Machine Vision and Applications (2015) 26:369–385
DOI 10.1007/s00138-015-0668-z

ORIGINAL PAPER

# Backward-Simulation Particle Smoother with a hybrid state for 3D vehicle trajectory, class and dimension simultaneous estimation

**Andrea Romanoni · Domenico G. Sorrenti · Matteo Matteucci**

Received: 27 July 2014 / Revised: 10 November 2014 / Accepted: 7 February 2015 / Published online: 25 February 2015


**Abstract** The estimation of the 3D trajectory, the class and the dimensions of a vehicle represents three relevant tasks for traffic monitoring. They are usually performed by separate sub-systems and only few existing algorithms cope with the three tasks at the same time. However, if these tasks are integrated, the trajectory estimation enforces the classification with temporal consistency, and at the same time, the estimation of the vehicle class and dimensions can be used to increase the trajectory estimate accuracy. In this work, we propose an algorithm to estimate the 3D trajectory, the class and the dimensions of vehicles simultaneously by means of a Backward-Simulation Particle Smoother whose state contains both continuous (vehicle pose and dimensions), and discrete (vehicle class) quantities. To integrate the class estimate in the Particle Smoother we model the class prediction as a Markov Chain. We performed experimental tests on both simulated and real datasets; they show that the pose and dimension estimation reaches centimeter-accuracy and the classification accuracy is higher than 95 %.

**Keywords** 3D trajectory reconstruction · Vehicle classification · Particle filtering · Hybrid state representation

A. Romanoni (✉) · M. Matteucci
Politecnico di Milano, DEIB, Via Ponzio 34/5, 20133 Milano, Italy
e-mail: andrea.romanoni@polimi.it

M. Matteucci
e-mail: matteo.matteucci@polimi.it

D. G. Sorrenti
Universitá degli Studi di Milano-Bicocca, DISCo Building U14, Viale Sarca 336, 20126 Milano, Italy
e-mail: domenico.sorrenti@disco.unimib.it

## 1 Introduction

Three very common tasks in traffic monitoring are the estimations of the 3D trajectory, the class and the dimensions of a vehicle. The vehicle trajectory estimation is usually called *tracking*, and it is frequently performed by means of a Bayesian Filter [14–16,29]. The vehicle classification algorithms usually compare the vehicle image to a set of models, each representing a different vehicle class. Some classification algorithms also accomplish the dimension estimation by fitting an adaptive model on the image data.

Few contributions focus on the integration of these tasks, although a single estimator capable of estimating the 3D trajectory, the class and the vehicle dimensions simultaneously would be of great interest for traffic flow monitoring and vehicle counting. The interaction among these pipelines could also lead to more effective results with respect to those obtained by the separate sub-systems. Indeed, trajectory estimation enforces the classification with temporal consistency, while vehicle class and dimension estimation increases the model-based trajectory estimation accuracy. Moreover, the dimension estimation with an adaptive model makes the classification more robust, and in turn, the trajectory more accurate, since a model that fits well the vehicle image has a reduced bias on the pose estimation.

With the aim of more accurate and reliable performance in traffic monitoring, in this paper, we propose two main contributions to accurately accomplish the aforementioned three estimation tasks simultaneously. First, we design a tracking system with a hybrid state whose peculiarity is the simultaneous representation of continuous (pose and dimensions) and discrete (class) components. To take into account the twofold nature of this state representation, we model the transition between classes as a Markov Chain. The second contribution is the estimation by means of a Backward-Simulation Parti-

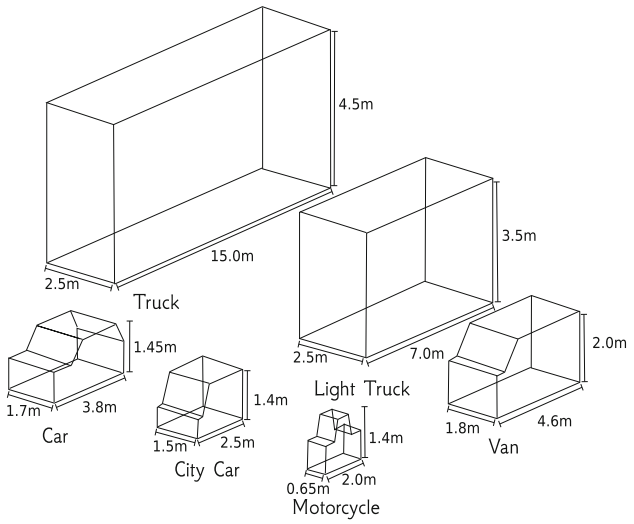



**Fig. 1** The six different class models. Their dimensions change during in the estimation process; here, we report the initial values

cle Smoother that reaches more accurate results with respect to a Particle Filter and handles suitably well the hybrid nature of the state.

The proposed system significantly improves the capabilities of the systems presented in [17,26,28]. In those contributions we developed two Monte Carlo estimation systems to reconstruct the trajectory of a vehicle and to estimate its class independently: one of the two systems implements a Viterbi-based approach, while the other is a Particle Smoother. In [28], we showed the reasons why the Particle approach overcomes the Viterbi-based algorithm. The current paper presents a significantly different approach; indeed, we propose a single Monte Carlo estimator to obtain a joint estimation of the trajectory, the class and the dimensions of tracked vehicles; conversely, in [17,26], we employed a set of independent estimators, one for each fixed-sized model, then we choose the most likely one. Therefore, in those approaches, the class and the dimensions were not estimated jointly by the Monte Carlo estimators (neither the Viterbi-based nor the Particle Smoother). Moreover, in the previous works we adopted a simple parallelepiped model, while in the proposed system we adopt more realistic models for each vehicle class (see Fig. 1), and now, we jointly estimate its dimensions.

In Sect. 2 we overview existing systems that perform 3D tracking and classification. In Sect. 3 we outline the presented algorithm and in Sect. 4 we explain the design of our Backward-Simulation Particle Smoother with the hybrid state. Finally, in Sect. 5, we show experimental results on synthetic and real datasets.

## 2 Related works

Vehicle trajectory estimation is usually named as *tracking*: 2D tracking algorithms estimate the trajectory of an object in the image plane, while 3D ones compute the trajectory in the world coordinates. To simplify the 3D tracking task, the existing algorithms assume camera calibration to be known (see [8]). In a vehicle tracking scenario, they also assume the Ground Plane Constraint: a vehicle always lies on the road plane [13], so its unknown poses have 3 degrees of freedom instead of 6. In recent tracking systems, 3D tracking is the most widespread approach when dealing with vehicles (see [5]).

Both 2D and 3D tracking are usually based on Kalman Filter (KF), or on the Extended Kalman Filter (EKF) [2]. A KF estimates iteratively the evolution of the vehicle pose; beginning with a convenient initialization, it predicts the pose at time $t$ from the pose at time $t-1$, according to a motion model (prediction step), and then, it estimates the pose at time $t$ by adjusting the predicted state according to its likelihood with respect to a measure of the observed vehicle (update step).

The likelihood calculation is a challenging step for 3D trackers because of the comparison between a 2D measure on the image plane and the 3D predicted pose. To accomplish this comparison, most 3D tracking algorithms project a 3D model of the vehicle on the image plane, and then they compare this projection with the vehicle measurement, which lays on the image plane too. Some of them, e.g., [11,15,32], compute the likelihood as the distance of all the line segments of the projected model from the edges of the vehicle image, i.e., the more a segment is close to an edge, the more the predicted pose is likely. An analogous approach computes the gradient of the image and compares the projected model with it: the more a segment is perpendicular to high gradient direction, the more the predicted pose is likely [13]. These two approaches are named *edge based*, and in the update step, they minimize the distance between the model segments (in the former case), or their normals (in latter case), and the image edges, or the gradients. These approaches are very robust to brightness changes, but they are affected by local minima [13]: if the model is not sufficiently close to the right position, the minimization procedure could estimate a wrong vehicle position where the model segments are close to the wrong edges (or gradients). Therefore, edge-based tracking requires that the (estimated) pose of the model before the minimization process is very close to the real tracked vehicle.

On the other hand, the so-called *region-based* approaches [6,18,31] compute the likelihood as the overlap of the vehicle blob (connected region in the image, which corresponds in our case to the vehicle) with the convex hull of the 3D model projection on the image plane. The authors in [3] propose a sophisticated approach that compares the vehicle silhouette to an appearance-based model. In [3,6,18] the tracking is performed with a Kalman Filter, while in [31] the authors propose a Markov Chain Monte Carlo (MCMC) method to

collect hypotheses and put them together via the Viterbi algorithm to form the vehicles trajectories. This approach relies on blob estimation with a background subtraction algorithm (see [20,25,27]); therefore, it is more sensitive to brightness changes with respect to edge-based ones. On the other hand, the comparison via overlap is more robust to local minima issues and the vehicle model can be a coarse estimation of the tracked one (see [13]).

Few of the region-based algorithms, we have listed integrate the computation of the 3D trajectory with vehicle classification. For instance in [6,11,19], the authors use a set of models representing different vehicle classes and estimate the vehicle class as the most likely model. This is a very straightforward approach, but if none of the models fits well the tracked vehicle, it may lead to misclassification. To make the classification more robust, the authors in [12] fit a box model with variable dimensions to the vehicle blob, then they infer the vehicle class. This work presents good dimension estimation results, but does not provide a unified framework to estimate directly both the dimensions and the class.

Other authors focus their attention on the vehicle dimension estimation process. The algorithm in [24] estimates the dimension by means of a Generalized Deformable Model (box shaped) parametrized with respect to the vanishing point of the images; the authors focus on the problem of vehicle inter-occlusion, and their system neglects the tracking and dimension estimation problems. The algorithm proposed by [10] provides a unified framework to estimate both the 3D vehicle trajectory and the vehicle dimensions by means of a deformable 3D box model, but its authors do not face the class estimation problem. In a more recent system proposed in [34], the authors fit a complex 12-segment model to the vehicle edges so as to properly handle very different vehicle shapes; however, they do not estimate the vehicle 3D trajectory.

Our overview clarifies that an integrated approach to estimate the vehicle 3D trajectory, its class and its dimensions simultaneously does not currently exist, at least to the best of our knowledge. The literature contribution that is closer to our objectives is the ones in [4,6], which perform 3D tracking via Kalman Filter (KF) and vehicle classification with a region-based approach; however, this proposals do not estimate the vehicle dimensions. In [4,6], the 3D pose likelihood computation is based on the overlap score between the model projection and the blob region, but the authors could not directly integrate this likelihood in the KF update step; to overcome this limitation, they propose to extract some hypotheses around the KF-predicted 3D position and to choose the most likely one as the new vehicle measurement. By doing this, they discard hypotheses that could provide a better estimate of the vehicle position, although resulting in a lower likelihood because of occlusions, perspective distortions, and/or noise.

In this paper, instead, we propose a Backward-Simulation Particle Smoother which natively deals with hypotheses weighting; moreover, we adopt a hybrid filter state to estimate, at the same time, the 3D pose, the class and the dimensions of a vehicle.

## 3 The proposed approach

As in [6] we choose a region-based approach to estimate the vehicle 3D trajectory rather than an edge-based one. Indeed, a region-based approach does not need an accurate model of the tracked vehicle, and it is well known for being more robust with respect to edge-based ones to the local minima issue during the pose estimation process [13]. Finally, the vehicle classification results to be more robust with respect to the edge-based approach; indeed, it is quite easy to distinguish if a blob comes from a car, a truck, or a motorcycle. An empirical evidence of this effectiveness is that most classification algorithms adopt a region-based approach.

As mentioned above, Kalman Filter tracking has a serious limitation when we adopt a region-based approach: in the measurement integration step, a simple overlap score between the image blob and the vehicle 3D model projected on the image plane cannot be used directly. We overcome this issue by means of a Monte Carlo approach, in particular adopting the so-called *Backward-Simulation Particle Smoother* (BSPS): this method natively weights hypotheses with a likelihood score, that, in our case, can be derived from the overlap between the vehicle blob and the model back-projection on the image plane. Moreover, BSPS can estimate multi-modal distributions, and although we use a BSPS for each vehicle, we like the perspective of being able to estimate the 3D pose, class and dimension of more than one vehicle at a time with a single estimator, similarly to what has been proposed in [31].

Since we aim at estimating also the actual dimensions of each vehicle, we cannot rely on a fixed-size vehicle model for all the classes. Instead, we use a set of models whose dimensions can change during the estimation process. In Fig. 1 we show the models we use and we also report the default, i.e., initial dimensions for each of them. We adopt six classes: Car, Truck, Light Truck, Van, City Car and Motorcycle, being the difference between Car and City Car related to dimensions and shape, see Fig. 1. This choice is induced by the kind of vehicles that traverse the roundabout in our region, and they are sufficient to represent the majority of the vehicles in our experiments. However, if needed, we could use other models very easily, for instance bike, sedan, hatchback or SUV classes which might be more appropriate for different application scenarios.

Our current implementation of the proposed algorithm relies on the outcome of the 2D data association performed

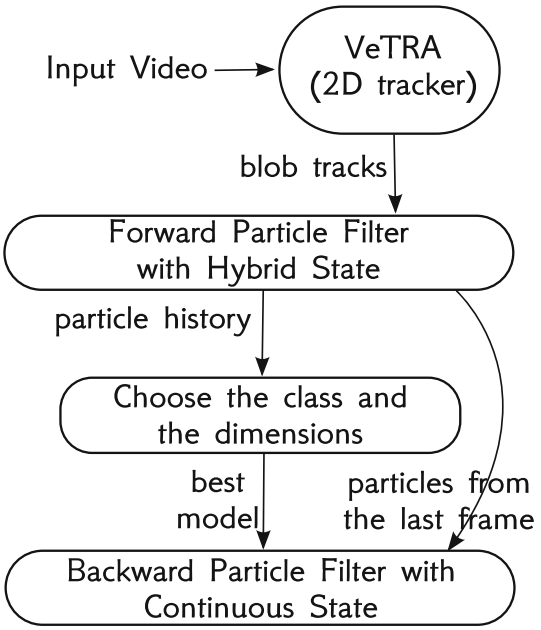



**Fig. 2** Schema of the proposed algorithm

by a 2D tracking algorithm, named VeTRA (see [21,22]); therefore, we already know which blob is associated to which vehicle. As a consequence, our algorithm focuses on the state (3D pose, class and dimensions) estimation process and does not accomplish the full tracking task, as it does not perform the data association in 3D. In Fig. 2 we show a sketch of the current implementation of the proposed algorithm and its integration with VeTRA [22].

## 4 Backward-Simulation Particle Smoothing with a hybrid state

A Backward-Simulation Particle Smoother is an estimator that applies a first Particle Filter iteration to follow the evolution of the state, i.e., in our case, the trajectory of the vehicle, and then, starting from the last set of particles it performs a backward filtering with a second Particle Filter. For a detailed and formal explanation see [7,30, p. 167]. In the following, we explain how we design the Particle Filter that performs the forward iteration, named Forward Particle Filter (FPF) and in the end, we describe the Backward Particle Filter (BPF).

A Particle Filter (PF) is a well-known Bayesian estimator that asymptotically reaches the Maximum A-Posteriori estimate of the distribution of a stochastic variable $s_t$ (for an introduction to PF see [1]). A PF represents the state probability distribution by means of a set of samples from the state probability distribution. Starting from a convenient initialization (see Sect. 4.3), the PF estimation entails a prediction step (or state transition), which generates a set of state proposal samples from the previous state samples, and an update step, which usually encompasses a resampling stage, based on the likelihood of the samples.

Our algorithm implements a PF with a hybrid state for each vehicle detected by the VeTRA 2D tracker. At time $t$, the vehicle state is the following:

$$s_t = \begin{bmatrix} [x_t \;\; y_t \;\; \theta_t] \\ [l_t \;\; h_t \;\; w_t] \\ c_t \end{bmatrix} = \begin{bmatrix} p_t \\ d_t \\ c_t \end{bmatrix}; \tag{1}$$

it is composed by the vehicle pose $p_t$ (where $(x_t, y_t)$ are the coordinates of the vehicle centroid on the road plane and $\theta_t$ is the vehicle orientation), its dimensions $d_t$ (where $l_t$ is the length, $h_t$ the height, and $w_t$ the width) and its class $c_t$. The first six state variables are continuous, while the vehicle class $c$ is a discrete quantity.

### 4.1 The prediction step

A big challenge to design a PF with a hybrid state is to define a convenient state transition function that accomplishes the prediction step of the PF. We model the transition between two consecutive states with a motion model $f$ for the vehicle centroid 3D coordinates, and a transition function $g$ for the class and the dimensions:

$$s_t = \begin{bmatrix} p_t = f(p_{t-1}) \\ \begin{bmatrix} d_t \\ c_t \end{bmatrix} = g(d_{t-1}, c_{t-1}) \end{bmatrix}. \tag{2}$$

This subdivision in the transition function keeps the prediction of the pose $p_t$ independent from the class and dimensions: we predict a new pose, by applying the classical constant velocity motion model, as in most tracking algorithms, while in the following section we explain how we deal with the prediction of dimensions and class (i.e., the discrete component of the state). Although a possibly more accurate model for predicting the vehicle pose, taking into account its class and dimensions, could be conceived, this independence assumption is not in contrast with the idea of jointly estimating the trajectory, the class and the dimensions: indeed only the prediction step is affected, while the resampling stage estimates jointly the three state subspaces. In principle, the proposed framework is able to deal also with a motion model where the 3D pose prediction depends on class and dimensions (e.g., it may change the orientation variances together with the dimensions), but this has not been implemented in the current version of the algorithm and thus we mention it here only for completeness. In the latter case, i.e., prediction model that takes into account class and dimensions, $p_t = f(p_{t-1})$, would become $p_t = f(p_{t-1}, c_{t-1}, d_{t-1})$.

One contribution of this paper is the design of a convenient function $g$ that manages the state transition of both the discrete quantity $c$ and the related dimensions of the vehicle. The transition from class $c_{t-1}$ to $c_t$ takes into account that

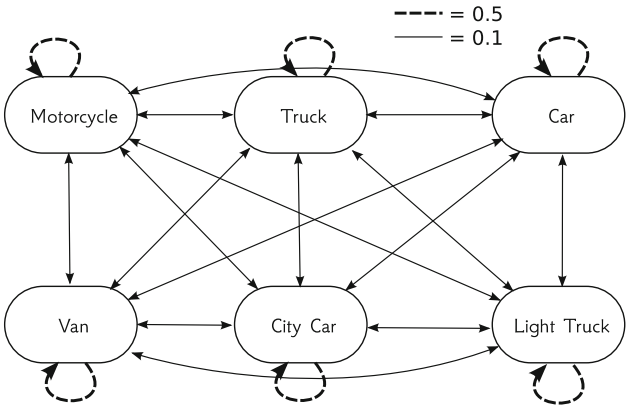



**Fig. 3** The Markov Chain that models the class transition

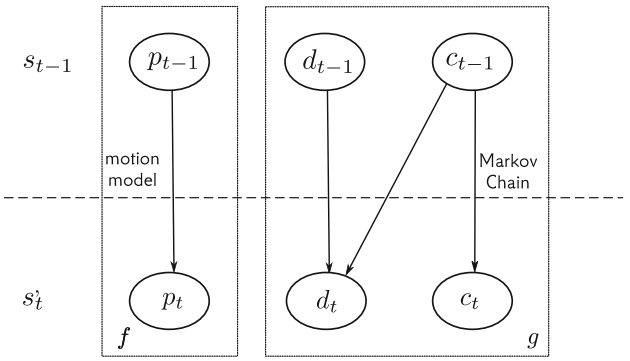



**Fig. 4** Schema of the simple approach to predict a new state (i.e., particle). First the new pose and class are computed, then a new value for the dimension is extracted, relying on the class choice

**Table 1** Transition matrix $T$

| | Car | Truck | Light truck | Van | Motorcycle | City car |
|---|---|---|---|---|---|---|
| Car | 0.5 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| Truck | 0.1 | 0.5 | 0.1 | 0.1 | 0.1 | 0.1 |
| Light truck | 0.1 | 0.1 | 0.5 | 0.1 | 0.1 | 0.1 |
| Van | 0.1 | 0.1 | 0.1 | 0.5 | 0.1 | 0.1 |
| Motorcycle | 0.1 | 0.1 | 0.1 | 0.1 | 0.5 | 0.1 |
| City car | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.5 |

the true class of a vehicle does not change over time. Nevertheless, our belief about it can change, due to ambiguities in the model projection and vehicle measurement in the image; for instance, a car may be misclassified as a light trucks or a van, due to the perspective. We assume that the transition of the belief of the class $c_{t-1}$ to a class $c_t$ only depends on the previous class belief $c_{t-1}$; we model this transition as the Markov Chain as in Fig. 3. We illustrate the transition dependencies in the graphical model in Fig. 4.

We model the probability of a class change with a transition matrix $T$ represented in Table 1. We choose the probability of the class change equal to 0.5, and we spread it evenly among all other classes (i.e., if $c_{t-1} = \text{Car}$, then $p(c_t = \text{Car}) = 0.5$ and $p(c_t \neq \text{Car}) = 0.5 = \sum_{c_t \neq \text{Car}} 0.1$).

A more complex choices of $T$ would be possible: $T$ could be estimated, for instance, by computing the confusion matrix from real datasets. Nevertheless, we choose this $T$ context-independent not to bias the results with respect to a specific dataset, and above all, we found that the $T$ in Table 1 leads to very good results.

The value of $c_t$ affects in turn the definition of the dimensions $d_t = [l_t, h_t, w_t]$. If the class $c_t$ differs from $c_{t-1}$, then $d_t$ is set to the reference dimensions of the new class model; for each class, we fixed the average reference dimensions depicted in Fig. 1. Instead, if the class $c_t$ matches $c_{t-1}$, then we sample $d_t$ from the following distribution:

$$d_t \sim \mathcal{N}\left(d_{t-1}, \begin{bmatrix} \sigma^2_{l^c} & 0 & 0 \\ 0 & \sigma^2_{h^c} & 0 \\ 0 & 0 & \sigma^2_{w^c} \end{bmatrix}\right) \tag{3}$$

where the values of $\sigma^2_{l^c}$, $\sigma^2_{h^c}$ and $\sigma^2_{w^c}$ represent the variances of dimensions of each vehicle class. We set these variances according to the heuristic: "larger dimensions are associated to larger variances", and zeroed, for simplicity, the non-diagonal elements; thus, these values depend on the vehicle class. For instance, the variance of a truck length is larger than the variance of a motorcycle or a car length (see Sect. 5 for further consideration about these values).

Notice that Eq. (3), in principle, would allow a vehicle of a class to take a size that is totally unrealistic, as no explicit mechanism for limiting the dimensions is imposed. Such possibility is neglected, because of two reasons. First, the standard deviations in Eq. (3) refer to intra-class variations, so, for instance, it is extremely unlikely to obtain a sample representing a car with the size of a truck. Second, the resampling step would prevent this drift of the dimensions

What is outlined so far hides a critical issue. If $T$ is the transition matrix, $n$ steps of the Markov Chain, without any observation, lead to an $n$-step transition probability represented by the transition matrix $T^n$, which usually converges to a value $\hat{T}$. In Table 2 we show the transition matrix $T$ after six steps: the transition probabilities have almost the

**Table 2** Transition matrix $T$ after 6 step, i.e., $T^6$

| | Car | Truck | Light truck | Van | Motorcycle | City car |
|---|---|---|---|---|---|---|
| Car | 0.171 | 0.166 | 0.166 | 0.166 | 0.166 | 0.166 |
| Truck | 0.166 | 0.170 | 0.166 | 0.166 | 0.166 | 0.166 |
| Light truck | 0.166 | 0.166 | 0.170 | 0.166 | 0.166 | 0.166 |
| Van | 0.166 | 0.166 | 0.166 | 0.170 | 0.166 | 0.166 |
| Motorcycle | 0.166 | 0.166 | 0.166 | 0.166 | 0.170 | 0.166 |
| City car | 0.166 | 0.166 | 0.166 | 0.166 | 0.166 | 0.170 |

same values, which means leveling the probability of class change. Even if we rely on the resampling stage of the PF to have the class converge to a likely value, we have found experimentally that enforcing this resampling step is needed. From Eq. (2) we can write the prior probability of $s_t$ as:

$$p(s_t|s_{t-1}) = p(p_t|p_{t-1})p(d_t, c_t|d_{t-1}, c_{t-1}). \quad (4)$$

From the state $s_{t-1}$ we propose a new particle in the following way. We sample $p_t$ from the probability $p(p_t|p_{t-1})$ according to the motion model as previously presented. Then, we adopt an MCMC-based strategy [23] to sample $d_t$ and $c_t$ from $p(d_t, c_t|d_{t-1}, c_{t-1})$ .

We extract $d'_t$ and $c'_t$ according to the transition of the Markov Chain and the dimension estimation of Eq. (3); we define $s_t^1 = [p_t, d'_t, c'_t]^T$ and $s_t^2 = [p_t, d_{t-1}, c_{t-1}]^T$; we extract a random number $\beta$ in [0, 1). Finally, we define

$$P(d_{t-1}, c_{t-1}, d'_t, c'_t) = \frac{P(d_{t-1}, c_{t-1})}{P(d'_t, c'_t) + P(d_{t-1}, c_{t-1})} \quad (5)$$

and the new state $s_t$ becomes

$$s_t = \begin{cases} [p_t, d'_t, c'_t]^T, & \text{if } \beta > P(d_{t-1}, c_{t-1}, d'_t, c'_t) \\ [p_t, d_{t-1}, c_{t-1}]^T, & \text{if } \beta < P(d_{t-1}, c_{t-1}, d'_t, c'_t) \end{cases}. \quad (6)$$

### 4.2 The update step

After the prediction step, the PF compares each predicted state (i.e., particle) with the current measurement of the vehicle, i.e., the blob on the image extracted by VeTRA 2D. In this update step, the PF weights each predicted particle according to its likelihood.

To understand how this likelihood is computed, let $z_t$ be the measure, i.e., the blobs extracted by the VeTRA 2D tracking system, and let

$$\hat{s}_t = [x_t, y_t, \theta_t, l_t, h_t, w_t, c_t]^T$$

be the current predicted state particle. Likelihood calculation follows these four steps:

1. Project on the image plane a model of class $c_t$ with dimensions $l_t, h_t, w_t$, which is located on the road plane in $(x_t, y_t)$ and with orientation $\theta_t$ (e.g., the red polygon in Fig. 5).
2. Compute overlap area:
   - Compute the blob area $A_{\text{blob}}$ (the yellow + green region in Fig. 5);
   - Compute the visible model projection area $A_{mv}$ (the blue + green region in Fig. 5);
   - Compute the overlap area between the blob and the model projection $A_{\text{overlap}}$ (Fig. 6).

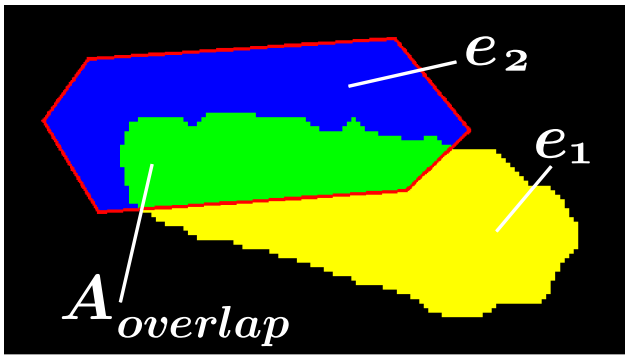



**Fig. 5** Regions involved in the likelihood calculation: $e_1$ captures how much of the measured blob is not captured by the predicted projected model, $e_2$ captures how much of the predicted model is not reflected by the measured blob

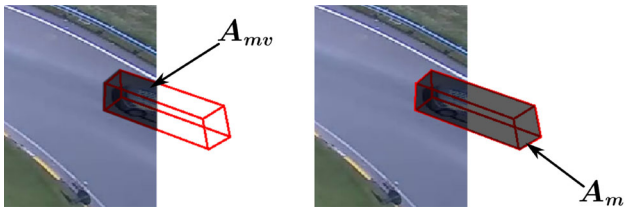



**Fig. 6** Difference between $A_{mv}$ and $A_m$ (*shaded area*)

3. Compute the two errors $e_1$ and $e_2$ (Fig. 5), as

$$e_1 = A_{\text{blob}} - A_{\text{overlap}} \quad (7)$$

$$e_2 = A_{mv} - A_{\text{overlap}} \quad (8)$$

4. Define the score term:

$$e = \frac{\lambda_1 e_1 + \lambda_2 e_2}{A_{\text{blob}}}, \quad (9)$$

where $\lambda_1$ and $\lambda_2$ are the weights for $e_1$ and $e_2$ such that $\lambda_1 + \lambda_2 = 1$ (we set simply $\lambda_1 = \lambda_2 = 0.5$).

Likelihood is then defined as $p(z_t|s_t) = 1 - e$. In some cases it may happen that $1 - e < 0$, i.e., $A_{\text{blob}} < \lambda_1 e_1 + \lambda_2 e_2$; this essentially corresponds to two possible configurations: (a) the blob is negligible by itself, due to background subtraction failure, or (b) the model under evaluation is very big with respect to the blob. In both cases, we choose to assign a likelihood equal to 0. In the former situation, i.e., case (a), all the particles have likelihood equal to 0 and the estimator applies only the prediction step; this is the expected behavior, since the observation for the current state is almost missing. In the latter situation, i.e., case (b), we force the likelihood to 0 since the current state model deeply differs from the observed blob.

**Algorithm 1**: Resampling algorithm

**Input**: $\{\hat{s}_t^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$ states-particles to resample and their weights (i.e. their likelihood)

**Output**: $\{\bar{s}_t^j\}_{j=1}^{N_{\text{samples}}}$ resample states-particles

Create the Cumulative distribution function (CDF):
Initialize the CDF: $c_1 = 0$;
**for** $i = 2$ ***to*** $N_{samples}$ **do**
  $c_i = c_{i-1} + w_t^i$;
Start from the beginning of the CDF: $i = 1$;
Sample the first point: $u_1 \sim \mathbb{U}[0, N_s^{-1}]$;
**for** $j = 1$ ***to*** $N_{samples}$ **do**
  Scan the CDF: $u_j = u_1 + N_{\text{samples}}^{-1}(j-1)$;
  **while** $u_j > c_i$ **do**
    $i = i + 1$;
  $\bar{s}_t^j = \hat{s}_t^i$;

The overall likelihood reads as

$$p(z_t|s_t) = \begin{cases} 1-e, & \text{if } 1-e > 0 \\ 0, & \text{if } 1-e < 0 \end{cases}. \tag{10}$$

Only the visible part of the model projection concurs to compute the likelihood score. So $A_{mv}$ is not the entire area of the projected model, but the projected area cropped to the current camera image. Figure 6 shows the difference between $A_{mv}$ and the area of the entire projected model, $A_m$.

Our algorithm, as described so far, works on a single-camera setting, but we can easily extend it to a multi-camera scenario by enhancing the likelihood computation as follows. Let $z_t^j$ be the blob of one vehicle perceived by the $j$th camera ($C_j$), and $n_c$ the number of cameras. Let $p_j(z_t^j|s_t)$ be the likelihood calculated for each camera $j$ according to Equation (10); this value is weighted according to how much of the vehicle is perceived by $C_j$, i.e., according to the weight $w_j = \frac{A_{mv}^j}{A_m^j}$. The overall likelihood becomes a weighted average of the likelihood for each camera:

$$p_{\text{tot}}(z_t|s_t) = \frac{\sum_{j=1}^{n_c} w_j p_j(z_t^j|s_t)}{\sum_{j=1}^{n_c} w_j} \tag{11}$$

The weight $w_j = \frac{A_{mv}^j}{A_m^j}$ is designed to take into account different situations. Therefore, if $N$ cameras perceive the whole vehicle, we weight each corresponding likelihood $\frac{1}{N}$; if one camera does not see the vehicle, its contribution to the likelihood computation is 0; if one camera observes the vehicle only partially, its contribution to the overall likelihood computation is proportional to the observed vehicle percentage.

Each sample likelihood becomes the sample's weight; then, we update our Particle Filter by performing the classical resampling stage [1], which steers the particles according to the information carried by the new measurement, encoded in the weights' distribution. In the Algorithm 1, we sketch the classical resampling algorithm we have used.

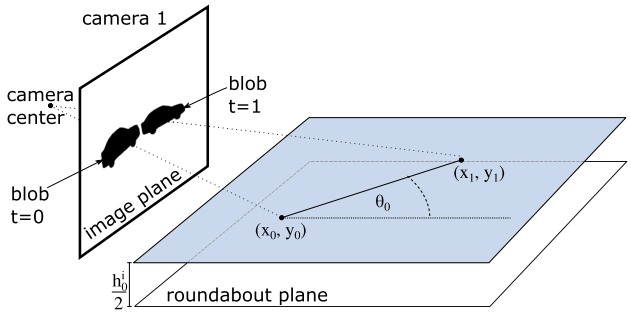



**Fig. 7** Computation of initial value of $\theta$

### 4.3 The initialization

Up to now we assumed the PF to be somehow already initialized, but to achieve a proper functioning, we have to provide a convenient set of sample to bootstrap our PF and thus provide such initialization. To this extent, assume we have $N_{\text{samples}}$ samples in the PF (currently in our implementation $N_{\text{samples}} = 1200$); for each sample $i$, we choose uniformly a random class and we set the value $d$ of this sample to the reference dimensions of the sampled class (see Fig. 1). Then, let $t = 0$ the frame where the vehicle first completes blob is visible, i.e., when the vehicle is completely inside the camera field of view, thus sufficiently far from the image margin. Project the blob at frame 0 and the blob at frame 1 on the plane parallel to the ground, passing through the center of the vehicle model under evaluation, i.e., with height equals to $\frac{h_0^i}{2}$ when evaluating the $i$th particles; let the resulting points be $(x_0, y_0)$ and $(x_1, y_1)$, respectively, and $\theta_t$ defined as the orientation of the vector from $(x_0, y_0)$ to $(x_1, y_1)$ (see Fig. 7).

We now extract one sample from a trivariate Gaussian distribution having mean $(x_0, y_0, \theta_0)$ and a diagonal covariance structure reflecting the independence of the three components (we set the diagonal elements to $\sigma_x^2 = 0.5$ m, $\sigma_y^2 = 0.5$ m and $\sigma_\theta^2 = \frac{\pi}{6} rad$). Independence is a reasonable assumption to simplify the tuning of the system, but a non-diagonal covariance matrix could be used if available. Each of these samples represents a vehicle state, i.e., vehicle position and orientation on the roundabout plane at time 0. A simplified example of this process is shown in Fig. 8 and in Algorithm 2 we summarize this process.

### 4.4 The choice of the model

After each vehicle filter reaches the end of the trajectory, i.e., the vehicle disappears from the image, we select the class and the dimensions of the model.

**Algorithm 2**: Initialization of the Forward Particle Filter

**Input**: $b_1 \dots b_T$ blobs history from VeTRA 2D
**Output**: $\{s^i\}_{i=1}^{N_{\text{samples}}}$

$j$ = First blob completely inside the image;
**for** $i = 1$ ***to*** $N_{samples}$ **do**
  $c_1^i, d_1^i$ = Extract a random class and its default dimensions;
  $x_1, y_1$ = Back-project the centroid of $b_j$ in 3D;
  $x_2, y_2$ = Back-project the centroid of $b_{j+1}$ in 3D;
  $\theta_1$ = Orientation from $(x_1, y_1)$ to $(x_2, y_2)$;
  $p_1$ = extract a sample from

$$\mathbb{N}\left([x_1, y_1, \theta_1], \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}\right) \tag{12}$$

  $s^i = p_1, d_1, c_1$

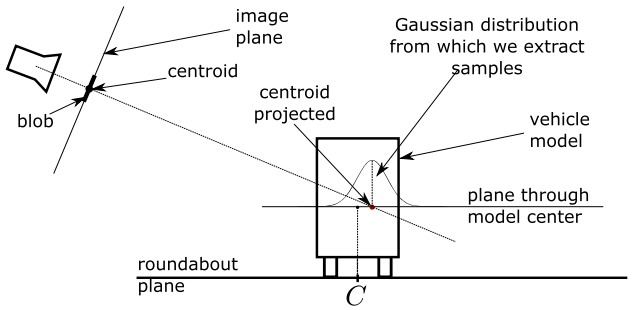



**Fig. 8** Example of 2D vehicle center projected on the plane passing through the 3D model center. We extract the pose samples from the Gaussian centered on this projection

The first step is to choose the class at the end of the transit: at each frame, we count the occurrences of each class among the the particles with the highest likelihood score, then we choose the class $c$ that appears more frequently.

Then, we define the dimensions of the model by computing the mean of the dimensions of the particles of class $c$, weighted according to the likelihood score.

In Algorithm 3, we show the pseudo-code of this model choice routine.

### 4.5 The Backward pass

Once we have selected the vehicle model, i.e., the vehicle class and dimensions, we refine the reconstructed 3D trajectory by means of a Backward PF that implements the Backward-Simulation of the Particle Smoother. The state of the Backward Particle Filter is

$$s_t^{\text{back}} = \left[x_t^{\text{back}}, y_t^{\text{back}}, \theta_t^{\text{back}}\right]^T. \tag{13}$$

The initialization of this estimator is based on the last particles extracted by the (Forward) Particle Filter belonging to the selected class, and just to consider the reverse motion of the vehicle, we apply a symmetric backward motion model with respect to the Forward Particle Filter. In the Algorithm 4, we show the pseudo-code of the whole proposed system.

**Algorithm 3**: Selection of the Best Model algorithm

**Input**: $\{s_t{}^i, w_t^i\}_{i=1,t=1}^{N_{\text{samples}}, N_{\text{frames}}}$
**Output**: $c$ class of the selected model
**Output**: $d$ dimensions of the selected model

Choose the best set of particles for each frame:
**for** $t = 1$ ***to*** $N_{frames}$ **do**
  $\{\bar{s}_t, \bar{w}_t\}$ = Pick the $\frac{N_{\text{samples}}}{100}$ samples with the highest weight;

Count the frequency of each class:
Initialize a vector $\{\text{counter}^j\}_{j=1}^{N_{\text{classes}}} = 0$ **for** $t = 1$ ***to*** $N_{frames}$ **do**
  **for** $i = 1$ ***to*** $\frac{N_{samples}}{100}$ **do**
    curClass = class of $\bar{s}_t^{\text{best}-i}$;
    $\text{counter}^{curClass} = \text{counter}^{curClass} + 1$;

Choose the best class and dimensions:
$c = max(\text{counter})$;
$\{\bar{s}_{best}, \bar{w}_{best}\}$ = samples and weight in $\{\bar{s}_t, \bar{w}_t\}$ with class $= c$;
$d$ = average of the dimensions of $\bar{s}_{best}$ weighted over the corresponding weight in $\bar{w}_{best}$;

## 5 Experimental results

The aim of our algorithm is to estimate the trajectory on the road plane, the class and the dimensions of vehicles from a sequence of images. We set up different experiments to evaluate the performances of our proposal. Our evaluations are based on two classical metrics and two robust metrics to evaluate errors statistics (see [9]): the mean of absolute values (MAE), the standard deviation (StD), the median and the Median Absolute Deviation (MAD), where, for a certain $\mathbf{x}$ vector, the MAD is defined as $median(|x_i - median(\mathbf{x})|_{\forall i})$. In our case, the vector $\mathbf{x}$ represents the errors between the estimated state and the ground truth along the trajectory. The MAE and the median error represent the absolute error of the trajectory estimate, while the StD and the MAD are error dispersion indexes; the smaller they are, the more accurate the algorithm is. We use both kinds of metrics, i.e., classical and robstified ones, since non-robust metrics give a general idea about the accuracy of the estimate, but in our experiment, we noticed that the highest errors are concentrated in the beginning and the ending of the trajectory, especially for the Monza dataset and this biases significantly the evaluation. So, to provide a more realistic accuracy evaluation, we reported also the two robust metrics which drop the transient errors.

In the following experiments, we choose the vehicle model reference dimensions as listed in Table 3.

**Algorithm 4**: Complete algorithm

**Input**: $b_1 \dots b_T$ blobs history from VeTRA 2D
**Output**: $p_1 \dots p_T$ vehicle's 3D pose
**Output**: $c$ class and dimensions of the vehicle

Forward Particle Filter(FPF):

$\{s_j{}^i\}_{i=1}^{N_{\text{samples}}}$, $j$ = FPF Initialization;
$\{w_j{}^i\}_{i=1}^{N_{\text{samples}}}$ = Likelihood of $\{s_1{}^i\}_{i=1}^{N_{\text{samples}}}$;
FPF:
**for** $f = 2$ **to** $T$ **do**
  **for** $i = 1$ **to** $N_{samples}$ **do**
    $\hat{s}_i$ = Predict from $s_{i-1}$ with the forward motion model;
    $w_i$ = Likelihood of $\hat{s}_i$;
  $\{s_t{}^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$ = Resample($\{\hat{s_t}{}^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$);

$c, d$ = Choose the Best Model ($\{s_t{}^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$);

Backward Particle Filter (BPF), keeps $c, d$ fixed:

BPF Initialization:
**for** $i = 1$ **to** $N_{samples}$ **do**
  $(x_T, y_T, \theta_T)$ = Extract the 3D pose from $s_T^i$;
  $w_i$ = Likelihood of $(x_T, y_T, \theta_T)$

BPF:
**for** $t = T - 1$ **to** 1 **do**
  **for** $i = 1$ **to** $N_{samples}$ **do**
    $\hat{s}_i$ = Predict from $s_{i+1}$ with the backward motion model;
    $w_i$ = Likelihood of $\hat{s}_i$;
  $\{s_t{}^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$ = Resample($\{\hat{s_t}{}^i, w_t^i\}_{i=1}^{N_{\text{samples}}}$);

Extract the vehicle trajectory:
**for** $t = T - 1$ **to** 1 **do**
  $\mathbb{S}_t = s_t^i | w_t^i >= mean(\{w_t^i\}_{i=1}^{N_{\text{samples}}})$;
  $p_t$ = average($\mathbb{S}_t$);

**Table 3** Reference dimensions of the models and their variances

| | $l$ (m) | $h$ (m) | $w$ (m) | $\sigma_l^2$ | $\sigma_h^2$ | $\sigma_w^2$ |
|---|---|---|---|---|---|---|
| Car | 3.8 | 1.45 | 1.7 | 0.25 | 0.15 | 0.15 |
| Truck | 15.0 | 4.5 | 2.5 | 2.5 | 0.5 | 0.3 |
| Light truck | 7.0 | 3.5 | 2.5 | 0.8 | 0.3 | 0.3 |
| Van | 4.6 | 2.0 | 1.8 | 0.25 | 0.15 | 0.15 |
| Motorcycle | 2.0 | 1.4 | 0.65 | 0.05 | 0.05 | 0.05 |
| City car | 2.5 | 1.4 | 1.5 | 0.005 | 0.005 | 0.005 |

### 5.1 Synthetic dataset

We tested the algorithm on two synthetic datasets (Fig. 9); to make the simulation realistic, we used the projection parameters estimated with camera calibration on the real data corresponding to two real datasets. In the first case, the camera is very far from the scene: it is on a high pole in the center of a roundabout, hence the scene is observed from a distant point of view. This condition gives us the possibility to evaluate our algorithm on curvilinear trajectories similar to those performed by the real vehicles. In the second case, the camera is very close to the scene: the camera is on a short pole beside the road, hence the camera is so close to the scene that large vehicles do not fit into the image plane: all trajectories are linear, but the substantial perspective distortion creates very challenging conditions. See Fig. 10 for some examples of the two real views (on the first row the *Ghisalba* setup and on the second row the *Monza* one).

Once we choose the calibration matrix to generate the data, we generated the trajectories and performed the experiments as follows. For each camera, we defined a set of possible trajectories: for the synthetic *Ghisalba*, we choose three ground truth trajectories, while in the synthetic Monza we created a linear trajectory with two motion equations (constant and variable velocity) for each of the two directions. For each pose in the trajectory, we created the history of the blobs by projecting the vehicle model on the image plane. These blobs became the input to our algorithm. In the first dataset, we used a vehicle model for each of the six different classes: car, truck, motorcycle, van, light truck, and city car. In the second dataset we used only a subset of these classes (i.e., car, motorcycle, van and city car): since the camera was very close to the road, a truck or a light truck would have covered almost the entire image for a large amount of time, making their shapes impossible to be distinguished.

The results obtained in the simulated datasets represent an upper bound to the performance of our algorithm in terms of accuracy, since the blobs are noise free and their shapes correspond exactly to the model's ones. In Table 4, we show the errors for the model dimensions and pose estimates on the two simulated scenarios. The algorithm correctly classified the vehicle in all the test cases (100 % classification accuracy). Moreover, the pose estimation results to be accurate (centimeter-accuracy for $x$ and $y$ and less than a degree for $\theta$). Notice that the pose estimation errors increase monotonically along with the class dimensions (i.e., motorcycle, city car, car, van, light truck and truck), since the dimension variance increases too.

### 5.2 Real datasets

We also tested our algorithm on two custom real datasets (*Ghisalba* and *Monza*) and on the standard *PETS 2000* sequence.[1] A fair comparison, i.e., evaluation on common datasets annotated with ground truth information against other state-of-the-art methods, was not possible. Indeed, we are proposing a novel algorithm which jointly estimates the 3D trajectory, the class, and the dimensions of the tracked vehicles; up to now the existing algorithms compared their

[1] PETS 2000: First IEEE International Workshop on Performance Evaluation of Tracking and Surveillance.

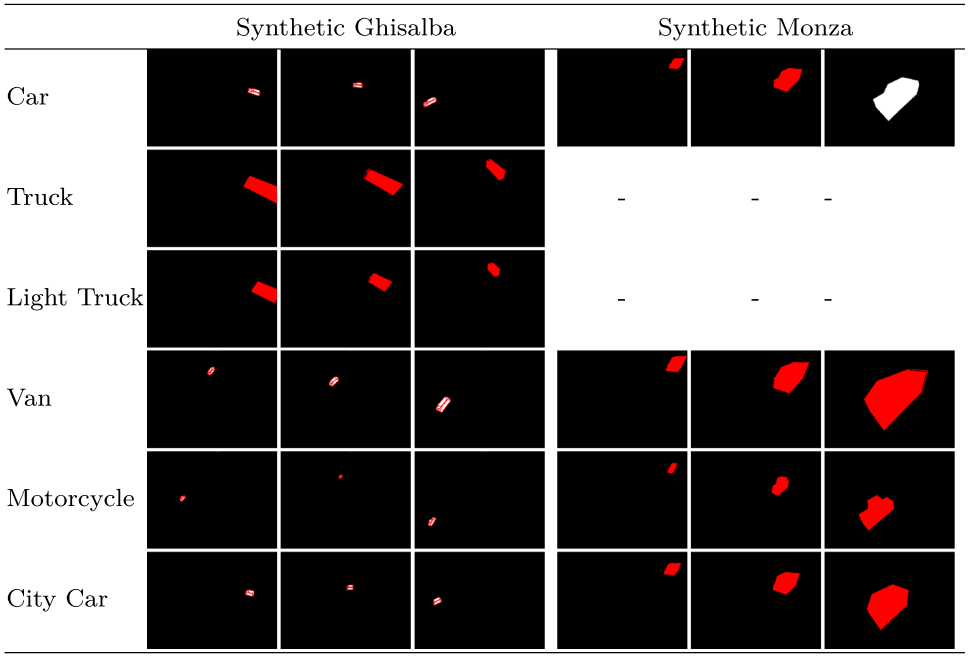



**Fig. 9** Examples of the estimation results in the synthetic datasets. In the Monza datasets, trucks and light trucks cannot be perceived due to the camera perspective

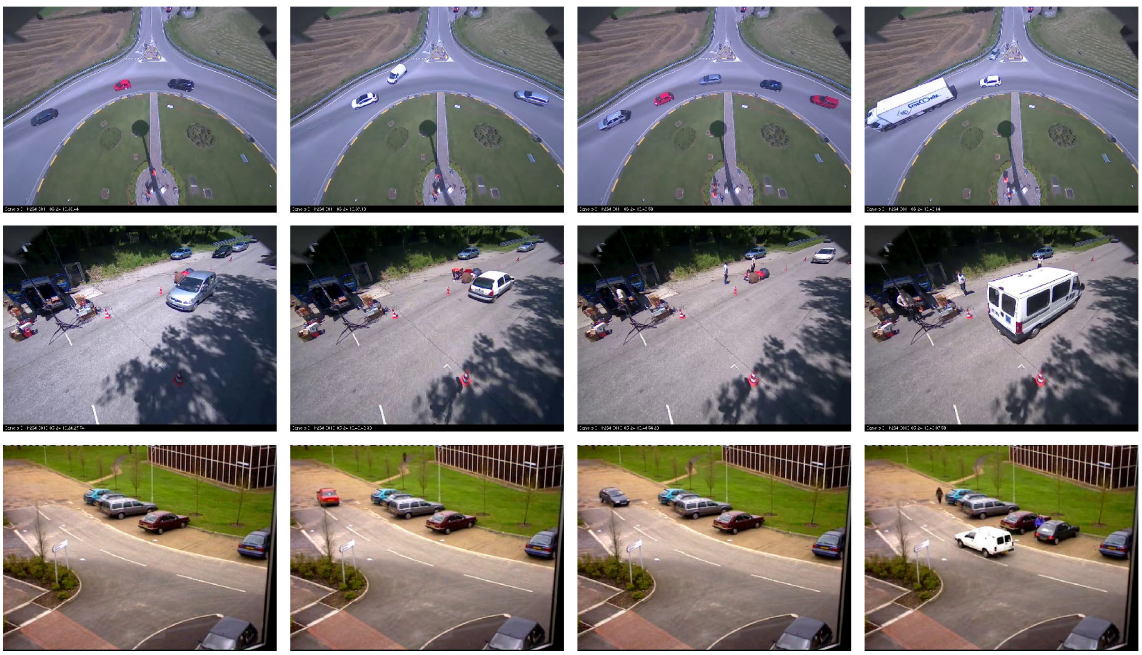


**Fig. 10** Illustrative frames from the real datasets; each row shows a different dataset: from *top* to *bottom* we have *Ghisalba*, *Monza* and *PETS 2000*

performances on 2D trajectory estimation and on the tracking task (not exactly the same of trajectory reconstruction). Moreover, the existing datasets do not provide a complete 3D trajectory annotation to evaluate our estimate; at most, they provide the calibration matrix, so we could evaluate our algorithm only qualitatively by watching the video results. Among the proposed joint tasks, we could only evaluate the classification performances, but on different datasets with respect to state of the art methods.

In this section, we show a wide variety of experiments that emphasize the accuracy reached with our method, to be ascribed to the joint estimation of the trajectory, class and dimensions.

In Fig. 10 we show three illustrative sequences of these datasets, while in Fig. 11 we show some results of the estimated pose, class and dimension. In the video at http://youtu.be/MRbwfRHQ_Ws and http://youtu.be/Hho35ng_eNI we illustrate more examples.

**Table 4** Errors of 3D pose and vehicle dimension estimated in the synthetic dataset. We list errors on the x and y coordinates, and on the orientation

| | Synthetic Ghisalba | | | | Synthetic Monza | | | |
|---|---|---|---|---|---|---|---|---|
| | Mean | StD | Median | MAD | Mean | StD | Median | MAD |
| Car | | | | | | | | |
| $x$ (m) | 0.059 | 0.072 | 0.014 | 0.058 | 0.250 | 0.813 | 0.006 | 0.017 |
| $y$ (m) | 0.138 | 0.185 | 0.061 | 0.072 | 0.540 | 0.709 | −0.399 | 0.276 |
| $\theta$ (deg) | 1.300 | 1.674 | 0.542 | 0.764 | 4.205 | 9.423 | −0.741 | −1.545 |
| $L$ (m) | 0.128 | 0.248 | −0.008 | 0.013 | 0.177 | 0.250 | 0.032 | 0.216 |
| $H$ (m) | 0.187 | 0.076 | −0.221 | 0.017 | 0.066 | 0.085 | −0.035 | 0.043 |
| $W$ (m) | 0.126 | 0.092 | 0.130 | 0.087 | 0.090 | 0.140 | 0.007 | 0.050 |
| Truck | | | | | | | | |
| $x$ (m) | 0.114 | 0.187 | 0.046 | 0.073 | – | – | – | – |
| $y$ (m) | 0.230 | 0.358 | 0.159 | 0.269 | – | – | – | – |
| $\theta$ (deg) | 1.451 | 1.847 | 0.195 | 1.097 | – | – | – | – |
| $L$ (m) | 0.516 | 0.517 | 0.503 | 0.497 | – | – | – | – |
| $H$ (m) | 0.210 | 0.318 | −0.074 | 0.012 | – | – | – | – |
| $W$ (m) | 0.075 | 0.078 | −0.080 | 0.023 | – | – | – | – |
| Motor-cycle | | | | | | | | |
| $x$ (m) | 0.032 | 0.050 | 0.003 | 0.024 | 0.019 | 0.033 | −0.009 | 0.012 |
| $y$ (m) | 0.092 | 0.151 | 0.023 | 0.040 | 0.146 | 0.200 | −0.038 | 0.100 |
| $\theta$ (deg) | 1.291 | 1.601 | 0.573 | 0.905 | 1.652 | 2.523 | −0.001 | 1.122 |
| $L$ (m) | 0.031 | 0.054 | −0.018 | 0.032 | 0.026 | 0.034 | 0.015 | 0.011 |
| $H$ (m) | 0.014 | 0.015 | −0.009 | 0.006 | 0.026 | 0.039 | 0.000 | 0.035 |
| $W$ (m) | 0.020 | 0.024 | −0.012 | 0.015 | 0.017 | 0.015 | 0.023 | 0.005 |
| Van | | | | | | | | |
| $x$ (m) | 0.048 | 0.050 | 0.040 | 0.022 | 0.091 | 0.034 | −0.040 | 0.048 |
| $y$ (m) | 0.100 | 0.137 | 0.034 | 0.051 | 0.113 | 0.034 | 0.162 | 0.205 |
| $\theta$ (deg) | 1.099 | 1.590 | 0.206 | 0.687 | 1.600 | 2.424 | −0.515 | 0.582 |
| $L$ (m) | 0.164 | 0.150 | −0.011 | 0.058 | 0.285 | 0.313 | 0.310 | 0.006 |
| $H$ (m) | 0.088 | 0.053 | 0.085 | 0.048 | 0.024 | 0.031 | 0.010 | 0.007 |
| $W$ (m) | 0.110 | 0.119 | −0.104 | 0.048 | 0.083 | 0.082 | −0.051 | 0.029 |
| Light truck | | | | | | | | |
| $x$ (m) | 0.145 | 0.181 | 0.071 | 0.078 | – | – | – | – |
| $y$ (m) | 0.301 | 0.344 | −0.152 | 0.264 | – | – | – | – |
| $\theta$ (deg) | 1.365 | 1.879 | 0.347 | 0.863 | – | – | – | – |
| $L$ (m) | 0.282 | 0.112 | 0.313 | 0.070 | – | – | – | – |
| $H$ (m) | 0.369 | 0.450 | 0.227 | 0.278 | – | – | – | – |
| $W$ (m) | 0.454 | 0.741 | −0.039 | 0.083 | – | – | – | – |
| City car | | | | | | | | |
| $x$ (m) | 0.031 | 0.038 | 0.015 | 0.019 | 0.027 | 0.445 | −0.014 | 0.019 |
| $y$ (m) | 0.093 | 0.122 | 0.029 | 0.056 | 0.157 | 0.193 | −0.163 | 0.255 |
| $\theta$ (deg) | 1.620 | 2.077 | 0.634 | 1.040 | 2.463 | 3.004 | 0.424 | 3.894 |
| $L$ (m) | 0.003 | 0.003 | −0.002 | 0.002 | 0.006 | 0.005 | 0.006 | 0.005 |
| $H$ (m) | 0.003 | 0.005 | 0.001 | 0.001 | 0.003 | 0.002 | 0.003 | 0.001 |
| $W$ (m) | 0.003 | 0.005 | −0.000 | 0.004 | 0.003 | 0.003 | −0.003 | 0.001 |

In the Ghisalba dataset the camera is fixed on a very high-lighting pole in a roundabout, in the same point of view adopted in the the first synthetic dataset. One of the captured vehicles was equipped with a RTK-GPS and an inertial sensor: these sensors give accurate estimates of the vehicle positions and orientations, and represent a reliable ground

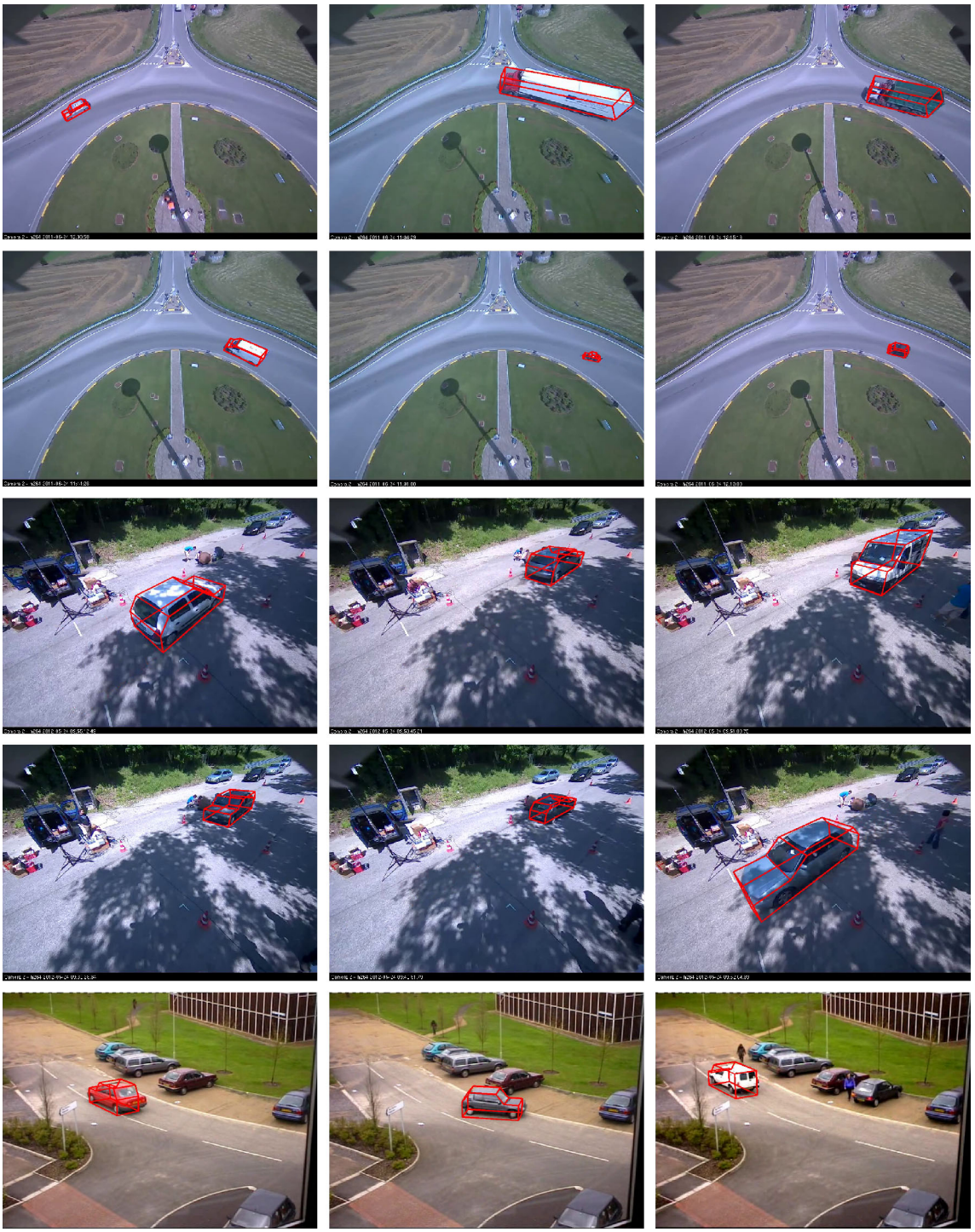


**Fig. 11** Examples of the estimation results in the real datasets. From *top* to *bottom*: two rows from Ghisalba, two rows from Monza, and the last row from the PETS2000 dataset

truth. To obtain a proper result, the comparison between poses requires the two poses (the estimated and the ground truth) to be taken at the same time. However, synchronization between RTK-GPS devices, inertial sensors and camera was clearly not perfect; in addition, the sensors and the camera working frequencies were different. Then, the estimate provided by the sensors and our algorithm cannot be considered perfectly synchronous.

In Table 5 we show the errors of the 3D pose estimate comparing the 2D VeTRA estimator, a simple Particle Fil-

**Table 5** Accuracy comparison among 2D VeTRA, our system w/o the backward pass and the proposed system, i.e., with the backward pass, in the real scenario (Ghisalba). Errors are in meters

| | Mean | StD | Median | MAD |
|---|---|---|---|---|
| $x$ (m) | | | | |
| VeTRA 2D [22] | 0.520 | 0.688 | 0.292 | 0.338 |
| w/o Back. Pass | 0.318 | **0.188** | −0.322 | **0.139** |
| w/ Back. Pass | **0.211** | 0.200 | **–0.175** | 0.154 |
| $y$ (m) | | | | |
| VeTRA 2D [22] | 0.789 | 2.013 | –0.317 | 0.284 |
| w/o Back. Pass | 0.185 | 0.303 | 0.026 | 0.187 |
| w/ Back. Pass | **0.159** | **0.222** | **0.017** | **0.159** |
| $\theta$ (deg) | | | | |
| VeTRA 2D [22] | – | – | – | – |
| w/o Back. Pass | 3.356 | 11.607 | **0.323** | 3.374 |
| w/ Back. Pass | **2.827** | **6.051** | 1.720 | **2.397** |

Best values are in bold

**Table 6** Accuracy comparison among the 3D trajectory reconstruction-only [28] and the proposed system, i.e., with the backward pass in the real scenario (Ghisalba). Errors are in meters

| | Mean | StD | Median | MAD |
|---|---|---|---|---|
| $x$ (m) | | | | |
| [28] | 0.247 | 0.445 | –0.180 | 0.236 |
| Proposed | **0.211** | **0.200** | **–0.175** | **0.154** |
| $y$ (m) | | | | |
| [28] | 0.189 | 0.238 | 0.058 | **0.135** |
| Proposed | **0.159** | **0.222** | **0.017** | 0.159 |
| $\theta$ (deg) | | | | |
| [28] | 4.882 | **4.942** | 4.156 | 2.446 |
| Proposed | **2.827** | 6.051 | **1.720** | **2.397** |

Best values are in bold

ter and the Backward-Simulation Particle Smoother. Results show the effectiveness of the backward pass: the Particle Smoother clearly overcomes VeTRA and the Particle Filter, without the backward iteration; the only metric supporting the standard Particle Filter is the median of orientation errors, likely due to a little bias. Nevertheless, the backward pass corrects most of the significant errors on it as Fig. 12 shows. In particular, in Fig. 12 we plot the position and orientation errors both in the Particle Filter and in Backward-Simulation Particle Smoother case (we plotted all vehicle transits one after the other to make the plot concise). A small bias affects the $x$ estimate, it is likely to be ascribe to the ground truth: indeed, the GPS station mounted on the vehicle was not perfectly in the centroid of the vehicle. In addition to the comparison, Table 5 shows that our algorithm produces very accurate estimates, especially taking into account the synchronization issue which may introduce additional errors. However, the videos http://youtu.be/MRbwfRHQ_Ws and http://youtu.be/Hho35ng_eNI show qualitatively very accurate estimates.

In Table 6 we compared the proposed system, which jointly estimates the trajectory, the class and the dimensions of the tracked vehicles, against the system proposed in [28] which performs a disjoint estimation of the class after the trajectory estimation through the Backward-Simulation Particle Smoother (i.e., in [28] the Backward-Simulation Particle Smoother estimates the trajectory for a set of different models independently, then the proposed system chooses the most likely model). The results in the table show how the joint estimation improves significantly the accuracy of the

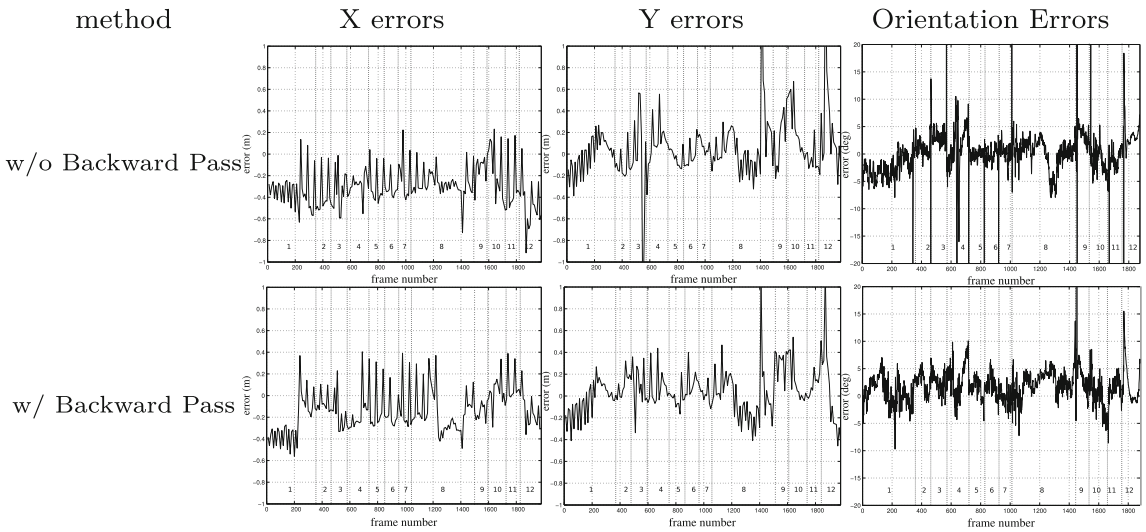



**Fig. 12** Plots of the estimation results in the Ghisalba dataset. We plot subsequently the errors related to all the transits of the sensor-equipped vehicle

**Table 7** Confusion matrix with the classification results

| Estimated real | Car | Truck | Light truck | Van | Motor-cycle | City car |
|---|---|---|---|---|---|---|
| Car | 230 | 0 | 0 | 0 | 0 | 0 |
| Truck | 0 | 21 | 2 | 0 | 0 | 0 |
| Light truck | 0 | 3 | 48 | 2 | 0 | 0 |
| Van | 7 | 0 | 3 | 61 | 0 | 0 |
| Motorcycle | 0 | 0 | 0 | 0 | 7 | 0 |
| City car | 1 | 0 | 0 | 0 | 0 | 4 |
| Class accuracy | 97 % | 88 % | 90 % | 97 % | 100 % | 100 % |

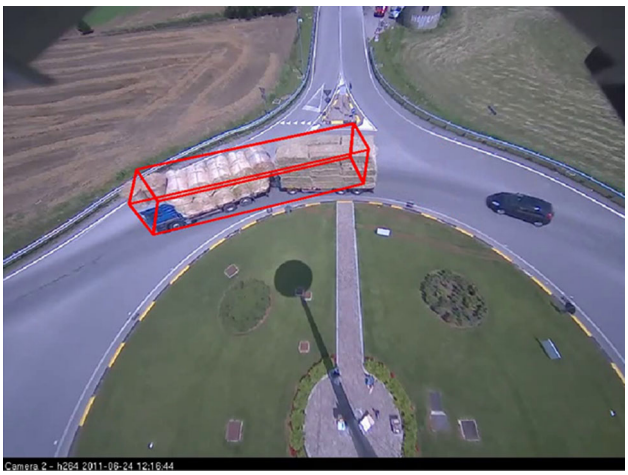


**Fig. 13** Example of the inaccurate representation of an articulated truck

estimate confirming the claims we proposed in the introduction of this paper.

In the Monza dataset, the camera is mounted few meters over the ground plane and observes a straight road (as in the second point of view of the synthetic dataset). In this case and in the PETS 2000 sequence, we did not have the ground truth for the 3D pose. Thus, the pose estimation accuracy cannot be reported for these two datasets, but both the Fig. 11 and the provided videos show very good estimation results.

In the synthetic dataset we aimed at evaluating the accuracy of the dimension estimation process. Unfortunately, in the real cases we could not provide a reliable numeric evaluation for the dimension estimates because of the errors affecting the camera calibration, mainly due to perspective effects.

In the *Ghisalba* dataset we could not use the classical chessboard pattern, together with the classical software tools to calibrate the extrinsic parameters, since the ground plane was too far from the cameras (around 25 m); thus, we georeferenced some landmarks on the ground planes with the RTK-GPS and used the DLT algorithm [8] to estimate the extrinsic calibration matrix: the obtained calibration is accurate on the ground plane but rapidly degrades as soon as the height increases. In the *Monza* dataset, the challenging perspective (and camera distortions) makes the extrinsic calibration very noisy: the point at the infinity is quite unstable and the standard chessboard extrinsic calibration process [33] provides a poor estimate. In those two scenarios, when the model fits the vehicle image, i.e., when the algorithm provides a good estimation, it is likely that the dimension values are different from the real ones, and we cannot ascribe the error to our algorithm.

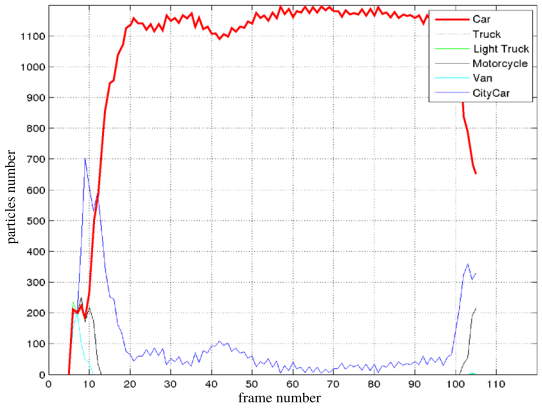
In the *PETS 2000* dataset, the algorithm shows accurate estimate of the vehicle dimensions qualitatively. In this case, the calibration is known but we are not able to find the vehicle producer and model so we cannot infer the real dimensions.

In Table 7, we list the results of classification over a set of 389 vehicles captured in the three scenes (only 8 of them belong to the Monza dataset and 3 belongs to the PETS2000 sequence, and these are all correctly classified). We annotated the datasets manually before running our algorithm. The classification results quite accurate for cars, city cars, motorcycles and vans, and the algorithm properly classifies most of the light trucks too. Trucks, are usually classified correctly, but the dimensions are sometimes not precisely estimated. The reason behind such issue stems from the fact that trucks are usually articulated: if an articulated truck turns significantly, the shape of the corresponding blob is quite different from the model (see Fig. 13) and our algorithm sometimes tends to estimate as a light truck only one of the two articulated parts of the truck.

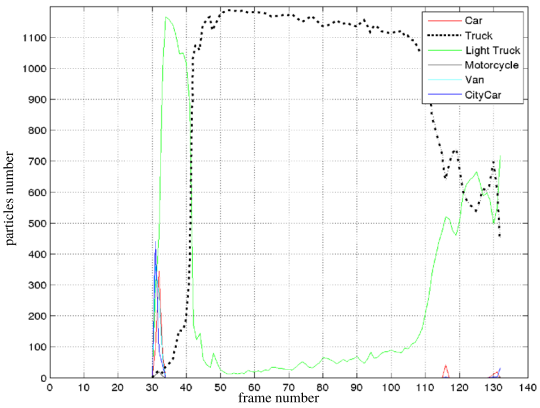
**Table 8** Classification accuracy comparison

| References | Classification accuracy (%) |
|---|---|
| [18] | 91.5 |
| [4] | 89.8 |
| [3] | 92.1 |
| Proposed | 95.3 |

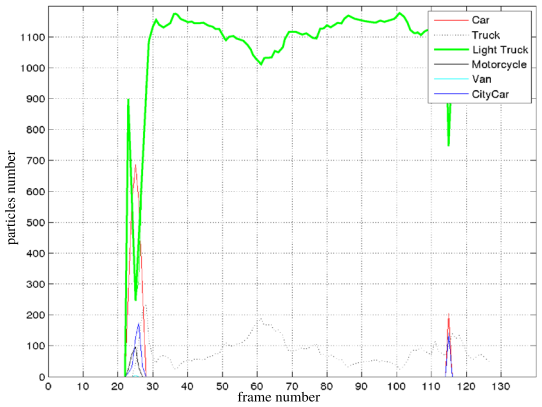
In Table 8 we show the comparison with the average classification accuracy of the state-of-the-art methods. As mentioned in the beginning of this section, the datasets are different with respect to the ones provided in this paper; however, the scenarios considered for our experiments are similar, so a qualitative comparison is possible. The table shows that our algorithm reaches 95.3 % classification accuracy, better than the other algorithms, despite the issue with the articulated trucks (not appearing in the datasets on which the other algorithms were tested). We computed this accuracy as $\frac{N^{cc}_{\text{vehicles}}}{N_{\text{vehicles}}}$, where $N^{cc}_{\text{vehicles}}$ is the number of vehicles correctly classified, and $N_{\text{vehicles}}$ is the total number of vehicles.



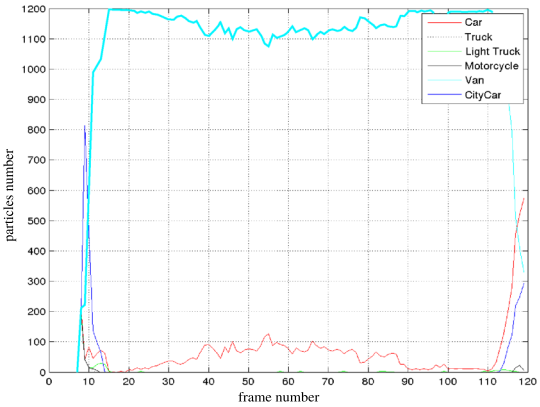

*Vehicle 33 (final estimated class: car)*



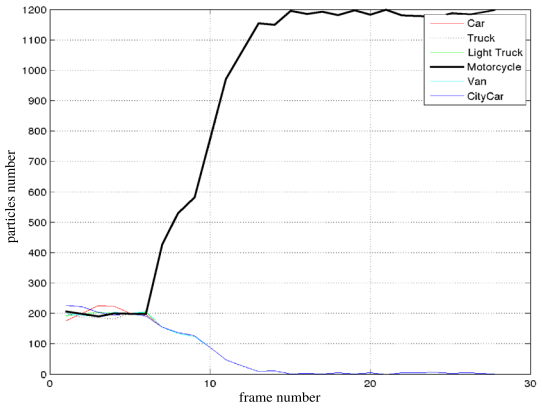

*Vehicle 52 (final estimated class: truck)*



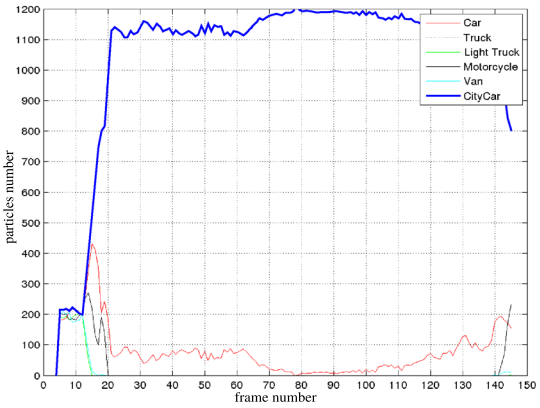

*Vehicle 35 (final estimated class: light truck)*




*Vehicle 37 (final estimated class: van)*




*Vehicle 176 (final estimated class: motorcycle)*




*Vehicle 222(final estimated class: city car)*

**Fig. 14** Six examples of how the class estimate evolves for six different vehicle transits (one example for each class). Each *plot* shows how many particles belong to the six classes, frame-by-frame. The *last region* of the plots shows a change in the class distribution except for the motorcycle case; this happens when the vehicle reaches the image borders, so the corresponding blob looks like a vehicle from smaller class

In Fig. 14 we report six examples of how the class distributions evolve among the particles of the proposed Backward-Simulation Particle Smoother; each example corresponds to a different estimated class value. In each plot we represent, for each frame, how many particles belongs to each of the six different classes. This figure shows how the filter states converge to the right class value (each plot in the figure refers to a single vehicle for a different class). At the end of the trajectory the filter state tends to diverge from the final class, but this is due to the fact that the vehicle is disappearing from

the image and its blob looks like a vehicle in a smaller class. We did not introduce any heuristic to deal with the latter issue since it turns out not being critical in terms of overall classification accuracy.

## 6 Conclusions and future works

In this paper, we presented a Backward-Simulation Particle Smoother with a hybrid state to estimate simultaneously the 3D trajectory, the class, and the dimensions of the observed vehicles from an image sequence. In vehicle monitoring, those are three relevant tasks; the existing systems only perform one or two of them simultaneously, usually by means of one or more vehicle models. A unifying framework is worthy, since the synergy of the three processes makes the estimation robust: trajectory estimation enforces classification with temporal consistency, and at the same time, the knowledge about the vehicle class and dimensions can be used to increase the accuracy of the trajectory estimate.

As in many vehicle tracking systems, we framed our algorithm according to the Bayesian estimation framework. Differently from existing algorithms, our proposed estimator deals with continuous (pose and dimensions) and discrete (class) quantities, and this requires an estimator with a hybrid state. To manage the hybrid state in a Bayesian Estimator, the major challenge came from the design of a convenient state transition function for the prediction step. We split the prediction step into two stages: in the first one, we generated a new pose according to a classical motion model; in the second one, we choose the class and the dimension by modeling the class transition as a Markov Chain.

Moreover, in this paper we proposed to accomplish the Bayesian estimation via the Backward-Simulation Particle Smoother, and the reasons were twofold. First, a Monte Carlo estimation process, as the Particle Smoother, resulted to be the most convenient choice to deal with the likelihood estimation (compared to the Kalman Filter or Smoother), when adopting a region-based approach. In particular, with the Particle Smoother we could simply compute an overlap score between the vehicle model projection and the vehicle measurement to weigh the pose hypotheses. Secondly, we choose a Particle Smoother to perform a more accurate estimation with respect to the filtering approach; in particular, the Backward-Simulation Particle Smoother let us to estimate both the poses and the vehicle model, i.e., the class and the dimensions, with the Forward iteration, while we used the Backward iteration to refine the poses estimate.

We tested our algorithm on both synthetic and real datasets. The synthetic datasets gave us a quantitative evaluation on an ideal case especially for pose and dimension estimation: the results are very accurate, usually the error was about few centimeters in displacement and less than one degree. In the three real datasets (Ghisalba with ground truth, Monza and PETS 2000), the overall accuracy of the pose and dimension estimate is good. The classification success rate is more than 95 %. The only weak results are related to the articulated truck estimation.

As a future work, we aim at introducing and managing a more complex truck model which should take into account the articulation, and a more complex motion model.

**Acknowledgments** This work has been partially supported by the Italian Ministry of University and Research (MIUR) through the "SMELLER" project.

## References

1. Arulampalam, M., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. Signal Process. IEEE Trans. **50**(2), 174–188 (2002). doi:10.1109/78.978374
2. Bar-Shalom, Y., Li, X.R., Kirubarajan, T.: Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software. Wiley, New York (2001)
3. Buch, N., Orwell, J., Velastin, S.A.: 3d extended histogram of oriented gradients (3dhog) for classification of road users in urban scenes (2009)
4. Buch, N., Orwell, J., Velastin, S.A.: Urban road user detection and classification using 3d wire frame models. IET Comput. Vis. **4**(2), 105–116 (2010)
5. Buch, N., Velastin, S., Orwell, J.: A review of computer vision techniques for the analysis of urban traffic. Intell. Transp. Syst. IEEE Trans. **12**(3), 920–939 (2011). doi:10.1109/TITS.2011.2119372
6. Buch, N., Yin, F., Orwell, J., Makris, D., Velastin, S.: Urban vehicle tracking using a combined 3D model detector and classifier. In: Velásquez, J., Ríos, S., Howlett, R., Jain, L.: (eds.) Knowledge-Based and Intelligent Information and Engineering Systems, Lecture Notes in Computer Science, vol. 5711, pp. 169–176. Springer, Berlin (2009). doi:10.1007/978-3-642-04595-0_21
7. Godsill, S.J., Doucet, A., West, M.: Monte carlo smoothing for nonlinear time series. J. Am. Stat. Assoc. **99**, 465 (2004)
8. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, vol. 2. Cambridge University Press, Cambridge (2000)
9. Huber, P.J.: Robust Statistics. Springer, New York (2011)
10. Johansson, B., Wiklund, J., Forssén, P., Granlund, G.: Combining shadow detection and simulation for estimation of vehicle size and position. Pattern Recognit. Lett. **30**(8), 751–759 (2009). doi:10.1016/j.patrec.2009.03.005
11. Koller, D., Daniilidis, K., Nagel, H.H.: Model-based object tracking in monocular image sequences of road traffic scenes. Int. J. Comput. Vis. **10**, 257–281 (1993). doi:10.1007/BF01539538
12. Lai, A., Fung, G., Yung, N.: Vehicle type classification from visual-based dimension estimation. In: Proceedings of Intelligent Transportation Systems 2001, IEEE, pp. 201–206 (2001)
13. Lepetit, V., Fua, P.: Monocular model-based 3D tracking of rigid objects: a survey, vol. 1. In: Proceedings of Foundations and Trends in Computer Graphics and Vision, pp. 1–89 (2005)
14. Lou, J., Tan, T., Hu, W.: Visual vehicle tracking algorithm. Electron. Lett. **38**(18), 1024–1025 (2002)
15. Lou, J., Tan, T., Hu, W., Yang, H., Maybank, S.: 3-D model-based vehicle tracking. Image Process. IEEE Trans. **14**(10), 1561–1569 (2005)

16. Marslin, R., Sullivan, G.D., Baker, K.: Kalman filters in constrained model-based tracking. Proc. British Mach. Vis. Conf. **1991**, 371–374 (1991)
17. Matteucci, M., Rizzi, D., Romanoni, A., Mussone, L.: 3d image processing of vehicular trajectories in roundabouts. In: Proceedings of World Conference on Transportation Research, pp. 1–11 (2013). http://www2.wctr2013rio.com/publications/1077/index.html
18. Messelodi, S., Modena, C., Zanin, M.: A computer vision system for the detection and classification of vehicles at urban road intersections. Pattern Anal. Appl. **8**, 17–31 (2005). doi:10.1007/s10044-004-0239-9
19. Messelodi, S., Modena, C.M., Zanin, M.: A computer vision system for the detection and classification of vehicles at urban road intersections. Pattern Anal. Appl. **8**(1–2), 17–31 (2005)
20. Migliore, D.A., Matteucci, M., Naccari, M.: A revaluation of frame difference in fast and robust motion detection. In: Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks, ACM, pp. 215–218 (2006)
21. Mussone, L., Matteucci, M., Bassani, M., Rizzi, D.: An innovative method for the analysis of vehicle movements in roundabouts based on image processing. J. Adv. Transp. (2011)
22. Mussone, L., Matteucci, M., Bassani, M., Rizzi, D.: Traffic analysis in roundabout intersections by image processing. In: Proceedings of the 18th IFAC World Congress, vol. 18, pp. 14922–14927 (2011)
23. Neal, R.M.: Probabilistic inference using markov chain monte carlo methods (1993)
24. Pang, C., Lam, W., Yung, N.: A method for vehicle count in the presence of multiple-vehicle occlusions in traffic images. Intell. Transp. Syst. IEEE Trans. **8**(3), 441–459 (2007)
25. Piccardi, M.: Background subtraction techniques: a review. In: Proceedings of Systems, Man and Cybernetics, 2004 IEEE International Conference vol. 4, pp. 3099–3104 (2004). doi:10.1109/ICSMC.2004.1400815
26. Romanoni, A.: Ricostruzione 3d delle traiettorie veicolari da immagini di una o più telecamere (2012). M.Sc. Thesis (in Italian)
27. Romanoni, A., Matteucci, M., Sorrenti, D.: Background subtraction by combining temporal and spatio-temporal histograms in the presence of camera movement. Mach. Vis. Appl. **25**(6), 1573–1584 (2014).10.1007/s00138-013-0587-9
28. Romanoni, A., Mussone, L., Rizzi, D., Matteucci, M.: A comparison of two monte carlo algorithms for 3d vehicle trajectory reconstruction in roundabouts. Pattern Recognit. Lett. **51**(0), 79–85 (2015). doi:10.1016/j.patrec.2014.09.003
29. Salih, Y., Malik, A.S.: Comparison of stochastic filtering methods for 3D tracking. Pattern Recognit. **44**(10–11), 2711–2737 (2011). (Comparison among filters)
30. Särkkä, S.: Bayesian Filtering and Smoothing, vol. 3. Cambridge University Press, Cambridge (2013)
31. Song, X., Nevatia, R.: Detection and tracking of moving vehicles in crowded scenes. In: Proceedings of Motion and Video Computing, 2007, IEEE Workshop on WMVC '07, p. 4 (2007). doi:10.1109/WMVC.2007.13
32. Tan, T., Sullivan, G., Baker, K.: Model-based localisation and recognition of road vehicles. Int. J. Comput. Vis. **27**, 5–25 (1998). doi:10.1023/A:1007924428535
33. Zhang, Z.: A flexible new technique for camera calibration. Pattern Anal. Mach. Intell. IEEE Trans. **22**(11), 1330–1334 (2000)
34. Zhang, Z., Tan, T., Huang, K., Wang, Y.: Three-dimensional deformable-model-based localization and recognition of road vehicles. Image Process. IEEE Trans. **21**(1), 1–13 (2012). doi:10.1109/TIP.2011.2160954

**Andrea Romanoni** received the B.S. and M.S. degrees in Computer Engineering from Politecnico di Milano in 2009 and 2012, respectively. He then was a Research Assistant at Università degli Studi Milano–Bicocca, for the SMELLER project. Currently, he is a Ph.D. student at Politecnico di Milano, founded by the SINOPIAE project. His research interests are in Computer Vision and Robotics: in particular, in Vehicle Tracking and 3D Urban Reconstruction.

**Domenico G. Sorrenti** obtained a maturità classica (high school, with literature and philosophy orientation, diploma) at Liceo G. Carducci, Milano, in July 1981. He studied Electronic Engineering at Politecnico di Milano and obtained the Laurea (Master) in February 1989. He then enrolled in the Ph.D. programme in Computer Engineering and Automation at Politecnico di Milano, and obtained the Ph.D. degree in December 1992. Currently, he is associate professor in computer engineering with the Department Informatica, Sistemistica e Comunicazione of Università degli Studi di Milano–Bicocca. His research interests are in robotic perception for autonomous vehicles, and video-surveillance. His research has been funded by EEC, National agencies (mainly MIUR), Lombardy region, and private companies.

**Matteo Matteucci** received the Laurea degree in Computer Engineering from Politecnico di Milano, in 1999, the M.S. in Knowledge Discovery and Data Mining from Carnegie Mellon University in 2002, and the Ph.D. in Computer Engineering and Automation from Politecnico di Milano, in 2003. Since 2005 he is a Faculty member at Politecnico di Milano. Currently, he is an associate professor in the Department of Electronics Information and Bioengineering. He has published more than 100 contributions on refereed journal and conferences. His research activity is in Robotics mainly focused on perception, tracking, sensor fusion, simultaneous localization and mapping. He has been involved in the Euron Special Interest Group on Good Experimental Methodologies and Benchmarking and is in the IEEE RAS Standard Group for the definition of IEEE P1873/D1 Draft Standard for Robot Map Data Representation for Navigation.