spnav Documentation

Release 0.9

Stanley Seibert

February 04, 2012

CONTENTS

The `spnav` module provides a Python interface to the `libspnav` C library, which allows you to read events from a Space Navigator 3D mouse on Linux systems. These input devices simultaneously report linear force and rotational torque applied by the user to the device, along with button events. See:

http://www.3dconnexion.com/products/spacenavigator.html

for more information about the Space Navigator.

Any device supported by `spacenavd` is supported by the `libspnav` and therefore the `spnav` Python module. This includes not only the current USB devices sold by 3dconnexion, but older serial-based devices that were sold under many brand names.

For more information about `spacenavd` and `libspnav`, see:

http://spacenav.sourceforge.net/

CHAPTER

ONE

DOCUMENTATION

1.1 Setup

1.1.1 Prerequisites

To access a Space Navigator (or compatible) device in Linux, you need to run a daemon in the background. The official 3dconnexion drivers provide such a server, but the open source spacenav project provides a vastly superior daemon that I highly recommend.

`spacenavd` can communicate input events with client software using either the X11-based protocol supported by the 3dconnexion drivers, or a local UNIX socket-based protocol. The `libspnav` client library, also produced by the *spacenav* project, can use either protocol.

If you are using Ubuntu 11.04, you can install `spacenavd` and `libspnav` with the following command:

```
sudo apt-get install spacenavd libspnav0
```

Otherwise, you will need to download the sofware from:

> http://spacenav.sourceforge.net

and install it manually.

1.1.2 Package Installation

The `spnav` Python module can installed from PyPI with the command:

```
sudo easy_install spnav
```

or installed from source by running the usual Python installation procedure:

```
sudo python setup.py install
```

The `spnav` module requires `ctypes`, which is standard in Python 2.5 and later, although I have only tested spnav with Python 2.7.

1.1.3 Tips

- `spacenavd` supports USB devices with no additional configuration file, but serial devices do need the port name set in `/etc/spnavrc`.

- Neither `spacenavd` nor the 3dconnexion damon support more than one Space Navigator device connected to a single computer.
- Serial devices may have a different convention for the orientation of the y and z axes. You might need to flip them in the configuration file.
- The X11-based protocol works automatically with X11 forwarding and SSH, allowing you to send input events to software running on a remote computer. Note that `libspnav` and the `spnav` Python module need to be installed on the remote computer for this to work.
- If you experience strange permission problems when the `spacenavd` daemon is started automatically by the Ubuntu boot scripts. If you are having trouble, stop the daemon:

  ```
  sudo service spacenavd stop
  ```

  and then start the daemon manually from a X terminal window:

  ```
  sudo spacenavd
  ```

  Alternatively, try using the direct UNIX socket protocol.

1.2 Usage

Reflecting the design of `libspnav`, the `spnav` Python module can be used two ways, depending upon which protocol you use to communicate with the Space Navigator daemon. Both protocols emit the same event objects.

1.2.1 Space Navigator Events

Space Navigator events come in two varieties: *motion* and *button*.

Motion events result from the application of force to the 3D mouse controller. The strain gauges inside the controller cap can simultaneously resolve both linear force and rotational torque, giving 6 degrees of freedom. The linear force is reported as a signed integer 3-vector, corresponding to the x, y, and z components of the force. The rotational torque is also reported as a signed integer 3-vector, with the components corresponding to torque around the x, y, and z axis.

Button events are generated when a button on the Space Navigator controller is pressed or released. They consist of a button number and a boolean indicating the type of state transition ("pressed" or "released").

See *Event Classes* for details on the event classes.

1.2.2 UNIX Socket Protocol

The UNIX socket protocol is suitable when the client and daemon process will coexist on the same computer. It also allows for the creation of console applications that use the Space Navigator without an X Server.

First, the connection to the Space Navigator daemon must be opened:

```
>>> from spnav import *
>>> spnav_open()
```

The open connection is to a single device and global to the process. An `SpnavConnectionException` will be raised if the connection cannot be made.

Events are generated from device input by `spacenavd` and sent to all connected clients. To perform a blocking wait for the next event, use:

```
>>> event = spnav_wait_event()
```

Warning: `spnav_wait_event()` blocks execution inside the underlying C function in `libspnav`. As a result, the user will not be able to interrupt your Python application with Ctrl-C. `spnav_poll_event()` is almost always a better alternative.

To poll the library to see if an event is available, use:

```
>>> event = spnav_poll_event()
```

If no event is available, the function returns `None`, otherwise it returns an event.

As long as a force is applied to the controller, `spacenavd` will continuously send events to all the clients. If your client does even a moderate amount of computation in response to a Space Navigator event (like rendering a 3D scene, for example), many events will queue up before the next event can be retrieved. This will give the appearance of lag, as motions performed some time in the past are processed too late. In these situations, it is better to clear the event queue after significant calculations:

```
>>> spnav_remove_events(SPNAV_EVENT_MOTION)
```

Typically, only motion events should be removed, although button events can be removed with the `SPNAV_EVENT_BUTTON` argument, and both types of events can be removed from the queue with the `SPNAV_EVENT_ANY` option.

When finished, the socket connection is closed with:

```
>>> spnav_close()
```

1.2.3 X11 Protocol

The X11 protocol was defined by 3dconnexion and is used by the official Space Navigator drivers, as well as `spacenavd`. It uses the X server as a conduit to pass Space Navigator events wrapped up as XEvents to applications, similar to other input devices. This allows the Space Navigator to be used with remote applications via SSH X-Forwarding. However, the X11 protocol can only be used with graphical applications, as will be seen in the following example. If you are writing a console application, you must use the UNIX socket protocol described above.

I have been able to successfully use the X11 protocol with pygame, so the remainder of this usage tutorial will assume you are using `pygame` in your application. Other windowing toolkits may work, and you can always fall back to the UNIX socket protocol.

Once we initialize Pygame and create a window, we can obtain the window manager information and open the connection:

```
>>> wm_info = pygame.display.get_wm_info()
>>> spnav_x11_open(wm_info['display'], wm_info['window'])
```

The X11 protocol communicates with XEvents of a type that are ignored by Pygame by default. Next, we need to enable delivery of these events:

```
>>> pygame.event.set_allowed(pygame.SYSWMEVENT)
```

Now Space Navigator events will be returned in a Pygame event loop:

```
while True:
    for event in pygame.event.get():
        spnav_event = spnav_x11_event(event.event)
        if spnav_event is not None:
            print 'Space Navigator Event:', spnav_event
```

Much the same as with the UNIX socket protocol, Space Navigator events can queue up during extended processing. This creates a lag between current motion by the user and the arrival of those motion events to the front of the queue. There is no `spnav_remove_events()` analog for the X11 protocol, as the queue is handled outside of `libspnav`. However, one can adjust the previous event loop to only return the most recent Space Navigator event:

```
while True:
    for event in pygame.event.get(pygame.SYSWMEVENT)[-1:] \
                 + pygame.event.get():
        spnav_event = spnav_x11_event(event.event)
        if spnav_event is not None:
            print 'Space Navigator Event:', spnav_event
```

When finished, the connection is closed with the same function as in the UNIX socket protocol:

```
>>> spnav_close()
```

1.3 Reference

The `spnav` module interface exactly mirrors the C API of `libspnav`, but the C union of event structs has been replaced with Python classes.

1.3.1 Event Classes

Event types are identified by module constants:

`spnav.`**`SPNAV_EVENT_MOTION`**

Linear and rotation force applied to controller.

`spnav.`**`SPNAV_EVENT_BUTTON`**

Button pressed or released.

`spnav.`**`SPNAV_EVENT_ANY`**

Either motion or button event. Only used with `spnav_remove_events`.

class `spnav.`**`SpnavEvent`**(*ev_type*)
Space Navigator Event Base class

***ev_type*: int** Type of events. Either `SPANV_EVENT_MOTION` or `SPNAV_EVENT_BUTTON`.

class `spnav.`**`SpnavMotionEvent`**(*translation*, *rotation*, *period*)
Space Navigator Motion Event class

***translation*: 3-tuple of ints** Translation force X,Y,Z in arbitrary integer units

***rotation*: 3-tuple of ints** Rotation torque around axes in arbitrary integer units

***period*: int** Corresponds to spnav_event_motion.period in libspnav. No idea what the meaning of the field is.

class `spnav.`**`SpnavButtonEvent`**(*bnum*, *press*)
Space Navigator Button Event class

Button events are generated when a button on the controller is pressed and when it is released.

***bnum*: int** Button number

***press*: bool** If True, button pressed down, else button released.

1.3.2 UNIX Socket Protocol

spnav.**spnav_open**()
Open connection to the daemon via AF_UNIX socket.

The unix domain socket interface is an alternative to the original magellan protocol, and it is *NOT* compatible with the 3D connexion driver. If you wish to remain compatible, use the X11 protocol (spnav_x11_open, see below).

Raises SpnavConnectionException if connection cannot be established.

spnav.**spnav_wait_event**()
Blocks waiting for Space Navigator events.

Note that the block happens inside the libspnav library, so you will not be able to interrupt this function with Ctrl-C. It is almost always better to use spnav_poll_event() instead.

Returns: An instance of SpnavMotionEvent or SpnavButtonEvent.

spnav.**spnav_poll_event**()
Polls for waiting for Space Navigator events.

Returns: None if no waiting events, otherwise an instance of SpnavMotionEvent or SpnavButtonEvent.

spnav.**spnav_remove_events**(*event_type*)
Removes pending Space Navigator events from the queue.

This function is useful to purge old events that may have queued up after a long calculation. It helps to keep your application appearing more responsive.

***event_type*: int** The type of events to remove. SPNAV_EVENT_MOTION or SPNAV_EVENT_BUTTON removes just motion or button events, respectively. SPNAV_EVENT_ANY removes both types of events.

spnav.**spnav_close**()
Closes connection to the daemon.

1.3.3 X11 Socket Protocol

spnav.**spnav_x11_open**(*display*, *window*)
Opens a connection to the daemon, using the original magellan X11 protocol. Any application using this protocol should be compatible with the proprietary 3D connexion driver too.

***display*: PyCObject containing X11 Display struct** X11 display pointer

***window*: int** X11 window handle

Raises SpnavConnectionException if Space Navigator daemon cannot be contacted.

spnav.**spnav_x11_event**(*xevent*)
Examines an arbitrary X11 event to see if it is a Space Navigator event.

Returns: None if not a Space Navigator event, otherwise an instance of SpnavMotionEvent or SpnavButtonEvent is returned.

spnav.**spnav_close**()
Closes connection to the daemon.

1.3.4 Exceptions

exception spnav.**SpnavException**(*msg*)
Base class for all spnav exceptions.

exception spnav.**SpnavConnectionException**(*msg*)
Exception caused by failure to connect to source of spnav events.

exception spnav.**SpnavWaitException**(*msg*)
Exception caused by error while waiting for spnav event to arrive.

CHAPTER

TWO

DEVELOPMENT

The source repository for `spnav` is located at:

http://bitbucket.org/seibert/spnav/

You can download the source code with Mercurial:

```
hg clone http://bitbucket.org/seibert/spnav/
```

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*