Combining Model- and Template-based Vehicle Tracking for Autonomous Convoy Driving

Carsten Fries, Thorsten Luettel and Hans-Joachim Wuensche

***Abstract*— This paper presents a robust method for vehicle tracking with a monocular camera. A previously published model-based tracking method uses a particle filter which needs an initial vehicle hypothesis both at system start and in case of a tracking loss. We present a template-based solution using different features to estimate a 3D vehicle pose roughly but fast. Combining model- and template-based object tracking keeps the advantages of each algorithm: Precise estimation of the 3D vehicle pose and velocity combined with a fast (re-) initialization approach. The improved tracking system was evaluated while driving autonomously in urban and unstructured environments. The results show that poorly visible vehicles can be tracked during different weather conditions in real-time.**

Fig. 1: Vehicle tracking for autonomous convoy driving.

I. Introduction

In recent years, research on autonomous driving has intensified constantly. Many fields of application exist where advanced driver assistance systems have to detect other vehicles. Some common ones are the collision avoidance system, adaptive light control or adaptive cruise control. E.g. an adaptive cruise control situation is driving on the highway while an autonomous system keeps the distance to the car in front. One or more sensors like RADAR, LIDAR or camera sensors are used to obtain information about the surrounding environment [1]–[3]. The measurement range goes from the 1D distance up to the complete 3D position, orientation and velocity.

Highways have smooth trajectories with low curvature changes, which is a big advantage for systems that measure the vehicle distance. In dense city traffic or unstructured environments, e.g. small forest tracks, vehicle detection must be more accurate. This paper focuses on a low-cost solution for following a leading vehicle with just one monocular vision sensor (see fig. 1). It is based on the work about model-based vehicle tracking by Manz et al. [4].

The outline of the paper is as follows: Section I introduces fields of application and related work on vehicle tracking methods. Section II describes our algorithm in three subsections. Experimental results collected while driving in urban and non-urban environments are presented in section III. Finally, conclusions are given in section IV.

A. Related Work

The related work in this paper has a focus on non-stationary passive vision systems and their algorithms. I.e. the popular but expensive LIDAR and RADAR sensors will not be considered. Passive vision sensors are frequently used because they are low-priced and have low power consumption. Most publications work with a single camera that is mounted behind the windshield of the vehicle [5]–[10]. They track multiple vehicles without precise information about each vehicle. The algorithms are primarily designed for motorways with clearly visible road markings, a uniform surface and small change in road curvature.

It is common to train a classifier offline, e.g. [5] train a classifier with the fast computable Haar-like features [6]. They detect vehicle rear sides from highway participants that have the same traveling direction. In contrast, [7] train SVM classifiers to detect all sides of a vehicle. They use features called Histogram of Orientated Gradients (HOG) and were able to detect any vehicle side on intersections. The classifier also responds with a positive detection if a vehicle is only partially visible. In another publication they basically use the same monocular procedure of vehicle detection, but with stereo position estimation [8]. The vehicle is detected in each camera image and the vehicle location is extracted from the depth map which is created by the stereo system. Other publications focus on stereo-based techniques where objects are segmented, detected and tracked in 3D point clouds [9], [10]. Some authors analyze the optical flow, e.g. to detect vehicles which have an opposite driving direction. If the ego motion is known, vehicles with the same driving direction are detectable as well [5].

Furthermore filters are often used, e.g. Kalman filtering is applied to estimate state values over time [8]. Within the last few years the multidimensional particle filter became very popular. The particle filter can verify up to thousands of hypotheses for the object of interest.

Notably, only few published approaches with manually generated 3D vehicle models had robust tracking results [4], [11].

All authors are with department of Aerospace Engineering, Autonomous Systems Technology (TAS), University of the Bundeswehr Munich, Neubiberg, Germany. Contact author email: carsten.fries@unibw.de

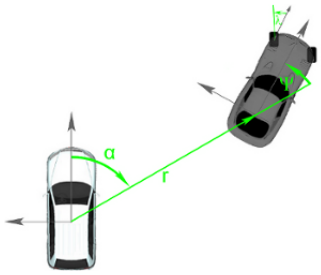
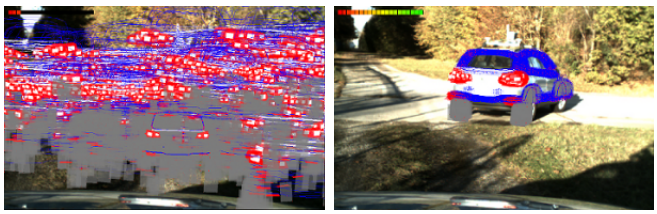


Fig. 2: The state values: Relative translation $\{\mathrm{r}, \alpha, \mathrm{h}\}$ and orientation $\{\psi, \theta, \phi\}$ in cylindrical coordinates, and the object's velocity v and steering angle λ.



(a) (b)

Fig. 3: Particle filter (re-) initialization: (a) Without vehicle prediction. (b) Prediction from template-based tracking.

II. Vehicle Tracking

Two independent applications realize the estimation of the 3D pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}} = \{\mathrm{r}, \alpha, \mathrm{h}, \psi, \theta, \phi\}$ between the ego and the tracked vehicle. The translation $\{\mathrm{r}, \alpha, \mathrm{h}\}$ and orientation $\{\psi, \theta, \phi\}$ is given in cylindrical coordinates. In addition to the relative vehicle pose, the velocity v and the steering angle λ are estimated as well. On the one hand there is the model-based variant which is the main tracking system. It is based on previous work of [4], uses a particle filter and reaches high precision in the estimation of the 3D vehicle pose and velocity. Figure 2 visualizes some of the eight state values. It needs a detailed geometric 3D feature model, called tracking model, which must be available for each vehicle type. The big disadvantage of using a particle filter combined with a complex 3D tracking model is the need for (re-) initialization, e.g. at the start or in case of a loss of visual detection (see fig. 3(a)). For this purpose, a template-based application applies (re-) initialization with its roughly determined vehicle pose and velocity (see fig. 3(b)). Although, the algorithm has a lower accuracy in turns, it needs little processor performance and can detect a vehicle extremely fast and robustly in different weather conditions and during partial occlusions.

The following three subsections describe each tracking application and their combination.

A. Model-based Tracking

The model-based approach uses a multidimensional particle filter and requires a detailed 3D tracking model for each vehicle type. It consists of many features like significant vertices, edges and colored regions and is generated manually.

Fig. 4: Building a template-model from three reference images.

The particle filter derives one thousand hypotheses in the form of 3D poses from the tracked vehicle. The projection of each hypothesis onto the image plane enables the evaluation with the image features to determine the hypotheses weights. Subsequently the particles are sorted by their weight. It turns out that a reliable estimation of the pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}}$ is the arithmetic mean value from the best 10% of the entire particle set. Then $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}}$ is used to project the tracking model to the image plane. A subsequent feature evaluation yields the current tracking quality.

For a detailed description of the model-based tracking we refer to the previous publication [4].

B. Template-based Tracking

This subsection describes the template-based approach which is necessary for (re-) initialization of the particle filter. Figure 3 illustrates the problem if no prediction of the object of interest is available. The goal of the template-based approach is a rough but fast estimation of the 3D vehicle pose and velocity with just three template images. The fundamental steps are the elimination of unnecessary image information combined with an analysis of cascaded image windows in a top down fashion. We introduce a dynamic region growing procedure to segment the images. Furthermore, we show that many image features can be used to detect and verify a vehicle in a time efficient way like cascaded image windows.

1) Pre-processing Steps: The algorithm is based on three template images of the vehicle. The images must be the rear, left and front view of the vehicle to enable image tracking from all viewing directions. The left and front side and the left and rear side should be orthogonal to each other (see fig. 4).

a) Building the Template-model: The three images are used to build a full 3D template-model where the right side is generated by flipping the left image. To create the model in the correct size we developed a feature detector. This detector detects, reads and analyzes the license plate to estimate the real boundary size of the vehicle. This can be done because all license plates in the European Union follow a standardized pattern. The algorithm starts with the license plate detection in the rear image. Vehicle width and height are determined and used for scaling the two images which shows the rear and front side. The image with the left side is only scalable with the vehicle height. After image scaling the image texture which is not inside the convex hull of the vehicle boundary is eliminated by a region growing algorithm. Chapter II-B.2.b introduces the algorithm.

b) Searching and Adding Features: After model generation the vehicle-specific feature extraction starts. Only those

features are useful that are mostly invariant to image translation, scaling, rotation, illumination changes and distortion. This is necessary for a reliable detection during changes of the view direction, weather and environment conditions. This allows us to find feature correspondences between the template-model and images where each feature from the model has an assigned 3D point value. This idea is originally based on [12] where the so-called 3D feature model (surface points + features) was published.

The search algorithm starts with the image view of the rear side. The image is converted to the HSV colorspace to detect the two red rear lights and the license plate from the vehicle. In this process, different conditions must hold. E.g. the symmetry and orientation of and between both rear lights and the license plate are verification criteria. Also the standardized license plate geometry and availability and readability of the characters and numbers are analyzed.

Furthermore, these features are necessary to verify the higher-level classifiers that will be introduced next.

c) Training Cascade Classifiers: A well-trained classifier can improve the performance enormously if it is used in a top-down architecture.

Haar-like [6], local binary pattern (LBP) [13] and HOG [14] features are suitable to train a classifier. In this paper, each feature type is used to train a classifier per vehicle side. The offline training of these classifiers is computationally expensive but in the online application their evaluation needs low computational cost. Furthermore, these classifiers have a high response rate, which means they give many true and a lot of false positives. However they are used in a top-down architecture where false positives are filtered out with the previously presented features (e.g. red rear lights, license plate).

Training of a classifier needs a large collection of negative and positive samples of grayscale images. Negative samples are background images that don't contain the object of interest. In contrast, positive samples show the object of interest whose image location is also known. An object of interest is the front, left or rear side of the vehicle and the location is defined by an upright rectangle $(x, y, width, height)$. The images show urban and non-urban environments with different weather conditions. We use around 10,000 negatives and 20,000 automatically generated positive samples to train a classifier, i.e. the latter requires no manual user selection, which is not common in the literature. This is only possible due to the 3D template-model (built by three images). An algorithm generates 3D vehicle poses randomly which are used to project the template-model to the image plane. Thanks to the template-model, the locations of all vehicle sides are known.

Using the negative and positive samples, three classifiers, the {left+right}, front and rear vehicle side, are trained for every feature type (Haar, LBP and HOG). The training algorithm tries to find the best weights to separate these two classes, the negatives and positives [15]. We use the Gentle AdaBoost Algorithm [6] for training the classifiers. AdaBoost is short for "Adaptive Boosting" and is an iterative algorithm to improve the classification accuracy stage by stage based on a sequence of weak classifiers. The training time of all classifiers is two days in total on the computer described in III.

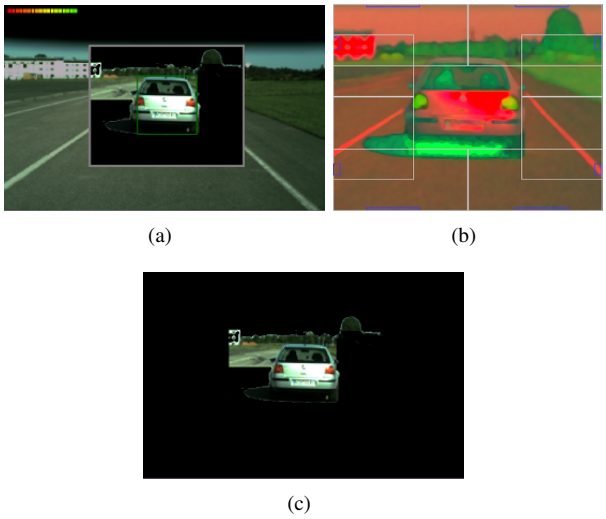


Fig. 5: Region growing: (a) $\mathrm{ROI}_{\mathrm{kalman}}$ (gray-bordered rectangle). (b) Start windows of cell flooding algorithm (white rectangles). Flooding color is determined in the blue rectangles. (c) Eliminated image area.

Further information about cascade training and feature types can be found in [6], [13] and [14].

Note that the template-model generation, feature extraction and classifier training are done once for each new type of leading vehicle in an offline pre-processing step.

2) Detection and Verification: This section describes the online vehicle detection as well as its verification.

a) UKF Vehicle Prediction: An Unscented Kalman Filter (UKF) is applied to predict the vehicle pose $\mathrm{P}_{\mathrm{ego}\rightarrow\mathrm{obj}} = \{\mathrm{r}, \alpha, \mathrm{h}, \psi, \theta, \phi\}$ and velocity v. 3D ego-motion is considered in the prediction step. The predicted pose is projected into the image plane to span a region of interest ($\mathrm{ROI}_{\mathrm{kalman}}$) which is the upright boundary of the full vehicle bounding box. Next, the $\mathrm{ROI}_{\mathrm{kalman}}$ is resized relatively to the last detection time. I.e. only a small area of the image must be evaluated while the vehicle is detected continuously, e.g. in a 50 ms cycle. This procedure greatly reduces the processing time. $\mathrm{ROI}_{\mathrm{kalman}}$ is visualized in figure 5(a) with an gray-bordered rectangle.

b) ROI Clean Up: An important step in our algorithm is the elimination of unnecessary image information to enable real-time performance. For this reason, we introduce the multi-dynamic region growing algorithm, called cell flooding, which tries to remove image pixels (x, y) which do not belong to the tracked vehicle.

The algorithm only needs a hypothesis which is in this case the $\mathrm{ROI}_{\mathrm{kalman}}$. The main idea of cell flooding is to flood cascaded image cells from the outer area to the $\mathrm{ROI}_{\mathrm{kalman}}$ center to detect the vehicle silhouette in n sub-cells. The algorithm works as follows: First, the $\mathrm{ROI}_{\mathrm{kalman}}$ is split into cells (see fig. 5(b)). For each cell the arithmetic mean HSV

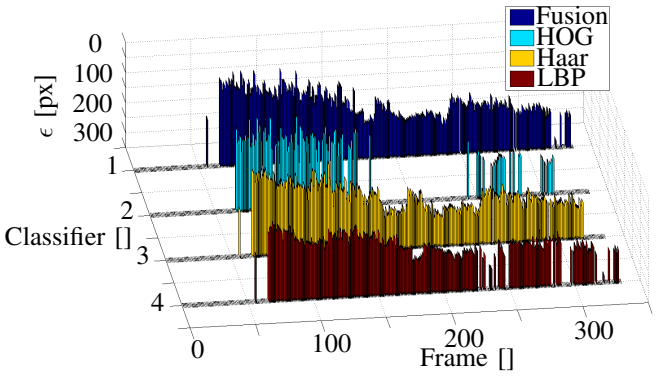


Fig. 6: Comparing classifier detection: The ϵ-axis represents the true positives in which a high bar indicates a low pixel error of the responded upright rectangle. The error is estimated by summation of the root-mean-square-error (RMSE) of the four quadrangle edges between the upright rectangle from the classifier response and the ground truth quadrangle. A ground truth quadrangle is the projection of the DGPS generated pose $\mathrm{P}_{\mathrm{ego}\rightarrow\mathrm{obj}}$ to the image plane. In the first 60 frames is the tracked vehicle not visible. Then it appears and the autonomous driving start.

color value $\left\{\overline{h}, \overline{s}, \overline{v}\right\}$ is estimated. Then the outer cells are processed first and the pixel processing grows in the direction to the center of the $\mathrm{ROI}_{\mathrm{kalman}}$. A weight w is the distance to the $\mathrm{ROI}_{\mathrm{kalman}}$ center and influences the aggressiveness of the region growing. The flooding stops in a sub-cell if a change of the flooding color $\left\{\overline{h}, \overline{s}, \overline{v}\right\}$ occurs. We determine a color change in the HSV colorspace if one of the following simplified conditions is not valid:

$$\left(\overline{h} - \Delta h * w\right) > (x,y)_h < \left(\overline{h} + \Delta h * w\right) \quad (1)$$

$$\left(\overline{s} - \Delta s * w\right) > (x,y)_s < \left(\overline{s} + \Delta s * w\right) \quad (2)$$

$$\left(\overline{v} - \Delta v * w\right) > (x,y)_v < \left(\overline{v} + \Delta v * w\right) \quad (3)$$

The Δ values in the conditions are user-defined thresholds that determine how greedy the flooding algorithm works. We choose a low delta threshold for the hue color channel (e.g. $\Delta h = 1$). The Δs and Δv values are defined higher, e.g. $\Delta s = \Delta v = 5$. Furthermore the color of the neighbor pixels are a quality criterion, too, and cells are not independent. If one cell detects a part of the vehicle silhouette, i.e. when a color change occurs, then the flooding proceeds more carefully in the neighbor cells by decreasing the weights. One result of the cell flooding algorithm can be seen in figure 5(c). The eliminated image pixel are black colored.

c) Pre-detection with Classifiers: After the clean-up step most of the image information in the $\mathrm{ROI}_{\mathrm{kalman}}$ should belong to the vehicle we want to detect. Therefore, the trained classifiers are used for each side. Thanks to the clean-up process and an algorithm allowing for parallel processing, the time effort is very low (approx. 1 ms). For a further reduction of effort, a combination of only HOG and LBP or HOG and Haar reached a similarly well detection rate. The classifier detection works in a multi-scale approach where the minimum and maximum object size depend on the last positive detection time. E.g. if the left side of the vehicle was detected in the last 50 ms cycle, then the minimum and maximum size should be close to the last detected

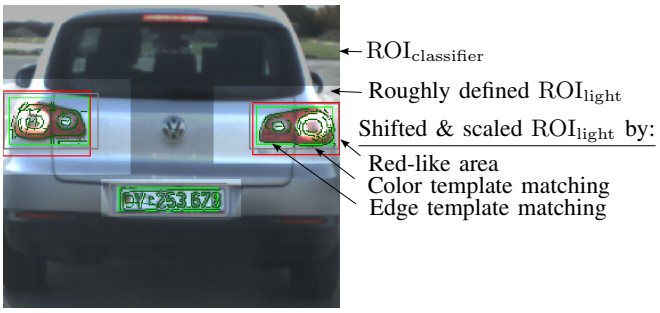


Fig. 7: Stepwise refinement of $\mathrm{ROI}_{\mathrm{classifier}}$.

size. A single classifier responds with one or more upright rectangles: $\mathrm{ROI}_{\mathrm{classifier}_{1\ldots n}}$. E.g. one classifier response is visible in figure 5(a) and labeled with a green-bordered rectangle. In case a classifier responds with many detections in a small image area, then grouping of the responses is applied. The fusion of classifier responses is executed with the following prioritization. HOG is more reliable than Haar which is more valid than LBP. Figure 6 visualizes the true positives of the three classifier types. The fusion of all three classifier responses is also visualized, which shows the effectiveness if classifiers are combined.

d) Stepwise Refinement of $\mathrm{ROI}_{\mathrm{classifier}}$*:* The classifiers respond with just one or more upright rectangles, but each rectangle can be a false positive and is not precise enough to accurately estimate the vehicle pose. The following steps deal with this.

First, the two rear lights are localized if they are visible (see fig. 7). This is realized in a top-down approach where an ROI is refined stepwise. Depending on the $\mathrm{ROI}_{\mathrm{classifier}}$ a roughly defined left rear light $\mathrm{ROI}_{\mathrm{light_l}}$ is placed in the left middle position of the $\mathrm{ROI}_{\mathrm{classifier}}$. In this $\mathrm{ROI}_{\mathrm{light_l}}$ red-like area is extracted. The current $\mathrm{ROI}_{\mathrm{light_l}}$ is shifted to the centroid of the red-like area with a size reduction of the $\mathrm{ROI}_{\mathrm{light_l}}$. The reduction depends on the size of the biggest connected red area. The right rear light $\mathrm{ROI}_{\mathrm{light_r}}$ is processed correspondingly. Furthermore we developed an active direction indicator detector that is used to correct the $\mathrm{ROI}_{\mathrm{light}}$s. An active direction indicator is determined by monitoring the blinking frequency of the amber color inside each $\mathrm{ROI}_{\mathrm{light}}$. If the frequency is constant over a time interval and has a frequency range of 1.5 ± 0.5 Hz, then an active direction indicator is detected.

Second, the license plate is localized. The initial search area for the $\mathrm{ROI}_{\mathrm{plate}}$ is estimated by the rear lights positions and the $\mathrm{ROI}_{\mathrm{classifier}}$ size. Consequently a rectangle with four 90° angles is searched which has a white foreground and dark texture in the form of small dark blobs. If the distance of the vehicle is no more than 10 m, the numbers and letters will be read from the license plate. This permits to distinguish vehicles with the same vehicle type.

Third, we do color and edge template matching for the rear lights and the license plate by minimizing a similarity score with the normalized cross correlation and the normalized correlation coefficient. In addition, we recommend minimizing

a dissimilarity score by estimating the normalized squared differences. Note that the edge template matching results in a set of 3D-to-2D point correspondences. These point matches between the 3D-template-model and 2D-image are later used to solve a Perspective-n-Point problem, respectively to estimate the relative vehicle pose (see chap. II-B.3).

At last, different symmetry checks are performed, such as the plate and the two rear lights must span a triangle where the two rear light angles must have similar values:

$$\angle(\boldsymbol{C}_{plate}, \boldsymbol{C}_{light_l}, \boldsymbol{C}_{light_r}) = \angle(\boldsymbol{C}_{light_l}, \boldsymbol{C}_{light_r}, \boldsymbol{C}_{plate}) \tag{4}$$

For these comparisons each centroid $\boldsymbol{C} = (x, y)$ of the ROIs $\in \{\mathrm{ROI}_{\mathrm{light_l}}, \mathrm{ROI}_{\mathrm{light_r}}, \mathrm{ROI}_{\mathrm{plate}}\}$ is appropriate. The special case of

$$\left|\boldsymbol{C}_{plate_x} - \boldsymbol{C}_{light_{l_x}}\right| = \left|\boldsymbol{C}_{plate_x} - \boldsymbol{C}_{light_{r_x}}\right| \tag{5}$$

$$\left|\boldsymbol{C}_{plate_y} - \boldsymbol{C}_{light_{l_y}}\right| = \left|\boldsymbol{C}_{plate_y} - \boldsymbol{C}_{light_{r_y}}\right| \tag{6}$$

allow an update of a template image.

3) Pose and Velocity Estimation: The pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}} = \{\mathrm{r}, \alpha, \mathrm{h}, \psi, \theta, \phi\}$ and object velocity v are estimated by the UKF.

a) Performing Measurement for UKF: The vehicle pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}}$ is calculated by recursively solving a Perspective-n-Point problem [16], [17] using the 4D-approach [18]. In more detail, we estimate the 3D translation and rotation between the bounding box center of the ego vehicle and the rear axis of the tracked vehicle (see fig. 2). This can be done by using intrinsic and extrinsic calibration data which describes the transformation between the image and ego coordinate system. The points from the Perspective-n-Point problem are the 3D-to-2D matching points from the edge template matching.

Note that a lot more features can be evaluated to improve the pose estimation (e.g. front lights, dark tires, windows, side doors, etc.).

b) UKF Innovation: An innovation of the UKF is applied if a vehicle detection was positively verified. The innovation is executed by using the current pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}}$ between the ego and the tracked vehicle. I.e. the UKF measurements are compared to their UKF prediction. Furthermore, an update of the UKF measurement covariances can be applied if an active direction indicator was detected. In this case, the measurement noise in the yaw state component ψ is increased because a turn is expected.

The filter innovation results in the final estimation of pose $\mathrm{P}_{\mathrm{ego}\to\mathrm{obj}}$ and object velocity v.

C. Combination of Model- and Template-based Tracking

Figure 8 demonstrates how the two tracking mechanisms interact. The template-based tracking supports the model-based tracking with a rough 3D vehicle pose and velocity. This is required in two situations: If the system starts the first time and if the model-based tracking loses the tracked vehicle. In these situations, a (re-) initialization of the particle filter is not possible. For this purpose, the template-based tracking resets all thousand particles of the particle filter with its current pose estimation. This results in a fast (re-) initialization.

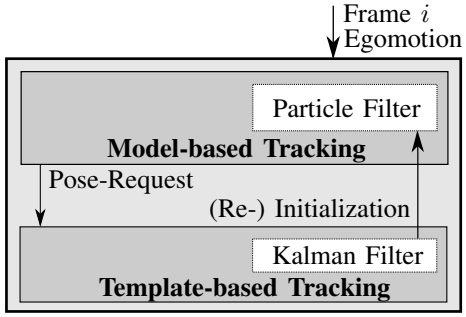


Fig. 8: Overview of the tracking architecture.

	r[m]	α[°]	h[m]	ψ[°]	θ[°]	ϕ[°]	$v[\frac{\mathrm{m}}{\mathrm{s}}]$
max_m	34.4	34.1	1.20	59.29	7.98	5.64	27.81
ϵ_m	0.47	0.32	0.06	1.50	1.16	0.45	0.38
ϵ_t	1.07	1.12	0.16	7.24	2.96	0.73	0.56

TABLE I: Tracking accuracy compared to ground truth: Evaluated values are the relative translation $\{\mathrm{r}, \alpha, \mathrm{h}\}$ and orientation $\{\psi, \theta, \phi\}$ in cylindrical coordinates and the object velocity v. The test route took about 15 minutes to drive. max_m denotes the maximal state values and ϵ_m the RMSE of the model-based tracking used for convoy driving. ϵ_t denotes the error of the template-based tracking used for (re-) initialization.

III. Experimental Results

The tracking system was evaluated on synthetic and real-world data, e.g. at the 2012 European Land-Robot Trial (ELROB). This section presents some practical results while driving autonomously with our robot car MuCAR-3 (Munich Cognitive Autonomous Robot Car 3$^{\mathrm{rd}}$ Generation [19]). We evaluated the tracking algorithms with ground truth data, generated by IMU coupled to RTK-DGPS. Our INS of type OxTS RT3003 reaches a precision of approximately $5\,\mathrm{cm}$ for ground truth data. The INS estimates the object velocity with an accuracy of $0.2\,\frac{\mathrm{m}}{\mathrm{s}}$. The transformation from the ego to the object position results in the ground truth pose $\mathrm{P}_{\mathrm{GT}_{\mathrm{ego}\to\mathrm{obj}}}$. Table I lists and compares the system (model-based) and template-based tracking accuracy which is denoted with the RMSE. The difference in the estimation of the yaw state component ψ is significantly high.

The tracking algorithms are tested on the MuCAR-3 on-board computer, an Intel Xeon L5640 Dual CPU Hexa Core. Besides tracking, other applications (e.g. obstacle detection, path planning) that are required for autonomous driving also operate on this system. The tracking applications are software triggered in a 50 ms cycle because the used CMOS camera is hardware triggered by this cycle. The camera has a resolution of 752 x 480 pixels and a lens of 8 mm focal length. It is mounted on a platform that allows a maximum yaw gaze of $\pm 45^{\circ}$ and a pitch rotation up to $\pm 12^{\circ}$. This allows extending the vision to keep objects of interest in the image center.

Running the applications in parallel, the template- and model-based algorithms need 5-20 ms and 25-35 ms, respec-

(a) (b) (c) (d)

Fig. 9: Model-based tracking: (a) Light changes. (b) Snow. (c) Turn. (d) Partial occlusion.

tively, so that the overall cycle time of 50 ms is not exceeded. We compared the ground truth data with the template-based tracking and determined a false positive rate of zero. This can be traced back to the cell flooding method and the cascaded ROI proceeding, which removed unnecessary image information.

We tested the tracking algorithms while driving over hundreds of kilometers in an autonomous convoy. In these scenarios a leading vehicle drives on urban and unstructured environments and the autonomous vehicle MuCAR-3 handled the throttle, break and steering. Only the vision-based tracking algorithms were used for continuous estimation of the relative pose $\mathrm{P}_{\mathrm{ego}\rightarrow\mathrm{obj}}$. Communication between these vehicles was exclusively used for generating the ground truth data. The algorithm was tested with different real world vehicles, e.g. a VW Golf 4, VW Tiguan or an Audi A4. The system proved its robustness while driving in different weather conditions, e.g. by rain, snow, sunshine, dust or fog. Also, it is invariant against partial occlusion (e.g. bushes, high grass or windshield wiper). The system runs in real-time in a 50 ms cycle while driving autonomously. Thereby the 20 fps and the high pose estimation accuracy allows speeds of up to $28\,\frac{\mathrm{m}}{\mathrm{s}}$. Furthermore, some visual impressions are shown in figure 9 and on our Youtube channel: http://youtu.be/ObvW0Kx2IUc

IV. Conclusions

This paper presented a monocular vehicle tracking system for driving autonomously in urban and unstructured environments. The fundamental idea is to fuse the advantages of model- and template-based tracking. The model-based method can estimate the 3D vehicle pose and the velocity with a high accuracy. It uses a particle filter that needs a vehicle hypothesis at system start and in case of a tracking loss. For this purpose, the particle filter is (re-) initialized by the template-based tracking application. This can detect the tracked leading vehicle very quickly. As a result an autonomous convoy can drive without tracking interruptions. The results show that poorly visible vehicles can be tracked in different weather conditions and in real-time.

Acknowledgment

The authors gratefully acknowledge funding by the Federal Office of Bundeswehr Equipment, Information Technology and In-Service Support (BAAINBw).

References

[1] A. Petrovskaya and S. Thrun, "Model Based Vehicle Detection and Tracking for Autonomous Urban Driving," *Autonomous Robots*, vol. 26, no. 2-3, pp. 123–139, 2009.

[2] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base*, ser. Springer Tracts in Advanced Robotics, vol. 56. Springer, 2009.

[3] A. Mueller, M. Manz, M. Himmelsbach, and H.-J. Wuensche, "A Model-Based Object Following System," 2009.

[4] M. Manz, T. Luettel, F. von Hundelshausen, and H.-J. Wuensche, "Monocular Model-Based 3D Vehicle Tracking for Autonomous Vehicles in Unstructured Environment," 2011.

[5] J. Choi, "Realtime On-Road Vehicle Detection with Optical Flows and Haar-like feature detector," University of Illinois at Urbana-Champaign, Computer Science Research and Tech Reports, 2006.

[6] P. Viola and M. Jones, "Robust Real-time Object Detection," 2001.

[7] S. Sivaraman and M. M. Trivedi, "Real-Time Vehicle Detection Using Parts at Intersections," 2012.

[8] ——, "Combining Monocular and Stereo-Vision for Real-Time Vehicle Ranging and Tracking on Multilane Highways," 2011.

[9] C. D. Pantilie and S. Nedevschi, "Real-time Obstacle Detection in Complex Scenarios Using Dense Stereo Vision and Optical Flow," 2010, pp. 439–444.

[10] A. Barth and U. Franke, "Tracking Oncoming and Turning Vehicles at Intersections," 2010, pp. 861–868.

[11] J. Lou, T. Tan, W. Hu, H. Yang, and S. J. Maybank, "3-D Model-Based Vehicle Tracking," *IEEE Transactions on Image Processing*, vol. 14, no. 10, pp. 1561–1569, 2005.

[12] C. Fries, *Bildbasierte Objekterkennung und Lageschätzung für Serviceroboter*. AVM - Akademische Verlagsgemeinschaft München, 2011.

[13] T. Ojala, M. Pietikainen, and D. Harwood, "Performance Evaluation of Texture Measures with Classification Based on Kullback Discrimination of Distributions," in *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 1 – Conference A: Computer Vision & Image Processing*, 1994, pp. 582–585.

[14] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," 2005, pp. 886–893.

[15] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[16] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," vol. 8, no. 2, pp. 155–166, 2009.

[18] E. D. Dickmanns, *Dynamic Vision for Perception and Control of Motion*. Springer Verlag, 2007.

[19] M. Himmelsbach, F. von Hundelshausen, T. Luettel, M. Manz, A. Mueller, S. Schneider, and H.-J. Wuensche, "Team MuCAR-3 at C-ELROB 2009," in *Proceedings of 1st Workshop on Field Robotics, Civilian European Land Robot Trial*, 2009.