Continuous-Time Estimation for Dynamic Obstacle Tracking

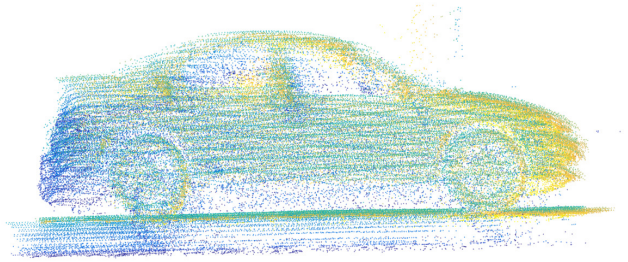
Arash K. Ushani, Nicholas Carlevaris-Bianco,
Alexander G. Cunningham, Enric Galceran, and Ryan M. Eustice

***Abstract*— This paper reports on a system for dynamic obstacle tracking for autonomous vehicles. In this work, we seek to simultaneously estimate *both* the trajectory of the obstacle and the obstacle's shape. These two tasks are inherently coupled—given only noisy partial views, one cannot accurately estimate the trajectory of an obstacle if its shape is unknown, nor can one estimate its shape without knowing its trajectory. To address this challenge, we note that simultaneous localization and mapping (SLAM), where a robot must build a map of the environment while localizing itself within the map, presents similar challenges. By treating the obstacle's shape as a "map" in the obstacle's moving reference frame, we can formulate the obstacle tracking and shape estimation similarly to SLAM. Additionally, we use a continuous time estimation framework to incorporate sensor data that is collected at a fast rate (e.g., light detection and ranging (LIDAR)). Using these methods, we are able to obtain smooth trajectories and crisp point clouds for tracked obstacles. We test our proposed tracker on real-world data collected by our autonomous vehicle platform and demonstrate that it produces improved results when compared to a standard centroid-based extended Kalman filter (EKF) tracker.**

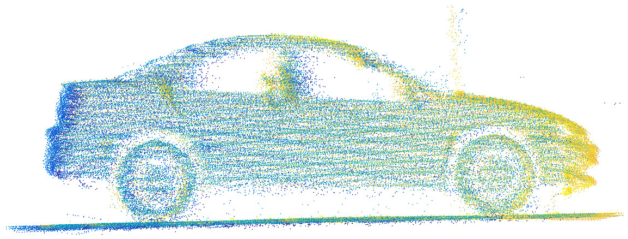
I. Introduction

As autonomous cars continue to develop, one important challenge is to be able to accurately track dynamic obstacles in the environment, such as other vehicles, bicycles, or pedestrians. Accurate estimates of obstacle positions and velocities are essential to any planning framework that seeks to create safe trajectories. Not only does a planner need to avoid obstacles in its environment, but often planners will also try to predict the future actions of vehicles given an accurate trajectory history [1, 2]. Additionally, a history of an obstacle's pose or its point cloud representation can be used to classify the obstacle [3, 4].

Obstacle trackers can be prone to errors and biases if they do not adequately model the shape of the obstacle being tracked. For example, consider an autonomous vehicle driving past a parked car while tracking it. At first, only the rear of the car is observed. As the parked car is passed, only the side is observed. Afterwards, only the front face is observed. During this process, if we do not maintain an estimate for the obstacle model, our tracker may falsely believe that this parked car has moved due to the change in our perspective. Errors such as this can have adverse effects for a planning system that relies on obstacle tracking to produce a safe plan for the car. To account for biases such as this, we can use an estimate of some model or representation of the obstacle's structure.



(a) Original point cloud estimate given by the baseline system



(b) Point cloud estimate given by our proposed method

Fig. 1: Our proposed system tracking a car. Points in the point cloud are colored by the time when they were observed, from blue to yellow. The crispness of the generated point cloud is reflective of the tracking accuracy. Best viewed in color.

Another challenge is how to incorporate high rate sensor data. Many current autonomous vehicles rely on one or more light detection and ranging (LIDAR) laser sensors that are collecting data at high rates, for example, collecting over 700,000 points per second. Many current trackers make an implicit assumption that a set of data or a discrete "snapshot" (i.e., set of LIDAR observations in succession collected from a single scan of the obstacle) is collected at a single point in time [5]. While this helps provide some known structure to the obstacle, this does not account for any movement of the obstacle while the snapshot is being collected, leaving these trackers prone to errors. This can be exacerbated when there are multiple sensors or different sensor modalities that are not synchronized. However, due to the fast rate of observations coming from these sensors, it quickly becomes intractable to try to compute the pose at each time associated with an observation. One method of handling sensors with fast data rates is to use continuous time estimation, as proposed by Furgale et al. [6] and further developed by Anderson and Barfoot [7] and Anderson et al. [8]. This allows us to represent the obstacle's path not as a discrete

*This work was supported by a grant from Ford Motor Company via the Ford-UM Alliance under award N015392.

A. Ushani, A. Cunningham, E. Galceran, and R. Eustice are with the University of Michigan, Ann Arbor, MI 48109, USA {aushani, alexgc, egalcera, eustice}@umich.edu.

N. Carlevaris-Bianco was with the University of Michigan, Ann Arbor, MI 48109, USA during the tenure of this work. nickcarlevaris@gmail.com.

set of poses, but as a linear combination of continuous basis functions, reducing the number of variables by several orders of magnitude.

In this work, we address the above issues of errors and biases by noting that the challenges inherent to accurate obstacle tracking as described above are similar to those in a typical simultaneous localization and mapping (SLAM) problem. There are a few differences between obstacle tracking and SLAM, however. For example, in obstacle tracking the "map" is in the reference frame of the obstacle. Additionally, we are estimating the obstacle's motion, rather than our own. Nonetheless, we will show that we can solve the obstacle tracking problem using a formulation similar to that of state of the art SLAM formulations. Specifically, our contributions are:

1) Modeling and solving of obstacle tracking using a formulation similar to SLAM.
2) Incorporating continuous-time estimation tools to handle fast rate sensors.
3) Evaluating this method on a real-world dataset.

We show that this approach leads to more accurate tracks on obstacle models compared to a standard centroid-based extended Kalman filter (EKF) tracker. We evaluate the performance in terms of tracking error and point cloud crispness on a real-world dataset we collected using an autonomous vehicle.

II. Related Work

A popular approach to LIDAR-based obstacle tracking is to extract some observation from each LIDAR snapshot and feed this measurement into a filtering framework such as a Kalman filter. This is often based on some geometric property of the observed snapshot, such as a bounding box [9] or a centroid [10].

Several obstacle trackers make use of some simple obstacle model. There are different ways of estimating this model. Petrovskaya and Thrun [11] use anchor points (such as the center or corner of the perceived obstacle) to estimate a geometric model in order to aid in tracking. Kaestner et al. [12] use a generative model to extract a bounding box from the obstacle. Darms et al. [13] model obstacles as points or as boxes, depending on the situation, and track the obstacles' state in a Kalman filter. Vu and Aycard [14] fit a box model to the obstacle. In the aerospace field, Baum and Hanebeck [15] approximate an extended object with a simple geometric shape such as an ellipse and then estimate the parameters of this shape in the tracking problem. While these simplistic methods are easy to implement and are often fast, they make assumptions about the obstacle model that can lead to tracking errors, which we seek to avoid.

There have been attempts to use iterative closest point (ICP) for obstacle tracking [16]. Snapshots can be aligned together and the relative poses can be used as observations in a Kalman filter. However, this has been shown to be slow and prone to local minima, especially when there are errors in correctly segmenting and associating LIDAR observations with an obstacle [17].

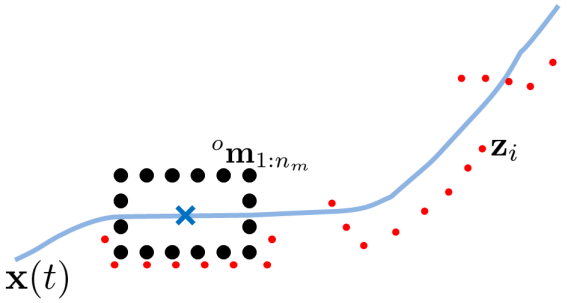


Fig. 2: A high level description of the problem. The obstacle moves along some path $\mathbf{x}(t)$, displayed in blue. The obstacle is modeled by n_m model points, each depicted as a black dot. We have observations such as $\mathbf{z}_i$, each depicted as a red dot. Note that each snapshot roughly captures the geometric shape of the obstacle, but is prone to errors due to the obstacle's motion during its collection. In this work, we do not make the assumption that the points $\mathbf{z}_i$ in a single snapshot correspond to the same obstacle pose $\mathbf{x}(t)$.

Other techniques for registering snapshots have been explored. Held et al. [5] propose a method to search for a 2D translation to register the most recent snapshot with previously observed snapshots of the obstacle. This can then be refined for rotation and finally used in a Kalman filter. However, this makes the implicit assumption that sensor data is available in snapshots and does not model the true nature of the timing of the sensor, leaving it prone to errors due to the obstacle's movement during the collection of the snapshot.

Some obstacle trackers take a somewhat different approach and use an occupancy grid. Tanzmeister et al. [18] and Danescu et al. [19] use a grid-based method where every cell maintains a particle filter and is classified as being a static or dynamic obstacle based on the statistics of the particles in it.

Our proposed obstacle tracker seeks to leverage a generalized obstacle model that can be used to track any kind of obstacle (e.g., car, motorcycle, bicycle, or pedestrian). Additionally, we are interested in leveraging continuous time estimation, as proposed by Furgale et al. [6], in order to properly handle the fast rate sensors commonly used in this area, accounting for the obstacle's motion during the collection of a snapshot. This technique has been shown to be a successful approach to SLAM problems with high rate sensors [7] as is the case in our domain.

III. Problem Statement

Let $\mathbf{x}(t)$ be the state of the obstacle at time t, and let $\mathbf{z}_{1:n_z}$ be our n_z observations, with each $\mathbf{z}_i \in \mathbb{R}^3$ and each associated with a time t_i. Each observation is in the world frame, and we assume that the vehicle has a good localization system such that uncertainty in the world frame position of these observations is negligible over the time scale of tracking.

Let ${}^{o}\mathbf{m}_{1:n_m}$ be a point cloud representation of the obstacle, consisting of n_m points expressed in the obstacle frame. (We represent a point in the obstacle frame as ${}^{o}\mathbf{p}$. Points are in

the world frame otherwise.) We explore the parameter n_m in §V. This is depicted in Fig. 2.

We seek to find the maximum of the joint posterior density:

$$\mathbf{x}(t)^\star, {}^o\mathbf{m}^\star_{1:n_m} = \underset{\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m}}{\operatorname{argmax}} \; p(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m} \,|\, \mathbf{z}_{1:n_z}), \quad (1)$$

solving for the obstacle pose over time $\mathbf{x}(t)$ and model $\mathbf{m}$. Note that we have framed the problem as a maximum a posteriori estimation problem, similar to the formulation that is used in [6].

We define the obstacle state at time t as $\mathbf{x}(t) = \left[x, y, z, \phi, v, v_z, \dot{\phi}\right]^\top$, where the 3D position of the obstacle in the world frame is given by (x, y, z), ϕ and v are, respectively, the current heading of the obstacle and its forward speed, v_z is the speed in the vertical direction, and $\dot{\phi}$ is the rate of turning. We assume that the obstacle's roll and pitch are negligible for performance reasons, as any typical obstacle in our environment would be constrained to a small roll and pitch. However, if desired, roll and pitch could be added to the state vector.

The front end to our system is similar to that of Leonard et al. [20]. It segments LIDAR observations according to which obstacle they belong to. These segments are then linked through time to provide obstacle tracks. The state of these tracks is then estimated by using the centroid of each segment as a measurement for an EKF. This serves as our baseline tracker in §V.

We use this EKF estimate to initialize our proposed tracking method. Similar to other trackers such as [5], we assume that we have good segmentation of the LIDAR observations and data association of the segments through time.

IV. Method

Our method simultaneously optimizes for obstacle state over time $\mathbf{x}(t)$ as well as the point cloud model for the obstacle ${}^o\mathbf{m}_{1:n_m}$ through an iterative batch optimization process. The following sections describe the formulation for the models used, followed by the optimization procedure.

A. Formulation

Starting from (1) and applying Bayes' rule, we have:

$$p\big(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m} | \mathbf{z}_{1:n_z}\big) = \eta p\big(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m}\big) p\big(\mathbf{z}_{1:n_z} | \mathbf{x}(t), {}^o\mathbf{m}_{1:n_m}\big), \quad (2)$$

where η is a normalization constant. We make the assumption that the obstacle trajectory and the obstacle model are independent:

$$p\big(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m}\big) \approx p\big(\mathbf{x}(t)\big) p\big({}^o\mathbf{m}_{1:n_m}\big), \quad (3)$$

and thus we arrive at:

$$p\big(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m} | \mathbf{z}_{1:n_z}\big) \approx \eta p\big(\mathbf{x}(t)\big) p\big({}^o\mathbf{m}_{1:n_m}\big) p\big(\mathbf{z}_{1:n_z} | \mathbf{x}(t), {}^o\mathbf{m}_{1:n_m}\big). \quad (4)$$

We now describe the three terms.

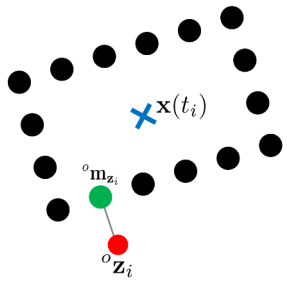


Fig. 3: An illustration of our measurement model. The observation, the red point, is associated with the closest model point, the green point. The black points represent other points that make up the obstacle model.

1) Process Model: $p\big(\mathbf{x}(t)\big)$ is the process model of the obstacle. Our method is generic for any motion model. In our application, we will use a constant velocity unicycle model $\dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t)\big) + \mathbf{w}(t)$ where:

$$\mathbf{f}\big(\mathbf{x}(t)\big) = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ v_z \\ \dot{\phi} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad (5)$$

and $\mathbf{w}(t)$ is zero-mean Gaussian noise with covariance $\mathrm{Q}_u\delta(t-t')$, where $\delta(t)$ is the Dirac delta function. This yields [21]:

$$p\big(\mathbf{x}(t)\big) \propto \exp\left\{\int_{t_0}^{t_f} \mathbf{e}_u(\tau)^\top \mathrm{Q}_u^{-1} \mathbf{e}_u(\tau) d\tau\right\}, \quad (6)$$

where

$$\mathbf{e}_u(\tau) = \dot{\mathbf{x}}(\tau) - \mathbf{f}(\mathbf{x}(\tau)), \quad (7)$$

and t_0 to t_f represents the timespan over which we wish to compute $p\big(\mathbf{x}(t)\big)$.

2) Obstacle Model: $p({}^o\mathbf{m}_{1:n_m})$ is our prior on the obstacle model. We use this prior to enforce that obstacles be of a reasonable size, and do so by weakly constraining each model point to be near the origin:

$$p({}^o\mathbf{m}_{1:n_m}) = \prod_{i=1}^{n_m} \mathcal{N}({}^o\mathbf{m}_i; \mathbf{0}, \mathrm{R}_m), \quad (8)$$

where $\mathrm{R}_m = \mathrm{diag}(\sigma_x^2, \sigma_y^2, \sigma_z^2)$, modeling the largest expected length, width, and height of obstacle we wish to track.

3) Measurement Model: $p(\mathbf{z}_{1:n_z} | \mathbf{x}(t), {}^o\mathbf{m}_{1:n_m})$ is our measurement model. We treat our measurements $\mathbf{z}_{1:n_z}$ as being conditionally independent given the state $\mathbf{x}(t)$ and the model ${}^o\mathbf{m}_{1:n_m}$. For each $\mathbf{z}_i$, we first associate our observation with a point in the model $\mathbf{m}$. We do this by projecting $\mathbf{z}_i$ from the world frame into the obstacle frame at time t_i to find ${}^o\mathbf{z}_i$. Then, we find the nearest neighbor of ${}^o\mathbf{z}_i$ in ${}^o\mathbf{m}_{1:n_m}$ to find ${}^o\mathbf{m}_{\mathbf{z}_i}$. Thus, our measurement model becomes:

$${}^o\mathbf{z}_i = {}^o\mathbf{m}_{\mathbf{z}_i} + \mathbf{n}_i, \quad (9)$$

$$\mathbf{e}_{z_i} = {}^o\mathbf{z}_i - {}^o\mathbf{m}_{\mathbf{z}_i}, \quad (10)$$

where $\mathbf{n}_i \sim \mathcal{N}(\mathbf{0}, \mathrm{R}_z)$ and ${}^o\mathbf{z}_i$ is the measurement $\mathbf{z}_i$ projected into the obstacle frame according to $\mathbf{x}(t_i)$, as shown in Fig. 3.

B. Continuous Time

In a discrete-time setting, we would instantiate discrete variables representing the obstacle state at $\mathbf{x}(t_0), \ldots, \mathbf{x}(t_{n_z})$. However, we treat each observation from our LIDAR sensor to be a separate measurement (as opposed to a snapshot of observations all assumed to have been taken at the same time). Creating a state variable for each one of these observations can quickly become intractable.

To address this issue, we use continuous time estimation [6] where we model the state of the car as a linear combination of temporal basis functions:

$$\mathbf{x}(t) = [\boldsymbol{\phi}_\mathbf{1}(t),\ \boldsymbol{\phi}_\mathbf{2}(t),\ \ldots,\ \boldsymbol{\phi}_\mathbf{n}(t)]\,\mathbf{c} \tag{11}$$

$$= \Phi(t)\mathbf{c}. \tag{12}$$

As our basis functions, we select B-splines of degree 4 [22]. Each B-spline function has limited support, which helps make the problem sparse.

Thus, instead of solving for a large number of state variables, the variables we are solving for are reduced to just the vector of weights $\mathbf{c}$ that operates on the basis functions. The number of variables is reduced by several orders of magnitude, depending on the resolution of B-splines desired.

We thus define the full set of variables that we are solving for as:

$$\boldsymbol{\theta} = \begin{bmatrix} \mathbf{c} \\ {}^o\mathbf{m}_{1:n_m} \end{bmatrix}. \tag{13}$$

C. Gauss-Newton

We formulate a cost function for optimization by taking the negative log-probability of (1), yielding:

$$-\log\left(p(\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m} | \mathbf{z}_{1:n_z})\right) = k + J_m + J_u + J_z, \tag{14}$$

where

$$J_m = \sum_{i=1}^{n_m} \frac{1}{2} \mathbf{m}_i^\top \mathrm{R}_m \mathbf{m}_i, \tag{15}$$

$$J_u = \int_{t_0}^{t_f} \mathbf{e}_u(\tau)^\top \mathrm{Q}_u^{-1} \mathbf{e}_u(\tau) d\tau, \tag{16}$$

$$J_z = \sum_{i=1}^{n_z} \mathbf{e}_{z_i}^\top \mathrm{R}_z \mathbf{e}_{z_i}^\top, \tag{17}$$

and k is a constant.

To solve for $\boldsymbol{\theta}$, we start with an initial guess $\bar{\boldsymbol{\theta}}$, we linearize each of J_m, J_u, and J_z about $\bar{\boldsymbol{\theta}}$. For each, we find $\frac{\delta J}{\delta \boldsymbol{\theta}}^\top$, which is of the form $\mathrm{A}\boldsymbol{\delta\theta} + \mathbf{b}$. We take $\mathrm{A}_m + \mathrm{A}_u + \mathrm{A}_z = \mathrm{A}$ and $\mathbf{b}_m + \mathbf{b}_u + \mathbf{b}_z = \mathbf{b}$. Thus, we have the Gauss-Newton update step $\mathrm{A}\boldsymbol{\delta\theta} = -\mathbf{b}$, by which we iteratively update $\boldsymbol{\theta}$ until convergence.

When solving, we compute a full batch update using all of the data for the obstacle, recomputing the data associations for the measurement model for each iteration. Note that the matrix A is sparse due to the limited support of the splines, which allows the use of sparse linear solvers that scale to larger numbers of points.

To initialize $\boldsymbol{\theta}$, we use the centroid-based EKF tracker described in §III. Specifically, we first fit B-splines to the pose estimate history given by the EKF tracker to generate our initial guess for $\mathbf{c}$. Then, we project the measurements $\mathbf{z}_{1:n_z}$ into the obstacle frame and subsample these points linearly in time to generate our initial guess for ${}^o\mathbf{m}_{1:n_m}$.

Fig. 4: Our autonomous platform, a Ford Fusion, which we used in our experiments. There are four spinning Velodyne HDL-32E 3D LIDAR scanners on the roof. Additional sensors, such as an Applanix POS-LV 420 INS, and compute resources are in the trunk of the car.

V. Experimental Results

A. Setup

Our proposed method was evaluated using a Ford Fusion autonomous platform, as shown in Fig. 4. Among other sensors, this autonomous car has four Velodyne HDL-32E 3D LIDAR scanners spinning at roughly 10 Hz. An Applanix POS-LV 420 INS is used for our vehicle's pose estimation. A global localization system similar to [23] allows us to leverage prior maps of the ground plane to improve the performance of the baseline tracker described in §III. Experiments were run on a 2.80 GHz Intel Core i7-3840QM CPU.

We replicate the experimental methodology used by Held et al. [5] by looking at tracking error and point cloud crispness. We evaluated our method on a dataset collected around the University of Michigan, Ann Arbor North Campus.

We evaluate our method in two ways. First, to evaluate the quality of the positional tracking, we identify parked cars in our dataset and evaluate the performance of our tracker on these obstacles. The tracking system does not know that these obstacles are not moving, and thus it tracks them over time. Because we know these obstacles are stationary, we essentially have a ground truth we can use to evaluate our tracker. We call this the stationary set, containing 52 obstacles each observed for an average of 6.05 s. Then, we look at the crispness of the point cloud created by dynamic obstacle tracks. If we are properly tracking the obstacle, then we would expect to see a clear, crisp point cloud view of it. We call this the dynamic set, containing 52 obstacles each observed for an average of 6.28 s. In both cases, we have

TABLE I: The tracking performance of our proposed tracker (for two values of n_m) as compared to the baseline EKF centroid tracker over the stationary set.

Tracker Error	**Baseline EKF**	**Proposed** $n_m = 300$	**Proposed** $n_m = 3000$
Pos. RMSE	0.388 m	0.162 m	0.183 m
Vel. RMSE	0.543 m/s	0.314 m/s	0.328 m/s
ϕ *RMSE*	0.527 rad	0.071 rad	0.090 rad
$\dot{\phi}$ *RMSE*	0.139 rad/s	0.026 rad/s	0.027 rad/s

TABLE II: The entropy of the point cloud of our proposed system as compared to the baseline EKF centroid tracker. *Ground Truth* is available for the stationary set by taking the LIDAR observations in the world frame.

Tracker Entropy	**Ground Truth**	**Baseline EKF**	**Proposed** $n_m = 300$
Stationary Set	1.920	2.237	2.103
Dynamic Set	—	2.860	2.756

manually discarded obstacles that are a result of errors in segmentation.

B. Parameter Selection

We find that a relatively sparse representation of the obstacle's point cloud model ${}^o\mathbf{m}_{1:n_m}$ can still produce good results while greatly improving the runtime performance, even when the model consists of just a few hundred points. For these results, we set n_m to 300 points. In fact, often if we set n_m too large, the model can tend to overfit any inaccuracies in the initialization from the baseline system, leading to poorer results, as reflected in Table I. Note that after optimizing for $\mathbf{x}(t)$, we can then reproject all of our observations $\mathbf{z}_{1:n_z}$ according to $\mathbf{x}(t)$ into the obstacle's reference frame. Thus, we can recreate the full, dense, point cloud.

The number of measurements n_z that we have grows quickly. Even if we only observe an obstacle for a few seconds, we quickly accumulate hundreds of thousands of observations. We downsample our observations to 5000 observations per obstacle track when computing $p(\mathbf{z}_{1:n_z}|\mathbf{x}(t), {}^o\mathbf{m}_{1:n_m})$.

For the process model, we set Q_u according to the maximum accelerations and turning rates we expect a vehicle to undertake. Therefore, we set:

$$\mathrm{Q}_u = \mathrm{diag}\big(0.01^2\ \mathrm{m}^2, 0.01^2\ \mathrm{m}^2, 0.01^2\ \mathrm{m}^2, 0.001^2\ \mathrm{rad}^2, \\ 0.9^2\ (\mathrm{m/s})^2, 0.1^2\ (\mathrm{m/s})^2, 0.08165^2\ (\mathrm{rad/s})^2\big).$$

For the obstacle model, we considered the largest size of the obstacles we wish to track, and thus set $\sigma_x = \sigma_y = \sigma_z = 10$ m.

For the measurement model, we set $\mathrm{R}_z = \sigma_z \mathrm{I}$. When choosing σ_z, we must consider both error due to the noise of the LIDAR sensor and error due to the relative sparsity of our model points. We chose $\sigma_z = 0.50$ m.

For the B-splines, we have a choice of how many B-spline basis functions to use in representing our state. Using more splines would allow us to more finely represent our state over time; on the other hand, the more splines we use, the larger $\boldsymbol{\theta}$ will be, taking longer to solve. Thus, there is a tradeoff to consider between accurate representation of state versus computation time. We use a basis function per dimension for every 0.5 s that the obstacle is tracked.

C. Pose Results

We report on the tracking error of the stationary set. We compute position error and heading error by comparing the (x, y, z) and ϕ estimate over time to the mean position and heading, respectively. We compute velocity and rate of turning error by comparing the velocity and heading estimate to an expected value of 0 m/s and 0 rad/s, respectively. These results are shown in Table I. We see a clear improvement in tracking performance over the baseline EKF centroid tracker as described earlier, particularly in heading. Additionally, note that the performance of the proposed tracker is better with $n_m = 300$ than $n_m = 3000$, as explained above.

D. Point Cloud Crispness

In the general case, with obstacles that are not known to be stationary, we evaluate the performance of our proposed tracker with regards to the quality of the generated point could. We want a crisp point cloud, or equivalently one with low entropy. A crisper point cloud is indicative of better tracking performance. We evaluate our proposed tracker by using the point cloud entropy as defined by Sheehan et al. [24]:

$$H[{}^o\mathbf{z}_{1:n_z}] = -\log\left(\frac{1}{n_z^2}\sum_{i=1}^{n_z}\sum_{j=1}^{n_z}\mathcal{N}({}^oz_i - {}^oz_j; \mathbf{0}, 2\sigma^2\mathbf{I})\right).$$

The parameter σ allows us to tune the resolution at which we evaluate crispness. We set $\sigma = 5$ cm.

The results are shown in Table II for both the stationary set and the dynamic set. Note the improvement in both cases of our proposed method over the baseline. To compute the ground truth entropy of the stationary set, we project all of the LIDAR observations into the world frame using the known trajectory of our platform. As the obstacles in the stationary set are not moving in the world frame, this represents the best point cloud we can construct and thus is the lowest entropy we should expect from an ideal tracker. We can see that our proposed method reduces the entropy of the point cloud by about $42\ \%$ relative to the ground truth entropy as compared to the baseline tracker.

The stationary set has less entropy (i.e., it is more crisp) in general when compared to the dynamic set. This is due to the fact that in the stationary set, we commonly traveled close to cars parked on the side of the road, creating a denser point cloud. This is opposed to the dynamic set, where obstacles are often being tracked from a distance (for example, following a car on the road) or in different lanes. Regardless, the improvement in point cloud crispness is evident in both the stationary set and the dynamic set.

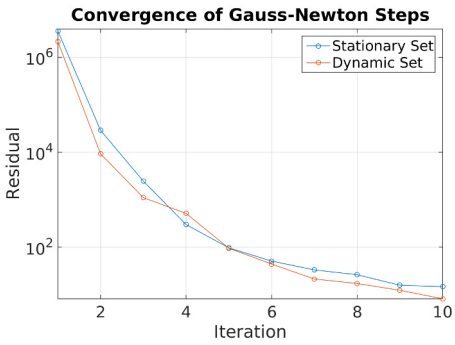
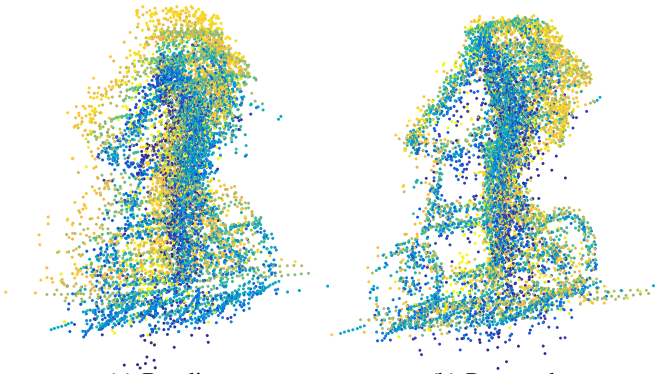


Fig. 5: The average residual after each iteration over the stationary and dynamic set (log scale). The decreasing residuals are indicative of better agreement in the process, measurement, and obstacle models. Note how our proposed method converges quickly.



(a) Baseline (b) Proposed

Fig. 6: Our proposed method tracking a person riding a bicycle. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.

E. Convergence

We determine how well our method converges by considering the residual of the Gauss-Newton step at each iteration. As can be seen in Fig. 5, the error converges consistently on the tracking scenarios that we have evaluated. The decreasing residuals are indicative of better agreement in the process model, measurement model, and obstacle model. In our experiments, the Gauss-Newton process is stopped when convergence is reached or after a maximum of 10 iterations.

F. Runtime

We find that our proposed method takes an average of 42.9 ms per iteration for our choice of parameters. This was measured on an 2.80 GHz Intel Core i7-3840QM CPU. Note that the matrix A in the Gauss-Newton step is sparse, as only about 2 % of the elements are non-zero.

VI. Discussion

We find that our system generally works well under nominal conditions. The tracking error for position, velocity, heading, and rate of heading are all improved with respect to the baseline EKF tracker. Additionally, the resulting point cloud is crisper.

Unlike other obstacle trackers that rely on certain models (such as a bounding box), we are able to track any object. For example, see Fig. 6 for our tracker working on a bicyclist.

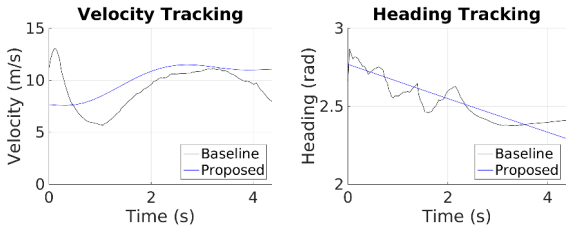


Fig. 7: A comparison of the estimated velocity and heading profiles for the bus displayed in Fig. 8.

We find that heading and lateral position (i.e., perpendicular to the direction of travel) have particularly low error, even when initialized poorly. For example, in Fig. 8, we see that our proposed tracker has managed to develop a crisp model of a bus despite a significant amount of heading error in the baseline method. As can be seen in Fig. 7, our proposed tracker creates a smoother velocity and heading estimate, which is much more realistic for a bus that cannot change its velocity or heading as abruptly as the baseline tracker would suggest.

Certain situations are known to be troublesome. For example, sometimes we encounter the situation shown in Fig. 9 where we might have multiple slightly translated views of the car, which is indicative of tracking error. While this is still a significant improvement over the EKF centroid tracker, our proposed tracker has some trouble resolving the discrepancy along the direction of travel. This is likely due to bad initialization, leading the tracker to create an obstacle model that has two translated instances of the rear face of the car. Even though this is incorrect, this obstacle model is consistent with the measurements and the tracker will believe that it has correctly tracked and modeled the obstacle. We believe that these issues can be addressed with a better prior on the obstacle model, better initialization, or explicit ray casting.

VII. Conclusions

We have demonstrated how tools from SLAM and continuous-time estimation can be applied to obstacle tracking. We showed how this approach yields improved results when compared to a baseline EKF centroid-based tracking system. Future work will consider better priors and models, such as a mesh obstacle model, and also tradeoffs between improved error functions and runtime. Additionally, we will consider better modeling the relationship between the obstacle trajectory and the corresponding point cloud model.

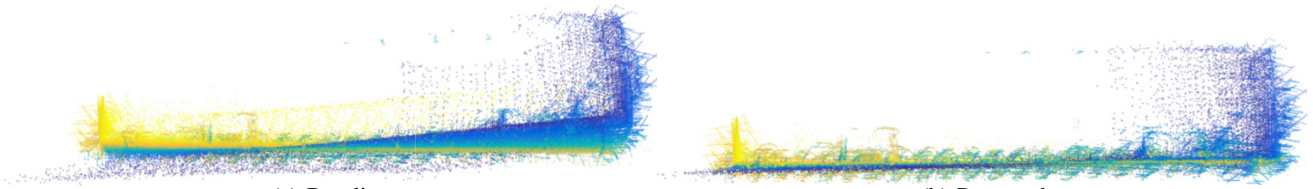
References

[1] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson., "Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction," in *Proc. Robot.: Sci. & Syst. Conf.*, Rome, Italy, July 2015.

[2] W. Xu, J. Pan, J. Wei, and J. Dolan, "Motion planning under uncertainty for on-road autonomous driving," in *Proc. IEEE Int. Conf. Robot. and Automation*, Hong Kong, China, 2014, pp. 2507–2512.

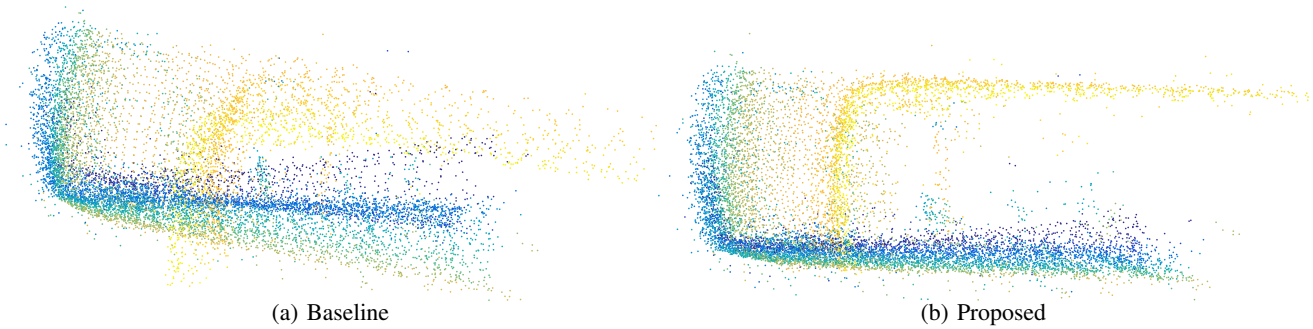
[3] A. Teichman and S. Thrun, "Tracking-based semi-supervised learning," *Int. J. Robot. Res.*, vol. 31, no. 7, pp. 804–818, 2012.

[4] B. Douillard, D. Fox, F. Ramos, and H. Durrant-Whyte, "Classification and semantic mapping of urban environments," *Int. J. Robot. Res.*, vol. 30, no. 1, 2010.



(a) Baseline (b) Proposed

Fig. 8: Our proposed method tracking a bus, top-down view. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.



(a) Baseline (b) Proposed

Fig. 9: A fault case of our proposed tracker. While the tracking performance is improved, there is still error in the direction of travel. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.

[5] D. Held, J. Levinson, S. Thrun, and S. Savarese, "Combining 3d shape, color, and motion for robust anytime tracking," in *Proc. Robot.: Sci. & Syst. Conf.*, Berkeley, CA, USA, 2014, pp. 1–8.

[6] P. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," in *Proc. IEEE Int. Conf. Robot. and Automation*, St. Paul, MN, USA, 2012, pp. 2088–2095.

[7] S. Anderson and T. Barfoot, "Towards relative continuous-time SLAM," in *Proc. IEEE Int. Conf. Robot. and Automation*, Karlsruhe, Germany, 2013, pp. 1033–1040.

[8] S. Anderson, F. Dellaert, and T. D. Barfoot, "A hierarchical wavelet decomposition for continuous-time SLAM," in *Proc. IEEE Int. Conf. Robot. and Automation*, Hong Kong, China, 2014, pp. 373–380.

[9] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3d environment," in *Proc. IEEE Intell. Veh. Symp.*, Madrid, Spain, 2012, pp. 802–807.

[10] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Proc. IEEE Intell. Veh. Symp.*, Baden-Baden, Germany, 2011, pp. 163–168.

[11] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Auton. Robot.*, vol. 26, no. 2-3, pp. 123–139, 2009.

[12] R. Kaestner, J. Maye, Y. Pilat, and R. Siegwart, "Generative object detection and tracking in 3d range data," in *Proc. IEEE Int. Conf. Robot. and Automation*, St. Paul, MN, USA, 2012, pp. 3075–3081.

[13] M. Darms, P. Rybski, and C. Urmson, "Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments," in *Proc. IEEE Intell. Veh. Symp.*, Eindhoven, Netherlands, 2008, pp. 1197–1202.

[14] T.-D. Vu and O. Aycard, "Laser-based detection and tracking moving objects using data-driven markov chain monte carlo," in *Proc. IEEE Int. Conf. Robot. and Automation*, Kobe, Japan, 2009, pp. 3800–3806.

[15] M. Baum and U. D. Hanebeck, "Extended Object Tracking with Random Hypersurface Models," *IEEE Trans. on Aero. and Elec. Sys.*, vol. 50, pp. 149–159, 2014.

[16] A. Feldman, M. Hybinette, and T. Balch, "The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets," *J. Field Robot.*, vol. 29, no. 2, pp. 258–276, 2012.

[17] D. Held, J. Levinson, and S. Thrun, "Precision tracking with sparse 3d and dense color 2d data," in *Proc. IEEE Int. Conf. Robot. and Automation*, Karlsruhe, Germany, 2013, pp. 1138–1145.

[18] G. Tanzmeister, J. Thomas, D. Wollherr, and M. Buss, "Grid-based mapping and tracking in dynamic environments using a uniform evidential environment representation," in *Proc. IEEE Int. Conf. Robot. and Automation*, Hong Kong, China, 2014, pp. 6090–6095.

[19] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Trans. Intell. Transp. Sys.*, vol. 12, no. 4, pp. 1331–1342, 2011.

[20] J. Leonard *et al.*, "Team MIT Urban Challenge technical report," Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts, Tech. Rep. MIT-CSAIL-TR-2007-058, 2007.

[21] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, Inc., 1970.

[22] C. de Boor, *A Practical Guide to Splines*. Springer Verlag (New York), 1978.

[23] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *Proc. IEEE Int. Conf. Robot. and Automation*, Anchorage, AK, USA, 2010, pp. 4372–4378.

[24] M. Sheehan, A. Harrison, and P. Newman, "Self-calibration for a 3d laser," *Int. J. Robot. Res.*, vol. 31, no. 5, 2012.