Multi-Vision System for Autonomous Driving

Daniel Rossi
239210@studenti.unimore.it

Riccardo Salami
243999@studenti.unimore.it

Filippo Ferrari
255914@studenti.unimore.it

October 21, 2022

Here we would like to introduce you to an ADAS (Advanced Driver Assistance Systems) that encompasses some of the latest Computer Vision and Deep Vision techniques and existing technologies suitable for driving support. We will present in detail our implementation of object detection within the road context, roadway recognition and speed limit traffic sign detection and classification.

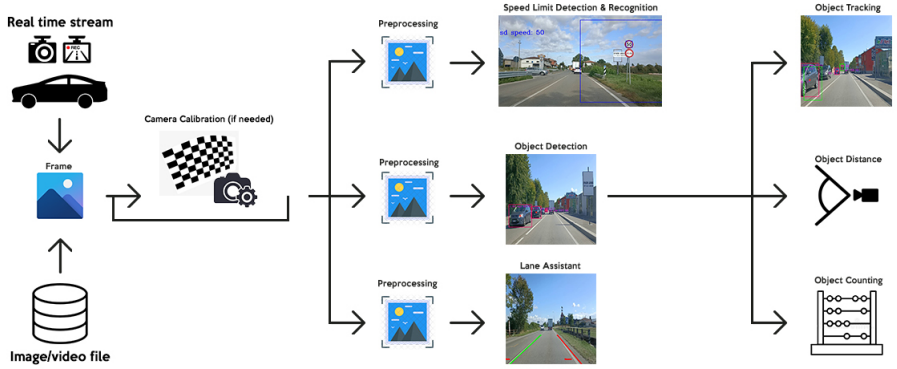


Figure 1: System's inference pipeline

1 Introduction

Our system takes as input single frames taken from a video, performs detection and recognition with 17 different classes, like cars, bicycles, pedestrian and so on, then the bounding boxes for each one of them, then performs detection and recognition of speed limits traffic signs, displaying the current speed limit in the frame.

The automotive domain has particular constraints on the available hardware on-board, so one cannot build a heavy and high power demanding system and then deploy it on a car. For this reason, we needed the fastest possible solution, so even though there are tons of Object Detection NNs working really well nowadays, we decided to choose Yolo because, as the name implies (i.e. "You Only Look Once"), it's an architecture built with the purpose of going as fast as possible, and for that reason we decided to use its latest implementation, which is the fastest and the best in terms of accuracy: Yolov7. The newest YOLO algorithm surpasses all previous object detection models and YOLO versions in both speed and accuracy. Since its fast and rather lightweight, it tipically requires less powerful hardware than its competitors. Since, as we said earlier, speed was our main concern, we coupled the Object Detection part with a more classical one, using famous Computer Vision algorithm that still prove to be rather good, even in the deep learning era.

2 Datasets

2.1 COCO

COCO[3], huge and well-known dataset developed by Microsoft, is the dataset with which YOLOv7 has been trained on by the authors of the paper: it contains around 330.000 images and 80 classes for object detection. Thanks to this dataset, YOLOv7 was able to achieve SOTA performances, wrt both speed and accuracy. Of course, not all classes were important to us (like cows or ties), so we kept the 17 classes that we needed and threw away the rest.

2.2 BDD100K

BDD100K[9], as the name suggests, contains 100.000 images labeled for object detection tasks. The street context images were captured inside the car's cabin, a questionable choice, nevertheless they allowed us to fine tune YOLOv7[2]'s detection layer. The flaw in these images, due to the way they were captured, is that within the image were parts of the car used by the person who captured them. We still chose to use this dataset because we also mount the camera inside the cockpit, but without framing parts of the car.
BDD100K can be used for different tasks: we used it by taking advantage of the object detection labels, which include 10 classes: pedestrian, rider, car, truck, bus, train, motorcycle, bicycle, traffic light, traffic sign.

2.3 ItalianSigns

Regarding the detection and classification of speed limit road signs, we created our own dataset, consisting of 361 labeled images, extracted from more than 50 video recordings in HD format (1280x720). These images include various luminance conditions (morning, afternoon, evening and night), they are stored in JPEG format with a compression level between 30 and 60 percent, and come from various areas within the province of Reggio Emilia.

Since we opted to use Classical Computer Vision approaches to address this task, it would have been sufficient to use a dataset with a number of images in the range of hundreds to test the algorithm; notwithstanding, the available datasets on the internet, such as the German Traffic Signs Recognition Benchmark and the Data set of Italian Traffic Signs didn't meet our demands. This is the main reason why we created ItalianSigns.

3 Approach

3.1 Classical Techniques

3.1.1 Camera Calibration

Because the wide-angle lens of the camera we use creates a barrel distortion, it was necessary to correct the image through camera calibration. Calibrating the camera is necessary because a distortion of the image causes structural changes in the objects within it, leading the elements in our pipeline to behave in not expected ways.
For the calibration task, we used Zhang's[10] technique: we printed a chessboard and we acquired dozens of images of it, in order to estimate the camera parameters and obtain undistorted images.

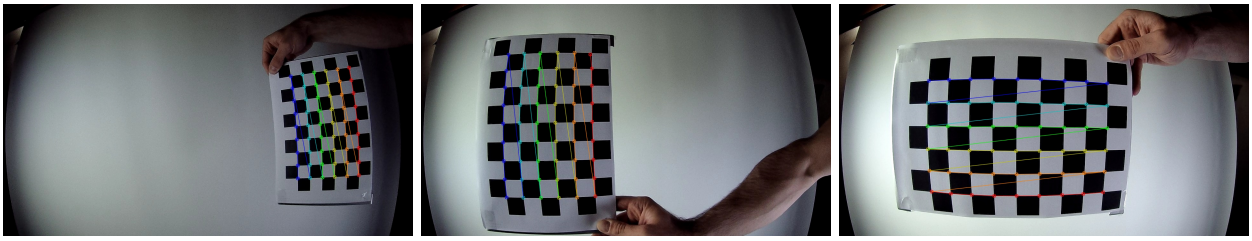


Figure 2: camera calibratin performed on our See3CAM_CU27

3.1.2 Speed Limit Detection and Classification

Since we are looking for traffic signs indicating the maximum speed, and they are most of the times located to the right with respect to the car, the first thing we do is to throw away a part of the image by setting 3 cut percentage parameters: 5% from the top, 5% from the bottom and 50% from the left left part of the image. This choice allows us to increase the execution performance and accuracy of the algorithm which will work on a smaller region of the image.

Regarding the location of the sign, our technique is based first on a preprocessing aimed at making the sign stand out from the background and then finding it within the image due to its circular shape.
To separate the road sign from the background we exploit a feature common to all speed limit signs, namely the red border. To perform this color separation we go to convert the image from RGB to HSV format, and then apply two masks to go and isolate the red color and its shades. Since the hue values of OpenCV's HSV space lie in the range [0, 180], we use a first mask with hue between [0, 10] and a second mask with hue between [160, 179] to select the shades of red.

Once we derive these two masks we add them together, and then we use the resulting mask to remove the background from the original image. Specifically we are going to perform a bitwise and between the HSV image

and the binary mask. We then go on to do the same thing on the original image, masking the red and removing the background. By eliminating the background we mean that everything that is not red color will become black.

At this point we extract the red channel from the masked RGB image, to be used as a gray image to be provided as input to a thresholding that exploits OTSU[6] between the values [200, 255]. To the result we apply a closing and dilation to remove some of the noise present in the binary frame. We call this frame "X" for simplicity, which we will go on to use in the next step. Now the algorithm goes to extract a new gray frame by taking the red channel again from the masked RGB image. To this gray image it then goes to perform a bitwise and using as a mask the frame "X" that we obtained in the previous step. The frame we obtain will be partially black and will partially contain pixel regions of the original image, where most likely one includes the sign (usually the largest).

In fact now the algorithm proceeds by going for contours within the image, i.e. it identifies regions, and then picks up the largest one. Once we possess the largest region, this most likely contains the sign and consequently we go on to crop this portion from the original frame.
Being able to work on a very small region of the original image therefore allows us to have a higher accuracy in locating the sign.

To find the exact location of the sign we take advantage of a second feature present in all road signs that are speed limits, namely the fact that they are circular. Now through Hough[7] we can find the exact location of the sign, and from the circle provided we go to derive the bounding box that encloses within it the sign itself.

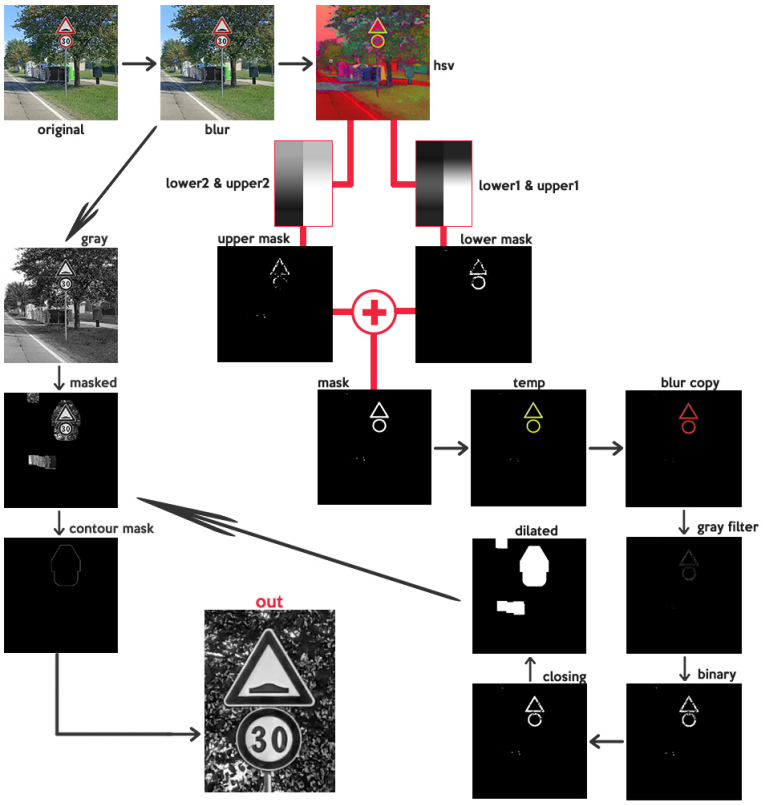


Figure 3: Traffic sign detection schema

Regarding now the classification of the road sign present in the preprocessed frame, the methodology proceeds as follows. We have collected a series of reference street sign images, specifically one for each speed limit that may be encountered on the street.
We then pass these images to SIFT[5], along with the newly acquired and preprocessed street image, to extract keypoints. Before running SIFT[5] on these images, we perform a resize, via the exact interpolation method, of them to obtain images with size (100x100). Images of the same sign but with different dimensions could then cause different keypoints to be extracted. In addition to being resized, the images that are provided as input to SIFT[5] are also binarized to minimize the possibility of errors.

Once the keypoints are extracted, we use the KNN[8] algorithm to classify the main image, which gives us scores on the closeness of the keypoints of the acquired image with the keypoints of the reference image. To improve accuracy during classification, we go on to add two constraints on the results we get from KNN[8] matching: the output is valid only if the best result is greater than or equal to 125% of the second best result and if it is greater

than a threshold k. For the classification part, other methods were considered and tested againist Sift: first of all we tried with a simple Template Matching, but it only worked when the traffic signs were in a perfect position, in fact even a slightly inclined traffic sign could mean a completely different result, and using Extended Template matching was utterly slow. We also tried using ORB, since it is a Keypoints Detection algorithm like SIFT[5], but fairly faster than the latter, however ORB detected a much larger number of keypoints w.r.t. SIFT[5], many of them located not on corners but on edges, so we had more false positives than with SIFT[5], also the speed were comparable since, although ORB is faster than SIFT[5] in computing the keypoints, since it detected many more keypoints than SIFT[5], the comparison with the KNN was also slower, negating the only advantage that ORB had on SIFT[5].

The algorithm has 66% accuracy in sign recognition, with 63% accuracy in finding the bounding box of the sign (66% precision and 93% recall), and a mAP of 5.42. The average execution time is 44.15ms

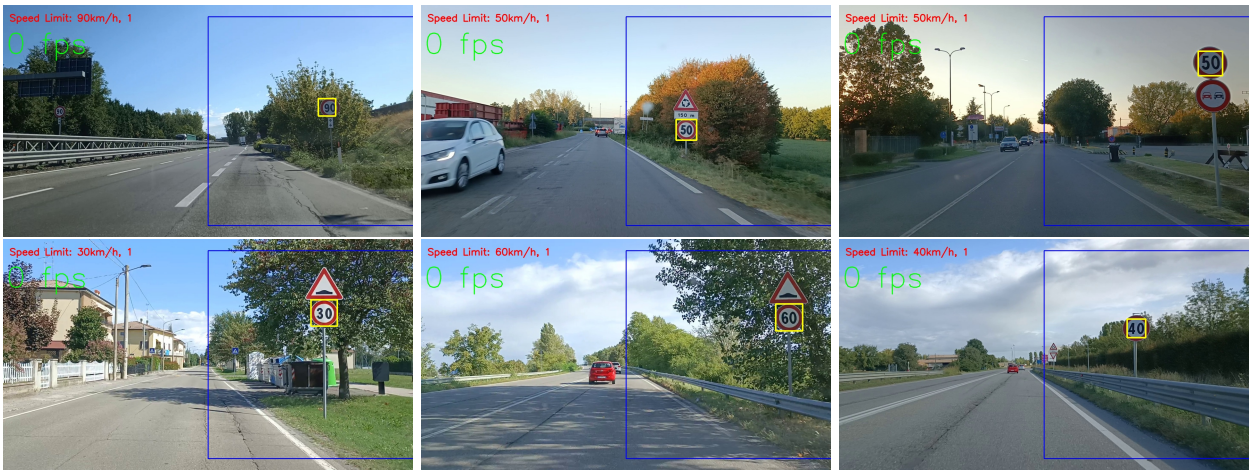


Figure 4: Traffic sign detection and classification done by our technique

3.1.3 Adaptive Lane Assistant

Much like with the previous paragraph, we were inspired by another ADAS of current modern cars, so we decided to implement another one, and that is the Lane Assistant, a useful software that is able to detect the lines at the sides of the road and notify the driver if he's crossing either the left or the right one, preventing him from endangering him and the other passengers.
The trickiest problem for this part was surely the removal of noise: especially in the country-side, grass often covers part of the road, including the side lanes, introducing noise when computing the edges, also very high level of luminance, like in sunny days in summer, highlight all the imperfections on the roads, especially cracks and broken parts; moreover, if the road is old all the imperfections are emphasized, with the additional problem of the lanes being fading or already gone because of time and weather.
The problem is that if there is a lot of noise, a lot of smoothing is needed in order to have a clean computation of the edges, but smoothing too much often results in lanes so faded and blurred that the edge detection fails to detect them correctly. For this reason, we decided to use the bilateral filter instead of the Gaussian one, since in this way we smooth a lot of the noise maintaining at the same time a somewhat clear separation of the lanes from the road, enabling us to capture that separation with edge detection. First, we remove the grass from the image, this removal is again done using HSV color-space. Then, we convert the input frame to Grayscale and compute the mean level of luminance for the whole image, this lets us know how much luminance there is, and for its lower part, in this way we know and the quality of the road.
Then, based on the mean luminance of the entire image, we decide whether to enable Equalization or not and the parameters to use for the bilateral filter, like the size of the kernel; we created 11 different configurations based on the luminance level in order not to smooth too much at night and too low when there is a lot of natural light.
Then, based on the luminance of the lower part of the image, we set the parameters for the Canny[1] edge detector accordingly. When we have computed the edges of the smoothed image, we apply a bit-wise and mask over the image to make sure that we only include the road, effectively eliminating many of the edges that could prevent us to properly the detect the lanes.
Finally, we look for possible lines through the Hough transform by splitting the frame in two halves, and looking for lines with particular theta on the left and on the right halves and return the resulting lines.
When drawing the lines on the frame, we look for the theta values of both lines: if the driver is safely inside the lanes, they are both green, otherwise if he is about to cross one of them, it instantly becomes red and a warning message is displayed.

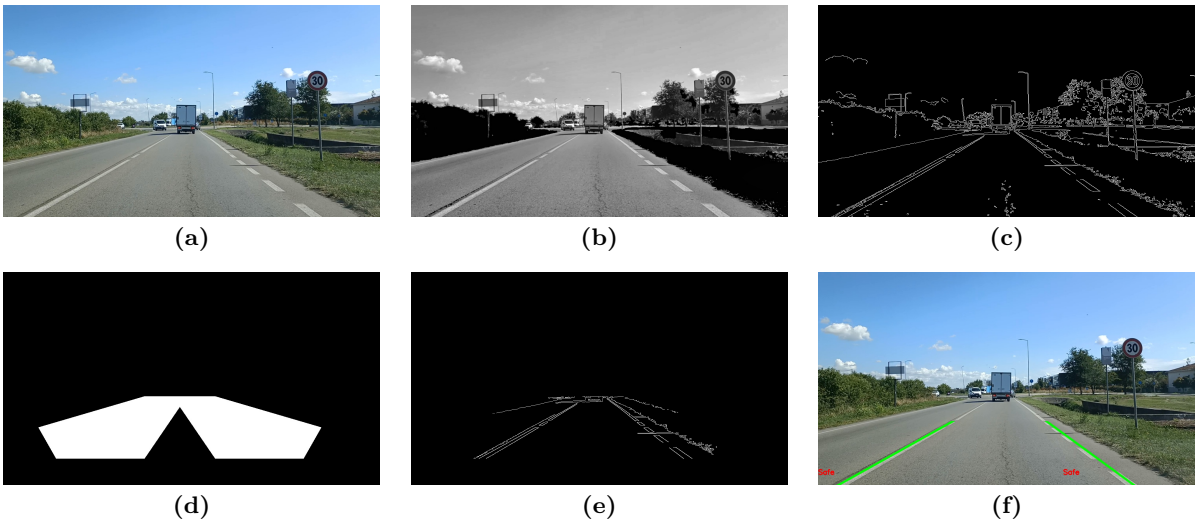


Figure 5: **(a)** Input RGB image **(b)** Image after preprocessing (grass removal, Bilateral Filter and also Equalization in this case) **(c)** canny Edge Detection **(d)** Optical bit-wise AND mask **(e)** Edges after the application of the mask **(f)** Result after lines computation with Hough Lines algorithm

3.2 Deep Learning Techniques

3.2.1 YOLOv7[2] Object Detection

To tackle the object detection task, we decided to use YOLOv7[2], the latest version of this object detection technique. Compared with its previous versions, YOLOv7[2] has major improvements from an accuracy, performance and model size perspective. In particular, mainly because of the small model size compared to other neural networks for object detection, and considering that the recognition task has to be performed on a system working in real-time, we thought that YOLOv7 was also the appropriate neural network to perform this task.
Our work done on this neural network focuses mainly on integrating the BDD100K[9] dataset, with the goal of fine tuning the it on latter. In fact, we created a class for managing the BDD100K[9] dataset that was structurally consistent with the one created by the original YOLOv7[2] authors used to manage the coco dataset. Next, it was necessary to implement our training and testing scripts for the neural network. Finally, we implemented an API that would allow us to do neural network inference on a single image, so we could have direct access to the output of the neural network and be able to integrate it with our script for real-time inference.

YOLOv7[2] was fine-tuned on images of size (640, 640), to which image augmentation filters, such as: colorJitter, fog, snow, rain, blur, etc., were applied. The training was carried out for a length of 50 epochs with batch-size of 160 images, on a server that included 5 NVIDIA RTX 3090 GPUs. The average duration of an epoch was 5 minutes and 20 seconds, and the training was completed in 4 hours and 20 minutes.

Training launch command: torchrun —nnodes=1 —nproc_per_node=5 custom_train.py —adam —batch-size 120 —device '0,1,2,3,4' —epochs 50 –multi-gpu 1 —name 'finetuning' —noautoanchor —save_period 1 —stride 32 —workers 32 —task 'train' —weights ' /last.pt' –hyp ' /hyp.yaml'

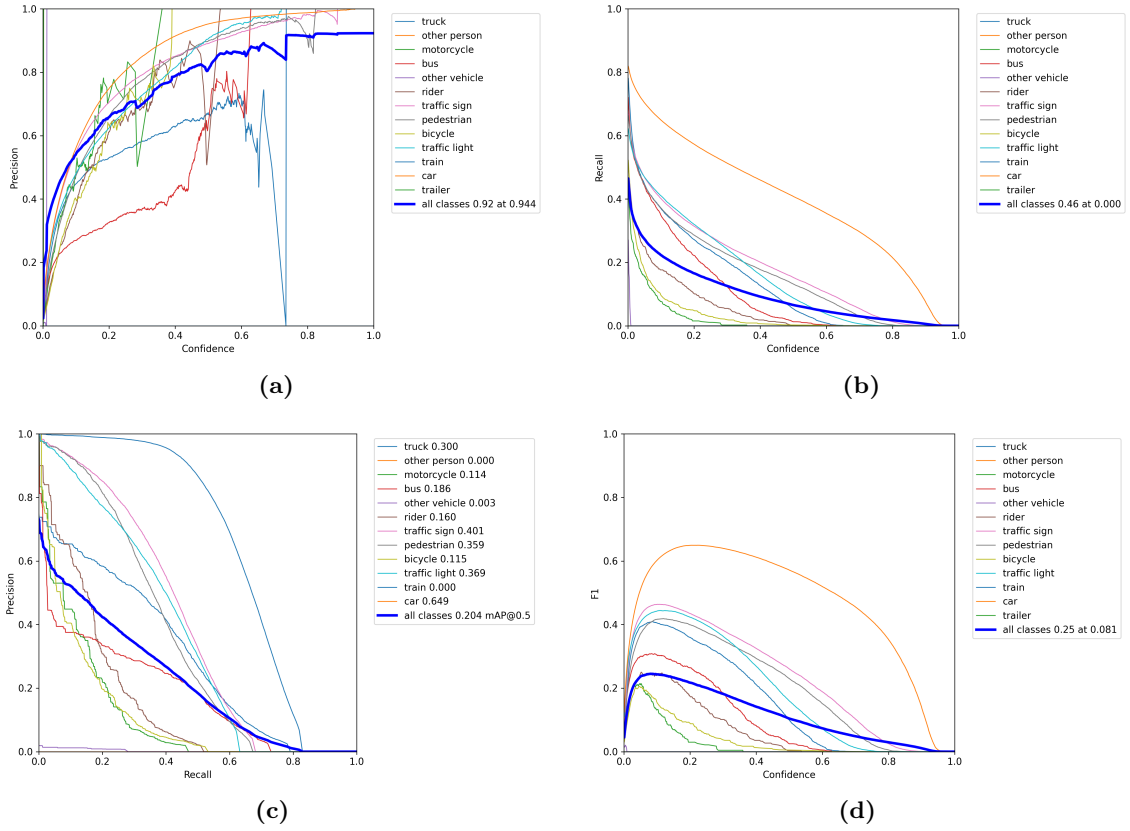
(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 6: **(a)** Batch 0 labels, **(b)** Batch 0 predictions, **(c)** Batch 1 labels, **(d)** Batch 1 predictions, **(e)** Batch 2 labels, **(f)** Batch 2 predictions.

We stopped fine-tuning at 50 epochs because we felt that the object recognition and classification performance was sufficient for the demonstration purpose of this project. Supporting this decision is the fact that during training each GPU consumed an average of 270W, for a total of 1350W of power consumed over 4.20 hours of training. From the graphs in [Figure 7] we can see that the model would need further fine tuning to improve its performance; however, on the other hand, we note that the car recognition statistics are satisfactory. This could be a consequent symptom that the dataset is unbalanced from an object point of view.

Finally, we want to conclude this section by making a point about the inference speed of this neural network. As also pointed out on the original YOLOv7 paper, the basic model of this neural network is capable of reaching up to about 160fps with batch size of size 1, in fact during the tests we performed on the samples we acquired on the way, this neural network always allowed us to work at least at 30fps.



(a) **(b)**

(c) **(d)**

Figure 7: **(a)** Precision curve, **(b)** Recall curve, **(c)** Precision over Recall curve, **(d)** F1 curve.

3.3 Hardware employed

To conduct tests and with the purpose of acquiring data to create the ItalianSigns dataset, we designed and built special mounts to house our cameras. Data were acquired both through the smartphone camera to which we connected via webserver, and through econ systems' See3CAM_CU27 camera, a night vision camera with a Sony Starvis sensor capable of providing us with a peak framerate of 100fps up to full-HD resolution.

Figure 8: See3CAM_CU27 Full HD Sony^{R} Starvis^{TM} IMX462 Ultra Low Light USB 3.1 Gen 1 Camera

To facilitate camera integration on the car, we purchased a standard GoPro mount, and designed two custom mounts for smartphones and the camera. The mounts were designed via FreeCAD and then 3D printed using PLA as the printing material.

Figure 9: These are our custom supports for both camera and smartphone

4 Future Developments

4.1 Tracking

Our pipeline then continues and once we are provided with outputs from YOLOv7[2], from those we perform bounding box tracking of a specific object. Starting with the bounding box which describe where the object is located, we carry out the tracking of it using Kalman[4] filters. These filters tend to produce increasingly accurate estimates of the object's future location as we add information about the trajectory it has traveled so far.

We initialized the filter with a state size of 4, and a measurement size of 2, proceeding to create the measurement matrix, the transition matrix, and the matrix used to process the noise. The Kalman[4] filter at work then takes advantage of the current bounding box and the ROI in HSV format to update the model that it exploits to perform the estimation and to subsequently provide us with a prediction of future position.
During testing, we noticed that the Kalman Filters produce an accurate prediction of the future position of the object only when the bounding box is large enough.

Since, particularly after being instantiated, the Kalman[4] filter produces an unreliable prediction, among future developments we consider including a system to evaluate the accuracy and reliability of the trajectory prediction. One solution would be to calculate the IOU between the current bounding box and the Kalman[4] filter prediction, and define a threshold. Supporting this solution is the fact that an object, particularly in the road context, is unlikely to move at such a high speed as to provide IOU=0 between bounding box and prediction made by tracking.

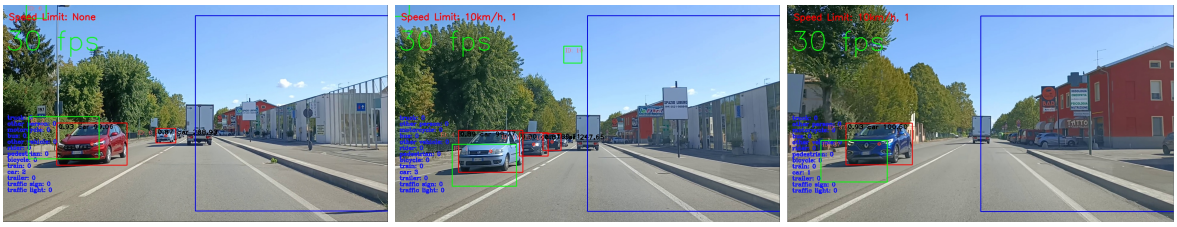


Figure 10: Experiment on multiple object tracking with Kalman Filters (green box)

4.2 Distance measurement of an Object

Another feature we wanted to add to the pipeline, which we believe is critical for an ADAS system, is the measurement of the distance to a recognized object within the frame. This is particularly useful for the implementation of driving safety systems. To calculate the distance we use a very simple procedure: we have a reference frame in which we recognize an object, whose size and distance from the camera we know. From this data we are able to extract the focal lenght, which we then use to calculate the distance to a generic object. This method is rather inaccurate and needs appropriate calibration according to the camera being used.
To improve the distance calculation technique it would be sufficient to use a stereo vision camera, which unfortunately we were unable to add to the project for mainly economic reasons. Having another camera would also allow us to develop a Brake Assistant System, based on the speed with which the distance of the vehicle in front of us changed in the last k frames.

5 Conclusions

In this paper we propose a computer vision pipeline composed by both state of the art neural network and more classical computer vision approaches. At the end, the results are satisfying for all the tasks, and promising for the tasks currently under development. In particular, we were able to show that the combination of the powerful capabilities of current state of the art neural networks with simplier, yet lightweight, more classical approaches, can give birth to a more complete, lighter and faster system, specially when there are constraints on the hardware, like in the Autonomous Driving contest. Our code is available at this link:
https://github.com/ProjectoOfficial/ComputerVisionProject

In conclusion, we would like to thank e-con Systems for providing us with the complimentary hrefhttps://www.e-consystems.com/usb-cameras/sony-starvis-imx462-ultra-low-light-camera.aspSee3CAM_CU27 camera that we used to carry out the road tests, and we would like to thank LeaderGPU for providing us with the server equipped with 5 RTX3090s for a total of 15 hours leveraged for fine tuning YOLOv7.

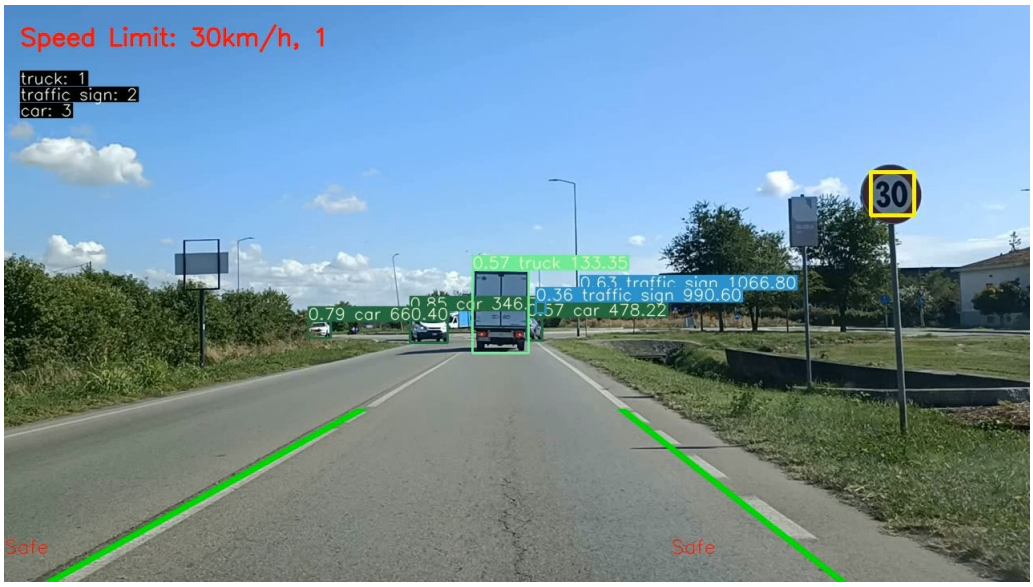


Figure 11: Final inference result with object detection, lane assistant and sign detection

References

[1] J. Canny. "A Computational Approach to Edge Detection." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[2] Alexey Bochkovskiy Chien-Yao Wang and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* URL: https://arxiv.org/abs/2207.02696. (v1: 07.06.2022).

[3] Piotr Dollar. *Microsoft COCO: Common Objects in Context.* URL: https://arxiv.org/pdf/1405.0312.pdf. (v3: 02.21.2015).

[4] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems." In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[5] D.G. Lowe. "Object recognition from local scale-invariant features." In: *Proceedings of the seventh IEEE international conference on computer vision* 2 (1999), pp. 1150–1157. DOI: 10.1109/ICCV.1999.790410.

[6] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms." In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[7] P.E. Hart R.O. Duda. "Use of the Hough transformation to detect lines and curves in pictures." In: *Communications of the ACM* 15.1 (1972), pp. 11–15. DOI: 10.1145/361237.361242.

[8] P. Hart T. Cover. "Nearest neighbor pattern classification." In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[9] Fisher Yu. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning.* URL: https://arxiv.org/pdf/1805.04687.pdf. (v2: 04.08.2020).

[10] Z. Zhang. "A flexible new technique for camera calibration." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.