Dataset: The dataset is a rearrangement from
https://data.mendeley.com/datasets/twxp8xccsw/8.

It contains 400 512x512 images each for both **Infrared (IR)** and Visible **Light (VL)**: 200 contains power line (file name started with TV) and 200 don't (file name start with TY). Also, it contains "ground-truth" images where they only highlight the power lines (if a file does not contain power line it will be total blank).

Please advised, the ground truths are NOT always true because some figures they are misaligned! You can check it manually or you can check the result of EDLine: If it get very low sensitivity for EDLine (<0.05), it probably encounter a misaligned ground truth. The existence of such data will lower the overall result. Please see the algorithm part.

You can filter the result in the future.

Files and Algorithms: There are two files are executable: ImageLookout.py and TestLinesDetection.py. The details of all files are listed below:

ImageLookup.py

When execute this file, it will create line detection results for a designated file (specified by image number). The line detection result contains 5 files: lines_GroundTruth.bmp, lines_Origin.bmp, lines_RHT.bmp, lines_EDLine.bmp, lines_PLineD.bmp. You can cross check the result.

TestLinesDetection.py

The main body of program. It will output the sensitivity and specificity of line detection as well as the average time for processing the image. The sensitivity and specificity is defined by the overlap with ground truth.

Sensitivity, also called True Positive Rate (TPR). Is defined by fraction of overlap between line detected and ground truth. Assume TP (true positive) is the total pixels of overlap, FN (false negative) is the pixels of power line which are not detected by the algorithm. TPR = TP/(TP+FN). The higher TPR means the algorithm is more sensitive to the power line detection. Namely it can detect more pixels in power lines. If the ground truth does not contain powerline TPR = None

Specificity, also called Complement False Positive Rate (1-FPR). Is defined by the fraction of pixels detected by algorithm which is belongs to the power line. Assume TP is the total pixel of overlap, FP (false positive)is the pixels detected by algorithm but not actual belong to power line. 1-FPR = 1-FP/(TP+FP) = TP/(TP+FP). Higher specificity means the algorithm is more specified to the power line and it can successfully reduce noises. If TP+FP = 0 it means no lines detected by algorithm. Then 1-FPR = 1.0

The algorithm is organized in two parts:
1. Use EdgeDrawing to extract the edges from images (same as PLineD paper)
2. Detect the power lines using three algorithms and compare the overall time, sensitivity and specificity.

I stored TPR and FPR (not 1-FPR) result for each image in the list variables called TPR_list_XXX and FPR_list_XXX (XXX is algorithm name). You can use that to plot the distribution of TPR or FPR. Or using this to detect the misaligned image. You can also modify the code to storage the output detection image result. Please see the introduction of TestTool.py

Below is the output of the algorithm (with 64GB RAM and 1.3GHz CPU, Linux Mint 18). The actual result should be higher if we remove the misaligned ground truth.

```
Preprocessing Image (EdgeDrawing)
100 images processed
200 images processed
300 images processed
400 images processed
Preprocess Complete
Time: 10.7s. Average Time: 0.027s
---------Randomlized Hough------------
Sensitivity: 0.237
Specificity: 0.584
Average time(including preprocessing): 0.0310s
----------EDLine detection-------------
Sensitivity: 0.759
Specificity: 0.578
Average time(including preprocessing): 0.0349s
--------------PLineD---------------
Sensitivity: 0.608
Specificity: 0.714
Average time(including preprocessing): 0.0355s
```

The total time of processing image consists of two parts: **EdgeDrawing time**(0.027s) **+ Line detection time** (<0.010s). The parameters have already well-tuned to get the balance of performance and time complexity. You can change the parameters to see the differences.

The result is interesting, RHT is the fastest algorithm but mainly because its code is already optimized by compiling the iteration to C codes. However, RHT doesn't catch the power line quite well (if you check the ImageLookup result, RHT only generate small line segments under edge detected by EdgeDrawing). EDLine capture the power line very well but it is kind of noisy. PLineD make a perfect balance on Sensitivity and Specificity.

Other files are:

EdgeDrawing.py Functions for EdgeDrawing. Almost the same as last project but I remove the EDLine Part. I also add an optimized function MergeEdges_() to merge the fragment edges into several large connected segments. However, I don't use it because it will consume a lot of time. You can use that for small images.

LineDetector.py Functions for line detectors, mainly EDLine and PlineD. Please see the comments in the file to match the code with paper.

TestTool.py Tools for evaluate the test result. In order to make the main file clean, I strongly suggest developing functions for testing in this file. The tools now contain functions for calculating TPR and FPR. And a function called confusion matrix. It will return the TP, FP, FN and the result of line detection (the line detection result image is smoothed to reflect a confidence interval of detection)

Others:

If you heard of Receiver operating characteristic (ROC) curve:

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

you can also plot the curve based on TPR and FPR by changing the parameters of algorithms (however to choose the parameter to change is also a challenge). The ROC curve will explain how the algorithm behave under different parameters. Check it if you have time.