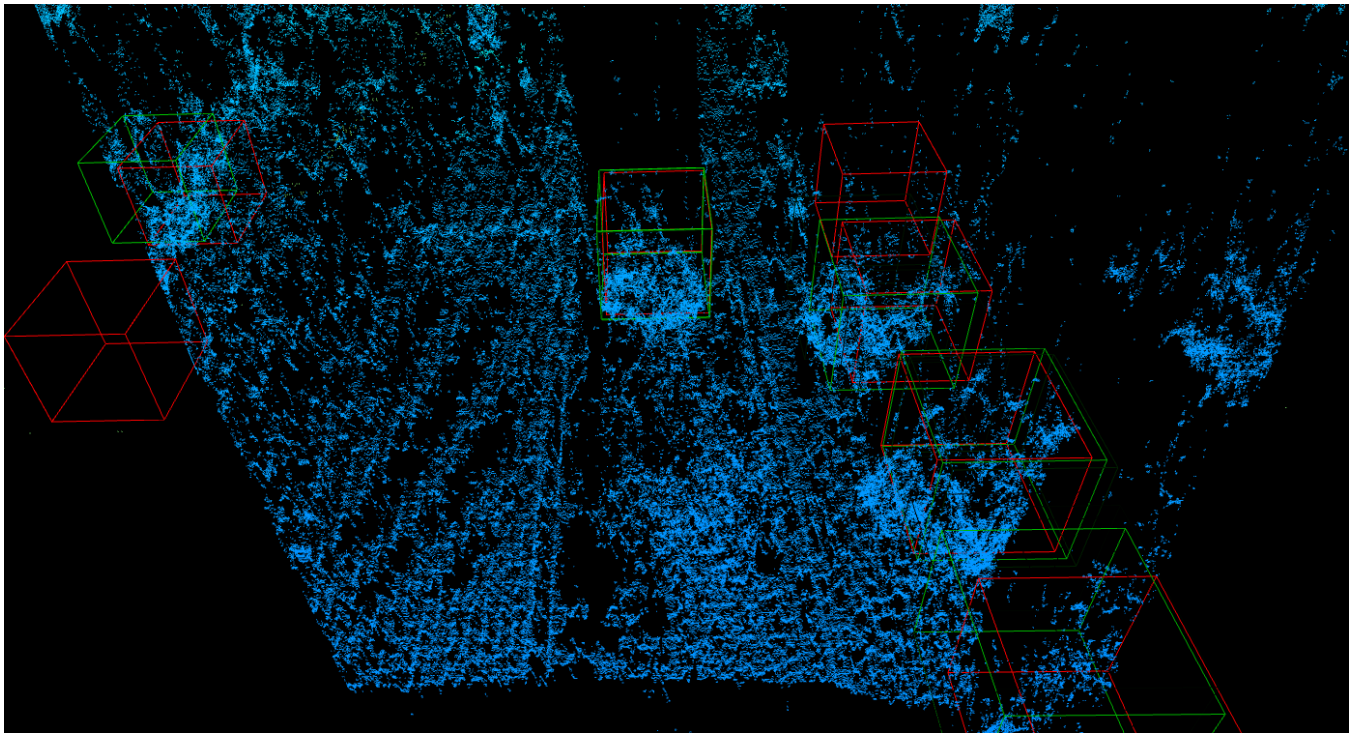

Figure 1: Visualization of our 3D Vehicle Detector. Predictions are green, groundtruth labels are red.

ROBUST 3D OBJECT TRACKING IN AUTONOMOUS VEHICLES

Eric Chan
erchan@stanford.edu

Anthony Galczak
agalczak@stanford.edu

Anthony Li
antli@stanford.edu

December 6, 2019

ABSTRACT

We present a stereo-camera-based 3D vehicle-tracking system that utilizes Kalman filtering to improve robustness. The objective of our system is to accurately predict locations and orientations of vehicles from stereo camera data. It consists of three modules: a 2D object detection network, 3D position extraction, and 3D object correlation/smoothing. The system approaches the 3D localization performance of LIDAR and significantly outperforms the state-of-the-art monocular vehicle tracking systems. The addition of Kalman filtering increases our system's robustness to missed detections, and improves the recall of our detector. Kalman filtering improves the MAP score of 3D localization for moderately difficult vehicles by 7.7%, compared to our unfiltered baseline. Our system predicts the correct orientation of vehicles with 78% accuracy.

1 Introduction

A requirement for safe autonomous vehicles is object detection in 3D space. By detecting cars and other obstacles, an autonomous vehicle can plan a route and avoid collisions. However, tracking 3D objects and rotations is a notoriously difficult problem. Object tracking is most commonly solved by LIDAR, which is accurate but also very expensive. We propose a method utilizing inexpensive stereo cameras. However, stereo cameras come with a major challenge: precision rapidly decreases with increasing distance from the cameras. As a result, the bounding boxes generated by most image-based 3D object detection systems tend to

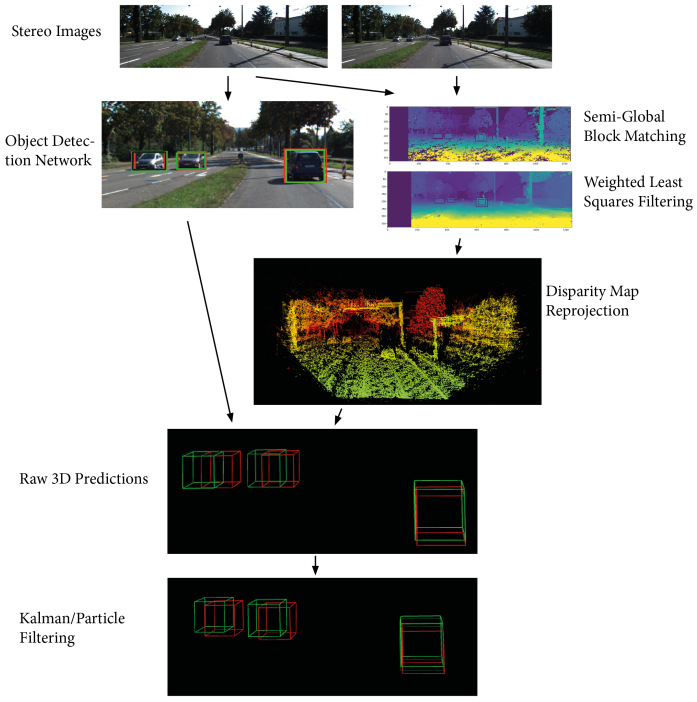

Figure 2: Diagram illustrating steps in our 3D Vehicle-Tracking System.

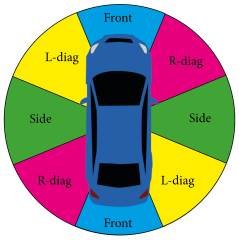

Figure 3: We discretize the observation angle into four distinct categories and treat orientation prediction as a classification problem.

jump around between frames, resulting in an unstable detection. Some camera-based detections are dropped entirely, leading to tracking loss.

Our goal is to predict stable and accurate 3D bounding boxes around vehicles in a self-driving car data set. We aim to provide a vehicle-tracking system that is robust to dropped predictions and more stable frame-to-frame than a pure neural network object detection approach. We are applying our model to the KITTI Object Tracking 2012 data set[2]. Our code, as well as a video demo, is viewable here.

2 Background

Most approaches for object tracking involve using LIDAR or additional inputs beyond standard RGB camera imagery[12], but our focus is on using a camera only approach. However, using more than one image is helpful for determining object depth[15]. The benefits of stereo vision for depth perception informed our choice to use the KITTI object tracking dataset, which has imagery from two forward facing cameras.

We wanted to build the computer vision component of our system on top of an existing object detection library, many of which are quite robust. To inform our selection of an existing library, we looked for a system fast enough to be run in real-time, but also accurate enough to reliably detect most objects. YOLO was an obvious answer for this need[11]. YOLO utilizes a single-pass convolutional neural network to detect objects in a grid constructed from the image.

We decided on using proven methods for tracking moving objects. Both Kalman filtering and Particle filtering are appropriate for the task of tracking multiple objects[16]. These filtering algorithms are often used for the task of tracking moving objects for self-driving cars[17].

3 Approach

We have split the complex problem of 3D vehicle tracking into three separate tasks–2D Object Detection, 3D Position Extraction, and 3D Object Correlation and Smoothing. The steps in our 3D detection pipeline are illustrated in Figure 2.

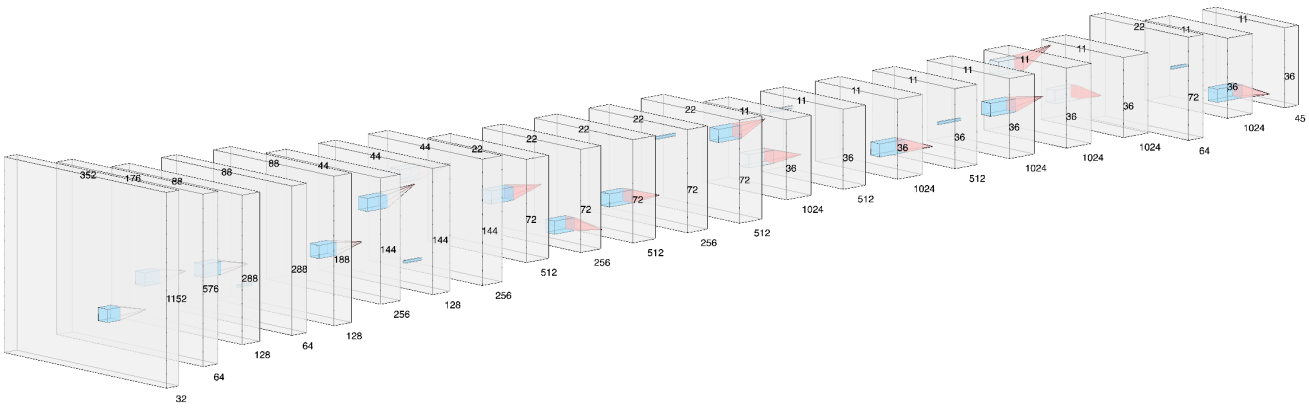

Figure 4: Modified YOLOv2 neural network for 2D object detection and orientation inference.

3.1 Object Detection Network

We utilize a YOLOv2 object detection network to extract image-space bounding boxes and observation angles from our imagery. Our choice of network was informed by prior research into 2D-object detection for vehicles. YOLOv2 has been shown to outperform YOLOv3 in both Mean Average Precision (MAP) and inference time on vehicle detection tasks[20]. While Faster-RCNN outperforms YOLOv2 in MAP, it is incapable of running in real-time for high frame rates[20]. Compared to YOLOv3[22] and Faster-RCNN, YOLOv2 has a good balance of accuracy and performance. Our network is based on the Tensorflow implementation of the YOLOv2, Darkflow [21].

We framed orientation prediction as a classification problem, and discretized the observation angles of vehicles into four separate categories, as shown in figure 3. The work of Rybski, et. al. [14] has shown that orientation estimation networks tend to confuse exactly opposite orientations–for example, front-facing and rear-facing cars, or drivers-side and passenger-side cars. Because absolute orientation is not necessary for predicting bounding box orientation–an approaching box is geometrically identical to a retreating box–we made the decision to treat opposite orientation as the same class. By reducing the number of classes predicted by the network, we increased the robustness of the system.

In order to fit our specific task, we modified the YOLOv2 network. Because we were focused on vehicles and orientation prediction rather than the more complicated task of general object detection, we simplified the network and reduced the number of filters in the deep convolutional layers. Since bounding box accuracy is very important to our task, we adjusted the layer sizes to operate on higher resolution images. We initialized the network using weights pre-trained on the COCO dataset, but fine-tuned it to the KITTI 2D Object Detection dataset. We also utilized K-means clustering to generate anchors that better fit the KITTI dataset. Our modified network structure is given in figure 4. Details of our network parameters are given in Appendix A. We experimented with training the network on stereo disparity images in addition to RGB images, but our results were not shown to improve significantly.

3.2 3D Position Extraction

After localizing vehicles in 2D image space, we used stereo correspondence algorithms to estimate the 3D positions of our detections. In order to calculate stereo disparity from our rectified image pair, we utilized Semi-Global Block Matching[10]. Unlike previous scan-line approaches that calculated disparity only along the horizontal epipolar lines, Semi-Global Block Matching optimizes and averages across multiple directions, resulting in a smoother estimation of pixel disparity.

Despite the improvements of SGBM over older disparity estimation algorithms, SGBM is still prone to errors in areas of half-occlusions, as well as regions with depth continuities. In order to further improve the accuracy of the disparity map, we computed both the left-to-right and right-to-left stereo correspondences. We then combined the two estimations into a smoother, more complete disparity map using Weighted Least Squares Filtering[5].

After computing the accurate disparity map, we reprojected the 2D image space points into 3D using a perspective transformation.

3.3 Kalman Filtering

To achieve the objective of building a tracking system that is stable and robust under short-term occlusions, we used Kalman filtering on top of the positions returned from the 3D position extraction step. The filters then return corrected positions based on position history, assuming that vehicles maintain consistent velocity. The Kalman filter is implemented using the *pykalman* library.

A significant challenge in multiple object tracking is data association. The object detection network is not designed to return the locations of detected vehicles in a consistent order from frame to frame. Therefore, we must match new detections to their previous locations. Our current heuristic is to match each position to the Kalman filter that recorded the closest previous-frame position. However, if this distance exceeds a certain cutoff, we assume that the position represents a new vehicle and instantiate a new Kalman filter.

Each instantiated Kalman filter models six parameters: x position, x velocity, y position, y velocity, z position, and z velocity. We encode the following constant-velocity transition model:

$$x_t = x_{t-1} + dt * xvelocity_{t-1} \quad y_t = y_{t-1} + dt * yvelocity_{t-1} \quad z_t = z_{t-1} + dt * zvelocity_{t-1}$$
$$xvelocity_t = xvelocity_{t-1} \quad yvelocity_t = yvelocity_{t-1} \quad zvelocity_t = zvelocity_{t-1}$$

When filters are initialized, the initial position is set to the initial observation and the initial velocity is set to zero. Initial position covariance is set to 1 and initial velocity covariance is set to 50, since we are initially very unsure about velocity.

Any Kalman filter that is not updated in a given frame is updated with a null position so the filter assumes that the vehicle will continue to move at the last known velocity. If a filter is not updated for several consecutive frames, we assume the vehicle is no longer present and we remove the filter from memory.

Pseudocode for the Kalman filter implementation can be represented as follows:

```
kalmanfilters = []
for each frame f in sequence:
    run object detection on f
    for each detection d:
        find the filter k in kalmanfilters closest to d that has not been assigned to a detection
        if k is too far away (distance > dist_cap) or if there are no filters:
            initialize a new filter k'
            add k' to kalmanfilters
            set k'.confidence to d.confidence
            set k'.staleness to 0
            add observation [d.x, d.y, d.z] to k' to generate filtered position k'.filtered_x/y/z
            mark k' as assigned
        else if k is not too far away:
            add observation [d.x, d.y, d.z] to k to generate filtered position k.filtered_x/y/z
            set k.confidence to d.confidence
            set k.staleness to 0
            mark k as assigned

    for each filter k in kalmanfilters that is still not assigned to a detection:
        # Add a None observation to generate a corrected position via velocity interpolation
        add observation [None, None, None] to k and generate filtered position k.filtered_x/y/z
        decay k.confidence by a constant decay_rate
        add 1 to k.staleness
        if k.staleness exceeds max_staleness:
            remove k from kalmanfilters

    # Now we can construct an updated set of detections from the filters.
    updated_detections = []
    for each filter k in kalmanfilters that has been assigned:
        add ([k.filtered_x, k.filtered_y, k.filtered_z], k.confidence) to updated_detections

    # Optionally, add detections for filters that were not updated in this frame
    for each filter k in kalmanfilters that has not been assigned:
        if k.confidence is large enough (> restoration_conf_threshold):
            add ([k.filtered_x, k.filtered_y, k.filtered_z], k.confidence) to updated_detections

    save updated_detections for accuracy evaluation
```

Figure 5: Pseudocode for filter initialization and association

This algorithm has several notable tunable parameters. First, the *dist_cap* parameter can be tuned to vary how close a filter's last recorded position must be to a detected vehicle position in order for the filter to be matched to the detected vehicle.

When a filter is matched to a detected vehicle, the confidence associated with the filter is set to the confidence of the detection returned by the object detection network. The *decay_rate* can be tuned to adjust how quickly the confidence associated with a filter changes if the filter is unable to be matched to a detected object in a frame.

When a filter is matched to a detected vehicle, its *staleness* value is reset to 0. When a filter is not matched, *staleness* value is increased by 1. The *max_staleness* parameter can be adjusted to control how high a filter's staleness can get before the filter is assumed to be no longer useful and removed.

Finally, we can control whether we produce detections from filters that are not matched to a detected vehicle in a given frame. Generally we expect this to be enabled so that we continue tracking vehicles during occlusions, which is one of the main benefits of implementing filtering. The *restoration_conf_threshold* parameter can be increased to only produce detections from non-matched filters that still have a high confidence value.

3.4 Particle Filtering

We also performed cursory testing and evaluation with a custom particle filtering implementation. In our particle filtering implementation, filters are initialized with an initial (x, y, z) position observation. 100 particles are then generated uniformly in a 1-meter radius around the initial observed position. When a new observation is received, the last three velocities (or however many are available, if fewer than three) are averaged and all particles are moved by the averaged velocity. Then, a weight is assigned to each particle based on its distance to the observation following a 3D Gaussian centered on the observation location with a covariance of 9 meters. A particle is sampled according to these weights and the sampled particle location is returned as the filtered position. If an observation is too far away from all of the particles causing all particle weights to be near zero, particles are regenerated uniformly in a 1-meter radius around the observation and the position/velocity history of the filter is cleared.

Filter association with detected vehicles is performed using the same procedure as implemented for Kalman filtering (Figure 5).

3.5 Datasets

We train our 2D object detector network on the KITTI Object Detection 2012 Dataset. Each image is labeled with a ground truth 2D bounding box and a ground truth observation angle.

We evaluate our tracking accuracy on the KITTI Object Tracking 2012 dataset. The dataset provides sequential stereo camera data along with ground-truth 3D bounding box labels.

3.6 Evaluation Metrics

We evaluate our system using the criteria suggested by the KITTI vision benchmark. KITTI defines difficulty classes for detections. Vehicles classified as "Easy" are greater than 40 pixels in image-space height and fully visible. Vehicles classified as "Moderate" or "Medium" are greater than 25 pixels in height and "partly occluded". Vehicles classified as "Hard" are greater than 25 pixels in height and "difficult to see" according to the authors of KITTI.

To measure localization accuracy for 3D tracking, we calculate Precision-Recall curves and a MAP metric. We consider a prediction to be a True Positive if the center-of-mass of the prediction lies within 1.5 m to the center-of-mass of its corresponding groundtruth location. In our results, we also evaluate at 3 m, 5 m, and 10 m distance thresholds.

To measure orientation prediction accuracy, we compare the predicted orientation class against the groundtruth orientation class for each matched vehicle pair. We calculate orientation prediction precision as the ratio of correctly classified vehicles to total orientation predictions.

For 2D object detection, we calculate Precision-Recall curves and a MAP metric. We consider a prediction to be a True Positive if the Intersection over Union (IoU) with an unmatched groundtruth box exceeds 0.5.

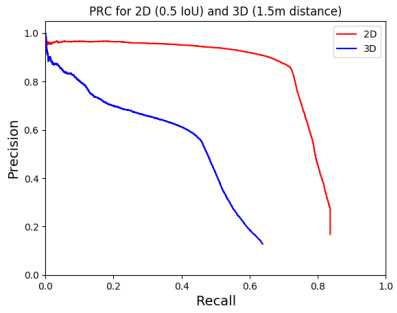

Figure 6: PR curve for 2D tracking vs. 3D tracking.

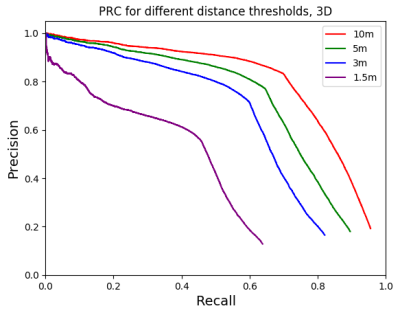

Figure 7: PR curve for each distance threshold, using Kalman filtering.

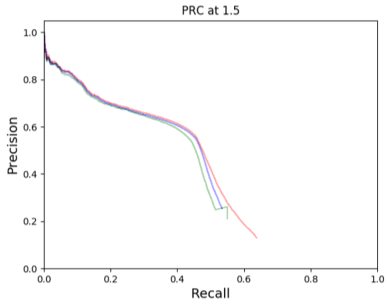

Figure 8: PR curve for Kalman (red), unfiltered (blue), and Particle (green).

Table 1: Mean Average Precision comparison of our unfiltered, Kalman filtered, and Particle filtered results against state-of-the-art Monocular and LIDAR vehicle-detection systems.

	Unfiltered	**Kalman**	Particle	Mono[13]	LIDAR[19]
Easy	0.799	0.8079	0.7251	.1805	0.8661
Med	0.3737	0.4027	0.3671	.1498	0.7763
Hard	0.1198	0.1269	0.1192	.1342	0.7606

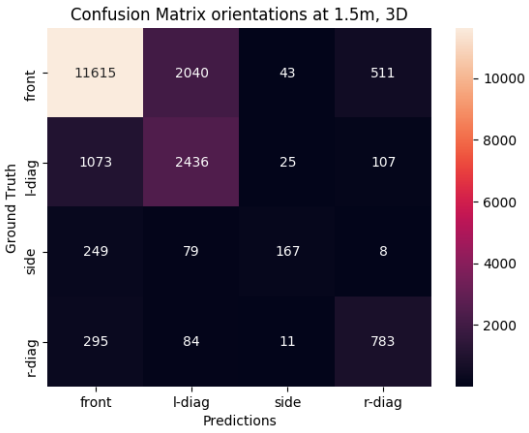

Figure 9: Confusion matrix for orientation predictions.

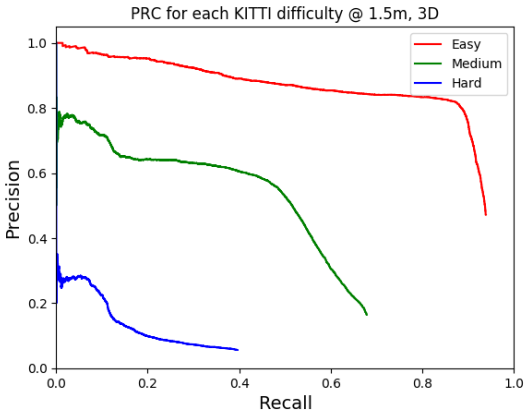

Figure 10: PR curve for each KITTI difficulty.

4 Results

Our combined pipeline is able take in raw stereo images and output filtered 3D bounding boxes. A visualization of our current results is given in Figure 1.

4.1 Localization Performance

Figure 10 and table 1 compare the performance of our 3D tracking system at different difficulties. Our system performs very well on "Easy" detections–vehicles that are fully visible and relatively close to the camera. Our system has lower performance on vehicles that are far away and partially or significantly occluded, but still performs adequately. These results indicate that as expected, localization precision decreases with increasing distance to the cameras.

Figure 7 gives average precision curves for our system at different distance thresholds. Our system is able to localize the majority of vehicles to within 1.5 m of their real-world positions. It is able to localize nearly all vehicles to within 3 m of their real-world positions. Our system, especially at long ranges, is not perfectly precise, and will not give millimeter-perfect localization. However, we contend that the precision is adequate for most autonomous vehicle tasks.

4.2 Orientation Prediction Performance

The confusion matrix given in figure 9 compares our predicted orientations to groundtruth orientations. Our system correctly predicts the orientation of cars 76.8% of the time. The most common prediction errors are between adjacent angles. For example, the network is more likely to predict a front-facing car to be diagonal than it is to be completely sideways.

Because many training images are taken from highway driving, the KITTI dataset is heavily biased towards front-facing cars. There are few training images captured at intersections, so the performance of "side" predictions is comparatively low. We believe that additional training examples with more varied orientations would improve our network's orientation prediction performance.

4.3 Filtering performance

We evaluated the 3D localization performance of three different filtering models: an unfiltered baseline model, a particle filtered model, and a Kalman filtered model. The precision-recall curves for 3D localization with particle filtering and Kalman filtering are compared to unfiltered results in Figure 8. MAP values for the three filtering methods are given in Table 1.

Kalman filtering improved baseline 3D localization performance for all three difficulties. However, the greatest improvements came in localizing medium and hard vehicles, where Kalman filtering improved MAP by 7.7% and 5.9% respectively. Filtering provides two benefits: robustness to dropped predictions and smoothing of sensor noise.

The raw prediction system sometimes fails to detect vehicles that are difficult to detect, "dropping" predictions. Our filtering solution provides robustness to dropped predictions by propagating tracked vehicles. When our object detector fails to predict a vehicle for 1-2 frames, the filter continues tracking through the brief loss of detection, updating the position of our estimate using our constant-velocity assumption.

Kalman filtering additionally improves the robustness of our tracking to measurement errors and noise. By aggregating multiple noisy samples over time, Kalman filtering allows us to extract a better estimate of a tracked vehicle's true position. The "moderate"

and “hard” difficulty vehicles are further away and more heavily occluded than the “easy” vehicles, so measurement noise of our raw localization performance tends to be worse. It thus makes sense that Kalman filtering provides the greatest benefits to the more difficult vehicles.

We did not see an improvement over the baseline image object detection using particle filtering. However, it is likely that we have not yet identified optimal particle filtering parameters which would improve performance for this application.

5 Discussion

5.1 Comparison to LIDAR

Table 1 compares our localization performance to a state-of-the-art LIDAR vehicle-detection system. As expected, LIDAR outperforms our model at all difficulties [19]. However, Figure 7 shows a compelling result when we relax the constraints on how precise the detections need to be. The performance of our model increases dramatically when we relax the detection distance threshold to 3 m. The takeaway is that although our system lacks the fine-grained precision of LIDAR, it can still adequately track most vehicles.

Our system achieves comparable precision to LIDAR for vehicles that are “Easy” to detect, but performs significantly worse for more difficult vehicles. Many of these difficult to detect vehicles are cars that are smaller than 40 pixels in image-space height, i.e. cars that are far away from the cameras. Part of our tracking difficulty likely comes from the inherent imprecision of stereo depth perception at longer ranges, where LIDAR possesses a significant resolution advantage. However, precise tracking at long range is often unimportant to autonomous vehicle tasks, where nearby objects pose the greatest collision risk. Because our system is comparable to LIDAR at close range, we believe it offers a compelling alternative to LIDAR.

5.2 Comparison to Monocular 3D object tracking

Table 1 compares our localization performance to a state-of-the-art Monocular 3D vehicle detection system, *Disentangling Monocular 3D Object Detection* by Simonelli, et al. [13]. Our system significantly outperforms the monocular object detector, achieving a 450% MAP improvement for “Easy” detections and a 270% improvement for “Medium” difficulty vehicles while performing comparably for “Hard” vehicles. Note that the monocular object detector MAP values are evaluated at a distance threshold of 2 m compared to our less-forgiving default threshold of 1.5 m; the performance gap would likely increase further at the same distance threshold.

5.3 2D vs. 3D object tracking

Figure 6 compares the accuracy of our 2D predictions to that of our 3D predictions. 2D detection performance is significantly greater than 3D detection performance. This is partially due to the "dimensionality curse"–the much larger scale of the 3D volume makes 3D localization an inherently more difficult problem.

However, the performance gap is also due to the limitations of our stereo ranging system. Our 3D detector performs much worse on vehicles that are significantly occluded. When objects are significantly occluded, our ranging method, which operates on the principle of median filtering, often returns the distance to the *occluding* object rather than the occluded vehicle. We hypothesize that a more robust ranging algorithm could produce significant performance improvements for difficult vehicles.

5.4 Data Association

Data association is a surprisingly difficult challenge in the arena of driving because both the tracked object and the camera are moving. We examined several approaches for correlating the objects from the previous states to the current state, known as “solving the data association problem”. Compared to the methods used in the literature[17], our approach (Figure 5) is fairly naive. For example, one method in the literature uses features from the object in the image to correlate multiple detection systems to solve the data association problem[18]. In addition, this method considers six different possible actions to take for each filter after an observation is made as opposed to our two ”match“ and ”no match“ actions. We felt that this level of complexity was out of the scope of this project.

6 Conclusion

We have implemented a successful 3D object tracking algorithm using only stereo cameras. Kalman filtering has shown to create a performance improvement over raw CV object detection. Our performance could likely be significantly increased by implementing a more robust range-detection algorithm and a more robust data association algorithm.

7 Group member contributions

Anthony Li worked on implementation/tuning of Kalman and particle filtering along with the filter correlation algorithm.

Anthony Galczak worked on initial implementation of C-based Darknet (YOLOv3) implementation, data grooming, and all of the evaluation metrics and figures.

Eric Chan worked on the 2D object detection network, implemented 3D position extraction, and implemented the evaluation metrics.

Anthony Galczak and Eric Chan are also utilizing this project for CS230. While much of this project is relevant to both courses, Kalman and particle filtering implementation and evaluation represent the main extension for CS238.

References

[1] https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/overview

[2] (n.d.). Retrieved from http://www.cvlibs.net/datasets/kitti/eval_tracking.php.

[3] Holger Caesar and Varun Bankiti and Alex H. Lang and Sourabh Vora and Venice Erin Liong and Qiang Xu and Anush Krishnan and Yu Pan and Giancarlo Baldan and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. arXiv preprint arXiv:1903.11027, 2019.

[4] Waymo Open Dataset: An autonomous driving dataset. https://www.waymo.com/open. 2019.

[5] Dongbo Min, Sunghwan Choi, Jiangbo Lu, Bumsub Ham, Kwanghoon Sohn, and Minh N Do. Fast global image smoothing based on weighted least squares. Image Processing, IEEE Transactions on, 23(12):5638–5653, 2014.

[6] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[7] K. Bernardin, R. Stiefelhagen: Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. JIVP 2008.

[8] Andreas Geiger. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12). IEEE Computer Society, Washington, DC, USA, 3354-3361.

[9] Mousavian, A., Anguelov, D., Flynn, J., & Kosecka, J. (2017). 3D Bounding Box Estimation Using Deep Learning and Geometry. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.597

[10] Hirschmuller, H. (2005). Accurate and Efficient Stereo Processing by Semi Global Matching and Mutual Information. CVPR .

[11] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: CVPR (2016)

[12] Rangesh, Akshay. No Blind Spots: Full-Surround Multi-Object Tracking for Autonomous Vehicles Using Cameras & LiDARs. 23 Feb. 2018.

[13] Simonelli, Andrea et al. "Disentangling Monocular 3D Object Detection." ArXiv abs/1905.12365 (2019): n. pag.

[14] Rybski, Paul & Huber, Daniel & Morris, Daniel & Hoffman, Regis. (2010). Visual Classification of Coarse Vehicle Orientation using Histogram of Oriented Gradients Features. IEEE Intelligent Vehicles Symposium, Proceedings. 921 - 928. 10.1109/IVS.2010.5547996.

[15] Ginhoux, R. & Gutmann, Steffen. (2001). Model-based object tracking using stereo vision. Proceedings - IEEE International Conference on Robotics and Automation. 2. 1226 - 1232 vol.2. 10.1109/ROBOT.2001.932778.

[16] Marrón-Romera, Marta & Garcia Garcia, Juan Carlos & Sotelo, Miguel-Angel & Cabello, Mariana & Pizarro, Daniel & Huerta, Francisco & Cerro, Jaime. (2007). Comparing a Kalman Filter and a Particle Filter in a Multiple Objects Tracking Application. 1 - 6. 10.1109/WISP.2007.4447520.

[17] Janai, Joel. Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art. ArXiv, 18 Apr. 2017.

[18] W. Choi, C. Pantofaru and S. Savarese, "A General Framework for Tracking Multiple People from a Moving Camera," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 7, pp. 1577-1591, July 2013.

[19] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, Jiaya Jia; The IEEE International Conference on Computer Vision (ICCV), 2019, pp. 1951-1960

[20] Wang, Yizhou. Object Detection on KITTI Dataset Using YOLO and Faster R-CNN. 20 Dec. 2018, http://yizhouwang.net/blog/2018/12/20/object-detection-kitti/.

[21] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 6517-6525. doi: 10.1109/CVPR.2017.690

[22] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.

8 Appendix A - Network parameters

Training Hyperparameters	
Minibatch Size	8
Learning Rate	1.00E-04
Optimizer	ADAM
Beta1:	0.9
Beta2:	0.999

Layer	Output Height	Output Width	Output Depth	Filter Height	Filter Width
Convolutional	352	1152	32	3	3
Max-Pool	176	576	32	2	2
Convolutional	176	576	64	3	3
Max-Pool	88	288	64	2	2
Convolutional	88	288	128	3	3
Convolutional	88	288	64	1	1
Convolutional	88	288	128	3	3
Max-Pool	44	144	128	2	2
Convolutional	44	144	256	3	3
Convolutional	44	144	128	1	1
Convolutional	44	144	256	3	3
Max-Pool	22	72	256	2	2
Convolutional	22	72	512	3	3
Convolutional	22	72	256	1	1
Convolutional	22	72	512	3	3
Convolutional	22	72	256	1	1
Convolutional	22	72	512	3	3
Max-Pool	11	36	512	2	2
Convolutional	11	36	1024	3	3
Convolutional	11	36	512	1	1
Convolutional	11	36	1024	3	3
Convolutional	11	36	512	1	1
Convolutional	11	36	1024	3	3
Convolutional	11	36	1024	3	3
Convolutional	11	36	1024	3	3
Concat	22	72	512		
Convolutional	22	72	64	1	1
Concat	11	36	1280		
Convolutional	11	36	1024	3	3
Convolutional	11	36	45	1	1