

Traffic Monitoring System

Assignment

Cristian Gariboldi



einride

Deep Learning Engineer

Interviewer: Taigo Maria Bonanni

Index

1	Vehicle Detection and Tracking	5
1.1	Vehicle Detection	5
1.1.1	Pre-processing	5
1.1.2	Bounding Boxes	6
1.1.3	Confidence Scoring	6
1.1.4	Non-Maximum Suppression (NMS)	6
1.2	Tracking	6
1.2.1	Centroid Computation	6
1.2.2	Hungarian Algorithm	7
1.3	Vehicles Counting	7
1.3.1	Auto-Calibration	7
1.3.2	Color and Class Filtering	8
1.4	Fine-Tuning on Domain Specific Dataset	9
2	Speed Estimation	10
2.1	Homography	10
2.2	Speed Computation and Filtering	11
3	Position Prediction	12
3.1	Kalman Filter	12
3.1.1	Model Architecture	12
3.2	Transformer	14
3.2.1	Problem Statement	14
3.2.2	Model Architecture	14
3.2.3	Training Setup	14
3.3	Results	15
3.3.1	MAE per horizon (pixels)	15
3.3.2	RMSE per horizon (pixels)	15
3.3.3	Summary stats	15
4	Vision Language Models	17
4.1	SmolVLM-256M	17
4.1.1	Scene Understanding	18
4.2	llava-v1.6-34b	20
4.2.1	Anomaly Detection	20
4.2.2	Automatic Homography Calibration	21

List of Figures

1.1	YOLO11 is the latest model and delivers state-of-the-art performance. The nano version is suitable for real-time applications.	5
1.2	Auto-Calibration of virtual gates for counting incoming and outgoing vehicles.	8
1.3	Sample image data for fine-tuning.	9
2.1	Interactive and automatic homography tool.	11
4.1	Frame used for scene understanding.	19
4.2	VLM as anomaly detector.	20
4.3	Reference image for automatic camera calibration.	21

Abstract

This is my technical report of Einride assignment for the Deep Learning Engineer position.

In this paper, I collected all the technical choices and algorithms I selected and designed for solving the assigned tasks of the challenge plus an additional experiment I conducted on my own and which I propose here, going beyond the requirements of the challenge, leveraging and integrating VLMs within the pipeline. The work is divided in 4 main chapters:

1. Vehicle detection and tracking
2. Speed estimation
3. Position prediction
4. Bonus point: complex scene understanding using VLMs

Enjoy the reading!

1 Vehicle Detection and Tracking

1.1 Vehicle Detection

For this task, we detect vehicles on the road using YOLO11n exported to ONNX format. The base YOLO11n network has been pre-trained to detect different vehicle classes such as cars, vans, buses, trucks and motorbikes, so it is already deployable within our pipeline.

In `detector.onnx.py` we perform:

1. Input preprocessing (letterbox + RGB + normalization);
2. Model inference;
3. Post-processing (confidence scoring, bounding boxes, de-normalization and class-wise non maximum suppression).

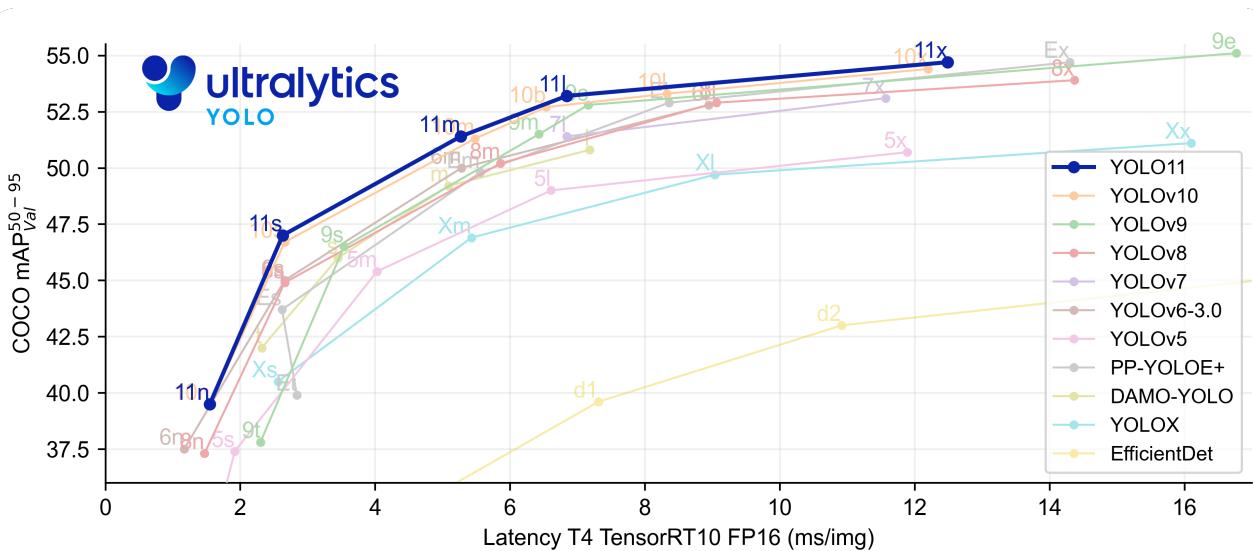


Figure 1.1: YOLO11 is the latest model and delivers state-of-the-art performance. The nano version is suitable for real-time applications.

1.1.1 Pre-processing

Here, we resize the original frame to the classical input size of the network (640) while preserving aspect ratio and avoid distortion. To fill the remaining area, we use constant padding.

1.1.2 Bounding Boxes

YOLO11n predicts a vector box geometry and class logits. The model could potentially output both center-based boxes $[x_c, y_c, w, h]$ or corner-based boxes $[x_1, y_1, x_2, y_2]$. In the first case, we take the values in network coordinate frame and then convert them to pixel coordinates. Finally, we map them back to image coordinates.

1.1.3 Confidence Scoring

YOLO11n provides both an objectness score o and per-class softmax scores p_c . So, we compute a detection score $s = op_c$.

Thanks to this detection score, we are able to filter predictions by a configurable thresholds.

1.1.4 Non-Maximum Suppression (NMS)

NMS is very useful for removing overlapped bounding boxes of the same object. Practically speaking, we keep the box with the highest detection score s , then we remove all other boxes with an intersection over union value greater than a defined threshold.

1.2 Tracking

The detection is further enhanced with a tracker based on centroid positions of bounding boxes and assignment leveraging the Hungarian algorithm. The goal of multi-object tracking is to:

- Maintain persistent ids for moving objects across frames;
- Reconstruct object trajectories, which are the centroid positions over time;
- Provide robust associations suitable for vehicle counting, speed estimation, position predictions that we will discuss in the next chapters.

Basically, for each detection in a frame (which contains bbox coordinates and class/score), we define a set of active tracks. The tracker must compute an assignment mapping tracks to detections, in order to understand if detections of different frames belong to the same object, and consequently, id.

1.2.1 Centroid Computation

Before explaining the core logic of the tracking algorithm, let's define the centroid computed by the tracker.

$$c = \left(\frac{x_1 + x_2}{2}, y_2 \right) \quad (1.1)$$

As you can see, I decided to use the bottom-center centroid because it is less sensitive to partial occlusion of the top of the vehicle. Consequently, gate crossing (will be analyzed later) is more precise. Additionally, this choice also improves speed estimation as bottom points of bbox lie closer to the road surface. The homography will be obtained from road references, so transforming pixel coordinates to world coordinates will lead to a more precise speed estimation when considering bottom-center centroids.

1.2.2 Hungarian Algorithm

The Hungarian algorithm is an optimization algorithm designed to solve the assignment problem. For our specific task, we define the Euclidean distance as the cost function we want to minimize.

Let's suppose that in one specific frame the active tracks has the centroids in $[c_{ti}]_{i=1}^M$ and detections have centroids in $[c_{dj}]_{j=1}^N$.

The cost function is defined as:

$$C_{i,j} = \|c_{ti} - c_{dj}\| \quad (1.2)$$

which is the Euclidean distance we want to minimize.

The assignment problem implemented in `scipy.optimize.linear_sum_assignment` returned matched row/column indices (i, j) .

After the assignment, we reject the matches with a cost $C_{i,j}$ higher than a defined threshold. The accepted matches instead update the associated tracks with the new bbox and centroid. Unmatched detections create a new object id.

1.3 Vehicles Counting

1.3.1 Auto-Calibration

In order to count incoming and outgoing vehicles, I decided to place two virtual gates on the image. By tracking the vehicles with the algorithm explained above, if an object first pass through to bottom gate and then to the top one, we count it as outgoing vehicle, and vice versa for incoming vehicles. If the tracker is accurate, this strategy is very robust.

But counting vehicles with two virtual gates requires placing them at an appropriate position, so that it is clearly detectable which vehicles are incoming and outgoing. To avoid hardcoding the position of the gates and make the solution more scalable for different kind of cameras, angles or environment, I defined an auto-calibration method, that automatically places the virtual gates in the optimal position, taking into consideration the average position of vehicles in the first n frames.

The auto-calibration algorithm works as follow:

1. For the first n frames (tunable parameter), the detector runs and for every detection of vehicles we collect the bottom y -coordinate of the bboxes;
2. After collecting the first N samples, we compute two quantiles of the empirical distribution of sampled y positions:

$$y_{top} = Q_{top}(y_{sample}) \quad (1.3)$$

$$y_{bottom} = Q_{bottom}(y_{sample}) \quad (1.4)$$

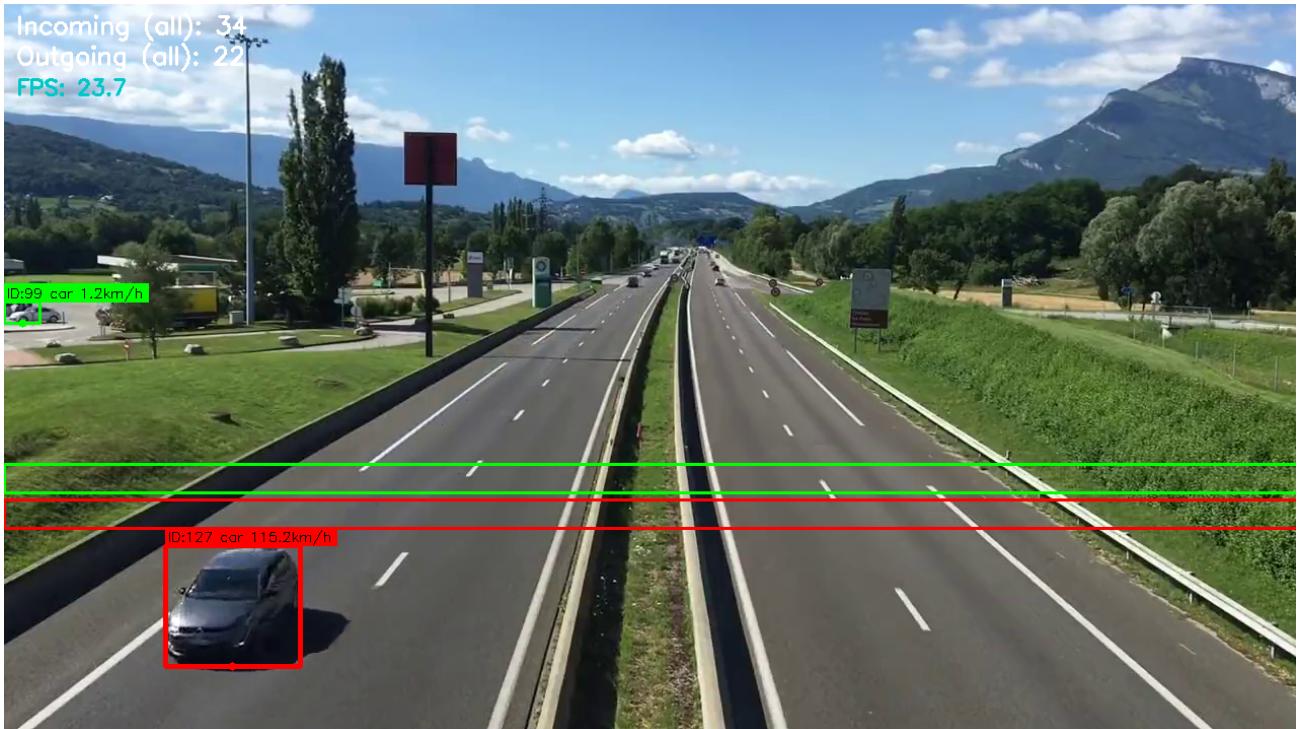


Figure 1.2: Auto-Calibration of virtual gates for counting incoming and outgoing vehicles.

1.3.2 Color and Class Filtering

In order to only count vehicles with specific colors or classes, we implement a method via YAML config file, to make runtime behavior configurable and scalable without code changes. The following filters have been implemented:

- Color filter (white, red, blue etc.);
- Class filter (car, van, truck etc.);
- Area filter (bbox pixel min/max area);

- Combined filter (a combination of the above filters).

For color filter, we first compute a ROI of the detected bbox (to avoid roofs or windshields), so that we can focus on the vehicle body, which probably reflect more its main color.

All color checks then operate in HSV, which is more robust for color detection. If number of pixels of the same color in the interested area are more than a defined thresholds, then we assign that color to the bbox (some cleanup and noise reduction is done to reduce single-pixel outliers and small bright spots).

1.4 Fine-Tuning on Domain Specific Dataset

In order to make the base YOLO11n model more expert in identifying vehicles in the traffic, I decided to fine-tune it on an open source labeled dataset.

The dataset includes more than 1000 training labeled images with bboxes in highway traffic environments, as you can see in Fig. 1.3.

It only provides three different classes, namely "bus", "car" and "truck".

I have already exported the fine-tuned model to ONNX format and it is already included in the software to be tested within the whole pipeline (see instructions in the README file).



Figure 1.3: Sample image data for fine-tuning.

2 Speed Estimation

The speed estimation component leverages a homography to map image-space detection to ground-plane coordinates (meters), it computes instantaneous speeds from successive observations and produces a smooth speed estimate in m/s then converted in km/h.

Let's define P as the position of an object in the world-coordinates, the instantaneous speed is computed as follow:

$$v_{inst} = \frac{\|P_{t_1} - P_{t_0}\|}{t_1 - t_0} [m/s] \quad (2.1)$$

Finally, it detects the fastest vehicle in the frame, highlighting its bbox in red.

2.1 Homography

In order to understand how to map pixel coordinates to ground-plane coordinates, I designed a tool which interactively ask you to select 4 points in the image, and to insert the real world measurements in meters. Consequently, the tool creates the following json file which is then used by our speed estimator to convert pixels coordinates to world ones.

```
{
    "image_points": [ [x1, y1], [x2, y2], [x3, y3], [x4, y4] ],
    "world_points": [ [X1, Y1], [X2, Y2], [X3, Y3], [X4, Y4] ],
    "units": "meters",
    ...
}
```

In this json file, $[x_i, y_i]$ correspond to pixel location and $[X_i, Y_i]$ to the corresponding world-plane location.

The homography H is a 3x3 matrix that best satisfy $P_i = Hp_i$, where $P_i = [X_i, Y_i, 1]^T$ and $p_i = [x_i, y_i, 1]^T$.

In detail:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.2)$$

For this specific challenge, I spotted one frame of the video clearly showing a truck. I estimated the lanes to be approximately 3.6 meters large and the length of the truck to be around 25 meters, as showed in Fig. 2.1.

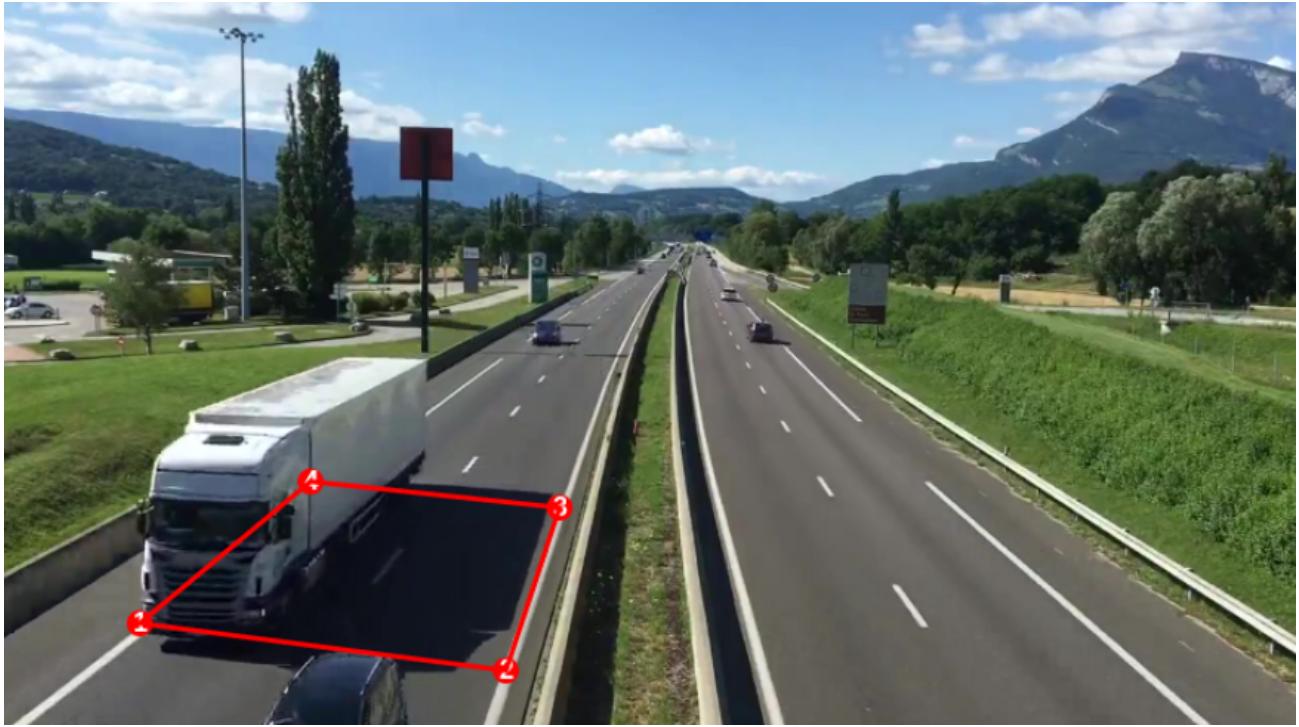


Figure 2.1: Interactive and automatic homography tool.

2.2 Speed Computation and Filtering

Once the homography has been correctly set up, the speed estimation process follows the following logic:

1. Given two observations of the same vehicle, we compute its centroids (x_0, y_0) at time t_0 and (x_1, y_1) at time t_1 .
2. We convert those locations in world-plane coordinates using homography H :

$$P_0 = (X_0, Y_0) = H(x_0, y_0)$$

$$P_1 = (X_1, Y_1) = H(x_1, y_1)$$

3. We compute the instantaneous speed as:

$$v_{inst} = \frac{\|P_1 - P_0\|}{t_1 - t_0} \text{ [m/s]}$$

and we convert it in km/h.

The instantaneous speed can be noisy due to jitters or localization errors, so a filter is applied for smoothing out the measurements.

The filter is the **Exponential Moving Average**, defined as:

$$v_n = \alpha v_n + (1 - \alpha)v_{n-1} \quad (2.3)$$

where v_n is the most recent instantaneous speed and v_{n-1} the previous smoothed speed. $\alpha \in (0, 1]$ is a configurable parameter, low values correspond to more aggressive smoothness.

3 Position Prediction

For predicting future positions of detected vehicles, I decided to design two different models: a light-weight online predictor that uses Kalman filters (constant velocity model) and a learning-based approach leveraging a transformer to predict vehicle image positions for the next m frames, together with metrics that evaluate predictions against ground-truth labels.

3.1 Kalman Filter

The reason why I selected a linear Kalman filter with a constant velocity motion model, besides its simplicity and its light-weight computations, is because the environment we want to analyze is very suitable for the aforementioned assumptions: being in a highway setting, vehicles typically follow lanes and are moving at constant speed, so the linear Kalman filter is a simple and robust solution, especially for short-term predictions. Additionally, it is a strong deterministic baseline against which to compare learning-based approaches.

3.1.1 Model Architecture

Let's define the 4D state vector as:

$$X_t = \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix} \quad (3.1)$$

which represents pixel coordinates and velocities in px/s.

The discrete dynamic system (with constant velocity) is defined as:

$$X_{t+1} = FX_t + W_t \quad (3.2)$$

with

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

where $W_t \approx N(0, Q)$ is the process noise, which in our case models acceleration and/or model mismatch. Specifically, $Q = \text{diag}(\sigma_p^2, \sigma_p^2, \sigma_v^2, \sigma_v^2)$ with positions' and velocities' noise.

Since we don't observe velocities directly, but only the positions, the measurement model is defined as follow:

$$Z_t = HX_t + V_t \quad (3.4)$$

where Z_t represents the measured position from the detector (bottom center centroid),

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

because we only measure x and y , not velocities.

$V_t \approx N(0, R)$ is the measurement Gaussian noise. Specifically, $R = \sigma_{meas}^2 I_2$, where I_2 is a 2x2 identity matrix. R models uncertainty in our detector measurements.

The state/covariance prediction and update steps follow the following logic and mathematical formulations:

1. State prediction (new state from previous state):

$$\hat{X}_{t|t-1} = F\hat{X}_{t-1|t-1} \quad (3.5)$$

2. Covariance prediction:

$$P_{t|t-1} = FP_{t-1|t-1}F^T + Q \quad (3.6)$$

3. Total expected error in predicted measurement:

$$S_t = HP_{t|t-1}H^T + R \quad (3.7)$$

4. Kalman gain (measurement vs prediction):

$$K_t = P_{t|t-1}H^T S_t^{-1} \quad (3.8)$$

5. State update (update prediction with measurement):

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t(Z_t - H\hat{X}_{t|t-1}) \quad (3.9)$$

6. Covariance (uncertainty) update:

$$P_{t|t} = (I - K_t H)P_{t|t-1} \quad (3.10)$$

In order to predict the position m frames into the future, where the future measurements are not available yet, we do filter cloning. Specifically, we deep copy our current Kalman filter, including both state estimate and state uncertainty. Then, without further measurements (open-loop), for each step t from 1 to m , we use the predict step of the Kalman filter to project the vehicle's state forward based on our constant velocity motion model (so instead of measuring new state, we use its predicted state).

3.2 Transformer

I define in this section the learning-based approach I designed to perform position prediction of detected vehicles using a transformer architecture.

3.2.1 Problem Statement

For each tracked object we observe a sequence of centroids in pixel-coordinates.

For a single sample, we observe up to n past positions. The learning objective is to predict future positions up to m frames.

To focus more on motion and simplify modeling, we convert global positions to relative displacements, so that input features become past frame-to-frame deltas and target future displacements are relative to the last observed/predicted position.

So, the model learns to map input states to output relative displacements, which are then converted back to absolute coordinates.

The dataset used for training and validation is constructed with gt observations, where we store the centroid positions observed by the detection module described in the first chapter.

3.2.2 Model Architecture

Before feeding the centroid positions into the transformer directly, they are first mapped with a learnable linear projection, which transforms the original 2-dimensional deltas ($\Delta x, \Delta y$) into a higher-dimensional space, creating new feature representations. This new tensor is the input of the transformer.

We use a standard sine/cosine positional encoding and apply it to the input tensor.

Afterwards, a stack of transformer encoder layers, using multi-head self-attention, learns relationships across these timesteps.

Finally, a small multilayer perceptron maps the transformer embeddings directly to all future steps at once (not step by step), producing a sequence of predicted values.

3.2.3 Training Setup

We employ Mean Squared Error (MSE) as the loss function, which is the per-sample squared Euclidean distance summed across all future steps. We use Adam optimizer with a learning rate of $lr = 1^{-3}$, a batch size of 128 and typical training up to 40 epochs with early stopping based on validation loss. During validation, the metrics computed per epoch are per-horizon Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

3.3 Results

Now, let's compare the results of both methods in predicting future positions at different horizons based on previous observed frames. We measure the performance in two different ways: the linear Kalman filter with constant velocity motion model and the transformer model, both described in previous chapters. The metrics are MAE and RMSE. The results show that the transformer outperforms the kalman filter for different horizons.

3.3.1 MAE per horizon (pixels)

 Table 3.1: Per-horizon MAE (pixels), $m = 10$.

Method (n)	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
Kalman (n=3)	8.40	12.60	16.68	20.83	25.00	28.89	32.95	36.83	40.56	44.14
Kalman (n=6)	8.49	12.67	16.72	20.80	24.70	28.42	32.13	36.07	39.71	43.53
Kalman (n=10)	8.25	12.01	15.94	19.85	23.60	27.48	31.16	34.70	38.17	41.68
Transformer (n=3)	4.10	6.84	9.35	11.61	13.86	15.97	18.13	20.24	22.37	24.47
Transformer (n=6)	3.81	5.92	7.79	9.56	11.17	12.91	14.39	16.01	17.58	19.17
Transformer (n=10)	3.66	5.52	7.05	8.36	9.58	10.82	12.02	13.21	14.61	15.88

3.3.2 RMSE per horizon (pixels)

 Table 3.2: Per-horizon RMSE (pixels), $m = 10$.

Method (n)	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
Kalman (n=3)	14.05	20.03	25.76	31.72	37.45	42.68	48.26	53.52	58.60	63.41
Kalman (n=6)	14.65	20.61	25.99	31.59	36.88	41.94	47.02	52.12	56.78	61.74
Kalman (n=10)	13.65	18.62	24.28	29.55	34.30	39.41	44.12	48.69	53.17	57.70
Transformer (n=3)	10.13	14.26	17.90	21.20	24.19	27.10	29.93	32.71	35.47	38.17
Transformer (n=6)	9.73	13.15	16.04	18.60	20.88	23.10	25.14	27.15	29.18	31.25
Transformer (n=10)	9.43	12.48	14.77	16.68	18.37	20.08	21.75	23.49	25.21	26.97

3.3.3 Summary stats

 Table 3.3: Aggregated mean MAE and mean RMSE across horizons ($m = 10$).

Method (n)	Mean MAE	Mean RMSE
Kalman (n=3)	27.07	39.55
Kalman (n=6)	24.93	38.93
Kalman (n=10)	23.03	36.35
Transformer (n=3)	15.14	25.11
Transformer (n=6)	11.78	21.42
Transformer (n=10)	9.53	18.92

The transformer model substantially outperforms the simple Kalman filter baseline in all horizons and for different n values. Moreover, by increasing n we can see consistent improving performance across all horizons. The Transformer benefits from more context: the attention/encoder can leverage longer motion patterns and hence predict future motion better. The reason why the transformer improves its accuracy with higher n values is because it learns statistical motion patterns (acceleration, steering, typical approach/turn behaviors) from the distribution of training tracks (not in our case since it is a highway so we don't have large turns or steering). More observed frames increase the signal-to-noise ratio for those patterns and allow the encoder to resolve recent accelerations and direction changes.

The Kalman filter instead, only shows modest improvements with larger n values. This is expected: Kalman filter is essentially a parametric constant-velocity model — extra past frames mainly help initialization of velocity and covariance but do not enable modeling of non-linear maneuvers.

4 Vision Language Models

This chapter aims to go beyond pure detection, tracking and prediction of objects in the environment. The goal is to provide analysis of complex semantic understanding and reasoning about the environment.

Indeed, previous models I considered for solving the tasks (such as YOLO11n, Kalman filter and the transformer), are limited to specific tasks. However, VLMs world knowledge can be very useful for more scalable solutions and complex understanding tasks. I show in this chapter some possible integrations of VLMs with our traffic monitoring system pipeline, to see if they can add value to our pipeline. They can be integrated to solve many other tasks than the ones I show in this chapter. The only limit is imagination (and computational capabilities). I designed two different integrations:

1. Production ready solution, using the smallest VLM in the world in ONNX format.
Suitable for deployment in embedded systems;
2. Research oriented solution, leveraging a bigger VLM in Pytorch format. The scope is to push the limits of VLMs understanding when computational capabilities are not the main issue.

4.1 SmolVLM-256M

SmolVLM is the smallest multimodal model in the world, making it a strong candidate for on-device applications, while maintaining good performance on multimodal tasks such as image captioning, visual question answering, and storytelling based on visual content. It can run inference on one image with under 1GB of GPU RAM.

There are a bunch of possible applications we can solve by integrating SmolVLM within our pipeline, for example:

- "Semantic Filter" for Advanced Counting: when YOLO11n detects a bbox, we crop it and give it as input to the VLM, which instead of classifying the object as "car", "truck" etc, it is also able to provide the exact model like "Fiat Punto" or "Mini Cooper" without further training;
- VLM-Powered Automatic Camera Calibration for Speed Estimation: instead of manually building the homography.json file, we can design an automatic pipeline (I did it, will show in next chapters) in which we ask the VLM to automatically detect reference objects in the image and design the homography;
- High level scene understanding and anomaly detection.
- Environmental condition tagging.

4.1.1 Scene Understanding

The input of the VLM are a frame of the video and a text prompt, which in our case is the following:

”Describe in detail the traffic and the environment in this image.”

In the case of Fig. 4.1, the output of the VLM is the following:

The image depicts a two-lane highway with a clear demarcation of lanes. The lanes are divided into two parallel sections, each separated by a median. The highway is flanked by trees and shrubs, and the road is bordered by a median. The road is wide enough for vehicles to pass each other, with no visible traffic signs or markings.

In the center of the image, there is a large sign with a red signboard. The sign is rectangular and has a white background with black text. The text on the sign is partially visible, but it appears to be a warning or a disclaimer. The sign is mounted on a pole, and it is attached to a metal post.

On the left side of the image, there is a large tree with a few branches and leaves. The tree is located near the center of the image, and it is partially obscured by the sign.

On the right side of the image, there is a large tree with a few branches and leaves. The tree is located near the center of the image, and it is partially obscured by the sign.

In the background, there are several large mountains covered with trees and greenery. The mountains are partially obscured by the trees and shrubs, but the sky above is visible, with a few clouds.

The image captures a moment of traffic flow, with vehicles moving in both directions on the highway. The vehicles appear to be moving at a moderate speed, as indicated by the clear lane markings and the absence of any visible traffic signs. The road is wide enough for both vehicles to pass each other, and the median is clear and well-maintained.

The image is taken from a slightly elevated perspective, which gives a clear view of the road and the surrounding environment. The overall atmosphere of the image is calm and orderly, with no signs of any significant activity or disturbances.

Analysis and Description

The image is a clear depiction of a two-lane highway with a clear demarcation of lanes. The highway is divided into two parallel lanes, each separated by a median. The road is wide enough for both vehicles to pass each other, with no visible traffic signs or markings. The trees and shrubs on either side of the road are lush and green, indicating a healthy and well maintained environment.

The sign on the left side of the image is a warning sign, indicating that the road is not to be used by pedestrians or vehicles that are not authorized to pass each other. The sign is mounted on a metal post, and it is partially obscured by the trees and shrubs.

The image captures a moment of traffic flow, with vehicles moving in both directions on the highway. The road is wide enough for both vehicles to pass each other, with no visible traffic signs or markings. The trees and shrubs on either side of the road are lush and green, indicating a healthy and well-maintained environment.

Conclusion

The image depicts a two-lane highway with a clear demarcation of lanes. The highway is divided into two parallel lanes, each separated by a median. The road is wide enough for both vehicles to pass each other, with no visible traffic signs or markings. The trees and shrubs on either side of the road are lush and green, indicating a healthy and well-maintained environment.

This image captures a moment of traffic flow, with vehicles moving in both directions on the highway. The road is wide enough for both vehicles to pass each other, with no visible traffic signs or markings. The trees and shrubs on either side of the road are lush and green, indicating a healthy and well maintained environment.

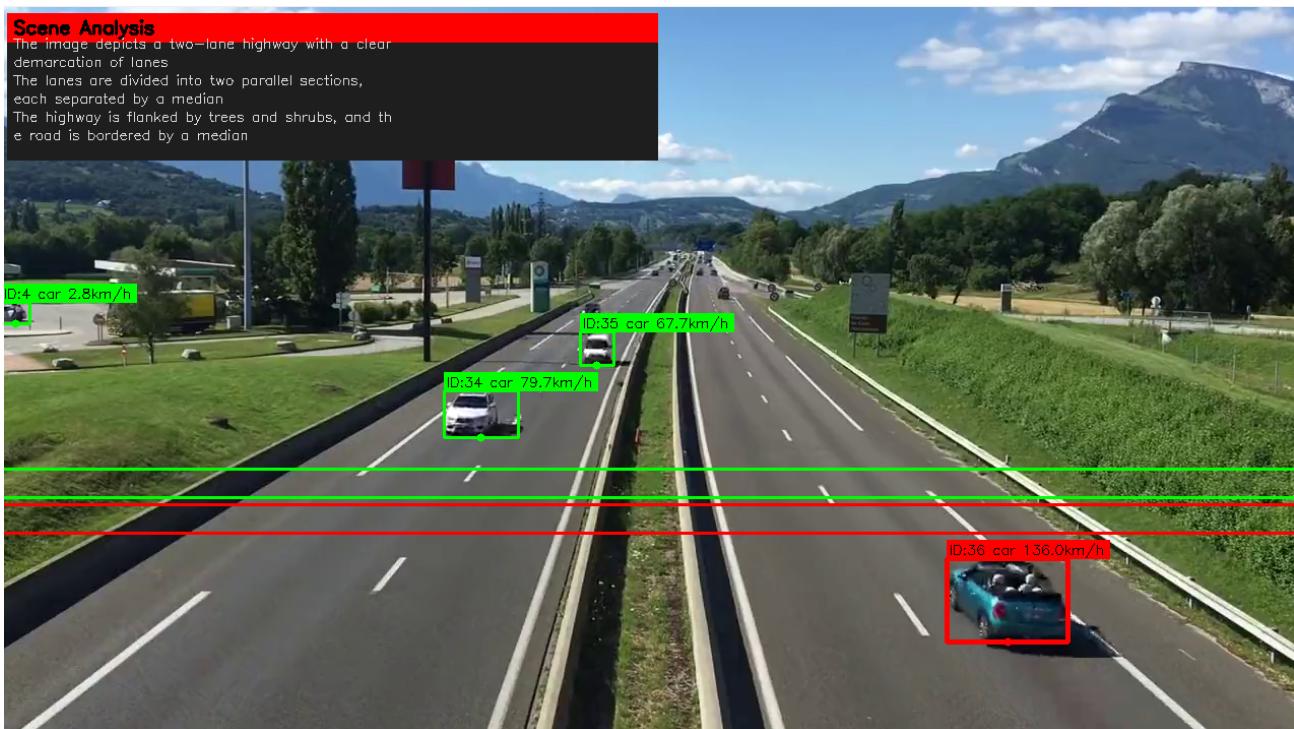


Figure 4.1: Frame used for scene understanding.

4.2 llava-v1.6-34b

This research oriented analysis is thought to leverage big VLMs for solving more complex tasks.

4.2.1 Anomaly Detection

Detecting automatically if an accident has occurred is a complex but extremely useful task for traffic monitoring systems. This way, we can send an emergency signal to the operators that can immediately assist the people involved.

In this task, I selected a video showing an accident, and the VLM is integrated in the system to supervision and detect anomalies in the traffic.

In Fig. 4.2 you can see the input frame of the video along with the VLM output.

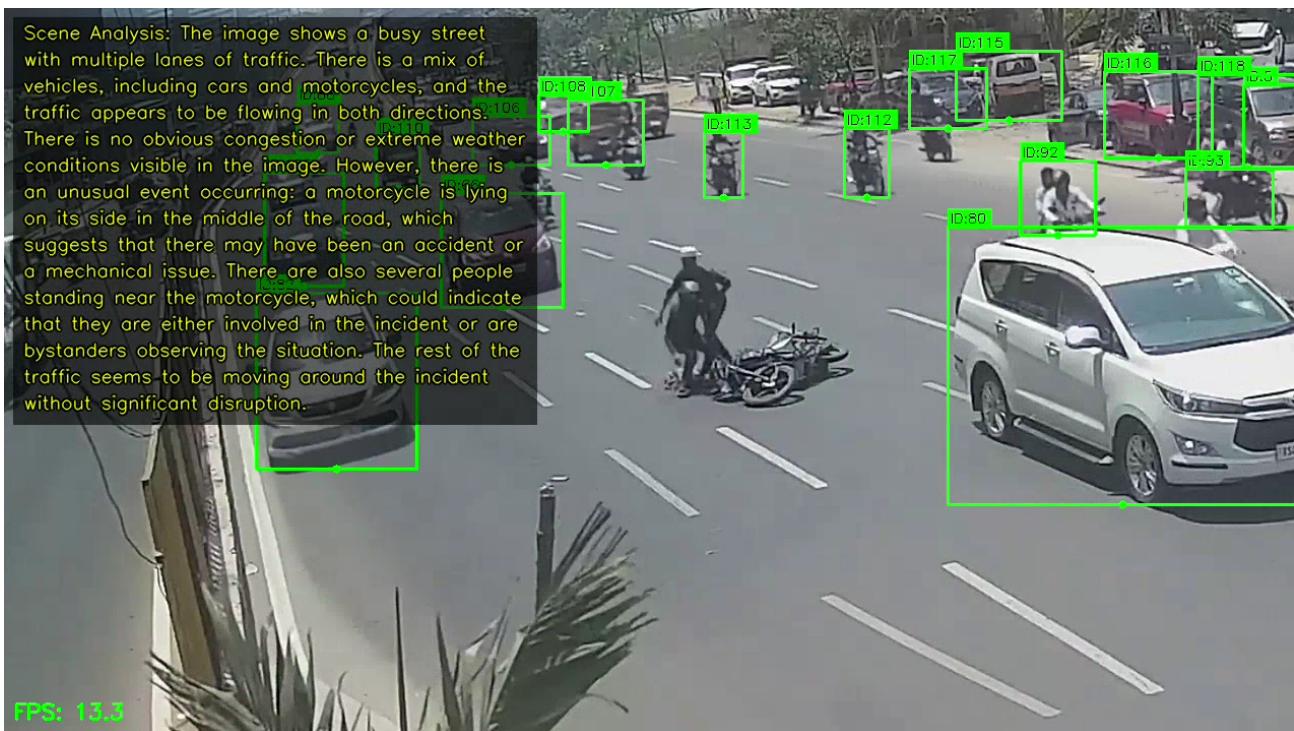


Figure 4.2: VLM as anomaly detector.

The VLM is able to correctly detect the motorbike laying on the ground, which is not a common situation in normal traffic environments, so, it understands that this anomaly could be the consequence of an accident. The VLM can be programmed to also activate the emergency signal so that the operators can intervene at the right location.

4.2.2 Automatic Homography Calibration

As described in previous chapter, in order to estimate real-world speed of vehicles in the image, we need to convert pixel coordinates to ground-plane ones. To do that, I designed an interactive tool that lets you select 4 points in the image (of a reference object) and insert the measurements in meters, so that it can build the homography.

This work can be expensive when calibrating cameras in different locations, so in this chapter I show how to design an automatic pipeline with VLMs, that automatically find reference objects and provide the homography ready to plug in for speed estimation.

Let's see if their output is accurate.

The reference image is showed in Fig. 4.3.

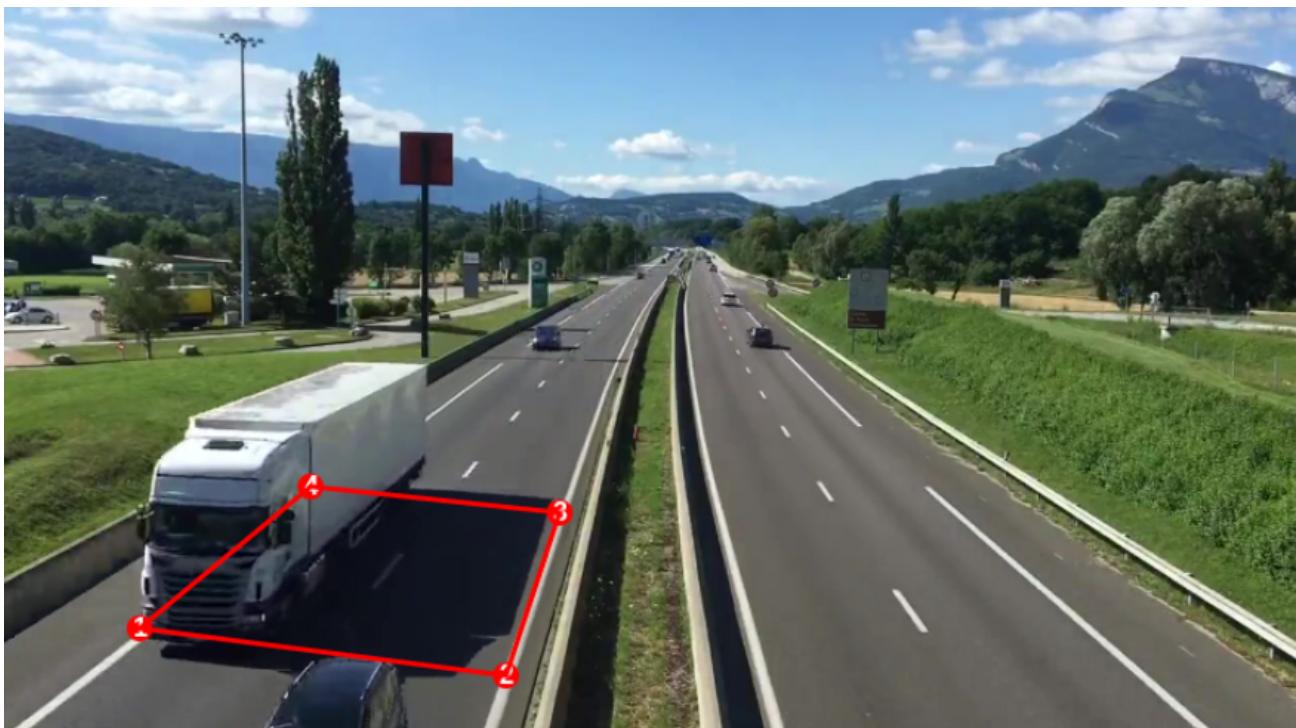


Figure 4.3: Reference image for automatic camera calibration.

We prompt the VLM with the image and the following text:

```

prompt = (
    f"You are a meticulous AI assistant specializing in photogrammetry and traffic camera"
    " calibration."
    f"You will be given an image from a traffic camera with dimensions"
    " {image_width}x{image_height} pixels."
    f"Your task is to generate a JSON object for homography calibration. Follow a strict"
    " chain-of-thought process before providing the final JSON.\n\n"
    f"Step 1: Object Identification. Verbally describe the best rectangular, co-planar"
    " object on the road surface. A good example is a single, complete white lane"
    " marking (the painted rectangle, not the entire lane). \n\n"
    f"Step 2: Real-World Size Estimation. Verbally explain your reasoning for the"
    " real-world 'width' and 'length' of this object in meters. A standard highway"
)

```

```

    ↪ lane_marking_in_many_regions_is_3_meters_long_and_0.2_meters_wide. Use this
    ↪ knowledge_to_estimate_the_dimensions.\n\n"
f"Step 3: Pixel Coordinate Identification. Verbally list the four pixel coordinates
    ↪ '[x,y]' for the corners of the rectangle you identified. CRITICAL_INSTRUCTION:
    ↪ These MUST be integer pixel coordinates within the image bounds of
    ↪ {image_width}x{image_height}. Do NOT use normalized coordinates between 0 and
    ↪ 1.\n\n"
f"Step 4: Construct World Coordinates. Based on your estimated 'width' and 'length'
    ↪ from Step 2, create the 'world_points' array, starting the first point at the
    ↪ origin '[0.0,0.0]'.\n\n"
f"Step 5: Assemble the Final JSON. Finally, combine the 'image_points' and
    ↪ 'world_points' into a single JSON object. The JSON must be enclosed in triple
    ↪ backticks(````).\n\n"
"Let's think step by step. Begin your analysis now."
)

```

The output of the VLM is the following for Fig. 4.3:

Step 1: Object Identification

The best rectangular, co-planar object on the road surface is the white lane marking. It
 ↪ appears to be a standard highway lane marking, which is typically 3 meters long and
 ↪ 0.2 meters wide.

Step 2: Real-World Size Estimation

Given the standard dimensions of a highway lane marking, we can estimate that the lane
 ↪ marking in the image is approximately 3 meters long and 0.2 meters wide.

Step 3: Pixel Coordinate Identification

The pixel coordinates for the corners of the rectangle are:

```
[0.131,0.419,0.373,0.673]
[0.403,0.419,0.550,0.673]
[0.658,0.419,0.802,0.673]
[0.797,0.419,0.945,0.673]
```

Step 4: Construct World Coordinates

The world coordinates for the corners of the rectangle are:

```
[0.0, 0.0]
[3.0, 0.0]
[3.0, 0.2]
[0.0, 0.2]
```

Step 5: Assemble the Final JSON

```
json
{
    "image_points": [
        [0.131,0.419,0.373,0.673],
        [0.403,0.419,0.550,0.673],
        [0.658,0.419,0.802,0.673],
```

```
[0.797,0.419,0.945,0.673]
],
"world_points": [
[0.0, 0.0],
[3.0, 0.0],
[3.0, 0.2],
[0.0, 0.2]
]
}
```

This JSON object can now be used for homography calibration.

It clearly does not make any sense.

Even if it provides an excellent High-Level Reasoning, for example in step 1, where the model correctly identified the "white lane divider" as the best object for calibration. This is a great choice (even if it didn't detect the truck).

In step 2, its reasoning that highway lanes are about 3.7 meters wide is excellent. It correctly used real-world knowledge to inform its task.

This shows that our Chain-of-Thought prompt is successfully forcing the model to think about the problem in a structured way.

The Critical Problem is the Spatial Inaccuracy. The model completely failed at the most important part of the task: finding the actual pixel coordinates.

Moreover, I have asked for four [x,y] corner points, but the model gave me four bounding boxes ([x1, y1, x2, y2]). It has hallucinated an object detection task instead of a point identification task.

Even worse, it completely ignored the explicit, critical instruction to use integer pixel coordinates. It produced normalized coordinates (values between 0.0 and 1.0).

This makes the image_points data completely unusable for homography. This is a common limitation of many current VLMs—they are great at saying what is in an image, but terrible at saying precisely where it is in pixels.

To solve this, specific fine-tuning aiming at improving spatial awareness of VLMs is necessary to make this tool useful.