License Plate Recognition + Interpretation

By Sarim Qureshi (squres25), Daniel Levert (dlever2), Zak Jamal (zmjamal2), Mohammed Akif (makif3), Tosan Egbesemirone (oegbes2)

Abstract

This project is product-oriented license plate recognition and interpretation program that uses uses concepts such as edge-detection, gray-scaling, cropping, and automation testing (not taught in this class) to produce a solution.

1. Introduction

This project is designed to be a license plate recognition and interpretation program. The program takes an input image and detects a license plate from the image (using cropping, filtering, and edge detection techniques with the OpenCV library). The plate is then cropped and sent to a function that is responsible for recognizing the text written on the plate (using text recognition techniques with the Tesseract OCR library). Then the text is passed to an automation script (with the Selenium WebDriver library), which navigates to a car recognition database FAXVIN, whereupon the script will search the license plate and retrieve the car's information. The program will then output the license plate, VIN, Year, Make, and Model of the queried car onto the terminal.

1.1. Process

The program `img_processing.py` defines two functions that were utilized in this project to extract the letters and numbers from license plates then pass them back to *Tesseract OCR* for interpretation. These functions are `plate_recognition` and `plate_to_text`. `plate_recognition` takes the path of the desired image then converts the image to grayscale

using a bilateral filter and the canny edge detection function from *OpenCV*. After the image is modified, we find the contours of the image. Once this is completed, `plate_recognition` then runs an algorithm that sorts through all the contours in order to find which one looks the most similar to a rounded rectangle. Once the best contour of the original image is found, a new modified image which is cropped to the license plate is returned. `plate_to_text` is then called which processes our new modified image. This function interacts with *Tesseract OCR* to call `image_to_string` which analyzes our modified image and then converts it to text. Since Illinois license plates are 7-8 characters long there is a hard check to make sure that the length of text does not exceed this limit. `automation_script.py` contains the function `return_info`. This function connects to the *Selenium WebDriver* to actually do a license look up on the text that was extracted in `plate_to_text` in FAXVIN. If successful in running, it returns the plate, the VIN, the year, the make, and the model. This function also exits the program if no information can be found. The `main.py` file calls these functions in the order of `plate_recognition`, `plate_to_text`, `return_info`, and then outputs the vehicle information onto the console.

1.2. Language + Requirements

This program was designed using Python 3.6 and it's libraries:

- OpenCV
- Tesseract OCR
- Selenium WebDriver
- unittest
- sys
- os
- time

The program also has a specific browser specification:

- Google Chrome, Version 97 or earlier

2. Installation and Configuration

2.1. For Mac

1. Download tesseract *(with Homebrew)*

```
> brew install tesseract
```

2. Download pytesseract

```
> pip install pytesseract
```

OR

```
> pip3 install pytesseract
```

3. Configure Tesseract path

Comment out line 15 of `img_processing.py`.

```
15   # pytesseract.pytesseract.tesseract_cmd = r"[PATH]"
```

Since Mac does not require a .exe installation for tesseract, we do not need to give a path variable.

4. Download OpenCV

```
> pip install opencv-python
```

OR

```
> pip3 install opencv-python
```

5. Download Chrome WebDriver

Download the driver version corresponding to the version of Chrome you have as well as your operating system from https://chromedriver.chromium.org/downloads.

6. Configure WebDriver path

Extract the `chromedriver` file into `/usr/local/bin/`. Then comment out line 23 of `automation_script.py` and uncomment line 25

```
23   # PATH = executable_path='chromedriver.exe'
```

```
25   PATH = executable_path='/usr/local/bin/chromedriver'
```

2.2. For Windows

1. Download tesseract

```
> pip install tesseract
```

OR

```
> pip3 install tesseract
```

2. Download pytesseract

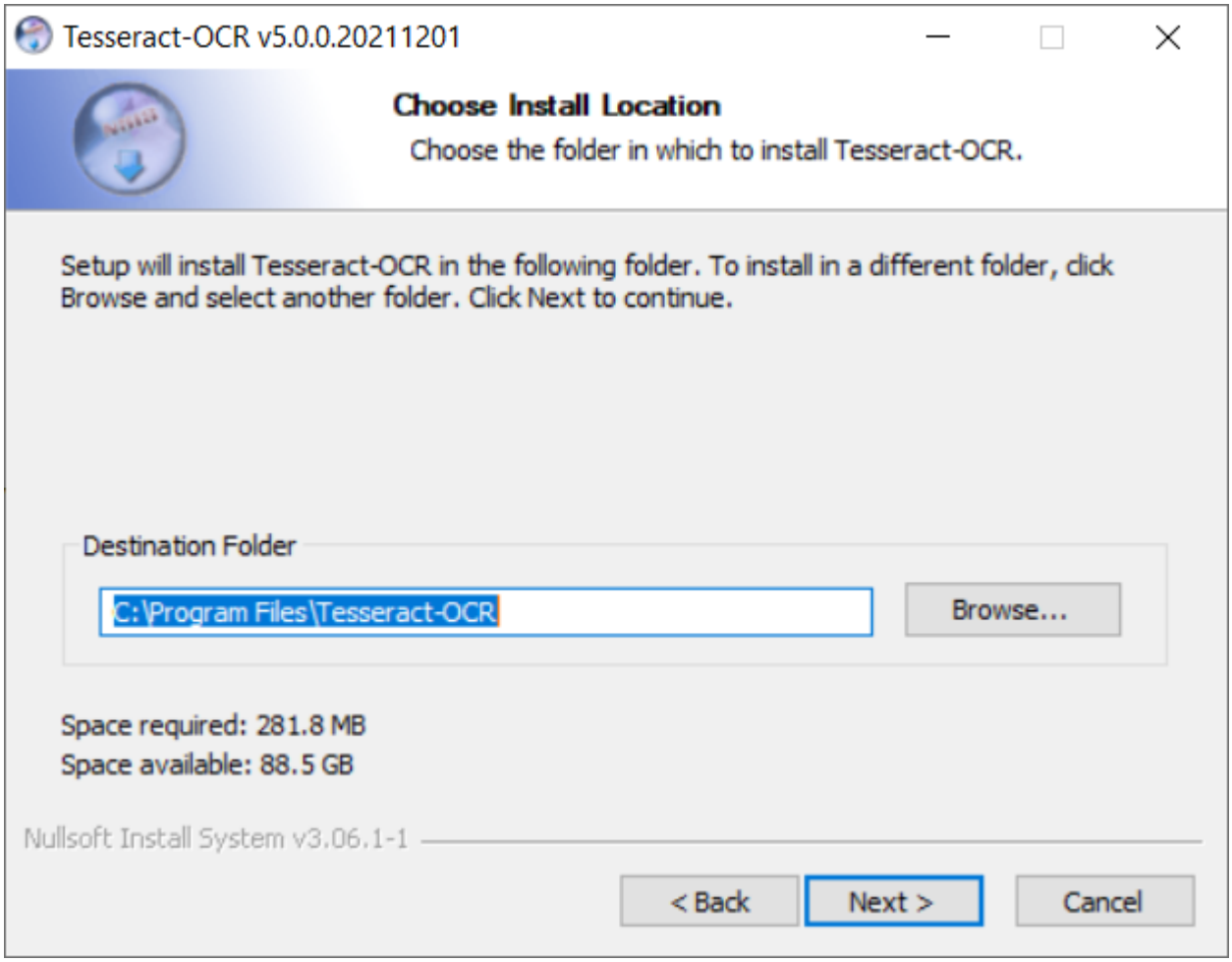
```
> pip install pytesseract
```

OR

```
> pip3 install pytesseract
```

3. Configure Tesseract path

Navigate to this installer page, download the setup wizard pertaining through your system, and run the executable. Copy the path shown during this step of the setup:



This is where your `tesseract.exe` file will be located. Upon installation, navigate to the copied path, ensure that `tesseract.exe` is there, copy it's path, and place it onto the string placeholder of line 17 in `img_processing.py`

```
17   pytesseract.pytesseract.tesseract_cmd = r"[PATH]"
```

4. Download OpenCV

```
> pip install opencv-python
```

OR

```
> pip3 install opencv-python
```

5. Download Chrome WebDriver

Download the driver version corresponding to the version of Chrome you have as well as your operating system from https://chromedriver.chromium.org/downloads.

6. Configure WebDriver path

Extract the `chromedriver.exe` file into your project directory. Then uncomment line 23 of `automation_script.py` and comment out line 25

```
23   PATH = executable_path='chromedriver.exe'
```

Note: You may need to give the absolute path for `chromedriver.exe` *in which case, edit line 24*

```
25   # PATH = executable_path='/usr/local/bin/chromedriver'
```

3. Constraints

- Input file types are limited to .jpeg and .jpg extensions
- License plate must be visible and of a rectangular or almost rectangular shape in the image
- License plate must be from Illinois
- The text of the license plate has to be of length greater than or equal to 7
- Input image must not be too reflective or too bright

4. Fail Cases

`fail1.jpg`

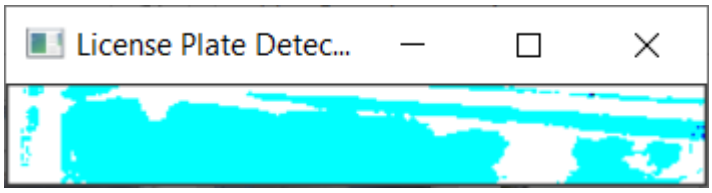
When these images are being processed, the grill is captured instead of the plate. This is likely because the program interprets the grill as an almost rectangular shape.

`fail2.jpg`

When this image is being processed, some random rectangular-like shaped is captured in the corner. This is likely because of the high brightness the window of the car generates, which confuses the program.

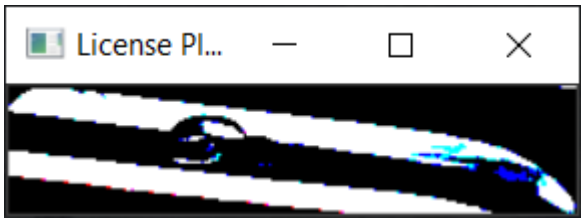
`fail3.jpg`

When this image is being processed, the program captures the ledge located above the license plate. This is likely because the shading difference between the right and left side of the original image confuses the program into thinking the location of the plate actually consists of two different entities.



`fail4.jpg`

When this image is being processed, the program captures the grill of a car that is reflected off the car intended to be captured. Due to the nature of some car paint being reflective, the program is unable to differentiate between multiple solid entities.

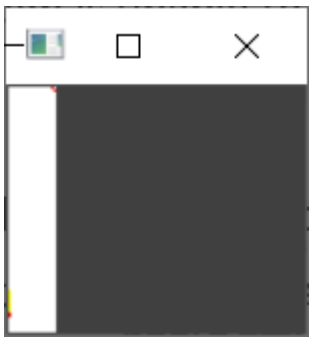


`fail5.jpg`

When this image is being processed, the program captures a portion of the license plate that it deems to be rectangular-like. This is likely because of the angle at which the picture shows the license plate to be.

`fail6.jpg`

When this image is being processed, the program captures practically nothing. This is likely due to the overall high brightness of the entire image itself.



4.1. Discussion

In terms of processing an image and extracting its plate, this program is not without flaw. The conditions of the image (brightness, reflectivity, angle, background noise) are all major factors that determine the optimal output. The best conditions have been noted to be when a picture is taken during the middle of the day (due to the neutral brightness of the sun), from 5+ feet away, zoomed in to reduce the maximum amount of background noise, and at a perfectly straight angle.

4.2. Other Fail Cases

Text Recognition

Although the program may process and extract the plate, the reliability of recognizing the text on the plate is not very high. Tesseract OCR requires very specific image parameters (as was mentioned in **4.1** to be able to succeed.

Crop Configuration

To alleviate the text recognition issue, an attempt was made to try and crop the image as close as possible for better recognition. As a result, lines 57-59 of `img_processing.py` consists of three possible configurations, each of them guaranteeing different results for different images. Only one of them can be uncommented at a time, but the most successful results have been seen when line 58 is uncommented.

```
57   # plate = img[y:y+h, x:x+w]
58   plate = img[y+30:y+h-20, x:x+w]
59   # plate = img[y+40:y+h-30, x+20:x+w-20]
```

However, occasionally the cropping statements on lines 58 and 59 generate an error regarding the window size. When that happens, the best policy would be to ensure both lines 58 and 59 are commented, and then to uncomment line 57.

```
57   plate = img[y:y+h, x:x+w]
58   # plate = img[y+30:y+h-20, x:x+w]
59   # plate = img[y+40:y+h-30, x+20:x+w-20]
```

You may also uncomment line 72 if you would like the program to show you the cropped license plate

```
72   cv2.imshow("License Plate", outImg)
```

5. Test Cases

A subdirectory titled `test_pics` has been provided for the user's convenience. The directory contains sample images that have been tested for correctness or partial correctness in `test.py`. The images that partially recognize the text within the license plate are tested for the best possible sequence alignment. The images that fully and correctly recognize the text within a plate are tested for equality.

6. Run *(from Command Line)*

Ensure that you are within the CS415_Project directory. Then invoke:

```
> python main.py
```

OR

```
> python3 main.py
```

You will then be prompted to enter the image path. In the example below, one of the test images is being given as input.

```
Enter image path (with extension): test_pics/pass1.jpg
```

6.1. Run Test Cases

Ensure that you are within the CS415_Project directory. Then invoke:

```
> python -m unittest test.py
```

OR

```
> python3 -m unittest test.py
```

7. References

Praveen. (2020, August 5). License plate recognition using opencv python. Medium. Retrieved December 9, 2021, from https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c.

OpenCV: Automatic License/Number Plate Recognition (ANPR) with python. PyImageSearch. (2021, November 5). Retrieved December 9, 2021, from https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/.

Kevadiyasmeet@kevadiyasmeet. (2020, July 17). License Plate recognition with opencv and TESSERACT OCR. GeeksforGeeks. Retrieved December 9, 2021, from https://www.geeksforgeeks.org/license-plate-recognition-with-opencv-and-tesseract-ocr/.