Precision Tracking with Sparse 3D and Dense Color 2D Data

David Held, Jesse Levinson, and Sebastian Thrun

Abstract— **Precision tracking is important for predicting the behavior of other cars in autonomous driving. We present a novel method to combine laser and camera data to achieve accurate velocity estimates of moving vehicles. We combine sparse laser points with a high-resolution camera image to obtain a dense colored point cloud. We use a color-augmented search algorithm to align the dense color point clouds from successive time frames for a moving vehicle, thereby obtaining a precise estimate of the tracked vehicle's velocity. Using this alignment method, we obtain velocity estimates at a much higher accuracy than previous methods. Through pre-filtering, we are able to achieve near real time results. We also present an online method for real-time use with accuracies close to that of the full method. We present a novel approach to quantitatively evaluate our velocity estimates by tracking a parked car in a local reference frame in which it appears to be moving relative to the ego vehicle. We use this evaluation method to automatically quantitatively evaluate our tracking performance on 466 separate tracked vehicles. Our method obtains a mean absolute velocity error of 0.27 m/s and an RMS error of 0.47 m/s on this test set. We can also qualitatively evaluate our method by building color 3D car models from moving vehicles. We have thus demonstrated that our method can be used for precision car tracking with applications to autonomous driving and behavior modeling.**

I. INTRODUCTION

Precise object tracking is a key requirement for safe autonomous navigation in dynamic environments. For example, tracking is essential when changing lanes in heavy traffic, in order to ensure that there is sufficient room for the lane change based on the current and expected future positions of nearby vehicles. Accurate tracking can also help an autonomous car predict when another car is about to enter an intersection; observing that another car is starting to inch forward often indicates the driver's intention to enter the intersection. Detecting such subtle movements is important for predicting the behavior of other drivers, and precision tracking is necessary for this and many other driving scenarios.

In addition to the plethora of online applications of tracking, precise tracking also enables significant advances in offline mapping and assists in analyzing the behavior of dynamic obstacles. For example, accurate offline tracking can be used to build structural and behavioral models of other cars. As we show, precisely aligning and then accumulating multiple observations of moving rigid targets allows us to generate accurate 3D models from moving objects.

Manuscript received September 16, 2012.

D. Held, J. Levinson, and S. Thrun are with the Computer Science Department, Stanford University, Stanford, California 94305 USA {davheld, jessel, thrun}@cs.stanford.edu

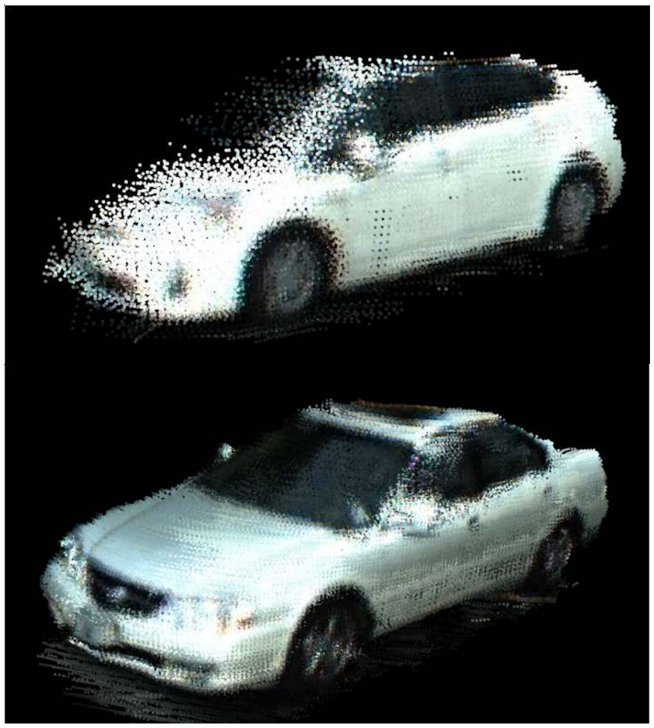

Fig. 1. Two 3D car models created using the color-assisted grid search alignment method. Note that these cars were moving while the alignment was being performed.

Often, tracking is a component in a perception pipeline that also includes segmentation and classification algorithms. In this paper, we take segmentation and classification as given and focus solely on the precise tracking and velocity estimation of objects that have already been segmented and classified as vehicles. We are specifically interested in advancing the state of the art in tracking accuracy, with the goal of tracking the position and velocity of vehicles with lower error than previously published work. Therefore, we use the laser point-cloud-based segmentation, data association, and classification algorithms described in [1] and focus strictly on precision tracking for the remainder of the paper.

II. RELATED WORK

There have been many efforts at using laser rangefinders for tracking. In general, these trackers fit 2.5D rectangular bounding boxes to the 3D point clouds, followed by a Kalman or particle filter [2], [3] to smooth the results. Some have explicitly modeled occlusions to improve performance [4], [5], and most recently, others have considered broader classes of obstacles beyond vehicles [6], [7].

Rather than estimating object boundaries with rectangles, several laser-based approaches have modeled objects as a collection of points. Feldman et al. [8] used ICP to track participants in sports games using single-beam lasers. Vanpoperinghe et al. [9] tracked vehicles using a single-beam laser in this manner.

Recently, several hybrid systems have been developed that use both depth and image data for tracking. Initial work by Wender and Dietmayer [10] used laser scanners to initialize hypotheses for visual car trackers. Other groups have up-sampled range data to assign each pixel in an image to a 3D position [11]–[13]. Manz et al. inferred the 3D location of a vehicle using only a monocular camera but with the knowledge of a previously-acquired 3D model [14]. Variants of ICP incorporating color have been explored, such as work by Men et al. [15].

In this paper, we present a novel approach for precision car tracking by combining a sparse laser and a high-resolution camera. First, we project the laser points onto the camera image and use bilinear interpolation to estimate the 3D position of each pixel. We also estimate which pixels belong to occluding objects and remove those from the interpolated point cloud. We then use a color-augmented filtered grid search algorithm to align successive frames of colored point clouds. This alignment method enables us to precisely track a vehicle's position and velocity, achieving much higher tracking accuracies than the baseline algorithms. We can also use this alignment method to build dense color 3D object models from moving objects. By leveraging the unique strengths of sparse laser rangefinders and dense 2D camera images, we can achieve high tracking accuracies.

Finally, we present a novel method for quantitatively evaluating the accuracy of our tracking algorithm. Despite the importance of measuring tracking accuracy across a variety of tracked objects, existing literature largely fails to do so. Many papers that combine tracking with segmentation and classification only report the binary accuracy of the segmentations and/or classifications, with no quantitative evaluation of the accuracy of the tracks' distance or velocity estimates [4], [5], [8]. Of those that do quantify distance or velocity accuracy, most do so on only one or a few tracks, either by equipping a single target vehicle with a measuring apparatus [14], or by hand-labeling objects or points from a small number of tracks [7], [9].

In contrast, our approach measures accuracy by tracking parked cars in a local reference frame in which they appear to be moving relative to the ego vehicle. To our knowledge, this is the first attempt to quantitatively evaluate the accuracy of velocity estimates from a tracking algorithm in a fully automatic framework on a large number of test vehicles. We achieve these tracking velocity estimates without requiring human intervention or labeling of data. Consequently, we are able to present significantly more comprehensive quantitative results than have been previously published in the tracking literature.

III. INTERPOLATION

We are initially given the 3D points of a vehicle that we desire to track. To combine information from our sparse 3D point-cloud with a high-resolution 2D camera image, we determine which pixels belong to this object and estimate their 3D location.

We would like to estimate which pixels belong to the target vehicle and which might belong to an occluding object. To do this, we first project all of the 3D points for the target vehicle onto the image. For every pixel p in the 2D convex hull of this projection, we find the nearest projected points in each of the 4 quadrants (upper left, upper right, lower left, and lower right) surrounding the target pixel. Suppose these 4 points are f_1, f_2, f_3, f_4.

Then, we search for potential occluding points. We find the 3D points from the laser that are located between the sensor and the target vehicle; these points might belong to an occluding object. We then project all of these points onto the 2D image. We find the nearest projected points from this set in each of the 4 quadrants surrounding the target pixel. Suppose these projected points, the potential occluders, are a_1, a_2, a_3, a_4.

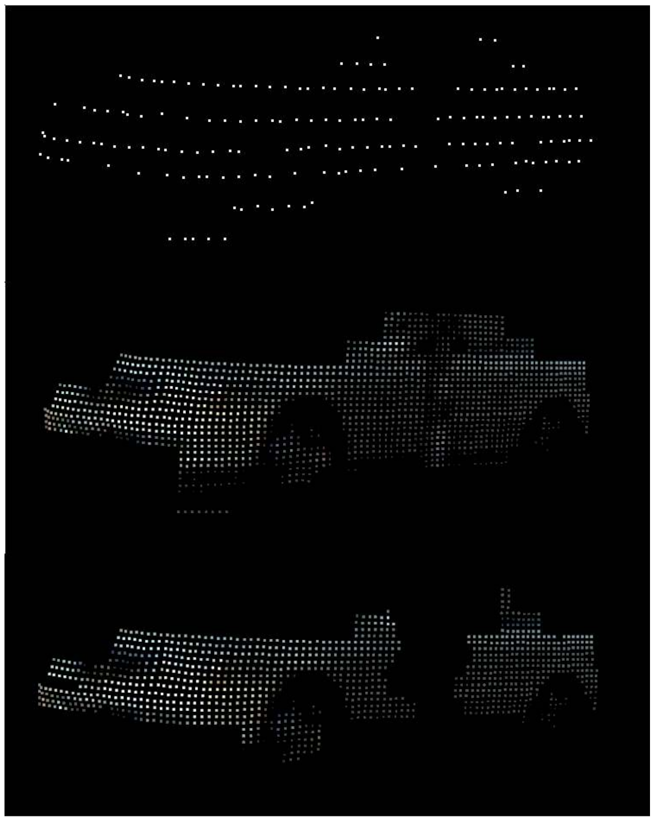

Fig. 2. Top: Sparse achromatic 3D points obtained using a laser. Middle: Up-sampling the point cloud by interpolating pixels from a camera image. The 3D location of each pixel is estimated using bilinear interpolation. Bottom: Removing pixels corresponding to occluding objects, e.g. a pole.

We then compare a_i to f_i for $i \in \{1, 2, 3, 4\}$. Suppose that, for each projected point a_i, the corresponding 3D point is a'_i, and similarly, the corresponding 3D point for f_i is

f'_i. Then if $\exists i$ s.t. $d(a'_i, f'_i) > d_{max}$, where $d(a'_i, f'_i)$ is the 3D Euclidean distance between a'_i and f'_i and d_{max} is an appropriately chosen threshold, then we say that a'_i is potentially occluding our target pixel p and we do not include pixel p in the interpolation.

Although this method for estimating occluded pixels is conservative, a typical image is very dense, so we can afford to miss a few object-pixels; the cost of including a pixel from an occluding object is that it could alter the value of the color-based energy function and cause us to find a poor alignment. An example of this occluding object removal is shown in Figure 2. The thresholds for this step were chosen using examples from a validation set which was recorded on a separate day and time from the test set.

Once we have determined that a pixel is likely part of the target object, we estimate its 3D location. We perform bilinear interpolation using the projected points f_1, f_2, f_3, f_4 found above. Suppose that f_1 and f_2 are the two upper projected points and f_3 and f_4 are the two lower projected points, as shown in Figure 3. We first linearly interpolated between f_1 and f_2, and then we linearly interpolate between f_3 and f_4. Finally, we linearly interpolate between these two interpolated positions to estimate the final 3D position of the pixel p.

To do this, we first compute the fraction s_1 and s_2 of the horizontal distance between pixel p and each of the neighboring pixel pairs, as

$$s_1 = \frac{p_u - f_{1,u}}{f_{2,u} - f_{1,u}}$$
$$s_2 = \frac{p_u - f_{3,u}}{f_{4,u} - f_{3,u}}$$

where the u subscript denotes the horizontal coordinate for a pixel (u,v) in image space. Then we perform bilinear interpolation as

$$p_t = f_1 + s_1(f_2 - f_1)$$
$$p_b = f_3 + s_2(f_4 - f_3)$$
$$s_3 = \frac{p - p_b}{p_t - p_b}$$
$$p_{interpolated} = p_b + s_3(p_t - p_b)$$

where s_1, s_2, and s_3 are computed in pixel-space whereas the interpolated points p_t, p_b, and $p_{interpolated}$ are computed in both pixel space and separately in 3D coordinates. The resulting $p_{interpolated}$ is close to p in pixel space, and the corresponding 3D location of $p_{interpolated}$ is used as the estimated 3D location of pixel p. This is shown schematically in Figure 3.

If there are less than 4 projected points surrounding pixel p, then this pixel is near the 2D contour of the tracked object (in the 2D image) and we are not able to accurately estimate its 3D position; thus we ignore these border pixels as well. Last, as a sanity check, we verify that the interpolated location of this pixel lies within the 3D convex hull of the original object points recorded from the laser; if it does not, then we ignore the pixel. Finally, we also include the 3D

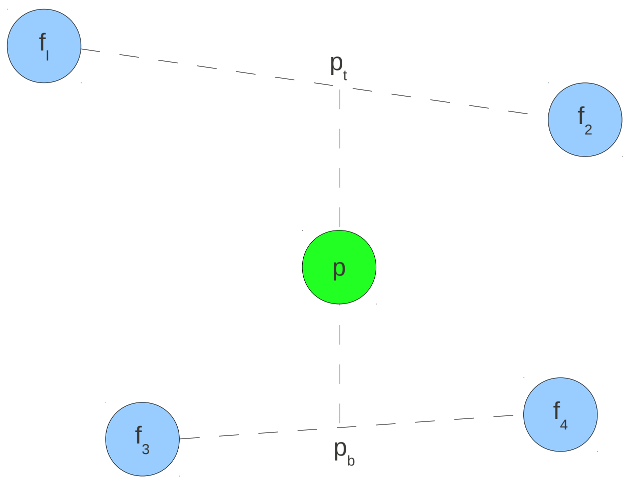

Fig. 3. Diagram of the bilinear interpolation used to estimate the 3D position of an image pixel p, shown in green. The 4 nearest surrounding laser points projected onto the image are shown in blue. First, the top 2 nearby laser points are linearly interpolated to get p_t. Next, the bottom 2 laser points are linearly interpolated to get p_b. Finally, p_t and p_b are themselves interpolated to get the final 3D position estimate for the pixel p.

positions of each of the original laser points from the target object, colored by their projected location onto the image. These interpolation steps result in a dense color 3D point cloud, as shown in Figure 2.

IV. ALIGNMENT

To estimate the velocity of the target object, we find the best alignment of the object's dense color 3D point cloud from one time step to the next. This alignment directly gives the translation and rotation of the target vehicle between two sensor readings. If the times of the readings are known, then the vehicle's velocity can be precisely determined.

To find the optimal alignment, we perform a pre-filtered grid search optimization. First, we list a number of candidate alignments for a coarse search in the 6D space of possible alignments. In our implementation, the candidate alignments are centered around the alignment that causes the two object centroids to coincide. Around this initial guess we first determine the most likely 2D positions in the ground plane. We initially propose a 2D array of positions spanning 2 m in each direction, spaced 10 cm apart.

Next, we use an occupancy grid to filter each of these candidate locations. Because this is just a coarse pruning step, we can significantly downsample the original point cloud to speed up this step. In our implementation, we down-sampled the point cloud to contain about 150 points before performing this step. To create the 2D occupancy grid, we then take our model point cloud (from the previous time step) and record whether each square was occupied by some point in the model when projected onto the ground plane. We use this occupancy grid to score each of our candidate locations based on the percentage of points that land either in or adjacent to an occupied square. Any location that has a score less than 0.9 times the maximum score can get pruned away. This simple heuristic often allows us to reduce the original search space by 1 or 2 orders of magnitude.

We can similarly prune the vertical search space (perpendicular to the ground plane). We form a set of candidate translations in the vertical direction spanning 1 m, spaced 10 cm apart. We create a 1-dimensional occupancy grid and project points onto the vertical-axis to prune the search space in the vertical direction. As before, we prune away any vertical translation that has a score less than 0.9 times the maximum score.

Next, we take the remaining candidate translations and score each alignment according to an energy function. The energy function is given by

$$\sum_i \left[(\hat{d}_i - \hat{d}_j)^2 + w_1 (v_i - v_j)^2 \right] + w_0 \cdot n_0$$

where we iterate over each of the current points i and find the closest point j in the model that is within the range of our step size (initially 10 cm as described above). For each pair of points we compute the Euclidean distance $(\hat{d}_i - \hat{d}_j)$ and the color-distance in value-space $(v_i - v_j)$. The value color space was chosen based on a number of experiments on a validation set to be the most useful color space for car alignment; cars often have large dark areas (e.g. tires) or light areas (e.g. the body or hubcaps) for which the hue angle is not well-defined but the value is very informative. We weight the value by w_1, chosen to be 10 using a validation set.

Next we compute the number of points n_0 that did not have a match to our model within our search radius. We weight this by w_0, chosen to be $10 \cdot 75^2$ using our validation set. This parameter roughly corresponds to the fact that the value distance $v_i - v_j$ will probably be within 75 for a pair of matched points. This parameter was also chosen using our validation set.

Using this energy function, we choose the candidate location with the highest score from our initial coarse search. Finally we iteratively reduce the step size and find the new best location, still using the above energy function. After our initial coarse optimization we assume that the energy function is locally convex in the 6D alignment space, so for the fine-grained search steps we vary each dimension independently (though we continue to search the ground plane directions jointly, as they generally correspond to the largest directions of motion for a vehicle). All rotations are taken about the centroid of the tracked object. After finding the optimal alignment, we record the optimal transformation and divide by the time difference between the sensor readings to determine the tracked vehicle's velocity.

Importantly, our contribution assumes nothing about the tracked object's motion model. In order to highlight both the precision and generalizability of our scan alignments, all of our results are derived from raw scan alignments, without the benefit of the smoothing that arises from filtering with a motion model. Consequently, our results are applicable even to erratically moving objects, and they do not suffer from any biases that result from motion assumptions. At the same time, our algorithm could easily be incorporated as part of the measurement model into traditional Bayesian filtering techniques, such as Kalman or particle filters.

V. SYSTEM

The raw 3D point cloud for each frame is directly sensed using a Velodyne HDL-64E S2 rotating 64-beam LIDAR that spins at 10 Hz, returning 100,000 points per spin over a horizontal range of 360 degrees and a vertical range of 26.8 degrees. As a result, the points associated with distant objects will be relatively sparse. We combine these points with the high-resolution camera images obtained from 5 Point Grey Ladybug-3 panoramic RGB cameras which use fish-eye lenses to capture 1600x1200 images at 10 Hz. The laser is calibrated using the method of [16], and the camera is calibrated to the laser using the method of [17]. We use the Applanix POS-LV 420 inertial GPS navigation system to obtain the vehicle pose. As explained in section VI-A, we also use this navigation system to obtain ground truth estimates for tracking.

VI. RESULTS

As explained above, in this paper we use the laser point-cloud-based segmentation, data association, and classification algorithms described in [1]. Our precision tracking algorithm takes as its input the tracks output from [1] that are classified as vehicles. Note that the segmentation algorithm used in [1] is not particularly accurate and often over-segments a single vehicle into multiple pieces. Our method will nonetheless work directly with these segments, showing that our method is robust to a range of segmentation errors.

A. *Quantitative Evaluation Method*

To quantitatively evaluate the results of the tracking algorithm, we present a novel approach based on tracking parked cars in a local reference frame. In a local reference frame, a parked car appears to be moving, and tracking the car in this local reference frame is equivalent to tracking a moving car in a world reference frame. However, because we have logged our own velocity, we can compute the ground-truth relative velocity of a parked vehicle and quantitatively evaluate the precision of our tracking velocity estimates.

To determine which cars from our logfile are parked, we compute the displacement of the centroid for each track in the fixed world reference frame. Because of occlusions and viewpoint variation, we estimate that the centroid of the 3D points on a parked car may appear to move by as much as 3 m. Thus we iterate over all tracks and check this criterion until we obtain a list of tracks associated with potentially parked cars. However, some of these tracks might still correspond to moving vehicles, so we manually filter out the moving vehicles from this set. The remaining parked cars will be used for our quantitative evaluation.

Next, we attempt to track each parked car in a local reference frame in which it appears to be moving relative to the ego vehicle. During our logfile, we drive past each of these parked cars. However, in a local reference frame, we appear to be stationary, and the parked cars appear to be moving past us. Thus by tracking parked cars in a local reference frame, we are simulating the situation of tracking moving cars in a global reference frame. The importance of this approach is that, because we know the true velocity of a parked car in a global reference frame ($i.e.$ 0 m/s) we can use a reference frame transform to obtain the ground truth for the perceived velocity of the parked car in a local reference frame. Thus we can quantitatively evaluate the accuracy of our tracking method, without requiring any human annotations that are likely to be noisy, inaccurate, and time-consuming. In contrast, our quantitative evaluation is accurate and can be tested on a large number of vehicles by comparing the tracked velocity to the ground truth recorded by our vehicle.

Nonetheless, it should still be noted that although we can use parked cars in a local reference frame to quantitatively evaluate our performance, there are some differences between tracking cars in this manner compared to tracking actual moving cars in a world reference frame. On the one hand, the segmentation and tracking issues of cars in a parking lot are often even more severe than in normal driving situations. Parked cars are often occluded by other nearby parked cars, causing frequent undersegmentations or difficult occlusions for alignment. Despite these difficulties, we are able to use our tracking method to accurately track cars in this scenario.

On the other hand, the background and illumination changes due to shadows do not change significantly as we drive past a parked car, making tracking parked cars in some ways a bit easier than tracking moving cars. Still, reflections on parked cars can vary as we change our position to the parked car relative to the sun. Additionally, viewpoint and occlusions can both vary as we drive past a parked vehicle.

To obtain ground truth data for evaluation, we must compute the relative velocity of a parked car in a local reference frame. Using our navigation system, we obtain ground truth knowledge of our own instantaneous velocity, $v = (v_{tx}, v_{ty})$. The relative velocity of the tracked car in our local reference frame resulting from this translation is simply given by $v_t = (-v_{tx}, -v_{ty})$

Next, we must add the effect of our rotational velocity on the relative position of the tracked car. If we rotate by $\Delta\theta$, and we are tracking a parked car located at position (x, y) in our local reference frame, then the perceived change in position of the tracked car as a result of our rotation is given by

$$\Delta x_\theta = r(-\Delta\theta)(-\sin\theta)$$
$$\Delta y_\theta = r(-\Delta\theta)\cos\theta$$

where $r = \sqrt{x^2 + y^2}$, $\theta = \arctan(y/x)$, and $\Delta\theta$ is the yaw of the ego vehicle. This is shown schematically in Figure 4.

If the yaw of the ego vehicle occurs over Δt seconds, then the total relative velocity of the tracked vehicle can be computed as:

$$v_x = -v_{tx} + \Delta x_\theta/\Delta t$$
$$v_y = -v_{ty} + \Delta y_\theta/\Delta t$$

However, the direction of this velocity is in the orientation of the world coordinates, whereas the estimated velocity from tracking is in a local reference frame. To compare the estimated velocity to the ground truth, we rotate the estimated velocity into the orientation of the world reference frame.

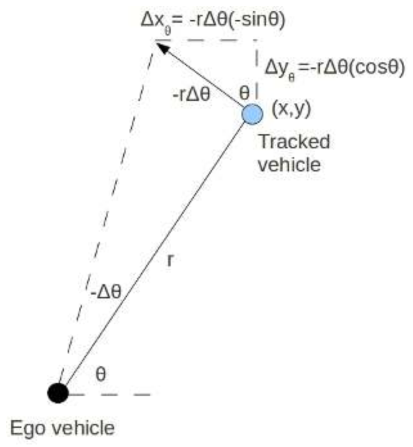

Fig. 4. Relative motion of a tracked vehicle resulting from yaw of the ego vehicle.

B. Tracking performance

From a 6.5 minute log file, we drive past 558 cars parked in nearby parking lots or on the side of the road. We track cars that are oversegmented into multiple pieces, but we do not attempt to track cars that are undersegmented together with lamp posts or other nearby cars. We also eliminate tracks for which the colors are washed out by lens flare. In a real autonomous driving scenario, a hood can be placed over the camera to block out most of the direct sunlight and avoid lens flare most of the time. We also evaluate our performance with the color weighted at $w_1 = 0$ to determine the performance of our method when color is not available, such as in lens flare, at night, or in other poor visibility weather conditions.

Our implementation runs at a quarter of real-time speeds on a single threaded i7 Intel processor, taking about half a second per frame. However, a GPU implementation, multi-threading, or a faster processor could all significantly reduce the runtime. Additionally, a slightly less accurate but real-time version of the algorithm is also presented in which the interpolation step is skipped entirely, which simultaneously saves time from interpolating and immediately results in a significantly down-sampled point cloud compared to the interpolated version. The offline version is still useful for accurate behavior modeling, while the online version can be used for real-time tracking and autonomous driving.

TABLE I
TRACKING ACCURACY

Tracking Method	Frames with > 50 points		All frames	
	RMS error (m/s)	Mean absolute error (m/s)	RMS error (m/s)	Mean absolute error (m/s)
Color-augmented grid search with interpolation	**0.47**	0.27	**0.86**	**0.40**
Color-augmented grid search (no interpolation - real time)	0.54	0.32	0.88	0.45
Color-augmented ICP with interpolation	0.60	**0.25**	1.00	0.41
Grid search with interpolation (no color)	0.67	0.31	1.05	0.46
ICP with interpolation (no color)	0.77	0.38	1.10	0.55
Centroid difference	1.00	0.61	1.27	0.77

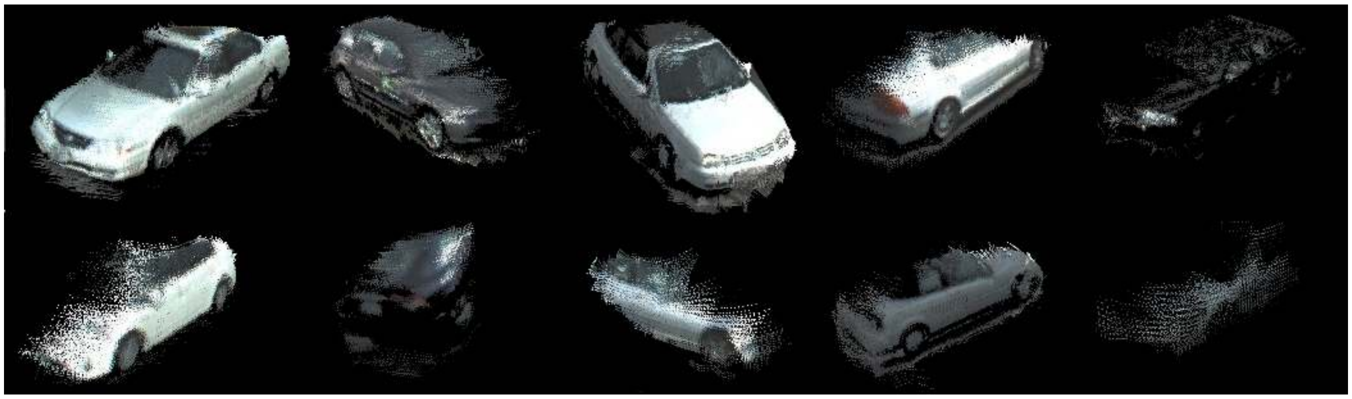

Fig. 5. Car models built using our interpolation and color-augmented grid search method, obtained from both moving and stationary cars.

We show performance of our method for frames with any number of points, and separately we show our performance considering only frames that contain at least 50 raw laser points (before interpolation), for which there is enough information to obtain better tracking accuracies. In our logfile, 75% of all frames contain at least 50 raw laser points. Although our method sometimes works with fewer than 50 points, the accuracy of our approach depends on the color and shape variation of the frames being aligned. To run this method on frames with fewer than 50 points, it is recommended to incorporate a motion model prior, since the data will be less informative. However, in cases with more than 50 points, the data is sufficient to accurately determine the velocity, and no motion model is needed.

After eliminating tracks with lens flare or under-segmentation issues, we obtain velocity estimates from 19,526 frames and 465 separate tracked vehicles if we only include frames with at least 50 raw laser points. If we include frames with any number of points, then we have estimates from 26,254 frames and 466 separate vehicles.

The results can be seen in Table I. Our full method using interpolation and color-augmented grid search has an RMS (root mean square) error of 0.47 m/s and a mean absolute error of 0.27 m/s for frames with at least 50 points. The remaining errors are mostly a result of down-sampling and early pruning of correct alignments as well as ambiguities resulting from heavy occlusions. If color is not available, due to lens flare, nighttime, or poor weather conditions, the algorithm can be run with w_1 set to 0. The result is a small drop in accuracy to 0.67 m/s RMS error and 0.31 m/s mean absolute error. The simplest baseline approach is simply to align the centroids of successive frames, which achieves 1.00 m/s RMS error and 0.61 m/s mean absolute error.

Although the above algorithm takes an average of 2.9 seconds per frame and is thus not real-time, real-time results can be achieved to obtain an online algorithm by skipping the interpolation step. The resulting algorithm takes only 86 ms per frame, on average. This simplification results in only a small decrease in accuracy, with 0.54 m/s RMS error and 0.32 m/s mean absolute error, which is comparable to the full offline algorithm. This indicates that while interpolation clearly boosts performance, the sensor fusion of combining shape and color is perhaps the biggest contributor to the accuracy of our results.

Another approach is to use ICP with interpolation. ICP is a relatively popular method for point cloud alignment. Although there are many variations of ICP, it is fundamentally a hill-climbing approach that is susceptible to getting stuck in local minima and relies on good initializations. On the other hand, the pre-filtered grid search methods presented here were both faster and more reliable than the ICP implementation that we tested with.

We used the ICP implementation from the PCL library of [18] and allowed it to run for 50 iterations, which is 7 times slower than real-time (including the time for interpolation), in order to ensure convergence. We tested both the standard ICP algorithm as well as a color-augmented version, in which nearest neighbors are computed in (x, y, z, value) space. The standard ICP algorithm performed considerably better than just aligning the centroids, achieving an RMS of 0.77 m/s for frames with at least 50 points. The color-augmented ICP with interpolation performed even better, with 0.60 m/s RMS error and 0.25 m/s mean absolute error. While the mean error for the color-augmented ICP algorithm is lower than any of the the other methods tested, the high RMS error indicates how ICP can easily go off-track and get stuck in a local minimum that is far from the optimal alignment.

ICP also does especially poorly when including frames with segmentation errors, for which the RMS error is 0.73 m/s (for frames with at least 50 points). In the same scenario, the pre-filtered grid search has an RMS accuracy of 0.50 m/s. Thus our method is also much more robust to segmentation errors than ICP, which again is sensitive to the initialization and can get stuck in a local minimum.

We can also view the effect of distance on accuracy in Figure 6, considering frames with any number of points. The color-augmented grid search performs the best at all distances, with RMS errors dropping to 0.14 m/s for cars within 5 m. At distances of 20 m or greater, the color-augmented grid search with no interpolation, which runs in real-time, has only slightly worse performance. The method that uses only centroids performs significantly worse at all distances.

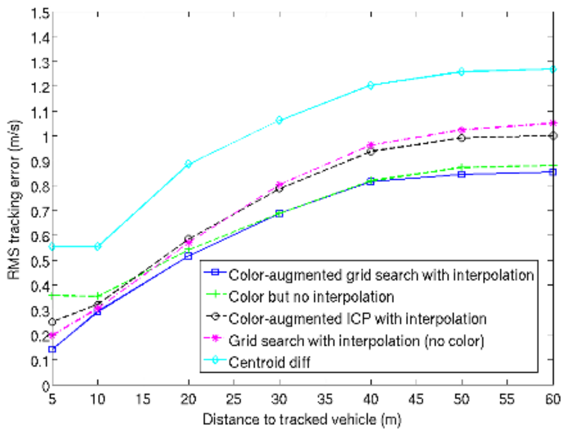

Fig. 6. Tracking accuracy as a function of distance to the tracked car.

C. Model Building

In addition to quantitatively evaluating our tracking performance in Section VI-B, we can also qualitatively evaluate our performance by building car models using our tracking algorithm. One disadvantage of the quantitative evaluation of Section VI-B is that we can evaluate only on parked cars tracked in a local reference frame (in which they appear to be moving relative to the ego vehicle). On the other hand, we can qualitatively evaluate our method by viewing the car models built from tracking both stationary vehicles (in a local reference frame in which they appear to be moving) as well as moving vehicles (which are actually moving in world-coordinates). This allows us to demonstrate that our tracking method will work in real-world scenarios for estimating the velocity of other moving vehicles.

The alignment method described in Section IV can be used to accumulate point clouds from moving objects to build full 3D models. Rather than just aligning to the previous frame, we align to the accumulated model built from all previous frames. These models are obtained automatically using the track extraction and classification pipeline from [1] and the alignment method from this paper. See Figures 1 and 5 for examples of car models built using this method. Some of the models in Figure 5 are obtained from tracking moving cars, whereas other models are obtained from tracking parked cars in a relative reference frame in which they appear to be moving.

Our tracking method is accurate even in the difficult evaluation scenario of tracking parked cars in a local reference frame with heavy occlusions from nearby parked cars. We expect that our method will be even more accurate in a scenario in which the tracked objects are moving, which may result in fewer segmentation errors. The models shown in Figure 5 demonstrate that this method works well for tracking moving vehicles. While we do not have any direct quantitative measure of our tracking accuracy from moving vehicles, the quality of these models demonstrates that the method works well for tracking moving cars. Additionally, the evaluation in Section VI-B gives an estimate on what kinds of quantitative accuracies one might expect in this case.

VII. CONCLUSION

We have demonstrated that our interpolation and color-augmented grid search algorithm provides significantly more accurate velocity estimates compared to previous methods. This method runs in near-real time and can be used for offline behavior modeling. An online version that is nearly as accurate is also presented that can be used for real-time tracking and autonomous driving.

We have also developed a novel method for quantitatively evaluating the accuracy of our tracking method, by tracking parked cars in a relative reference frame. This method allows us to evaluate our approach on a large test set of 466 separately tracked vehicles and quickly determine the accuracy of our velocity estimates without any human annotations.

The alignment method can also be used to automatically generate a collection of color 3D object models. These object models can be used for other fine-grained classification tasks, and our precise velocity estimates can be used to learn accurate behavior models for vehicles in various driving scenarios.

ACKNOWLEDGMENTS

We would like to thank Dave Jackson for his insight into the reference frame transform and for his encouragement throughout this research. We would also like to thank Jake Lussier for his research into previous tracking methods.

References

[1] A. Teichman, J. Levinson, and S. Thrun. Towards 3d object recognition via classification of arbitrary object tracks. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4034 –4041, May 2011.

[2] D. Streller, K. Furstenberg, and K. 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, 2002.

[3] D. Ferguson, M. Darms, C. Urmson, and S. Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1149 –1154, june 2008.

[4] Anna Petrovskaya and Sebastian Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26:123–139, 2009.

[5] N. 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, may 2012.

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

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

[8] 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*, pages n/a–n/a, 2012.

[9] E. Vanpoperinghe, M. Wahl, and J.-C. Noyer. Model-based detection and tracking of vehicle using a scanning laser rangefinder: A particle filtering approach. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 1144 –1149, june 2012.

[10] S. Wender and K. Dietmayer. 3d vehicle detection using a laser scanner and a video camera. *Intelligent Transport Systems, IET*, 2(2):105 –112, june 2008.

[11] J. Dolson, Jongmin Baek, C. Plagemann, and S. Thrun. Upsampling range data in dynamic environments. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1141 –1148, june 2010.

[12] Derek Chan, Hylke Buisman, Christian Theobalt, and Sebastian Thrun. A noise-aware filter for real-time depth upsampling. In Andrea Cavallaro and Hamid Aghajan, editors, *ECCV Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications*, pages 1–12, Marseille, France, 2008.

[13] A. Harrison and P. Newman. Image and sparse laser fusion for dense scene reconstruction. In *Proc. of the Int. Conf. on Field and Service Robotics (FSR)*, Cambridge, Massachusetts, July 2009.

[14] M. Manz, T. Luettel, F. 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, may 2011.

[15] Hao Men, B. Gebre, and K. Pochiraju. Color point cloud registration with 4d icp algorithm. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1511 –1516, may 2011.

[16] Jesse Levinson and Sebastian Thrun. Unsupervised calibration of multi-beam lasers. In *Proc. 12th IFRR Int'l Symp. Experimental Robotics (ISER'10)*, New Delhi, India, December 2010.

[17] Jesse Levinson and Sebastian Thrun. Automatic calibration of cameras and lasers in arbitrary scenes. In *Proc. 13th IFRR Int'l Symp. Experimental Robotics (ISER'12)*, Quebec City, Canada, June 2012.

[18] 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.