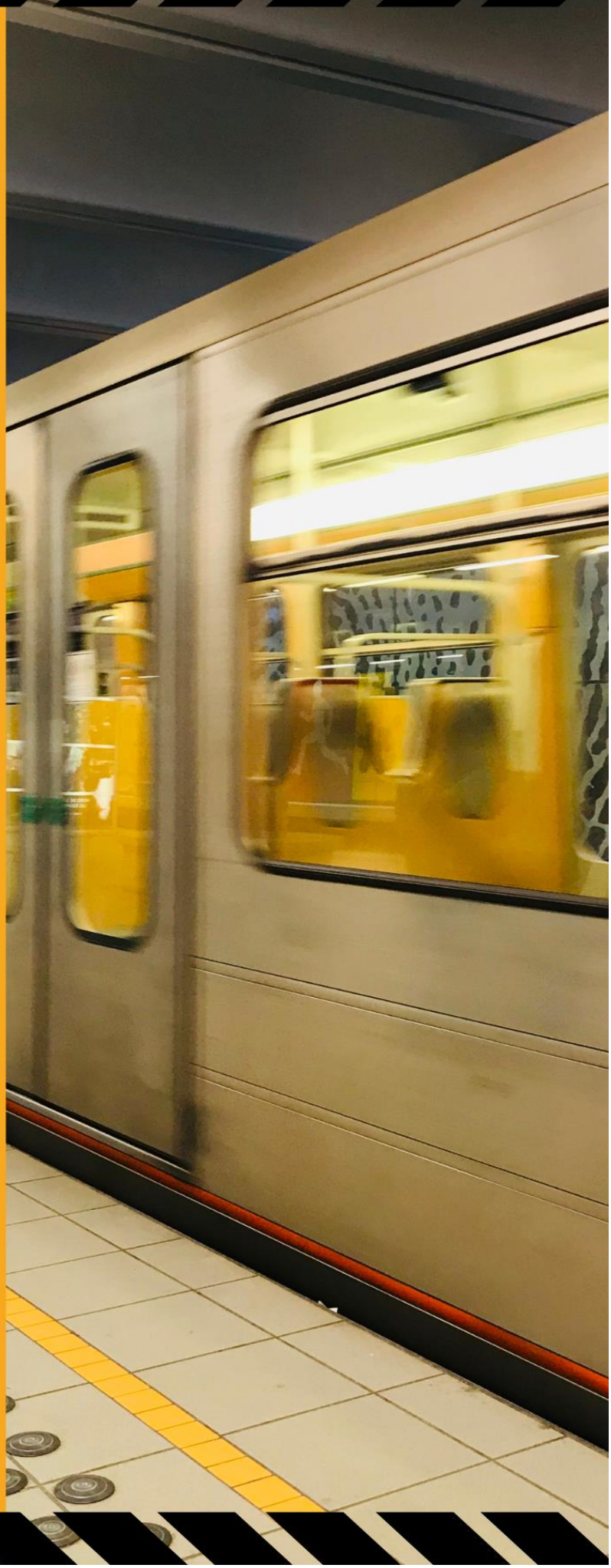
METRO SAFETY SYSTEM

POWERED BY AI



Metro Safety System

Digital Egypt Pioneers Initiative (DEPI) Graduation Project

Group Code: CLS_GIZ_AIS5_S2e

Submitted By:

Hanin Sherif

Ahmed Gabr

Ahmed Fawzy

Mahmoud El-Sayes

Ibrahim Ahmed

Yousef Elzahaby

October 2024

Table of Contents

Introduction

Crossing the safety yellow line on subway platforms poses a serious risk to life, and the consequences can be devastating. Last year alone, it is estimated that hundreds of people tragically lost their lives worldwide due to incidents involving stepping beyond this line. While accidents are common, some individuals also use this dangerous zone as a means for suicide, adding to the growing concern about platform safety. In response to this alarming reality, we have decided to launch our project, "Metro Safety System," with the goal of saving lives and enhancing the safety of passengers on subway platforms.

Project Overview: Metro Safety System

The "Metro Safety System" is a solution designed to enhance the safety of subway passengers by monitoring platform activity in real-time. Our system analyzes video streams from the subway's surveillance cameras, detecting both the presence of people on the platform and the safety yellow line. When a person touches or crosses this critical safety boundary, the system instantly triggers an alert to notify security person. This alert enables them to quickly review the camera footage, assess the situation, and take immediate action to prevent potential accidents or harm. By providing real-time awareness of dangerous scenarios, our system aims to reduce accidents and suicides on subway platforms, ensuring a safer environment for all passengers.

Data Utilization

Training Data

To train our model, we utilized two distinct datasets. The first is the "Station Platform and Detecting Braille Blocks 2" dataset from Roboflow, which contains four classes: railway track, braille blocks, platform, and stop braille blocks. For our project, we focused on two classes: **railway track** and **stop braille blocks**, as these are critical in defining the boundaries of the safety zone. The second dataset we employed is the "Person" dataset from Roboflow too, which consists of a single class: **person**. This dataset is essential for enabling the system to detect individuals present on the subway platform, ensuring accurate monitoring of human activity.

Inference Data

For real-time operation, we used video footage from a CCTV camera placed at the metro station to run inference on our trained model. This live data enables the system to continuously analyze the platform environment, detect when someone crosses the safety line, and immediately alert security personnel to take necessary actions. The use of actual station footage ensures the model performs effectively in the environment it is designed for, contributing to the overall reliability and safety of the system.

Datasets

- Roboflow station Platform and Detecting Braille Blocks 2 Dataset: https://universe.roboflow.com/hakujou/station-platform-and-detecting-braille-blocks-2/browse?queryText=&pageSize=200&startingIndex=0&browseQuery=true

- Roboflow person Dataset : https://universe.roboflow.com/abner/person-hgivm/browse?queryText=&pageSize=50&startingIndex=50&browseQuery=true

videos for inference

CCTV footage from different metro stations:

https://drive.google.com/drive/u/1/folders/1DU69sk9IpIX1_hC_8KT48v4OtEXR1CqC

Data Preprocessing

The preprocessing we need for this project is simple and usually done in the annotation stage like resizing the image to the same size, Augment the data and we made sure to use the data directly from Roboflow that outputs all the images in one size (640), and we used YOLO auto augmentation.

While preprocessing we realized the mismatching size between the person class data and all other metro elements, so we decided to divide them into 2 models instead of one to make sure the data analytics are handled the right way.

Methodology

We used two YOLO v8m models after many tries with different models and sizes to reach the best accuracy:

- Yellow line detection model: this model detects the yellow line that specifies the danger zone on the platform, it also detects the railway.
- Person detection model: this model was trained with a very well annotated person dataset to handle complex scenarios the right way.

After getting the predictions from the models we process them as the following:

- Yellow line: for this annotation we use a hybrid algorithm between color segmentation, masking and edge detection in the bounding box area to detect exactly where the line is and return the with the start and end point of the line.
- Railway: we use this annotation for many things which are: detecting if a train is in the station or not to specify when the danger area is a danger area, detecting on which side the

platform is in the early stages of the code to help specifying the danger area is on which side.

- Person: the most critical annotations because of complex scenarios and is treated with thresholds and an algorithm to find the exact foot position of the person trying to reach the best accuracy.

Then, the magic starts:

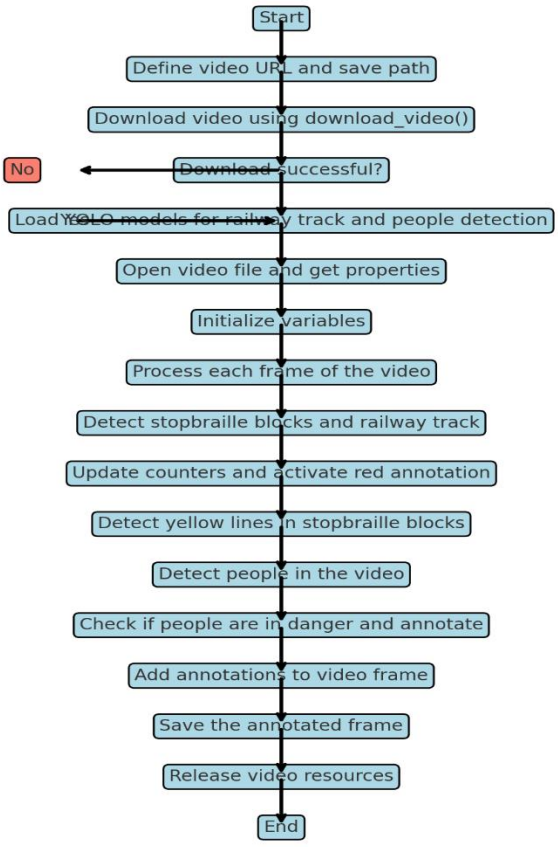


Fig 1: code flowchart

The code flow starts with a video link uploaded on a cloud service, then the video is downloaded to the work space, video is processed frame by frame, detecting objects, updating flags and variables, drawing annotations, triggering alerts, then an output video is produced.

Wrong annotations, fallbacks and strange scenarios are also handled using many different variables updated frame by frame to check on these and fall safely (e.g. yellow line isn't detected in some frames, railway wrong detections, train movement start which also considered dangerous, etc.).

Key features

- Most cases are handled trying to save more lives.
- Large masking range including almost every metro platform line color with auto completion using the start and end point also.
- The project gives different types of alerts as an output:
    - Red annotation around the danger zone detected persons.
    - Counter updated frame by frame for danger zone detected persons.
    - Flashing red alert if anyone crossed the line.

Fig 2: output sample

Deployment

For deployment we had several issues:

- Large model sizes, which we handled by uploading them to Dropbox in a downloadable format and updated our code to download them in their specified paths before running.
- Input/output handling, which was solved by using a cloud storage (google drive for simplicity) which we upload the input video to, get the link from our website which is send to the model and downloaded into the work space and for the output part we save the output to a constant path which the website handles and reads the output video with alerts from.
- End points: as explained we have simple post and get methods which creates a request with the on-cloud video link and returns with the output video.

After handling these issues, we developed an end point that takes the video URL and returns the output video file.

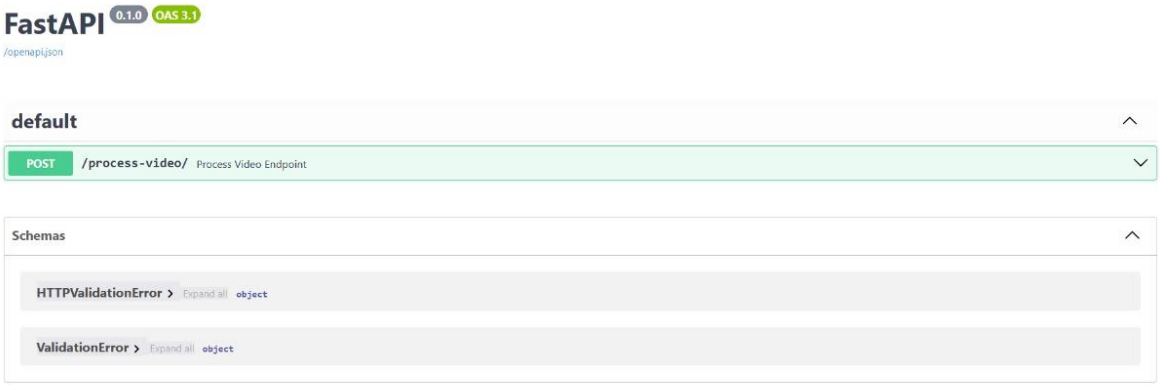


Fig 3: End points testing

Then we uploaded the project on GitHub and used Azure virtual machine service for deployment and creating APIs.

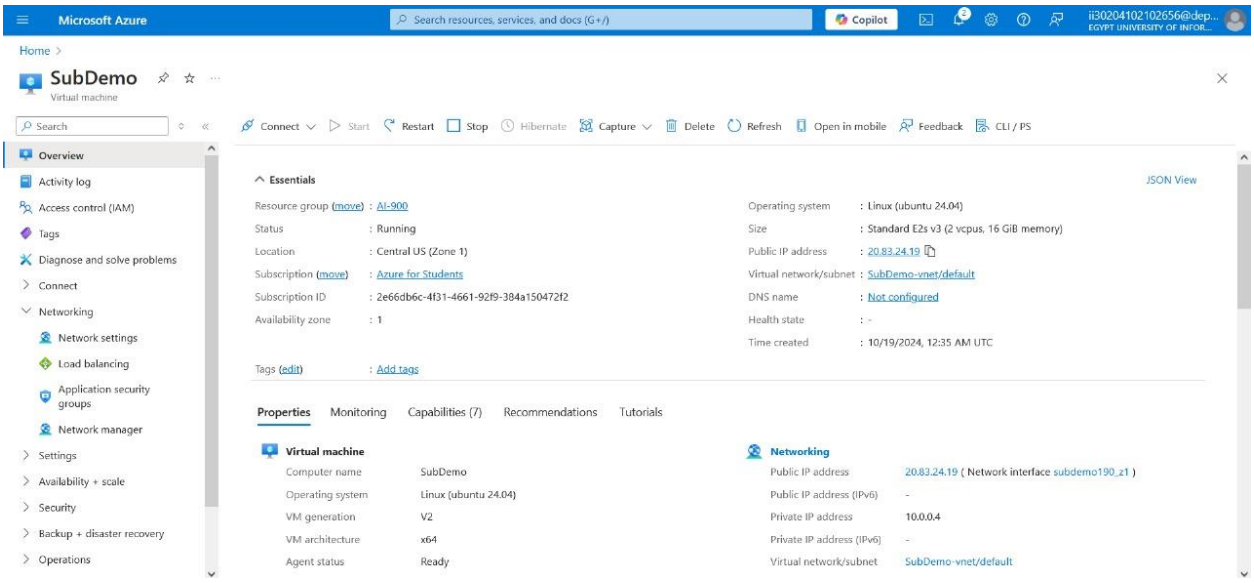


Fig 4: Microsoft Azure Virtual Machine Project

Gui

For Gui we implemented a simple website that explains our idea.

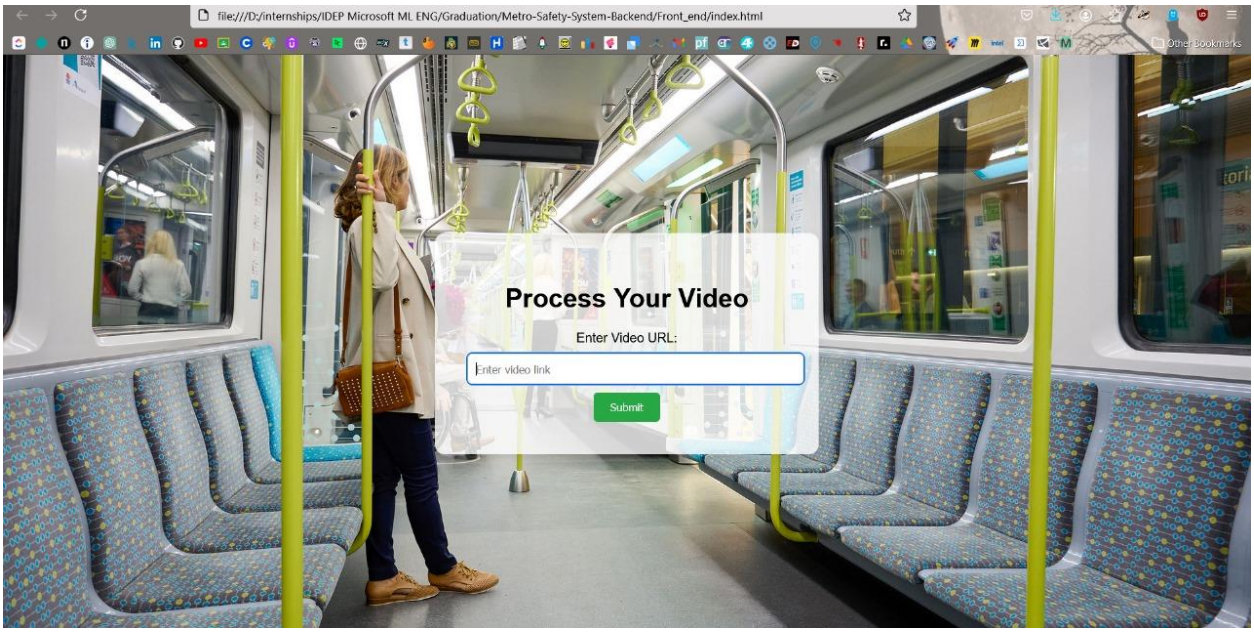


Fig 5: website UI

Our way to success

In this part I will talk about some of the approaches we tried but didn't work out:

Approach 1: fixed line with only train and person detections

In this approach we tried to ask the user to draw a fixed line on the screen and we only detected persons and trains, it worked out but there were many problems: wrong hand drawing, the line in actual scenarios isn't always straight from the camera perspective, if the camera moved a little it gives wrong alerts, train detection wasn't perfect due to its fast movement which confuses the models.

Approach 2: person, train and line detection

This approach didn't really workout, we had bad train detection as explained in the previous approach, also the line bounding box doesn't really tell us the position of the line it only detects it as a huge box, we even tried annotating manually our own dataset on Roboflow, but it didn't work out in the end: https://universe.roboflow.com/hanin-lhbac/yellow-line-ogwjr-mj3fd

Approach 3: ResNet segmentation for the line with object detection for train and person

Same as the others with the train annotations, but we got much better achieving our goal with the line, the problem is that it really doesn't detect much and it needs large amounts of data to train, which isn't available, we tried preparing 300 photos, we also tried a trained model on the same line but the results weren't what we aimed for.

Trained model repo: https://github.com/smirnovlad/SubwayDetectionAssistant

Future updates

Till now the project works on CCTV footage with a simple website, but in the future, we aim to handle live video photage which we prepared our code for by making the project process every frame individually.

The project only handles one platform per camera but not both which is the usual case, but if we needed, we can update it to handle both from the same feed.

Our work

- Project on GitHub: https://github.com/Ibrahim2656/Metro-Safety-System-Backend.
- Roboflow: https://universe.roboflow.com/hanin-lhbac/yellow-line-ogwjr-mj3fd
- Notebooks, models, testing outputs, etc. on Google Drive: https://drive.google.com/drive/u/1/folders/147fl297fVgvUA1COiA4sAZeABm0zpIQC
- Microsoft Azure Virtual Machine Deployment Project.

Appendix

Task assignment

Member\tasks	Task 1	Task 2	Task 3
Hanin Sherif (Team Leader)	Final Approach Models & Algorithms	3rd Approach Segmentation Model	Data Collection & Annotation Project
Ahmed Fawzy	1st Approach Yolo Model	2nd Approach Yolo Model	Data Collection
Mahmoud Elsayes	Annotation Project	Data Preprocessing (3rd Approach)	
Ahmed Gabr	Data Preprocessing (3rd Approach)	3rd Approach (Segmentation) Research	
Ibrahim Ahmed	Back end & End points	Deployment On Azure Virtual Machine	