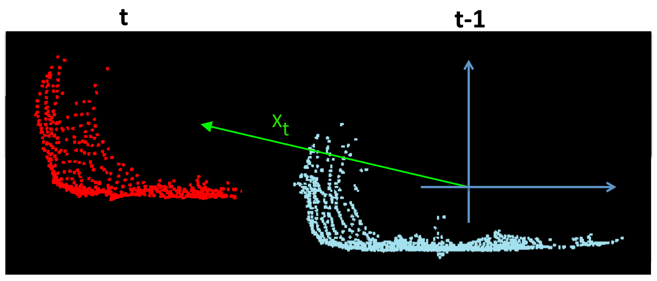
Combining 3D Shape, Color, and Motion for Robust Anytime Tracking Supplementary Material

I. Probabilistic Model

A. State Space

Below we describe, in more detail, the probabilistic model that we use for tracking. 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 Supplementary Figure 1. The position state variable thus measures how far this object has moved since the previous observation.



Supplementary Figure 1. 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.

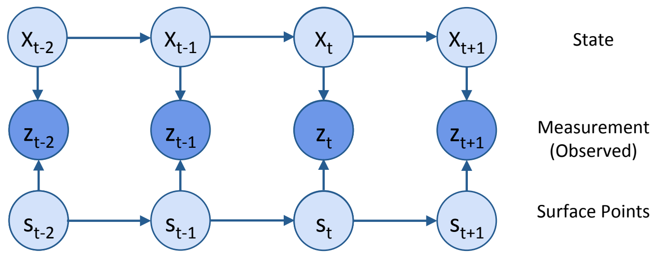
We assume that the rotational velocity of the tracked object is small relative to the frame rate of the sensor. This is often the case for people, bikes, and cars moving in urban settings. Thus, the rotational velocity is not included in the state. If one is interested in estimating the rotational velocity,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

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 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. Another possibility is to divide the state space into a 2D projection onto the ground and a 1D projection onto the vertical axis. One can then track objects separately in each of these projected spaces, as in Held et al. [3]. In our experiments, the resulting method is very fast but loses some accuracy due to the projection.

B. Dynamic Bayesian Network

The Dynamic Bayesian Network upon which our model is built is shown in Supplementary Figure 2. In our model, we have added 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.



$$\begin{tikzcd}
X_{t-2} \arrow[r] \arrow[d] & X_{t-1} \arrow[r] \arrow[d] & X_t \arrow[r] \arrow[d] & X_{t+1} \arrow[d] \\
Z_{t-2} & Z_{t-1} & Z_t & Z_{t+1} \\
S_{t-2} \arrow[r] \arrow[u] & S_{t-1} \arrow[r] \arrow[u] & S_t \arrow[r] \arrow[u] & S_{t+1} \arrow[u]
\end{tikzcd}$$

Supplementary Figure 2. 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 [6] include a geometry variable in which they model objects as 2D rectangles and estimate their width and length. SLAM systems use a similar variable to represent the environment map [8]. 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 $p(s_{t,i})$ 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, $\{z_{t,1} \dots z_{t,n}\} \in z_t$. 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 $s_{t,i}$, 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 Supplementary Figure 1, 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 Supplementary Figure 2, 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 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

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

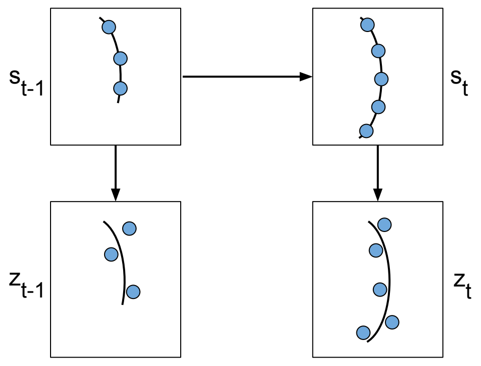
for some constants k_1 and k_2. We can now simplify equation 4 as

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

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 Supplementary Figure 3.



Supplementary Figure 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. Furthermore, because of occlusions, changes in viewpoint, deformations, and random sampling, the visible surface points change from s_{t-1} to s_t.

II. Tracking

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

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

where η is a normalization constant. The first term is our measurement model. In the standard Bayes filter algorithm [8], one would use conditional independence assumptions to simplify this as

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

However, in our case the latent variables $s_1, \dots s_t$ 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:

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

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 in equation 6 is obtained from our motion model, which will be described in Section II-C.

A. *Measurement Model Derivation*

We now derive the measurement model, using the Dynamic Bayes Net from Supplementary Figure 2. 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 2. 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 Supplementary Figure 2 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,

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

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

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

We have used the fact that the convolution of two Gaussians is another Gaussian, following the standard derivation as in Thrun et al. [8]. We also have from before that

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

We can now evaluate the integral of equation 8 to get

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

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

To compute the measurement model in practice, let $\bar{z}_{t-1} = z_{t-1} + x_{p,t}$; 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

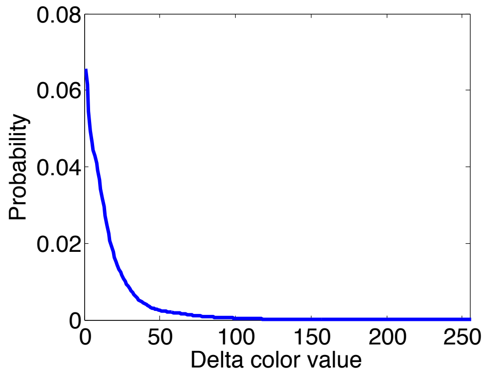
$$\begin{aligned} & p(z_t \mid x_t, z_{t-1}) \\ & \quad = \eta \left(\prod_{z_j \in z_t} \exp\left(-\frac{1}{2} (z_j - \bar{z}_i)^T \Sigma^{-1} (z_j - \bar{z}_i) \right) + k \right) \end{aligned} \tag{11}$$

where Σ is given by $\Sigma = 2\Sigma_e + \Sigma_r$.

B. *Tracking with color*

We will now describe in more detail how to compute the measurement model $p(z_t \mid x_t, z_{t-1})$ when incorporating color information. 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. 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 Supplementary Figure 4. The distribution closely follows a Laplacian distribution, as expected [4, 7, 5].



Supplementary Figure 4. 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 chosen color distribution into the measurement model derived in Section II-A. 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

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

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 many other unmodeled causes, the observed color can change drastically between two frames. We thus propose that there is some probability $p(\neg C)$ that all of the the observed colors will change in a way that is not modeled by the learned color distribution from Supplementary Figure 4. We can then write the color match probability from equation 12 as

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

where $p_c(s_{t,j} \mid s_{t-1}, V, C)$ is the learned color model distribution from Supplementary Figure 4 (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 a given color, given that the color does not 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, we do not expect the colors to match very well. Therefore, we set $p(C)$ to be a function of the sampling resolution, as

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

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 such 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 $\bar{z}_{t-1} = z_{t-1} + x_{p,t}$; 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 then 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 Supplementary Figure 4 (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.

C. Motion Model

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 multivariate Gaussian to the set of probabilities. We compute the mean μ_t and covariance Σ_t by by weighting each state by its probability, as

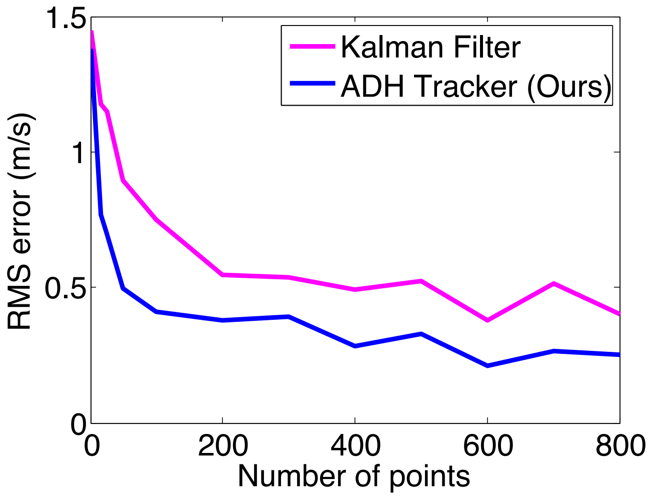
$$\begin{aligned} \mu_t &= \sum_i p(x_{t,i} \mid z_1 \ldots z_t)\, x_{t,i} \\ \Sigma_t &= \sum_i p(x_{t,i} \mid z_1 \ldots z_t)(x_{t,i} - \mu_t)^T (x_{t,i} - \mu_t) \end{aligned}$$

where $x_{t,i}$ is the state vector. 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 [8].

III. Results

Here we provide some additional results and analysis of our method. First, 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 23.5%.

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 Supplementary Figure 5. 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 the 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.

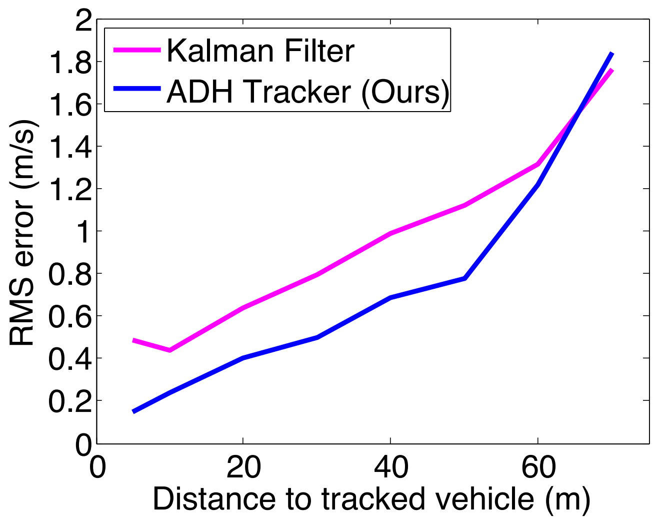


Supplementary Figure 5. RMS Error as a function of the number of points for each tracked object.

We can similarly compute the RMS error as a function of the distance to the tracked object, shown in Supplementary Figure 6. As before, for distant objects when the observed point cloud is very sparse, our method performs similarly to the centroid-based Kalman filter. As the distance to the tracked object decreases, the observed point cloud becomes more dense and our RMS error decreases relative to the Kalman filter.

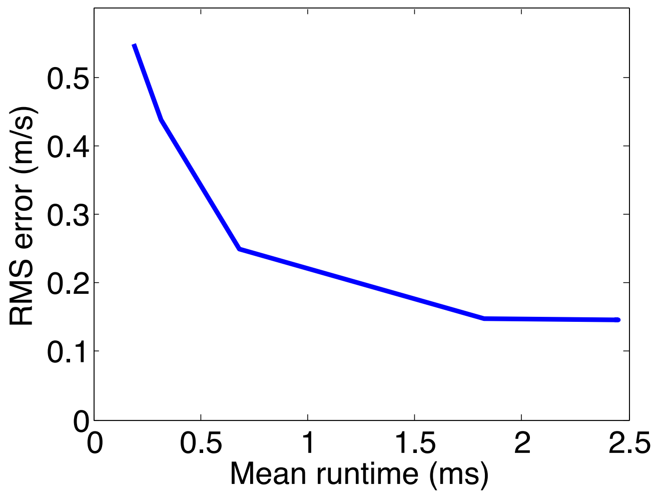
These two charts also can be used to predict the RMS error for tracked objects at different distances. For example, Supplementary Figure 6 shows that our RMS error is 0.15 m/s for close objects, and our error increases as the distance to the tracked object increases and as the number of observed points decreases.

Although it may appear from Figure 6 in our paper that our accuracy saturates after 1 ms, the results in this figure are averaged over 515 tracked vehicles, and distant cars are too sparse to benefit much from sampling our state space at a higher resolution. However, we do see a benefit from increased sampling when tracking nearby cars. For example, Supplementary Figure 7 shows the runtime vs accuracy for objects that are within 5 m from the ego vehicle. It is clear from this figure that the accuracy of our method continues to improve as the method is run for longer. A user can thus tune the desired runtime or tracking accuracy for each object based



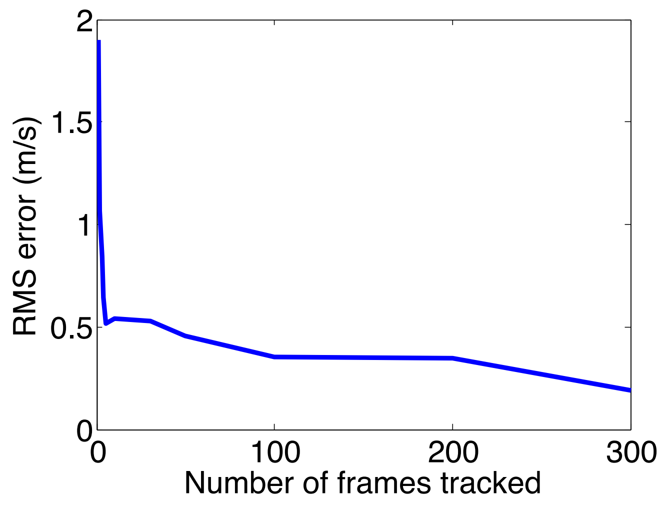
Supplementary Figure 6. RMS Error as a function of the distance to each tracked object.

on the needs of the system.



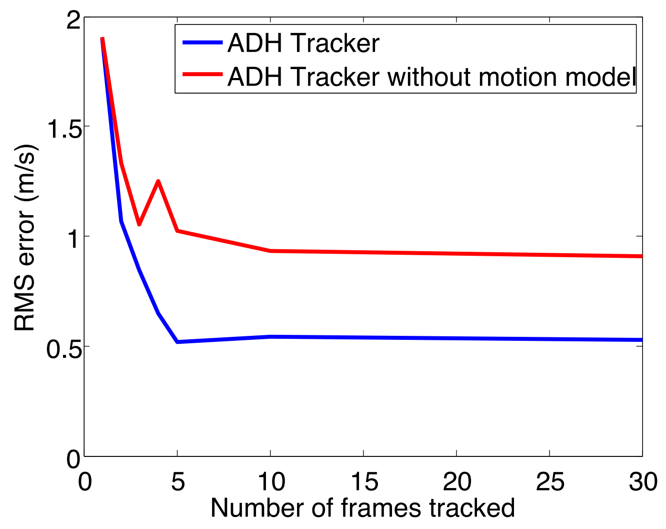
Supplementary Figure 7. RMS Error vs runtime for objects that are within 5 m of the ego vehicle. Note that when the method is allowed to have a longer runtime, the accuracy continues to decrease.

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 Supplementary Figure 8. This figure indicates that our error is very large when we have only seen an object for 5 or less frames. One reason for this is that, before we have observed 5 frames, we do not yet have a good estimate of the tracked object's past motion. However, after observing an object for more than 5 frames, our motion model can be used to place a prior on the motion, thus reducing our error. Additionally, when first observing an object, the object may initially be mostly occluded. After 5 frames, the more of the object may be visible, leading to better tracking.



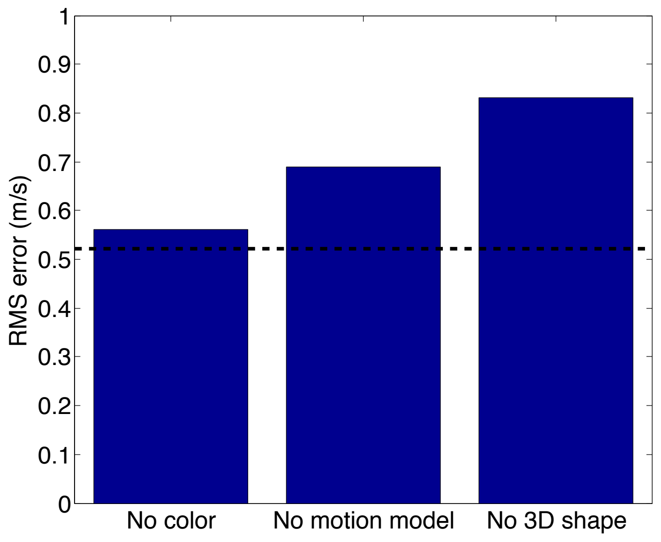
Supplementary Figure 8. RMS Error as a function of the number of frames seen so far for each tracked object.

We can disambiguate these effects by looking at Supplementary Figure 9, in which we show the result of our method with and without the use of a motion model. When a motion model is not used, the only benefit of tracking an object for more frames is that more of the object will be visible than when the object is first observed. 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.



Supplementary Figure 9. RMS Error as a function of the number of frames seen so far for each tracked object. We compare using our method both with and without a motion model. Here we cut off the graph at a maximum of 30 frames in order to more closely see the effect when only a small number of frames is visible.

We can understand the effect of different components of our system by looking at Supplementary Figure 10. The full method is shown as a black line on the figure, combining 3D shape, color, and a motion model for an RMS error of 0.52 m/s (on average across 515 tracked vehicles). Removing color causes the error to increase by 7.6% to 0.56 m/s. Removing the motion model causes the error to increase by 32.2% to 0.69 m/s. Removing the 3D shape (by using a centroid-based Kalman filter) causes the error to increase by 59.5% to 0.83 m/s. Thus, using the full 3D shape for tracking is crucial for accurate tracking.



Supplementary Figure 10. RMS Error as a function of the distance to each tracked object.

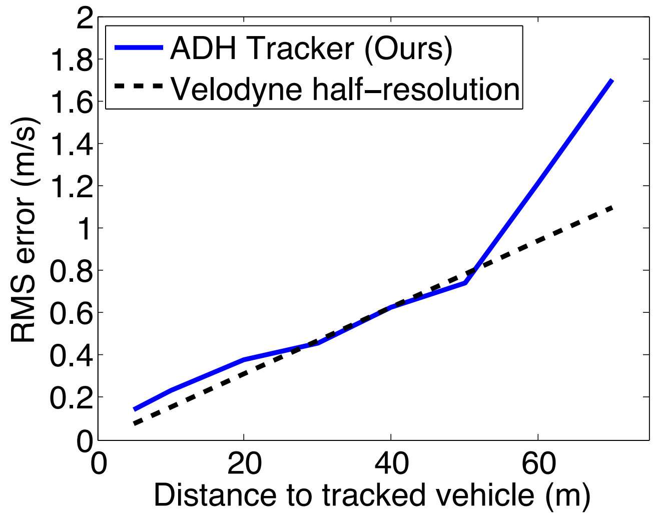
It is interesting to compare the performance of our method to that of a radar, which can also be used to measure velocity of moving objects. The Bosch LLR3 Radar can estimate velocities to within 0.12 m/s [1]. However, radar only estimates velocity in a single dimension: in the direction from the radar to the tracked object. This one-dimensional estimate is not useful for robotics or autonomous driving, in which we need a 2D estimate of the velocity of each moving object to estimate where each object is moving. Our method returns a 2D velocity estimate, and by searching over vertical motion and over rotation can be made to return a 6D velocity estimate.

We can also analyze the different sources of error of our evaluation method, which attempts to measure how accurately we are estimating the velocity of different objects. 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% of the error of our method, which has an RMS error of 0.52 m/s when using color. The Velodyne Lidar, on the other hand, has reported errors of less than 2 cm, which would account for 3.8% of our total error [2]. We further note that our velocity estimates are averaged across 515 tracked objects and 31,994 separate frames. The standard error of our velocity estimates is 0.003 m/s, computed as $SE = s/\sqrt{n}$, where s is the standard deviation and n is the number of samples, indicating that our

RMS estimates are stable.

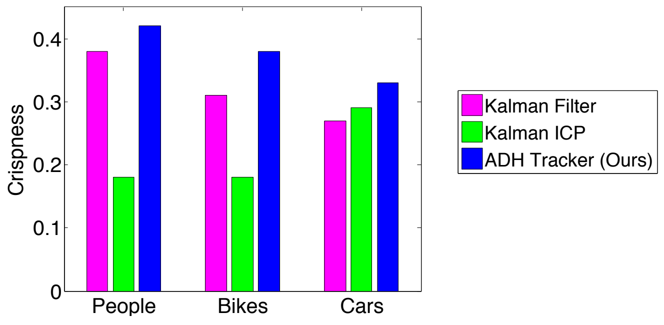
By looking at the mean velocity error, we can determine if there is a bias in our velocity estimates. When tracking using color, our mean velocity error is -0.03 m/s, which is 5.6% of our total error. A completely unbiased tracker would have a mean velocity error of 0, when averaged over an infinite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

It is also interesting to compare the error of our method to the half-resolution of our sensor. The resolution of our sensor decreases 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 Supplementary Figure 11. We see from this figure that, within 50 m, our accuracy nearly matches the half-resolution of the sensor. Of course, 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 the 3D Lidar for tracking.



Supplementary Figure 11. 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.

We can further understand the performance of our tracker by building object models using the velocity estimates produced by our tracker, and then evaluating the crispness of these models. This procedure is described in Section VIII-C of the paper. We can plot these crispness scores as a function of the object type, shown in Supplementary Figure 12. From this figure, 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 have many local optima, leading ICP to get stuck in a local optimum and result in a poor alignment.



Supplementary Figure 12. Crispness of models built using our tracker, compared to models built using the baseline methods.

References

[1] Chassis systems control lrr3: 3rd generation long-range radar sensor. URL http://www.bosch-automotivetechnology.com/media/db_application/downloads/pdf/safety_1/en_4/lrr3_datenblatt_de_2009.pdf.

[2] Velodyne lidar hdl-64e datasheet. URL http://velodynelidar.com/lidar/products/brochure/HDL-64E%20S2%20datasheet_2010_lowres.pdf.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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