Design and Verification of Algorithms for Object Detection and Tracking Using Lidar Data

Marco Roggero[1], Prashant Arora[2], Gael Goron[2], Anand Raja[2], Elad Kivelevitch[2]
1: The MathWorks GmbH, Aachen, Germany
2: The MathWorks Inc., Natick (MA), USA

Abstract*—Lidar sensors are essential for autonomous vehicles and most ground moving robots. In their short scan cycles, these sensors generate a large number of points containing information that can be used to detect obstacles in the surrounding environment. While the process of extracting information, detecting and tracking relevant objects, and filtering noise or road reflections is complex, it needs to be reliable and accurate. In this work, we explain how to preprocess raw point clouds from lidar sensors in MATLAB® to generate detections for conventional trackers that assume one detection per object per sensor scan. We then define a cuboid model to describe kinematics, dimensions, and measurements of extended objects being tracked with a joint probabilistic data association (JPDA) tracker and use an interacting multiple model (IMM) filter. Finally, we mention how to generate C code from the algorithm and verify execution results.

***Keywords*—lidar; point cloud; detection; tracking; autonomous driving**

I. INTRODUCTION

This paper shows how to track vehicles using measurements from a lidar sensor mounted on top of an autonomous vehicle, referred to as the ego vehicle. The reported example illustrates the workflow in MATLAB® for processing point clouds and tracking objects. The lidar data used in this example is recorded from a highway driving scenario.

Due to the high resolution of lidars, each scan contains a large number of 3D points, commonly known as a point cloud. This data must be preprocessed to extract objects of interest, such as cars, cyclists, and pedestrians, commonly referred to as obstacles. Point cloud regions belonging to obstacles are clustered and each cluster is converted to a bounding box detection. Detected objects are then tracked with a joint probabilistic data association (JPDA) tracker and an interacting multiple model (IMM) filter. Both the JPDA[1,2,3] tracker and the IMM filter[1] are available as part of Sensor Fusion and Tracking Toolbox™[4].

This work is organized as follows: in the next section, we describe how lidar data is processed to obtain bounding box detections. In section 3, we describe the models, filter, and tracker used to track the object bounding boxes, followed by results. Finally, section 6 provides a summary and future directions.

II. LIDAR DATA PROCESSING

A. Reading Lidar Data

The lidar data used in this example was recorded using a Velodyne® HDL32E[5] sensor in PCAP format[6]. Each scan is stored as a 3D point cloud. Efficiently processing this data using fast indexing and search is key to the performance of the sensor processing pipeline. To achieve this efficiency a MATLAB `pointCloud` object is used, which internally organizes the data using a K-d tree data structure. The `velodyneFileReader` constructs the organized `pointCloud` for each lidar scan. The `Location` property of the `pointCloud` is an M-by-N-by-3 matrix containing the X, Y, and Z coordinates of points in meters. The point intensities are stored in the `Intensity` property of the `pointCloud` object.

To visualize streaming point cloud data, we have used the MATLAB object named `pcplayer`[7] and specified the region around the vehicle to be displayed.

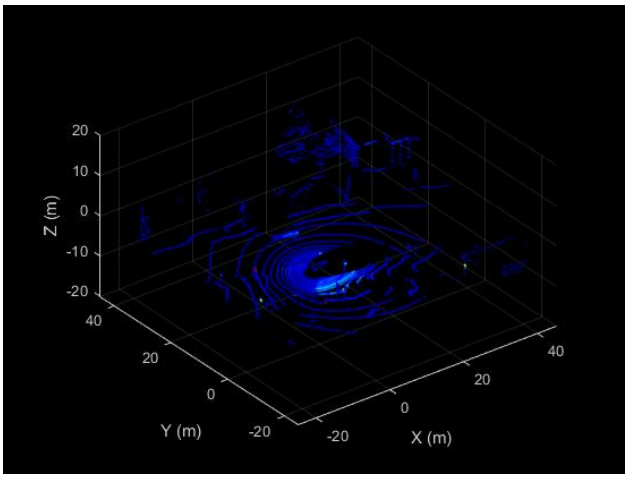

Fig 1: Visualization of one frame of the lidar point cloud.

Next, we describe how to segment points belonging to the ground plane, the ego vehicle, and nearby obstacles.

B. Segmenting the Ego Vehicle

Since the lidar is mounted on top of the vehicle, the point cloud may contain points belonging to the vehicle itself, such as on the roof or hood. To segment these points, it is necessary to know the dimensions of the vehicle and the sensor's mounting position in vehicle coordinates.

Results for one frame of data are shown in Figure 2.

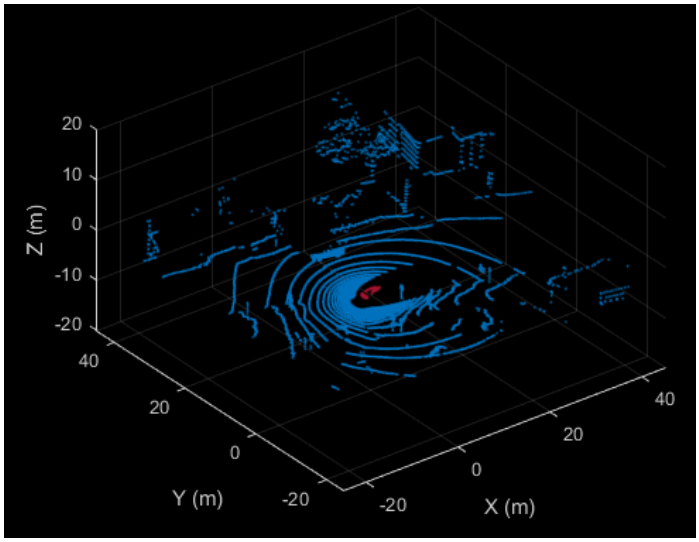

Fig 2: Reflection points generated by the ego vehicle. Knowing the vehicle dimensions and sensor position makes this segmentation possible.

C. Segmenting Ground Plane and Nearby Obstacles

To identify obstacles from the lidar data, we first segment points belonging to the ground plane using the `segmentGroundFromLidarData`[8] function. This function operates on the range image to label ground points using a connected components algorithm [11]. The result is shown in Figure 3: different colors are used for points generated from road reflections and the ego vehicle, and for still-unlabeled points that are potentially generated from dynamic objects, such as vehicles, pedestrians, and bicycles, or static objects, such as trees and buildings.

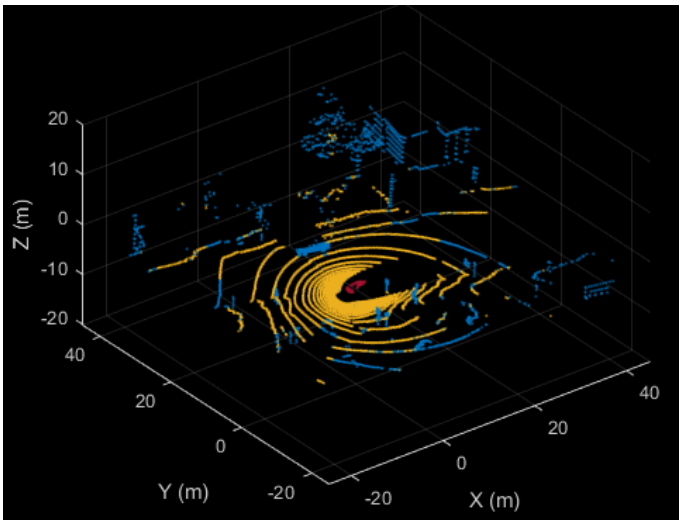

Fig 3: Different colors used for road reflections, the ego vehicle, and still-unlabeled points.

The next step consists of segmenting nearby obstacles by looking for all points that are not part of the ground or ego vehicle within regions of interest around the ego vehicle. This region of interest can be determined online based on the number of lanes on the highway or the maximum range of the sensor.

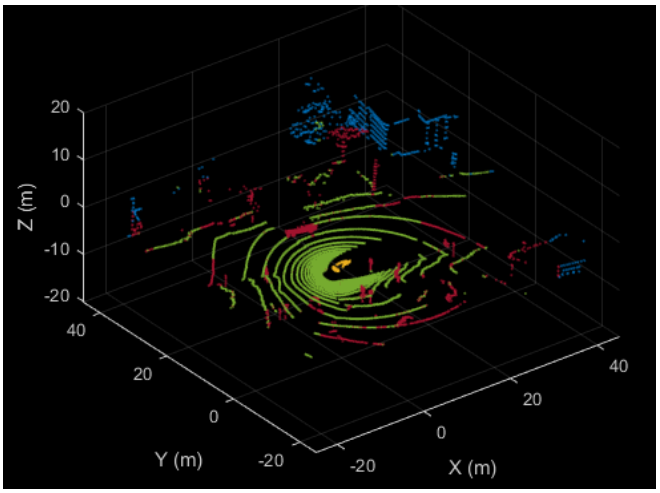

Fig 4: Nearby obstacles marked with red.

Once the point cloud processing pipeline for a single lidar scan is defined, it can be applied at each step to the scans obtained while driving or to the entire sequence of recorded data.

D. Detecting and Marking Relevant Objects

Points belonging to obstacles within a radius of interest from the ego vehicle can be clustered and marked with 3D bounding boxes containing the detected object and other information required by a tracker; for example, time of detection and uncertainty of the measurement. This type of structured information is necessary to feed tracking algorithms such as those that we describe in the next section.

III. MULTI-OBJECT TRACKING

A. Target State and Sensor Measurement Model

The first step in tracking an object is defining its state, and the models that define both the transition of the state and the corresponding measurement from the space. These two sets of equations are collectively known as the state-space model of the target. To model the state of vehicles for tracking using lidar, we use a cuboid model with the following convention:

$$x = [x_{kin}\ \theta\ l\ w\ h\ ]$$

where x_{kin} represents the motion of the target center with respect to the ego vehicle, θ represents the yaw angle, and $l, w,$ and h represent the length, width, and height of the target, respectively. We have used two state-space models: a constant velocity (cv) cuboid model and a constant turn-rate (ct) cuboid model. These models differ in the way they define the kinematic part of the state, as described below:

$$x_{cv} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \ \theta\ l\ w\ h]$$

$$x_{ct} = [x\ \dot{x}\ y\ \dot{y}\ \dot{\theta}\ z\ \dot{z}\ \ \theta\ l\ w\ h]$$

Decoupling the orientation of the bounding box, θ, and 2D velocity components, $\dot{x}$ and $\dot{y}$, assists in tracking objects in the ego vehicle frame, as the orientation of the box is not necessarily aligned with the vehicle motion.

Measurement models are used to describe how the sensor perceives the constant velocity and constant turn-rate states, respectively, and they return bounding box measurements. Because the state contains information about the size of the target, the measurement model includes the effect of center-point offset and bounding box shrinkage, as perceived by the sensor, due to effects such as self-occlusion [12]. This effect is modeled by a shrinkage factor that is directly proportional to the distance from the tracked vehicle to the sensor.

B. Setting Up the Tracker

The image below shows the complete workflow to obtain a list of tracks from a `pointCloud` input.

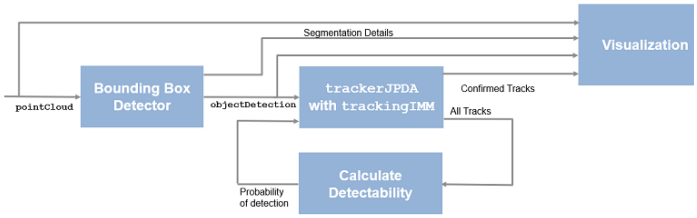

Fig 5: Workflow used to obtain tracks from the point cloud.

A joint probabilistic data association tracker (`trackerJPDA`[9]) coupled with an IMM filter (`trackingIMM`[10]) is used to track objects in this case. The IMM filter is configured to use the constant velocity and constant turn-rate models defined in the previous section. The IMM approach helps a track to switch between motion models and thus achieve good estimation accuracy during events such as maneuvering or lane changing.

IV. VISUALIZING THE RESULTS

The visualization is organized into three panels:

1. The left panel displays lidar preprocessing and tracking and shows the raw point cloud, segmented ground, and obstacles. It also shows the resulting detections from the detector model and the tracks of vehicles generated by the tracker.
2. The top-right panel displays the ego vehicle's 2D bird's-eye view of the scenario. It shows the obstacle point cloud, bounding box detections, and tracks generated by the tracker. For reference, it also displays the image recorded from a camera mounted on the ego vehicle and its field of view.
3. The bottom-right panel displays the tracking details and shows the scenario zoomed around the ego vehicle. It also shows finer tracking details, such as error covariance in estimated position of each track and its motion model probabilities, denoted by cv and ct.

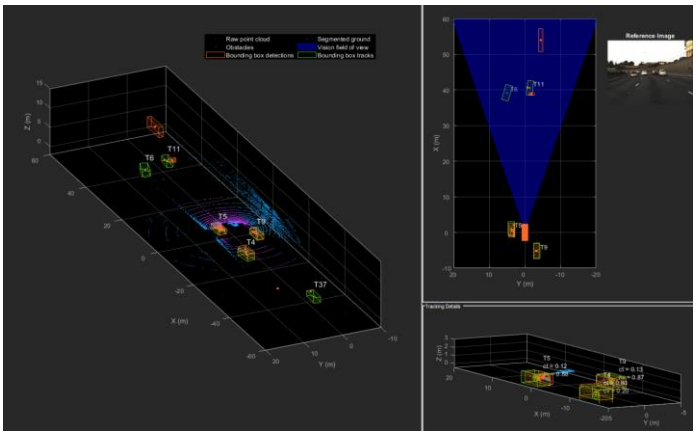

Fig 6: Lidar preprocessing and tracking, ego vehicle display, and tracking details.

Figure 6 shows the three panels. The tracks are represented by green bounding boxes. The bounding box detections are represented by orange bounding boxes. The detections also have orange points inside them, representing the point cloud segmented as obstacles. The segmented ground is shown in purple. The cropped or discarded point cloud is shown in blue.

A. C Code Generation

Once the algorithms for point segmentation, object detection, and tracking have been developed, fine-tuned, and tested, it is time to convert these algorithms into C code with MATLAB Coder™.

The number of confirmed tracks is the same for MATLAB and MEX code execution. This assures that the lidar preprocessing and tracking algorithm returns the same results with generated C code as with the MATLAB code.

V. Results

In this section, we discuss the ability of the combination of the lidar measurement model, joint probabilistic data association, and interacting multiple model filter to achieve a good estimation of the vehicle tracks. Videos and animations showing phenomena explained in the next three paragraphs are available on mathworks.com [13].

A. Track Maintenance

While most tracks maintain their IDs and trajectory also if the vehicle is not detected for a short time, some tracks can be lost when the tracked vehicle is missed by the sensor for a long time. We have also observed that the tracked objects are able to maintain their shape and kinematic center by positioning the detections onto the visible portions of the vehicles. For example, as some tracks move forward, bounding box detections start to fall on their visible rear portion and the track maintains the actual size of the vehicle.

B. Capturing Maneuvers

Using an IMM filter helps the tracker to maintain tracks on maneuvering vehicles. When a vehicle changes lane, the tracker is able to maintain a track on the vehicle during this maneuvering event. During this event, the probability of following the constant turn-rate model by the track increases.

C. Joint Probabilistic Data Association

Using a joint probabilistic data association tracker helps in maintaining tracks during ambiguous situations where vehicles have a low probability of detection due to their large distance from the sensor. The tracker can maintain tracks during events when one of the vehicles is not detected. Typical for the JPDA algorithm is that two tracks that are next to each other and at a large distance from the ego vehicle coalesce when one vehicle is not detected but separate quickly as soon as the vehicle is detected again.

VI. Summary and Future Directions

We have shown first how a raw point cloud can be preprocessed to generate detections for conventional trackers. In the second part of the paper, we have shown how to use the JPDA tracker with an IMM filter to track objects using the lidar sensor and generate C code from the algorithm and verify execution results.

There are two main directions in which this work can be extended. The first direction is to fuse the lidar tracks from this work with tracks coming from other sensor modalities; for example vision and radar sensors. Having multiple sensor modalities can improve tracking by providing complimentary data and resilience to weather conditions. The other direction is to use the code that was automatically generated to rapidly prototype a tracking solution for an autonomous vehicle.

References

[1] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, Artech House, 1999.

[2] D. Musicki, "Joint Integrated Probabilistic Data Association: JIPDA," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, 2004, pp. 1093-1099,.

[3] Y. Bar-Shalom, P. K. Willett, and X. Tian, *Tracking and Data Fusion: A Handbook of Algorithms*, YBS Publishing, 2011.

[4] Elad H. Kivelevitch, Greg Dionne, Trevor Roose, Prashant Arora, Brian Fanous, Ryan Salvo, Vincent Pellissier, Ric Losada, and Rick Gentile, "Sensor Fusion Tools in Support of Autonomous Systems," AIAA Scitech 2019 Forum. January 2019.

[5] https://velodynelidar.com/hdl-32e.html

[6] https://github.com/hokiespurs/velodyne-copter/wiki/PCAP-format

[7] https://mathworks.com/help/vision/ref/pcplayer.html

[8] https://de.mathworks.com/help/vision/ref/segmentgroundfromlidardata.html

[9] https://www.mathworks.com/help/fusion/ref/trackerjpda-system-object.html

[10] https://www.mathworks.com/help/fusion/ref/trackingimm.html

[11] Bogoslavski, I. "Efficient Online Segmentation for Sparse 3D Laser Scans." *Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 85, number 1, 2017, pp. 41-52.

[12] Arya Senna Abdul Rachman, Arya. "3D-LIDAR Multi Object Tracking for Autonomous Driving: Multi-target Detection and Tracking under Urban Road Uncertainties." (2017).

[13] https://de.mathworks.com/help/driving/examples/track-vehicles-using-lidar.html