Buildots Pipeline Exercise

An equirectangular image is a specific projection method that flattens 360 images, like the world map.

The goal in this exercise is to create an end-to-end pipeline that converts equirectangular images, into normally projected images (known as Gnomonic projection).

The pipeline has 4 stages: file downloader, projector, edge detection, image analyzer. Each service runs as a separate process (python <my_service>.py).

Each service is configured with an input and an output directory. When a new file appears in the service's input directory, it is processed and deleted.

Don't use the "watchdog" library for "listening" to the input folders. We are aiming for a solution which polls the input folder every X seconds.

Service #1: Images Downloader

For each JSON file put in input directory, the service shall download every image on the list into the output directory (using random UUID as filenames)

Input files should have the following structure:

```
{
        image_list: ["https://i.imgur.com/9wpKT4r.jpg", "https://i.imgur.com/RfQ3OoS.jpg"]
}
```

Service #2: Projector

For each equirectangular image in the input directory, the service shall perform a Gnomonic projection and save the image in the out directory.

We suggest using the following open-source library to perform the projection: https://github.com/NitishMutha/equirectangular-toolbox. Note that the author forgot debug code at rows 91-93 which can be commented out.

Service #3: Edge Detection

For each equirectangular image we want to output an image of it's edge detection. Use the opencv easy implementation to achieve this. You can see an example here.

We will have different hard-coded thresholds according to the format of the image.

- For ".png" images it will be (100, 200)
- For the rest it will be (100, 150)

Service #4: Analyzer

For each image, we are interested in creating a json file that holds metadata about the image. The service should support easy integration of additional analyzers.

List of analyzers that you should implement as base:

1. Image dimensions
2. Average color (average value for each channel: r,g,b)
3. Average B/W color (to convert to BlackWhite using "cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)"
4. Bonus: location of the darkest 20x20 area in the image (by average value)

When you have finished, please zip all of the .py and other relevant files to a folder with your name and send it to your Buildots interviewer.

Notes

1. Fewer features > lousy code. Don’t neglect things like meaningful names, good design, informative prints and others. Good design under some time constraint is the main thing we’re checking for in this exercise.
2. Make sure you handle the case when a service fails to process a request - don’t try and process an input infinite number of times.
3. Most Python libraries dealing with dir monitoring have issues.. Using polling over the library is good enough for our use case
4. Whenever you encounter thoughts like - “if I’d build this for production, I would do XXX” (testing, use persistent storage, extract some logic to a util, anything) - don’t implement it, but rather add a comment in place, or put general comments in some README files so I’ll know you thought about these issues and constraints
5. Assume everything can be done in memory for now, but, as listed above - comment on what you’d do differently when needed
6. Note that the input of services #2 and #3 is the same, so for service #1 you will need to have multiple output folders.
7. The input for service #4 is the output of service #2
8. Exercises are not perfect, ask your interviewer if you think there’s an issue you shouldn’t have encountered.
9. Please enjoy 😊