Visual Vehicle Detecting and Tracking using YOLOv3 and Kalman Filter

Zikun Zhou Yunzhong Li Hongjun Liu Chubo Zhou Zhiyu Yuan
University of Michigan
Ann Arbor, MI
kennyzzk@umich.edu yunzholi@umich.edu hongjunl@umich.edu zhouchu@umich.edu yuanzhiy@umich.edu

1. Introduction

The problem we are trying to solve is the multi-object tracking (MOT) for vehicles on the road. It is used in the development of autonomous vehicles for simultaneous localization and mapping, as well as planning the most optimal path to ensure a safe and robust driving experience in a daily basis. To provide a solution to this system, we need to use a combination of detector and tracker algorithms.

There are plenty of methods available with different properties to find the optimal and efficient solution for the problem. Some of them have achieved better results than others. For example, ByteTrack provides high-performing a algorithm with high enough accuracy. [13] However, considering the efficiency and limited computational resources in real-world scenario, the use of such algorithm that requires high-end computing power is not realistic. After taking deep research regarding different detection and classification algorithms, we finally created a robust solution balancing the efficiency and performance.

For tracking, we can use either SORT or DeepSort. We decided to use SORT, which uses a recursive filtering algorithm called Kalman Filter [1] to provide a geometric solution for tracking. By the nature of Kalman filter, it needs to be used on top of an accurate detection algorithm, which lifted obstacles and difficulties. However, we are able to solve this problem due to the existing number of high performing detection algorithms and classification neural network models in the modern CV industry that efficiently classify objects using deep convolutional networks, like YOLOv1-5 and Fast RCNN [3]. Additionally, we designed a new processing model for SORT's Kalman Filter to predict the location more accurately. We also implemented a greedy algorithm to predict the overall prediction of the car's trajectories. We choose to use YOLOv3 as our framework to train our own single-class object detector because it runs faster than FastRCNN algorithm despite being slightly less accurate. We find YOLOv3 is a perfect balanced fit for our problem on the mAP@.5 metric, the performance difference is negligible for human eyes [9]. We work on the data from the Boxy dataset, because it has a vast amount of high resolution and well-labeled dash cam footage to train our model [2].

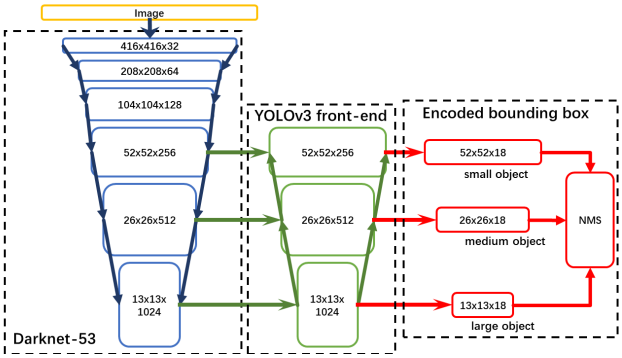


Figure 1. The architecture of YOLOv3. Blue part are the Darknet-53 feature extractor. Green part are the part of YOLOv3's front-end CNN. Red part are the output bounding boxes.

In general, we use a YOLOv3-based single-class detector to generate the bounding box and then use a high-efficiency greedy algorithm to match the new detection with the predicted bounding box from Kalman filter based on IoU metrics.

2. Approach

2.1. Object Detection

For the detector of the vehicles, we use the YOLOv3 as the framework [9]. Instead of directly borrowing the pre-trained network on the COCO dataset, we adjust its architecture to fit it with our single-class object detection task, as we only need to find the locations of vehicles. YOLOv3 uses the Darknet-53 as the backbone feature extractor and a front-end CNN that can output 3d tensors encoding the bounding box and the confidence in three different stages corresponding to large, medium, and small objects [6]. Then, it uses the non-max suppression to pick the best bounding box for each object.

Each stage in Fig. 1 is the output of several layers of convolution, batch normalization, sigmoid linear unit, and

bottleneck from the previous stage. The YOLOv3's architecture resembles a UNet, while it uses a Conv layer with stride instead of max-pooling to downsample.

We used a loss function similar to the one used in YOLOv2 [8]. However, as our network is a single-class object detector, the term corresponding to the classifier loss is removed from our loss function.

The first term of our loss function is the bounding box losses. For each proposed bounding box that contains the object, we sum up the MSE of its center and shape to the ground truth.

The second term is the objectness losses. For the best bounding box that overlaps the most with the ground truth, the loss is the squared difference between its confidence and 1. For other bounding boxes, the losses are the square of their confidence. However, those bounding boxes that are not best but good enough (overlap greater than 0.5) are ignored in the objectness losses.

Then, two losses are summed together with a coefficient λ_{bb}, the overall loss functions are:

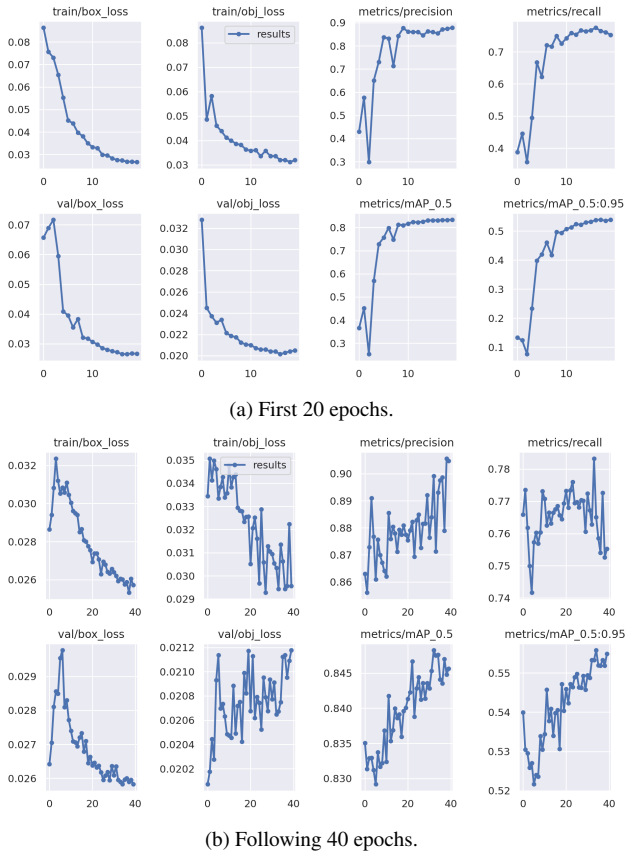
$$\begin{aligned} L =& \lambda_{bb} \sum_{i=0}^{B} 1_{obj}^{i} [(t_x^i - \hat{t}_x^i)^2 + (t_y^i - \hat{t}_y^i)^2 \\ &+ (t_w^i - \hat{t}_w^i)^2 + (t_h^i - \hat{t}_h^i)^2] \\ &+ \sum_{i=0}^{B} 1_{best}^{i} (1 - C^i)^2 + 1_{noobj}^{i} (C^i)^2 \end{aligned} \tag{1}$$

We start from the YOLOv3's pre-trained weight on the COCO dataset [5]. As the Boxy dataset has far too many images that are beyond our computation capability, we only pick one sequence 2016-09-27-14-43-04 as the training and validation set and another sequence 2016-10-26-12-49-56 as the test set [2]. Even the images in one sequence are still too much to run an epoch, so we pick images every 20 frames based on the assumption that adjacent frames will be similar.

We use stochastic gradient descent to train our model. We set the momentum to 0.937 and the weight decay to 0.0005. With batch size = 2, we first run the training with a learning rate of 0.01 for 20 epochs. Then we run another 40 epochs with a learning rate of 0.005 to further improve the performance.

2.2. Object Tracking

In general, object tracking consists of three main steps: 1) Get the initial set of object detection and create a unique ID of each initial detection. 2) Track each object as it moves through the frame and maintain its unique ID. 3) Create new IDs for additions and delete disappearing objects [10]. Here, we select SORT(Simple online and real-time tracking) tracking with Kalman Filter loop.



(a) First 20 epochs.

(b) Following 40 epochs.

Figure 2. The plot of loss, precision, recall and mAP of two trainings.

The method achieves accuracy comparable to state-of-the-art online trackers, despite using only a basic combination of familiar techniques (such as the Kalman filter and the Hungarian algorithm) for the tracking component. And Kalman filter has the following important features that tracking can benefit from: Prediction of object's future location; Correction of the prediction based on new measurements; Reduction of noise introduced by inaccurate detection; Facilitating the process of association of multiple objects to their tracks.

Figure 3 shows the block diagram of implemented SORT tracker together with Kalman Filter loop [12].

Tracker represents previous frame detection, and detection represents current frame detection. First, calculate the Intersection over Union matrix, representing the IoU value of the bounding box Current frame (detection) vs. previous bounding box frame (tracker). Hungarian algorithm applied to IoU matrix, finding matched and unmatched trackers and detection. After obtaining relevant vehicles, a Kalman filter loop is used to estimate the current position better than detection.

Kalman filter consists of two steps [4]: prediction and

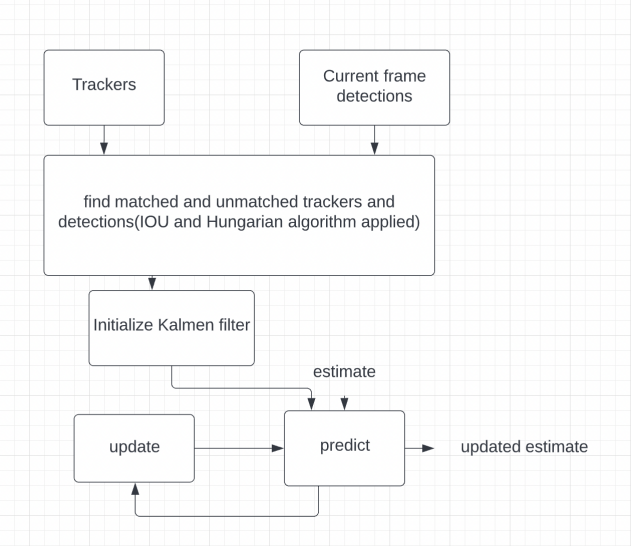


Figure 3. SORT Tracker block diagram and Kalman Filter loop.

update. The first step uses previous states to predict the current state. The second step uses the current measurement, such as detection bounding box location , to correct the state.

We apply constant velocity model of Kalman Filter. The state of our model is defined as:

$$X = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 & \dot{x_1} & \dot{y_1} & \dot{x_2} & \dot{y_2} \end{bmatrix} \quad (2)$$

where x_1 and y_1 is the coordinate of the top-left corner of the bounding box, x_2 and y_2 is the coordinate of the bottom-right corner of the bounding box. $\dot{x}_1$, $\dot{y}_1$, $\dot{x}_2$, $\dot{y}_2$ are four velocities respectively.

The measurement matrix H can thus be written as follows

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3)$$

We assume a constant velocity model, thus the state transition matrix F can be written as follows

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

Figure 4. One of the output of our network from the testset.

Model	Precision	Recall	mAP@.5	mAP@[.5:.95]
YOLOv3	0.674	0.662	0.668	0.465
Ours	0.901	0.778	0.847	0.597

Table 1. Our network is much better than its baseline.

where Δt represents time interval between two frame, which is assigned dynamically based on the timestamp of the frames.

3. Experiment

3.1. Object Detection

As mentioned in Sec. 2, we test our trained model on a different sequence from the Boxy dataset to avoid overfitting. From our observation, it works quite well. It can detect the vehicle that is almost occluded or the vehicle that is as shown in Fig. 4.

However, by just looking at the output, it is hard to tell how much better our model is comparing to the baseline, the YOLOv3's pre-trained model on the COCO dataset. We use precision, recall, average precision of IoU > 0.5 and the mean average precision of IoU > 0.5 to > 0.95 as four quantitative metrics to evaluate the performance.

From Tab. 1, we can see that our trained network beats its baseline in all 4 metrics. We attribute this improvement to the single-class detection model. The baseline's detection of vehicles may be interfered by objects in other classes while our network focuses on its only task

3.2. Object Tracking

The following is a demo output of our proposed tracking process. We attached two screenshots from our live demo video, which clearly indicates our box tracking trajectory.

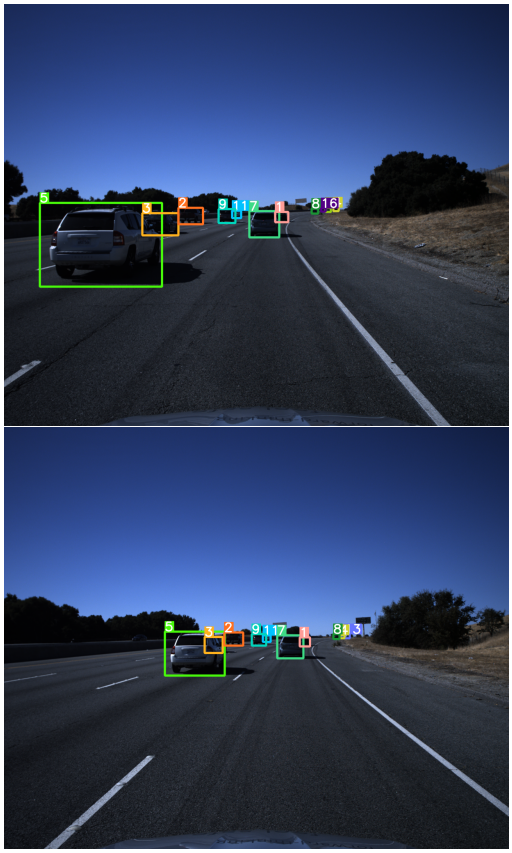


Figure 5. These two frames are 30 frames apart, the tracker successfully keeps tracking for all nearby vehicles and some of the faraway vehicles.

Qualitatively Speaking, the performance of tracking nearby vehicles are satisfying through naked eye observation. However, due to the limitations of the Kalman filter and IoU matching method, the tracking of vehicles in distance is less robust to the disturbance caused by the bumping of camera's carrier vehicle. Another disadvantage of using only IoU as the matching metric is that when multiple vehicles intersect and overlap, the labels may be interchanged.

As shown in Fig. 5, Although our tracker does not track vehicles faraway as accurately as those nearby, however, even for the human, these vehicles in distance are difficult to accurately distinguish and track. Also, these faraway vehicles are less important in Visual SLAM and path planning and can be filtered out in post-processing. Therefore the performance of our tracker is acceptable.

4. Implementation

4.1. Object Detection

We borrow the utility and model function from YOLOv3's github repository for training and evaluating also their pre-trained model as the baseline and the startup [11]. We write the dataloader of the Boxy dataset and the helper function for integrated test with the tracker.

4.2. Object Tracking

Firstly, we import Kalmen filter from filterpy library. Along with it, We implement IoU matching, Hungarian algorithm to match bounding boxes from our trackers and detection , which provides essential functionality to our tracking system. Additionally, for the Kalmen filter loop, we modified and enhanced the code to integrate with our detection results based on this GitHub repository of trajectory estimation.

Secondly, we replace Hungarian algorithm with a greedy match method, and redesign the processing model of the Kalmen filter to make it simpler and for possible better tracking performance.

5. Result and Conclusion

We provide a simple yet balanced solution which combines efficiency and performance to solve the multi object detecting and tracking problem. We focus on single-class MOT for vehicles on road especially. The main contribution can be summarized in three main point: adjusting the model architecture to best fitting our single-class object detection task, designing a new processing model for the SORT's Kalman filter, implemented a greedy algorithm to better matching the car's bounding box frame by frame, .

We utilize YOLOv3's pretrained model on COCO dataset to evaluate our object detection model, for which the result of our model surpasses the expected proposed initial baseline model. Aside from successful detection rate, we construct IoU matching and Hungarian algorithm which allows us to deliver a live demo which allows the camera to keep track of the car's moving trajectory. Our model provides balanced efficiency with satisfactory accuracy and stability, which we hope our fast yet effective system make it attractive in real-world application in autonomous driving field.

References

[1] https://www.cs.montana.edu/techreports/1314/Le.pdf. Accessed: 2022-4-25. 1

[2] Karsten Behrendt. Boxy vehicle detection in large images. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019. 1, 2

[3] Ross Girshick. Fast R-CNN. *arXiv [cs.CV]*, 2015. 1

[4] Lindsay Kleeman. Understanding and applying kalman filtering. https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/kalman/kleeman_understanding_kalman.pdf. Accessed: 2022-4-26. 2

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755. Springer International Publishing, Cham, 2014. 2

[6] Joseph Redmon. Darknet: Open source neural networks in C. https://pjreddie.com/darknet/. Accessed: 2022-4-24. 1

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[8] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. 2

[9] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. 2018. 1

[10] Arslan Siddique and Ilya Afanasyev. Deep learning-based trajectory estimation of vehicles in crowded and crossroad scenarios. In *2021 28th Conference of Open Innovations Association (FRUCT)*. IEEE, 2021. 2

[11] ultralytics. Yolov3: YOLOv3 in PyTorch > ONNX > CoreML > TFLite. https://github.com/ultralytics/yolov3. Accessed: 2022-4-24. 4

[12] Greg Welch and Gary Bishop. An introduction to the kalman filter. https://perso.crans.org/club-krobot/doc/kalman.pdf. Accessed: 2022-4-26. 2

[13] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. *CoRR*, abs/2110.06864, 2021. 1