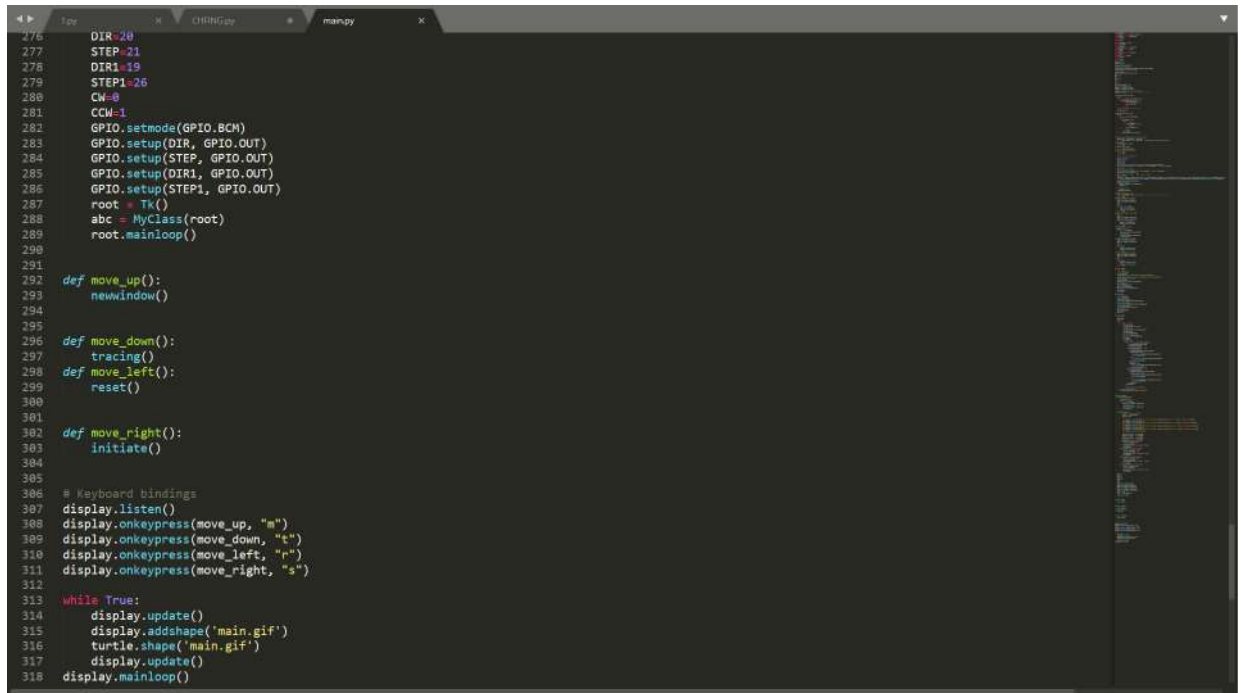
Fig. 2.21

```

232 GPIO.output(STEP, GPIO.HIGH)
233 sleep(delay)
234 GPIO.output(STEP, GPIO.LOW)
235 sleep(delay)
236
237 class MyClass:
238     def __init__(self, master):
239         frame = Frame(master)
240         frame.pack()
241
242         self.button = Button(frame, text="w : To move Telescope Upwards ", fg = 'red', command=self.func1)
243         self.button.pack(side='left')
244         self.button = Button(frame, text="a : To move Telescope Leftwards ", fg='brown', command=self.func2)
245         self.button.pack(side='left')
246         self.button = Button(frame, text="d : To move Telescope Rightwards ", fg = 'blue', command=self.func3)
247         self.button.pack(side='left')
248         self.button = Button(frame, text="s : To move Telescope Downwards ", fg = 'black', command=self.func4)
249         self.button.pack(side='bottom')
250
251         master.bind('w', self.func1)
252         master.bind('a', self.func4)
253         master.bind('d', self.func2)
254         master.bind('s', self.func3)
255     def func1(self, _event=None):
256         GPIO.output(DIR, CCW)
257         runmotor1(STEP)
258         totalang[1]+=((50*0.45)/(14*5.16))
259         D0=totalang
260     def func2(self, _event=None):
261         GPIO.output(DIR1, CCW)
262         runmotor1(STEP1)
263         totalang[0]+=((50*0.45)/(14*5.16))
264         D0=totalang
265     def func3(self, _event=None):
266         GPIO.output(DIR1, CW)
267         runmotor1(STEP1)
268         totalang[0]+=((50*0.45)/(14*5.16))
269         D0=totalang
270     def func4(self, _event=None):
271         GPIO.output(DIR, CW)
272         runmotor1(STEP)
273         totalang[1]+=((50*0.45)/(14*5.16))
274         D0=totalang

```

Fig. 2.22



The image shows a screenshot of a code editor with a dark theme. The editor has two tabs open: 'CHIRPy' and 'marspy'. The 'marspy' tab is active, displaying Python code. The code is a script for controlling a telescope using a Raspberry Pi. It includes GPIO setup for DIR, STEP, DIR1, and STEP1, and a Tkinter interface for controlling the telescope's movement. The code is as follows:

```
276 DIR=20
277 STEP=21
278 DIR1=19
279 STEP1=26
280 CW=0
281 CCW=1
282 GPIO.setmode(GPIO.BCM)
283 GPIO.setup(DIR, GPIO.OUT)
284 GPIO.setup(STEP, GPIO.OUT)
285 GPIO.setup(DIR1, GPIO.OUT)
286 GPIO.setup(STEP1, GPIO.OUT)
287 root = Tk()
288 abc = MyClass(root)
289 root.mainloop()
290
291
292 def move_up():
293     newWindow()
294
295
296 def move_down():
297     tracing()
298 def move_left():
299     reset()
300
301
302 def move_right():
303     initiate()
304
305
306 # Keyboard bindings
307 display.listen()
308 display.onkeypress(move_up, "u")
309 display.onkeypress(move_down, "d")
310 display.onkeypress(move_left, "l")
311 display.onkeypress(move_right, "r")
312
313 while True:
314     display.update()
315     display.addshape('main.gif')
316     turtle.shape('main.gif')
317     display.update()
318 display.mainloop()
```

Fig. 2.23

### 3 Electrical Assembly

Here are some pictures of the complete electrical system assembly:

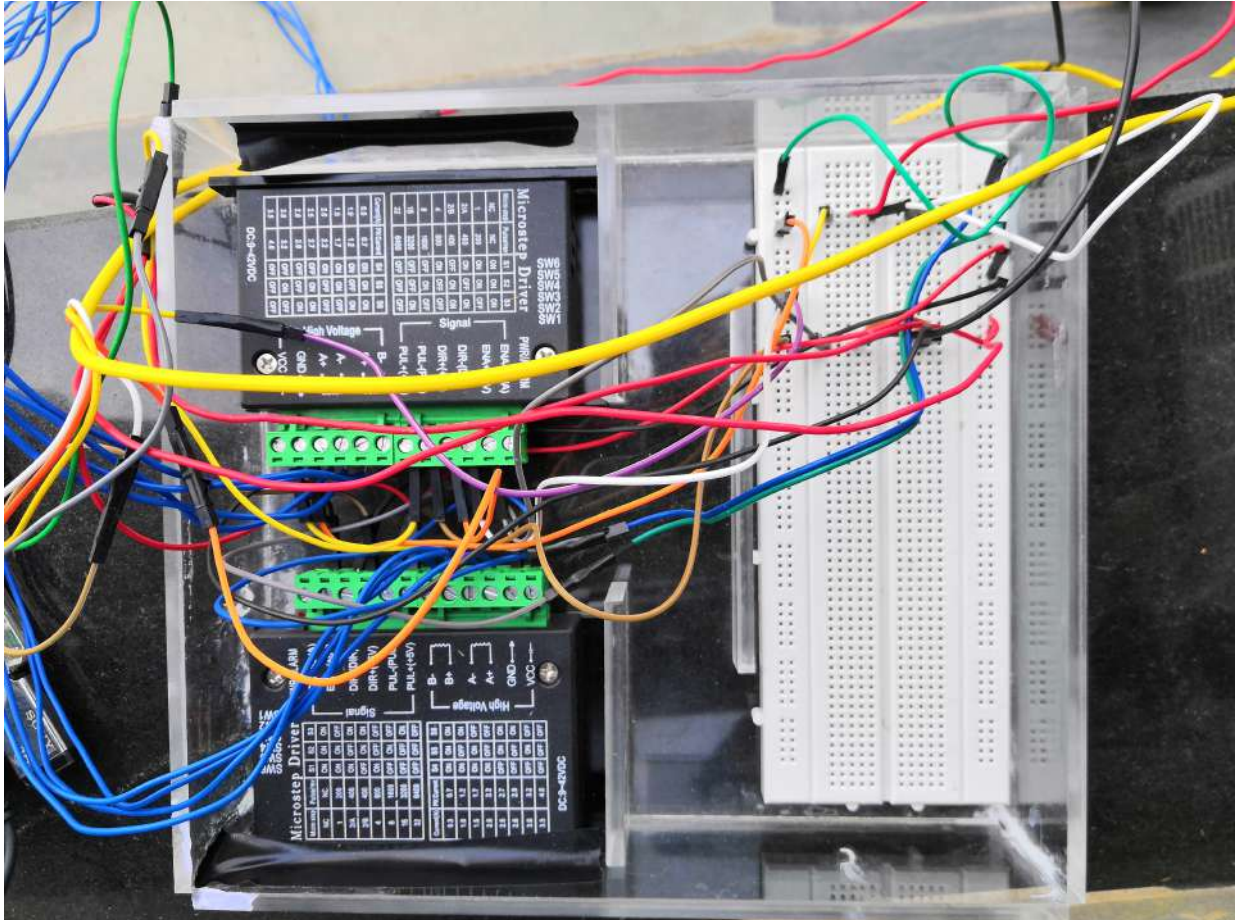


Fig. 3.1



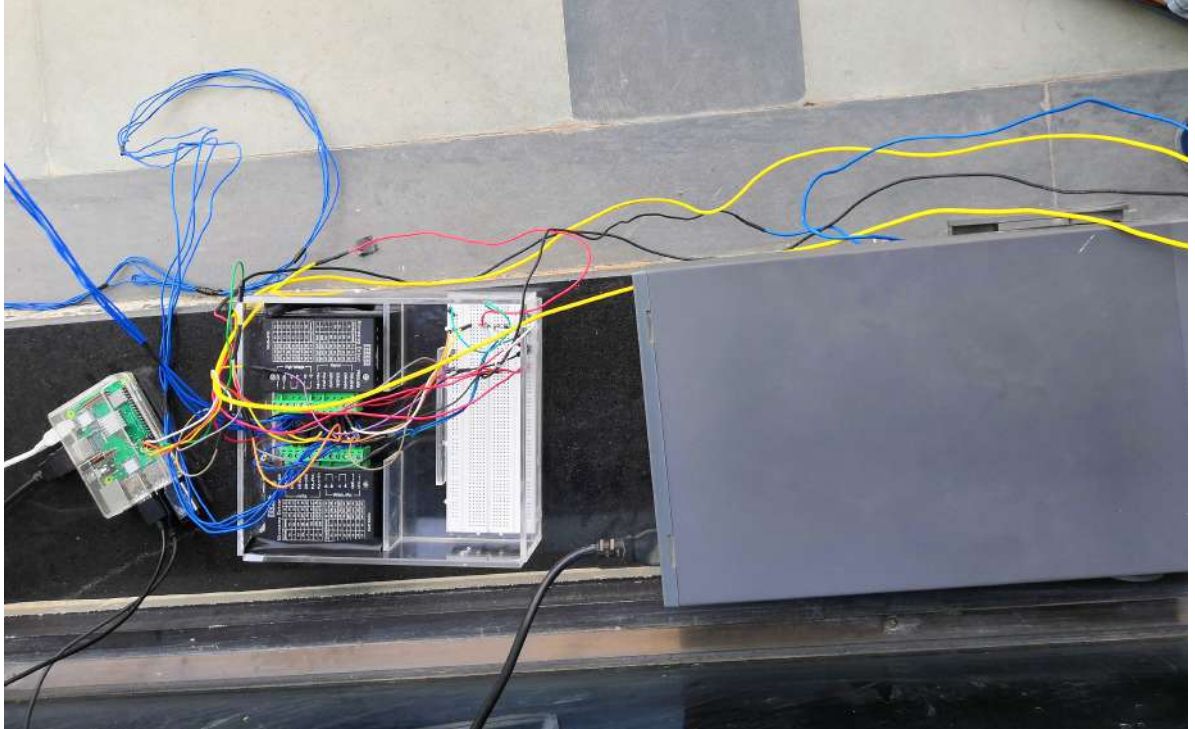


Fig. 3.2

The electrical assembly chiefly involves the following:

### 3.1 Motors and Drivers

The stepper motor requires the driver for micro-stepping. We used TB6600 stepper motor driver along with NEMA-17 45 kg cm planetary geared motor. The connections were made, as shown in the figure.(Photo)

Enable (+)	5 V
Enable (-)	Ground
Direction (+)	Raspberry Pi connection (GPIO)
Direction (-)	Ground
Pulse (+)	Raspberry Pi connection (GPIO)
Pulse (-)	Ground
$V_{cc}$	30 V
Ground	Ground

### 3.2 Connecting Motors to Raspberry Pi

We used General Purpose Input Output(GPIO) pins of Raspberry Pi to give the signals to rotate the motors. We used pins 20 and 21 for one motor and, 19 and 26 for another one. We require to convert degrees to the number of steps the motor to be rotated. The simple code to run the motors is attached here.

```
import RPi.GPIO as GPIO
from time import sleep
DIR = 20
STEP = 21
DIR1 = 19
STEP1 = 26
CW = 1
CCW = 0
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
def SPRcalculate(degree):
    SPR = int((degree/0.45)*14*5.16) # Calibration is done for NEMA-17 45 kg cm Planetary geared Motor
    # Steps per revolution for 1/8 microstepping
def Motorspin(STEP, DIR,d):
    # 14 is multiplied as gear ratio in NEMA-17 is 1:14
    GPIO.setup(DIR, GPIO.OUT)
    GPIO.setup(STEP, GPIO.OUT)
    CW = 1
    CCW = 0
    if (d>=0 and d<=360):
        GPIO.output(DIR, CW)
        SPR=SPRcalculate(d)
    return SPR
def ReverseMotor(STEP, DIR,d):
    CW = 1
    CCW = 0
    GPIO.setup(DIR, GPIO.OUT)
    GPIO.setup(STEP, GPIO.OUT)
    if (d>=0 and d<=360):
        GPIO.output(DIR, CCW)
        SPR=SPRcalculate(d)
    return SPR
def runmotor(SPR,STEP):
    delay=0.001/8
    for x in range(SPR):
        GPIO.output(STEP, GPIO.HIGH)
        sleep(delay)
        GPIO.output(STEP, GPIO.LOW)
        sleep(delay)
D=[0,0]
azi=float(input("Enter azimuth angle in Degrees:"))
ele=float(input("Enter elevation angle in Degrees:"))
D[0]=ReverseMotor(STEP, DIR, azi)
D[1]=Motorspin(STEP1, DIR1 ele)
runmotor(D[0],STEP)
runmotor(D[1],STEP1)
```

Fig. 3.3

The motors will run as per requirement after creating proper setup and running the code, as shown in section 2.5, Fig. 2.16 to Fig. 2.22.

## 4 Mechanical Assembly

Under mechanical assembly, the extra assets that needs to be attached with our system for its successful functioning were framed out. With the help of laser cutting on acrylic sheets and 3-D printing using PLA (polylactic acid) material, we were able to achieve very high precision of mechanical rotations and compactness in our system.

The following are the several items we printed/cut :

## 4.1 Gears and Base

There are four gears involved in the functioning. Two gears are attached with the motors. The other two gears are involved in rotating the telescope across the 'azimuth' and 'elevation' planes.

The images of different gears(Fig. 4.1) and the base(Fig. 4.2) are attached below.  
(Fig. 4.3 - gears and bases separated, Fig. 4.4 - gears and bases on Telescope)

The Motors are held on the base with the help of special screw system on the base.

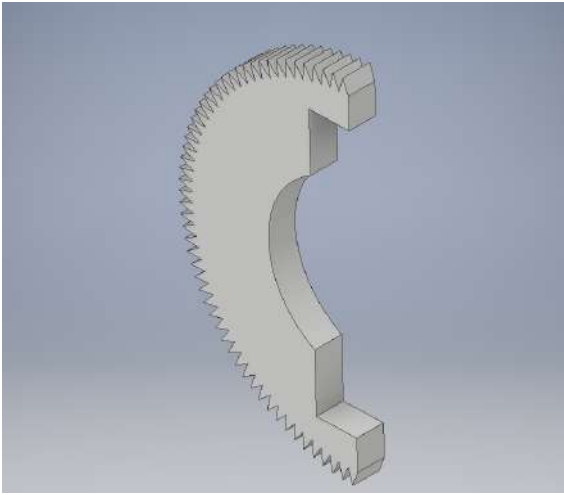


Fig. 4.1

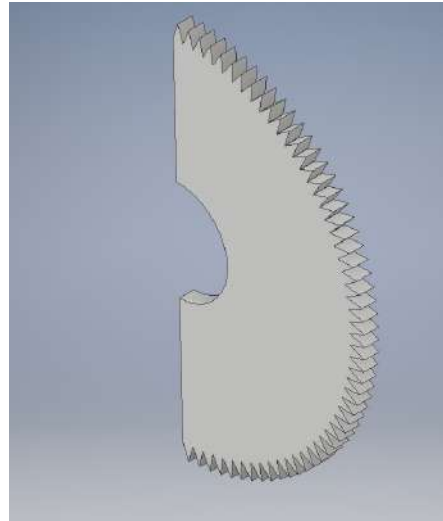


Fig. 4.2

Also the gears are fixed to the base which helps rotating the whole telescope on rotating the base attached with gears. The following picture will make it more clear.

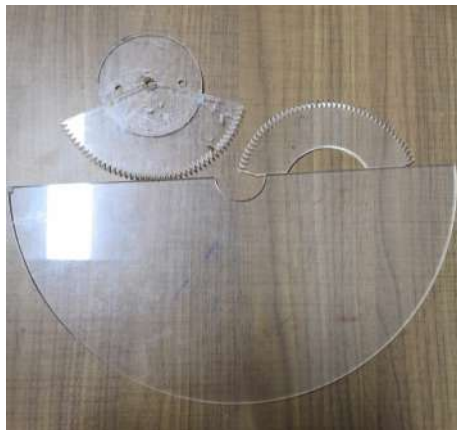


Fig. 4.3



Fig. 4.4





Fig. 4.5

## 4.2 Motor Holders

There are two motor holders for keeping the motors intact with the telescopic system. One motor holder is placed on an efficient set-up build up on the telescope stand as it can be seen in Fig. 3.6. The other motor holder is attached on the 'base platform' attached near the rim of the telescope holder.

Fig. 3.6 shows the snapshot of the ipt file of motor holder:

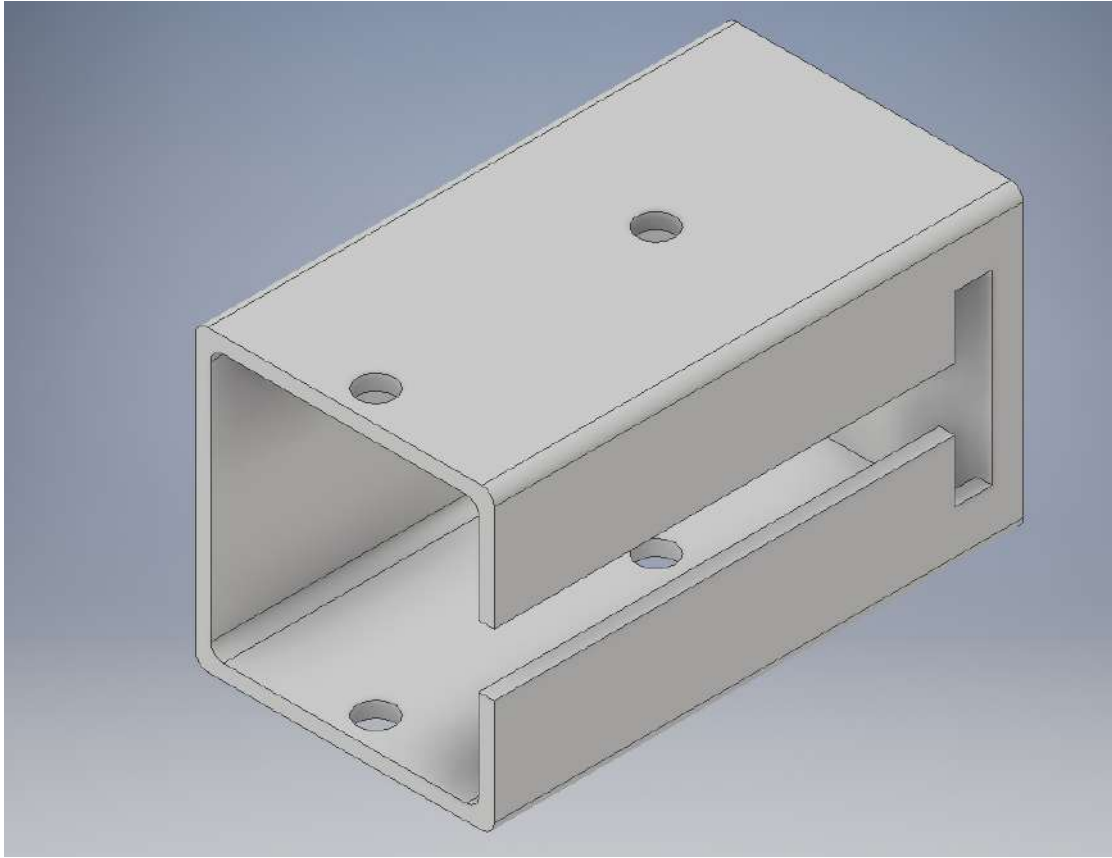


Fig. 4.6

### 4.3 Rotation Mechanism

There are two types of rotation that were achieved.

- Horizontal : Performing horizontal rotation was much easier as there were very less counter-torque in the direction of rotation. It is achieved by tightening a gears with the help of connectors and nut and screw system. The gear is rotated via motor using rear-reduction mechanism thus rotating the telescope. The connectors can be seen in the pictures attached below:

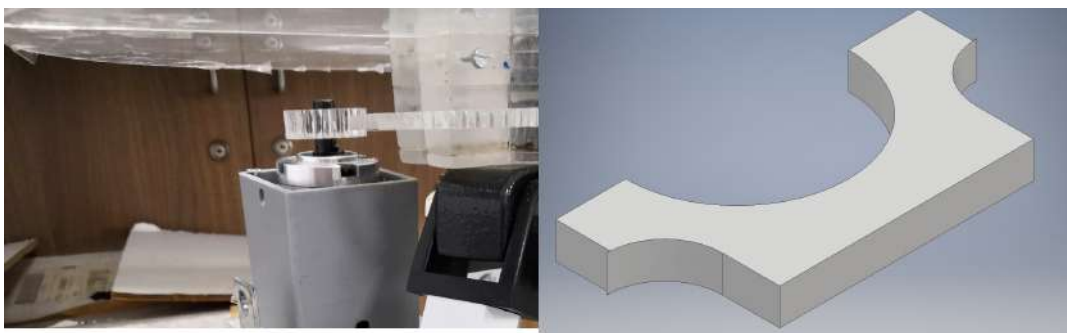


Fig. 4.7

- Vertical : Performing vertical rotation was a hard nut to crack as the rotation was to be performed against the gravity. For that purpose special connectors and torque-reducing mechanisms were designed, using long sheet(as can be seen in fig x,y). Below are some of the photos of the same:

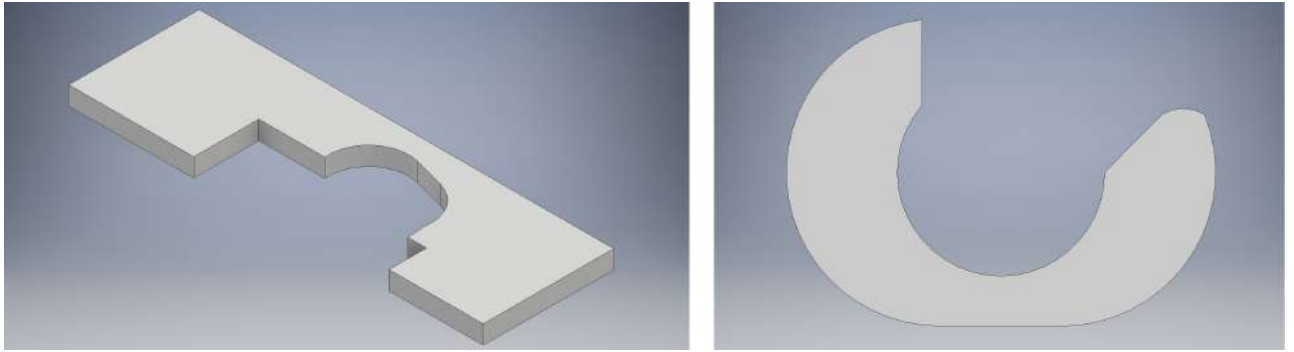


Fig. 4.8

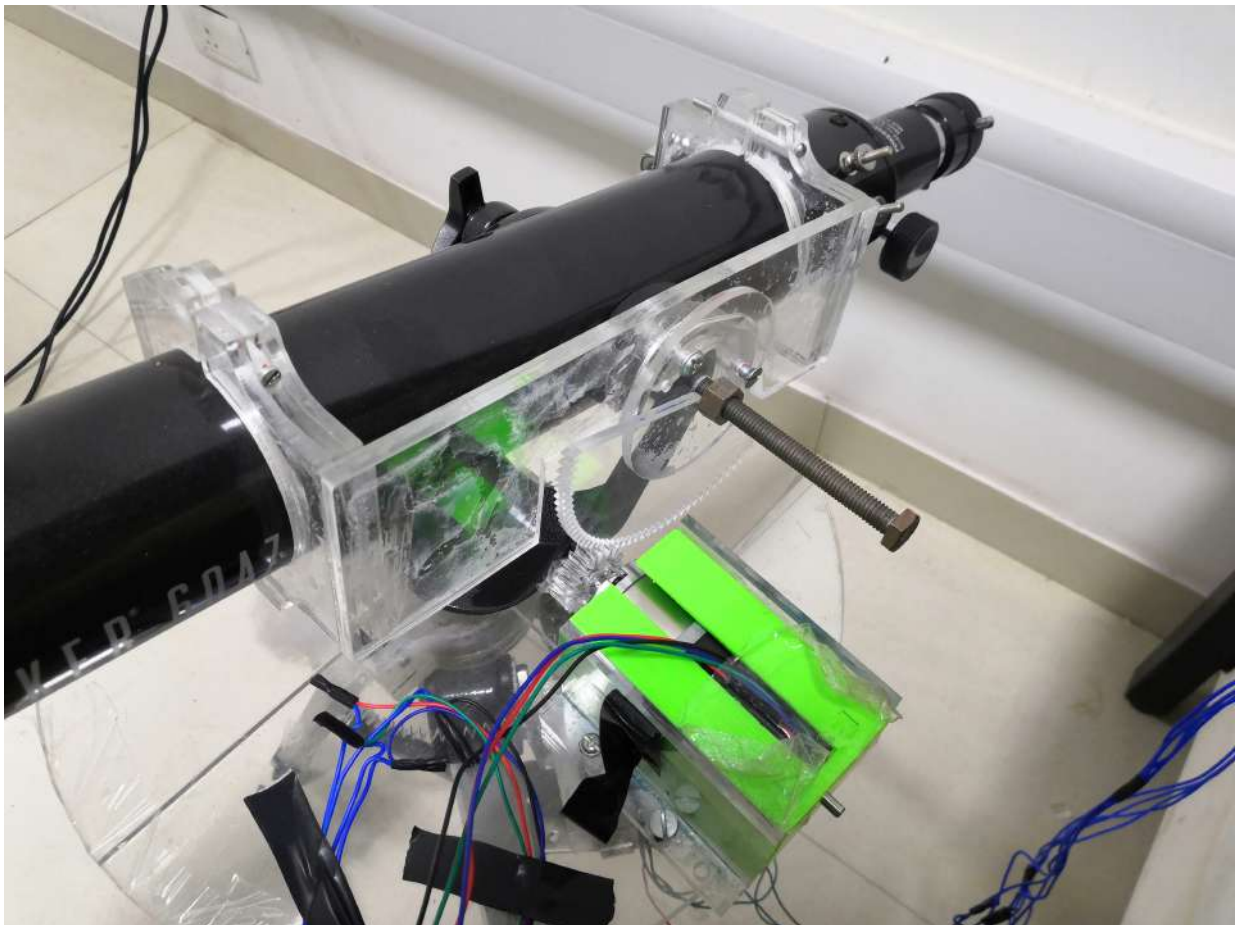


Fig. 4.9

## 5 The Complete Assembly

After extensively working out day and night, we finally completed the whole assembly. (Detailed description has been given above). We manufactured and assembled items, be it as small as a gear or as big as the base platform.

Attached below is the image depicting the complete setup of our 'Automated System'.



Fig. 5.1

## 6 Procedure to Use The System

Here is the complete set of procedures that are to be followed in order to use the system:

- Make all the necessary electrical connections, as written above, and ensure the motors are on and the gears are perfectly placed.
- Press button 1 in TelescoGalvans Application to start the Virtual Telescope server.



- Start main.py file in Raspberry pi.
- Enter the IP of Raspberry pi in button 2 form and click on "INITIATE". As soon as both of them connects, a new GUI window will open in R-pi.
- Select the appropriate option and then click on the button 3 in Desktop Application to start Stellarium.
- Now if its first time you are using this software, then connect the stellarium to the virtual telescope server. Below are the steps to do the same.
- That's it, now enjoy star gazing !!

Follow the following instructions to connect stellarium to virtual telescope server.

- Click on the configuration window in Stellarium.



Fig. 5.2

- Click on the configuration window in Stellarium.
- Now click on the plugins section in the configuration window:

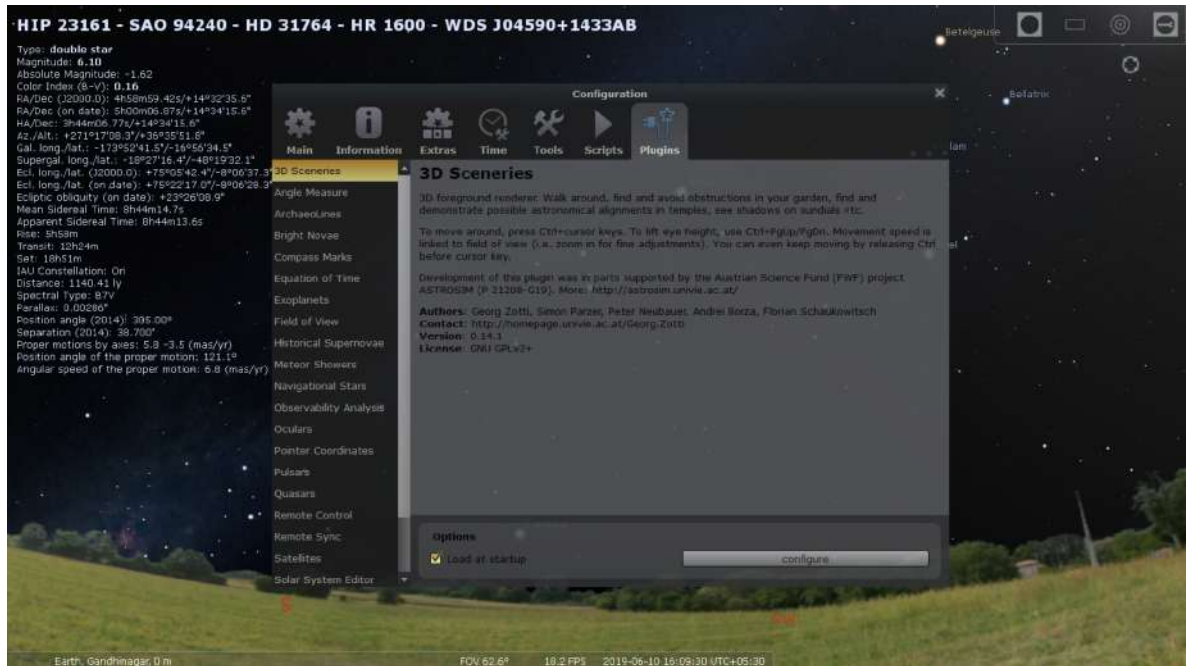


Fig. 5.3

- Click on the telescope control from the list of plugins in the left side of the configuration windows.

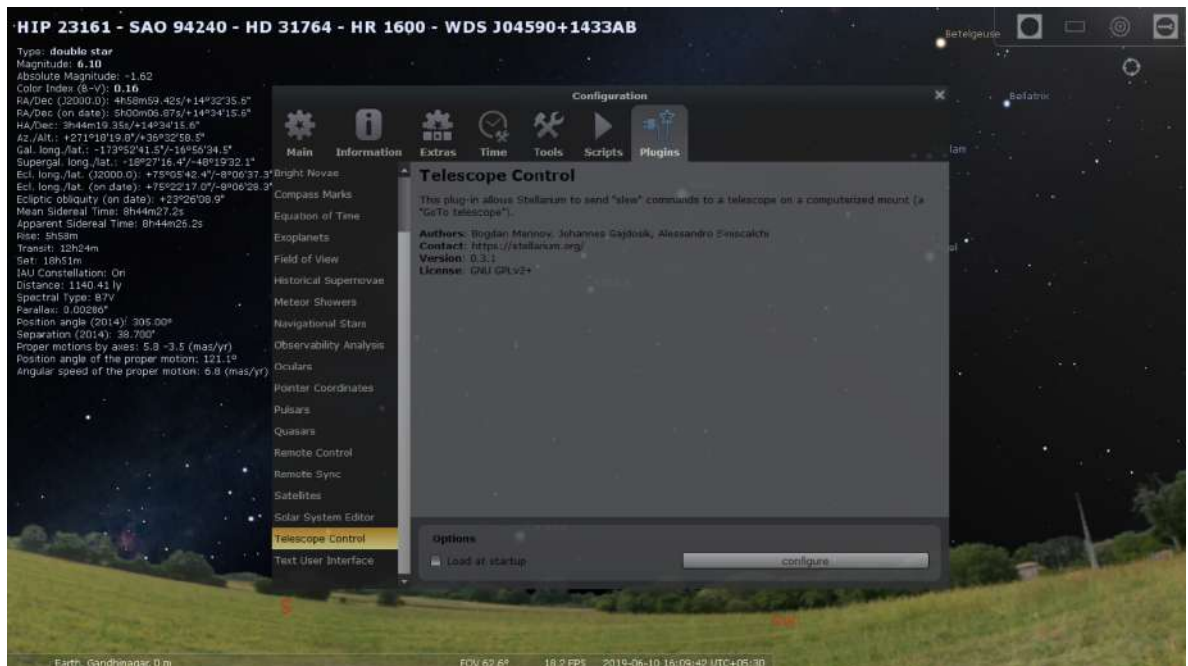


Fig. 5.4

- Click on the load at startup in options:

# :TelescoGalvans:

## Telescope Automation

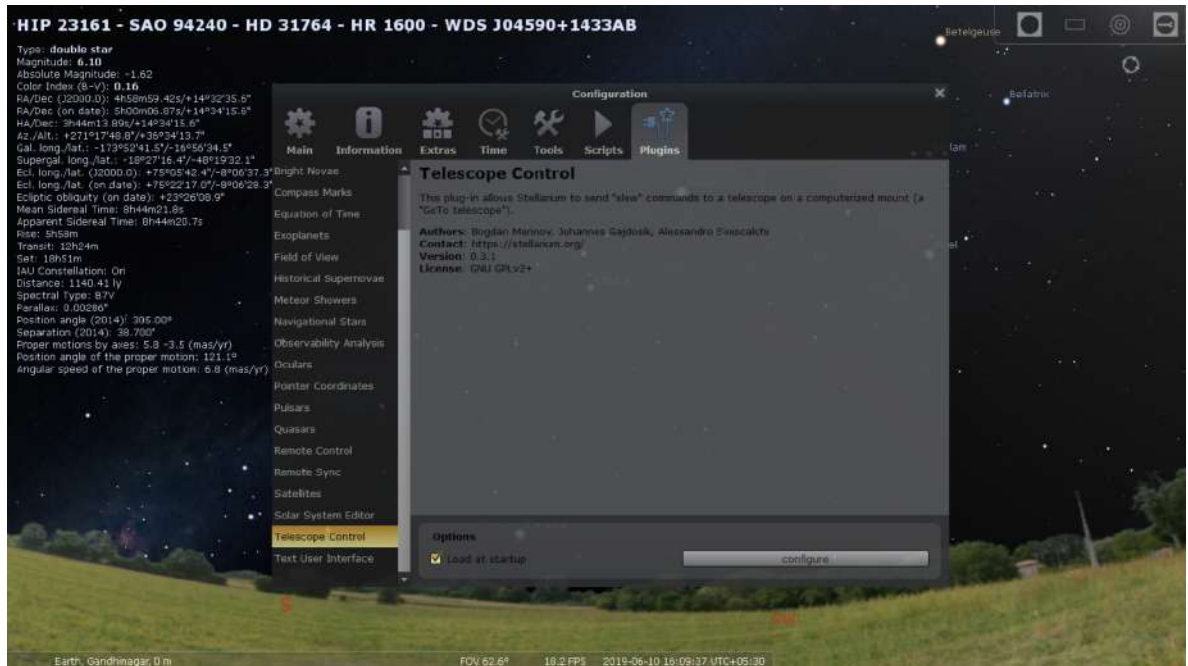


Fig. 5.5

- If you are not able to click on the configure button, then restart stellarium and again open the plugins in the configuration window and click on the configure button.

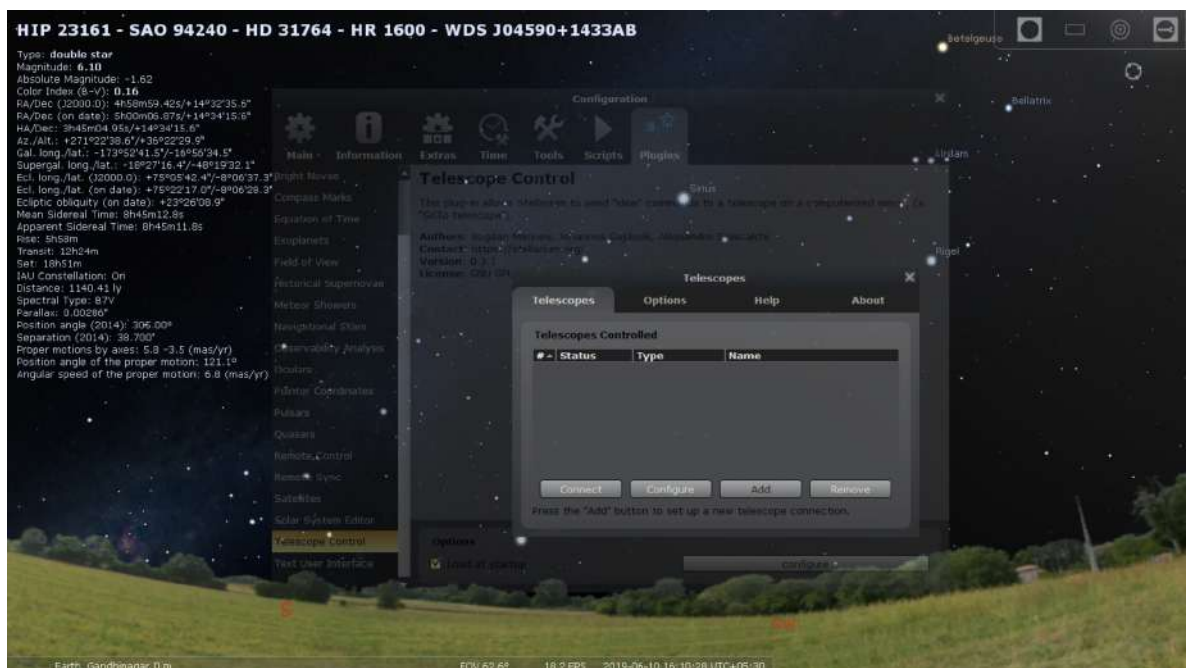


Fig. 5.6

- Now click on Add button.



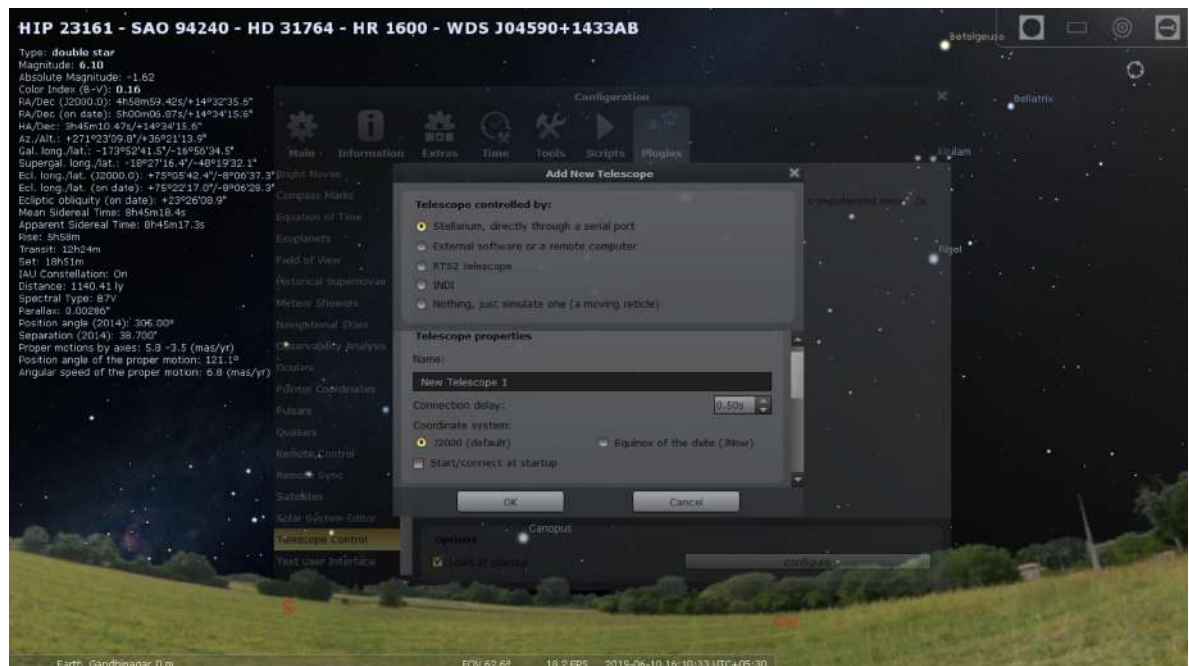


Fig. 5.7

- In telescope properties, click on Start/connect at startup.

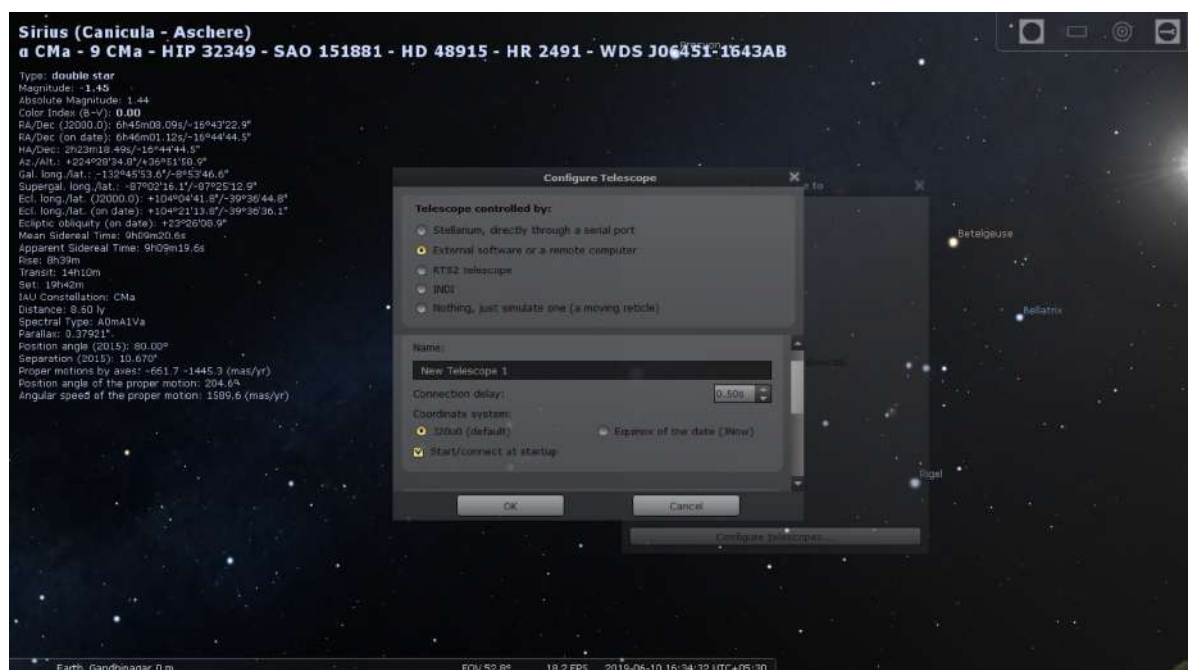


Fig. 5.8

- In telescope properties, scroll down and in the connections settings, type your PC's IP address.

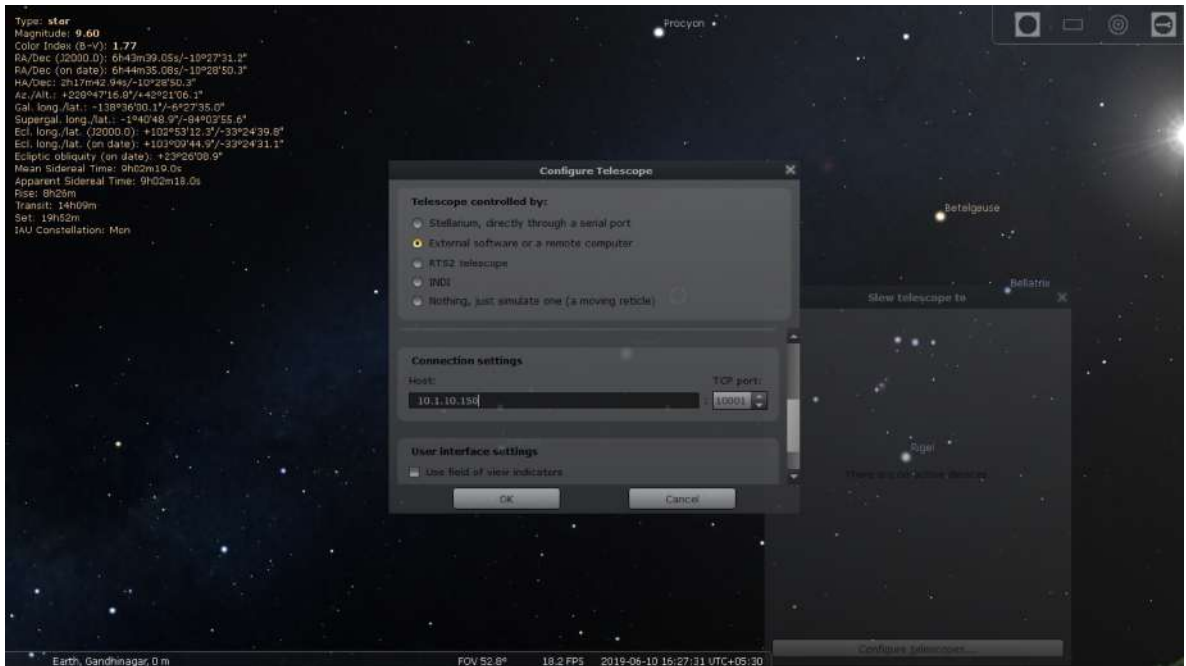


Fig. 5.9

- Click on OK.
- In our desktop application click on the first button to initiate the virtual telescope server.
- Click on connect



Fig. 5.10

- You'll see that the virtual telescope is connected.



Fig. 5.11

- Close all the windows in Stellarium and click on any star and press ctrl+1:



Fig. 5.12

## 7 Challenges Faced

A great and an enormous task is never an easy picking. Our team faced a great set of challenges from the nano level to the micro level. The challenges have

been discussed below in brief:

## 7.1 Precision-Conversion

The coordinates obtained from stellarium have 6 significant digits after decimal point. However, the conversion formula gives the coordinates having only 2 significant digits after decimal point.

## 7.2 Precision-Mechanical

The method of laser cutting on acrylic sheets gives a high precision but that precision is not up to the mark needed by us. This precision problem was encountered in the movement of different gears.

## 7.3 Strength of gears

The gears have been manufactured through laser cutting on acrylic sheet. However, they are brittle in nature.

# 8 Budget and timeline

For more details of the items, click on it in the below table.

Item	Price	Qty.	Amount in Rs.
Stepper Motor	3782	2	7564
Raspberry Pi	2990	2	5980
Telescope	5980	1	5980
HDMI to HDMI cable	375	1	375
Raspberry Pi case and heat sink	265	1	265
Stepper Motor Driver	1180	2	2360
Memory Card (32 GB)	363	1	363
Bread Board and Jumper Wires set	249	1	249
Miscellaneous Charges	629	-	629
		<b>Total</b>	23,765

In the given sheet is the time-line of our project.

# 9 Pros and Cons of the model

Every object in this world has its own pros and cons. The pros and cons of our telescopic system have been discussed below:

## 9.1 Pros of the model

- There are several celestial bodies which are not visible with the naked eyes. However, with this system it can be viewed clearly with great precision.
- Usually, the process of setting up the telescope takes a lot of time. However, with our system this problem has been resolved. It just takes a few seconds in setting up the telescope.

## 9.2 Cons of the model

- The telescopic system has a constrained motion in the elevation or the vertical plane. It can't go beyond  $70^\circ$  in this plane.

## 10 Conclusion

In the end, we have developed an "Automated Telescope System" which will automatically orient towards a celestial body when the input is given. Our team members have worked day and night to make the ends meet and successfully complete this tough task.

## 11 Vote of thanks

We would like to applaud for and thank one and all who have helped us during the course of the project.

- Prof. Shanmuganathan Raman
- Prof. Nihar Mohapatra
- Mr. Pankaj Vatwani
- Mr. Nirav Bhatt
- Mr. Palak Bagiya
- Mr. Utkarsh Nanda
- Mr. Jay Shah

## 12 Credits and References

- [1] Google dictionary Definition
- [2] Converting RA and DEC to ALT and AZ: <http://www.stargazing.net/kepler/altaz.html>
- [3] What are "Azimuth and Elevation" of a Satellite?: <https://www.celestis.com/resources/faq/what-are-the-azimuth-and-elevation-of-a-satellite/>
  - Robotic telescope: <https://en.wikipedia.org/wiki/Robotic-telescope>
  - Raspberry Pi website: <https://www.raspberrypi.org/downloads/noobs/>
  - <https://www.awwwards.com/>
  - <https://www.youtube.com/watch?v=kN1Czs0m1SU>
  - <https://electronjs.org/>
  - Raspberry Pi Stepper Motor Tutorial: [https://www.youtube.com/watch?v=LUbhPKBL\\_IU&feature=youtu.be](https://www.youtube.com/watch?v=LUbhPKBL_IU&feature=youtu.be)