The International Journal of Robotics Research
1–28
©The Author(s) 2015
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI:10.1177/0278364915593399
http://mms.sagepub.com

Robust Real-Time Tracking Combining 3D Shape, Color, and Motion

D. Held[*], J. Levinson, S. Thrun, and S. Savarese
Department of Computer Science, Stanford University, USA

Abstract
Real-time tracking algorithms often suffer from low accuracy and poor robustness when confronted with difficult, real-world data. We present a tracker that combines 3D shape, color (when available), and motion cues to accurately track moving objects in real-time. Our tracker allocates computational effort based on the shape of the posterior distribution. Starting with a coarse approximation to the posterior, the tracker successively refines this distribution, increasing in tracking accuracy over time. The tracker can thus be run for any amount of time, after which the current approximation to the posterior is returned. Even at a minimum runtime of 0.37 milliseconds per object, our method outperforms all of the baseline methods of similar speed by at least 25% in RMS tracking error. If our tracker is allowed to run for longer, the accuracy continues to improve, and it continues to outperform all baseline methods. Our tracker is thus anytime, allowing the speed or accuracy to be optimized based on the needs of the application. By combining 3D shape, color (when available), and motion cues in a probabilistic framework, our tracker is able to robustly handle changes in viewpoint, occlusions, and lighting variations for moving objects of a variety of shapes, sizes, and distances.

Keywords
Tracking, 3D, Real-time

1. Introduction

Many robotics applications are limited in what they can achieve due to unreliable tracking estimates. For example, an autonomous vehicle driving past a row of parked cars should know if one of these cars is about to pull out into the lane. Current state-of-the-art trackers give noisy estimates of the velocity of these vehicles, which are difficult to track due to heavy occlusion and viewpoint changes. Additionally, without robust estimates of the velocity of nearby vehicles, merging onto or off of highways or changing lanes become formidable tasks. Similar issues will be encountered by any robot that must act autonomously in crowded, dynamic environments.

Our tracker makes use of the full 3D shape of the object being tracked, which allows us to robustly track objects despite occlusions or changes in viewpoint. We place the 3D shape information in a probabilistic framework in which we combine

[*] Corresponding author; e-mail: davheld@cs.stanford.edu

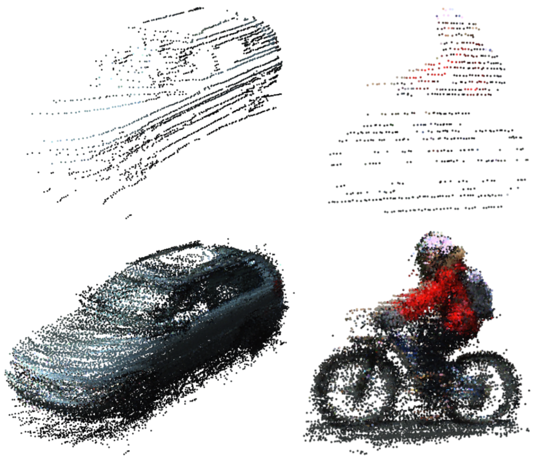

Fig. 1. Our method tracks moving objects very accurately, as seen by these models generated from successive frame-to-frame alignments from our tracker. The top row shows the individual frame of the tracked object with the largest number of points. The bottom row shows the model created by our tracker.

cues from shape, color, and motion. As we will show, the 3D shape, color, and motion each contribute to the performance of our system.

We make use of a novel grid-based method to sample velocities from the state space. Traditional grid-based approaches, such as histogram filters, are too slow to track multiple objects in real-time. We are able to finely sample from a large grid in real-time through the use of a novel method called *annealed dynamic histograms*. We start by sampling from the state space at a coarse resolution, using an approximation to the posterior distribution over velocities. As the sampling resolution increases, we anneal this distribution and the approximate distribution approaches the true posterior. At any point, the current approximation to the posterior can be returned, with tracking resolution or runtime chosen based on the needs of the application.

Our tracker presents a number of novel contributions over the previous state of the art. First, we introduce a new sampling method called *annealed dynamic histograms* to globally explore the state space in real-time. This method allows our tracker to estimate object velocities significantly more accurately than previous state-of-the-art approaches for real-time tracking. We also present a novel derivation of a measurement model using the idea of a latent surface, which gives us insight into how to select the model parameters. We extend this model to optionally include color, allowing us to combine color, 3D shape, and motion in a coherent probabilistic framework. By combining these different cues, our tracker is more robust to changes in viewpoint and occlusions. Finally, we perform a detailed quantitative analysis of how our method compares to state-of-the-art tracking methods when tested on a large number of tracked objects of different types (people, bikes, and moving cars) over varying levels of lighting, viewpoint changes, and occlusions. One of our evaluation metrics involves computing the crispness of different object models generated using our tracker, as seen in Figure 1.

This article extends our previous work that was presented in Held et al. (2013) and Held et al. (2014). We present the following novel contributions over our previous publications:

- A more detailed description of our probabilistic model (Section 4)
- A more detailed derivation of our measurement model (Sections 5.1 and 5.2)
- A description of two implementation improvements which reduce our RMS error by 10% and make our method 3 times faster than that of our previous publications (Section 5.4)
- A more detailed analysis of our results (Section 8.2)
- A visualization of examples where our method outperforms a baseline ICP method (Section 8.4)

For additional details on this project or to access the source code for our tracker, please see our project page at `http://stanford.edu/~davheld/anytime_tracking.html`.

2. Related Work

Tracking using 3D data has been an important challenge for many years. Traditionally, trackers that have depth data available have discarded almost all of the 3D information, representing an object either by its centroid (Levinson et al., 2011; Kaestner et al., 2012) or by the center of a bounding box (Leonard et al., 2008; Azim and Aycard, 2012; Streller et al., 2002) wrapped in a Kalman filter. Although these are computationally efficient approaches, they are not very accurate.

Another method is to fit a 2D rectangular object model to the point cloud of the tracked object (Petrovskaya and Thrun, 2008). This method is designed for tracking objects that have a roughly rectangular shape, such as cars, and thus cannot be used as a generic multi-purpose object tracker. The model in Petrovskaya and Thrun (2008) relies on detecting the corner of the car in order to position the rectangular model, whereas Wojke and Haselich (2012) and Darms et al. (2008) also handle the case where only one side of the car is visible. Our model is more general, in that it uses the full 3D shape and does not assume that the tracked object is rectangular or any other pre-specified shape.

To make use of the full 3D shape of the tracked object, some trackers have attempted to align the object's point clouds using ICP and its variants (Feldman et al., 2012; Moosmann and Stiller, 2013). Such trackers use a local hill-climbing approach to iteratively improve an alignment of two point clouds. However, these approaches depend heavily on starting from a good initial alignment, and their accuracy degrades when the initialization is not close to the true alignment, as has been shown by Olson (2009) and Held et al. (2013).

Grid-based methods are more common in SLAM systems (Simmons and Koenig, 1995) than in tracking, presumably because of the computational issues involved in using fine grids. Our method enables a fine grid to be used for real-time tracking by using a coarse-to-fine sampling with annealed dynamic histograms. This general approach to tracking is also related to the methods used in various multi-resolution grid-based SLAM systems (Burgard et al., 1998; Olson, 2000, 2009; Marcello et al., 2002; Ryde and Hu, 2010; Estivill-Castro and McKenzie, 2004).

Our work differs from the previous work on multi-resolution grid-based SLAM of Olson (2009) in a number of ways. First, our method is designed for tracking, whereas the work of Olson (2009) is designed for SLAM. Second, their approach to multi-resolution sampling is significantly slower than ours. In order to track many objects in real-time, the time to track each object must be kept small. We demonstrate that their method requires over an order of magnitude more sample evaluations to obtain the same accuracy. Finally, our derivation of the measurement model enables us to easily extend the model to incorporate color, leading to a significant improvement in accuracy. In contrast, their measurement model was constructed based on a more heuristic reasoning and is thus less easily extensible.

Our method of annealed dynamic histograms is related to the deterministic annealing methods for optimization. In deterministic annealing, an optimum is found for an approximate distribution, and the distribution is gradually annealed as the optimal solution is refined (Rose, 1998). Related methods known as “shaping" have been used in reinforcement learning, starting with an easier task and progressing to increasingly difficult tasks (Randlov and Alstrom, 1998; Konidaris and Barto, 2006). This class of methods is also known as “graduated optimization" and has been applied to the related problem of image alignment (Zitnick, 2012). In contrast, our goal is not optimization but rather to estimate the posterior distribution over velocities for a tracked object. In a general sense, our method is similar in that we start by sampling from an approximation to the posterior distribution and then we refine our approximation over time.

Our method differs from previous tracking approaches in that we globally explore the state space in real-time, as opposed to ICP and similar methods which perform hill-climbing from a given initialization. We also combine 3D shape, color, and motion information, unlike standard ICP and scan-matching techniques that use the 3D shape alone. Our method is general and can robustly track moving objects in real-time, despite heavy occlusions or viewpoint changes that can occur in real-world tracking scenarios.

3. Tracking Pipeline

As a pre-processing step, we use a point-cloud based segmentation and data association algorithm, which segments objects from the background into clusters and associates these object clusters between successive time frames (Teichman et al., 2011). Our tracking method then estimates the velocity of each of the pre-segmented tracked objects. We introduce a new technique called annealed dynamic histograms to globally explore the search space in real-time. We start by sampling the state space with a coarse grid, and for each sample, we compute the probability of the state using the model described in Section 5. We then subdivide some of the grid cells, as described in Section 6, to refine our distribution. Over time, our distribution becomes increasingly accurate. After the desired runtime or tracking resolution, the tracker can be stopped, at which point the current posterior distribution over velocities can be returned.

4. Probabilistic Model

4.1. State Space

Below we describe the probabilistic model that we use for tracking. In Section 6 we will describe how we use this model as part of our annealed dynamic histogram framework. The state variable x_t is defined as $x_t = (x_{t,p}, \dot{x}_t)$, where $x_{t,p}$ is the linear position and $\dot{x}_t$ is the velocity of the tracked object. Because we are interested in tracking to estimate the motion of objects, the position state variable $x_{t,p}$ measures the change in position relative to the last observation. To achieve this, after each observation, we set the origin of the coordinate system to be located at the centroid of the previous observation, as shown in Figure 2. The position state variable thus measures how far this object has moved since the previous observation.

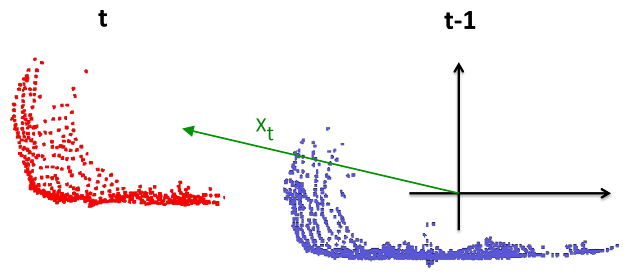

Fig. 2. The coordinate system for our state space. On the right is the tracked object observed at time t-1, and on the left is the new observation at time t. At each time step, we place the origin of our coordinate system on the center of the previous observation.

We assume that the rotational velocity of the tracked object is small relative to the frame rate of the sensor. This is often the case for people, bikes, and cars moving in urban settings. Thus, the rotational velocity is not included in the state. If one is interested in estimating the rotational velocity, then after obtaining the posterior over translation one can optionally search for the optimal rotation, as described in Section 6.4.

Our model is general and can be used to track objects moving in three dimensions. However, objects that we are interested in (people, bikes, and cars) are confined to move along the ground surface. Thus, to speed up our method, we assume that tracked objects exhibit minimal vertical motion within the frame rate of the sensor. Our state space therefore only models motion along the ground surface. This assumption results in a significant speedup of our method, with minimal effect on the accuracy. For settings in which one wants to track objects moving vertically, one can append the vertical dimension to the state space, resulting in a slightly slower method. Alternatively, one can incorporate an elevation map to predict vertical motion due to elevation changes.

4.2. Dynamic Bayesian Network

The Dynamic Bayesian Network upon which our model is built is shown in Figure 3. We wish to use the 3D shape of the object being tracked in a Bayesian probabilistic framework, which allows us to combine the 3D shape with information from color and motion. To do so, we include in our model a latent surface variable s_t, which corresponds to a set of points sampled from the visible surface of the tracked object. Without this term, the measurement z_t would be independent of the previous measurement z_{t-1} conditioned on the state x_t. Such a statement is false if our measurement includes the 3D shape of a tracked object, which stays relatively consistent from one time step to the next. Indeed, this independence assumption would prevent us from comparing 3D measurements of current and past observations for tracking. To enable us to incorporate the 3D shape of objects into our tracker, we add a variable s_t that represents the latent surface of the tracked object.

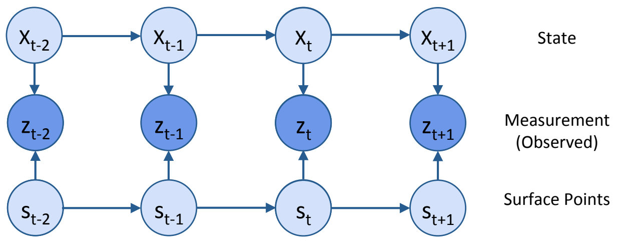

Fig. 3. Dynamic Bayesian Network representing our model for a tracked object.

The latent surface variable is related to similar notions from previous work. For example, Petrovskaya and Thrun (2008) include a geometry variable in which they model objects as 2D rectangles and estimate their widths and lengths. SLAM systems use a similar variable to represent the environment map (Thrun et al., 2005). Both of these methods attempt to explicitly model the geometry of the tracked object or environment. In contrast, we do not wish to explicitly model the object's shape, but rather we will integrate over shapes and focus on estimating the target object's velocity.

We represent the latent surface s_t as a collection of n points $\{s_{t,1} \dots s_{t,n}\} \in s_t$ sampled from the visible surface of the tracked object at time t. The prior $p(s_t)$ on these points is a uniform distribution over the maximum size of a tracked object. This prior decomposes as a product of the priors for each point in s_t, i.e. $p(s_t) = \prod_j p(s_{t,j})$.

The measurement z_t represents the set of n observed points at time t, $\{z_{t,1} \dots z_{t,n}\} \in z_t$. Each measurement $z_{t,j}$ is drawn from a corresponding latent surface point $s_{t,j}$. Because of sensor noise, the observed measurements z_t will not lie exactly on the object surface and hence will not be exactly equal to s_t. The observed points z_t are generated from the latent surface s_t and the state x_t via the following procedure: for each latent surface point $s_{t,j}$, Gaussian noise is added based on the sensor resolution Σ_e to create a noisy point $\tilde{s}_{t,j}$. The point $\tilde{s}_{t,j}$ is now shifted according to the current object position $x_{t,p}$ to generate the measurement $z_{t,j}$ at the appropriate location. Thus, we can write that

$$z_{t,j} \sim \mathcal{N}(s_{t,j}, \Sigma_e) + x_{t,p}. \tag{1}$$

As stated previously and shown in Figure 2, the previous measurements z_{t-1} are centered on the origin of the coordinate system. The points in z_{t-1} are noisy observations of the previous surface s_{t-1}. Thus for each point $z_{t-1,i} \in z_{t-1}$ from the previous observation, we have that

$$z_{t-1,i} \sim \mathcal{N}(s_{t-1,i}, \Sigma_e). \tag{2}$$

This creates an additional conditional independence assumption which is not encoded in the graphical model from Figure 3, namely that

$$p(z_{t-1} \mid x_t, s_{t-1}) = p(z_{t-1} \mid s_{t-1}). \tag{3}$$

The term $p(s_t, \mid s_{t-1})$ represents the probability of sampling points s_t from the currently visible object surface given the previously sampled points s_{t-1}. The sampled points may have changed due to occlusions, viewpoint changes, deformations, and random sampling. We suppose that every point $s_{t,j} \in s_t$ could have either been generated from a previously visible portion of the object surface at time $t-1$ or from a previously occluded portion. If $p(V)$ represents the prior probability of sampling from a previously visible surface, then we can write:

$$\begin{aligned} p(s_{t,j} \mid s_{t-1}) = & p(V)\, p(s_{t,j} \mid s_{t-1}, V) + \\ & p(\neg V)\, p(s_{t,j} \mid s_{t-1}, \neg V) \end{aligned} \tag{4}$$

We model $p(s_{t,j} \mid s_{t-1}, V)$ as a Gaussian, $s_{t,j} \sim \mathcal{N}(s_{t-1,i}, \Sigma_r)$ where Σ_r models the variance resulting from the sensor resolution as well as from object deformations, and $s_{t-1,i}$ is the nearest corresponding (latent) surface point from the previous frame. The sensor resolution changes as a function of distance, and Σ_r is computed accordingly for each tracked object.

The term $p(s_{t,j} \mid s_{t-1}, \neg V)$ represents the probability that $s_{t,j}$ is generated given that the surface from which it is sampled was previously occluded. If we have an occlusion model for the previous frame, we can use the occlusion model to compute this probability. Otherwise, we can assume that any region that was not previously visible was previously occluded. We can generically write this as

$$p(s_{t,j} \mid s_{t-1}, \neg V) = k_1\, (k_2 - p(s_{t,j} \mid s_{t-1}, V))$$

for some constants k_1 and k_2. We can now simplify equation 4 as

$$p(s_{t,j} \mid s_{t-1}) = \eta\, (p(s_{t,j} \mid s_{t-1}, V) + k) \tag{5}$$

where η is a normalization constant and k acts as a smoothing factor, and

$$\begin{aligned} \eta &= p(V) - p(\neg V) k_1 \\ k &= p(\neg V) k_1 k_2 / \eta. \end{aligned}$$

The sampling process is illustrated in Figure 4.

5. Tracking

We now describe how we use the Dynamic Bayesian Network described in Section 4.2 to track moving objects and estimate their velocities. Our goal is to estimate $p(x_t \mid z_1 \ldots z_t)$, the probability of the state x_t given the past observations. Using Bayes' rule, we can rewrite this as

$$p(x_t \mid z_1 \ldots z_t) = \eta\, p(z_t \mid x_t, z_1 \ldots z_{t-1})\, p(x_t \mid z_1 \ldots z_{t-1}) \tag{6}$$

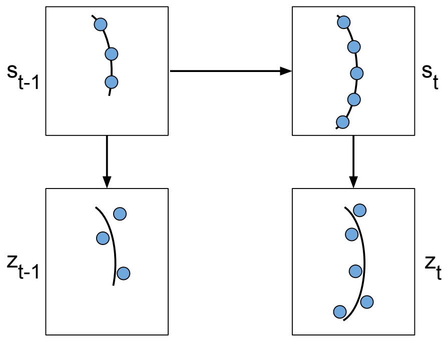

Fig. 4. Illustration of the sampling of surface points s_t and measurement points z_t. Because of sensor noise, the measurements z_t will not lie exactly on the object surface. Furthermore, because of occlusions, changes in viewpoint, deformations, and random sampling, the visible surface points change from s_{t-1} to s_t.

where η is a normalization constant. The first term is our measurement model. In the standard Bayes filter algorithm (Thrun et al., 2005), one would use conditional independence assumptions to simplify this as

$$p(z_t \mid x_t, z_1 \ldots z_{t-1}) = p(z_t \mid x_t).$$

However, in our case the latent variables $s_1, \ldots s_t$ prevent us from making this simplification. Because all observations come from the same object surface, they cannot be considered independent of each other. Therefore, we make a slightly different approximation:

$$p(z_t \mid x_t, z_1 \ldots z_{t-1}) \approx p(z_t \mid x_t, z_{t-1}) \tag{7}$$

Intuitively, the current observation is most strongly affected by the previous observation, rather than the entire past history of observations. Although tracking could be improved by using the entire past history of observations, this would add a computational cost that we wish to avoid. The second term on the right-hand side of equation 6 is obtained from our motion model, which will be described in Section 5.3.

5.1. Measurement Model Derivation

We now derive the measurement model, using the Dynamic Bayes Net from Figure 3. We can first write equation 7 using the joint distribution as

$$\begin{aligned} p(z_t \mid x_t, z_{t-1}) &= \int p(z_t, s_t \mid x_t, z_{t-1}) \, \mathrm{d}s_t \\ &= \int p(z_t \mid s_t, x_t) \, p(s_t \mid x_t, z_{t-1}) \, \mathrm{d}s_t \end{aligned} \tag{8}$$

where we have used the chain rule of probability and the conditional independence assumptions from the model of Figure 3. The second term inside the integral can be further expanded as

$$p(s_t \mid x_t, z_{t-1}) = \int p(s_t, s_{t-1} \mid x_t, z_{t-1}) \, \mathrm{d}s_{t-1} \tag{9}$$

Using independence assumptions from Figure 3 as well as the independence assumption from equation 3, we can further expand the term inside this integral as

$$\begin{aligned} p(s_t, s_{t-1} \mid x_t, z_{t-1}) &= p(s_t \mid s_{t-1})\, p(s_{t-1} \mid x_t, z_{t-1}) \\ &= \eta\, p(s_t \mid s_{t-1})\, p(z_{t-1} \mid x_t, s_{t-1})\, p(s_{t-1}) \\ &= \eta\, p(s_t \mid s_{t-1})\, p(z_{t-1} \mid s_{t-1})\, p(s_{t-1}) \end{aligned} \tag{10}$$

where η is a normalization constant. The term $p(s_{t-1})$ is a constant and can thus be absorbed by the normalization constant η. The next two terms are given by equations 2 and 5,

$$\begin{aligned} p(z_{t-1} \mid s_{t-1}) &= \mathcal{N}(z_{t-1}; s_{t-1}, \Sigma_e) \\ p(s_t \mid s_{t-1}) &= \eta\, (\mathcal{N}(s_t; s_{t-1}, \Sigma_r) + k) \end{aligned}$$

where η is a normalization constant and k is a smoothing term. We can now evaluate the integral in equation 9 to get

$$p(s_t \mid x_t, z_{t-1}) = \eta\, (\mathcal{N}(s_t; z_{t-1}, \Sigma_r + \Sigma_e) + k)$$

We have used the fact that the convolution of two Gaussians is another Gaussian, following the standard derivation as in Thrun et al. (2005). We also have from equation 1 that

$$p(z_t \mid s_t, x_t) = \mathcal{N}(z_t; s_t + x_{t,p}, \Sigma_e).$$

We can now evaluate the integral of equation 8 to get

$$p(z_t \mid x_t, z_{t-1}) = \eta\, (\mathcal{N}(z_t; z_{t-1} + x_{t,p}, \Sigma_r + 2\Sigma_e) + k),$$

again using the fact that the convolution of two Gaussians is another Gaussian.

To compute the measurement model in practice, let $\bar{z}_{t-1} = z_{t-1} + x_{t,p}$; in other words, let $\bar{z}_{t-1}$ be the points z_{t-1} shifted by the position variable in the state x_t. Then, for each point $z_j \in z_t$, let $\bar{z}_i$ be the closest corresponding point in $\bar{z}_{t-1}$. We then compute the measurement model probability as

$$\begin{aligned} &p(z_t \mid x_t, z_{t-1}) \\ &= \eta \left(\prod_{z_j \in z_t} \exp\left(-\frac{1}{2} (z_j - \bar{z}_i)^T \Sigma^{-1} (z_j - \bar{z}_i) \right) + k \right) \end{aligned} \tag{11}$$

where η is a normalization constant and k is a smoothing factor. The covariance matrix Σ is given by $\Sigma = 2\Sigma_e + \Sigma_r$, with covariance terms for Σ_e due to sensor noise and Σ_r due to the sensor resolution. To perform this computation, we first shift the previous points z_{t-1} by the proposed velocity x_t to obtain the shifted points $\bar{z}_{t-1}$. Then, for each point z_j in the current frame, we find the corresponding nearest shifted point $\bar{z}_i \in \bar{z}_{t-1}$. Given these correspondences, the measurement model can then be computed using equation 11.

5.2. Tracking with color

Our 3D-based model can be used even if no color information is available. However, if color is available (such as from a video camera), we can incorporate color matches into our probabilistic model. To leverage color, we learn the probability distribution over color for correctly aligned points. To do so, we build a large dataset of correspondences (with a 5 cm maximum distance) between colored laser returns from one laser spin and each of their spatially nearest points from the subsequent spin, aligned using our recorded ego motion. An example visualization of tracking a single point is shown on our project page at `http://stanford.edu/~davheld/anytime_tracking.html`. By observing how the color of this point changes as we move past it, we can learn a probability distribution for color changes over a single frame.

We build a normalized histogram of the differences in color values between each point and its closest neighbor from the next spin. The difference histogram we obtain is shown in Figure 5. The distribution closely follows a Laplacian distribution, as expected (Huang and Mumford, 1999; Sun et al., 2008; Odom and Milanfar, 2006).

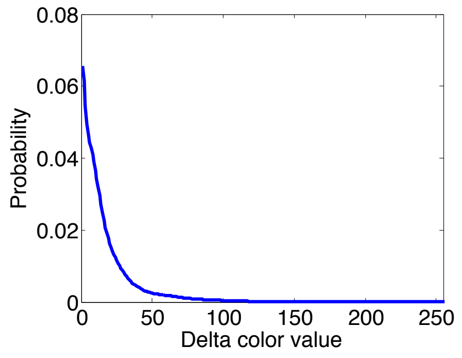

Fig. 5. The probability that the color of a point will change by some amount over one frame.

In theory, we could incorporate multiple color channels into our model. However, such a model would require us to learn the covariances between different color channels. Instead, we simplify the model by incorporating just a single color channel (blue), chosen using a hold-out validation set. Although adding other color channels could provide additional benefit, we show improved tracking performance with just one color channel alone.

We can now incorporate the learned color distribution into the measurement model derived in Section 5.1. The term $p(s_{t,j} \mid s_{t-1}, V)$ from equation 4 represents the probability of sampling point $s_{t,j}$ given that the surface from which it is sampled was previously visible at time $t-1$. We now expand this term as a product of spatial and color probabilities:

$$p(s_{t,j} \mid s_{t-1}, V) = p_s(s_{t,j} \mid s_{t-1}, V)\, p_c(s_{t,j} \mid s_{t-1}, V) \tag{12}$$

where $p_s(s_{t,j} \mid s_{t-1}, V)$ represents the probability based on the spatial match with the points in s_{t-1}, and $p_c(s_{t,j} \mid s_{t-1}, V)$ represents the probability based on the color match with the points in s_{t-1}. As before, we model the spatial match probability as $p_s(s_{t,j} \mid s_{t-1}, V) = \mathcal{N}(s_{t,j}; s_{t-1,i}, \Sigma_r)$, where Σ_r models the variance resulting from the sensor resolution as well as from object deformations and $s_{t-1,i}$ is the nearest corresponding (latent) surface point from the previous frame.

For the color match probability, we consider that, due to changes in lighting, lens flare, or other unmodeled causes, the observed colors of an entire image can change drastically between two frames. We thus propose that there is some probability $p(\neg C)$ that all of the the observed colors in an image will change in a way that is not modeled by the learned color distribution from Figure 5. We can then write the color match probability from equation 12 as

$$\begin{aligned} p_c(s_{t,j} \mid s_{t-1}, V) = &p(C)\, p_c(s_{t,j} \mid s_{t-1}, V, C) + \\ &p(\neg C)\, p_c(s_{t,j} \mid s_{t-1}, V, \neg C) \end{aligned} \tag{13}$$

where $p_c(s_{t,j} \mid s_{t-1}, V, C)$ is the learned color model distribution from Figure 5 (parameterized as a Laplacian) and $p_c(s_{t,j} \mid s_{t-1}, V, \neg C) = 1/255$ is the probability of a point having any color, given that the color does not need to match to that of a nearby point from the previous frame.

The parameter $p(C)$, the probability that the color should match between two aligned points, must be chosen with care. Some thought reveals that, when we are coarsely sampling the state space (see Section 6), we do not expect the colors to match very well. Therefore, we set $p(C)$ to be a function of the sampling resolution, as

$$p(C) = p_c \exp\left(\frac{-r^2}{2\sigma_c^2}\right) \tag{14}$$

where r is the sampling resolution and σ_c is a parameter that controls the rate at which $p(C)$ decreases with increasing resolution. Thus, when we are sampling coarsely, we get a smaller value for $p(C)$, meaning we do not expect the colors to match at a coarse resolution. As we sample more finely, $p(C)$ increases until $p(C) = p_c$ when $r = 0$, so that we expect the colors to match more precisely at a finer sampling resolution.

In practice, we compute the measurement model incorporating color as follows: As before, let $\bar{z}_{t-1} = z_{t-1} + x_{t,p}$; in other words, let $\bar{z}_{t-1}$ be the points z_{t-1} shifted by the position variable in the state x_t. Then, for each point $z_j \in z_t$, let $\bar{z}_i$ be the closest corresponding point in $\bar{z}_{t-1}$. For each point, we compute the spatial probability as

$$p_s(z_j \mid x_t, z_{t-1}) = \exp\left(-\frac{1}{2}(z_j - \bar{z}_i)^T \Sigma^{-1} (z_j - \bar{z}_i)\right)$$

We compute the color probability as

$$\begin{aligned} p_c(z_j \mid x_t, z_{t-1}, V) =& p(C)\, p_c(z_j \mid \bar{z}_i, V, C) + \\ & p(\neg C)\, p_c(z_j \mid \bar{z}_i, V, \neg C) \end{aligned}$$

where $p(C)$ is given by equation 14, $p_c(z_j \mid \bar{z}_i, V, C)$ is given by the the learned color model distribution from Figure 5 (parameterized as a Laplacian), $p(\neg C) = 1 - p(C)$, and $p_c(z_j \mid \bar{z}_i, V, \neg C) = 1/255$. Finally, we compute the total measurement probability as

$$\begin{aligned} p(z_t \mid x_t, z_{t-1}) =& \eta \Bigg(\prod_{z_j \in z_t} p_s(z_j \mid x_t, z_{t-1})\, p_c(z_j \mid x_t, z_{t-1}, V) + \\ & k_3 \left(k_4 - p_s(z_j \mid x_t, z_{t-1})\right) \Bigg) \end{aligned}$$

where k_3 can be computed from k in equation 11 as $k_3 = k/(k+1)$. The parameter k_4 is a smoothing parameter that must be chosen via cross-validation, and in our case we set it to 1.

5.3. Motion Model

Unlike ICP or a scan-matching algorithm, we also incorporate a motion model into our tracker. As we will show, adding a motion model significantly improves tracking performance. To build the motion model, we take all of the values for $p(x_t \mid z_1 \ldots z_t)$ from the previous frame and fit a multi-variate Gaussian to the set of probabilities. We compute the mean μ_t and covariance Σ_t by weighting each state by its probability as

$$\begin{aligned} \mu_t &= \sum_i p(x_{t,i} \mid z_1 \ldots z_t)\, x_{t,i} \\ \Sigma_t &= \sum_i p(x_{t,i} \mid z_1 \ldots z_t)(x_{t,i} - \mu_t)(x_{t,i} - \mu_t)^T \end{aligned}$$

where $x_{t,i}$ is the state vector of sample i. Once a Gaussian over the posterior is obtained, the result is used in the standard constant velocity model of a Kalman filter. This is a standard method, so we refer the reader to a basic text on the subject for details (Thrun et al., 2005).

5.4. Implementation

In Held et al. (2014), we compute the measurement model by using a kd-tree to search for the nearest-neighbor for each point. We also divide the space into a grid and cache the log probability for each grid cell after each search. Thus, for each grid-cell, we only perform a single search in the kd-tree. Afterwards, we simply perform a quick table lookup of the cached result. In our implementation, we set the discretization of the measurement grid equal to the state-space sampling resolution (described in Section 6).

Here we present an alternative implementation that allows our tracker to run even faster. Rather than caching the kd-tree search results, we instead use a pre-caching technique. We divide the space into a grid, as before. Then, for each point $z_i \in z_{t-1}$, we pre-cache in each grid cell the probability of a point from z_t falling into such a grid cell, using the measurement model of equation 11. Because all points for a given object have the same covariance Σ, much of the work for this computation can be done once and then reused for all points. Using pre-caching, we avoid having to perform relatively costly kd-tree searches. We will show that this pre-caching technique gives us a large speedup over performing kd-tree searches, with no loss in accuracy.

Additionally, we note that equation 11 is not symmetric with respect to z_t and z_{t-1}, especially if the two point clouds are different sizes. Specifically, suppose that z_t is much larger than z_{t-1} due to points becoming unoccluded. In such a case, we will incur a large penalty for each point in z_t that does not have a nearby match in z_{t-1}. Thus, our tracker will focus on aligning the densest part of z_t in order to minimize the number of unmatched points. This will often lead to an incorrect alignment estimate.

We can rectify this problem by choosing the larger of z_t and z_{t-1} (in terms of the number of points) to play the "role" of z_{t-1} and the smaller point cloud will play the "role" of z_t in equation 11. Thus, as described above, we divide the space into a grid, and we pre-cache the larger of z_t and z_{t-1} in this grid. We then iterate over each point from the smaller of z_t and z_{t-1} and look up probabilities in the grid to compute the measurement model of equation 11. By performing the computation in this manner, we are able to get a large improvement in performance, as we will show. It is important to remember, when implementing this method, that the resulting alignment will give the opposite of the actual velocity if z_{t-1} and z_t were switched to perform this computation.

6. Annealed Dynamic Histograms

Using the techniques described above, the measurement model and motion model probabilities are very quick to compute for any particular candidate alignment between frames. However, we must compute these probabilities separately for every state x_t considered. If we densely sample the state space, then there will be a large number of computations to perform, rendering this method too slow for real-time use. In order to enable our method to globally explore the state space in real-time, we introduce a new technique called *annealed dynamic histograms*, which we will explain below.

6.1. Derivation

Our overall approach to dynamic histograms can be visualized in Figure 6. We start by coarsely dividing the state space into grid cells and computing the probability $p(x_t|z_1 \ldots z_t)$ for each cell. We then recursively expand some of the cells, subdividing each cell into k sub-cells. We now derive a method for deciding which cells to divide and for computing the probability of each of the new sub-cells.

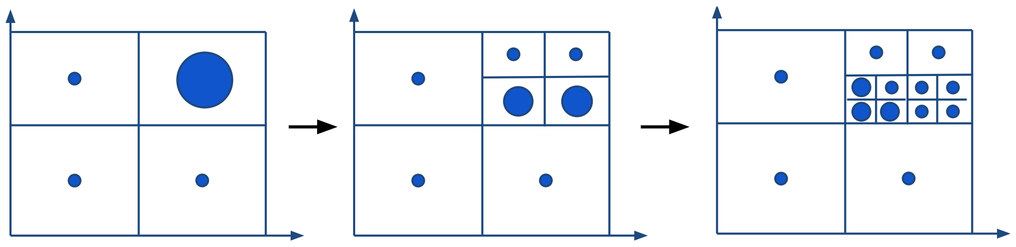

Fig. 6. We decompose the state space using annealed dynamic histograms. Here we show the dynamic decomposition, starting from a coarse sampling on the left and refining the distribution over time. The size of the circle is proportional to the sample's alignment probability.

The first step is to divide the state space into a coarse grid. We next choose some cells to subdivide into k sub-cells, based on a criteria that we will establish. Let R be the (possibly non-contiguous) set of all cells that we choose to subdivide into sub-cells c_i. We can compute the discrete probability of the sub-cell $c_i \in R$ as follows:

$$\begin{aligned} p(c_i) &= p(c_i \cap R) \\ &= p(c_i \mid R)\, p(R) \\ &= \frac{p(x_i \mid z_1 \dots z_t)|c_i|}{\sum_{j \in R} p(x_j \mid z_1 \dots z_t)|c_i|}\, p(R) \\ &= \eta\, p(x_i \mid z_1 \dots z_t)\, p(R) \end{aligned}$$

where $|c_i|$ is the volume of sub-cell c_i. We thus have the property that $\sum_{i \in R} p(c_i) = p(R)$. The key here is that the normalization constant η depends only on other sub-cells in region R. Thus, the probability values of all cells outside of region R are unaffected by this computation.

We can now derive the criteria for choosing which cells to divide, based on minimizing the KL-divergence between our histogram and the true posterior. Suppose distribution B is the current estimated distribution before we divide a given grid cell, and distribution A is the new estimated distribution after we divide the grid cell into k sub-cells. In distribution A, the k new sub-cells can each take on separate probabilities, allowing us to more accurately approximate the true posterior. The KL-divergence between distributions A and B measures the difference between the old, coarser approximation to the posterior and the new, improved approximation to the posterior. The size of the KL-divergence indicates how much we can improve our approximation to the posterior by dividing a given grid cell.

Specifically, suppose that we have a cell whose discrete probability is P_i, and we are deciding whether to divide this cell. Before dividing, we can view the cell as having k sub-cells, each of whose probability is constrained to be P_i/k. After dividing, each of the sub-cells can take on a new probability p_j, with the constraint that $\sum_{j=1}^{k} p_j = P_i$. This situation is illustrated in Figure 7.

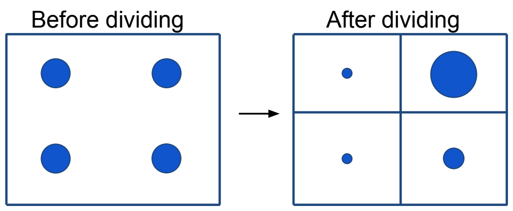

Fig. 7. Before dividing a cell, we can view it as having sub-cells whose probabilities are all constrained to be equal. After dividing, each of the sub-cells can take on its own probability.

The KL-divergence between these two distributions can be computed as

$$D_{KL}(A||B) = \sum_{j=1}^{k} p_j \ln\left(\frac{p_j}{P_i/k}\right)$$

where B is the distribution before dividing and A is the distribution after dividing. The KL-divergence will obtain its maximum value if $p_{j'} = P_i$ for some j' and $p_j = 0$ for all $j \neq j'$. In this case, we have

$$D_{KL}(A||B) = P_i \ln k \tag{15}$$

If the computation for each of the k sub-cells takes t seconds, then the maximum KL-divergence per second is then equal to $P_i \ln k/(k\,t)$. If our goal is to find the histogram that matches as closely as possible to the true posterior, then we should simply divide all cells whose probability P_i exceeds some threshold p_{min}. Similarly, to achieve the maximum benefit per unit time, we should choose k to be as small as possible. For a histogram in d dimensions, we use $k = 3^d$, splitting cells into thirds along each dimension.

6.2. Annealing

Initially, when we sample the state space at a very coarse resolution, we would not expect to find a good alignment between the previous frame and the current frame for the tracked object. In fact, the sampling resolution (shown in Figure 6) introduces another source of error into our model. We thus increase the variance of the measurement model Gaussian by some amount Σ_g, proportional to the resolution of the state-space sampling. Our measurement model variance now becomes $\Sigma = 2\Sigma_e + \Sigma_r + \Sigma_g$. As we sample regions of the state space at a higher resolution, as shown in Figure 6, Σ_g decreases towards 0. The measurement model is thus accordingly annealed as we refine our sampling of the state space, motivating the name for our method of “annealed dynamic histograms." The complete method can then be implemented as shown in Algorithm 1.

6.3. Using the Tracked Estimate

The tracker can return information about the tracked object’s velocity in any format, as requested by the planner or some other component of the system. For example, to minimize the RMS error, as in Section 8.2, we return the mean of the posterior. On the other hand, to build accurate object models, as in Section 8.3, we use the mode of the distribution. A simple planner might request either the mean or the mode of the posterior distribution, as well as the variance. A more sophisticated planner could make use of the entire tracked probability histogram, in the form of a density tree (Thrun et al., 2001).

We will now demonstrate that the mean of the distribution will minimize the RMS error. Assume that the true distribution over velocities is $p(v)$, where the distribution is based on the probabilistic noise in our measurement (from the sensor noise and the sensor resolution). To minimize the RMS error, we want to find the velocity $\hat{v}$ that minimizes

$$\min_{\hat{v}} \mathbb{E}(\hat{v} - v)^2$$

where we take the expectation with respect to $p(v)$. To minimize this, we first rewrite the expectation as a sum:

$$\min_{\hat{v}} \sum_{v} p(v)(\hat{v} - v)^2$$

```
Input  : Initial coarse tracking hypotheses $H_0$, initial sampling resolution $g_0$, desired sampling resolution $g_{des}$
Output: A set of cells $c_i$ and probabilities $p(c_i)$
$H_{new} \leftarrow H_0, g \leftarrow g_0, p(R) \leftarrow 1$;
while $g > g_{des}$ do
    $\Sigma_g \leftarrow g\mathrm{I}$;
    $\Sigma \leftarrow 2\Sigma_e + \Sigma_r + \Sigma_g$;
    /* Compute probability of velocities                                    */
    for each cell $c_i \in H_{new}$ do
        $\hat{x}_{t,i}$ = center of cell $c_i$;
        Compute $p(z_t \mid \hat{x}_{t,i}, z_{t-1})$ from equation 11;
        Compute $p(\hat{x}_{t,i} \mid z_1 \dots z_{t-1})$ from the motion model;
        $\tilde{p}(\hat{x}_{t,i} \mid z_1 \dots z_t) \leftarrow p(z_t \mid \hat{x}_{t,i}, z_{t-1})\, p(\hat{x}_{t,i} \mid z_1 \dots z_{t-1})$; // Unnormalized probability
    end
    /* Normalize probabilities                                              */
    $\eta \leftarrow \sum_{c_i \in H_{new}} \tilde{p}(\hat{x}_{t,i} \mid z_1 \dots z_t)$;
    for each cell $c_i \in H_{new}$ do
        $p(c_i) \leftarrow \eta\, \tilde{p}(\hat{x}_{t,i} \mid z_1 \dots z_t)\, p(R)$
    end
    /* Finely sample high probability regions                               */
    $H_{new} = \emptyset$;
    for each cell with $p(c_i) > p_{min}$ do
        Subdivide cell $c_i$ by $k$ along each dimension and add to $H_{new}$;
    end
    $p(R) \leftarrow \sum_{c_i \in H_{new}} p(c_i)$;
    $g \leftarrow g/k$;
end
```

Algorithm 1: ADH Tracker

We then take the derivative with respect to $\hat{v}$ and set it equal to 0 to get

$$\begin{aligned}
&2\sum_v p(v)(\hat{v} - v) = 0 \\
&\sum_v p(v)\hat{v} = \sum_v p(v)v \\
&\hat{v}\sum_v p(v) = \sum_v p(v)v \\
&\hat{v} = \mu_v
\end{aligned}$$

where we have used the fact that $p(v)$ is a probability distribution, so $\sum_v p(v) = 1$. The expression $\sum_v p(v)v$ is simply the mean of v, μ_v, thus demonstrating that the mean of the distribution, rather than the mode, will minimize the RMS error. Depending on which objective is important for a particular application, the tracker can return the mean, the mode, or the entire distribution, as described above.

6.4. Estimating Rotation

For some applications, the rotation of the tracked object might need to be estimated in addition to the translation. To estimate the rotation, we first find the mode of the posterior distribution over the translation. We then perform coordinate descent in each rotation axis, holding translation fixed. Based on the application, we can search over yaw only, or we can also search over roll and pitch. Because we are searching separately over each axis of rotation, this optimization is relatively quick to perform. We incorporate a search over rotation when we evaluate our tracker by building 3D object models in Section 8.3.

7. System

We obtain the 3D point cloud used for tracking with a Velodyne HDL-64E S2 LIDAR mounted on a vehicle. The Velodyne has 64 beams that rotate at 10 Hz, returning 130,000 points per 360 degree rotation over a vertical range of 26.8 degrees. We color these points with high-resolution camera images obtained from 5 Point Grey Ladybug-3 RGB cameras, which use fish-eye lenses to capture 1600x1200 images at 10 Hz. The laser is calibrated using the method of (Levinson and Thrun, 2010). The vehicle pose is obtained using the Applanix POS-LV 420 inertial GPS navigation system. Experiments were performed single-threaded on a 2.8 GHz Intel Core i7 processor.

8. Results

In order to measure the robustness of our tracker, we must evaluate on a large number of tracked objects. Therefore, we cannot use the evaluation methods of Manz et al. (2011) and Moosmann and Stiller (2013), in which a small number of tracked vehicles are equipped with a measuring apparatus. Instead, we propose two methods to automatically evaluate our velocity estimates on a large number of tracked objects. First, using the evaluation method of Held et al. (2013), we track parked cars in a local reference frame in which they appear to be moving. This allows us to directly measure the accuracy of our velocity estimates, described in Section 8.2. We also evaluate our tracker by building models of tracked objects using our estimated velocity. We compute a crispness score, as in Sheehan et al. (2012), to compare how correctly the models were constructed. We use this method to evaluate our tracking accuracy on a large number of people, bikes, and moving cars, described in Section 8.3. Using these two evaluation methods, we are able to determine the robustness of our tracker on a wide variety of objects of different types and in different conditions.

8.1. Choosing parameters

Parameters for our model, as well as parameters for the baseline methods, were chosen by performing a grid search using a training set that is completely separate from our test set. The training set consists of 13.6 minutes of logged data, during which time we drove past a total of 622 parked cars. Using this training set, the final parameters chosen for our system are as follows, for an object with a horizontal sensor resolution of r meters and a state-space sampling resolution of g: $k = 0.8$, $\Sigma'_e = 2\Sigma_e = (0.03\,\text{m})^2\text{I}$, Σ_r = (r/2)I, and $\Sigma_g = g\text{I}$, where I is the 3x3 identity matrix. For the dynamic histogram, we initialize from a coarse resolution of 1 m, and we continue until the resolution of the resulting histogram is less than max(r, 0.05 m), with $p_{min} = 10^{-4}$. For our color model, we have $p_c = 0.05$ and $\sigma_c = 1$ m. Because our data contains many frames with lens flare as well as underexposed frames, our system learns to assign a low priority to color matches. Our learned color distribution for aligned points is given by a Laplacian with parameter $b = 13.9$. For all methods tested, we downsample the current frame's point cloud to no more than 150 points and the previous frame to no more than 2000 points, where the current frame z_t and previous frame z_{t-1} are chosen as described in Section 5.4.

8.2. Evaluation: Relative Reference Frame

Dataset. The first approach we take to quantitatively evaluate the results of our tracker is to track parked cars in a local reference frame in which they appear to be moving, as is done in Held et al. (2013). In our 6.6 minute test dataset, we drive past 557 parked cars. However, in a local reference frame, each of these parked cars appears to be moving in the reverse direction of our own motion. Because we have logged our own velocity, we can compute the ground-truth relative velocity of a parked vehicle in this local reference frame and quantitatively evaluate the precision of our tracking velocity estimates. Furthermore, as we drive past each car, the viewpoint and occlusions change over time. We are thus able to evaluate our method's response to many real-world challenges associated with tracking. We will show in Section 8.3 further quantitative results when tracking moving objects of different classes (cars, bikes, and pedestrians), to demonstrate that our method works on moving objects and generalizes across object types.

During the 6.6 minutes of our test set, our trajectory follows the path shown in Figure 8. During this test set, the relative motion of vehicles that we are tracking take a range of linear and angular velocities, as shown in Figure 9. The mean linear velocity is 5.6 m/s, with a standard deviation of 4.2 m/s. This range of linear velocities is the result of two effects. First, the ego vehicle changes its own velocity throughout the dataset, causing the relative velocity of nearby vehicles to change accordingly. Second, the ego vehicle makes a number of rotations throughout the dataset, as can be seen in Figure 8. When the ego vehicle rotates, the relative motion of nearby objects will vary based on their distance to the ego vehicle. These two effects combine to produce a range of relative velocities for the different vehicles in our test set, as shown in Figure 9.

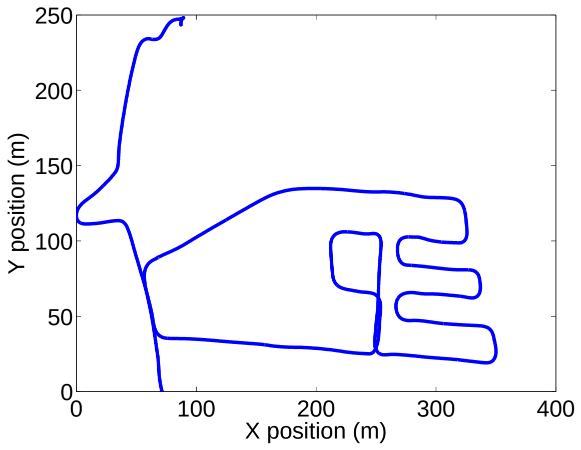

Fig. 8. Path taken by our vehicle while recording the data used for our test set.

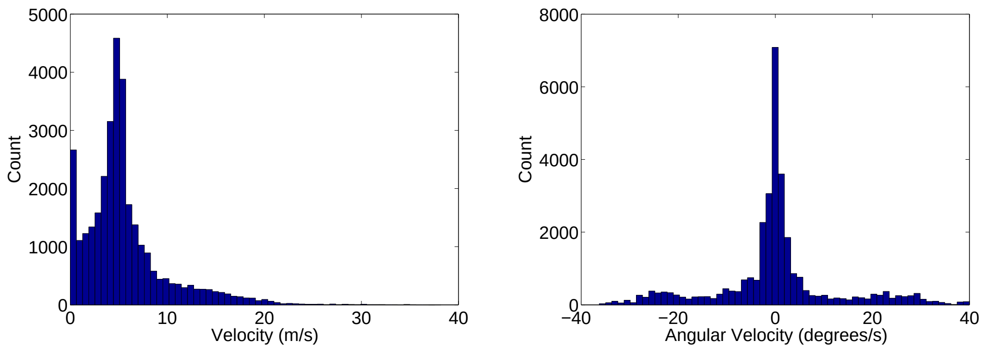

Fig. 9. Distribution of relative velocities of vehicles in our test set. Left: Linear velocities. Right: Angular velocities.

We ignore tracks that contain major undersegmentation errors, in which the tracked object is segmented together with another object, as this leads to an ambiguity about the correct ground-truth alignment. We thus filter out 7% of our initial tracks based on segmentation issues, leaving us with 515 properly segmented cars to track. We do not filter out tracks in which the tracked object has been oversegmented into multiple pieces.

Baseline Comparison. To compare the robustness of different tracking methods, we compute the RMS tracking error for each method. The results are shown in Figure 10. First, we evaluate a Kalman filter with a measurement model given by the centroid of the tracked points. Because of its speed and ease of implementation, this is a popular method, used in Levinson et al. (2011) and Kaestner et al. (2012). As can be seen in Figure 10, this method is extremely fast, completing in less than

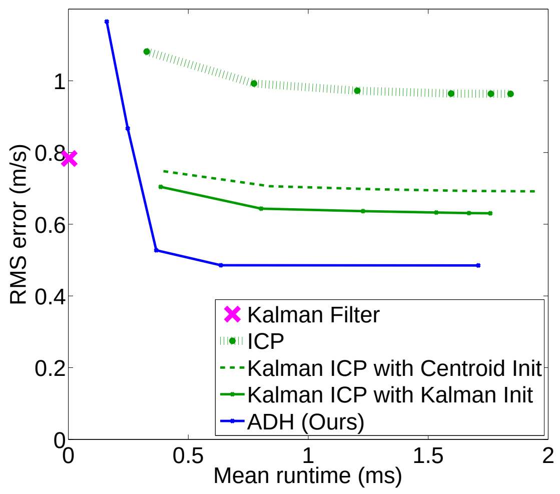

Fig. 10. RMS error vs runtime of our method compared to several baseline methods. The centroid-based Kalman filter runs in 0.004 ms, which is difficult to visualize on the scale of this plot. Note that the method "Kalman ICP with Kalman Init" is a novel baseline method. The ADH tracker is more accurate than the baselines while still running in real time.

0.1 milliseconds. However, the method is not very accurate, producing an RMS error of 0.78 m/s across the 515 cars in our test set.

Next, we evaluate a number of variants of ICP, the iterative closest point method. First, we evaluate the basic point-to-point ICP algorithm, using the implementation from PCL (Rusu and Cousins, 2011). This basic method is used for tracking by Feldman et al. (2012). As is standard, we initialize ICP by aligning the centroids of the tracked object. We run ICP for 1, 5, 10, 20, 50, and 100 iterations, and the results are shown in Figure 10. It is clear from Figure 10 that the basic ICP algorithm does not perform well for tracking, with an RMS error 23% worse than that of a simple Kalman filter. As we will show, this decrease in accuracy is because the standard ICP algorithm does not make use of a motion model, which is crucial for robust tracking.

We next compare to a Kalman filter with ICP used as the measurement model. Combining ICP with the motion model in a Kalman filter makes the method much more robust to failures of ICP. Because ICP is dependent on its initialization, we test three different strategies to initialize this method. First, we try initializing ICP by aligning the centroids of the tracked object, as we did above. We also try initializing ICP using the mean prediction from the motion model, as was done by Moosmann and Stiller (2013). Last, we try first running the centroid through a Kalman filter and using the output as the initialization for ICP.

Figure 10 compares these three methods. Initializing using the mean prediction from the motion model is not shown on this plot because the performance is significantly worse than the other methods tested, giving an RMS error of 2.6 m/s. An analysis of why this occurs reveals that this method performs well when the tracked object is moving at a relatively constant velocity, but it performs poorly when the object is quickly accelerating or decelerating. Thus, initializing ICP with the mean prediction of the motion model is a poor choice for tracking objects that can quickly change velocity.

Using a Kalman filter with ICP, initialized by aligning the centroids, performs reasonably, with an improvement of 11.7% over a simple Kalman filter. Finally, if we run the centroid through a Kalman filter and use the output as the initialization for ICP (and use the ICP result as the measurement model for another Kalman filter), then we get the best performance of the ICP baseline methods that were tested, with an RMS error of 0.63 m/s. To the best of our knowledge, this is a novel baseline method that has not been tested previously in the tracking literature. However, all versions of ICP suffer from the problem of choosing a good initialization. Our method, in contrast to all of the ICP methods, is more robust in that does not depend on the initialization.

We also compare to the method of Held et al. (2013), without using color. This method performs poorly compared to our baselines, giving an RMS error of 1.04 m/s. The main reason for this poor performance is that this method does not make use of a motion model. As has already been shown, adding a motion model makes a significant difference when tracking moving objects. This method also takes an average of 86 milliseconds per frame and hence is probably not appropriate for a real-time system.

Our method achieves an accuracy of 0.53 m/s after running for just 0.37 milliseconds. Even at this minimum runtime, our method performs 32.7% better than a simple Kalman filter and 25% better than all of the baseline methods of similar speed, as shown in Figure 10. If our method is allowed to run for 0.64 milliseconds, our method achieves an accuracy of 0.49 m/s, which is 23% better than all baseline methods that were tested. Note also that this is an average over 515 tracked vehicles, and that the tracking accuracy varies as a function of distance. For example, for objects within 5 m, our method achieves an RMS error of 0.15 m/s, whereas for objects 65 m away, our method achieves an RMS error of 1.4 m/s.

If color images are available, then we can easily extend our measurement model to incorporate color, as described in Section 5.2. This is one advantage of our derivation of the measurement model, as opposed to the more heuristic justification of Olson (2009) and Thrun et al. (2005). With the addition of color, our RMS error decreases by an additional 10.4%, achieving an RMS error of 0.43 m/s. Note that many frames in our test set contain heavy shadows or lens flare. We thus expect color to make an even bigger difference when lens flare and similar exposure problems are avoided.

It is interesting to compare the performance of our method to that of a radar, which can also be used to measure the velocity of moving objects. The Bosch LRR3 Radar can estimate velocities to within 0.12 m/s (*Chassis Systems Control LRR3: 3rd generation Long-Range Radar Sensor*, 2009). However, a radar only estimates velocity in a single dimension: in the direction from the radar to the tracked object. This one-dimensional estimate is of limited use for robotics or autonomous driving, in which we need a 2D estimate of the velocity of each object to estimate where each object is moving. Our method returns a 2D velocity estimate, and by searching over vertical motion and over rotations our method can be made to return a 6D velocity estimate.

In this paper, we modified our method to align the smaller set of points to the larger set of points, as explained in Section 5.4, instead of always aligning the more recent points z_t to the points from the previous frame z_{t-1}. By aligning the smaller set of points to the larger set, we get a 10% improvement in RMS error, from 0.54 m/s to 0.49 m/s. If we are using the version of our method that incorporates color, then the improvement is even bigger at 12.6%, from 0.50 m/s RMS error to 0.43 m/s. This change is mostly responsible for the large drop in the error of our current method compared to the older version published in Held et al. (2014).

We also described in Section 5.4 that we are now using a fast pre-caching scheme to compute the measurement model, as opposed to using kd-tree searches as in Held et al. (2014). The resulting method is almost 3 times faster at the same level of resolution, reducing the speed from 1.8 milliseconds per frame to 0.64 milliseconds per frame at a sampling resolution of 3.7 cm. By avoiding kd-tree searches and re-using the covariance computations for each point, we are able to obtain a significant speedup.

Sampling Analysis. One of the novel additions of our method is the use of annealed dynamic histograms, as described in section 6, to speed up our tracker. In Figure 11, we show the decrease in speed if we were to densely sample the search space. Densely sampling is about 138 times slower for approximately the same level of accuracy. When searching over alignments, we expand all cells whose probability exceeds a minimum threshold P_{min}. If we were to instead only expand the single highest probability cell on each step, our RMS error would increase by 46.3%.

We also compare to the multi-resolution model from Olson (2009). Olson computes a low-resolution model (0.3 m) and a high-resolution model (0.03 m). The low-resolution model is computed by taking a maximum over the high-resolution regions. Sampling then alternates between the low and high resolution models until the method has found the maximum of the high resolution model. Due to the way in which the low resolution model is constructed, this model is guaranteed

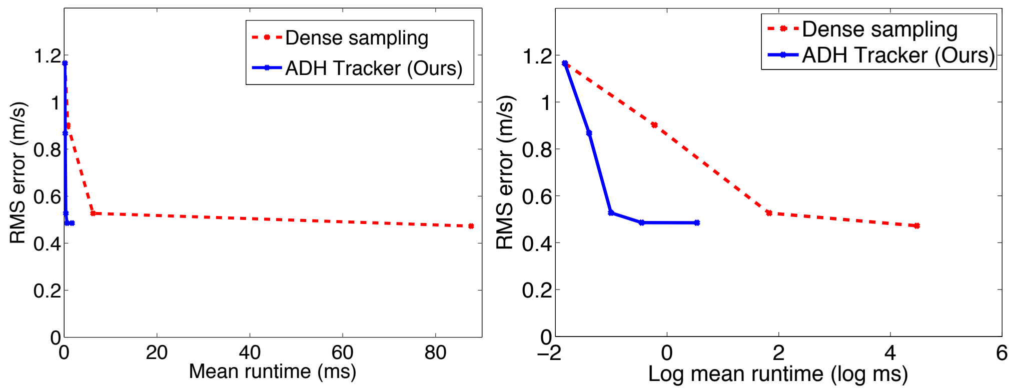

Fig. 11. Left: Accuracy vs mean runtime using annealed dynamic histograms compared to densely sampling the state space. Right: Same plot visualized on a log time scale.

to find the global maximum of the high resolution model. Indeed, if we sample from the search space using the method from Olson (2009) and take the mean of the resulting distribution, we get an RMS accuracy of 0.49 m/s, which is the same accuracy that we get by sampling with the ADH tracker.

However, the sampling method of Olson (2009) is much slower than the sampling method of the ADH tracker. The ADH tracker takes about 0.64 ms per object per frame, whereas the tracker of Olson (2009) takes 14.5 ms per object per frame, or a 24 times reduction in speed. The main reason that the Olson tracker is slower is that it requires more samples from the state space to find the mode, requiring an average of 3899 samples, whereas the ADH tracker uses an average of only 172 samples. The ADH tracker thus requires 23 times fewer samples, which almost completely accounts for the difference in runtime. Furthermore, if a hard limit is placed on the number of samples taken by the Olson tracker, the error rate increases dramatically, indicating that early-stopping to decrease the runtime is not a viable option for the Olson tracker.

There are multiple differences between the sampling methods of the ADH tracker and that of the Olson tracker. One difference is that the ADH tracker can sample at many levels of resolution. In our implementation, the ADH tracker begins with a coarse sampling of 1 m. The tracker then repeatedly reduces the sampling resolution by a factor 3, thus sampling at 0.33 m, then 0.11 m, and finally at 0.03 m. In contrast, the Olson tracker samples at only 2 resolutions: first a coarse sampling at 0.3 m, followed by a fine sampling at 0.03 m.

To understand how important this difference was, we first modified the ADH tracker so that it also samples only at 2 resolutions: a coarse sampling of 0.3 m followed by a fine sampling of 0.03 m, just like the Olson tracker. Despite this change, the ADH tracker is still more efficient, running in 2 ms, or a speedup of a factor of 7 over the Olson tracker. This speedup factor of 7 must then be based on a fundamental difference in the sampling method between the Olson and ADH trackers. We can thus conclude that the ADH tracker gains a 3.3 times speedup over the Olson tracker by using a 4-level resolution model rather than a 2-level resolution model, and the ADH tracker gains an additional 7 times speedup based on the method of sampling alone.

The cause of the factor of 7 speedup over the Olson tracker is based on the way in which samples are generated. To sample at a coarse resolution, the Olson tracker takes a maximum over the corresponding high resolution regions. In contrast, the ADH tracker increases the noise of the measurement model by an amount proportional to the sampling resolution. Taking a maximum as in the Olson tracker provides an extreme level of smoothing and thus too many samples at the coarse resolution appear to have a high probability. In contrast, increasing the measurement noise in the ADH tracker still leads to an informative distribution at the coarse resolution, in that low probability regions at a high resolution tend to correspond to low probability regions at the coarse resolution. Thus, in the ADH tracker, the coarse resolution is more informative about which regions are most likely to contain the mode of the high resolution distribution. As a result, the ADH tracker requires fewer samples to find the mode.

A further factor of 3.3 speedup is obtained using a 4-level sampling resolution model rather than just the 2-level sampling resolution model of the Olson tracker, as explained above. Unfortunately, it is not possible to simply extend the Olson tracker to more than 2 levels of resolution. The Olson tracker uses the coarse resolution to find the mode of the higher resolution, and the method is considered to be converged once the mode of the higher resolution is discovered. In a 4-level sampling resolution model, the lowest resolution model would continue sampling until it has found the mode of the 2nd lowest resolution model. However, this mode does not necessarily correspond to the mode of the highest resolution model, so stopping at this point would not be helpful. Further, the method is already much slower than the ADH tracker, so continuing to sample would also not be practical. Thus, the Olson tracker suffers by not being extensible to more than a 2-level sampling resolution model, whereas the ADH tracker naturally extends to many levels of sampling resolution.

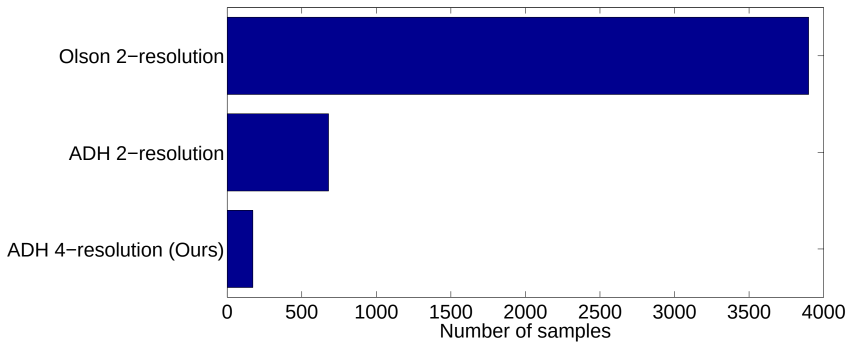

Fig. 12. Number of samples required by different methods to create the same tracking accuracy. Top: The tracker of Olson (2009). Middle: Our tracker, modified to have only 2 levels of sampling resolution. Bottom: Our tracker, using our full model with 4 levels of sampling resolution.

These results are summarized in Figure 12. The method of Olson (2009) requires an average of 3899 samples to achieve an accuracy of 0.49 m/s. The ADH tracker, modified to have only 2 levels of sampling resolution like Olson (2009), requires only 680 samples on average to achieve the same accuracy. Finally, our full method, with 4 levels of sampling resolution, requires only 172 samples to achieve the same level of accuracy. As explained above, the runtime is proportional to the number of samples, so the greater number of samples required by Olson (2009) leads to a significantly slower method, which can make the method not feasible for real-time tracking in autonomous driving applications. A visualization of the number of samples required by the three methods are shown for a specific example in Figure 13.

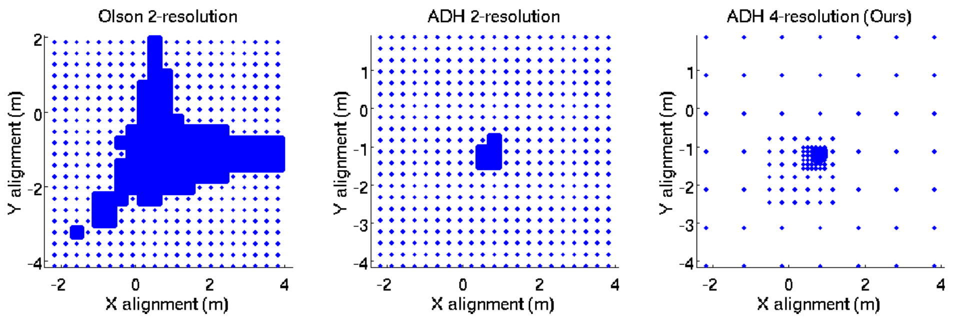

Fig. 13. Visualization of the number of samples required by three different sampling methods to achieve the same level of accuracy. Each point represents a single sample. Regions with a large number of closely-spaced samples appear as a solid color. Left: The tracker of Olson (2009). Middle: Our tracker, modified to have only 2 levels of sampling resolution. Right: Our tracker, using our full model with 4 levels of sampling resolution.

Error Analysis. In order to better understand the performance of our tracker, we evaluate how the performance varies as a function of the number of points observed by our 3D sensor on the tracked object. The results are shown in Figure 14. As shown, the RMS error decreases as the number of tracked points increases, and our ADH tracker outperforms the centroid-based Kalman filter baseline for any number of points. The two methods have similar performance when the number of points is small, since the ADH Tracker cannot take advantage of the 3D shape when there are not many visible points on the tracked object. As the number of visible points increases, the ADH Tracker is able to increase its tracking accuracy. The accuracy of the Kalman filter also improves as the number of points increases, probably due to the decreased occlusions for close objects which also have a large number of points. The difference in performance between the two methods shows the benefit of using the full 3D shape for varying numbers of observed points on the tracked object.

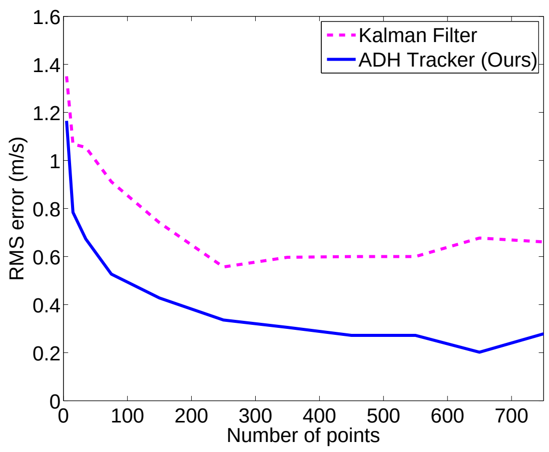

Fig. 14. RMS Error as a function of the number of points for each tracked object.

We can similarly compute the RMS error as a function of the distance to the tracked object, shown in Figure 15. Our method consistently outperforms the centroid-based Kalman filter at all tracked distances. Figures 14 and 15 can also be used to predict the RMS error for each tracked object. For example, Figure 15 shows that our RMS error is 0.15 m/s for nearby objects, and our error increases as the distance to the tracked object increases.

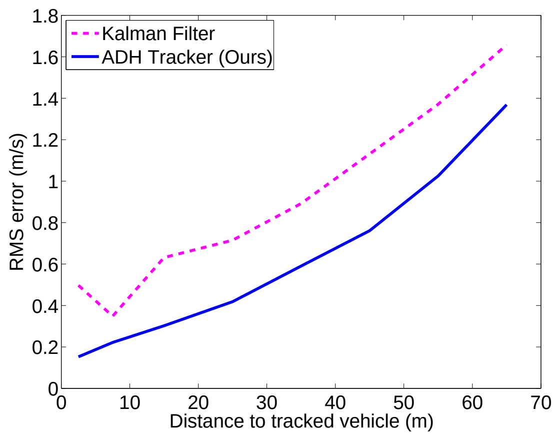

Fig. 15. RMS Error as a function of the distance to each tracked object.

Because of our motion model, we would expect the error of our method to decrease as the number of frames that we have seen an object increases, as we get a better estimate of the prior motion of the tracked object. This effect is shown in Figure 16. This figure indicates that our error is very large when we have only seen an object for 3 or fewer frames.

One explanation is that, before we have observed 4 frames, we do not yet have a good estimate of the tracked object's past motion. After observing an object for at least 4 frames, our motion model can be used to place a prior on the motion, thus reducing our error. Another possible explanation is that, when first observing an object, the object may initially be mostly occluded. After 4 frames, more of the object becomes visible, leading to better tracking.

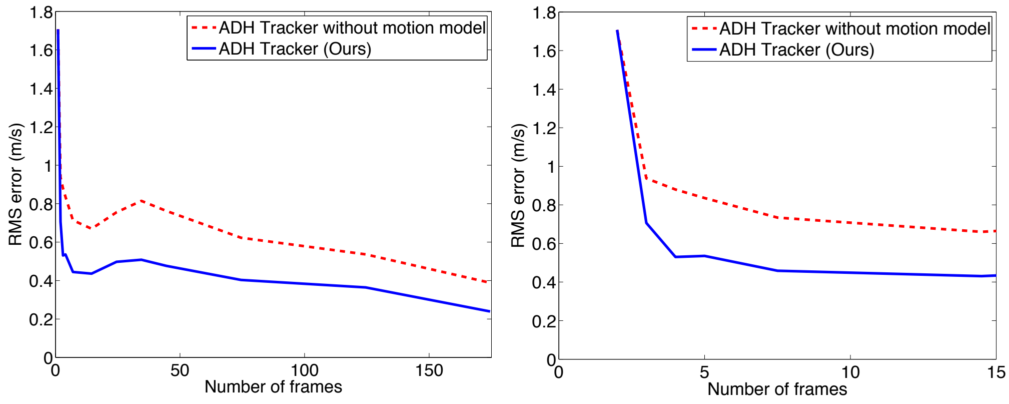

Fig. 16. Left: RMS Error as a function of the number of frames seen so far for each tracked object. We compare the results of our method with and without a motion model. Right: RMS Error as a function of the number of frames, viewed from 0 to 15 frames to more closely see the accuracy when initially tracking an object. Note that we can only begin estimating the object's velocity after seeing it for at least 2 frames.

We can disambiguate these effects by looking at the result of our method with and without the use of a motion model, as shown in Figure 16. When a motion model is not used, the only benefit of tracking an object for more frames is that more of the object will become visible. The difference between the two curves shows the benefit of using a motion model as the number of frames increases. As expected, for the first frame, the performance of the two methods is identical. After a few frames are observed, the tracker with a motion model significantly outperforms the version without a motion model.

In Figure 17 we can see that the velocity of the tracked object affects the tracking accuracy. Faster objects are more difficult to track accurately, due to the changing occlusions and viewpoints as the object moves. However, this figure shows that the ADH Tracker is more robust than ICP to tracking fast-moving objects and outperforms ICP for all linear velocities. For objects with a large angular velocity, both methods perform similarly, but the ADH tracker excels when tracking objects with a small angular velocity. Most objects moving in an urban scene have a small angular velocity for the majority of their motion, so increasing the tracking accuracy for objects with small angular velocity is important.

We claimed in Section 6.3 that the mean of the posterior distribution will minimize the RMS error. We can now verify that the RMS error decreases by 7.5% if we use the mean of the distribution instead of the mode. In a multimodal distribution, the mode will select the single point with the highest probability. However, if two modes have similar probabilities, the mode will often be far from the ground-truth alignment. The mean, on the other hand, will consider both modes and choose an estimate in between them. As explained in Section 6.3, using the mean will minimize the RMS tracking error.

We can understand the effect of different components of our system by looking at Figure 18. The full method combines 3D shape, color, and a motion model for an RMS error of 0.43 m/s (on average across 515 tracked vehicles). Removing color causes the error to increase by 14% to 0.49 m/s. Removing the motion model causes the error to increase by 44% to 0.62 m/s. Removing the 3D shape (by using a centroid-based Kalman filter) causes the error to increase by 81% to 0.78 m/s. Thus, using the full 3D shape and a motion model are crucial for accurate tracking.

We can also analyze the different sources of error of our evaluation method to determine the reliability our ground-truth data. First, by running SLAM on our dataset, we can show that our position estimate has an RMS error of 1.5 mm. Because our sensor has a framerate of 10 Hz, this equates to a velocity error of 0.015 m/s. Thus, our position error can account for less than 3.5% of the error of our method, which has an RMS error of 0.43 m/s when using color. The Velodyne Lidar,

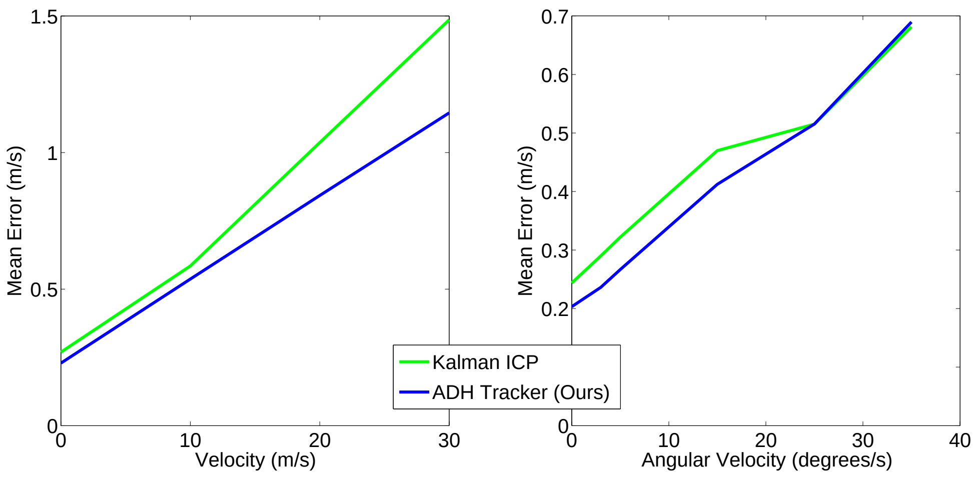

Fig. 17. Mean error as a function of the linear velocity (left) or angular velocity (right) of the tracked object.

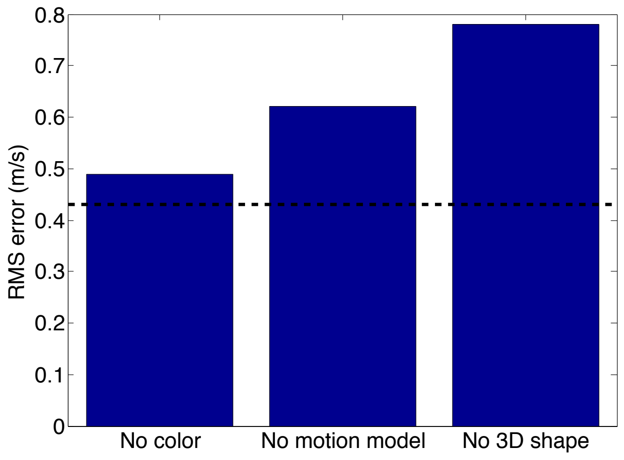

Fig. 18. RMS Error for different versions of the ADH tracker. The RMS error of the full ADH tracker is indicated by the black dashed line. Using the centroid instead of the full 3D shape would cause the biggest increase in error, shown by the bar on the right. Not using a motion model leads to the increase in error shown in the middle, and not using color leads to the increase in error shown on the left.

on the other hand, has reported errors of less than 2 cm, which could account for 47% of our total error (*Velodyne Lidar HDL-64E Datasheet*, 2010). However, because we have calibrated our Velodyne using the calibration method of Levinson and Thrun (2010), we expect the actual error of our Velodyne measurements to be much less.

By looking at the mean velocity error, we can further determine if there is a bias in our velocity estimates. When tracking using color, our mean velocity error is -0.03 m/s, which is 5.6% of our total error. A completely unbiased tracker would have a mean velocity error of 0, when averaged over an infinite number of tracked objects. Based on the error analysis above, we conclude that, if our method is biased, the bias is relatively small.

It is also interesting to compare the error of our method to the half-resolution of our sensor. The laser returns of our sensor become increasingly sparse as the distance to the tracked object increases. We define the resolution to be the horizontal spacing between sensor measurements at a given distance. We can see the half-resolution of our sensor in Figure 19, which we compare to the RMS alignment error. Our accuracy nearly matches the half-resolution of the sensor. By using a motion model, we can potentially improve our accuracy below the half-resolution for objects that maintain a constant velocity. However, for objects that change their velocity in unpredictable ways, the half-resolution represents a limit on the potential accuracy of our method when using a 3D Lidar for frame-to-frame tracking.

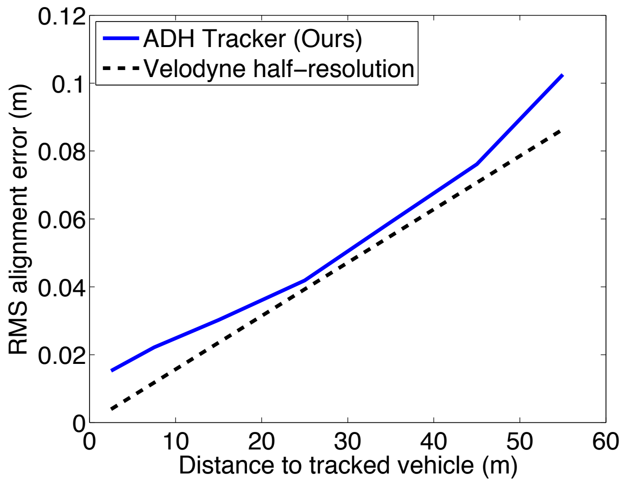

Fig. 19. RMS Error as a function of the distance to each tracked object, when tracking with color. We also show the half-resolution of our 3D sensor.

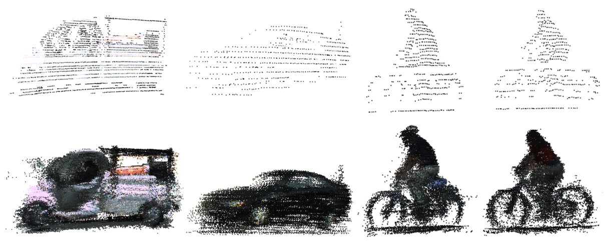

Fig. 20. Models obtained by tracking objects. The top row is the individual frame of the tracked object with the highest number of points. The bottom row is the model created by our tracker. Note that we are only performing frame-to-frame alignment when building these models.

8.3. Evaluation: Model Crispness

To evaluate the accuracy of our tracking estimates on a variety of moving objects, we build models of tracked objects by aligning the point clouds using our estimated velocity. These models can be visualized in Figures 1 and 20. For each model, we compute a crispness score (Sheehan et al., 2012) to evaluate how correctly the models were constructed; a crisp model corresponds to an accurately tracked object. As can be seen in Figure 21, if tracking is not accurate, the resulting model will be very noisy. A movie visualizing this model being constructed can be seen on our project page at `http://stanford.edu/~davheld/anytime_tracking.html`.

For each tracked object, we compute a crispness score as

$$\frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T}\frac{1}{n_i}\sum_{k=1}^{n_i}G(x_k-\hat{x}_k,2\Sigma)$$

where T is the number of frames for which the object is observed, n_i is the number of points in the ith frame, $\hat{x}_k$ is the point in frame j nearest to point x_k in frame i, G is a multi-variate Gaussian, and Σ controls the penalty for matches of different distances (Sheehan et al., 2012). Our crispness score has a minimum value of 0 and a maximum value of 1.

Using the crispness score, we evaluate our tracker on 135 people, 79 bikes, and 63 moving cars. For this evaluation, we use two test sets that were recorded 1 month earlier and 6 months later than the test set used in Section 8.2. In addition, the test set for the previous section was recorded around sundown, whereas the test sets in this section were recorded closer to noon. By testing during different seasons and times of day, we further demonstrate our robustness to changes in location, season, and lighting.

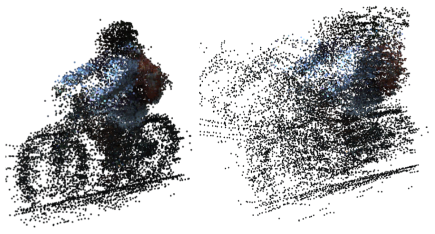

Fig. 21. A comparison of the models built with different tracking methods. Left: Our Method. Right: Kalman ICP.

Table 8.3 shows the crispness scores for tracking people, bikes, and moving cars. We evaluate our our method as well as two high performing baseline methods, selected based on performance in Section 8.2. When at least 100 points are visible, our method outperforms all other methods across all object classes. In Table 8.3, only frames with at least 200 points are used to compute the crispness score.

Tracking Method	Object Class		
	People	Bikes	Moving Cars
Kalman Filter	0.38	0.31	0.27
Kalman ICP	0.18	0.18	0.29
ADH (Ours)	**0.42**	**0.38**	**0.33**

Table 1. Crispness scores based on the 3D models built using the velocity estimates of different tracking methods. A crisper model generally correlates with greater tracking accuracy.

We see that, although the Kalman ICP method performs well for cars, it performs poorly for people and bikes. This is likely due to the shape of people and bikes. Because cars have a smooth, convex shape, ICP is able is more easily find the optimal alignment. On the other hand, people and bikes do not have well-defined faces, and bikes are highly non-convex, leading ICP to get stuck in a local optimum and resulting in a poor alignment. The centroid-based Kalman filter performs reasonably well on this metric, although it performed poorly in the metric from section 8.2. However, our method performs the best on this metric across all object classes as well as on the previous metric. This evaluation demonstrates that our method is robust to the class and shape of the object being tracked.

8.4. Qualitative Comparison

Figure 22 shows some examples where the ADH tracker outperforms the Kalman ICP baseline. Kalman ICP is initialized as described above, in which the centroid of each point cloud is placed through a Kalman filter to predict the initial alignment for ICP. Although this was the best method for initializing ICP among the methods that we tried in Section 8.2, the initialization is often still poor. Each of the examples shown in Figure 22 are cases in which ICP begins with an incorrect initialization. As a result, ICP converges to a poor local optimum. Furthermore, although the ICP baseline incorporates a motion model both to obtain a good initialization and for post-alignment smoothing, the ICP optimization does not incorporate a motion model into the alignment itself. As a result, the ICP alignment is often unrealistic according to the motion model. For instance, the second and third examples in Figure 22 show that ICP has rotated the point cloud in a manner that is unrealistic according to the motion model.

The ADH tracker deals with both of these issues. By performing a coarse-to-fine search, the ADH tracker avoids local minima. The ADH tracker inflates the measurement noise proportional to the sampling resolution in order to smooth out the objective function when sampling coarsely, which helps the model perform a global optimization. Second, each sample of the ADH tracker is evaluated using both the measurement model and the motion model, and regions with a high probability are sampled more finely. Thus our method incorporates the motion model when deciding where to sample, so unrealistic alignments such as those shown by ICP in Figure 22 are less likely to occur.

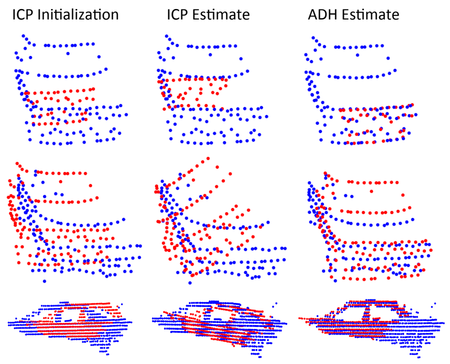

Fig. 22. Examples where the ADH tracker outperforms ICP initialized with a Kalman filter. Each row is a separate example of an attempted alignment between red points and blue points. Left column: Initial alignment for ICP. Middle column: Final alignment from ICP. Right column: Alignment using the ADH tracker. (Best viewed in color).

9. Conclusion

We have introduced a new technique called annealed dynamic histograms to robustly track moving objects in real time. We have demonstrated that combining information from 3D shape, color, and motion allows us to track objects much more accurately than using only one or two of these cues. We have also shown that grid-based methods can be made both fast and accurate by annealing the measurement model as we refine our distribution. This approach also allows us to globally explore the search space, avoiding the local minima of other approaches. For long-term autonomy in dynamic environments, objects must be tracked under a wide variety of lighting, viewpoint changes, and occlusions, so robust tracking is crucial for safe operation.

10. Acknowledgments

We would like to thank the Stanford Autonomous Driving Team and the Computational Vision and Geometry Lab, especially Alex Teichman and Dave Jackson for useful discussions and suggestions, and Brice Rebsamen for maintaining our research platform and code repository.

11. Funding

This work was supported by DARPA UPSIDE [grant number A13-0895-S002].

References

Azim, A. and Aycard, O. (2012), Detection, classification and tracking of moving objects in a 3d environment, *in* ‘Intelligent Vehicles Symposium (IV), 2012 IEEE’, IEEE, pp. 802–807.

Burgard, W., Derr, A., Fox, D. and Cremers, A. B. (1998), Integrating global position estimation and position tracking for mobile robots: the dynamic markov localization approach, *in* ‘Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on’, Vol. 2, IEEE, pp. 730–735.

Chassis Systems Control LRR3: 3rd generation Long-Range Radar Sensor (2009).

Darms, M., Rybski, P. and Urmson, C. (2008), Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments, *in* 'Intelligent Vehicles Symposium, 2008 IEEE', IEEE, pp. 1197–1202.

Estivill-Castro, V. and McKenzie, B. (2004), Hierarchical monte-carlo localization balances precision and speed, *in* 'Australasian Conference on Robotics and Automation'.

Feldman, A., Hybinette, M. and Balch, T. (2012), 'The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets', *Journal of Field Robotics* **29**(2), 258–276.

Held, D., Levinson, J. and Thrun, S. (2013), Precision tracking with sparse 3d and dense color 2d data, *in* 'Robotics and Automation (ICRA), 2013 IEEE International Conference on', IEEE, pp. 1138–1145.

Held, D., Levinson, J., Thrun, S. and Savarese, S. (2014), Combining 3d shape, color, and motion for robust anytime tracking, *in* 'Proceedings of Robotics: Science and Systems', Berkeley, USA.

Huang, J. and Mumford, D. (1999), Statistics of natural images and models, *in* 'Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.', Vol. 1, IEEE.

Kaestner, R., Maye, J., Pilat, Y. and Siegwart, R. (2012), Generative object detection and tracking in 3d range data, *in* 'Robotics and Automation (ICRA), 2012 IEEE International Conference on', IEEE, pp. 3075–3081.

Konidaris, G. and Barto, A. (2006), Autonomous shaping: Knowledge transfer in reinforcement learning, *in* 'Proceedings of the 23rd international conference on Machine learning', ACM, pp. 489–496.

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S. et al. (2008), 'A perception-driven autonomous urban vehicle', *Journal of Field Robotics* **25**(10), 727–774.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M. and Thrun, S. (2011), Towards fully autonomous driving: Systems and algorithms, *in* 'Intelligent Vehicles Symposium (IV), 2011 IEEE', pp. 163 –168.

Levinson, J. and Thrun, S. (2010), Unsupervised calibration for multi-beam lasers, *in* 'International Symposium on Experimental Robotics'.

Manz, M., Luettel, T., von Hundelshausen, F. and Wuensche, H.-J. (2011), Monocular model-based 3d vehicle tracking for autonomous vehicles in unstructured environment, *in* 'Robotics and Automation (ICRA), 2011 IEEE International Conference on', IEEE, pp. 2465–2471.

Marcello, R., Sorrenti, D. G. and Marchese, F. M. (2002), A robot localization method based on evidence accumulation and multi-resolution, *in* 'Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on', Vol. 1, IEEE, pp. 415–420.

Moosmann, F. and Stiller, C. (2013), Joint self-localization and tracking of generic objects in 3d range data, *in* 'ICRA', pp. 1146–1152.

Odom, D. and Milanfar, P. (2006), Modeling multiscale differential pixel statistics, *in* 'Electronic Imaging 2006', International Society for Optics and Photonics, pp. 606504–606504.

Olson, C. F. (2000), 'Probabilistic self-localization for mobile robots', *Robotics and Automation, IEEE Transactions on* **16**(1), 55–66.

Olson, E. B. (2009), Real-time correlative scan matching, *in* 'Robotics and Automation, 2009. ICRA'09. IEEE International Conference on', IEEE, pp. 4387–4393.

Petrovskaya, A. and Thrun, S. (2008), 'Model based vehicle tracking for autonomous driving in urban environments', *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland* **34**.

Randlov, J. and Alstrom, P. (1998), Learning to drive a bicycle using reinforcement learning and shaping, *in* 'Proceedings of the Fifteenth International Conference on Machine Learning', pp. 463–471.

Rose, K. (1998), 'Deterministic annealing for clustering, compression, classification, regression, and related optimization problems', *Proceedings of the IEEE* **86**(11), 2210–2239.

Rusu, R. B. and Cousins, S. (2011), 3D is here: Point Cloud Library (PCL), *in* 'IEEE International Conference on Robotics and Automation (ICRA)', Shanghai, China.

Ryde, J. and Hu, H. (2010), '3d mapping with multi-resolution occupied voxel lists', *Autonomous Robots* **28**(2), 169–185.

Sheehan, M., Harrison, A. and Newman, P. (2012), ‘Self-calibration for a 3d laser’, *The International Journal of Robotics Research* **31**(5), 675–687.

Simmons, R. and Koenig, S. (1995), Probabilistic robot navigation in partially observable environments, *in* ‘IJCAI’, Vol. 95, pp. 1080–1087.

Streller, D., Furstenberg, K. and Dietmayer, K. (2002), Vehicle and object models for robust tracking in traffic scenes using laser range images, *in* ‘Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on’, IEEE, pp. 118–123.

Sun, J., Xu, Z. and Shum, H.-Y. (2008), Image super-resolution using gradient profile prior, *in* ‘Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on’, IEEE, pp. 1–8.

Teichman, A., Levinson, J. and Thrun, S. (2011), Towards 3d object recognition via classification of arbitrary object tracks, *in* ‘Robotics and Automation (ICRA), 2011 IEEE International Conference on’, IEEE, pp. 4034–4041.

Thrun, S., Burgard, W., Fox, D. et al. (2005), *Probabilistic robotics*, Vol. 1, MIT press Cambridge.

Thrun, S., Fox, D., Burgard, W. and Dellaert, F. (2001), ‘Robust monte carlo localization for mobile robots’, *Artificial intelligence* **128**(1), 99–141.

Velodyne Lidar HDL-64E Datasheet (2010).

Wojke, N. and Haselich, M. (2012), Moving vehicle detection and tracking in unstructured environments, *in* ‘Robotics and Automation (ICRA), 2012 IEEE International Conference on’, IEEE, pp. 3082–3087.

Zitnick, C. L. (2012), Seeing through the blur, *in* ‘Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)’, IEEE Computer Society, pp. 1736–1743.