Raspberry Pi

行車攝錄儀

Author: Nick Lee

Preparations

Plug in the camera module. Update the system:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo raspi-config
```

Select **Interfacing Options**, enable **Camera**.

Finish and **reboot**.

Install LCD driver

Plug in LCD.

Go to the following page:

http://www.waveshare.com/wiki/5inch_HDMI_LCD

Look for a zip file called:

LCD-show-??????.tar.gz

and download it to Raspberry Pi.

```
$ tar zxf LCD-show-??????.tar.gz  # unzip
$ cd LCD-show
$ sudo ./LCD5-show
```

It should reboot, and you should see system messages printing on LCD as it is starting up.

Calibrate touchscreen

The mouse pointer is not entirely accurate. The touchscreen needs calibration.

```
$ sudo dpkg -i -B xinput-calibrator_0.7.5-1_armhf.deb
$ DISPLAY=:0.0 xinput_calibrator
```

`DISPLAY=:0.0` to force graphical display to LCD screen.

There will be a prompt for **four-point calibration** on the LCD screen. Click the points one by one to finish the process.

The new calibration data should be displayed in the terminal, like this:

```
Section "InputClass"
        Identifier      "calibration"
        MatchProduct    "ADS7846 Touchscreen"
        Option  "Calibration"   "162 3988 195 3862"
EndSection
```

Put them in the file **`/etc/X11/xorg.conf.d/99-calibration.conf`** to make the calibration permanent.

Reboot and check the touchscreen again.

The LCD may turn dark after a period of inactivity. Don't panic. Touch it to turn it back on.

raspistill

capture photo

raspistill -o out.jpg	Capture photo to a file.

To view the photo, you have to use VNC or connect a monitor through HDMI. Open the file manager, and double-click on the file.

raspistill -o out.jpg -t 1000	Wait 1 second before capturing. Default is 5 seconds.
raspistill -o out.jpg -hf -vf	Flip image horizontally and vertically.
raspistill | less	Display options.

Interesting options:

-t, --timeout
-hf, --hflip
-vf, --vflip
-w, --width
-h, --height
-ss, --shutter
-ifx, --imxfx

raspivid

capture video

raspivid -o vid.h264	Record a 5-second video.
omxplayer -p -o hdmi vid.h264	View the video on an HDMI display. This will not work on VNC because **omxplayer** makes use of hardware acceleration in the GPU (graphics processing unit), which is only available to HDMI output.
raspivid -o vid.h264 -t 2000	Record a 2-second video.
raspivid | less	Display options. They are similar to those of **raspistill**.

Capturing to a file

```
from time import sleep
from picamera import PiCamera

camera = PiCamera()
camera.resolution = (1024, 768)
sleep(2)  # Camera warm-up time
camera.capture('foo.jpg')
```

But how do you view the photo?

- Use **File Manager** on the graphical desktop
- Use **SFTP** to transfer the file to PC
- Use **VNC** to log into graphical desktop remotely

The **picamera** package has its own very good documentations. Explore it yourself if you want to know more:

http://picamera.readthedocs.io/

Recording video to a file

```
import picamera

camera = picamera.PiCamera()
camera.resolution = (640, 480)

camera.start_preview()
camera.start_recording('video.h264')

camera.wait_recording(5)

camera.stop_recording()
camera.stop_preview()
```

How do you view the video?

- Use **omxplayer** from command line
- Use **SFTP** to transfer the file to PC, then use **VLC** (or any video player) to play it.

Overlaying text on the output

The camera includes a rudimentary annotation facility which permits up to 255 characters of ASCII text to be overlaid on all output.

```
import picamera

camera = picamera.PiCamera(resolution=(1280, 720), framerate=24)

camera.annotate_background = picamera.Color('black')
camera.annotate_text = 'Hehehehe'

camera.start_preview()
camera.start_recording('video.h264')
camera.wait_recording(3)
camera.annotate_text = 'Wahahahahaha'
camera.wait_recording(3)
camera.stop_recording()
camera.stop_preview()
```

Recording over multiple files

```
import picamera

camera = picamera.PiCamera(resolution=(640, 480))
for filename in camera.record_sequence(
        '%d.h264' % i for i in range(1, 5)):
    camera.wait_recording(5)
```

This would split the recording into 4 files, each 5 seconds long.

We supply **a sequence of values** to **`record_sequence()`** so it knows what filenames to record into. We will use the same technique later on to record **indefinitely** into a sequence of files each 15 minutes long.

Split video into longer clips

In real life, five seconds is too short for a video clip. A reasonable length is 15 minutes. How do we make them 15 minutes long?

Another issue to consider is this. Suppose the program starts on the hour (e.g. 12:00, 1:00, 2:00, etc), video clips will then be expected at 00, 15, 30, and 45 minute. Very easy to remember and visualize.

However, next time the program may start at 5 past the hour (e.g. 12:05, 1:05, 2:05, etc), and video clips will be expected at 05, 20, 35, and 50 minute. This creates an inconsistency that makes it less user-friendly.

A better strategy is this. Regardless of when recording starts initially, **clips always begin at 00, 15, 30, and 45 in the hour**.

This implies `wait_recording()` should always wait 15 minutes, **except for the first clip**. If the first clip starts at 10, `wait_recording()` should only wait 5 minutes. We need an intelligent way to determine how long the first clip has to wait.

```
def seconds_to_go():
    length = 15 * 60

    now = datetime.now()
    seconds = now.minute * 60 + now.second

    return length - seconds % length
```

Record indefinitely

In real life, we also want to record indefinitely.

So far, we have supplied a finite sequence of filenames to `camera.record_sequence()`. To record indefinitely, we have to supply an **infinite sequence**.

We also have to decide how to compose filenames. A common method is to use the time of recording. I suggest the format YYYYMMDD-HHmm, e.g. 20160708-0715, followed by the appropriate extension.

Back to the original issue, how do we supply an infinite sequence of filenames?

```
clip_directory = '/home/pi/where/are/the/clips'

def datetime_filename():
  while 1:
    yield (
      os.path.join(
        clip_directory,
        datetime.now().strftime('%Y%m%d-%H%M') + '.h264'))
```

Any function with the `yield` keyword is called a generator. The above generator produces a filename according to current time.

It is a good practice to gather all clips one place. That's why I define `clip_directory` above.

The following page shows you a complete working sample.

```
import picamera
import os.path
from datetime import datetime

clip_directory = '/home/pi/where/are/the/clips'

def record(camera):
    def datetime_filename():
        while 1:
            yield (
                os.path.join(
                    clip_directory,
                    datetime.now().strftime('%Y%m%d-%H%M') + '.h264'))

    def seconds_to_go():
        length = 15 * 60
        now = datetime.now()
        seconds = now.minute * 60 + now.second
        return length - seconds % length

    for filename in camera.record_sequence(datetime_filename()):
        camera.wait_recording(seconds_to_go())


video_size = (640, 480)
preview_dimension = (160, 0, 640, 480)  # x, y, w, h

camera = picamera.PiCamera(resolution=video_size)
camera.start_preview(fullscreen=False, window=preview_dimension)

record(camera)
```

GPS

Plug in GPS module. Check its presence on USB:

```
$ lsusb
```

Install software:

```
$ sudo apt-get install gpsd gpsd-clients
$ sudo pip3 install gpsd-py3
```

`gpsd` is a program that monitors one or more GPS receivers attached through serial or USB ports, making all data of the sensors available to be queried on TCP port 2947.

Check if port 2947 is being listened on:

```
$ netstat -ant
Proto Recv-Q Send-Q Local Address     Foreign Address    State
tcp        0      0 127.0.0.1:2947    0.0.0.0:*          LISTEN
```

Discover GPS module's device path. It should be something like **/dev/ttyUSB0** or **/dev/ttyACM0**.

Edit file `/etc/default/gpsd`, modify `DEVICES` to your GPS module's device path and add the -n option (which will be useful later for syncing the system clock to GPS time):

```
DEVICES="/dev/tty????"
GPSD_OPTIONS="-n"
```

Reboot, or restart gpsd service:

```
$ sudo systemctl restart gpsd.service
```

GPSD client

```
$ cgps
```

You should see a very primitive user interface displaying GPS data. Ignore those weird characters marking the borders - I failed to fix them on PuTTY.

When indoor, the GPS sensor may not be able to get a fix (location), but it should get a time from GPS satellites. The time may not be accurate initially. Be patient. Give it a few minutes.

You should mostly see two types of reports rolling in:

TPV - a time-position-velocity report
SKY - a sky view of the GPS satellite positions

Type **q** to quit.

In a graphical environment (e.g. VNC), you may also try `xgps`. It is a better-looking version of `cgps`.

To better understand the GPS clients:

http://www.catb.org/gpsd/client-howto.html

To better understand TPV and SKY reports:

http://www.catb.org/gpsd/gpsd_json.html

Read GPS data

```
import time
import gpsd

gpsd.connect()

while 1:
    try:
        packet = gpsd.get_current()
        print(packet.position())
    except gpsd.NoFixError:
        print('No position yet')
    finally:
        time.sleep(3)
```

Warning: Do not name your file `gpsd.py`. Python will mistake it as the gpsd module you are importing.

Put date/time and location on video

Recall that, to overlay text on video, we have to modify the property `camera.annotate_text`. We compose a string based on the current date/time and the GPS location, and assign that string to `camera.annotate_text`. It should also be updated every second to give the perception of a running clock.

I use **two threads** to achieve those effects. One thread reads GPS locations; the other refreshes `camera.annotate_text` every second.

In total, we now have 3 threads:

- **main thread** records video in 15-minute clips.
- **GPS thread** reads locations.
- **annotate thread** updates the text overlay on video every second.

A complete working sample, named **`gpscam_5inch_py3.py`**, is on:

https://gist.github.com/nickoala

Enough disk space?

Video files can be quite large. The way video is encoded, the more the movements, the larger the disk space required. I share my experience below:

Video resolution: 640 x 480
Frame per second: 30
Duration: 15 minutes
File size: **20M** (little to no movement) - **400M** (lots of movements)

Depending on the amount of movements, the disk can be filled up rather quickly. We need a way to clean up old files.

Cleaning up old files

How old a file do we want to remove? Another way to look at it is: How long do we want to keep video? Let's say, **we want to keep the latest 3 hours of video**.

Plan A: We look at the current system time, and remove clips that are more than 3 hours old.

But this plan won't work. Imagine:

1. Last evening, you drove home from work. The clips recording your drive home were kept on the Pi.

2. This morning, you prepare to drive to work. As you turn on the Pi, it sees last evening's clips being more than 3 hours old, and removes them.

Recent clips are not very secure. We need a better plan.

Plan B: Regardless of time, we keep the newest bunch of clips. Each clip being 15-minute, 3 hours is equivalent to 12 clips. That is, we always keep the most recent 12 clips. Anything older is removed.

```
import os
import glob

clip_directory = '/PATH/TO/THE/CLIPS'

files = glob.glob(os.path.join(clip_directory, '*.h264'))  ❶
files = sorted(files,
               key=lambda path: os.stat(path).st_mtime,  ❷
               reverse=True)  ❸

for path in files[12:]:  ❹
    os.remove(path)  ❺
```

❶ Select all h264 files from target directory. I want to be specific about h264 because I don't want to accidentally remove other files.

❷ Sort the files by modification time. The `key` parameter is a *function* that extracts the modification time from a path. After extraction, the modification time is used as the key to sort elements in `files`.

❸ Make sure the most recent goes first.

❹ Anything beyond the 12th element ...

❺ ... are removed.

What happens if `files` is shorter than 12 elements? Would line ❹ give any problem?

Run cleanup script periodically

How often do you want to run the cleanup script? Let's run it **every minute** so old files get cleaned up quickly.

Linux has a task scheduler called **cron**. To run a program periodically, you add a **cron job**.

```
$ crontab -e
```

It leads you to a text editor (e.g. nano), in which you add this line:

```
* * * * * python3 /PATH/TO/CLEANUP/SCRIPT
```

Save and **exit**. And this command will be run every minute.

Those five stars correspond to five fields:

```
# ┌──────────── min (0 - 59)
# │ ┌────────── hour (0 - 23)
# │ │ ┌──────── day of month (1 - 31)
# │ │ │ ┌────── month (1 - 12)
# │ │ │ │ ┌──── day of week (0 - 6, Sunday=0)
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ │
# * * * * *  command to execute
```

You may schedule a command in a lot of ways, e.g. at 00:01 every night, or at 15:20 every Wednesday, or every 2 hour on the 1st day of January, April, July, and October. There are a lot of examples on the web.

Run on Startup

Add a system service file:
`~/.config/systemd/user/gpscam.service`

```
[Unit]
Description=GPS Camera Service
After=graphical.target

[Service]
ExecStart=/usr/bin/python3 /home/pi/gpscam/gpscam.py

[Install]
WantedBy=default.target
```

Load the new stuff:

```
$ systemctl --user daemon-reload
```

To start/stop the service by hand:

```
$ systemctl --user start gpscam.service
$ systemctl --user stop gpscam.service
```

To enable/disable it on startup:

```
$ systemctl --user enable gpscam.service
$ systemctl --user disable gpscam.service
```

GPS as a Time Service

Raspberry Pi has no built-in real time clock. When it loses power, the time is lost too. Every time it starts up, it obtains the current time from a time server on the internet. It will sync with the time servers periodically to keep the system clock correct.

If the Pi does not have internet connection, time-keeping may be a problem. Another source of time is GPS satellites. They emit timestamps. We can sync clock using GPS signals.

Disable systemd-timesyncd

Raspbian's default time keeper lacks the ability to use GPS signals as clock signals. Before installing a new and more powerful time keeper, we should first disable the existing one:

```
$ sudo systemctl stop systemd-timesyncd
$ sudo systemctl disable systemd-timesyncd
```

Then, we can install a new one:

```
$ sudo apt-get install ntp
```

NTP, Network Time Protocol

The program to sync clock is called NTP, Network Time Protocol. It constantly talks to a number of time servers on the internet.

$ ntpq -p

```
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+103.253.42.176  91.189.94.4      3 u   12   64  377  377.560    0.880   0.802
*218.189.210.4   128.252.19.1     2 u   13   64  377   17.296   19.322   0.314
```

Pay attention to a few things here.

On each line, **the first character** indicates the usefulness of that time source:

+, *****, **#** are useful
−, **x**, or blank are not useful

Reach tells whether it was polled successfully. 0 is not good. Anything bigger than 0 is good.

Poll tells how often (in seconds) it is polled.

When tells how long (in seconds) since it was last polled.

Further references:

https://www.freesoftwareservers.com/wiki/ntpq-p-output-explained-5406783.html

http://tech.kulish.com/2007/10/30/ntp-ntpq-output-explained/

/etc/ntp.conf

We need to tell ntp where to get the GPS time.

In file `/etc/ntp.conf`, find a section that looks like this:

```
pool 0.debian.pool.ntp.org iburst
pool 1.debian.pool.ntp.org iburst
pool 2.debian.pool.ntp.org iburst
pool 3.debian.pool.ntp.org iburst
```

Before this section, add the following lines:

```
server 127.127.28.0 minpoll 4 maxpoll 4 iburst prefer
fudge 127.127.28.0 time1 +0.105 flag1 1 refid GPSD stratum 1
```

127.127.28.0 is a special address pointing to a **shared memory** location. GPSD writes data to it, ntp reads from it.

`minpoll 4` and `maxpoll 4` tells it to check the time every $2^4 = 16$ seconds.

Restart ntp:

```
$ sudo systemctl restart ntp.service
```

Use `ntpq -p` to check if the system clock is picking up GPS time.

Desktop Shortcut

It is easy to create a desktop shortcut for your program.

Put these content into a file named `start-gpscam.desktop` in the directory `~/Desktop`:

```
[Desktop Entry]
Name=Start GPS Cam
Exec=systemctl --user start gpscam.service
Type=Application
Icon=/usr/share/pixmaps/python3.xpm
```

Try to create another icon to **stop** the program.

Disable Screensaver

Edit the file `/etc/xdg/lxsession/LXDE-pi/autostart`.
Add these lines:

```
@xset s noblank
@xset s off
@xset -dpms
```

It may work, or it may not. There are slightly different ways on the web to try to make it work. I have tried several, but none worked for me. I hope you have better luck.

Remove Wastebasket from desktop

1. Right-click on desktop
2. Select **Desktop Preferences**
3. Select the tab **Desktop Icons**
4. Uncheck **Show "Wastebasket" icon on desktop**

Hide Taskbar

1. Right-click on taskbar
2. Selecct **Panel Settings**
3. Select the tab **Advanced**
4. Check **Minimize Panel when not in use**

Hide Mouse Pointer

```
$ sudo apt-get install unclutter
```

Change Screen Orientation

Edit file `/boot/config.txt`. Look for the line:

`dtoverlay=waveshare32b:rotate=<some number>`

Change the rotate angle. Reboot. Then, **re-calibrate the touchscreen**.

Change Camera's Output Orientation

```
camera.hflip = True
camera.vflip = True
camera.rotation = 90
```

Convert h264 to mp4

```
$ sudo apt-get install gpac
```

To concatenate many files to a new file (if the destination file exists, its original content will be erased):

```
$ MP4Box -cat a.h264 -cat b.h264 -new c.mp4
```

To concatenate many files to an existing file (if the destination file does not exist, it will be created):

```
$ MP4Box -cat a.h264 -cat b.h264 c.mp4
```

Send mp4 to Telegram

```
import sys
import os
import glob
import subprocess
import telepot

TOKEN, chat_id = sys.argv[1:]
bot = telepot.Bot(TOKEN)

h264_directory = '/PATH/TO/H264/DIRECTORY'
mp4_directory = '/PATH/TO/MP4/DIRECTORY'

h264_paths = glob.glob(os.path.join(h264_directory, '*.h264'))
h264_paths = sorted(h264_paths,
                    key=lambda path: os.stat(path).st_mtime,
                    reverse=True) ❶

if len(h264_paths) >= 2:
    h264_target = h264_paths[1]  ❷

    mp4_filename = os.path.splitext(  ❹
                       os.path.basename(  ❸
                           h264_target))[0] + '.mp4'  ❺

    mp4_path = os.path.join(mp4_directory, mp4_filename)

    if not os.path.exists(mp4_path):  ❻
        code = subprocess.call(['MP4Box', '-quiet',  ❼
                                '-cat', h264_target,
                                '-new', mp4_path])
        if code == 0:  ❽
            with open(mp4_path, 'rb') as f:
                bot.sendVideo(chat_id, f)  ❾
```

注意：Telegram 只容許 50MB 以下的檔案

❶ 將 h264 檔由新至舊排序

❷ 選擇次新。不要最新，因為它正被編寫（錄像途中）

❹ 將檔名拆成前後半部，例如 abc.h264 變成 abc 與 h264

❸ 抽出路徑的檔名部分，例如/home/pi/abc.h264 變成 abc.h264

❺ 將檔名伸延 mp4 補上，即係 abc 變做 abc.mp4，就是將要製造的 mp4 檔名

❻ 如果 mp4 不存在的話……

❼ 咁就用 MP4Box 將 h264 轉 mp4

❽ 如果 MP4Box 成功執行……

❾ 咁就寄給 Telegram 用戶，大功告成！畢業！