Configuring Playstation 3 Controllers

If you're building a robot you will at some point probably want a way to manually drive it around. The Playstation3 controller, also known as the SixAxis, makes for a great option - it connects over bluetooth, has a bundle of different buttons, sticks and motion sensors, and is readily available. You'll probably google for how to make it work with the Raspberry Pi and then, if your experience is anything like mine, you'll find that every single online guide is a) a copy of one original document and b) doesn't work. Having solved all these problems, I thought I'd be nice and write the method down here in the hope that no-one else has to waste the time I've just spent on it...

A note on pairing

One of the reasons the SixAxis isn't as easy as it could be to use is how pairing works. Normal bluetooth devices will establish a link between the device and the host once, then the host can initiate connection using this previously stored information. In the case of the SixAxis, it's actually the controller that initiates the process, so we have to do some setup beforehand. We need to tell the controller to which bluetooth host it should attempt to connect, and we need to tell the host (the Pi) that it should allow the controller's connection.

Hardware

This guide assumes you're using a Raspberry Pi (I'm using a Pi 2, but there's no reason this wouldn't work with older ones). You'll also need a USB bluetooth dongle and, obviously, a SixAxis controller. I've only tried this with genuine Sony ones, many of the cheaper ones you'll find online are clones, they should work but YMMV.

Bluetooth dongles

Some people are finding this guide does not work. I suspect this is down to the bluetooth dongle, having eliminated everything else in the process. The one I'm using is an Asus USB-BT400, it's tiny and supports all the current Bluetooth standards. If you get this to work with a different dongle can you let me know on twitter at @approx_eng_ and I'll add it to this list:

- Asus USB-BT400

Software

Note 1 - this assumes you've set up git and installed a public key with github, you don't have to do this but you'll need to modify some of the git commands below if you haven't. You can set up public keys using the instructions at https://help.github.com/articles/generating-ssh-keys/#platform-all

Note 2 - this is also assuming you're starting from a clean installation of the latest Jessie based Raspbian. Other distributions may need varying combinations of dev libraries etc. For testing I was using the minimal installation with filename `2015-11-21-raspbian-jessie-lite.zip` but these instructions should apply to any recent version. As always, it's not a bad idea to run `sudo apt-get update` and `sudo apt-get upgrade` to get any changes to packages since your distribution was built.

You'll need to install some packages on your Pi first, and enable the bluetooth services:

```
pi@raspberrypi ~ $ sudo apt-get install bluetooth libbluetooth3
libusb-dev
pi@raspberrypi ~ $ sudo systemctl enable bluetooth.service
```

You also need to add the default user to the `bluetooth` group:

```
pi@raspberrypi ~ $ sudo usermod -G bluetooth -a pi
```

You must now **power cycle** your Pi. Do not just reboot, actually shut down, pull the power, wait a few seconds and reconnect. This may be overkill, but it's been the best way I've found to consistently have the next steps succeed.

Pairing

Get and build the command line pairing tool:

```
pi@raspberrypi ~ $ wget http://www.pabr.org/sixlinux/sixpair.c
pi@raspberrypi ~ $ gcc -o sixpair sixpair.c -lusb
```

Firstly we need to tell the controller the address of the bluetooth dongle. To do this you need to connect the controller to your Pi with a mini-USB cable. Also make sure your Pi is powered from an external supply - the extra power needed when you connect the controllers can be too much for a laptop USB socket and you'll get random errors or the process won't work at all. The 'sixpair' command, run as root, updates the controller's bluetooth master address:

```
pi@raspberrypi ~ $ sudo ./sixpair
Current Bluetooth master: 5c:f3:70:66:5c:e2
Setting master bd_addr to 5c:f3:70:66:5c:e2
```

You should see a message indicating that the bluetooth master address on the controller has been changed (you can specify the address to which it should change, the default with no arguments is to use the first installed bluetooth adapter, which is what you want unless for some reason you've got more than one plugged in). The controller will now attempt to connect to your bluetooth dongle when you press the PS button (don't do this just yet, it won't work). The example above shows that no change has been made, as this particular controller had been paired with the dongle before, but you should see two different addresses - the first is the address the controller was trusting, the second is the one it now trusts.

Next we need to configure the bluetooth software on the Pi to accept connections from the controller.

Disconnect your controller from the USB port, and run the 'bluetoothctl' command as a regular user (you don't need to be root for this):

```
pi@raspberrypi ~ $ bluetoothctl
[NEW] Controller 5C:F3:70:66:5C:E2 raspberrypi [default]
... (other messages may appear here if you have other bluetooth
hardware)
```

Now re-connect your controller with the mini-USB cable. You should see messages in the terminal indicating that something has connected (but don't worry if you don't, as long as something useful appears in the next step!)

Type ‘devices’ in the terminal. You will see a list of possible devices, including at least your SixAxis controller. You need to take note of the MAC address of the controller for the next step:

```
[bluetooth]# devices
Device 60:38:0E:CC:OC:E3 PLAYSTATION(R)3 Controller
... (other devices may appear here)
```

Type ‘agent on’ and then ‘trust MAC’, replacing MAC with the MAC address you noted in the previous step (they won’t be the same as mine!). Quit the tool once you’re done.

```
[bluetooth]# agent on
Agent registered
[bluetooth]# trust 60:38:0E:CC:0C:E3
[CHG] Device 60:38:0E:CC:0C:E3 Trusted: yes
Changing 60:38:0E:CC:0C:E3 trust succeeded
[bluetooth]# quit
Agent unregistered
[DEL] Controller 5C:F3:70:66:5C:E2
```

Disconnect your controller, you should now be able to connect wirelessly. To check this, first list everything in /dev/input:

```
pi@raspberrypi ~ $ ls /dev/input
by-id  by-path  event0  event1  event2  event3  event5  mice  mouse0
```

Now press the PS button, the lights on the front of the controller should flash for a couple of seconds then stop, leaving a single light on. If you now look again at the contents of /dev/input you should see a new device, probably called something like ‘js0’:

```
pi@raspberrypi ~ $ ls /dev/input
by-id    event0  event2  event4  js0   mouse0
by-path  event1  event3  event5  mice
```

If a new device has appeared here then congratulations, you have successfully paired your dongle and SixAxis controller. This will persist across reboots, so from now on you can just connect by pressing the PS button on the controller. Pressing and holding this button will shut the controller down - at the moment there's no timeout so be sure to turn the controller off when you're not going to be using it for a while.

Accessing the SixAxis from Python

You now have a joystick device in /dev/input, but how do you use it in your Python code?

There are two different approaches I've tried. You can use PyGame - this has the advantage that you might be using it already (in which case it's the simplest solution) and it's already installed in the system Python on your Pi. It has the drawback though that it requires a display - while I'm aware there are workarounds for this they're not really very satisfactory. The second option is to use the Python bindings for evdev - this is lightweight, but has drawback of being more complex to use and only working on linux, even if you're on a unix-like system such as OSX you can't use it whereas PyGame is generally suitable for cross-platform use. Because I only want to run this on the Pi and because I really need it to work cleanly in a headless environment I've gone with evdev, but there are arguments for both.

Actually using evdev isn't trivial, the best documentation I have is the code I wrote to handle it. I've created a Python class `triangula.input.SixAxis` and corresponding resource

`triangula.input.SixAxisResource` to make this simpler to work with. The class uses asyncore to poll the evdev device, updating internal state within the object. It also allows you to register button handlers which will be called, handles centering, hot zones (regions in the axis range which clamp to 1.0 or -1.0) and dead zones (regions near the centre point which clamp to 0.0).

By way of an example, the following code will connect to the controller (you'll get an exception if you don't have one connected) and print out the values of the two analogue sticks:

```
from triangula.input import SixAxis, SixAxisResource

# Button handler, will be bound to the square button later
def handler(button):
  print 'Button {} pressed'.format(button)

# Get a joystick, this will fail unless the SixAxis controller is
paired and active
# The bind_defaults argument specifies that we should bind actions to
the SELECT and START buttons to
# centre the controller and reset the calibration respectively.
with SixAxisResource(bind_defaults=True) as joystick:
    # Register a button handler for the square button
    joystick.register_button_handler(handler, SixAxis.BUTTON_SQUARE)
    while 1:
        # Read the x and y axes of the left hand stick, the right
hand stick has axes 2 and 3
        x = joystick.axes[0].corrected_value()
        y = joystick.axes[1].corrected_value()
        print(x,y)
```

You're welcome to pick up Triangula's libraries, they're uploaded to PyPi semi-regularly (get with 'pip install triangula') or from github. In either case you'll need to install one extra package first, without which the evdev module won't build:

```
pi@raspberrypi ~ $ sudo apt-get install libpython2.7-dev
```

Now you can get Triangula's code from github and build it to acquire the triangula.input module, you can then use this in your own code (there's nothing particularly specific to Triangula in it)

```
pi@raspberrypi ~ $ git clone git@github.com:basebot/triangula.git
pi@raspberrypi ~ $ cd triangula/src/python
pi@raspberrypi ~/triangula/src/python python setup.py develop
```

This will set up the libraries in develop mode, creating symbolic links into your python installation (I'm assuming here that you're using a virtual environment, because you should be - if you're not you'll need to run some of these commands as root)