Combining 3D Shape, Color, and Motion for Robust Anytime Tracking

David Held, Jesse Levinson, Sebastian Thrun, Silvio Savarese
Computer Science Department, Stanford University
{davheld, jessel, thrun, ssilvio}@cs.stanford.edu

***Abstract*—Although object tracking has been studied for decades, 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 0.7 milliseconds, our method outperforms all of the baseline methods of similar speed by at least 10%. 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.**

I. 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 cues from shape, color, and motion. As we will show, adding color and motion information gives a large benefit to our system compared to using the 3D shape alone. This information is especially useful for distant objects or objects under heavy occlusions, when detailed 3D shape information may not be available.

We make use of a grid-based method to sample velocities from the state space. Traditional grid-based approaches 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.

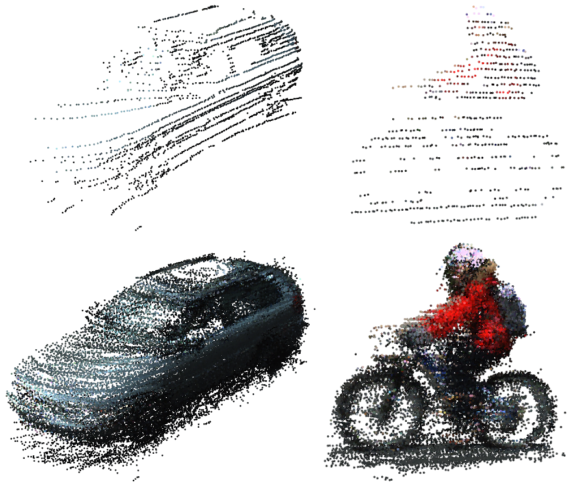

Fig. 1. Our method tracks moving objects very accurately, as seen by these models generated from successive frame-to-frame alignments from our tracker. The top row shows the individual frame of the tracked object with the largest number of points. The bottom row shows the model created by our tracker.

In this paper we present a number of novel contributions. First, we introduce a new technique 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 state-of-the-art approaches. 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. For additional details on this project, please see our project page at

http://stanford.edu/~davheld/anytime_tracking.html.

II. 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 [11, 8] or by the center of a bounding box [10, 1, 25] 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 [18]. 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 [18] relies on detecting the corner of the car in order to position the rectangular model, whereas [30, 3] 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 [5, 14]. 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 in [17] and [6].

Grid-based methods are more common in SLAM systems [24] 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. A related, though much slower and less accurate, method was used in Held et al. [6]. This general approach to tracking is also related to the methods used in various multi-resolution grid-based SLAM systems [2, 16, 17, 13, 22, 4].

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 [20]. Related methods known as "shaping" have been used in reinforcement learning, starting with an easier task and progressing to increasingly difficult tasks [19, 9]. This class of methods is also known as "graduated optimization" and has been applied to the related problem of image alignment [31]. 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 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.

III. 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 [27]. 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 IV. We then subdivide some of the grid cells, as described in Section V, 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.

IV. Probabilistic Model

Below we describe the probabilistic model that we use for tracking. In Section V we will describe how we use this model as part of our annealed dynamic histogram framework. Finally, in Section VI we add color-matching probabilities to our model. However, the model that we describe here can also be used if no color information is available.

The Dynamic Bayesian Network upon which our model is built is shown in Figure 2. The position and velocity of the tracked object are represented by the state variable x_t. We assume that the rotational velocity of the tracked object is small (relative to the frame rate of the sensor), which is often the case for people, bikes, and cars moving in urban settings. After obtaining the posterior over translation, we can optionally search over rotation, as described in Section V-D.

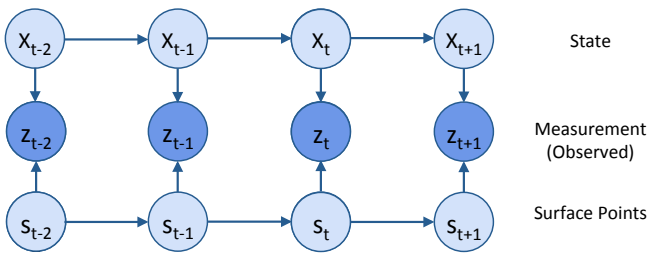

Fig. 2. Dynamic Bayesian Network representing our model for a tracked object.

We wish to use the 3D shape of the object being tracked in a Bayesian probabilistic framework, which allows us to combine the 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. From these points, the actual sensor measurements z_t are drawn and observed. 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. This process is illustrated in Figure 3. The measurement model for our tracker will be described in Section IV-A.

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 increases tracking performance. To build the motion model, we take all of the values for $p(x_t \mid z_1 \dots z_t)$ from the previous frame and fit a multivariate Gaussian to the set of probabilities. Once a Gaussian approximation to the posterior is obtained, the result is used in the standard constant velocity model of a Kalman filter [29].

A. Measurement Model Derivation

We now derive the measurement model using the Dynamic Bayes Net from Figure 2. We can first write our measurement model using the joint distribution as

$$p(z_t \mid x_t, z_{t-1}) = \iint p(z_t, s_t, s_{t-1} \mid x_t, z_{t-1}) \, \mathrm{d}s_{t-1} \, \mathrm{d}s_t$$

Using the independence assumptions from the model in Figure 2, we can expand the term inside the integral as

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

where η is a normalization constant. Note that we have used the conditional independence assumption $p(s_{t-1} \mid x_t, z_{t-1}) = p(s_{t-1} \mid z_{t-1})$, which is not encoded in the graphical model from Figure 2. This independence assumption holds because at each time step t we choose the centroid of z_{t-1} to be the origin of the coordinate system for our state variable. The position component of the state thus measures how far the tracked object has moved since the previous observation.

To compute the terms in equation 1, we first consider that, at each time frame t, a new set of points z_t is sampled independently from the surface of the object, s_t, as shown in Figure 3. This process is modeled by a Gaussian, with covariance given by the sensor noise Σ_e.

We also 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. This formulation enables our method to be robust to changes in viewpoint and occlusions. If $p(V)$ represents the prior probability of sampling from a previously visible surface, then we can write:

$$\begin{aligned} p(s_{t,j} \mid s_{t-1}) = &p(V) \, p(s_{t,j} \mid s_{t-1}, V) \; + \\ &p(\neg V) \, p(s_{t,j} \mid s_{t-1}, \neg V) \end{aligned} \quad (2)$$

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.

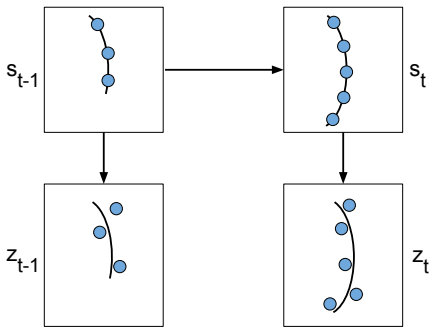

Fig. 3. Illustration of the sampling of surface points s_t and measurement points z_t. Because of sensor noise, the measurements z_t will not lie exactly on the surface. Further, because of occlusions, changes in viewpoint, deformations, and random sampling, the visible surface points change from s_{t-1} to s_t

The term $p(s_{t,j} \mid s_{t-1}, \neg V)$ represents the probability that $s_{t,j}$ was sampled 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 this 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

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

for some constants k_1 and k_2. We can now simplify equation 2 as

$$p(s_{t,j} \mid s_{t-1}) = \eta \left(p(s_{t,j} \mid s_{t-1}, V) + k_3\right)$$

where η is a normalization constant and k_3 acts as a smoothing factor.

Because we model each term in equation 1 as either a Gaussian or a Gaussian plus a constant, we can view this expression as a series of convolutions. The convolution of two Gaussians is simply another Gaussian, so our measurement model simplifies down to

$$\begin{aligned} &p(z_t \mid x_t, z_{t-1}) \\ &= \eta \left(\prod_{z_j \in z_t} \exp\left(-\frac{1}{2} (z_j - \bar{z}_i)^T \Sigma^{-1} (z_j - \bar{z}_i) \right) + k \right) \end{aligned} \quad (3)$$

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 3.

In practice, the measurement model is quickly computed by using a kd-tree to look up the nearest-neighbor for each point. We also divide the space into a grid and cache the log probability for each grid cell. Thus, for each grid-cell we only perform a single kd-tree lookup, and 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 V).

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 thus only models motion along the ground surface. This change results in a significant speedup of our method, with minimal effect on the accuracy. For environments in which this assumption is invalid, the full three-dimensional tracker may be used, or one may incorporate an elevation map.

V. Annealed Dynamic Histograms

Using the techniques described above, the measurement model and motion model probabilities are relatively quick to compute. However, we must compute these probabilities 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.

A. Derivation

Our overall approach to dynamic histograms can be visualized in Figure 4. We start by coarsely dividing the state space into grid cells and computing the probability $p(x_t|z_1 \dots z_t)$ 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.

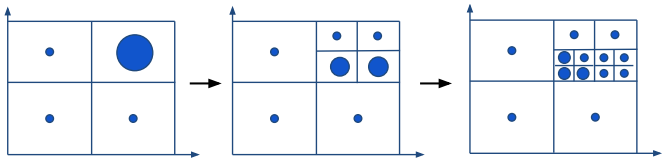

Fig. 4. We decompose the state space using annealed dynamic histograms. Here we show the dynamic decomposition, starting from a coarse sampling on the left and refining the distribution over time.

The first step is to divide the state space into a coarse grid. We next subdivide some of the cells 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 the 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 a 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, more coarse approximation to the posterior and the new, improved approximation to the posterior. The size of the KL-divergence gives a measure of how much we can improve our approximation to the posterior by dividing this 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 5.

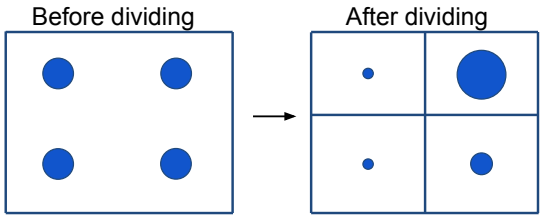

Fig. 5. Before dividing a cell, we can view it as having sub-cells whose probabilities are all constrained to be equal. After dividing, each of the sub-cells can take on its own probability.

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

$$D_{KL}(A||B) = P_i \ln k \tag{4}$$

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.

B. Annealing

Initially, when we sample the state space at a very coarse resolution, we do not expect to find a good alignment between the previous frame and the current frame for the tracked object. The sampling histogram (shown in Figure 4) 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 4, Σ_g decreases to 0. The measurement model is thus naturally 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.

C. 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 VIII-B, we return the mean of the posterior. On the other hand, to build accurate object models, as in Section VIII-C, 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 can make use of the entire tracked probability histogram, in the form of a density tree [28].

D. 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. We incorporate a search over rotation when we evaluate our tracker by building 3D object models in Section VIII-C.

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 3;
    Compute $p(\hat{x}_{t,i} \mid z_1 \ldots z_{t-1})$ from motion model;
    $\tilde{p}(\hat{x}_{t,i} \mid z_1 \ldots z_t) \leftarrow p(z_t \mid \hat{x}_{t,i}, z_{t-1})\, p(\hat{x}_{t,i} \mid z_1 \ldots z_{t-1})$; // Unnormalized probability
  end
  /* Normalize probabilities */
  $\eta \leftarrow \sum_{c_i \in H_{new}} \tilde{p}(\hat{x}_{t,i} \mid z_1 \ldots z_t)$;
  for *each cell* $c_i \in H_{new}$ **do**
    $p(c_i) \leftarrow \eta\, \tilde{p}(\hat{x}_{t,i} \mid z_1 \ldots z_t)\, p(R)$
  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

VI. Adding Color

A. Learning color models

Our laser-based motion tracking algorithm naturally lends itself to augmentation with simultaneous data from a traditional 2D video camera. To leverage color in our probabilistic model, 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. We then build a normalized histogram of the differences in color values between each point and its closest neighbor from the next spin. The difference histograms we obtain closely follow a Laplacian distribution, as expected [7, 26, 15].

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.

B. Tracking with color

We can now incorporate the chosen color distribution into the measurement model from Section IV-A. For any potential correspondence, there is some probability $p(C)$ that the colors match, and some probability $p(\neg C)$ that the colors do not match due to changes in lighting, lens flare, etc. We can then model the probability of this correspondence as a product of spatial and color probabilities:

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

where the spatial probability is computed as a Gaussian as before. We can model the color probability as

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

where $p(s_{t,j} \mid s_{t-1}, V, C)$ is the learned color model distribution (parameterized as a Laplacian), and $p(s_{t,j} \mid s_{t-1}, V, \neg C) = 1/255$ is the prior probability of a point having any given color. We can then use equation 2 to compute the measurement model probability.

When we are coarsely sampling the state space (see Section V), we do not initially expect the colors to match very well. Therefore, we set $p(C)$ to be a function of the sampling resolution r, as

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

where σ_c is a parameter that controls the rate at which $p(C)$ decreases with increasing resolution. As we sample more finely, $p(C)$ increases until $p(C) = p_c$ when $r = 0$, modeling that we expect colors to match more often at a finer sampling resolution.

VII. 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 100,000 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 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.

VIII. 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 [12] and [14], in which a small number of cars are equipped with a measuring apparatus. Instead, we propose two methods to automatically evaluate our tracking velocity estimates on a large number of tracked objects. First, using the evaluation method of Held et al. [6], 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. Second, we build models of tracked objects by aligning the point clouds based on our estimated velocity. We then compute a crispness score, as in Sheehan et al. [23], 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. Thus, 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.

A. 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 drive 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 is g: $k = 0.8$, $\Sigma'_e = 2\Sigma_e = (0.03\,\mathrm{m})^2\mathrm{I}$, Σ_r = (r/2)I, and $\Sigma_g = g\mathrm{I}$, 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 r, 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 probability 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 2000 points.

B. Evaluation: Relative Reference Frame

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 [6]. 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 VIII-C further quantitative results when tracking moving objects of different classes (cars, bikes, and pedestrians), to demonstrate that our method generalizes across object types.

We ignore tracks that contain major undersegmentation issues, in which the tracked object is segmented together with another object. 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.

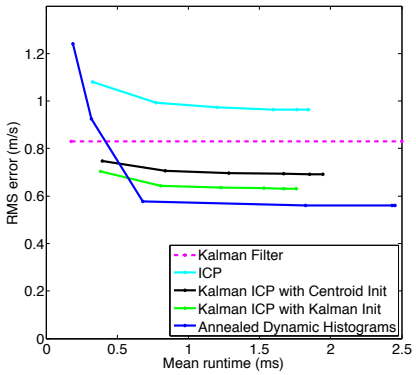

Fig. 6. RMS error vs runtime of our method compared to several baseline methods. Note that the method "Kalman ICP with Kalman Init" is a novel baseline method that we present in this paper.

To measure robustness, we compute the RMS tracking error for each method, and the results are shown in Figure 6. First, we compare to 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. [11] and Kaestner et al. [8]. As can be seen in Figure 6, this method is extremely fast, completing in 0.2 milliseconds. The method is also reasonable in accuracy, producing an RMS error of 0.83 m/s across the 515 cars in our test set.

Next, we compare to a number of variants of ICP, the iterative closest point method. First, we compare to the basic point-to-point ICP algorithm, using the implementation from PCL [21]. This basic method is used for tracking by Feldman et al. [5]. 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 6. It is clear from Figure 6 that the basic ICP algorithm does not perform well for tracking, with an RMS error 16% 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 in Moosmann and Stiller [14]. Last, we try first running the centroid through a Kalman filter and using the output as the initialization for ICP.

Figure 6 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.

The method of using a Kalman filter with ICP, initialized by aligning the centroids, performs reasonably, with an improvement of 17% 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 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 also 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. [6], 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. Their method also takes an average of 86 milliseconds per frame and hence is probably not appropriate for a real-time system.

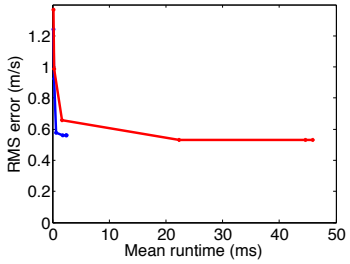

Fig. 7. Accuracy vs mean runtime using annealed dynamic histograms (blue) vs densely sampling the state space (red).

Our method achieves an accuracy of 0.58 m/s after running for just 0.7 milliseconds. Even at this minimum runtime, our method does 31% better than a simple Kalman filter and 10% better than all of the baseline methods of similar speed, as shown in Figure 6. If our method is allowed to run for 1.8 milliseconds, our method achieves an accuracy of 0.56 m/s, which is 11% 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 70 m away, our method achieves an RMS error of 1.8 m/s.

If color images are available, then we can incorporate color into our measurement model, as described in Section VI. With

Fig. 8. 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.

Fig. 9. A comparison of the models built with different tracking methods. Left: Our Method. Right: Kalman ICP.

the addition of color, our RMS error decreases by an additional 7%, thus achieving an RMS error of 0.52 m/s. Note that many frames in our test set contained heavy shadows or lens flare. We thus except color to make an even bigger difference when lens flare and similar exposure problems are avoided.

One of the novel additions of our method is the use of annealed dynamic histograms to speed up our tracker. In Figure 7, we show the decrease in speed if we were to densely sample the search space. Densely sampling is about 12 to 33 times slower for approximately the same level of accuracy.

C. Evaluation: Model Crispness

To evaluate the accuracy of our tracking on a variety of moving objects, we build models of tracked objects by aligning the point clouds based on our estimated velocity. These models can be visualized in Figures 1 and 8. For each model, we compute a crispness score [23] to evaluate how correctly the models were constructed; an accurate model corresponds to an accurately tracked object. As can be seen in Figure 9, if the tracking is not accurate, the resulting model will be very noisy.

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 x_k in frame i, G is a multi-variate Gaussian, and Σ controls the penalty for matches of different distances. 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 VIII-B. 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.

Table I 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 VIII-B. When at least 100 points are visible, our method outperforms all other methods across all object classes. In Table I, only frames with at least 200 points are used to compute the crispness score. The ICP-Kalman method performs poorly on people and bikes, presumably due to local minima. The centroid-based Kalman filter performs reasonably well, but our method performs the best across all object classes. This demonstrates that our method is robust to the class and shape of the object being tracked.

TABLE I
CRISPNESS SCORES

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	**0.42**	**0.38**	**0.33**

IX. 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.

X. 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. We also acknowledge the support of a DARPA UPSIDE grant A13-0895-S002.

References

[1] Asma Azim and Olivier Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 802–807. IEEE, 2012.

[2] Wolfram Burgard, Andreas Derr, Dieter Fox, and Armin B Cremers. 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*, volume 2, pages 730–735. IEEE, 1998.

[3] Michael Darms, Paul Rybski, and Chris Urmson. Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1197–1202. IEEE, 2008.

[4] Vladimir Estivill-Castro and Blair McKenzie. Hierarchical monte-carlo localization balances precision and speed. In *Australasian Conference on Robotics and Automation*, 2004.

[5] Adam Feldman, Maria Hybinette, and Tucker Balch. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *Journal of Field Robotics*, 29(2):258–276, 2012.

[6] David Held, Jesse Levinson, and Sebastian Thrun. Precision tracking with sparse 3d and dense color 2d data. In *International Conference on Robotics and Automation (ICRA)*, 2013.

[7] Jinggang Huang and David Mumford. Statistics of natural images and models. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

[8] Ralf Kaestner, Jérôme Maye, Yves Pilat, and Roland Siegwart. Generative object detection and tracking in 3d range data. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3075–3081. IEEE, 2012.

[9] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496. ACM, 2006.

[10] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J.Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163 –168, june 2011. doi: 10.1109/IVS.2011.5940562.

[12] Michael Manz, Thorsten Luettel, Felix von Hundelshausen, and H-J Wuensche. Monocular model-based 3d vehicle tracking for autonomous vehicles in unstructured environment. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2465–2471. IEEE, 2011.

[13] R Marcello, Domenico G Sorrenti, and Fabio M Marchese. A robot localization method based on evidence accumulation and multi-resolution. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 415–420. IEEE, 2002.

[14] Frank Moosmann and Christoph Stiller. Joint self-localization and tracking of generic objects in 3d range data. In *ICRA*, pages 1146–1152, 2013.

[15] David Odom and Peyman Milanfar. Modeling multiscale differential pixel statistics. In *Electronic Imaging 2006*, pages 606504–606504. International Society for Optics and Photonics, 2006.

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

[17] Edwin B Olson. Real-time correlative scan matching. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4387–4393. IEEE, 2009.

[18] Anna Petrovskaya and Sebastian Thrun. Model based vehicle tracking for autonomous driving in urban environments. *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, 34, 2008.

[19] Jette Randlov and Preben Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, 1998.

[20] Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11): 2210–2239, 1998.

[21] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[22] Julian Ryde and Huosheng Hu. 3d mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28 (2):169–185, 2010.

[23] Mark Sheehan, Alastair Harrison, and Paul Newman. Self-calibration for a 3d laser. *The International Journal of Robotics Research*, 31(5):675–687, 2012.

[24] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *IJCAI*, volume 95, pages 1080–1087, 1995.

[25] Daniel Streller, K Furstenberg, and Klaus Dietmayer. 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*, pages 118–123. IEEE, 2002.

[26] Jian Sun, Zongben Xu, and Heung-Yeung Shum. Image super-resolution using gradient profile prior. In *Computer*

Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008.

[27] Alex Teichman, Jesse Levinson, and Sebastian Thrun. Towards 3d object recognition via classification of arbitrary object tracks. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4034–4041. IEEE, 2011.

[28] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.

[29] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2005.

[30] Nicolai Wojke and M Haselich. Moving vehicle detection and tracking in unstructured environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3082–3087. IEEE, 2012.

[31] C Lawrence Zitnick. Seeing through the blur. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1736–1743. IEEE Computer Society, 2012.