MTR Transit Link

By Muhammad Hannan Javed

Last Updated: 7 February 2026

ABSTRACT

MTR Transit Link is a project designed to help Hong Kong MTR passengers quickly identify which section of an arriving train (Front, Middle, or Back) is closest to a chosen station exit. The system automatically downloads official station-layout PDFs from the MTR website, converts them to high-resolution PNG images, and analyses those images to locate platform train graphics, determine train direction, and detect exit markers. Train positions are inferred by detecting dark window pixels along platform train graphics using a combination of a moving bounding-rectangle scan and a straight-line extrapolation method; direction is determined from nearby directional arrows identified by text-coordinate context. Exit icons are detected by color- and shape-based scanning and then identified via OCR (with pre-processing) to recover exit labels. Results for each station are stored as JSON mapping each exit to the nearest compartment label (Front/Middle/Back). The app presents these mappings in a simple, two-click interface so commuters can rapidly plan where to stand on the platform and save walking time after alighting.

Table of Contents

List of Figures

1 Background

This project aims to optimize travel time for MTR passengers. The motivation comes from the common trend of passengers often finding themselves at the opposite end of the train relative to desired exit quite frequently when travelling to new stations. Saving a few mins by walking in advance can help in situations where each minute might count e.g. interviews, important meetings etc and scales up exponentially due to the vast number of passengers that travel daily. This project will assign closest compartments of train(s) to each of the exit(s) for all stations, with the assignment being Front, Middle or Back of the train.

A tedious but reliable way of knowing the closest compartment is to analyse the station layout from the MTR website[1]. However, this task takes significant time and effort as the passenger will need to navigate to website and find the desired station first. Furthermore, passengers are most likely to do this while they are travelling, and focusing to find the closest compartment require focus, which is difficult to do while on-the-move and in high crowd situations. The MTR Transit Link app aims to simplify this process by displaying the information in an easily accessible way. The only task will require finding the station name form the list however the list will be sorted alphabetically (unlike the website) so this task becomes easier as well.

2 Advantages Over Other Applications

Two popular apps that people use when commuting are Google maps and Citymapper. Google maps only inform the users about the exit itself that is closest to the destination, but not the compartment closest to that exit.

Citymapper offers an advantage over google maps by informing the closest compartment of train and is quite accurate. However, to do that the user needs to grant access to location, enable live location and enter the destination, which cumulatively becomes a longer task, especially when the user is only concerned about the closest compartment. The MTR Transit Link app allows faster access so users can just make two clicks to find section of the train closest to their desired exit. This might be preferable because passengers might have already travelled to that station and know their desired exit but forgot the closest train compartment.

3 Methodology

Before discussing the methodology, certain assumptions were made during the development:

- If a train was marked as the start of a line, no processing was performed for that train
- Arrows indicating train direction only pointed to the left or right, and trains did not go upwards or downwards in the layout

To start with, all station layouts were downloaded as PDF (Printable Downloadable Format) from the MTR website using a simple python web scraper.

[1] https://www.mtr.com.hk/en/customer/services/system_map.html

3.1 PDF to PNG

Since most OCR (Optical Character Recognition) engines work best with and are mostly used with images, all pdfs were converted to PNG images.

PNG format was preferred over JPEG since the latter is lossy format and preserves less information. The library used for converting was IronPdf. An advantage of this library is that it allows to convert to PNG images any custom DPI (dots per inch). A DPI of 250 was initially used to ensure high quality images optimal for OCR.

3.2 Finding Direction of Travel of Train

Google Vision API was used for figuring out the cartesian coordinates of the station line(s). Initially, text coordinates indicating the station line were stored – for instance coordinates of "East Rail Line to Admiralty" in figure 1. Later, these coordinates were used to determine the direction of the train – left or right – based on the arrow drawn close to this text. This turned out to be reliable as the arrows are always next to the text.

A simple approach for labelling train direction is illustrated in figure 1. It will be left if the arrow was found to the left and vice versa. This worked because there was no station where an arrow pointing to the left was found on the right side of the text and vice versa. Arrow always pointed away from text.

Now to find if the arrow is to the left or right of text, a sub image pixel analysis could be done by making a bounding rectangle box at left and right side of the coordinate, looking for any red pixels.

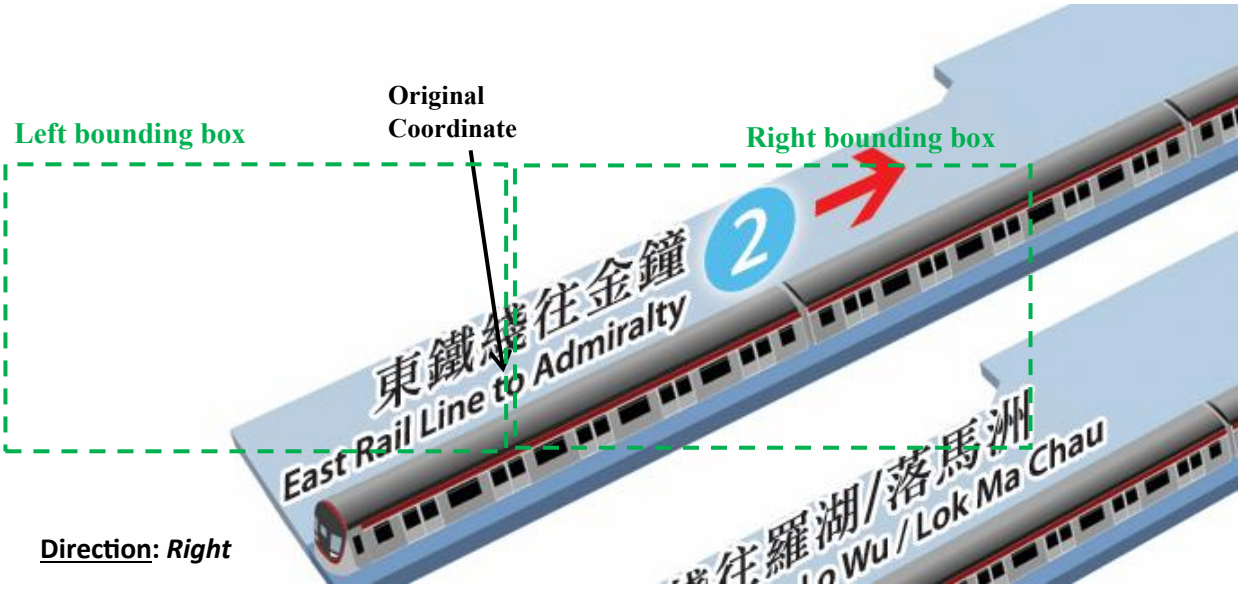

Fig 1. Bounding Boxes for Finding Arrows

3.3 Finding Coordinates of Train Compartments

Next step was to get the coordinates of each compartment – namely front, middle and back. There were several methods that could be used here. All methods are based on analysing black pixels in the RGB range (0, 0, 0) to (70, 70, 70) in the layouts since all train windows pixels

were within this range. Text would also be detected with these methods; however, some checks were put in place to avoid that.

3.3.1 Method 1: Moving Rectangle

All the pixels are analysed column by column. Whenever a black pixel is encountered in the RGB range, the next closest black pixel is scanned and so on shown by figure 2 and 3. Since the train lines are not horizontal, the rectangle spans both up and down when looking for next black pixel. This way it makes a scan of all train windows until no more are encountered.

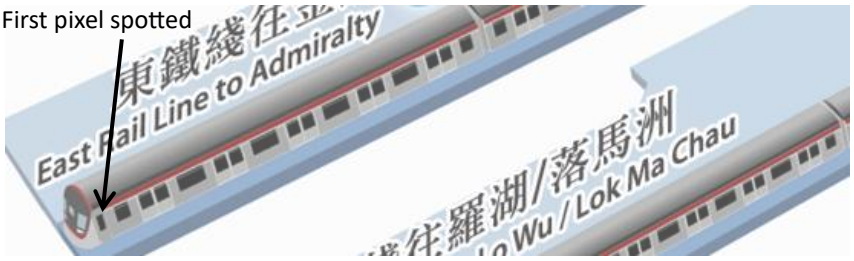

Fig 2. First Black Pixel Spotted

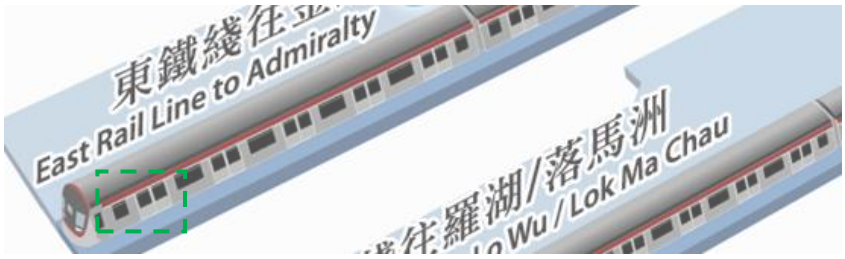

Fig 3. Bounding Box (green) for Next Black Pixel

However, it spotted black pixels which are not stations lines, e.g.

1. Text as a station line (shorter length)
2. Horizontal lines for indicating levels in figure 4

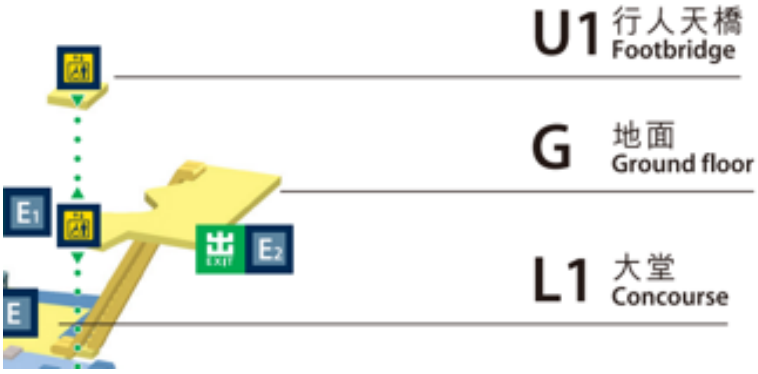

Fig 4. Horizontal Lines for Indicating Levels

However, it was easy to exclude these. For the first issue, since the length of the text will be much shorted than train lines, we can set a certain *minimum threshold* for the length of the line to be considered a train.

Similarly, the second issue could be solved by setting a *minimum change threshold* for y-coordinates. If there was no change in y-coordinate, then it is probably not a train line.

After tackling these problems, the results were still not reliable. Furthermore, some exceptional cases were found:

1. Off-track pixel chosen in figure 5 and 6

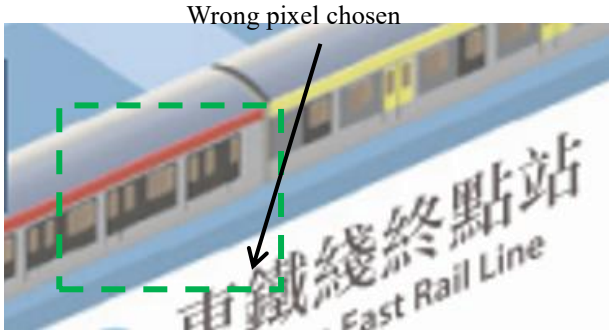

Fig 5. Wrong Pixel Chosen

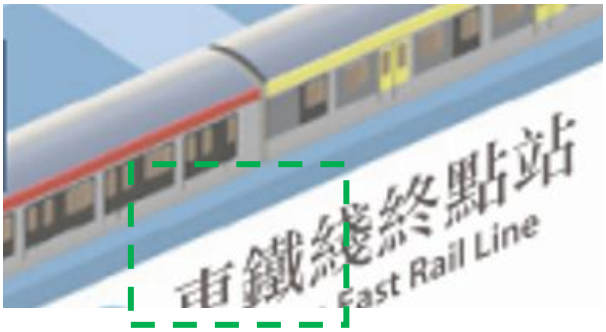

Fig 6. Bounding Box went off track

This was due to the order of finding pixels; algorithm looked for next pixel in top-bottom fashion. Even after adjustments, the problem persisted or gave rise to other problems.

2. The *Mei Foo* station in figure 7 was an exception to the algorithm since a large section of *Tuen Mun* line was obstructed and contained no black pixel, which made the algorithm terminate early

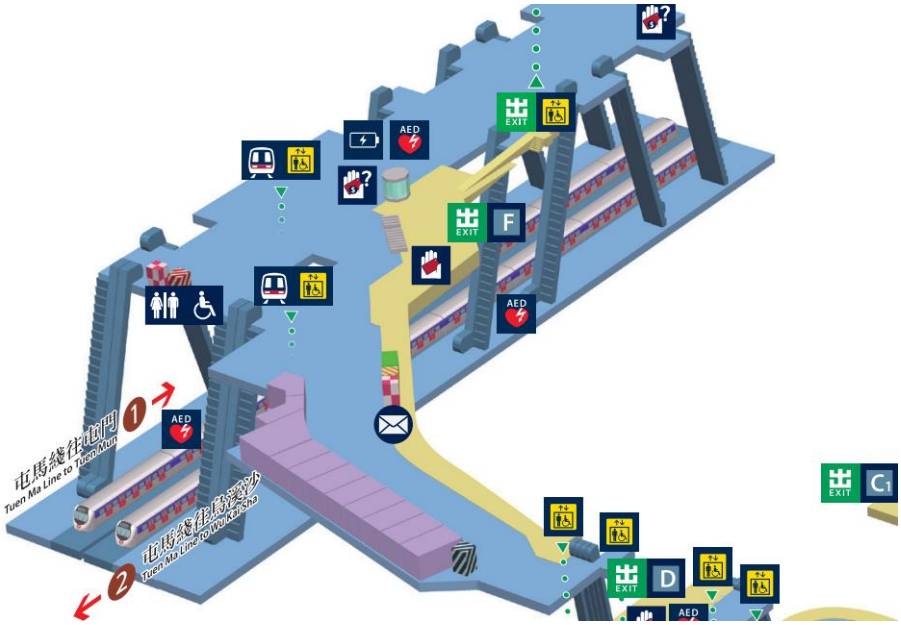

Fig 7. Mei Foo station layout – Tuen Mun line obstructed by upper floors

3.3.2 Method 2: Plotting Line

Another approach handled exceptional cases better and achieved better accuracy. Since a train was modelled straightly at most train stations, an equation for a straight line could be created that spanned across the train from its start to end drawn in figure 8. This required some first black pixels to be found to be extrapolated. A threshold of 50 pixels was initally used.

Fig 8. Straight Line (Green) through Train

The line inferred the ascent or descent of the train station from the change in x-y coordinates of the first 50 pixels. The pixels along the line were tracked until a white pixel was encountered. To avoid text and horizontal lines detection the same checks were placed.

However, new problems were encountered:

1. Like first issue in method 1, if first 50 pixels were off the line, then it changed the gradient greatly, resulting in the line going off the train quickly illustrated in figure 9

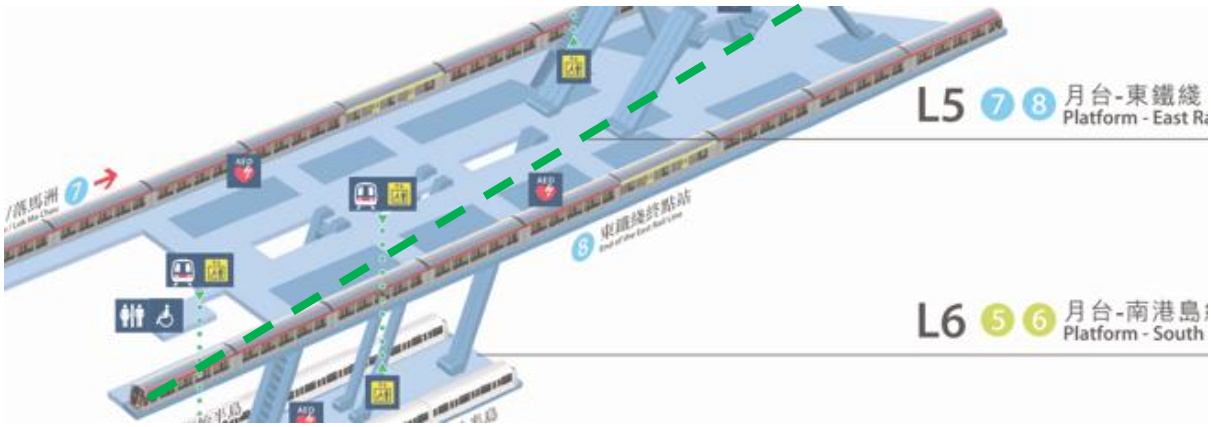

Fig 9. Wrong Line Calculated

2. Sometimes a white pixel was encountered on the roof of trains as in figure 10

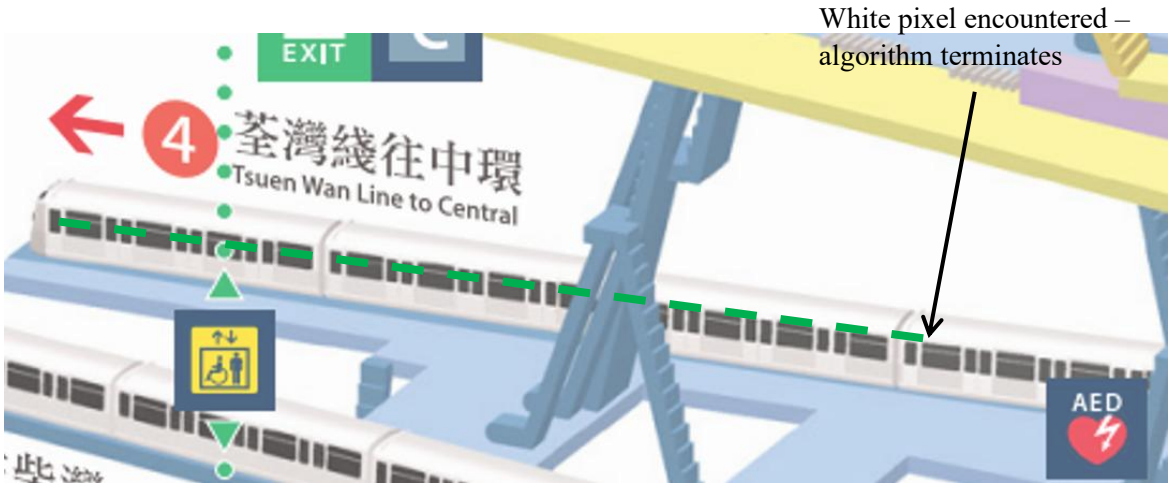

Fig 10. Early Termination of Algorithm after Founding a White Pixel

Although these miscalculations existed, this method overall proved to be better than method 1, and results were improvised together with some manual adjustment.

Once this method was used, the coordinates of front, middle and back of the train were figured out from the starting coordinate, the ending coordinate and the midpoint coordinate of the equation line after taking in account the direction of the train. Results were saved in JSON format.

One point to note is that only the x-coordinate was stored for this purpose. This is because train line mostly stopped horizontally at stations. Hence distance to an exit would be mapped just using 1 dimension – the difference in x-coordinate between exit and compartment.

An exception would be *Mei Foo* station, since *Tuen Mun* Line has a line with a significant difference in y-coordinate.

3.4 Finding Coordinates of Exits

Finding coordinates of exits proved to be easier. Reason being exits were presented as a flat, 2D image on the train station layout as shown in figure 11.

Fig 11. Image of Exit A

The only exception was found on the *Hung Hom* station, where a few exits had a rotated 3D image such as exit D2 in figure 12.

Fig 12. Image of Exit D2, Rotated

To obtain the coordinates of these exits, the RGB color range of the outer and inner boundary was pre-determined. The algorithm was straightforward:

1. Search the image for any pixel having RGB pixel in outer boundary color range.
2. If found, try to see if it is a square i.e. increment row and column values by one and if both stopped being the same pixel color value at the same time, then it must be a square.
3. Further check for inner RGB values since some other signs at station layout had same outer boundary RGB pixels.
4. If found, then it is an exit.

This proved to be quite effective – 100% coverage of exits in 95% of stations, with that 5% missing only a few exits.

After getting the exit sub image, next step was to identify the exit. To do this, Tesseract was used, a popular OCR library.

However, to increase accuracy, the image was be pre-processed before OCR was done. Namely two operations were performed – conversion to grayscale and increased contrast. The processed exit image is shown in figure 13.

Fig 13. Image of Exit C2 after Pre-processing

These two pre-processing operations greatly increased the accuracy of output from OCR. Only a few exceptions were found. For example, figure 13 above was incorrectly recognized as *G1* or *G*. A filtering script was written to clean these common incorrect patterns in the output. The results were then saved in JSON format.

3.5 Final Calculation

Once all stations were processed, the two corresponding files of a station were used for figuring out closest compartment. For example, figure 14 and 15 shows exit and train lines files for *AsiaWorld-Expo* Station respectively.


```
{
  "A": [
    519.0,
    1397.75
  ],
  "B": [
    2625.5,
    682.5
  ]
}
```


Fig 14. Exit File for *AsiaWorld-Expo* Station


```
{
  "Airport line to city": {
    "0": 2680.5,
    "1": 2087.5,
    "2": 1502.5
  }
}
```


Fig 15. Train Lines File for *AsiaWorld-Expo* Station

From these two files, it can be inferred that exit A is closest to back of train (denoted by "2") and exit B as front. The final file with calculation is shown in figure 16.


```
{
  "Airport line to city": {
    "A": "Back",
    "B": "Front"
  }
}
```


Fig 16. Final Calculated File for *AsiaWorld-Expo* Station

4 Application Demonstration

A simple application was built whose interface is shown in figure 17, 18 and 19.

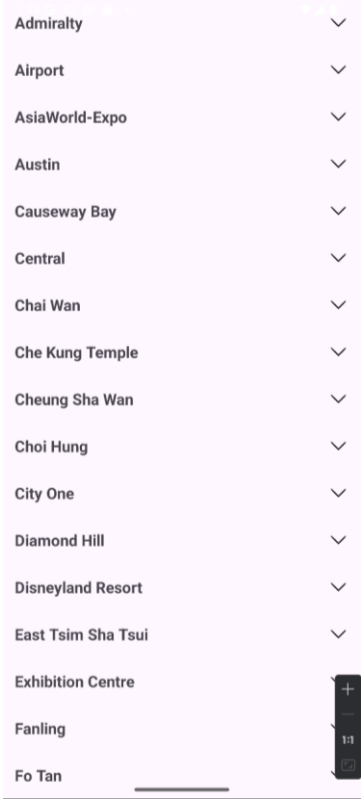

Fig 17. Home Screen

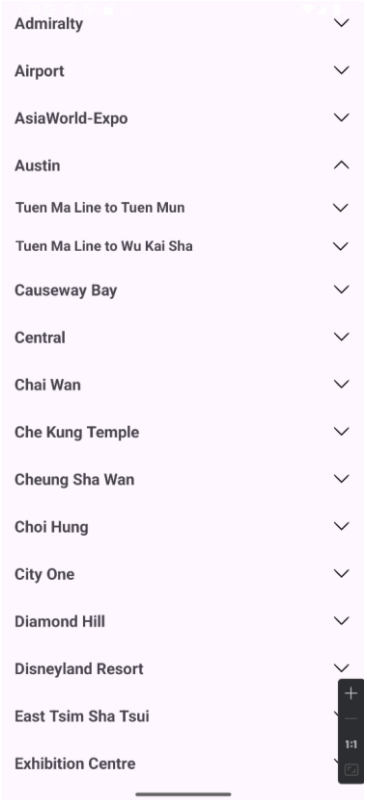

Fig 18. Expanded Austin Station

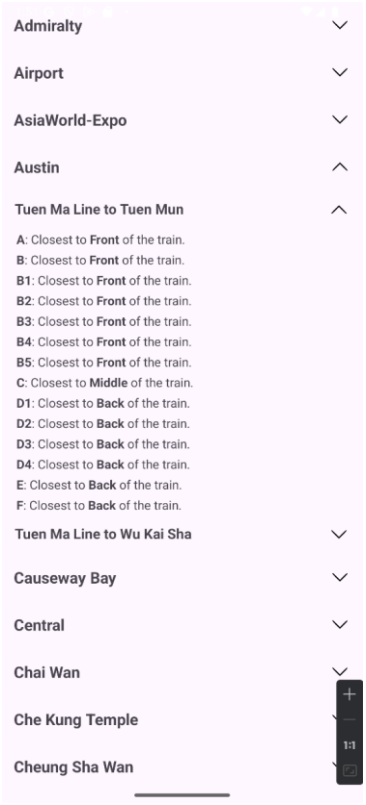

Fig 19. Expanded Station Line

5 Problems Encountered

5.1 Using Tesseract

Initially, simple open-source OCR libraries like Tesseract were planned to be used to perform all OCR tasks. However, some initial testing proved it to be unreliable as it could not identify text on stations. A major recurring problem was slanted station lines, since they were written in 3-dimensional manner as shown in figure 21.

Fig 20. 2D Text

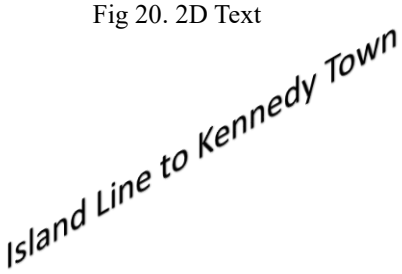

Fig 21. Slanted, 3D Text

For this reason, Google Vision API was used as it was more powerful in recognizing slanted text.

5.2 Sub Image Approach

For finding exits, OpenCV could be used for finding the sub images with within images.

However, as figure 22 and 23 show, this quickly proved to be a wrong approach as the size of exits varied across stations. Exit A could be 64 by 64 pixels in one station file but smaller in other.

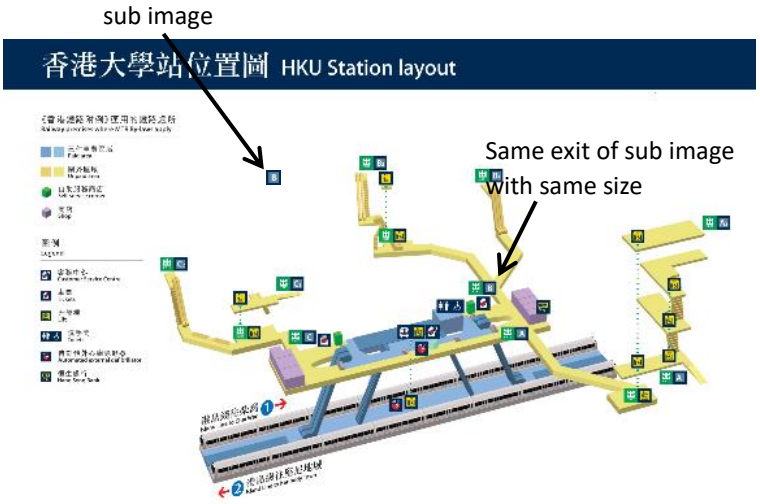

Fig 22. Sub Image and Exit of Same Size

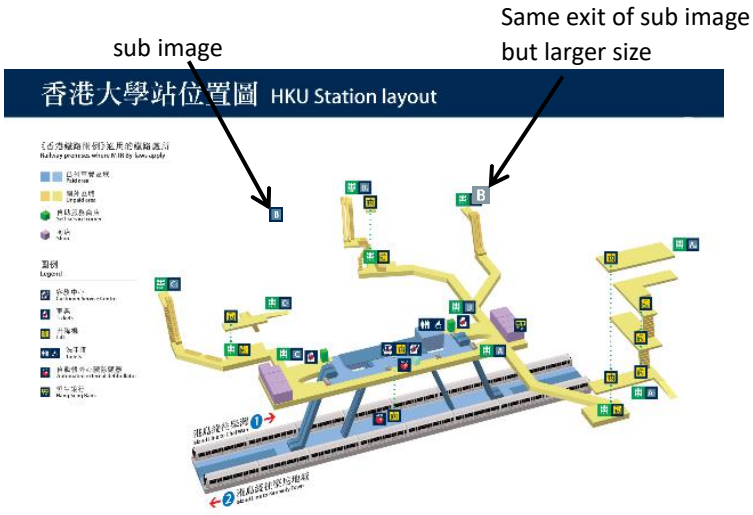

Fig 23. Sub Image and Exit of Different Sizes

Even after scaling up or down the size, the difference in resolution did not result in accurate outputs. The approach used proved more robust.

5.3 3D to 2D Approximation

Another limitation of the project arises from assuming the 3D layouts to be approximately same in 2D. Figure 24 shows Tsim Sha Tsui station, with a multitude of exits, that proved to be an exception where this assumption was wrong. The closest compartment to exit N_2 to the line *Tsuen Wan* Line to *Central* is the front compartment. But the program output differs because when using 2D image, it did not trace the route taken from the train to the exit, rather it took

the direct 2D distance from train doors to exit. So, it incorrectly mapped the closest compartment to be back of the train.

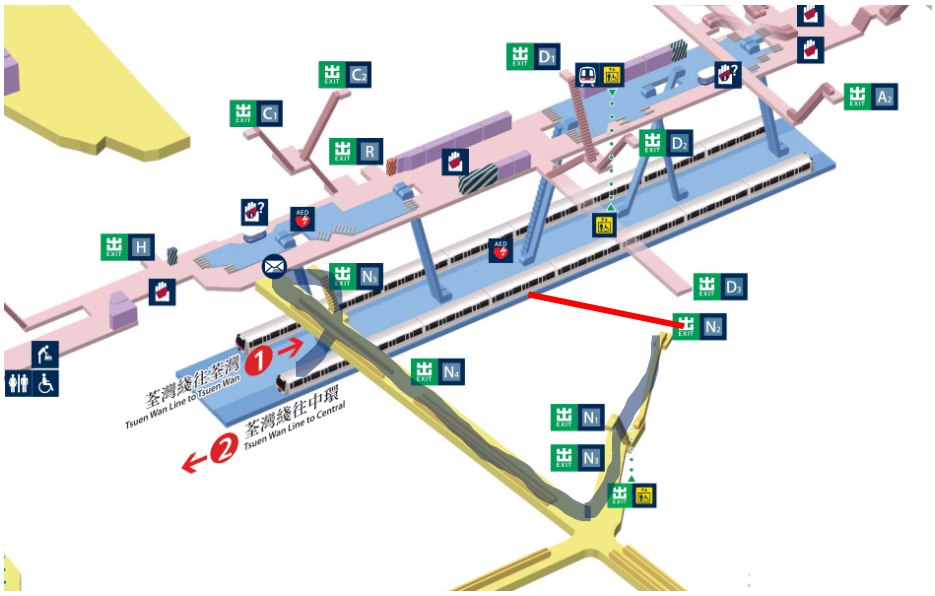

Fig 24. Incorrect and Correct Path from Train to Exit

Red – Program output **Blue** – Correct path

5.4 Minor Adjustments

During the development of the project, certain adjustments were made:

- For some unknown reason, *Exhibition Centre* station layout could not be converted to PNG by the IronPDF engine, so it was done using an online website[2]
- *Lok Ma Chau* station showed no exits, so it was not included in the results file

6 Prospects

The project had been a good learning opportunity and had achieved the desired result. However, there is still more that can be done on it.

People often change station lines within a station, and a lot of commute time might be composed of inter change between station lines. Hence the compartment closest to another train line could be mapped and displayed alongside exits. However, most stations have train tracks just parallel to each other, so *any* compartment is closest to that line. Additionally, this will result in much more information shown to the user making the interface complex.

To show the information in a more appealing way and to improve the user experience the map used by the official MTR HK mobile app could be used which enables users to click on the stations and display relevant information for that train station.

Finally, the app could be integrated within the official MTR HK mobile app to show users all the relevant information in one app. This will also reduce the number of apps user will need to download, making them rely on just a single app.

[2] https://pdf2png.com/

7 References

Finding coordinates of sub images:
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