# (TRACAR) Detection and Tracking of Vehicles

K. G. Lee, A. Padhye, K. Sun
*Electrical and Computer Engineering Department, University of California, Los Angeles*

***Abstract*—In order for an autonomous vehicle to truly be autonomous, it must be able to perceive and understand changes in its surroundings. Our project tackles the detection stage. For our baseline implementation, we wish to develop an embedded system that is able to detect vehicles from video with low compute and efficient memory usage. Our approach uses the YOLO pipeline for detection and correlation filter algorithms for tracking. Even though we were not able to produce the effect of low compute and efficient memory usage using the LCDK, we were able to develop a near real time detection and tracking system.**

***Index Terms*—Object Detection, Object Tracking**

## I. Introduction

### A. History with References

Object detection is a classical computer vision problem concerned with the localization and classification of objects in an image. It is a very active area of research in computer vision as it has many real-world applications across various fields of study [2, 3, 4, 5, 6, 7, 8, 9]. Medical imaging, ball tracking in sports, reverse image search, and self-driving cars are just a few of such applications [17, 18, 19, 20]. The concept of object detection reaches as far back as 1958 when an algorithm for pattern classification was first proposed by Rosenblatt [21]. However, due to the lack of computational power around that time [22], it wasn't until 2012 when AlexNet won the ImageNet Large Scale Visual Recognition Competition considerably that opened up computer vision in the modern era [23]. As of the time of this report, the paper has been cited over 58,000 times.

As object detection grew in prominence, attempting to optimize the computational resources used became an issue. As a result, visual object tracking began rising in popularity [16]. In particular, tracking has many applications in video processing. When an object is detected in one frame, it is useful to be able to track the object. This way, more information about the target and its activities can be extracted [24]. In addition, tracking is a lot easier than detection because tracking algorithms use fewer computations as opposed to running detection on every single frame.

### B. Global Constraints

In this project we develop a near real time system that can detect and track cars driving on roads. Modern day autonomous systems today require a large compute power which is not only economically expensive but also physically very bulky. We originally aimed to use the LCDK, a smaller, cheaper, low memory embedded platform to achieve the same goals. Although we were eventually not able to achieve the LCDK implementation, we were able to custom train YOLOv2, an object detection network and use trackers such as a correlation-based filter (MOSSE) and a neural network (SiamRPN) for tracking the vehicle. [7, 15, 16].

## II. Motivation

In our application, due to memory and compute restrictions, we opt to use a single-stage detector, specifically, YOLOv2. YOLOv2 is smaller than YOLOv3 and still exhibits good performance [8]. Because detection is usually very computationally heavy, we resort to tracking in order to reach near real time performance. In the case of tracking vehicles, we simplify our problem, for now, to a one vehicle scenario in which the vehicle is in full view (not occluded). We use a template matching algorithm known as MOSSE to track the frames succeeding the object detection frame. We chose this algorithm because it is known to be very fast while maintaining high accuracy. We also chose to implement another algorithm known as SiamRPN, a siamese network that uses CNNs to track the target. We then compare the different algorithms to see which has a better precision, frames per second (fps), recall, and finally the F1 score.

## III. Approach

### A. Team Organization

Each person's personal experiences were accounted for when allocating roles. Kevin Lee and Atharva Padhye both have had research experience in the past in object detection and object tracking. Kevin Sun has experience in coding both from previous classes as well as performing research. Kevin Lee worked on training our object detection network and implementing SiamRPN, Atharva Padhye implemented MOSSE, and Kevin Sun generated training labels and wrote a general framework for integrating YOLO with trackers. These roles ensured that the members could efficiently carry out their tasks due to their past experiences.

### B. Plan and Implementation

The initial plan was to finish the project over a 10-week period with 2 major milestones. The first milestone was to finish implementing the baseline object detection network as well as the pruned version of the network in Python; we would have also liked to have started the implementation on the embedded system. The plan was to finish the first milestone by week 3. For the second milestone, which we planned to finish by week 7, we were going to have the object detection implemented on the LCDK. At this point, we would have liked to start our stretch goal of streamlining our code for real-time performance and implementing object tracking on the LCDK. We would also like to potentially pretrain our network on additional datasets, outside of the UFPR-ALPR dataset.

The timeline for the actual implementation is that we managed to hit our first milestone at week 3 of obtaining an object detection network and having it work in Python. However, it was at this point that we decided we should also be working on object tracking in parallel with object detection. So, we did not meet our week 7 milestone of having object detection implemented on the LCDK. By week 8 we had finished implementation of object tracking, as well as obtaining a pruned object detection network in Python and we began the LCDK implementation. We tried our best to finish the LCDK implementation; however, by week 10 we still could not figure out all the bugs. So, we decided to change the objective of this project into comparing two different kinds of object tracking methods (MOSSE and SiamRPN) and see which one has the best performance. We successfully implemented and obtained metrics for both MOSSE and SiamRPN in Python.

## C. *Standard*

Object detection can generally be broken down into two types of detectors: single- stage and two-stage. Two-stage detectors process an image in two stages: the first stage proposes regions of interest, and the second stage classifies those regions. In contrast, single-stage detectors directly run detection over the image. As a result, single-stage detectors often are smaller and faster, but suffer lower accuracy. Some prominent examples of single-stage detectors are YOLO, SSD, and RetinaNet [4, 5, 6, 7, 8]. The most prominent two-stage detector is RCNN, Fast R-CNN, and Mask R-CNN [2, 3, 9, 25].

There is a distinct tradeoff between two stage object detectors and single stage object detectors: speed and accuracy. Two stage detectors are slower but have a much higher accuracy whereas one stage detectors are a lot faster but have lower accuracy [26]. In practice, the higher accuracy is preferred over the speed, so two stage detectors are more popular. In particular, Mask R-CNN is the preferred method [27]. However, in our system, due to the need for real-time performance, a single stage detector is preferred as we are willing to sacrifice accuracy for the speed.

Correlation filter trackers and Siamese trackers are commonly used for their speed and robustness. The Visual Object Tracking (VOT) dataset is an established visual tracking benchmark [10]. The leaderboard on VOT2019 shows a majority of correlation filter and Siamese approaches [11].

In practice, detection and tracking are performed jointly in a couple of different ways. In the Re-Identification (ReID) setting, the most common method is to perform detection as a first step, then perform tracking as a post-processing step [1, 12]. Another approach that has seen popularity recently is to train a single network to jointly perform detection and tracking [13, 14]. Our method is a simple, yet effective fusion of detection of tracking. We do not make any modifications to either our detector or tracker.

## D. *Theory*

MOSSE or Minimum Output of Sum Squared Error is an online filter-based tracking model [16]. The target is initially selected based on the desired object in the first frame, in our case the YOLO object detection would give the target. Then the target is tracked by correlating the filter over the original image. This correlation is computed in the frequency domain to speed up computation time and then its output is then transformed back into the spatial domain. So. the slowest parts of the algorithm are the FFT and IFFT computations, giving it a big O time of O(PlogP) where P is the number of pixels in the window [16]. MOSSE finds a filter that minimizes the sum squared error between the actual output and the desired output.

At the high level, SiamRPN performs like a correlation filter tracker in a learned embedding space [15]. SiamRPN is able to localize templates in a frame by embedding both down to a common embedding space and correlating the feature maps together. Based on this correlation output, the network is able to identify if, and where the template is in the frame. SiamRPN consists of two submodules: a Siamese network and a region proposal network. The Siamese network submodule consists of a common CNN to embed both the template and frame. Next we pass both embeddings into the classification branch and the regression branch of the region proposal network. Each branch has its own set of convolutional filters for the pair of inputs and convolves the outputs together. The structure of the classification and regression branches are the same. The classification branch predicts for each proposed bounding box the probability of the template laying within it and the regression branch predicts the bounding box coordinates.

## E. *Software/Hardware*

We used Google Colab to manage different users on a single code base. Using Google Colab, we were able to take advantage of its GPUs. It also allowed us to store data and files on Google Drive, which would be accessible to all team members. The entire code is written in python because of its speed and its supporting libraries. OpenCV, the open source computer vision library was particularly helpful to us for providing a YOLO implementation which allowed for easy integration with tracking.

From the hardware side, we initially planned on using Texas Instrument's LCDK board for its low power consumption and its speed. However, we were unsuccessful with this because of our difficulty exporting our trained YOLO weights and writing the corresponding C framework. Given more time, we could augment our project with hardware by transferring our system to a LCDK or other embedded system or modifying our system to classify a live video feed from a webcam or similar.

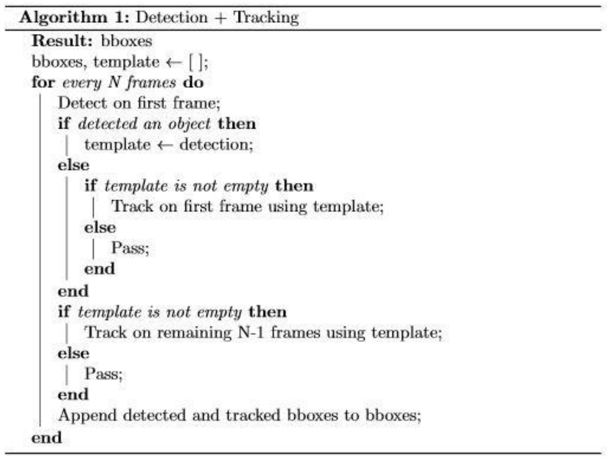
## F. *Operation*

There were four key steps necessary to build our system:
(1) Generate training labels,
(2) Train YOLO,
(3) Implement object trackers (MOSSE and SiamRPN),
(4) Combine trained YOLO with object trackers

To generate training labels, we wrote a script to iterate over all images, parse the label files, and write to a new file annotation in the YOLO format. We train YOLO with a batch size of 32 and 8 subdivisions, with width of 512 and height of 288, taking care to preserve the original aspect ratio of 16:9. We use a multistep learning rate scheduler, with `steps = -1,100,10000,12000` and `scales = .1, 1, .1, .1`. In addition, we turn off random resizing. All other hyperparameters are left to their default values.

We implemented MOSSE with the hyperparameters of learning rate are 0.00125, sigma is 0.1, number of pretrained is 64 to reduce tracker drift. We implement a pretrained SiamRPN and integrate it as is.

To combine detection and tracking, we split our input video into cycles of length N, where the last cycle can have <N frames. For each cycle, we perform detection at the first frame. Then we perform tracking over the remaining N-1 frames using the first frame detection as a template. This process is described in Algorithm 1.



```
Algorithm 1: Detection + Tracking
  Result: bboxes
  bboxes, template ← [ ];
  for every N frames do
      Detect on first frame;
      if detected an object then
          template ← detection;
      else
          if template is not empty then
              Track on first frame using template;
          else
              Pass;
          end
      end
      if template is not empty then
          Track on remaining N-1 frames using template;
      else
          Pass;
      end
      Append detected and tracked bboxes to bboxes;
  end
```

Fig. 1. Algorithm for Detection and Tracking.

## IV. Results

We evaluated our results on four standard metrics: frames per second, precision, recall, and the F1 score. Precision and recall are standard measures for object detection and are defined as the following:

$$Precision = \frac{TP}{TP + FP} = \frac{true\ object\ detection}{all\ detected\ boxes}$$

$$Recall = \frac{TP}{TP + FN} = \frac{true\ object\ detection}{all\ ground\ truth\ boxes}$$

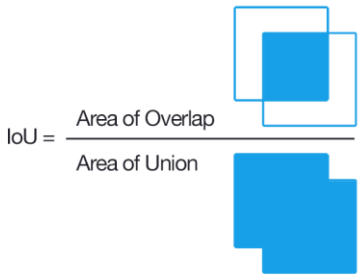


Fig. 2. Visualization of Intersection over Union calculation.

We consider a detection to be a true positive it has an IoU > 0.5 with the ground truth. See Fig. 1. to get intuition for IoU. Precision ranges between 0 and 1 and measures the accuracy of the system over all detections. Recall also ranges from 0 to 1 and measures the overall accuracy of the system. The F1 score is the harmonic mean of the precision and recall and is given by the following:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

We report FPS as a range because in the best case, we perform detection for 1 frame and tracking for N-1 frames with the detected template. In the worst case, we perform detection for all N frames every cycle, in which case the runtime is equal to Detection Only.

| Method | FPS | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Detection Only | 5 | 0.481 | 0.300 | 0.370 |
| Detection+Mosse (2 frames) | 5 | 0.471 | 0.396 | 0.430 |
| Detection+Mosse (5 frames) | 5 | 0.380 | 0.340 | 0.359 |
| Detection+SiamRPN (2 frames) | 5-7 | 0.526 | 0.424 | 0.470 |
| Detection+SiamRPN (5 frames) | **5-10** | **0.536** | **0.429** | **0.477** |

Table 1: Results on Different Methods. Bolded entries are the best-performing for their metric.

We notice that Detection+SiamRPN for (N=5) performs the best in every metric. This is because the Siamese network in SiamRPN is pretrained offline on large video object detection datasets, enabling it to learn more robust representations for the template and frame, in contrast to MOSSE, which is trained online. We would like to caution against taking the results at face value, however. The UFPR-ALPR dataset consists of videos taken over one second, with small appearance changes and with only one annotated vehicle in frame at all times. In a real-world setting where the number of vehicles per frame is variable and the scene changes more drastically between frames, it would be wise to use a smaller value of N to enable the algorithm to react to more dynamic scenes.

Just examining the F1 score, we notice that Detection+MOSSE for (N=5) performs worse than Detection only. This is because MOSSE is an online tracker, that is, it iteratively updates its template with every frame, and is very dependent on hyperparameters. We did not perform extensive hyperparameter optimization for MOSSE, but we also hypothesize that MOSSE is less robust than SiamRPN as mentioned before.

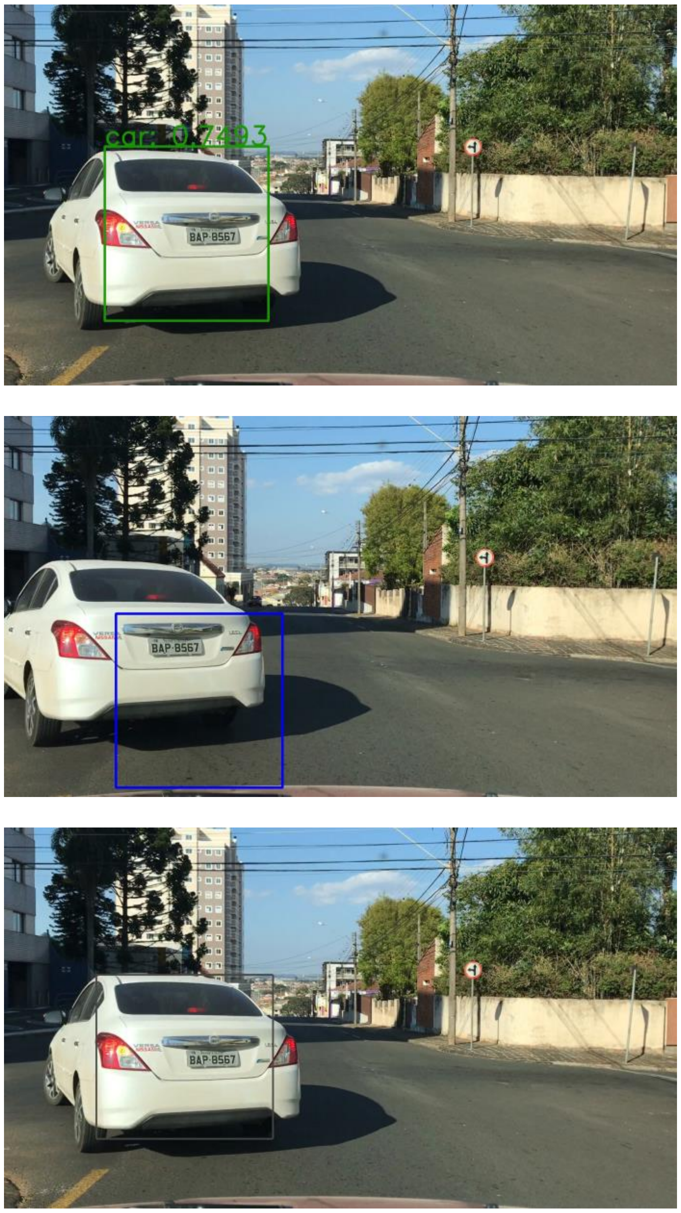


Fig. 3: Qualitative Results for Detection Only (top), Detection+MOSSE(N=5) (middle), and Detection+SiamRPN(N=5) (bottom). MOSSE suffers from tracker drift, while SiamRPN is robust to it. Results are from video track0139. See attached videos for full results.

## References

[1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *The IEEE Conference on Image Processing*, 2016.

[2] RossGirshick.Fastr-cnn.*InternationalConferenceonCom-puter Vision*, 2015.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[4] Tsung-YiLin,PriyaGoyal,RossGirshick,KaimingHe,and Piotar Dolla´r. Focal loss for dense object detection. *International Conference on Computer Vision*, 2017.

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Proceedings of the European Conference on Computer Vision*, 2016.

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

[7] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[8] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Conference on Neural Information Processing Systems*, 2015.

[10] Matej Kristen, Jiri Matas, Ales Leonardis, Tomas Vojir, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka Cehovin. A Novel Performance Evaluation Methodology for Single-Target Trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[11] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Roman Pflugfelder, Joni-Kristian Kamarainen, Luka Čehovin Zajc, Ondrej Drbohlav, Alan Lukezic, Amanda Berg, Abdelrahman Eldesokey, Jani Kapyla, and Gustavo Fernandez. The Seventh Visual Object Tracking VOT2019 Challenge Results. *ICCV 2019 workshops*, 2019.

[12] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple Online and Realtime Tracking with a Deep Association Metric. *arXiv:1703.07402*, 2017.

[13] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to Track and Track to Detect. *arXiv:1710.03958v2*, 2018.

[14] Philipp Bergmann, Tim Meinhardt, Laura Leal-Taixe. Tracking without bells and whistles. *arXiv:1903.05625*, 2019.

[15] Bo Li, Junie yan, Wei Wu, Zheng Zhu, Xiaolin Hu. High Performance Visual Tracking with Siamese Region Proposal Network. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[16] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, Yui Man Lui. Visual Object Tracking using Adaptive Correlation Filters. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[17] Yang Wang, Bogdan Georgescu, Terrence Chen, Wen Wu, Peng Wang, Xiaoguang Lu, Razvan Ionasec, Yefeng Zheng, Dorin Comaniciu. Learning-Based Detection and Tracking in Medical Imaging: A Probabilistic Approach. *Lecture Notes in Computational Vision and Biomechanics*, 2013.

[18] Yu-Chuan Huang, I-No Liao, Ching-Hsuan Chen, Tsi-Ui Ik, Wen-Chih Peng. TrackNet: A Deep Learning Network for Tracking High-Speed and Tiny Objects in Sports Applications. *arXiv:1907.03698*, 2019.

[19] Andras Attila Horvath. Object recognition based on Google's reverse image search and image similarity. *International Conference on Graphic and Image Processing*, 2015.

[20] Ramesh Simhambhatla, Kevin Okiah, Shravan Kuchkula, Robert Slater. Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions. *SMU Data Science Review*, 2019.

[21] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, 1958.

[22] Allen Newell. Perceptrons. An Introduction to Computational Geometry. *MIT Press*, 1969.

[23] *ImageNet Large Scale Visual Recognition Challenge 2012*, 2012. Accessed on: March 20, 2020. [Online]. http://image-net.org/challenges/LSVRC/2012/results.html

[24] Jin Zheng, Bo Li, Ming Xin, and Gang Luo. Structured fragment-based object tracking using discrimination, uniqueness, and validity selection. *Multimedia Systems*, 2019.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. Mask R-CNN. *arXiv:1703.06870*, 2017.

[26] Petru Soviany, Radu Tudor Ionescu. Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. *arXiv:1803.08707v3*, 2018.

[27] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, Rong Qu. A Survey of Deep Learning-based Object Detection. *arXiv:1907.09408*, 2019.