Sky Imager Aggregator Documentation for Programmer

In order to make the sky image aggregator fully functional we are using 3 python3 scripts. Two executable scripts: split.py and SendStorage.py and the library LibrforPi.py written for the needs of the program. Supervisor service controls the scripts execution.

LibrforPi.py

This python3 script represents the library that contains all of the functions we need. It starts by importing some of the other libraries needed which are: numpy, OpenCV, datetime, copy and http. You will need all of these libraries so you can use this code.

In the library you will find 10 unctions.

The first two functions are used to name the image and masking it.
The nameimage() function uses the time when the image is taken as a name. The second function maskImg() does pixel-wise multiplying of a previously generated black and white mask with the taken picture.

The next 3 functions calculate the sunrise and sunset time in order to pause the system over night and allow for storage to be sent. The first function finds today's date in a previously generate sun-cycle timetable and takes out the sunrise time and the sunset time. This information is sent to the calculate_pause_time() function. This function calculate in seconds how much time the system is going to be pause while the storage is sent. And after that this information is given to the pause_time() so it can compare current time to the sunset time and pause the system if the two match.

After this we have 3 functions dedicated to managing the storage. Firstly you have the stack_storage() function which checks the amount of images saved and deletes the excess. This is written in order for the storage not to exceed the amount of memory we have left on the Raspberry Pi. The function makes the storage act like a stack-like. It deletes the older images first making the storage a FIFO structure. Second we have the check_storage_content() which is written for a different purpose. In the SendStorage.py we need to know if if there is something in the storage before we try to sent its' content. And the last one is time_for_storage() which basically checks if it is sunset time in order to begin sending the content.

And the last two are delete_image() function which is used in several different points in the code and check_connectivity() which tries calling the google server to check if there is internet. Odds are that the google server is not going to be down and if there is an internet connection the server will respond.

split.py

The split.py program does the following:

1. Access the camera using the cameras

This is done by accessing the camera via url. The code is written for both cameras and you can find instruction on switching between the two in the code comments. Since the RaspberryPi is installed in a remote location and it probably will not have a display attached to it, camera stream displaying is not enabled. (If this option is needed you have a instruction in the code comments).
There is also a safety feature in case the camera is not connected properly.

```
if cap.isOpened():
```

2. Taking a picture and saving the picture

In order for a picture to bi sent to the server it must first be written on the disk. As the return from the function cv2.videoCapiture() is a stream and not a single image. We take an instance from the stream and save it on to the disk.
To make the capturing of the image more precise time-wise the code is written in a way that will only take pictures if the value of the second divided by 10 is 0. Before saving it however, we apply a mask on the image to black out the neighboring buildings and/or objects. This is done by pixel-wise multiplying of the image with a previously created mask in matlab.

3. Communication lines

This is done as follows:

```
if lfp.check_connectivity() == True:
        try:
```

```
subprocess.call(['python', 'pic.py', 'server'])
```

The first check is for internet connection. If there is an internet connection the if case passes. After that it finally it tries to pipe the pic.py server script.
If this code passes a single image has been sent to the server. After sending it the image is then deleted in order to preserve as much memory as we can.
Note: The change I have made is that the code does not pull a file named 'skyimage.jpg', but rather the newest ' .jpg' file in the folder, as all images will be named differently.

If for some reason or another the communication between scripts fails the code then proceeds to storing the image and prints out an error message.

```
except:
    sys.stdout.write('ERROR: calling server script fail!\n')
    sys.stdout.write('Storing image ...\n')
```

If ether of these cases fails the image is stored in the appropriate STORAGE folder for later sending.

SendStorage.py

SendStorage.py is a script created to send the images that for some reason were not sent on time.

The script goes through 3 checks before attempting to pipe the server script. The first check is if it is sunset time then it checks if there is anything to send and finally it checks for an internet connection.

To sum it up

The main reason we have two executable scripts is because python doesn't actually allow for multiprocessing. It doesn't use different cores for different processes instead swaps between the two processes. So to solve this issue we use two scripts that work in different times.

Both scripts are controlled by supervisor service. The configuration file is set up so the scripts start at power up. In the case of a crash the program would auto-

[1]pic.py is a python2 script written by Marek Maska with its' own explanation. I will however explain the changes I have made.

restart. Both processes run in the background and use 40% of the Raspberry Pi processor at worst.