De: **Michel Crausaz** Michel.Crausaz@unige.ch
Objet: Pour infos : Controlling Telescopes with Raspberry Pi and Mathematica | Raspberry Pi
Date: 11 mars 2015 10:27
À: Charles Maire Charles.Maire@unige.ch, Bruno Chazelas Bruno.Chazelas@unige.ch, Luc Jean-Marc Weber Luc.JM.Weber@unige.ch

http://www.raspberrypi.org/controlling-telescopes-with-raspberry-pi-and-mathematica/

Controlling Telescopes with Raspberry Pi and Mathematica

- 23 Comments

Eben: Here's a guest post from Tom Sherlock, describing how he's been able to control a telescope using a Raspberry Pi, Mathematica *and the Wolfram Language.*

As an amateur astronomer, I'm always interested in ways to use *Mathematica* in my hobby. In earlier blog posts, I've written about how *Mathematica* can be used to process and improve images taken of planets and nebulae. However, I'd like to be able to control my astronomical hardware directly with the Wolfram Language.

In particular, I've been curious about using the Wolfram Language as a way to drive my telescope mount, for the purpose of automating an observing session. There is precedent for this because some amateurs use their computerized telescopes to hunt down transient phenomena like supernovas. Software already exists for performing many of the tasks that astronomers engage in—locating objects, managing data, and performing image processing. However, it would be quite cool to automate all the different tasks associated with an observing session from one notebook.

Mathematica is highly useful because it can perform many of these operations in a unified manner. For example, *Mathematica* incorporates a vast amount of useful astronomical data, including the celestial coordinates of hundreds of thousands of stars, nebula, galaxies, asteroids, and planets. In addition to this, *Mathematica*'s image processing and data handling functionality are extremely useful when processing astronomical data.

Previously I've done some work interfacing with telescope mounts using an existing library of functions called ASCOM. Although ASCOM is powerful and can drive many devices associated with astronomy, like domes and filter wheels, it is limited because it only works on PCs and needs to be pre-installed on your computer. I wanted to be able to drive my telescope directly from *Mathematica* running on any platform, and without any special set up.

Telescope Serial Communication Protocols

I did some research and determined that many telescope mounts obey one of two serial protocols for their control: the Meade LX200 protocol and the Celestron NexStar protocol.

The LX200 protocol is used by Meade telescopes like the LX200 series as well as the ETX series. The LX200 protocol is also used by many non-Meade telescope mounts, like those produced by Losmandy and Astro-Physics.

The NexStar protocol is used by Celestron telescopes and mounts as well as those manufactured by its parent company, Synta, including the Orion Atlas/Sirius family of computerized mounts.

The full details of these protocols can be found in the Meade Telescope Serial Command Protocol PDF and the NexStar Communication Protocol PDF.

A notable exception is the Paramount series of telescope mounts from Software Bisque, which use the RTS2 (Remote Telescope System) protocol for remote control of robotic observatories. The RTS2 standard describes communication across a TCP/IP link and isn't serial-port based. Support for RTS2 will have to be a future project.

Since *Mathematica* 10 has added direct serial-port support, it's possible to implement these protocols directly in top-level Wolfram Language code and have the same code drive different mounts from *Mathematica* running on different platforms, including Linux, Mac, Windows, and Raspberry Pi.

Example: Slewing the Scope

Here's an example of opening a connection to a telescope mount obeying the LX200 protocol, setting the target and then slewing to that target.

Open the serial port ("/dev/ttyUSB0″) connected to the telescope:

```
theScope = DeviceOpen["Serial",
{"/dev/ttyUSB0", "BaudRate" -> 9600,
"DataBits" -> 8, "Parity" -> None,
"StopBits" -> 1}];
```

First we need a simple utility for issuing a command, waiting for a given amount of time (usually a few seconds), and then reading off the single-character response.

```
ScopeIssueCommand1[theScope_, cmd_String]:=
Module[{},
   DeviceWrite[theScope, cmd];
   Pause[theScopeTimeout];
   FromCharacterCode[DeviceRead[theScope]]
];
```

These are functions for setting the target right ascension and declination in the LX200 protocol. Here, the right ascension (RA) is specified by a string in the form of HH:MM:SS, and the declination (Dec) by a string in the form of DD:MM:SS.

```
ScopeSetTargetRightAscension[theScope_,str_String] := ScopeIssueCommand1[theScope,":Sr"<>str<>"#"];

ScopeSetTargetDeclination[theScope_,str_String] := ScopeIssueCommand1[theScope,":Sd"<>str<>"#"];
```

Now that we have the basics out of the way, in order to slew to a target at coordinates specified by RA and Dec strings, setting the target and then issuing the slew command are combined.

```
ScopeSlewToRADecPrecise[
   theScope_, ra_String, dec_String]:=
Module[{},
   ScopeSetTargetRightAscension[theScope,ra];
   ScopeSetTargetDeclination[theScope, dec];
   ScopeSlewTargetRADec[theScope]
];
```

We can also pass in real values as the coordinates, and then convert them to correctly formatted strings for the above function.

```
ScopeSlewToRADecPrecise[
   theScope_, ra_Real, dec_Real]:=
Module[{rah,ram,ras,rastr,dd,dm,ds,decstr},
   rah=ToString[IntegerPart[ra]];
   ram=ToString[IntegerPart[Abs[FractionalPart[ra]]*60]];
   ras=ToString[IntegerPart[FractionalPart[Abs[
      FractionalPart[ra]]*60]*60]];
   rastr=rah<>":"<>ram<>":"<>ras;
   dd=ToString[IntegerPart[dec]];
   dm=ToString[IntegerPart[Abs[FractionalPart[dec]]*60]];
   ds=ToString[IntegerPart[FractionalPart[Abs[
      FractionalPart[dec]]*60]*60]];
```

```
   decstr=dd<>":"<>dm<>":"<>ds;
   ScopeSlewToRADecPrecise[theScope, rastr, decstr]
];
```

Now we can point the scope to the great globular cluster in Hercules:

```
ScopeSlewToRADecPrecise[theScope,
AstronomicalData["M13","RightAscension"],
AstronomicalData["M13","Declination"]];
```

Slew the scope to the Ring Nebula:

```
ScopeSlewToRADecPrecise[theScope,
NebulaData["M57","RightAscension"],
NebulaData["M57","Declination"]];
```

And slew the scope to Saturn:

```
ScopeSlewToRADec[PlanetData["Saturn","RightAscension"],
PlanetData["Saturn","Declination"]];
```

When the observing session is complete, we can close down the serial connection to the scope.

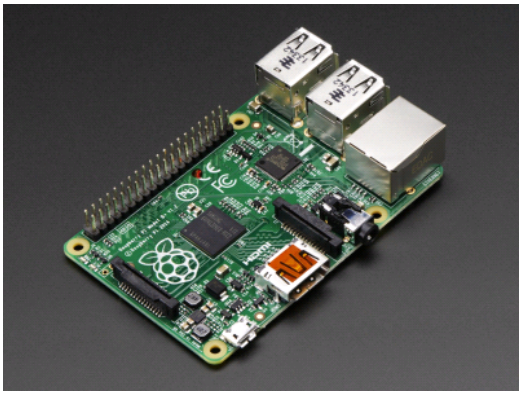
```
DeviceClose[theScope];
```

Please be aware that before trying this on your own scope, you should have limits set up with the mount so that the scope doesn't accidentally crash into things when slewing around. And of course, no astronomical telescope should be operated during the daytime without a proper solar filter in place.

The previous example works with *Mathematica* 10 on all supported platforms. The only thing that needs to change is the name of the serial port. For example, on a Windows machine, the port may be called "COM8″ or such.

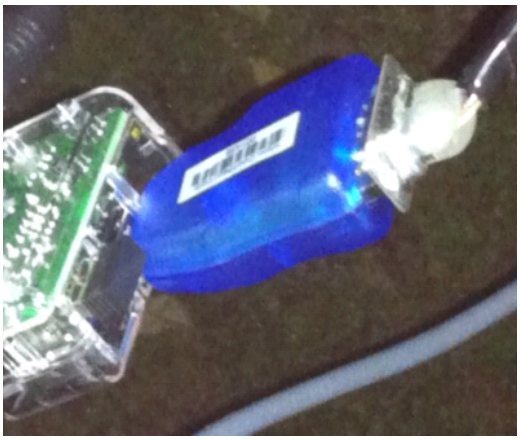
Telescope Control with Raspberry Pi

One interesting platform for telescope control is the Raspberry Pi. This is an inexpensive ($25–$35), low-power-consumption, credit-card-sized computer that runs Linux and is tailor-made for all manner of hackery. Best of all, it comes with a free copy of *Mathematica* included with the operating system.



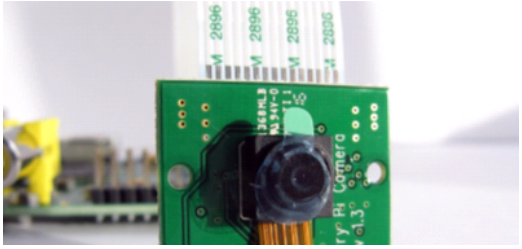
Since the Pi is just a Linux box, the Wolfram Language code for serial-port telescope control works on that too. In fact, since the Pi can easily be wirelessly networked, it is possible to connect to it from inside my house, thus solving the number one problem faced by amateur astronomers, namely, how to keep warm when it's cold outside.

The Pi doesn't have any direct RS-232 ports in hardware, but an inexpensive USB- to-serial adapter provides a plug-n-play port at /dev/ttyUSB0. In this picture, you can see the small wireless network adapter in the USB socket next to the much larger, blue, usb-to-serial adapter.



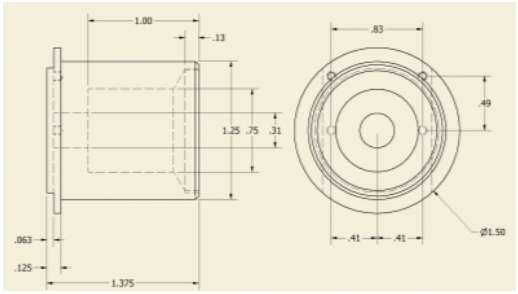
Astrophotography with the Pi

Once I had the Pi controlling the telescope, I wondered if I could use it to take pictures through the scope as well. The Raspberry Pi has an inexpensive camera available for $25, which can take reasonably high-resolution images with a wide variety of exposures.



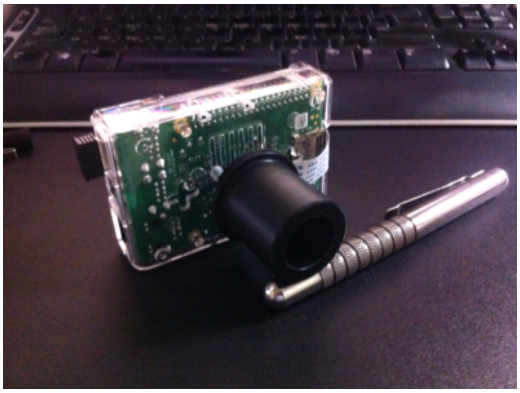
This isn't as good as a dedicated astronomical camera, because it lacks the active cooling needed to take low-noise images of deep sky objects, but it would be appropriate for capturing images of bright objects like planets, the Moon, or (with proper filtering) the Sun.

It was fairly easy to find the mechanical dimensions of the camera board on the internet, design a telescope adapter…

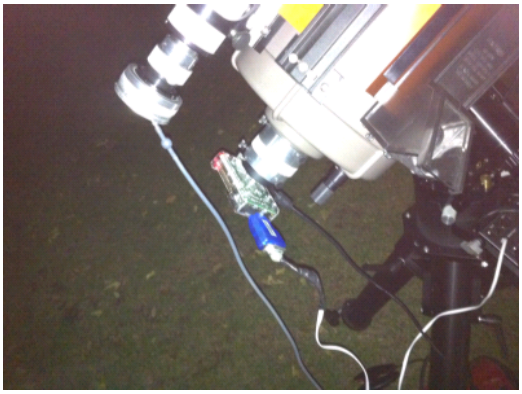


…and then build the adapter using my lathe and a few pennies worth of acetal resin (Dupont Delrin®) I had in my scrap box. The normal lens on the Pi camera was unscrewed and removed to expose the CCD chip directly because the telescope itself forms the image.

Note that this is a pretty fancy adaptor, and one nearly as good could have been made out of 1 1/4 plumbing parts or an old film canister; this is a place where many people have exercised considerable ingenuity. I bolted the adaptor to the side of the Pi case using some 2-56 screws and insulating stand-offs cut from old spray bottle tubing.

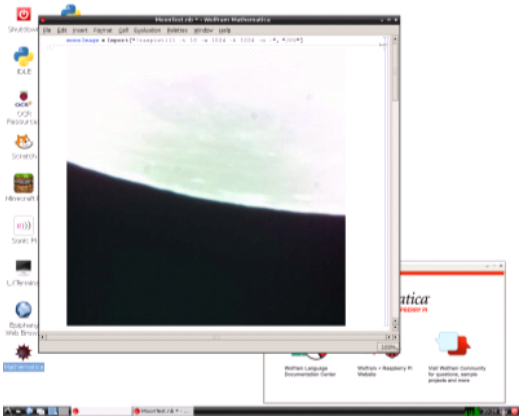


This is how the PiCam looks plugged into the eyepiece port on the back of my telescope, and also plugged into the serial port of my telescope's mount. In this picture, the PiCam is the transparent plastic box at the center. The other camera with the gray cable at the top is the guiding camera I use when taking long exposure astrophotographs.



Remotely Connecting to the PiCam

The Pi is a Linux box, and it can run vncserver to export its desktop. You can then run a vnc client package, like the free TightVNC, on any other computer that is networked to the Pi. This is a screen shot taken from my Windows PC of the TightVNC application displaying the PiCam's desktop. Here, the PiCam is running *Mathematica* and has imported a shot of the Moon's limb from the camera module attached to the telescope via the adapter described above.



It's hard to read in the above screen shot, but here is the line I used to import the image from the Pi's camera module directly into *Mathematica*:

```
moonImage=Import[
"!raspistill -ss 1000 -t 10 -w 1024 -h 1024 -o -",
"JPG"]
```

This command invokes the Pi's raspistill camera utility and captures a 1024×1024 image exposed at 1,000 microseconds after a 10-second delay, and then brings the resulting JPEG file into *Mathematica*.

One problem that I haven't solved is how to easily focus the telescope remotely, because the PiCam's preview image doesn't work over the vnc connection. One interesting possibility would be to have *Mathematica* take a series of exposures while changing the focus via a servo attached to the focus knob of the telescope.

Conclusion

Mathematica and the Wolfram Language provide powerful tools for a wide variety of device control applications. In this case, I've used it on several different platforms to control a variety of astronomical hardware.

@+

Michel Crausaz

```
                                !!!
             Amitiés           (ô ô)        Michel
+-----------------------------oOO--(_)--OOo-----------------------------+
| Michel Crausaz                 ~     | Observatoire Astronomique      |
| Mécanicien-Electronicien             | de l'Université de Genève      |
| Instrumentation   Informatique       |                                |
| Hardware Réseau   Webmaster          | 51, chemin des Maillettes      |
| Tél. : +41  22  379 22 00 Réception  | 1290 Sauverny                  |
| Tél. : +41  22  379 23 44 Direct     | Suisse                         |
| Fax  : +41  22  379 22 05            |                                |
| Email: michel.crausaz@unige.ch       | Natel : +41 79 602 74 36       |
| Web  : www.unige.ch/sciences/astro   | FTP : obsftp.unige.ch          |
+------------------------------oo0---0oo+-----------------------------+
```