Alps TouchPad Driver – version 0.2

Yotam Medini

December 25, 2010

Abstract

This is an informal description of the simple event-based Xorg driver for Alps touch-pad on a notebook

1 Why

Recent Linux kernel and the sophisticated **X11**/input **synaptics** driver provide many options for smart GUI gestures. Alas, with my **Umax 530T** ActionBook, I found that I could do without most of the smart features. But basic features such as smooth continuous pointer movement and simple middle button emulation, are either hard to configure or do not work the way I wish.

Thus I have decided to dive into Synaptics's (`http://web.telia.com/~u89404340/touchpad/`) code, that by now moved to

`http://cgit.freedesktop.org/xorg/driver/xf86-input-synaptics/`

and learn as much as I need. With this knowledge, I could have fun in having my own **X11** input driver for **alps** touch pad.

2 What

Here are the main design points:

1. Do *not* implement tapping.
2. Support emulation of middle button.
3. Pointer movement is independent of finger speed.
4. Implement with **C++**.
5. Avoid floating point computations.

3 How

To simplify things, using and support only the `event` protocol. Here there are two categories of events:

- Mouse-like Buttons — Left and Right buttons, Up and Down states.
- Pad finger movement — With x and y pad coordinates.

There are separate **C++** class handlers for each.

4 Button Handling

When middle button emulation is off, the logic is trivial. So for now, we assume middle button emulation is active. Here are the axioms:

- Left and Right button are symmetrically treated.
- For each logical button: Left, Middle, Right — the posted events must be alternating in their Down and up state.
- When middle button is logically Down, it will be considered UP, after *both* physical buttons are up.

4.1 When is Button Down?

Upon Left and Right buttons physical Down event, we spawn a timer. If it arrives before the peer button is pressed down, we consider it a simple Left or Right button-down event.

When a physical peer button is pressed down (before the timer) within a configured time delta, we consider this a middle-button Down event.

But if the middle-button is already logically down, we *ignore* this physical event.

4.2 When is Button Up?

This is slightly *more* complicated. Say we got a physical S-button Up event, where S is Left or Right. If the logical button reflecting the physical button is down, then we consider it S-Button logical Up event.

Otherwise, we must be in a state of logical middle button Down state. We consider middle button Up logical event, only if the peer of S is already up.

4.3 Conclusions

To satisfy our requirements we need to

- Remember the physical status of the "other" peer button.
- Remember the logical status of all 3 buttons.
- When simple quick button Down and Up events come *before* the timer event sent after the Down event, we have to return a *pair* of logical Down and Up events.

5 Touch Pad Handling

Say the pad rectangle coordinates are: $[x_{\min}, x_{\max}) \times [y_{\min}, y_{\max})$. The task is to map movement on the pad to pixel movements on the screen. Here, we will deal with the x-coordinate, the y-coordinate can be similarly treated.

Define

$$x_c = (x_{\min} + x_{\max})/2$$
$$r = (x_{\max} - x_{\min})/2$$

We want the ratio of screen change to pad change have value of $F_c = n_c/d_c$ in the center of the pad, and a value of $F_e = n_e/d_e$ in the pad left and right edges. The polynomial

$$p(x) = a(x - x_c)^3 + b(x - x_c)$$

should achieve it. Differentiating we get

$$p'(x) = 3a(x - x_c)^2 + b$$

leading to the equations:

$$\begin{aligned} F_c &= b \\ F_e &= 3ar^2 + b \end{aligned}$$

with the solution

$$a = (F_e - F_c)/3r^2 \qquad b = F_c$$

Now the polynomial looks like

$$\begin{aligned} p(x) &= a(x - x_c)^3 + b(x - x_c) \\ &= \big(a(x - x_c)^2 + b\big)(x - x_c) \\ &= \Big(\big((F_e - F_c)/3r^2\big)(x - x_c)^2 + F_c\Big)(x - x_c) \\ &= \frac{1}{3r^2 d_c d_e}\big((n_e d_c - n_c d_e)(x - x_c)^2 + 3r^2 n_c d_e\big)(x - x_c) \\ &= \big(A(x - x_c)^2 + B\big)(x - x_c)/D \end{aligned}$$

where

$$A = n_e d_c - n_c d_e \qquad B = 3r^2 n_c d_e \qquad D = 3r^2 d_c d_e$$

6 Configuration

Name	**Number**
`EmulateMidButtonTime`	1
`{x,y}}Pad{Min,Max}`	$2 \cdot 1 \cdot 2 = 4$
`{x,y}{Center,Edge}{Numerator,Denominator}`	$2 \cdot 2 \cdot 2 = 8$
`{x,y}Jump`	2
Total:	15

Here they are explicitly with their default values (DV).

Name	DV	Comment
`EmulateMidButtonTime`	75	Milliseconds between right and left clicks, 0 for no emulation
`xPadMin`	130	Pad left edge
`xPadMax`	840	Pad right edge
`yPadMin`	130	Pad top edge
`yPadMax`	640	Pad bottom edge
`xCenterNumerator`	1	n_c for x axis
`xCenterDenominator`	1	d_c for x axis
`xEdgeNumerator`	1	n_e for x axis
`xEdgeDenominator`	1	d_e for x axis
`yCenterNumerator`	1	n_c for y axis
`yCenterDenominator`	1	d_c for y axis
`yEdgeNumerator`	1	n_e for y axis
`yEdgeDenominator`	1	d_e for y axis
`xJump`	16	x-Jump threshold to be ignored
`yJump`	16	y-Jump threshold to be ignored

The configuration values may be specified in

- Configuration via **udev**.
- Configuration via **xorg.cong**.
- Configuration via **hal**. In recent GNU/Linux distributions this is considered obsolete.

6.1 Examples

6.2 Xorg Configuration

Here is a working example of `/etc/X11/xorg.conf`)

```
# Example working on UMAX 530T ActioBook.
Section "Device"
    Identifier  "NeoMagic Corporation NM2160 [MagicGraph 128XD]"
    Driver      "neomagic"
    BusID       "PCI:0:2:0"
EndSection

Section "Monitor"
    Identifier  "Generic Monitor"
    Option      "DPMS"
    HorizSync   31.5-48.5
    VertRefresh 40-70
EndSection

Section "Screen"
    Identifier  "Default Screen"
    Device      "NeoMagic Corporation NM2160 [MagicGraph 128XD]"
    Monitor     "Generic Monitor"
    DefaultDepth    16
    SubSection "Display"
        Depth       16
        Modes       "1024x768"
    EndSubSection
EndSection

Section "InputDevice"
    Identifier  "Configured Mouse"
    Driver      "alps"
    Option      "CorePointer"
    Option      "Device"        "/dev/input/mice"
EndSection

Section "ServerLayout"
    Identifier  "Default Layout"
    Screen      "Default Screen"
    InputDevice "Configured Mouse"
EndSection
```

6.3 UDEV configuration

The following `/etc/udev/rules.d/20-xorg-alps.rules` is suppose to select the driver for the **AlpsPS/2** device. But it seems that once the *synaptics* driver is found in the function `checkCoreInputDevices`

of `hw/xfree86/common/xf86Config.c`, then it is set to handle the touchpad. THis is regardless of **udev** settings. Therefore, rather than further struggling, I simplly moved it away, and used the good old `xorg.conf` configuration.

```
# Example working on UMAX 530T ActioBook.
Section "Device"
    Identifier  "NeoMagic Corporation NM2160 [MagicGraph 128XD]"
    Driver      "neomagic"
    BusID       "PCI:0:2:0"
EndSection

Section "Monitor"
    Identifier  "Generic Monitor"
    Option      "DPMS"
    HorizSync   31.5-48.5
    VertRefresh 40-70
EndSection

Section "Screen"
    Identifier  "Default Screen"
    Device      "NeoMagic Corporation NM2160 [MagicGraph 128XD]"
    Monitor     "Generic Monitor"
    DefaultDepth    16
    SubSection "Display"
        Depth       16
        Modes       "1024x768"
    EndSubSection
EndSection

Section "InputDevice"
    Identifier  "Configured Mouse"
    Driver      "alps"
    Option      "CorePointer"
    Option      "Device"        "/dev/input/mice"
EndSection

Section "ServerLayout"
    Identifier  "Default Layout"
    Screen      "Default Screen"
    InputDevice "Configured Mouse"
EndSection
```

6.4 HAL Configuration

Here is a configuration example that I used ti use with the (buy now obsolete) HAL system. It was expected to be placed in

`/usr/share/hal/fdi/policy/20thirdparty/12-x11-alps.fdi`

in *hal*'s (`.fdi`) syntax.

```
<?xml version="0.1" encoding="ISO-8859-1"?>
<deviceinfo version="0.2">
  <device>
    <match key="info.capabilities" contains="input.touchpad">
      <match key="info.product" contains="AlpsPS/2 ALPS">
```

```
        <merge key="input.x11_driver" type="string">alps</merge>
        <merge key="input.x11_options.EmulateMidButtonTime" type="string">75</merge>
        <merge key="input.x11_options.xPadMin" type="string">130</merge>
        <merge key="input.x11_options.xPadMax" type="string">840</merge>
        <merge key="input.x11_options.xCenterNumerator" type="string">1</merge>
        <merge key="input.x11_options.xCenterDenominator" type="string">1</merge>
        <merge key="input.x11_options.xEdgeNumerator" type="string">1</merge>
        <merge key="input.x11_options.xEdgeDenominator" type="string">1</merge>
        <merge key="input.x11_options.xJump" type="string">16</merge>
        <merge key="input.x11_options.yPadMin" type="string">130</merge>
        <merge key="input.x11_options.yPadMax" type="string">640</merge>
        <merge key="input.x11_options.yCenterNumerator" type="string">1</merge>
        <merge key="input.x11_options.yCenterDenominator" type="string">1</merge>
        <merge key="input.x11_options.yEdgeNumerator" type="string">1</merge>
        <merge key="input.x11_options.yEdgeDenominator" type="string">1</merge>
        <merge key="input.x11_options.yJump" type="string">16</merge>
     </match>
        <!--
        For other possible options, check CONFIGURATION DETAILS in alps source
        -->
    </match>
  </device>
</deviceinfo>
```

Note: Such configuration options used to be placed in some section inside `/etc/X11/xorg.conf`.

7 Who is Calling us Anyway?

This section started in the quest to find out who is calling our input routine `Alps::sReadInput`. More embarrassingly, in debugging phase, why *aren't* we being called.

Upon initialization, we set the field of `IDevPtr::read_input` with our static `Alps::sReadInput(...)` function.

Following is the assumed calling stack. In small type, the callback setting functions.

Function	**File**
xf86SigioReadInput	hw/xfree86/common/xf86Events.c
xf86AddEnabledDevice	hw/xfree86/common/xf86Events.c
xf86InstallSIGIOHandler	hw/xfree86/os-support/shared/sigio.c
xf86SIGIO	hw/xfree86/os-support/shared/sigio.c

8 Live Configuration

Similar to the inspiring synaptics driver, this driver allows for parameters configuration while running. This is implemented by

- Putting the parameters in shared memory (like synaptics does).
- Using a `alps-cfg.py` Python (and PyGtk) script that controls these parameters.

This driver feature can be canceled at compile time by defining the **C**/**C++** compiler flag `-DNO_SHM=1`.

The `alps-cfg.py` tool can work both in command-line mode or via GUI. See its options by entering:

```
# alps-cfg.py --help
```

9 Build & Install

In order to build, you may need to pre-install the following packages:
g++, **make**, **automake**, **x11proto-input-dev**, **xserver-xorg-dev**, **xutils-dev**.

To build issue the following:

```
$ ./autogen.sh
$ make
```

To install, issue (with *root* privileges):

```
$ cp src/.libs/alps_drv.so /usr/lib/xorg/modules/input
$ cd /usr/lib/xorg/modules/input
$ strip alps_drv.so
$ mv synaptics_drv.so synaptics_drv-inactive.so
```

Also edit `/etc/X11/xorg.conf` to set the `Driver` of the `"InputDevice"` section, see 6.2.