2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

A State Machine-based Multi-Vehicle Tracking Framework with Dual-Range Radars

Jiawei Huang[1] and Lichao Ma[2]

Abstract— **Vehicle tracking is an essential topic in autonomous driving. Currently most systems rely on radars and lidars to perform vehicle tracking. In this paper, we present a novel cross traffic vehicle tracking system with several unique contributions. First of all, it employs a state machine to manage the life cycles of particle filters, resulting in higher tracking robustness. Secondly, the entire software framework is designed to be extensible to support multiple sensors and tracking algorithms. Lastly, we implemented a sensor-vehicle co-simulator to evaluate the tracking performance. We show through experiments that our vehicle tracking system can track multiple vehicles up to 170m away with less than 1m average positional error. We also show that our proposed state machine improves tracking rate under frequent occlusion.**

I. INTRODUCTION

For an autonomous vehicle to operate safely on public roads, the ability to reliably and accurately track surrounding vehicles is crucial. Among common sensors used for vehicle tracking, millimeter wave radar is distinguished from the rest for its relative low cost, long-range operation, immunity against environmental factors and the Doppler velocity information it provides. As a result, radar sensors have been widely deployed in many current model vehicles, for example in Honda Sensing, Audi Pre Sense, BMW, etc. However, due to the nature of millimeter waves, the number of measurements returned by a moving object is usually very small. Therefore it is common to approximate the target as a point mass or assume fixed shape. In order to estimate the actual shape of the target vehicle, we need either high-resolution radars [1] or lidars [2], [3].

A vehicle tracking system is usually composed of a frontend which performs feature extraction and sensor fusion and a backend which does state estimation. Sensor fusion is an indispensable task as long as multiple sensors are involved. The most popular approach is to treat each sensors measurement as an independent observation of the underlying vehicle state [2], [4]. The backend usually uses one or more stochastic filtering techniques, e.g. Kalman filter and particle filter and mixture-model [5]. In recent years, researchers also start to apply deep learning to solve tracking problems [6].

Although there is a large body of work on vehicle tracking, in reality there are still many practical challenges. For example, tracking single vehicle is relatively easy but when there are multiple vehicles, the occlusion can cause the tracker to fail. The performance of tracking algorithms is also highly dependent on the sensor quality and characteristics. In this paper, we proposed uniques approaches to address some of these problems, such as occlusion, sensor blind spot, missing frames, etc.

Our work is based on the state and measurement model presented in [1]. However, there are several key differences. First of all, the radar we used is not high resolution and therefore at long distance the amount of measurements is very scarce. This presents challenges to the tracker. Secondly, we proposed a state machine framework to handle occlusions and bad sensor input. Finally, our method can track multiple targets.

The paper is organized as follows. In Section II, we give an overview of our SW system architecture and sensor configuration. In Section III, we elaborate on the state machine manager which is our key contribution. In Section IV, we introduce the simulator used to evaluate our tracking algorithm. Finally experimental results are presented in Section V.

II. SYSTEM ARCHITECTURE

The overall tracking system (Fig. 1) is partitioned into three components: a front end, a core, and a back end. The front end processes data directly from sensors. It is responsible for gathering and preprocessing sensor input, transforming data between different coordinates, performing data clustering and sensor fusion, and outputting tracking results. Outside the front end, data are sensor agnostic. The core manages both the front end and the back end by passing data between the two. It also contains a testing engine which is responsible for tuning tracker parameters, performance optimization and quantitative evaluation. The back end is the most complex and is responsible for the actual tracking. It has a hierarchy that we specially designed to support multiple object tracking and different tracking algorithms which we will discuss in detail in Section II-A.

A. Separation of front end and back end

The separation of front end and back end is the key element to enable support for multiple sensors and tracking algorithms. In principle, the back-end tracking algorithms such Kalman filter and particle filter do not require sensor-specific information. It is also not a scalable practice to include sensor-specific code in the back end. Consider the case to support m types of sensors and n types of tracking algorithms. If we do not separate front end and back end, we will need $m \times n$ implementations of tracker. That number

[1]Jiawei Huang is with Honda Research Institute, USA jhuang@honda-ri.com
[2]Lichao Ma is with Honda Research Institute, USA mma@honda-ri.com

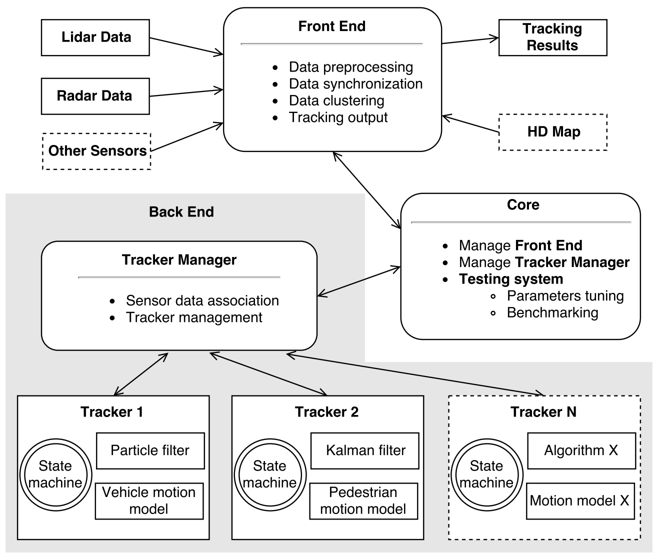

Fig. 1. Vehicle tracking system diagram. The mdoules in dashed lines show the scalability of the system.

reduces significantly to only $m + n$ once we partition front end and back end.

B. Extension to multiple sensors and tracking algorithms

The backend adopts a logical hierarchy of tracker manager, individual tracker and tracking algorithm. At the top level, a tracker manager manages all tracked objects. It creates a tracker for every tracked object, associates incoming measurements to every tracker and determines whether to split, join or remove trackers.

Each tracker tracks a single object. Internal to every tracker is a state machine (detailed in Section III) and one or more tracking algorithms (such as Kalman filter or particle filter). The tracker is a polymorphic container (implemented as a C++ abstract base class) which can be instantiated based on the underlying object type and tracking algorithm. For example, for cross traffic vehicle tracking, we implement a particle filter-based vehicle tracker which has unique motion and measurement model to describe vehicle dynamics and geometry. If the scenario changes to pedestrian tracking with Kalman filter, we just need to replace the particle filter with Kalman filter and implement the motion and measurement model suitable for pedestrians. A tracker can also contain multiple tracking algorithms. In such cases, they will be running in parallel and the tracker decides to pick the best performing algorithm for output or fuse the output from multiple algorithms.

The state machine serves as a low-level manager of the underlying tracking algorithm. It provides several benefits over using a bare tracking algorithm. Section III covers state machine in detail.

C. Parameter tuning and tracker evaluation

Another unique feature of our tracking system is the testing system. Since particle filter contains a large number of parameters, all of which affect the tracking performance, tuning these parameters for best performance becomes an important but laborious work. Closely integrated with the tracking system is a simulator which simulates the radar sensor as well as target vehicles. The simulator generates two types of output: the simulated sensor data and the ground truth vehicle trajectory. The sensor data are used as input to the tracking system, and the output is compared against the ground truth. Finally, we compute an error metric which describes the overall performance of the tracking system. We use root-mean-square error (RMSE) [1] (Eq. 1) as the performance metric. Here $\hat{x}_t$ represents the ground truth state variable and x_t is the corresponding value estimated by the tracker.

$$RMSE = \sqrt{\sum_{t=1}^{n} \frac{(\hat{x}_t - x_t)^2}{n}} \tag{1}$$

We adopted a parameter sweep methodology where a set of key parameters are selected and a discrete set of values are tested for each parameter. The simulator simulates every combination of the parameters and reports its error. Section IV has a more detailed treatment of the simulator. When the parameter space becomes large, a more efficient optimization method – gradient descent can be used. For gradient descent, we search the parameter space for the minimum error point along the negative error gradient direction.

D. Sensor Configuration

We used Delphi ESR dual-range radar which is designed for automotive applications. Important specifications of the radar are highlighted in Table I. It has two range modes: mid-range up to $60m$ with 90-degree field-of-view (FOV), and long-range up to $175m$ with 20-degree FOV. It switches between the two modes at 40Hz. Two ESRs are mounted on our test vehicles front corners, directly facing left and right. Fig. 2 shows the FOV of the left radar and the detection of a car approaching from the left.

TABLE I
DELPHI ESR TECHNICAL SPECIFICATIONS

Parameters	Long-range	Mid-range
Range	175m	60m
Range Accuracy	0.5m	0.25m
Range Rate	-100m/s to + 40m/s	
Range Rate Accuracy	0.12m/s	
Azimuth FOV	>20deg	>90deg
Update Frequency	>=20Hz	

It is clear in Fig. 2 that the radar measurements are generally quite sparse and there are small stretches of distance without any measurement. Once we receive the radar data, we first perform preprocessing by filtering out low-velocity and low-intensity objects. Then we perform clustering and association. We use the following radar clustering and association algorithm:

1. Combine short and long-range radar data.

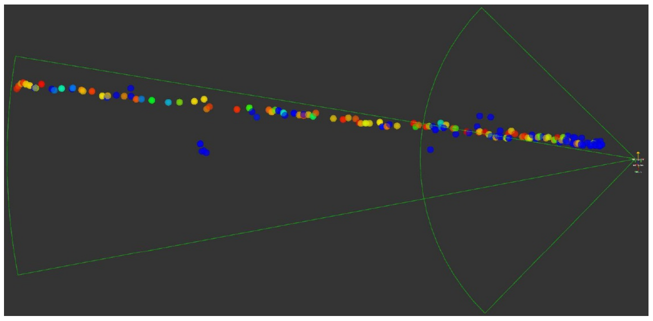

Fig. 2. The field-of-view (FOV) profile of Delphi ESR radar with dual range sensing. The colored radar points show the signal strength, with red being strong and blue being weak.

2. Filter data by velocity, strength and position.
3. Cluster radar points into radar clusters.
4. Associate each radar cluster to existing trackers
   a. Assign each radar cluster to its closest tracker if the distance is under a threshold
   b. Else create a new tracker using the cluster

Since we allow ego vehicle motion, it is necessary to transform sensor data from ego vehicle frame (F_v) to a world frame (F_w). This is because we want to perform tracking in a frame stationary relative to ground so that the target vehicle obeys the motion model. In order to obtain this transform, we need to know the accurate pose and velocity of ego vehicle. We used a localization algorithm which combines DGPS and visually detected lane markings to provide accurate ego pose and velocity [7]. Once this transform is obtained, we represent sensor data, vehicle state and particles all in F_w. However, since the measurement model is formulated in the ego vehicles frame F_v, we must take an extra step to convert particle state and sensor measurement back into F_v to evaluate the weight of each particle. Outside the weight evaluation routine, all quantities are expressed in F_w frame.

III. PARTICLE FILTER STATE MACHINE

Particle filter [8] is a popular method for state estimation. It is a Monte Carlo approach which is applicable to nonlinear and non-Gaussian systems. In essence, particle filter uses a set of random samples (called particles) with associated weights as discrete approximation of state posterior within dynamic Bayesian framework.

To formulate the problem, we need a motion model and a measurement model. We adopted the formulation in [1] where the motion model assumes constant velocity and yaw rate, and the measurement likelihood is the product of Doppler velocity likelihood, range likelihood and angle likelihood. We used the following 5-tuple state vector: $[x, y, \theta, v, \omega]$, representing 2-D vehicle position, heading angle, velocity and yaw rate.

However, during the testing stage, we found that the vanilla particle filter approach gave unsatisfactory results. For example, the raw radar data from a moving object are temporally unreliable. A moving object is not observable in every radar scan due to occlusion and sensor noise. This causes the particle filter to constantly reinitialize. In addition, due to the FOV of our dual-range radar, there is a blind spot region where no observation is received. A particle filter can easily diverge and lose track in this region. In order to address these issues, we propose to use a state machine to manage the behavior of every particle filter. The state machine can maintain the state of the tracker through the blind spot so that the filter does not need to be reinitialized after reappearing in FOV. State machine also improves the performance of multiple objects tracking. By knowing the current state of each tracker, the manager could allocate resources more efficiently. For example, when a tracker is first created, the manager could allocate more particles to it to speed up the convergence, whereas in a tracked state, the manager could retrieve these resources and reallocate to other newly detected objects.

A. States

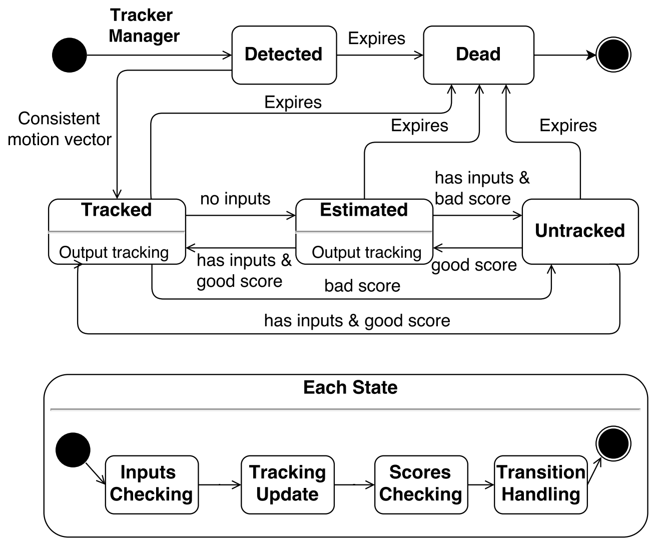

Fig. 3. State machine diagram. *has/no inputs* indicates whether enough inputs have been associated in the history window. *good/bad score* indicates whether enough good scores have been evaluated in the history window.

The state machine has 5 states, namely *Detected*, *Tracked*, *Estimated*, *Untracked*, and *Dead*. A tracker enters *Detected* state whenever a new detection cannot spawnedbe associated with an existing tracker. We use a tracker birth model to regulate the transition from *Detected* to *Tracked* state. *Tracked* and *Estimated* states are the two primary states where we output valid tracking results. Their main difference is that *Estimated* state lacks valid detection input, such as during temporary occlusion. At every time step, we evaluate a tracking score to indicate the quality of tracking. Whenever the score drops below a certain threshold, the tracker enters the *Untracked* state. If it remains *Untracked* for a set duration, it becomes *Dead* and will be removed. Fig. 3 shows a diagram of the state machine.

The tracker birth model is designed to provide a trade-off between response time to new targets and false alarm rate. First, detections are clustered and associated to existing trackers. All unassociated clusters become newly detected trackers and can be associated with future detections by spatial proximity. Over time, these *Detected* trackers will each contain a variable number of detections within a time window. The key is to find *consistent motion evidence* within these detections and only promote those tracker's states to *Tracked*. To estimate motion evidence, we take the first and last detection in the time window and compute the relative motion vector, which is simply the vector pointing from the first detection to the last detection. We call this our reference motion vector $\mathbf{V}_{ref}$ and resulting velocity v_{ref}. Using it as a reference, we count the number of detections *consistent* with $\mathbf{V}_{ref}$. A detection is consistent with $\mathbf{V}_{ref}$ if it lies close to the path of $\mathbf{V}_{ref}$ and its resulting velocity is also close to v_{ref}. These criteria are controlled by threshold parameters. If we find sufficient number (also a parameter) of consistent detections, we confirm this as a valid tracker and move it to *Tracked* state. Otherwise, we continue monitoring these candidate trackers but only maintain a fixed time window (set to $1s$ in our algorithm).

B. State Transition

The transition between states is determined by both the quality of input signals and particle filter evaluation scores. A circular buffer is used for storing the history of past inputs and scores. A transition is triggered if the average quality of past inputs and scores exceed the thresholds. By tuning the length of the buffer and the threshold of transition, we could greatly reduce the number of false positive and false negative detections. A third transition condition, which is a time-out mechanism, is also used to determine the lifetime of a tracker. If a tracker has not received any inputs for a set period of time, it will go directly to $Dead$ state. The lifetime value varies in different state so that a well tracked tracker lasts longer and a poorly tracked tracker dies quickly.

In addition to estimating the state vector S, we also need to estimate the discrete state s at each time step. Since our state machine is completely deterministic given the sensor input, the state transition can easily be represented as a function:

$$s_t = F(s_{t-1}) \tag{2}$$

C. Input and output

The output of a tracker is basically a smoothed result of the particle filter estimation. In other words, when $s \in \{$*Tracked*, *Estimated*$\}$,

$$S_t = w_s \cdot \overline{S_{particles}} + (1 - w_s) \cdot S_{t-1} \tag{3}$$

Here $\overline{S_{particles}}$ represents the average state of all valid particles. The update weight w_s is different for different states. In *Tracked* state, it is set to be high as the estimation is reliable. In *Estimated* state it is set to be low to eliminate the noise.

The aforementioned blind spot problem is now solved by the state machine. As shown in Fig. 4, when the vehicle is inside the blind spot, the tracker transitions to the *Estimated* state and keeps estimating the trajectory of the vehicle. When

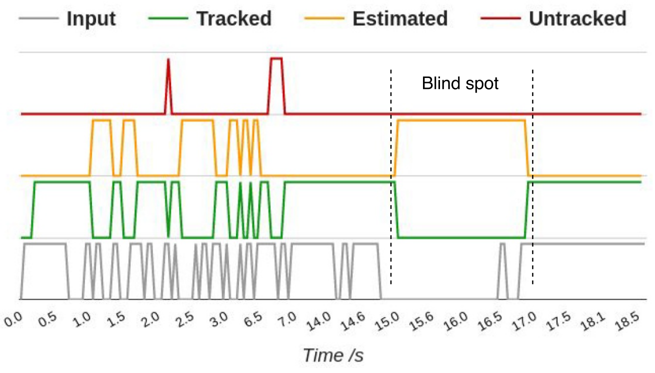

Fig. 4. Timing diagram of state transitions. Target vehicle is driving along a straight line with constant velocity. It passes through a sensor blind spot for 2s.

the vehicle reappears in the FOV, the tracker successfully continues tracking the vehicle.

The use of state machines provides several advantages over bare particle filter. First of all, it introduces hysteresis between neighboring states therefore the ease of transition from one state to another can be controlled by setting proper hysteresis thresholds. Secondly, the tracking quality is now directly monitored by the state machine. By exposing tracking quality to the outside, higher-level tracker manager can make intelligent decision on-the-fly to improve tracking quality, e.g. changing filter parameters, changing filter type, etc.

IV. SIMULATOR

To evaluate the tracking performance of vehicle tracker, most existing work rely on cost-expensive real sensor data collected in the field [9]. Although data collection could satisfy most of the simple traffic situation like highway car following, the effort to acquire a wide variety of traffic situations in urban environment is unbearable. We believe the best approach to develop and verify the tracking algorithm is to use a combination of simulated and real data. Using a simulator, it is easy to control the exact traffic scenario, sensor characteristics, noise and the environment variables, etc. Therefore, we can use simulators to exhaustively test all parts of the tracking system to achieve a higher level of confidence.

The simulator we used is built with ROS and Gazebo framework. The following sections will discuss the structures related to the vehicle tracking in three parts.

A. Sensor Model

We adopted the vehicle reflection model approach from [10] to generate the radar target lists. This approach is computationally less demanding than ray tracing and approximate to the real radar data. We also developed an interface to generate the same data format that is used for publishing Delphi ESR radar.

To make the generated radar data more realistic, we also modeled the sensor noise as independent Gaussian random variables on the measured ranges (ρ), range rates ($\dot{\rho}$) and bearing angles (ϕ). Their mean values are assumed to be zero and their standard deviations are obtained from ESR datasheet. In reality it is possible to miss radar returns and we model it using a *drop rate* of 0.3. That means we drop each radar ray return signal with 0.3 probability.

B. Agent Model

A vehicle agent is essentially a simplified vehicle model that follows a designated trajectory. Vehicle agents run in the Gazebo using the built-in physical engine and publishes their ground truths for benchmarking. In our simulation, we created three types of trajectories (Table II) with different number of agent vehicles. All agent vehicles move according to constant velocity and yaw rate model.

C. Simulation and Parameter Tuning

A script system is built to automate the simulation and parameter tuning process. The script could load a specific test scene that has several vehicle agents driving with the predefined trajectory. And it initializes the vehicle tracker in each run and logs its detection result and the ground truth from the simulator. Then a post process script is used to evaluate the result and visualize the data.

V. EXPERIMENTAL RESULTS

We used two sets of data to evaluate the performance of our tracking system: simulated and real. For simulated data, we used three types of target vehicle pattern: straight line, circle and figure eight. We varied the number of target vehicles to test our multiple target tracking capability while all test vehicles maintain constant velocity of 10m/s. Every test was repeated 100 times to calculate its RMSE. In the real case, we collect sensor data at a busy intersection from cross traffic vehicles. We provide two videos showing visual tracking results at `https://youtu.be/2I-YNL3qH7U` (simulated four cars figure 8 pattern) and `https://youtu.be/4mJFtqJUnsM` (real traffic data).

TABLE II

SIMULATION SETUP FOR MULTIPLE TARGET VEHICLE TRACKING

Scenario	Number of target vehicles	Course length (m)	Time gap between consecutive vehicles (s)
line	3	150	3
circle	3	314	4
figure 4	4	377	4

Our goal is to estimate the 2D pose $[x, y, \theta, v, \omega]$ of target vehicles. The position of the vehicle (x, y) is defined as the center of the rear axle. Ground truth for simulated data was generated by the simulator. Ground truth for real data was manually annotated from lidar scans collected by on-board lidar sensors.

As shown from Fig. 5 and 6, our trackers estimates generally follow the ground truth very closely, even when there are multiple targets present. This is quite challenging since in the scenario of circle and figure eight, there are frequent occlusions among target vehicles. In the case of straight line, the target vehicle pass through the radar blind zone for $2s$, and our tracker is still able to maintain good tracking throughout the process.

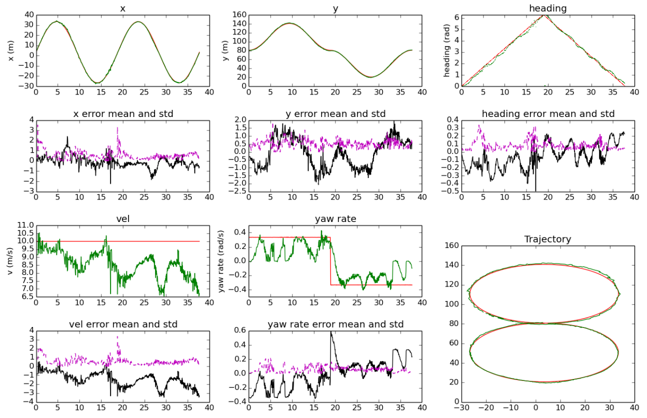

Fig. 5. Tracking performance on the figure-8 test course with 4 test vehicles. All x-axes are simulation time (s). Evaluated metrics are x, y, θ, v, ω and trajectory. The curves represent ground truth (red), tracking result (green), mean tracking error (magenta) and standard deviation (black).

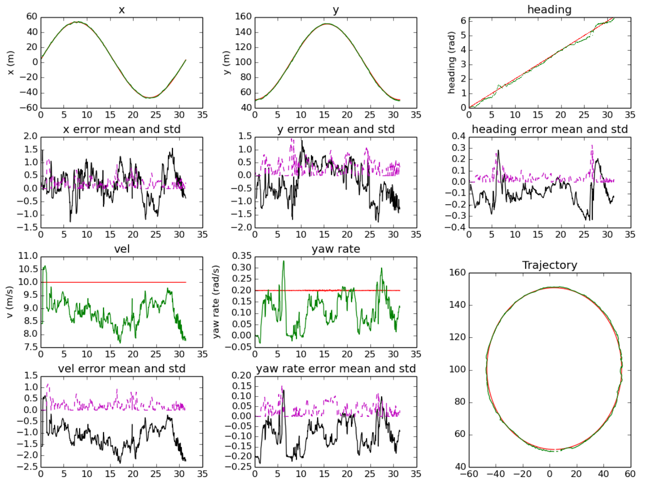

Fig. 6. Tracking performance on the circle test course with 3 test vehicles. All x-axes are simulation time (s).

Table III shows a summary of of overall tracking performance across different scenarios. Here track rate means the percentage of time a target vehicle is successfully tracked. We define successful tracking as positional error below $2.5m$ and angular error below $0.5rad$. As shown in the table, regardless of using state machine or not, the overall positional error and angular error are quite low while velocity exhibits higher error. However, there is significant track rate improvement while using the state machine, especially in figure 8 pattern. This is due to the more frequent occlusion in this scenario. In addition, velocity error is also much lower when we use the state machine. The track rate improvement is less pronounced in circle and line patterns because there is less occlusion, but our state machine-based approach consistently outperforms the baseline particle filter.

TABLE III

COMPARISON OF TRACKING RMSE BETWEEN OUR PROPOSED METHOD (WITH STATE MACHINE) AND BASELINE (WITHOUT) ON SIMULATED DATA

Scenario	x (m)	y (m)	θ (rad)	v (m/s)	ω (rad/s)	Track rate
line (baseline)	0.80	0.89	0.18	1.17	0.06	91.4%
circle (baseline)	0.76	0.85	0.16	1.49	0.13	94.4%
figure 8 (baseline)	0.74	1.10	0.19	2.40	0.23	71.2%
line (ours)	0.78	0.81	0.17	0.83	0.06	**92.3%**
circle (ours)	0.73	0.78	0.16	1.09	0.13	**97.3%**
figure 8 (ours)	0.70	0.99	0.20	1.70	0.22	**78.1%**

TABLE IV

COMPARISON OF TRACKING RMSE BETWEEN OUR PROPOSED METHOD (WITH STATE MACHINE) AND BASELINE (WITHOUT) ON REAL DATA

Scenario	x (m)	y (m)	Track rate
baseline	0.75	1.28	82.3%
ours	0.70	1.27	**84.2%**

To further evaluate the performance of our approach, we also collected real sensor data from a busy T-junction in an urban environment (7. We manually annotated the vehicle positions from lidar scan data, which give the most accurate position information. Due to the sparsity of the lidar data at long distance, we limit annotation region to $200m$ in cross traffic direction centered on ego car. The duration of the test was $159s$ and a total of 30 vehicles were annotated. Only (x, y) positions of vehicle centers were annotated because velocity and yaw angle cannot be annotated with sufficient accuracy. Since particle filter has run-to-run variation, we ran our tracking algorithm on the recorded data for 10 times and calculate the average. The tracking results are shown in Table IV.

On the real dataset, the tracking performance of our state machine approach only slightly outperforms the baseline. This is mostly due to: 1) there were few occlusions among target vehicles 2) the vehicle trajectories were mostly straight lines. However, this experiment conveyed the fact that our proposed algorithm is applicable to real world sensor data without performance degradation.

VI. CONCLUSIONS

In this paper we introduced a novel state machine based multi-vehicle tracker using radars. It has the advantage of overcoming inferior sensor measurements, occlusion and blind zone. It also tracks multiple targets reliably. The framework can be easily extended to support other types of sensors and tracking algorithms. The results were validated both using real sensor data and in simulation. As future improvement, we plan to incorporate lidars and improve our sensor fusion algorithm.

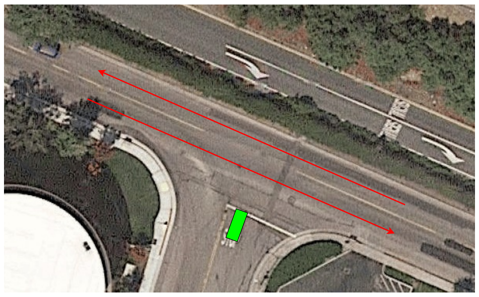

Fig. 7. Satellite image of the intersection used for real traffic experiment, showing the cross traffic direction and ego car position. Image source: Google Maps.

REFERENCES

[1] C. Knill, A. Scheel, and K. Dietmayer, "A direct scattering model for tracking vehicles with high-resolution radars," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 298–303.

[2] H. Cho, Y. W. Seo, B. V. K. V. Kumar, and R. R. Rajkumar, "A multi-sensor fusion system for moving object detection and tracking in urban driving environments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1836–1843.

[3] D. Ghring, M. Wang, M. Schnrmacher, and T. Ganjineh, "Radar/lidar sensor fusion for car-following on highways," in *The 5th International Conference on Automation, Robotics and Applications*, Dec 2011, pp. 407–412.

[4] M. Darms, P. E. Rybski, and C. Urmson, "An adaptive model switching approach for a multisensor tracking system used for autonomous driving in an urban environment," 2007.

[5] N. Schneider and D. M. Gavrila, *Pedestrian Path Prediction with Recursive Bayesian Filters: A Comparative Study*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 174–183.

[6] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference Computer Vision (ECCV)*, 2016.

[7] A. Cosgun, L. Ma, J. Chiu, J. Huang, M. Demir, A. M. Aon, T. Lian, H. Tafish, and S. Al-Stouhi, "Towards full automated drive in urban environments: A demonstration in gomentum station, california," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1811–1818.

[8] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, April 1993.

[9] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Autonomous Robots*, vol. 26, no. 2, pp. 123–139, Apr 2009.

[10] M. Buhren and B. Yang, "Simulation of automotive radar target lists using a novel approach of object representation," in *2006 IEEE Intelligent Vehicles Symposium*, 2006, pp. 314–319.