Master in
Computer Vision
Barcelona

C6 Project: Video Surveillance for Traffic Monitoring
Final Presentation

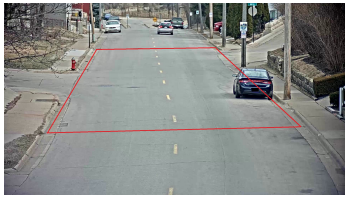
Group 3
Iker Garcia
Georg Herodes
Pablo Vega Gallego
Sígrid Vila Bagaria

0. Contents

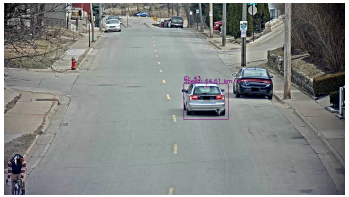
1. Affordable system to improve road safety
   a. Speed Estimation
   b. Our Own Data
2. Multi-Camera Tracking
3. Conclusions

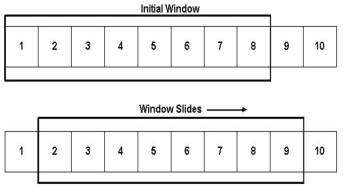
1. **Speed estimation:** Pipeline

Define area and find GPS coords → **Estimate the homography** → **Object detection & Tracking** → **Compute estimated speed** → **Estimation improvement**



$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$



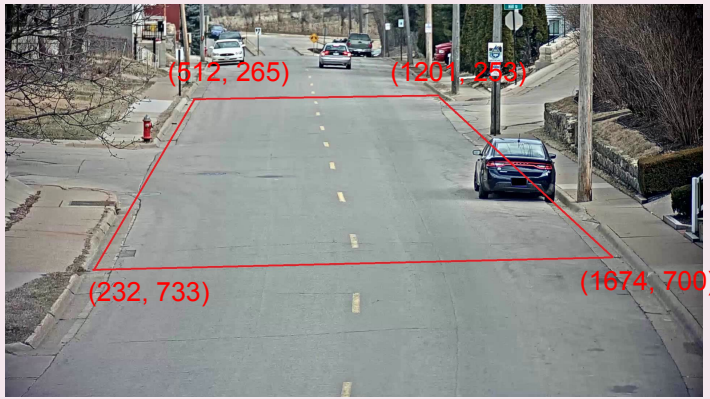


1.a Speed Estimation

1. Define a valid area and find GPS correspondences

1.1. Define valid area:

- We assume that the surface is COMPLETELY flat.
- Get coordinates for the 4 polygon vertices in pixels.
- Allows us to filter out vehicles.



1.2. Find GPS correspondences:

- Measure the real life dimensions of the defined area.
- Get the 4 correspondent coordinates for the 4 polygon vertices in meters.

2. Get homography matrix*

Used cv2.getPerspectiveTransform(), which uses the DLT algorithm.

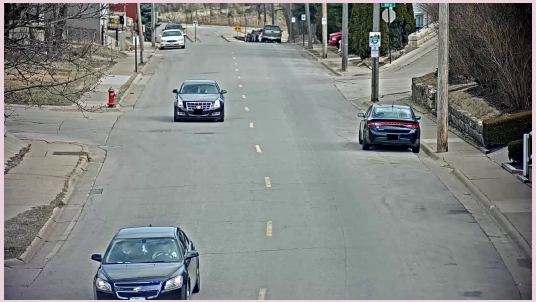
```
[[ 4.59874357e-02  2.75138504e-02 -3.08367375e+01]
 [ 1.83041188e-02  1.05096149e+00 -2.87876503e+02]
 [ 8.34061319e-05  6.70711916e-03  1.00000000e+00]]
```

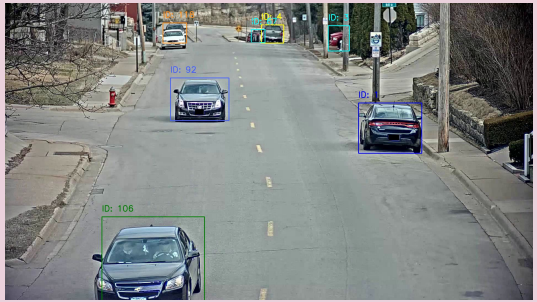
*In the case of this dataset, this matrix was already given, giving the conversion between 2d GPS coordinates into pixel coordinates.

1.a Speed Estimation

3. Object detection and tracking

- **Object detection:** YOLOv9*
- **Tracking:** DeepSORT





*Using YOLOv9-C weights, which will be defined later.

4. Compute estimated speed

Position last frame: [733, 277]	**Position current frame:** [732, 280]

Apply hom. matrix

Position last frame: [3.59484, 5.70630]	**Position current frame:** [3.582775, 6.7339277]

Get eucl. dist.

Eucl. dist: 1.02769m → Speed = distance /(1/fps) **10.27m/s = 36.99km/h**

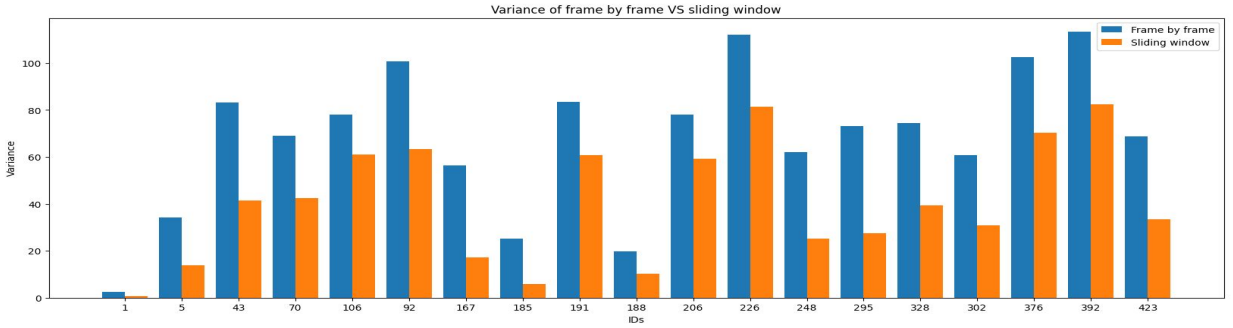
1.a Speed Estimation

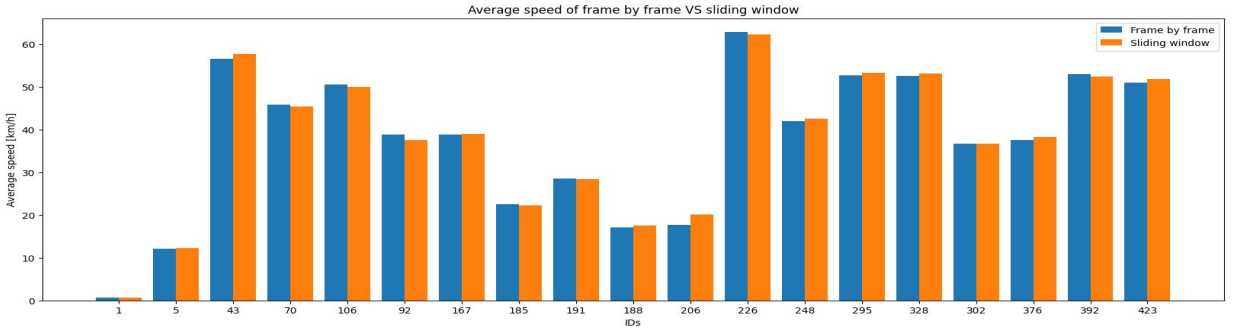
5. Improve speed estimation

A very **high variance** on the speed estimations leads us to find a solution:

⟶ Adding a **sliding window** that averages the actual value with the N last values.

As can be seen, the variance decreases significantly, while the overall average speed for every ID remains practically the same.





1.a Speed Estimation

ID	Estimated speed (km/h)
ID 1	0.70
ID 5	12.24
ID 43	57.77
ID 70	45.50
ID 106	50.01
ID 92	37.64
ID 167	39.04
ID 185	22.26
ID 191	28.48
ID 188	17.54

ID	Estimated speed (km/h)
ID 226	62.30
ID 248	42.63
ID 295	53.35
ID 328	53.19
ID 302	36.78
ID 376	38.30
ID 392	52.49
ID 423	51.95
ID 206	20.19

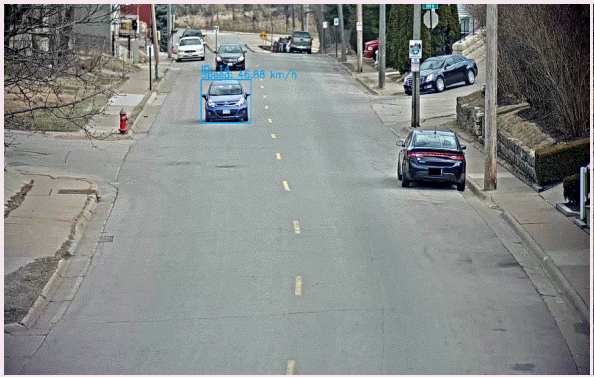
1.a Speed Estimation

6. Results

There is little to **no difference** between the speed estimations for **ground truth** annotations (GT) compared to our **YOLOv9** predictions.

→ YOLOv9 results were indistinguishable from ground truths as seen in Week 2.

Further insights will be gained from evaluating our system on unseen data.

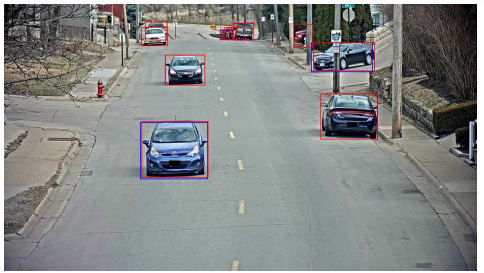


Ground Truth

Our predictions

Reminder: In Week 2 we used part of this same sequence to fine-tune YOLOv9.
The mAP is close to 1, any errors were caused by cars far in the background of the video that were lost due to resizing of images for YOLO inference.

YOLO v9	mAP@50	AP@50 Car	AP@50 Bike
Pre-trained	0.449	0.669	0.229
Fine-tuned	0.967	0.975	0.959



Ground Truth**, **fine-tuned model's predictions

1.b Speed Estimation with our own data I

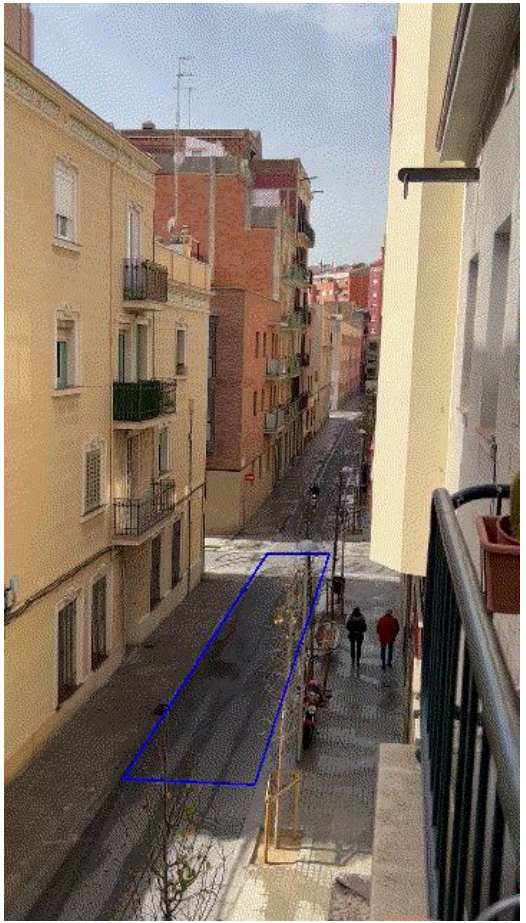
We created our own data by having Iker drive his motorbike down a street at 30 km/h (speedometer).

- Filmed from an elevated position.
- Used an area of approx. 21 x 3 meters as the ‘speed trap’ zone.
- We used pre-trained YOLOv9 weights.

Detections used	Average Speed	Speed variance
Annotations (GT)	27.97	3.73
YOLOv9	30.32	19.27

Average measured speeds were slightly under 30km/h…

BUT speedometers typically overestimate speed!



1.b Speed Estimation with our own data II

More difficult case:

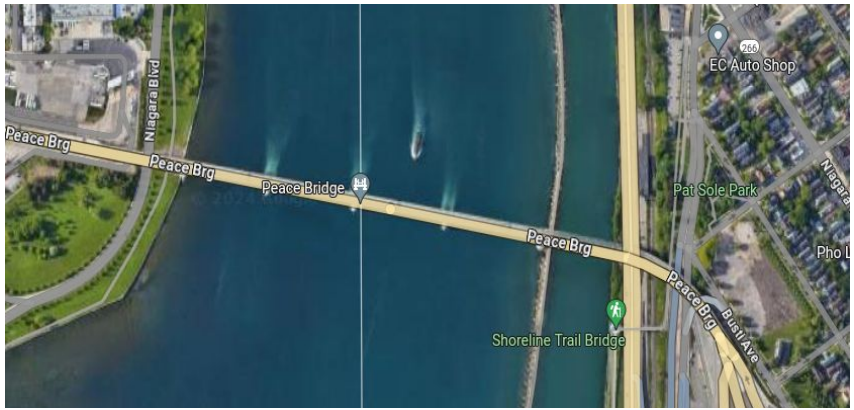
Livestream at a bridge between the US and Canada.

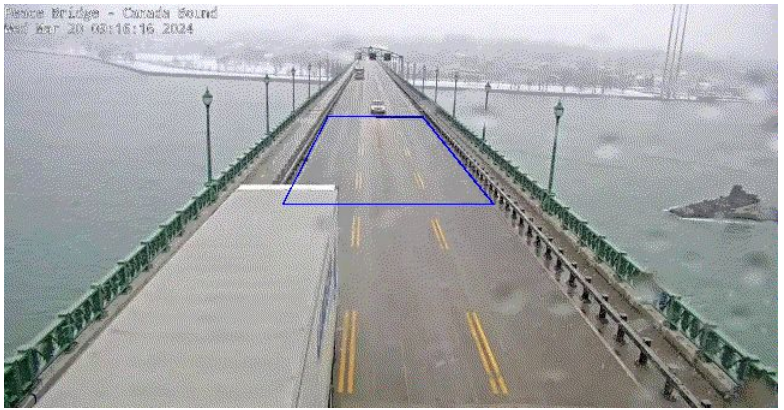
- Poor weather conditions: snow, water on the lens, fog.
- No ground truth info regarding speeds of vehicles.
- Specified a speed trap area of size 34m x 10m.

We used some simple sanity checks to make sure the results of the measurements make sense.

- Average measured car (ID 12) speed: 56.00 km/h
- Average measured car (ID 13) speed: 54.06 km/h
- Average measured truck (ID 1) speed: 40.34 km/h

The method worked reasonably well despite poor weather.



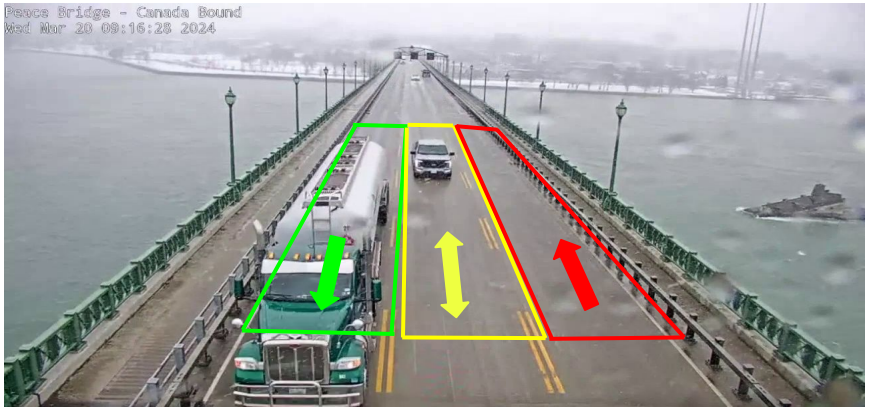


1.b Speed Estimation System Extensions

Currently, our method provides **insights into speeds** achieved on a specific road and the **amount of cars** using said road.

This information can be used to **interpolate data** regarding: peak hours, congestion, etc

Proper usage of ‘speed trap’ area(s) can provide lane-by-lane insights.



Extensions for analytics:

- Car make
- Car model/type
- Car color
- Action recognition

Extensions for enforcement:

- ANPR
- Red light camera
- Person detection
- Face recognition?

Red Light Camera

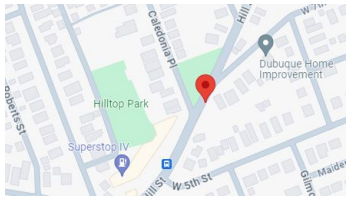
Compliance to GDPR and data protection/privacy regulations should be taken into account.

2. Multi-Camera Tracking: Pipeline

EXPERIMENTATION SETUP:
AMD Ryzen 9 5950X 16-Core Processor 3.40 GHz
NVIDIA Geforce RTX 4070 12 GB VRAM
64,0 GB RAM

Object detection → **Tracking** → **Compute the GPS coordinates** → **Intra-frame reidentification** → **Visualization and evaluation**






The following images are plotted on top of Google Maps, though the original street in the videos has been renovated, it is no longer the same :(

2. Multi-Camera Tracking: Object Detection

For the object detection, we decided to use the **YOLOv9** model with the pretrained weights.

We found it easy to fine-tune the model in case we wanted and also gave decent results on time execution, together with its good accuracy.

For the **object detection**, we decided to use the **YOLOv9** model with the **pretrained weights**.

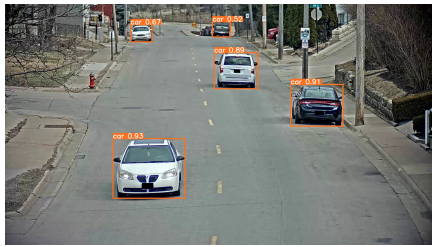
Model	Test Size	APval	AP50val	AP75val	Param.	FLOPs
YOLOv9-T	640	38.3%	53.1%	41.3%	2.0M	7.7G
YOLOv9-S	640	46.8%	63.4%	50.7%	7.1M	26.4G
YOLOv9-M	640	51.4%	68.1%	56.1%	20.0M	76.3G
YOLOv9-C	**640**	**53.0%**	**70.2%**	**57.8%**	**25.3M**	**102.1G**
YOLOv9-E	640	55.6%	72.8%	60.6%	57.3M	189.0G

[1]

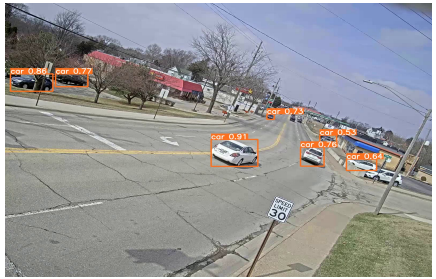
Speed: 0.7ms pre-process, 104.2ms inference, 48.6ms NMS per image at shape (1, 3, 640, 640).

S01/c01/1402

S03/c010/1201



S04/c032/88



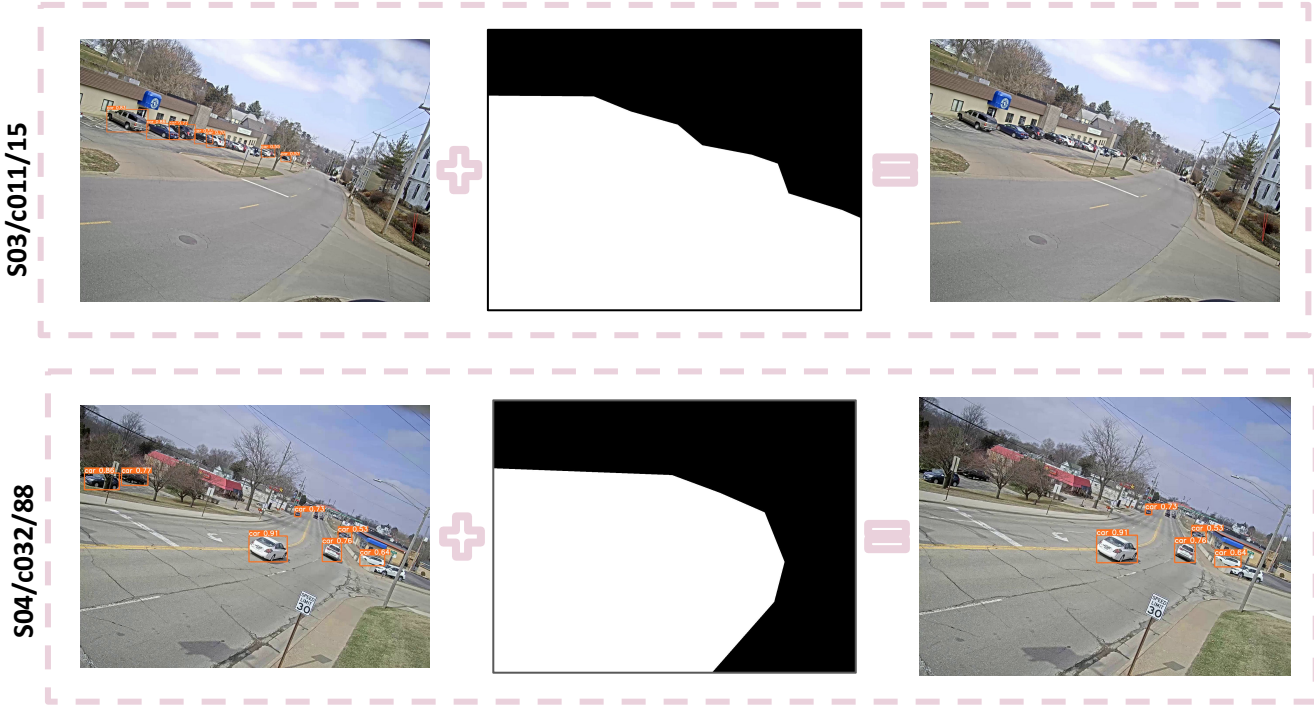
[1] https://github.com/WongKinYiu/yolov9

2. Multi-Camera Tracking: Data preprocessing

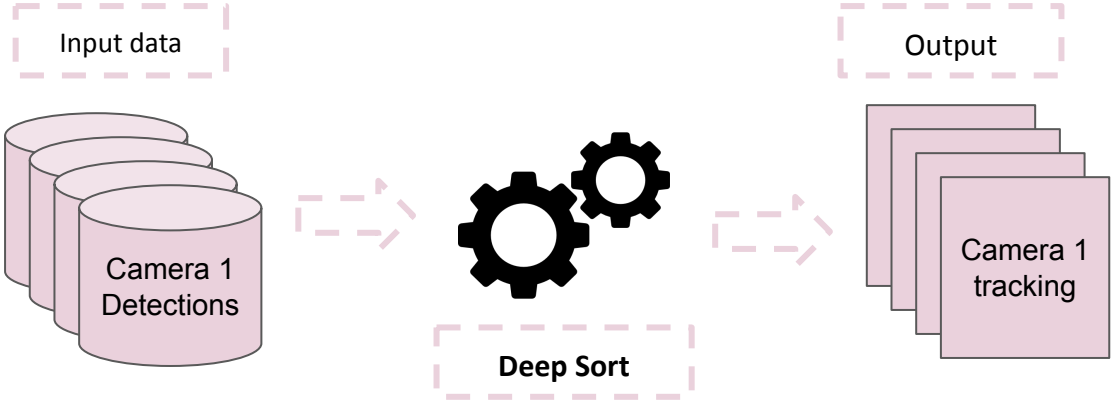
Based on dataset analysis, we can remove detections not relevant to the **Region of Interest** (ROI) image.

1. Check all detections if they are inside the ROI or outside.
2. In case they are not inside the ROI, we delete them from the predictions files.

Each of the cameras has an individual ROI.



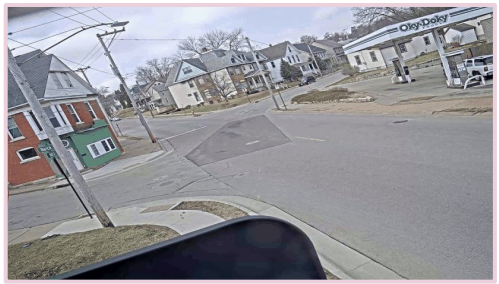
2. **Multi-Camera Tracking:** Tracking with Deep Sort



This is the same process we applied for the single camera multitracking tasks.

S001/c004

S003/c013

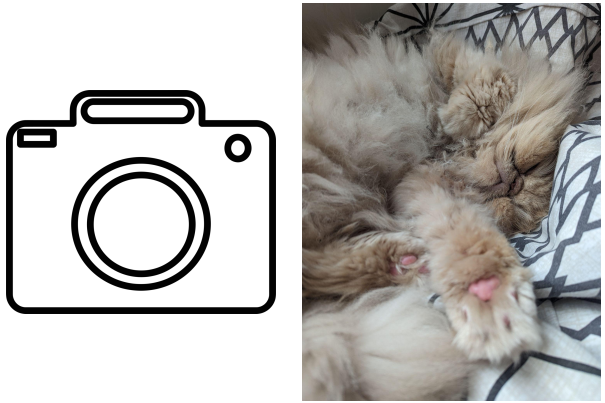


S004/c019

2. Multi-Camera Tracking: GPS coordinates

Once we only have the Bounding Boxes inside the ROI, we **project them to GPS coordinates**.

The dataset includes a txt file that with the values of a 3x3 matrix that move the GPS coordinates to image pixel coordinates.



Take an image

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1}$$

Apply the inverse of the homography

Get GPS coordinates

2. Multi-Camera Tracking: GPS coordinates

The next step is to **compute the GPS coordinates**
We use **Folium** to put the transformed detections to the GPS coordinates space.
This is done in 2 steps:

1. Compute the centre of the bounding box.

CENTER:
[1401.385 454.]

2. Transform the center of the bounding box into GPS coordinates

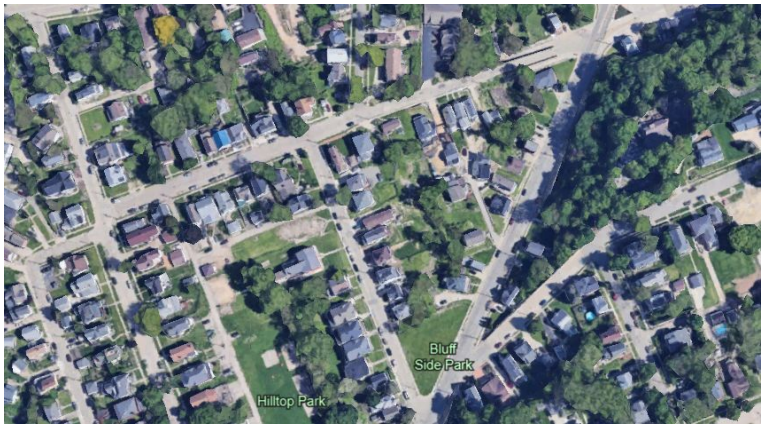
$$\begin{bmatrix} -82 & 159 & 17973 \\ -4.1 & 2 & 365 \\ 0.01 & 0.01 & 1 \end{bmatrix}^{-1} [1401.385 \quad 454. \quad 1.]^T = [\ 42.49748344\ -90.67348965]$$

Folium Map

Street view

2. Multi-Camera Tracking: GPS coordinates

We had some **problems** with the Folium library, as it doesn't have implemented the function to plot of trajectories by frames to create visualizations.
Solution: get a piece of map and plot the GPS values on it, mapping the GPS values to pixel coordinates.



Google Earth view of the street in SEQ03

For each GPS point:

1. Subtract the minimum latitude and longitude.
2. Divide by the difference between latitude and longitude.
3. Multiply by the image height and width.

Image Size:
height = 473
width = 855

Latitude and longitude boundaries:
min_lat = 42.49722222
max_lat = 42.49972222
min_lon = -90.67666667
max_lon = -90.67133333

2. Multi-Camera Tracking: Solving ambiguities

The same car may appear in multiple cameras → in the projection, the car appears multiple times due to projection error.

1. Detect the cars that are **very close** and **look very similar**.
2. Merge their tracklets.

2. Multi-Camera Tracking: Solving ambiguities

FOR ALL DETECTIONS IN A GIVEN (SYNCHRONIZED) FRAME:

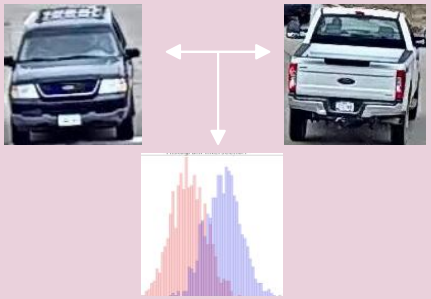
1. DISTANCE

Compute Haversine distance for all pairs BBoxes from different cameras.

Haversine distance because it is used for points on a sphere.

2. APPEARANCE

Compute a histogram of each BBox (32 bins* for each channel) and compare it with all remaining BBoxes.



3. MERGING

Merge all tracks that remained into a single ID.

For all tracklets considered to be the same vehicle:

1. Find the longest, keep the ID.
2. Eliminate all the others in frames that the longest is not.
3. Put the ID of the longes into the remainings of the others.

If the distance between a pair of BBoxes is smaller than a threshold, go to step 2.

If the appearance between a pair of BBoxes is smaller than a threshold, go to step 3.

**32 bins because it seemed like a good enough value, it is a parameter to optimize in further experiments. The same for all the thresholds used.*

2. Multi-Camera Tracking: Qualitative results

- It is hard to assess visually.
- Some cases remain to be solved due to lack of hyperparameter optimization.
- After intra-frame reidentification, some cars maintain their ID though the cameras.

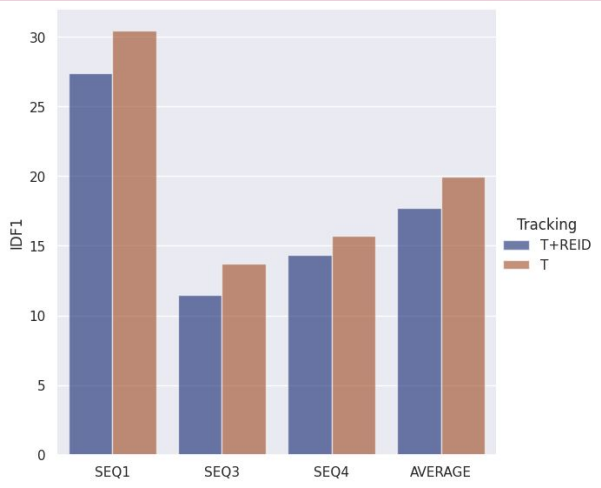
2. Multi-Camera Tracking: Qualitative results

- It is hard to assess visually.
- Some cases remain to be solved due to lack of hyperparameter optimization.
- After intra-frame reidentification, some cars maintain their ID though the cameras.

2. Multi-Camera Tracking: Quantitative results

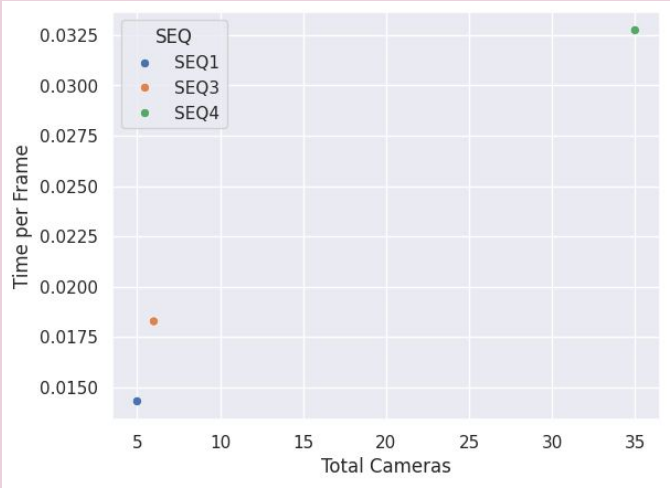
IDF1:

- Overall slight decrease in performance (2%).
- The main problem is inter-frame reidentification (for both approaches).



EXECUTION TIME:

↑ Number of cameras ↑ Time per frame



3. Conclusions

- Implemented homography-based method for speed measurement using monocular camera.
  - Achieved accurate results for known vehicle speeds, and reasonable results for unknown speeds.
- Identified ways to enhance system for comprehensive traffic analytics and law enforcement.
- Encountered issues with homography-based visualizations; only visualizations available for SEQ03.
- Planned implementation of metric learning for cross-camera matching, but time constraints prevented it.
- MCMT results fell short of expectations due to imperfect re-identification.

Future work:

- Implement inter-frame reidentification.
- Consider the velocity of potential tracking merging.
- Optimize thresholds for appearance and distance.
- Fine-tuning.
- For appearance, use the semantic segmentation of vehicles or use a learned method instead of using the entire bounding box.


Thank
YOU!