Sync your videos using reference audio

Reference audio is a common method for synchronization of videos from different cameras, when more expensive equipment that does this automatically is not available. The most common use case is of course filming a scene from several different angles, but there are other setups that may require the same technique. Recently, I had to film several scenes in succession, over which the same audio was playing. I needed to extract a part from each scene, so that exactly the same audio would be playing over each scene part (they all have the same duration, of course). So far, nothing is new. However, what is surprising is how easy it is to do that with a handful of open source tools, as I will be showing here. In particular we will be using Praat, ffmpeg and Python.

What is reference audio?

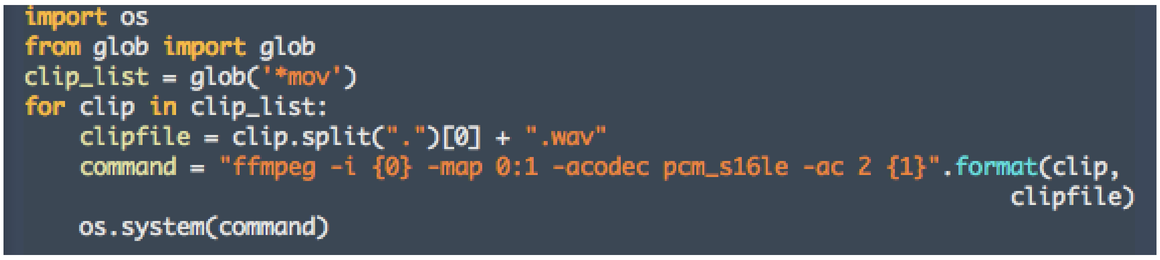
In filming/audio recording it is common to have a pilot/reference audio in order to keep everything in sync. If you are filming a scene, it is possible to set all your cameras to record audio from their own microphones. Depending on the setting/equipment differences etc, they will not all have the *same* audio, but it will be close enough for the method I will show here. One important note for outdoor filming: be aware of the speed of sound. If one of your cameras is 15 m away from the source of the audio (place of action), the sound arrives with a delay of about 40 ms. This may not be so crucial to you, but if you want to eliminate these differences, consider feeding the master audio to the cameras via cable (and splitters/mixer outs) and use that as reference audio. Whatever method you use, I will assume that you end up with a bunch of videos that have the reference audio in their audio track.

Step 1: Extracting the audio from the individual video files

I used **ffmpeg** to extract the audio (make sure you have a recent version, this is based on 2.3.3 on a Mac OS X). The command you need to give is this:

```
ffmpeg -i input.mov -map 0:1 -acodec pcm_s16le -ac 2 output.wav
```

Maybe you don't have MOV files but you have something else, but ffmpeg is good at converting many different formats. We will need to extract audio from all our cameras. If you have just two, then you can just do this twice, but what if you have more? I had to sync six video parts. In that case, we make use of a Python script using the very useful `glob` package:



```
import os
from glob import glob
clip_list = glob('*mov')
for clip in clip_list:
    clipfile = clip.split(".")[0] + ".wav"
    command = "ffmpeg -i {0} -map 0:1 -acodec pcm_s16le -ac 2 {1}".format(clip,
                                                                          clipfile)
    os.system(command)
```

The above code retrieves the list of *mov files in the current directory. It extracts the audio from each one into the current directory, under the same name, with the extension *wav.

Step 2: Finding the offset between two WAV files

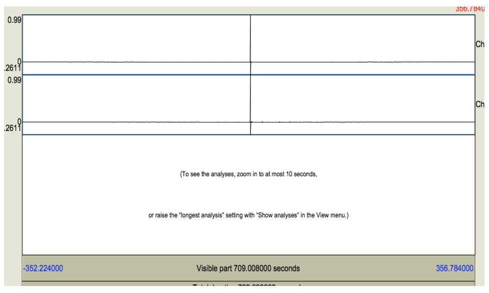
This is where the fun begins! In order to find the offset between the two WAV files, we use *cross-correlation.* Cross-correlation is a mathematical function that tells us how similar two series of numbers are. Consider the following two series:

```
1, 6, 3, 2, 7
2, 12, 6, 4, 14
```

Did you see the resemblance? The second series is the same as the first one but each element is multiplied by 2. Obviously they are very similar, so that the value of their cross-correlation function is 1(which is the maximum). Now, if we rotate the values of the second series (shift 1 position to the left, bring the leftmost item to the right) we get the following pair:

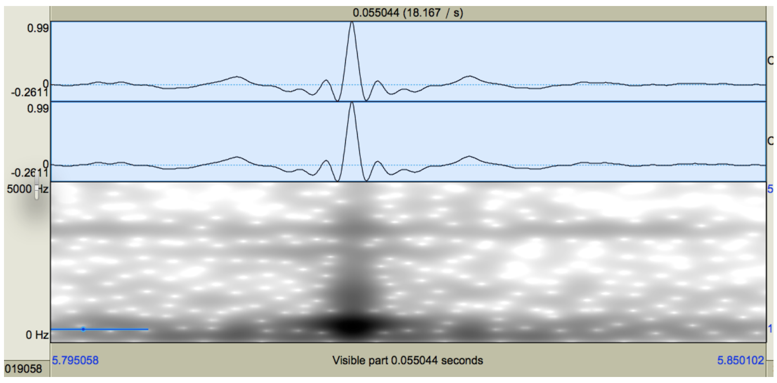
```
1, 6, 3, 2, 7
12, 2, 6, 4, 14, 2
```

The new value of the cross-correlation function is 0.45. This is because the series do not match any more, as one of them "lags" behind by 1 value. If we continue this with a lag of 2, we get a cross-correlation value of 0.15. The name of the game is *time delay analysis*, where we try to find the exact lag by computing the cross-correlation function across a range (and precision) of lags. Then the lag at which the *maximum* cross-correlation is found, is the *offset* between the two time series. Of course it is very time-consuming to do that, so optimizations exist. Luckily for us, the Praat software can readily compute a cross-correlation of two sounds (remember that waveforms are also in essence time series). Here is the cross-correlation computed by Praat for two sounds:

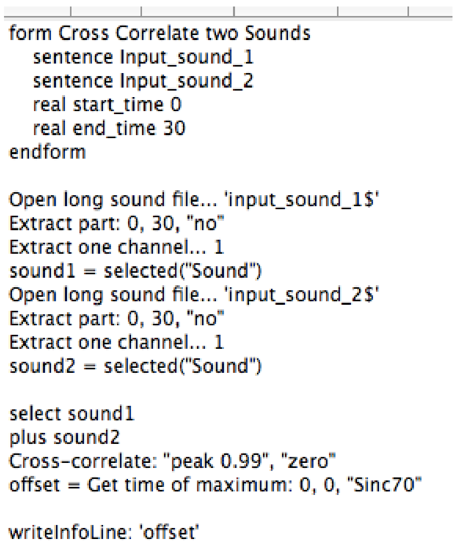


(To see the analyses, zoom in to at most 10 seconds,

or raise the "longest analysis" setting with "Show analyses" in the View menu.)



To the left you see the value of the cross-correlation function for the whole file, while the right image shows a zoom to a window with a duration of 55ms. As you see, the computation of the argmax (the time of maximum cross-correlation) is very precise in Praat. But this argmax is the *offset* between the two audio files, so it is basically all we want!. It is possible to do all this from the Praat GUI, but since I had many files, I used a Praat script instead (seen on the right).



```
form Cross Correlate two Sounds
   sentence Input_sound_1
   sentence Input_sound_2
   real start_time 0
   real end_time 30
endform

Open long sound file... 'input_sound_1$'
Extract part: 0, 30, "no"
Extract one channel... 1
sound1 = selected("Sound")
Open long sound file... 'input_sound_2$'
Extract part: 0, 30, "no"
Extract one channel... 1
sound2 = selected("Sound")

select sound1
plus sound2
Cross-correlate: "peak 0.99", "zero"
offset = Get time of maximum: 0, 0, "Sinc70"

writeInfoLine: 'offset'
```

The script takes four inputs. The first two are two sounds, while the next two define which part of the sounds will be used to compute the cross-correlation. Remember that all this is computationally expensive, so if you have a rough idea of how big the offset is, you can set a number slightly bigger than that. In my case, I expected about 10 seconds, so I put 30 to be on the safe side.

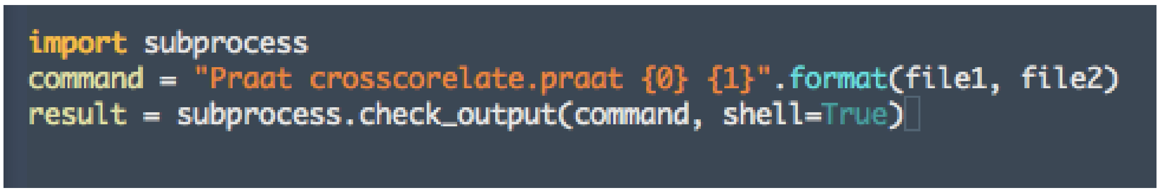
If your offsets are large, consider manually trimming your files approximately (within a minute or so, or as close as you can get without spending too much time) and then use those trimmed files in Praat. The script extracts the left channel of a stereo sound in order to compute the cross-correlation, as computing on stereo sounds takes twice the time (for each channel) with no precision gain in the outcome. The result is returned by the final line of the script (the writeInfoLine command).

Another useful aspect of Praat scripts is that they can be called from the command prompt/terminal/shell (depending on your OS). For example in Mac OS you can type in terminal:

```
Praat crosscorrelate.praat file1.wav file2.wav
```

Praat has to be in your PATH, otherwise you find it where you downloaded/installed it. If you type the above command, the result will be output to your console. This is because writeInfoLine outputs to *stdout* when the script is executed from a console (Windows users: use praatcon.exe instead of praat.exe in order to execute scripts from command prompt)

But, as we saw above, we can call shell commands from within a Python script. This time, because we need to collect the value output to stdout we do not use the *os* module, but instead we use *subprocess*:



```python
import subprocess
command = "Praat crosscorelate.praat {0} {1}".format(file1, file2)
result = subprocess.check_output(command, shell=True)
```

That's it! We have called our Praat script with two files as inputs: the WAV files previously extracted from the MOV files. The default values 0, 30 are used for the arguments I omitted. We now have the offset between the two files stored in the variable called *result.*

STEP 3: Trimming the video files

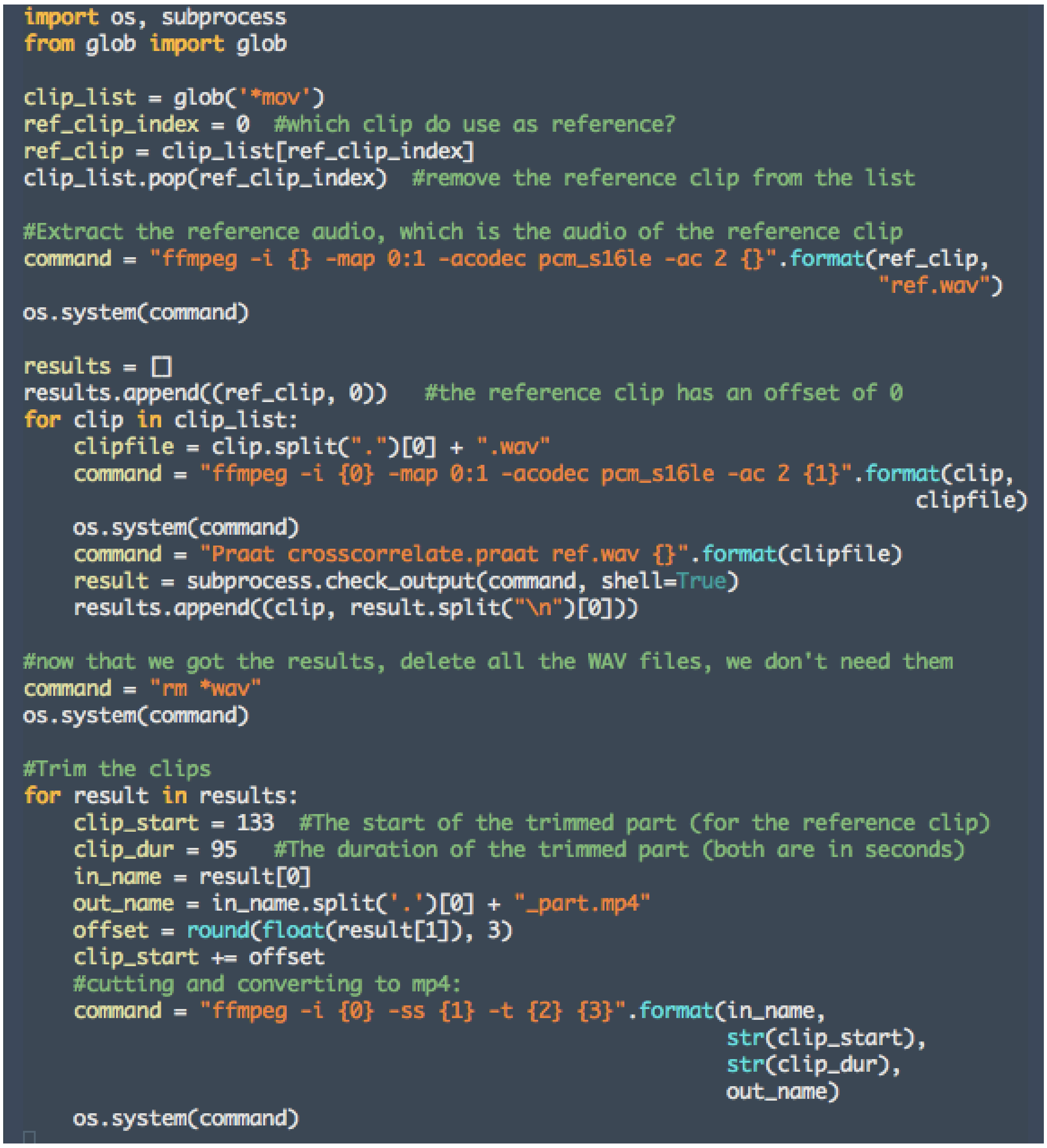
Now that we have the offset, we need to trim our video files accordingly. For this, we turn again to *ffmpeg*. The command I use for the trimming is this:

```
ffmpeg -i input.mov -ss 123.024 -t 95 -c:v copy -c:a copy output.mov
```

The -ss argument shows the time I want to my output file to start at, while the -t option shows its *duration.* Everything else stays the same (audio and video codec are just copied). However, it is possible to convert the video format in the same step, saving some time. I wanted to create MP4 (H.264) files, so I used this:

```
ffmpeg -i input.mov -ss 123.024 -t 95 out.mp4
```

Basically, we are done. We have done all the steps needed to sync our videos. But we need a nice Python script to bring it all together. Here is the one I used:



```python
import os, subprocess
from glob import glob

clip_list = glob('*mov')
ref_clip_index = 0  #which clip do use as reference?
ref_clip = clip_list[ref_clip_index]
clip_list.pop(ref_clip_index)  #remove the reference clip from the list

#Extract the reference audio, which is the audio of the reference clip
command = "ffmpeg -i {} -map 0:1 -acodec pcm_s16le -ac 2 {}".format(ref_clip,
                                                                    "ref.wav")
os.system(command)

results = []
results.append((ref_clip, 0))   #the reference clip has an offset of 0
for clip in clip_list:
    clipfile = clip.split(".")[0] + ".wav"
    command = "ffmpeg -i {0} -map 0:1 -acodec pcm_s16le -ac 2 {1}".format(clip,
                                                                        clipfile)
    os.system(command)
    command = "Praat crosscorrelate.praat ref.wav {}".format(clipfile)
    result = subprocess.check_output(command, shell=True)
    results.append((clip, result.split("\n")[0]))

#now that we got the results, delete all the WAV files, we don't need them
command = "rm *wav"
os.system(command)

#Trim the clips
for result in results:
    clip_start = 133  #The start of the trimmed part (for the reference clip)
    clip_dur = 95   #The duration of the trimmed part (both are in seconds)
    in_name = result[0]
    out_name = in_name.split('.')[0] + "_part.mp4"
    offset = round(float(result[1]), 3)
    clip_start += offset
    #cutting and converting to mp4:
    command = "ffmpeg -i {0} -ss {1} -t {2} {3}".format(in_name,
                                                        str(clip_start),
                                                        str(clip_dur),
                                                        out_name)
    os.system(command)
```

Here is a brief description of what the script does:

- Find all MOV files in current directory, put all filenames in a list
- Select one filename in the list to be the *reference* file and remove it from the list
- Extract the audio of the reference file (ref.wav)
- For each file in file list, cross-correlate its audio with ref.wav to find the offset (uses Praat). Add all results (filename, offset) to a list, which contains one item already: (ref filename, 0)
- Delete all WAV files
- For each item in the *results* list, trim the video at the given start and duration, taking into account the offset. The reference file will be trimmed at exactly the start time (its offset is zero), but each other file will be trimmed from a start that is shifted by the respective offset.
- Optionally convert the video files to the desired format