YoEstuve

MENU

Communications between Python and Stellarium (Stellarium Telescope Protocol)

By JuanRa | 25/08/2013 |Astronomy, Laser Pointer

In this post, I'm going to explain a way to simulate a telescope connection to Stellarium with Python, so that we can both receive coordinates pointed with it, and send coordinates to display the field of view indicator on the screen.

The coordinates that Stellarium sends are equatorial coordinates.

In the page of the Stellarium project we can find information about how to control a telescope with Stellarium. I opted for the client-server over TCP/IP option, so we're going to implement a "Stellarium Telescope Protocol" server with Python.

We only need to manage an asynchronous connection, so we'll use a simple socket with the asyncore Python library.

This protocol uses only two types of messages, one for server→client direction and other for client→server, with the following structure:

```
Scheme for server→client message

LENGTH (2 bytes, integer): length of the message
TYPE (2 bytes, integer): 0
TIME (8 bytes, integer): current time on the server computer in microseconds
    since 1970.01.01 UT. Currently unused.
RA (4 bytes, unsigned integer): right ascension of the telescope (J2000)
    a value of 0x100000000 = 0x0 means 24h=0h,
    a value of 0x80000000 means 12h
DEC (4 bytes, signed integer): declination of the telescope (J2000)
    a value of -0x40000000 means -90degrees,
    a value of 0x0 means 0degrees,
    a value of 0x40000000 means 90degrees
STATUS (4 bytes, signed integer): status of the telescope, currently unused.
    status=0 means ok, status
```

```
Scheme for client→server message

LENGTH (2 bytes, integer): length of the message
TYPE (2 bytes, integer): 0
TIME (8 bytes, integer): current time on the server computer in microseconds
    since 1970.01.01 UT. Currently unused.
RA (4 bytes, unsigned integer): right ascension of the telescope (J2000)
    a value of 0x100000000 = 0x0 means 24h=0h,
    a value of 0x80000000 means 12h
DEC (4 bytes, signed integer): declination of the telescope (J2000)
    a value of -0x40000000 means -90degrees,
    a value of 0x0 means 0degrees,
    a value of 0x40000000 means 90degrees
```

The values are transmitted in little-endian format, and we'll use **ConstBitStream** (code.google.com) to manage it.

The following code snippet displays the received coordinates in the terminal, and then returns them back to Stellarium to show the field of view indicator on the screen.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import math
import logging
from PyQt4 import QtCore
import asyncore, socket
from time import sleep, time
from string import replace
from bitstring import BitArray, BitStream, ConstBitStream
import coords

logging.basicConfig(level=logging.DEBUG, format="%(filename)s: %(funcName)s - %(levelname)s: %(me

## \brief Implementation of the server side connection for 'Stellarium Telescope Protocol'
#
#  Manages the execution thread to the server side connection with Stellarium
class Telescope_Channel(QtCore.QThread, asyncore.dispatcher):

    ## Class constructor
    #
    # \param conn_sock Connection socket
    def __init__(self, conn_sock):
        self.is_writable = False
        self.buffer = ''
        asyncore.dispatcher.__init__(self, conn_sock)
        QtCore.QThread.__init__(self, None)

    ## Indicates the socket is readable
    #
    # \return Boolean True/False
    def readable(self):
        return True
```

```
## Indicates the socket is writable
#
# \return Boolean True/False
def writable(self):
    return self.is_writable

## Close connection handler
#
def handle_close(self):
    logging.debug("Disconnected")
    self.close()

## Reading socket handler
#
# Reads and processes client data, and throws the proper signal with coordinates as parameter
def handle_read(self):
    #format: 20 bytes in total. Size: intle:16
    #Incomming messages comes with 160 bytes..
    data0 = self.recv(160);
    if data0:
        data = ConstBitStream(bytes=data0, length=160)
        #print "All: %s" % data.bin

        msize = data.read('intle:16')
        mtype = data.read('intle:16')
        mtime = data.read('intle:64')

        # RA:
        ant_pos = data.bitpos
        ra = data.read('hex:32')
        data.bitpos = ant_pos
        ra_uint = data.read('uintle:32')

        # DEC:
        ant_pos = data.bitpos
        dec = data.read('hex:32')
        data.bitpos = ant_pos
        dec_int = data.read('intle:32')

        logging.debug("Size: %d, Type: %d, Time: %d, RA: %d (%s), DEC: %d (%s)" % (msize, mty
        (sra, sdec, stime) = coords.eCoords2str(float("%f" % ra_uint), float("%f" % dec_int),

        #Sends back the coordinates to Stellarium
        self.act_pos(coords.hourStr_2_rad(sra), coords.degStr_2_rad(sdec))

## Updates the field of view indicator in Stellarium
#
# \param ra Right ascension in signed string format
# \param dec Declination in signed string format
def act_pos(self, ra, dec):
    (ra_p, dec_p) = coords.rad_2_stellarium_protocol(ra, dec)

    times = 10 #Number of times that Stellarium expects to receive new coords //Absolutly emp
    for i in range(times):
        self.move(ra_p, dec_p)

## Sends to Stellarium equatorial coordinates
#
#  Receives the coordinates in float format. Obtains the timestamp from local time
#
# \param ra Ascensión recta.
# \param dec Declinación.
def move(self, ra, dec):
    msize = '0x1800'
```

```
        mtype = '0x0000'
        localtime = ConstBitStream(replace('int:64=%r' % time(), '.', ''))
        #print "move: (%d, %d)" % (ra, dec)

        sdata = ConstBitStream(msize) + ConstBitStream(mtype)
        sdata += ConstBitStream(intle=localtime.intle, length=64) + ConstBitStream(uintle=ra, ler
        sdata += ConstBitStream(intle=dec, length=32) + ConstBitStream(intle=0, length=32)
        self.buffer = sdata
        self.is_writable = True
        self.handle_write()

    ## Transmission handler
    #
    def handle_write(self):
        self.send(self.buffer.bytes)
        self.is_writable = False

## \brief Implementation of the server side communications for 'Stellarium Telescope Protocol'.
#
#  Each connection request generate an independent execution thread as instance of Telescope_Char
class Telescope_Server(QtCore.QThread, asyncore.dispatcher):

    ## Class constructor
    #
    # \param port Port to listen on
    def __init__(self, port=10001):
        asyncore.dispatcher.__init__(self, None)
        QtCore.QThread.__init__(self, None)
        self.tel = None
        self.port = port

    ## Starts thread
    #
    # Sets the socket to listen on
    def run(self):
        logging.info(self.__class__.__name__+" running.")
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind(('localhost', self.port))
        self.listen(1)
        self.connected = False
        asyncore.loop()

    ## Handles incomming connection
    #
    # Stats a new thread as Telescope_Channel instance, passing it the opened socket as parameter
    def handle_accept(self):
        self.conn, self.addr = self.accept()
        logging.debug('%s Connected', self.addr)
        self.connected = True
        self.tel = Telescope_Channel(self.conn)

    ## Closes the connection
    #
    def close_socket(self):
        if self.connected:
            self.conn.close()

#Run a Telescope Server
if __name__ == '__main__':
    try:
        Server = Telescope_Server()
        Server.run()
    except KeyboardInterrupt:
        logging.debug("\nBye!")
```

We’ve extended the *QtCore.QThread* class because we’ll use this code (with some modifications) with a Qt user interface as a module. Also you can see that some aux functions are used to some unit conversions, for example *coords.rad_2_stellarium_protocol* and *coords.hourStr_2_rad*. These functions are implemented in another file “coords.py” that is imported as a module with “import coords”:

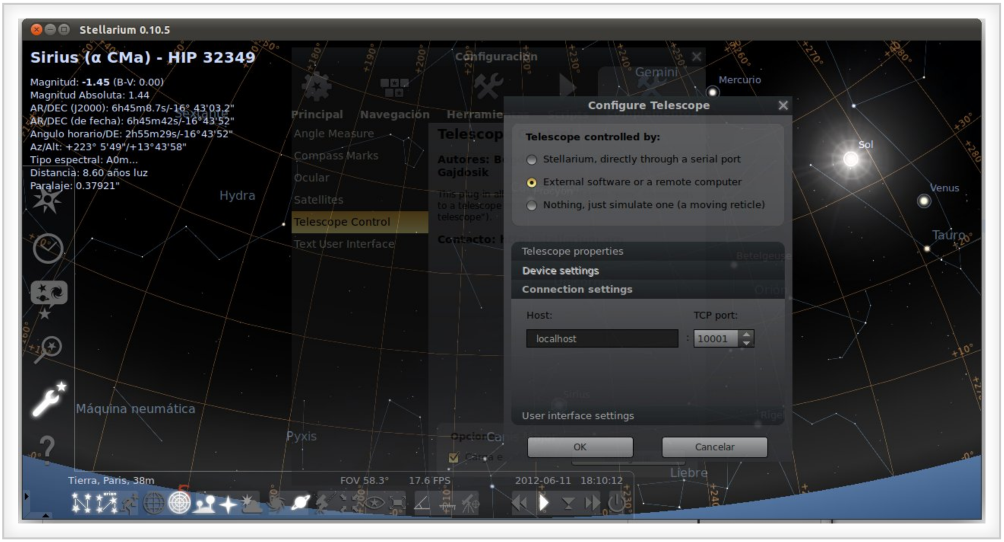
```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import math
import re
import logging
from time import strftime, localtime
from string import replace

# \brief Functions library for format conversions.
#
#  Contains the necessary functions to calculate most commons format conversions used by the
#  with the device and Stellarium.

## From radians to hours, with until six decimals of precision (float)
# (rads * 180)/(15 * pi)
#
# \param rads Radians in float format
# \return Float that represents the number of hours equivalent to the received radians
def rad_2_hour(rads):
    h = round( (rads * 180)/(15 * math.pi), 6)
    if h > 24.0:
        h = h - 24.0
    if h < 0.0:
        h = 24.0 + h
    return h

## Tranforms from degrees in string format to radians
# d = D°M'S'' => D+(M/60)+(S/60^2) degrees => D.d°
#
# \param d Degrees in string format ("D°M'S''" || "D.d°")
# \return Radians in float format
def degStr_2_rad(d):
    exp1 = re.compile('^-?[0-9]{,3}(°|º)[0-9]{,3}\'[0-9]{,3}([\']{2}|")
```

We just have to place both files in the same directory, (telescope_server.py and coords.py), give execution permissions (*chmod +x telescope_server.py*) and execute:

```
chmod +x telescope_server.py
./telescope_server.py
telescope_server.py: run - INFO: Telescope_Server running.
```

Now the Stellarium configuration. First we need to configure the telescope connection:

```
Configuration window > Plug-ins > Telescope Control > Configure Telescope
```



Sirius (α CMa) - HIP 32349
Magnitud: **-1.45** (B-V: 0.00)
Magnitud Absoluta: 1.44
AR/DEC (J2000): 6h45m8.7s/-16° 43'03.2"
AR/DEC (de fecha): 6h45m42s/-16°43'52"
Angulo horario/DE: 2h55m29s/-16°43'52"
Az/Alt: +223° 5'49"/+13°43'58"
Tipo espectral: A0m...
Distancia: 8.60 años luz
Paralaje: 0.37921"

Configure Telescope

Telescope controlled by:
- Stellarium, directly through a serial port
- [✓ External software or a remote computer]
- Nothing, just simulate one (a moving reticle)

Telescope properties
Device settings
Connection settings
Host: localhost
TCP port: 10001
User interface settings

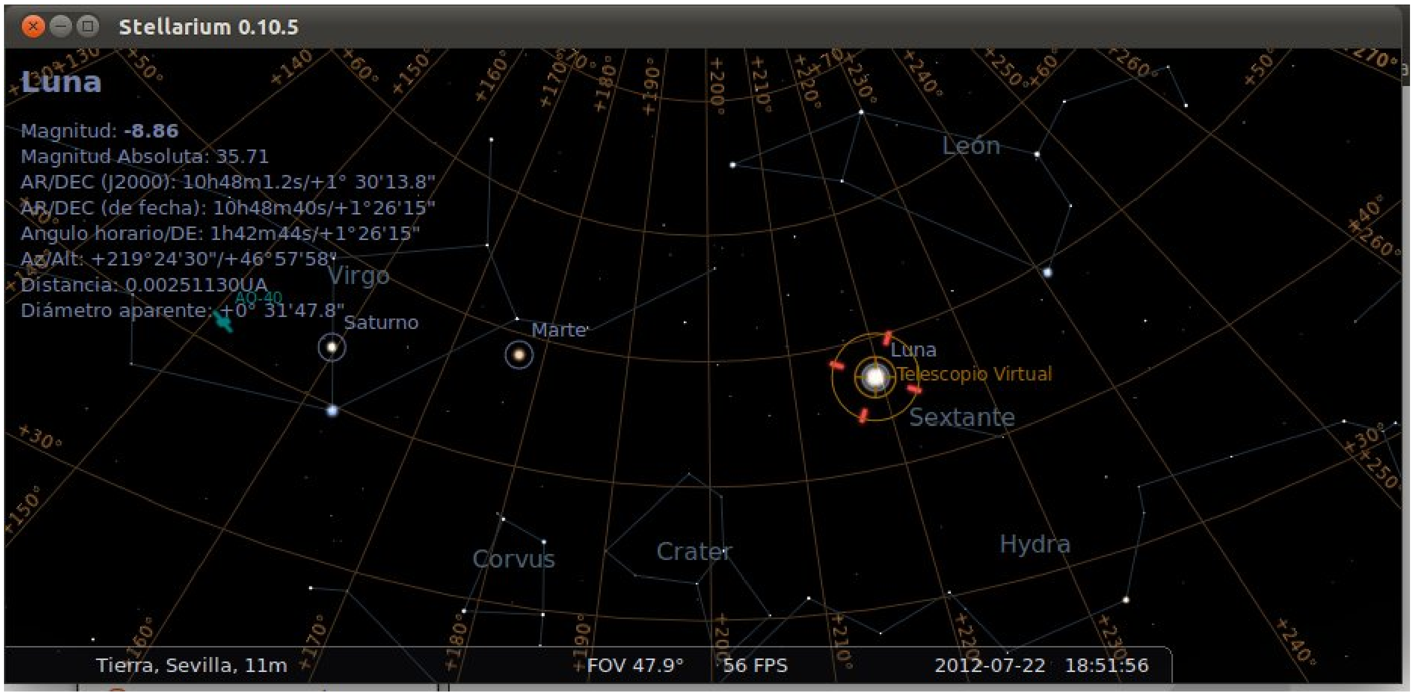
Now we set the connection mode, name, IP and port:

Connection mode: **External Software or a remote computer**
Name: **Virtual Telescope**
IP: **localhost**
Port: **10001**

Once connected, and assuming that we only have one configured device, we press “Ctrl + 1” to send the coordinates from Stellarium (the number 1 indicates the first configured device). Then we should see in console the received coordinates (among other debugging data):

```
./telescope_server.py
telescope_server.py: run - INFO: Telescope_Server running.
telescope_server.py: handle_accept - DEBUG: Connected: ('127.0.0.1', 37023)
telescope_server.py: handle_read - DEBUG: Size: 20, Type: 0, Time: 1342975913635132,
coords.py: rad_2_stellarium_protocol - DEBUG: (hours, degrees): (10.800277, 1.503900)
```

The script returns back the coordinates to Stellarium so that it displays the field of view indicator:



And we already have in our script the equatorial coordinates of any celestial object. Use it wisely 😉

Cheers!

Tagged astronomic laser pointer. Bookmark the permalink.

« Telescope control with Stellarium and Arduino

Communications between Python and Arduino (USB-Serial) »

15 Responses to *Communications between Python and Stellarium (Stellarium Telescope Protocol)*

Bhanesh Awasthi 12/11/2014 AT 2:44 PM

Hello,
I am an engineering student in Germany.Your posted project is really interested and well explained.
I am trying to interface my own created mount with stellarium using serial communication and I succeed to send RA and DEC to the stellarium and my scope is visible on the stellarium. My next task is to slew telescope reticle to the selected target. When I select the target and press Ctrl+1 and I check in the telescope debug file:-
sendCommand(Lx200CommandGetRa)
16386, 13:47:07.789861Z: Lx200Connection::sendCommand(Lx200CommandGetDec)

```
16386, 13:47:09.073935Z: Lx200Connection::sendCommand(Lx200CommandStopSlew)
16386, 13:47:09.074935Z: Lx200Connection::sendCommand(Lx200CommandSetSelectedRa(18:55:25))
16386, 13:47:09.074935Z:
Lx200Connection::sendCommand(Lx200CommandSetSelectedDec(-24:18:25))
16386, 13:47:09.074935Z: Lx200Connection::sendCommand(Lx200CommandGotoSelected)
```

My confusion is that How can I send the RA and DEC of the selected target to my main computer i.e. Raspberry Pi for further coordinate conversion?
My final task is to get the RA and DEC of the selected target in Pi and convert the equatorial coordinates according to the steps required by the stepper motor to move to the selected target.
It would be very helpful for me if you can give some suggestion.
Thanking in advance for your kind cooperation.

Reply

JuanRa 13/11/2014 AT 11:14 AM

Hi Bhanesh,
I'm not sure if I fully understand your issue, but anyway I'll give it a try 🙂
If you can already send coordinates to Stellarium that means you have already the socket connected to it, so that's the half of the way.
The function that receives the coordinates from Stellarium is "handle_read" in the "Telescope_Channel" class (see the code above).
In this function is where you can do the coordinate transformations. In my project I did that in Arduino, since my goal was to make the device as self-governed as possible.
By the way, here you have the full code of the project. Maybe it can be useful:
https://github.com/juanrmn/Arduino-Telescope-Control
In the testing folder, there is a script for debugging Stellarium communications. I think that if you run it and play a bit with the code you'll finally realise how it works.
I hope this helps you…
Regards.

Reply

Pingback: *Help converting python::bitstream usage to PHP equivalent | DL-UAT*

rickbassham 22/04/2015 AT 1:25 AM

Thanks for the code. I modifed it and am using it to automatically solve images taken by my camera, and then update Stellarium with my telescope's current location.

Reply

Fouad 18/01/2016 AT 10:14 PM

how to run python file please ?

Reply

JuanRa 19/01/2016 AT 10:14 AM

I assumed that the file has exec permissions in the post… I'm going to fix that, thanks 🙂
Meanwhile, to run a python file you can do:

```
python python_file.py
```

Or you can add exec permission to the file and then run it:

```
chmod +x python_file.py
./python_file.py
```

Reply

Alex Petrov 14/04/2016 AT 8:29 PM

Hello. I try to reproduce your project and I've some issues with converting coordinates from Horizontal to Equatorial. I am embarrassed by this particular line (in getECoords func.):
`(*ar) = atan2(EVC[1], EVC[0]) + (_k*(t-_t0));`
atan2 returns value in the range [-pi:pi] and you add the time. What the point? Now the value neither in radians nor in degrees. Can you explain more detailed this.
Thanks a lot.

Reply

ErnestoCD 27/04/2016 AT 2:48 PM

How do i syncronise Stellarium with the python script? i hope it can be done becouse i want to make it work without human action. thanks for everything, your code was ery usefull

Reply

Pingback: *Using SenseHat for azimuth and altitude | w01f359*

Yeb Havinga 06/11/2016 AT 3:03 PM

Hi. Thank you very much for your work and explanations. I want to use your telescope server as a submodule of a project of my own and therefore copied the code on a github project. Please let me know if you object to this, and I will remove the subproject.

Reply

JuanRa 08/11/2016 AT 8:05 PM

Hi Yeb, of course you can use the code for whatever you want. It's good to know that this can be useful for someone. Also I`ve seen you mentioned me on Github, that's enough for me 🙂
Cheers

Reply

Hector Santini 01/01/2017 AT 1:19 PM

Saludos JuanRa:
Do you have a complete version where everything runs on Raspberry Pi-3 (No Arduino)? The Pi-3 have 4 USB's and an integrated Wifi.
Gracias,
Hector

Reply

R. 07/09/2017 AT 1:54 AM

Hi,

THank you for this great work, I was wondering if you knew how to draw a trajectory in stellarium. For example, let's say i have the coordinates of my object and also the exposure time. Do you know if I can trace the trajectory of that object during the exposure time?

Thanks!

R.

Reply

JuanRa 12/09/2017 AT 8:43 AM

Hi,
I'm sorry but I don't know how to do that. It's been a long time since I was programming this, but as far as I can remember I don't think this is even possible. The Stellarium protocol was quite simple. Anyway as I said it was a long time ago, so perhaps now the protocol has new functions for that. It's an useful option indeed.

Reply

Benoit 06/08/2019 AT 9:50 AM

Hello, just to correct your introduction, the client->server message also needs a 4 bytes status at the end, bringing the total bytes of the message to 24 as stated in your code (first byte = 0x18 = 24).
If you forget it, and send the first byte as 20 (without the last 4 0's status), stellarium will fail to parse the message and even stopping the connection.
Anyway, this was a great help for me to program a c# server, so cheers and many thanks !
Can you tell us where you got the protocol informations ?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Visual Text

Paragraph

Name *

Email *

Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search OK

Recent Posts

Install and run Zeronet on your Raspberry Pi

Compile PHP 7 for Raspberry Pi (with memcached)

Configure a DNS server with your Raspberry Pi

Convert video from x265 to x264 codecs with Raspberry Pi

Javascript cross-domain localStorage

Payin’ Bills…

Archives

March 2016

February 2016

January 2016

September 2013

August 2013

August 2012

July 2012

Categories

Astronomía

Astronomy

JavaScript

Laser Pointer

Puntero Astronómico Láser

Raspberry Pi

Uncategorized

Links

DELL to HELL

Proyecto completo en GitHub (eng)

Taki's Home Page

Meta

Log in

Entries RSS

Comments RSS

WordPress.org