Argoverse: 3D Tracking and Forecasting with Rich Maps Supplementary Material

Ming-Fang Chang*[1,2], John Lambert*[1,3], Patsorn Sangkloy*[1,3], Jagjeet Singh*[1], Sławomir Bąk[1], Andrew Hartnett[1], De Wang[1], Peter Carr[1], Simon Lucey[1,2], Deva Ramanan[1,2], and James Hays[1,3]

[1]Argo AI, [2]CMU, [3]Georgia Institute of Technology

Abstract

In this supplementary material, we present additional details about our map (Section 1), our trajectory mining (Section 2), and our 3D tracking algorithm (Section 3).

1. Supplemental Map Details

In this section, we describe details of our map coordinate system and the functions exposed by our map API, and we visualize several semantic attributes of our vector map. Our map covers 204 linear kilometers of lane centerlines in Miami and 86 linear kilometers in Pittsburgh. In terms of driveable area, our map covers 788,510 m^2 in Miami and 286,104 m^2 in Pittsburgh.

1.1. Coordinate System

The model of the world that we subscribe to within our map and dataset is a local tangent plane centered at a central point located within each city. This model has a flat earth assumption which is approximately correct at the scale of a city. Thus, we provide map object pose values in city coordinates. City coordinates can be converted to the UTM (Universal Transverse Mercator) coordinate system by simply adding the city's origin in UTM coordinates to the object's city coordinate pose. The UTM model divides the earth into 60 flattened, narrow zones, each of width 6 degrees of longitude. Each zone is segmented into 20 latitude bands.

We favor a city-level coordinate system because of its high degree of interpretability when compared with geocentric reference coordinate systems such as the 1984 World Geodetic System (WGS84). While WGS84 is widely used by the Global Positioning System, the model is difficult to interpret at a city-scale; because its coordinate origin is located at the Earth's center of mass, travel across an entire city corresponds only to pose value changes in the hundredth decimal place. The conversion back and forth between UTM and WGS84 is well-known and is documented in detail in [13].

We provide ground-truth object pose data in the egovehicle frame, meaning a single SE(3) transform is required to bring points into the city frame for alignment with the map:

$$p_{city} = ({}^{city}T_{egovehicle})\,(p_{egovehicle})$$

Figure 1 shows examples of the centerlines which are the basis of our vector map. Centerline attributes include whether or not lane segments are in an intersection, and which lane segments constitute their predecessors and successors.

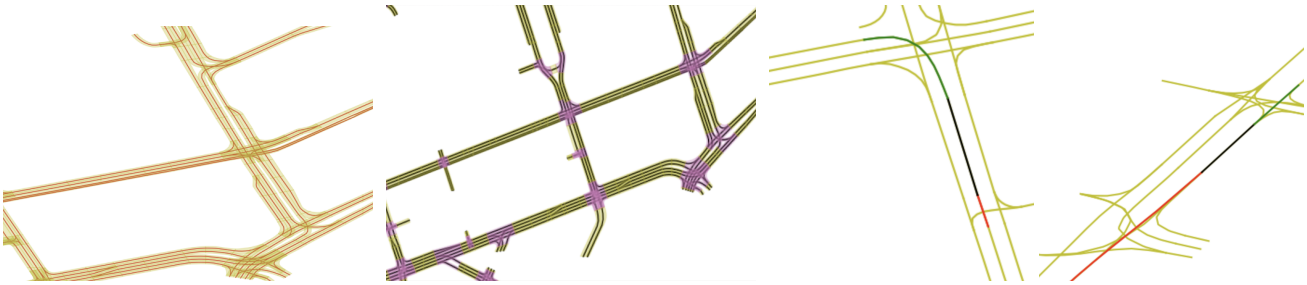
1.2. Map API and Software Development Kit

The dataset's rich maps are our most significant contribution and we aim to make it easy to develop computer vision tools that leverage the map data. Figure 3 describes several functions which we hope will make it easier for researchers to access the map. Our API is provided in Python. For example, our API can provide rasterized bird's eye view (BEV) images of the map around the egovehicle, extending up to 100 m in all directions. It can also provide a dense 1 meter resolution grid of the ground surface, especially useful for ground classification when globally planar ground surface assumptions are violated (see Figure 4).

These dense, pixel-level map renderings, similar to visualizations of instance-level or semantic segmentation [3], have recently been demonstrated to improve 3d perception and are relatively easy to use as an input to a convolutional network [14, 2].

We provide our vector map data in a modified OpenStreetMap (OSM) format, i.e. consisting of "Nodes" (waypoints) composed into "Ways" (polylines) so that the community can take advantage of open source mapping tools built to handle OSM formats. The data we provide is richer

*Equal contribution.



(a) (b) (c)

Figure 1: (a) Lane centerlines and hallucinated area are shown in red and yellow, respectively. We provide *lane* centerlines in our dataset because simple *road* centerline representations cannot handle the highly complicated nature of real world mapping, as shown above with divided roads. (b) We show lane segments within intersections in pink, and all other lane segments in yellow. Black shows lane centerlines. (c) Example of a specific lane centerline's successors and predecessors. Red shows the predecessor, green shows the successor, and black indicates the centerline segment of interest.

Figure 2: **Ring Camera Examples.** Scenes captured in Miami, Florida, USA (top) and Pittsburgh, Pennsylvania, USA (bottom) with our ring camera. Each row consists of 7 camera views with overlapping fields of view. Camera order is *rear_left, side_left, front_left, front_center, front_right, side_right, rear_right*

than existing OSM data which does not contain per-lane or elevation information.

2. Supplemental Details on Mined Trajectories for Forecasting

In this section, we describe our approach for mining data for trajectory forecasting. The scenarios challenging for a forecasting task are rare but with a vector map, they are easy to identify. We focus on some specific behavioral scenarios from over 1006 driving hours. For every 5 second sequence, we assign an *interesting* score to every track in that sequence. A high *interesting* score can be attributed to one or more of the following cases wherein the track is: at an intersection with or without traffic control, on a right turn lane, on a left turn lane, changing lanes to a left or right neighbor, having high median velocity, having high variance in velocity and visible for a longer duration. We give more importance to changing lanes and left/right turns because these scenarios are very rare. If there are at least 2 sufficiently important tracks in the sequence, we save the sequence for forecasting experiments. Further, the track which has the maximum *interesting* score and is visible through out the sequence is tagged as the *Agent*. The forecasting task is then to predict the trajectory of this particular track, where all the other tracks in the sequence can be used for learning social context for the *Agent*. There is also a 2.5 secs overlap between 2 consecutive sequences. This means the same track id can be available in 2 sequences, albeit with different trajectories.

3. Supplemental Tracking Details

In this section, we describe our tracking pipeline in greater detail.

3.1. Tracker Implementation Details

Because of space constraints we could not fit all details of our 3D tracking pipeline in the main paper. We do not claim any novelty for this 'baseline' tracker, but it works reasonably well, especially with map information to make the task easier (e.g. driveable area, ground height, and lane information). Our tracker tracks the position and velocity of surrounding vehicles from LiDAR data. The tracking pipeline has the following stages:

1. Segmentation and Detection. In order to segment a point cloud into distinct object instances, we exploit the complementary nature of our two sensor modalities. First,

Function name	Description
`remove_non_driveable_area_points`	Use rasterized driveable area ROI to decimate LiDAR point cloud to only ROI points.
`remove_ground_surface`	Remove all 3D points within 30 cm of the ground surface.
`get_ground_height_at_xy`	Get ground height at provided (x,y) coordinates.
`render_local_map_bev_cv2`	Render a Bird's Eye View (BEV) in OpenCV.
`render_local_map_bev_mpl`	Render a Bird's Eye View (BEV) in Matplotlib.
`get_nearest_centerline`	Retrieve nearest lane centerline polyline.
`get_lane_direction`	Retrieve most probable tangent vector $\in \mathbb{R}^2$ to lane centerline.
`get_semantic_label_of_lane`	Provide boolean values regarding the lane segment, including *is_intersection turn_direction*, and *has_traffic_control*.
`get_lane_ids_in_xy_bbox`	Get all lane IDs within a Manhattan distance search radius in the xy plane.
`get_lane_segment_predecessor_ids`	Retrieve all lane IDs with an incoming edge into the query lane segment in the semantic graph.
`get_lane_segment_successor_ids`	Retrieve all lane IDs with an outgoing edge from the query lane segment.
`get_lane_segment_adjacent_ids`	Retrieve all lane segment IDs of that serve as left/right neighbors to the query lane segment.
`get_lane_segment_centerline`	Retrieve polyline coordinates of query lane segment ID.
`get_lane_segment_polygon`	Hallucinate a lane polygon based around a centerline using avg. lane width.
`get_lane_segments_containing_xy`	Use a "point-in-polygon" test to find lane IDs whose hallucinated lane polygons contain this (x, y) query point.

Figure 3: Example Python functions in the Argoverse map API.

Figure 4: A scene with non-planar ground surface. The colored LiDAR returns have been classified as ground based on the map. Points outside the driveable area are also discarded. This simple distance threshold against a map works well, even on the road to the left which goes steeply uphill.

using Mask R-CNN [5], we obtain object masks in pixel space and discard any LiDAR returns whose image projection does not fall within a mask. We then geometrically cluster the remaining 3D LiDAR point cloud into separate objects according to density, using DBSCAN [4].

Others have proposed compensating for point cloud undersegmentation and oversegmentation scenarios by conditioning on the data association and then jointly track and perform probabilistic segmentation [6]. We can avoid many such segmentation failures with the high precision of our Mask R-CNN network [1]. We also eliminate the need for an object's full, pixel-colored 3D shape during tracking, as others have suggested [8, 7]. We prefer density-based clustering to connected components clustering in a 2D occupancy grid [9, 10] because the latter approach discards information along the z-axis, often rendering the method brittle.

To help focus our attention to areas that are important for a self driving car, we only consider points within the region of interest (ROI) defined by the driveable area map, and not on the ground.

While segmentation provides us a set of points belonging to an object, we need to determine if this is an object of interest that we want to track. Unlike in image space, objects in a 3D have consistent sizes. We apply heuristics that enforce the shape and volume of a typical car and thereby identify vehicle objects to be tracked. We estimate the center of an object by fitting a smallest enclosing circle over the segment points.

2. Association. We utilize the Hungarian algorithm to obtain globally optimal assignment of previous tracks to currently detected segments where the cost of assignment is based on spatial distance. Typically, tracks are simply

[1] We use a public implementation available at https://github.com/facebookresearch/maskrcnn-benchmark.

assigned to their nearest neighbor in the next frame.

3. Tracking. We use ICP (Iterative Closest Point) from the Point Cloud Library [12] to estimate relative transformation between corresponding point segments for each track. Then we apply a Kalman Filter (KF) [7] with ICP results as the measurement and a static motion model (or constant velocity motion model, depending on the environment) to estimate vehicle poses for each tracked vehicle. We assign a fixed size bounding box for each tracked object. The KF state is comprised of both the 6 dof pose and velocity.

3.2. Tracking Evaluation Metrics

We use standard evaluation metrics commonly used for multiple object trackers (MOT) [11, 1]. The MOT metric relies on centroid distance as distance measure.

- **MOTA(Multi-Object Tracking Accuracy)**:

$$MOTA = 100 * (1 - \frac{\sum_t FN_t + FP_t + ID_{sw}}{\sum_t GT}) \quad (1)$$

  where FN_t, FP_t, ID_{sw}, GT denote number of false negative, false positives, number of ID switches, and ground truths. We report MOTA as percentages.

- **MOTP(Multi-Object Tracking Precision)**:

$$MOTP = \frac{\sum_{i,t} D_t^i}{\sum_t C_t} \quad (2)$$

  where C_t denotes the number of matches, and D_t^i denotes the distance of matches.

- **IDF1 (F1 score)**:

$$IDF1 = 2\frac{precision * recall}{precision + recall} \quad (3)$$

  Where $recall$ is the number of true positives over number of total ground truth labels. $precision$ is the number of true positives over sum of true positives and false positives.

- **MT (Mostly Tracked)**: the ratio of trajectories tracked more than 80% of its lifetime.

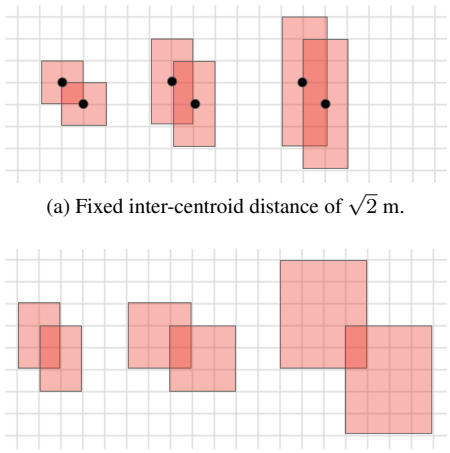
- **ML (Mostly Lost)**: the ratio of trajectories tracked less than 20% of its lifetime.

- **FP (False Positive)**: Total number of false positives

- **FN (False Negative)**: Total number of false negatives

- **IDsw (ID Switch)**: number of identified ID switches

- **Frag (Fragmentation)**: Total number of switches from "tracked" to "not tracked"



(a) Fixed inter-centroid distance of $\sqrt{2}$ m.

(b) Fixed 2D intersection area of 2 m^2.

Figure 5: We compare thresholding true positives (TP) and false positives (FP) of matched cuboid shapes using inter-centroid distance (above) versus using 2D/3D IoU (below). Above: fixed inter-centroid distance, from left to right: IoU values of 0.143, 0.231, 0.263. Below: fixed intersection area, from left to right, IoU values of 0.2, 0.125, 0.053.

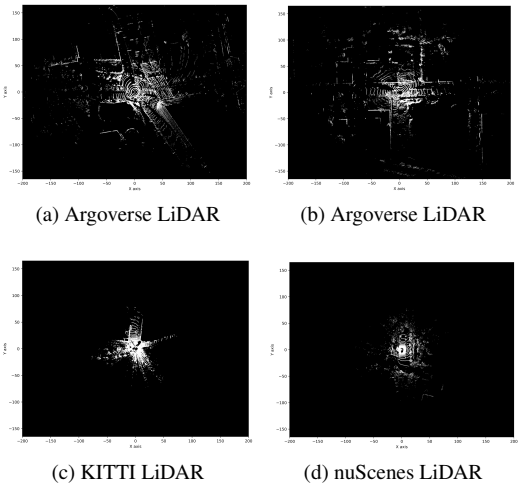
3.3. True Positive Thresholding Discussion

Intersection-over-Union (IoU) is designed as a scale invariant metric, meaning that doubling the size and relative overlap of two boxes will not change its value. However, we counter that 3d tracking evaluation should not be performed in a strictly scale invariant manner. *Absolute* error matters, especially in 3d. In 2d tasks (e.g. object detection) we have only pixels which could be any real world size, whereas in 3d we have absolute lengths. When using IOU as a TP/FP threshold, with fixed intersection area, IoU for larger vehicles is penalized unfairly (see Figure 5). On the other hand, with a fixed distance between associated centroids, the IoU increases with larger vehicles. In the LiDAR domain, these problems are exaggerated because the sampling density can be quite low, especially for distant objects. In 2d object detection, we rarely try to find objects that are 3 pixels in size, but small, distant objects frequently have 3 LiDAR returns and thus accurate determination of their spatial extent is difficult.

References

[1] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP J. Image and Video Processing*, 2008, 2008. 4

[2] S. Casas, W. Luo, and R. Urtasun. Intentnet: Learning to predict intention from raw sensor data. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of*



(a) Argoverse LiDAR (b) Argoverse LiDAR

(c) KITTI LiDAR (d) nuScenes LiDAR

Figure 6: Above: Sample LiDAR sweeps in the ego-vehicle frame, with marked x and y axes, with $x \in [-200, 200]$ and $y \in [-160, 160]$ for all plots. The Argoverse LiDAR has twice the range of the sensors used to collect the KITTI or nuScenes datasets, allowing us to observe more objects in each scene.

Machine Learning Research, pages 947–956. PMLR, 29–31 Oct 2018. 1

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[4] M. Ester, H. peter Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996. 3

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017. 3

[6] D. Held, D. Guillory, B. Rebsamen, S. Thrun, and S. Savarese. A probabilistic framework for real-time 3d segmentation using spatial, temporal, and semantic cues. In *Proceedings of Robotics: Science and Systems*, 2016. 3

[7] D. Held, J. Levinson, and S. Thrun. Precision tracking with sparse 3d and dense color 2d data. In *ICRA*, 2013. 3, 4

[8] D. Held, J. Levinson, S. Thrun, and S. Savarese. Combining 3d shape, color, and motion for robust anytime tracking. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014. 3

[9] M. Himmelsbach and H. Wünsche. Lidar-based 3d object perception. In *Proceedings of 1st International Workshop on Cognition for Technical Systems*, 2008. 3

[10] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. R. Pratt, M. Sokolsky, G. Stanek, D. M. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In *IEEE Intelligent Vehicles Symposium (IV), 2011, Baden-Baden, Germany, June 5-9, 2011*, pages 163–168, 2011. 3

[11] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, Mar. 2016. arXiv: 1603.00831. 4

[12] R. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1 –4, may 2011. 4

[13] J. P. Snyder. Map projections: A working manual. u.s. geological survey professional paper. page 61, 1987. 1

[14] B. Yang, M. Liang, and R. Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 29–31 Oct 2018. 1