SAS2340-2018

Using SAS® Cloud Analytic Services with Raspberry Pi Time-Lapse Photography

Matthew Parmelee, SAS Institute Inc.

ABSTRACT

While time-lapse image collection is a trivial problem to solve, it presents us with yet another data stream of IoT-type data that can be easily integrated and built out using the SAS® Cloud Analytic Services (CAS) Image Action Set.

Beginning with the release of the first wave of Raspberry Pi computers in 2012 (even earlier if we include the Arduino), the surge in popularity of low-cost, single-board computers has continued at breakneck pace over the past several years. While initially intended to be used in K-12 computer science education, the Raspberry Pi has been featured prominently as the core component of a massive number of electronics projects.

SAS Pittsburgh, currently located on the 29th floor of PPG Place, provides an excellent vantage point for time-lapse photography of the confluence of the Allegheny and Monongahela Rivers.

In this session, we'll walk through how to set up and integrate these two technologies.

INTRODUCTION

As with most data-focused endeavors, we'll split our problem into two subsets: acquiring the data and handling the data. Acquiring the data will necessitate building a physical rig to collect the data over the specified time frame, and handling the data will be turned over to CAS and other helpful software packages.

Fortunately for the casual hobbyist, this build requires little previous working knowledge. Anyone should be able to follow along and vary this approach to suit a wide variety of needs. You should expect to learn an entry-level amount about small, single-board computers, computational photography, video encoding, and the CAS image functions.

HARDWARE

To approach hardware selection for this project, we must first consider the goals of time-lapse photography. Time-lapse photography entails taking photos of the same scene at regularly timed intervals, typically so that the changes over time can be quickly visualized. As such, our build has two key requirements. First, it must involve a camera mounted in a manner that is as stable as possible. Second, we need to devise a method for automating camera shots so that they occur at a defined interval.

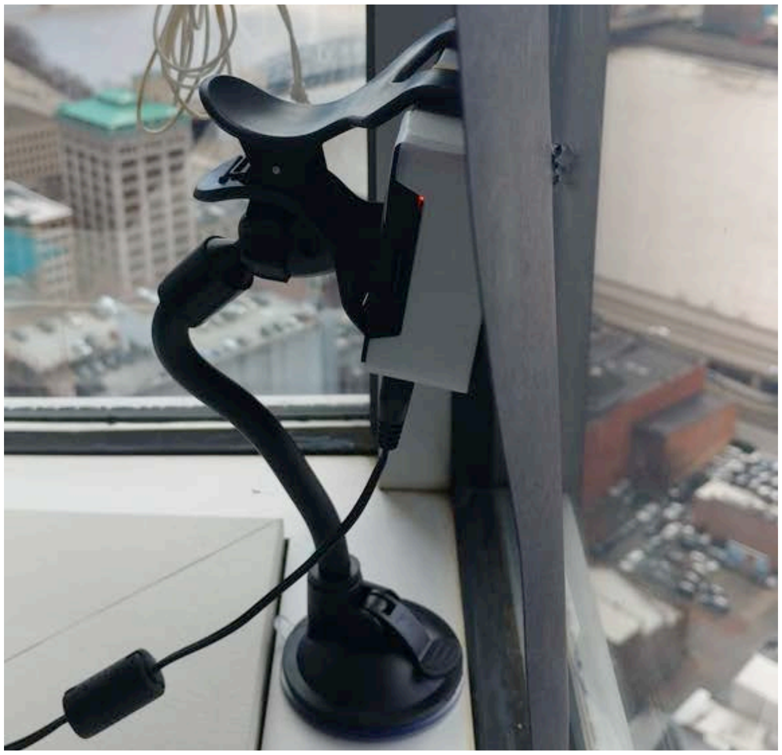

Figure 1. The SAS Pittsburgh Time-lapse Camera Looks Down at the Monongahela River.

RASPBERRY PI 3 MODEL B

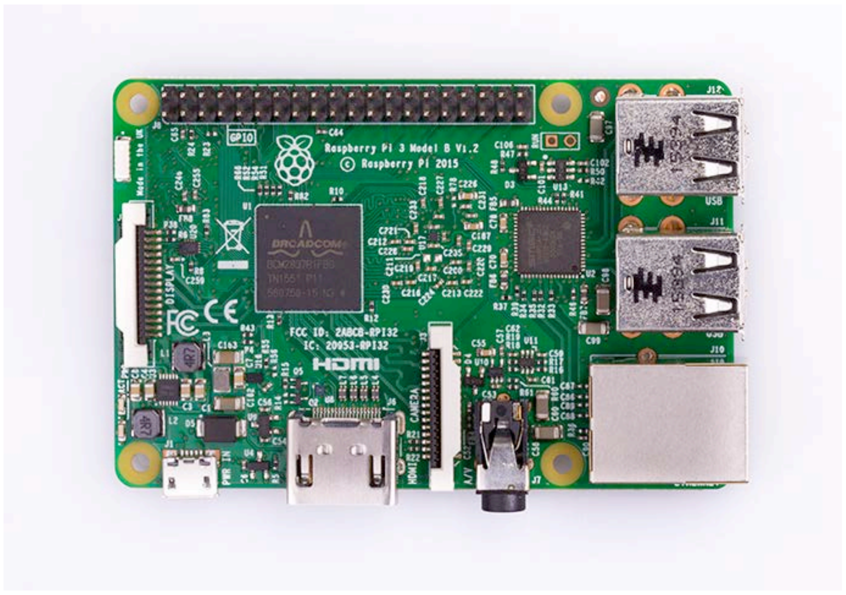

Figure 2. Raspberry Pi 3 Model B - $35

Delete

The Raspberry Pi 3 Model B is the most recent (as of this writing) full-size model. We could get by with most comparable models, but the onboard Wi-Fi is too good a feature to pass up from a maintenance perspective. To power and operate the Pi, you'll also need a micro USB power supply and a MicroSD card of at least 4GB.

RASPBERRY PI CAMERA MODULE

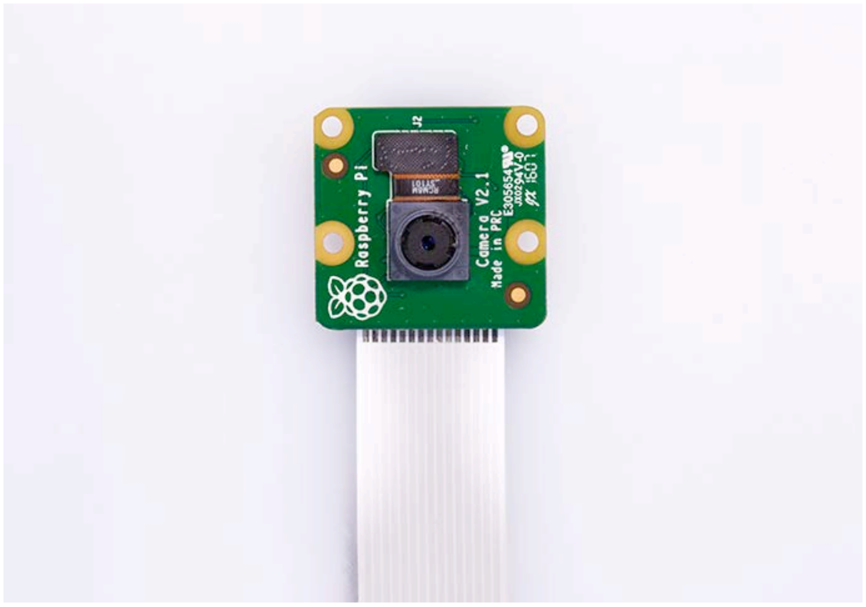

Figure 3. Raspberry Pi Camera Module V2 - $30

Delete

There are many compatible camera modules for the Raspberry Pi (including night vision!), but the “official” module V2 is usually the best bet in terms of compatibility and features.

MOUNTING HARDWARE

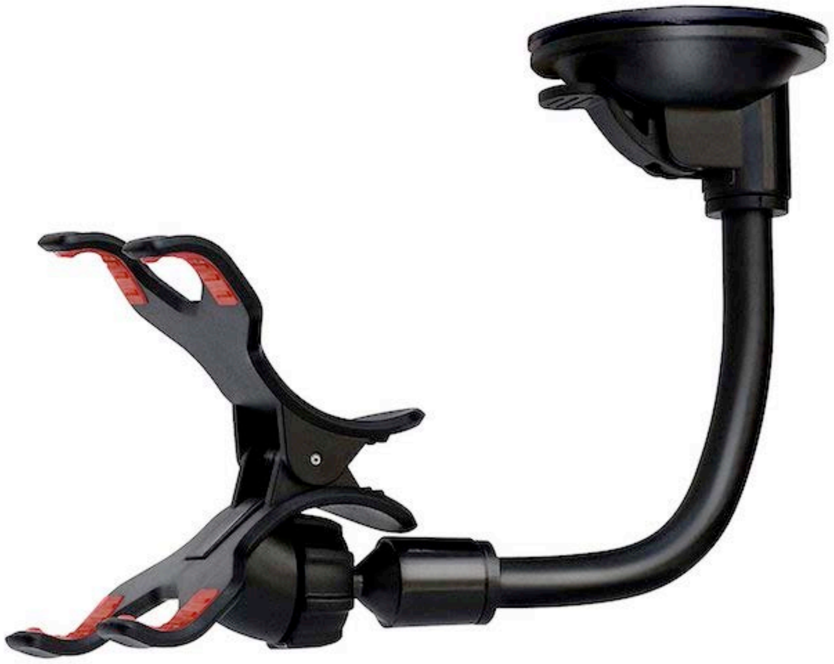

Figure 4. Car Cell Phone Mount - $8

Delete

Mounting the Raspberry Pi and camera module is largely a matter of your operating environment and a host of other varying factors. The only true requirement for time-lapse photography is a mounting solution that is as stable as possible while also being maneuverable enough to adjust to the perfect camera angle.

At SAS Pittsburgh, we're pretty high up on the 29th floor. We needed a solution that could be angled down a bit. We ended up going with this car cell phone mount (http://a.co/3dubqPj).

GLARE REDUCTION

Since most camera setups are going to be landscape views from behind a window vantage point, it's important that we aren't capturing the reflection of the camera itself in every shot. There's an endless ongoing conversation regarding how to best mitigate this issue, but we opted for a large piece of black card stock around the camera, which seems to get the job done.

SOFTWARE

With our camera setup stably mounted, it's time to solve our second problem: automating the camera. This will be done entirely via the Raspberry Pi and its camera module, so we'll need to select a suitable operating system and do a little bit of scripting.

RASPBIAN

Raspbian is the go-to operating system for the vast majority of Raspberry Pi projects. Based on the Debian operating system, it is a widely supported, feature-rich Linux distribution that is versatile enough for most projects.

CAMERA AUTOMATION

The core of time-lapse photography is simple: take a photo from the same exact position at a consistent interval for some period of time. We elected to simplify this as much as possible (since it's running for such a long time) and represent it as two parts: a script that takes a photo and a cronjob that runs the script with regular frequency.

```
# python3
import time
from picamera import PiCamera

camera = PiCamera()
camera.resolution = (1024, 768)
camera.start_preview()
time.sleep(2)
camera.capture('/home/pi/Camera/' + str(int(time.time())) + '.jpg')
```

The camera code is simple. It imports the camera library (PiCamera, included by default in most Raspbian distributions), initializes the camera's warm up routine, and snaps a photo. The script saves the photo with a filename that represents the current UNIX timestamp, and ends.

```
*/5 * * * * python /home/pi/bin/timelapse.py >/dev/null 2>&1
```

The cronjob is even simpler. All we're doing here is executing the above script once every five minutes by modifying the default user's crontab. (For those unfamiliar with this process, simply execute crontab –e.) The rest is handled by the system's cron daemon.

With these two processes running locally on the Pi, we have an autonomous and headless camera rig that we can allow to run for the desired time frame.

CAS

By this point, we should have gathered a sizable number of photos and can begin the process of passing them in to CAS, processing them, and retrieving the results.

GETTING IMAGES INTO CAS

The following CAS action will load the images from the provided directory and distribute them randomly across all available worker nodes.

```
# python3
from swat import *
cashost='sasserver.demo.sas.com'
casport=5570
casauth='~/.authinfo'
s = CAS(cashost, casport, authinfo=casauth, caslib="casuser")
s.loadactionset(actionset="tkimage")
s.image.loadImages(
      path='/path/to/pi/images',
      casout={'name':'imagesTable','replace':True}
)
```

PROCESSING IMAGES IN CAS

> ***NOTE: All of the following was done on the SAS® Viya® 3.1 (SAS® Visual Data Mining and Machine Learning with SAS® Studio and Python gold image.***

There are any number of transformations that can be performed on image data after that data is properly represented in CAS. But for the sake of simplicity in this example, we'll simply be converting the images to gray scale before returning them.

```
# python3
s.image.processImages(
      imageTable={"name":"imagesTable"},
      casout={'name':'processedImages','replace':True},
      imageFunctions=[{'options':{'functiontype':'CONVERT_COLOR'}}]
)
```

RETRIEVING PROCESSED IMAGES

Similar to the image loading step, retrieving images from the CAS server is a single simple action:

```
# python3
s.image.saveImages(
      caslib='casuser',
      subDirectory='SavedImages',
      images=vl(table='processedImages', path='_path_'),
      type='jpg'
)
```

TIME-LAPSE ANIMATION

After waiting days/weeks/months to collect our photos and finally running them through the filtering of our choice in the cloud, it's finally time to compile the fruits of our labor into a sleek video format.

FFMPEG

FFmpeg is an excellent tool for handling multimedia data and, if you've made it this far, you're going to have a whole lot of it (our example racks up 288 photos per day!).

After you have installed FFmpeg on your system, you can simply navigate to a folder that contains all your images and execute the following at the command line:

```
ffmpeg -r 24 -i %*.jpg -s 1024x768 out.mp4
```

This will iterate through all your photos (in the correct order, because we used UNIX timestamps as filenames) and create an MPEG4 video at a resolution of 1024x768 at 24 frames per second.

CONCLUSION

As we've learned, these two technologies are both easily integrated and easily decoupled. There's a lot of value in learning about these two platforms independently of each other, as well as the creativity to envision projects that could combine them.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Matthew Parmelee
SAS Institute Inc.
+1 (412-995-5115)
Matthew.Parmelee@sas.com