TimelapsePy

Version 1, July 28, 2023

Introduction

TimelapsePy is set of scripts written in python, used to control a raspberry pi camera to capture timelapse photos. Code is separated into smaller files based on the logic of the program flow. More importantly, this allows all the user-settings in a single file (*usersettings.py*), which simplifies the user experience. It is intended to be as automated as possible and ~~foolproof~~ fool-resistant for less experienced users.

Program Flow

Initialization

TimelapsePy can be ran manually or as a scheduled task, e.g., using *cron*. Once started the program loads several dependencies (other code libraries necessary to run), loads in the user specified settings, and gets the camera ready to take photos.

Event Loop

Next, TimelapsePy enters into an “event loop” that by default runs until power is lost, space is filled up, or interrupted by the user. In other words, the main function of the program occurs in an infinite while-loop.

By default, the program will continuously capture images at a specified interval, but the user can specify a daily start and stop time. The capture interval is a single loop interval. At the start of each loop, the program determines if it is in continuous mode, or if it is user-specified time mode. If in user-specified time mode, it checks the computer’s clock time and compares with the specified start and end times.

If the program is not in continuous mode, and it is outside the user-specified times, the program “sleeps” for the loop interval. If the program is in continuous mode, or the clock time reads within the user-specified times, the program enters the capture code block.

Capture Code Block

Once in the capture code block the program performs several tasks before capturing a photo. It takes a time measurement and creates a timestamp, which is used for naming the photo file. It also searches to see if there is a user-specified external storage device. If it finds the storage device, it determines if a user-specified directory (folder) exists on that drive. If it does not, the program creates the directory, captures the image, and saves it to the said directory.

If the user-specified external storage device is not found, the program looks for a fallback location: a user-specified folder in the user's Pictures directory. If the directory is not found, it creates it, captures the picture, and saves it to that location.

The program then performs some housekeeping functions, then takes a nap for the rest of the loop, then starts the event loop all over again. This all occurs within the user specified capture interval (more or less, the actual time might be slightly longer).

Program Setup

Download and Install Scripts

TimelapsePy is a set of non-complied python scripts that can be downloaded from the GitHub repository: www.github.com/benstanfish/TimelapsePy. Click the *code* button and select the *download zip* option.

The script titled *timelapse.py* is the main script that orchestrates the whole program, while the script titled *usersettings.py* is where the user can change default parameters.

The collection of scripts can be located anywhere, generally it's recommended to be located in a folder called TimelapsePy that you can create in the user directory, e.g., `/home/username/TimelapsePy` on a raspberry pi. Make sure to use your username in the path above.

Schedule with Cron

The preferred implementation is to automate the program by scheduling TimelapsePy to run each time the raspberry pi is booted up. There are several methods, but *cron* is generally the easiest and problem resistant.

In a terminal enter the command `crontab -e` to edit the crontab table. If it's your first time, it will ask you which text editor to use, you can select *nano* or your editor of choice. Once in the crontab file, scroll down to the bottom of the file and type in the following line: `@reboot python /home/username/TimelapsePy/timelapse.py &` then press Ctrl + X and type Y to save and close.

This command tells the raspberry pi that every time it boots up to start python and runs the script specified in the specified path; finally, the ampersand symbol at the very end is extremely important. It tells the system not to wait for the python script to finish running before it finishes booting up. If you forget the ampersand, the pi will never finish booting up because the python script has an infinite loop.

The next time you reboot, and any boot thereafter, the script will run. You can verify that the script has been scheduled to run by checking the crontab: in a terminal type `crontab -l` to list the scheduled jobs. You can also remove the scheduled task by typing `crontab -r`.

Verifying the Script is running

Note that *cron* is a task scheduling daemon, this means it's a background process that runs tasks without showing them to the user. The easiest way to see if the program is running is to navigate to the image capture directory and see if images are being written at the specified interval. There are shortcomings to this method, however, especially if you are outside the user-specified capture time frame.

The most technical way to see if the script is running is to use the terminal command `ps aux` which lists all the currently running processes. Among other things, command lists the PID (process ID), the process name and the command. In the case of TimelapsePy, the process name will be *python* and the command will be the path to timelapse.py that you specified in crontab, e.g., */home/username/TimelapsePy/timelapse.py*.

Interrupt (Kill) the Process

When TimelapsePy starts up, it takes control of the attached camera. While TimelapsePy is running, no other processes will be able to access the camera. In case you want to access the camera or stop TimelapsePy, in this case, you will need to terminate the process.

There are two ways to do this. First is force quitting the python process through the *task manager*, which can be found in the raspberry pi menu. The other method is to take note of the **PID** listed for the process in `ps aux` and then typing the following command in the terminal: `kill -9 processID` but make sure to replace *processID* with the actual PID number.

If you want to stop TimelapsePy from running at boot, you can access the crontab file again by typing crontab -e into a terminal then scrolling all the way down to the `@reboot python…` line and put a hashmark `#` at the very beginning of that line. This comments out that line of code that it does not run on startup. But by commenting the line out, it allows you to easily restore the functionality at a later date.

User Settings

The user settings file is where the general user should edit settings. If you are not sure what to put in, you can leave the defaults. If you make a mess of it, feel free to download a fresh copy of *usersettings.py* from www.github.com/benstanfish/TimelapsePy. Below is a summary of the default settings.

Summary of Default Settings

Basic settings for capturing images:

- `interval_in_seconds_between_capture = 10`
- `image_file_format = 'jpg'`
- `capture_continuously = True`
- `capture_start_hour = 4`
- `capture_end_hour = 18`

Camera configuration settings:

- `resolution = 'max'`
- `image_format = 'default'`
- `flip_horizontal = False`
- `flip_vertical = False`
- `override_default_focus_settings = False`
- `user_focus_mode = 'manual'`
- `user_focal_distance = 0`

Render to mp4 Video:

- `create_mp4 = False`
- `frame_rate = 24`

Advanced Editing

For those who are more versed with *Picamera2* or even *libcamera* libraries, feel free to edit the other modules, such as *timelapse.py*, etc.

Writing a Video File

As discussed above, TimelapsePy will attempt to write images to the named external storage device, if found. Failing that, it will write images to the specified fallback directory. If you set the `create_mp4 = True` (in *usersettings.py*)

then when the image capture loop ends, TimelapsePy will use ffmpeg to take all the images in the specified image folder to an mp4 file. The output file is written to the `/home/username/Video` folder.

Note that if `capture_continuously = True` then even if `create_mp4 = True`, the video file will never be created because TimelapsePy will never leave the event loop. If this or another condition prevents TimelapsePy from creating a video, you can always go back later and run *tvideo.py* to create the video.

Dependencies

In addition to the native Python libraries, make sure that you have the following libraries installed and up to date:

- libcamera
- Picamera2
- Numpy

You can always update everything by running the following terminal command: `sudo apt update && sudo apt upgrade`