MTR Transit Link

Introduction

I developed the "MTR Transit Link" project with the aim of optimizing travel time for MTR passengers. Have you ever found yourself arriving at a station only to realize that the exit you need is at the other end of the train? Inspired by my own desire to save time in such situations, I created this project to allocate train compartments (Front, Back, or Middle) not only to the closest exits but also to other train lines at each station.

While it is possible to manually search for MTR station layouts on Google, visit the MTR website, and download the layout image to determine the closest compartment, my goal was to simplify this process by providing an app where users can simply select the station and identify the compartment closest to their desired exit or exchange line.

Method

To accomplish this, I utilized web crawling skills using Python's BeautifulSoup library to automate the downloading of station layouts, replacing the need for manual downloads.

Since all the stations were in pdf, I converted them into png format, since Google vision is mainly designed for images and png contain more information than jpeg.

Once the layouts were obtained, I used a two-dimensional approximation of the 3D images to identify the closest exits. By assigning coordinates to each exit and systematically searching for them by using OpenCV library, I reduced the time complexity by ignoring non-existent exits. For example, if exit E is not found, I assume there are no exits F, G, H, and so on.

Additionally, a larger resolution for the exit signs and a relatively lower resolution for the actual station file resulted in a more accurate match for the exit. However, time significantly increased for searching so a balance had to be achieved.

To determine the train's position, I employed the Google Optical Vision API. By analysing the coordinates from the text displaying the line name and terminal station, I could identify the train's location.

The direction of travel could be found by looking at divided halves of the arrow subspaces. For example, if the arrow subspace contained more red pixels in the left half, then the arrow was pointing to the left.

Additionally, I analysed the subspaces surrounding the text to determine the direction of train. By examining the presence of white pixels, I could ascertain if the text was at the front or back of the train.

Using the simple coordinate distance formula, I calculated the distance between all three compartments of the train and each exit and used only the compartment with the smallest

value of distance, considering all the exits at a station. If there were multiple lines at a station, I repeated the process using the Google Vision API until all lines were accounted for.

Problems Encountered

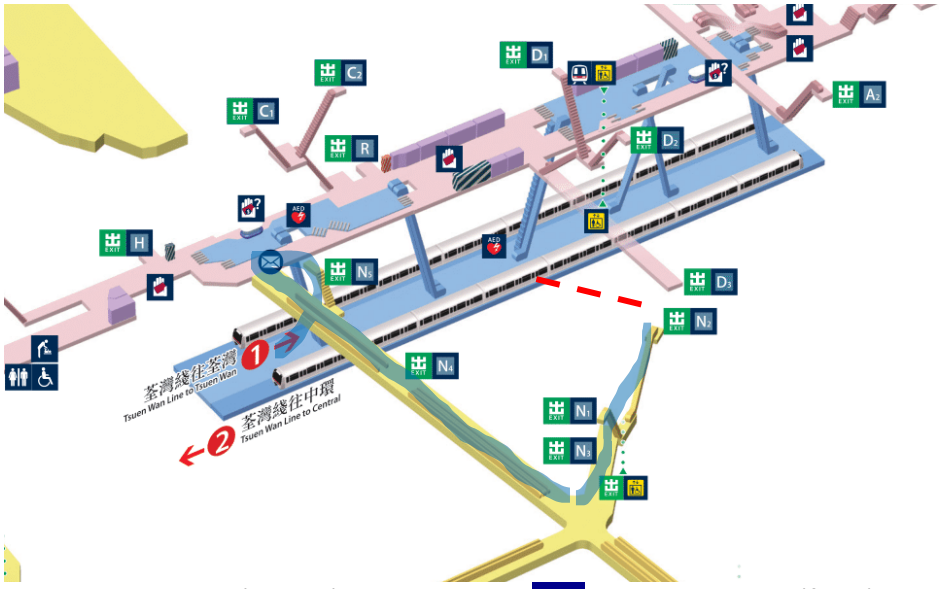
While working on this project, I encountered some challenges. Initially, I relied on the pytesseract library for optical character recognition (OCR), but it proved inadequate for recognizing slanted or 3D-styled text. To overcome this limitation, I switched to the more powerful Google Vision API, which significantly improved the accuracy of text recognition.

2-Dimensional text 3-Dimensional text

Another mistake I made was trying to find exit images downloaded from Google within the station layouts. However, due to resolution differences, this approach was ineffective. I rectified this issue by cropping the exits directly from the MTR station layouts.

Limitations

One limitation of the project arises from scaling down the 3D layouts to 2D representations. For example, in Tsim Sha Tsui Station, the closest compartment to exit N_2 is front, but our program may differ from om the observed closest compartment in the 3D layout. While this limitation cannot be systematically addressed, it only applies to a few stations, allowing for manual hardcoding of the correct compartment allocation.



Red – Program output (middle) Blue – Corrected path (front)

Assumptions

During the development of the project, I made certain assumptions:

- If a train was marked as the start of a line, no processing was performed for that train.
- Arrows only pointed to the left or right, without trains going upward or downward in the layout.

Advantages over Citymapper

Despite the existence of Citymapper, my app offers a distinctive advantage. Unlike Citymapper, which may require users to enter their current location, grant location access, specify their destination, and then find the appropriate train compartment within the suggested route, my app provides a more streamlined experience. Users only need to select the desired station and locate the exit in the list, saving them valuable time and effort.

By presenting this project, I aim to contribute to the optimization of MTR journeys and enhance the overall travel experience for passengers.

External Sources

Finding coordinates of subimages:

python - How to find subimage using the PIL library? - Stack Overflow

Finding multiple coordinates:
subimage/subimage/find_subimage.py at master · johnoneil/subimage · GitHub

Finding shapes in images:

python - How to detect different types of arrows in image? - Stack Overflow

Video on google vision setup:

Getting Started With Google Vision AI API In Python | Part 1 (youtube.com)
Getting Started With Google Vision AI API In Python | Part 2 (youtube.com)

Scraping text from webpages:

python - Scrape text from tag using beautifulsoup - Stack Overflow

JPG vs PNG:

JPEG vs. PNG: Which one should you use? | Adobe

IronPDF:

Python PDF to Image (Developer Tutorial) | IronPDF