MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Motion Planning of Autonomous Road Vehicles by Particle Filtering

Berntorp, K.; Hoang, T.; Di Cairano, S.

TR2018-164 December 07, 2018

Abstract

This paper describes a probabilistic method for real-time decision making and motion planning for autonomous vehicles. Our approach relies on that driving on road networks implies a priori defined requirements that the motion planner should satisfy. Starting from an initial state of the vehicle, a map, the obstacles in the region of interest, and a goal region, we formulate the motion-planning problem as a nonlinear nonGaussian estimation problem, which we solve using particle filtering. We assign probabilities to the generated trajectories according to their likelihood of obeying the driving requirements. Decision making and collision avoidance is naturally integrated in the approach. We develop a receding-horizon implementation and verify the method in simulated real road scenarios and in an experimental validation using a scaled mobile robot setup with car-like dynamics. The results show that the method generates dynamically feasible trajectories for a number of scenarios, such as collision avoidance, overtaking, and traffic-jam scenarios. In addition, the computation times and memory requirements indicate that the method is suitable for real-time implementation.

IEEE Transactions on Intelligent Vehicles

Motion Planning of Autonomous Road Vehicles by Particle Filtering

Karl Berntorp[1], Tru Hoang[1], and Stefano Di Cairano[1]

***Abstract*—This paper describes a probabilistic method for real-time decision making and motion planning for autonomous vehicles. Our approach relies on that driving on road networks implies a priori defined requirements that the motion planner should satisfy. Starting from an initial state of the vehicle, a map, the obstacles in the region of interest, and a goal region, we formulate the motion-planning problem as a nonlinear non-Gaussian estimation problem, which we solve using particle filtering. We assign probabilities to the generated trajectories according to their likelihood of obeying the driving requirements. Decision making and collision avoidance is naturally integrated in the approach. We develop a receding-horizon implementation and verify the method in simulated real road scenarios and in an experimental validation using a scaled mobile robot setup with car-like dynamics. The results show that the method generates dynamically feasible trajectories for a number of scenarios, such as collision avoidance, overtaking, and traffic-jam scenarios. In addition, the computation times and memory requirements indicate that the method is suitable for real-time implementation.**

I. INTRODUCTION

Autonomous vehicles are complex decision-making systems that integrate advanced and interconnected sensing and control components, see Fig. 1. While autonomous vehicles increasingly begin testing on public roads, production vehicles are more commonly being equipped with advanced driver-assistance systems (ADAS) such as adaptive cruise control and lane-change assist. This is driven by both safety and economic aspects such as the high number of traffic accidents associated with overtaking and lane-change maneuvers and potential fuel savings [1]. Fig. 1 provides a typical high-level system schematics for an autonomous vehicle [2]. At the highest level a route is planned through the road network by the route planner, based on a user-defined destination. The route plan can be given by intermediate goals $\mathcal{X}_{\text{goal}}$, for example, represented as possible lanes at an intersection and/or a particular road after an intersection. Then, a discrete decision-making layer is responsible for determining the local driving behavior of the vehicle. For instance, a decision could be any of turn right, stay in lane, turn left, or come to full stop in a particular lane at an intersection. The motion planner is then responsible for determining a desired trajectory that the vehicle should follow based on the outputs from the sensing and mapping module and the decision making module. The sensing and mapping module uses various sensor information, such as radar, Lidar, camera, and global positioning system (GPS) information, together with prior map information, to estimate the parts of the surroundings relevant to the driving scenario. Important requirements are that the trajectory computed by the motion planner is collision free, dynamically feasible, and possible to track by the vehicle controller.

This paper[1] develops a probabilistic method for integrated decision making and motion planning. Specifically, we pose the combined decision making and motion planning problem as an estimation problem, and leverage *particle filtering* for approximating the involved probability density functions (PDFs). Particle filtering is a sampling-based technique for solving the nonlinear filtering problem. The particle filter (PF) numerically approximates the PDF of the variables of interest given the measurement history, by generating random trajectories and assigning a weight to them according to how well they predict the observations. From the observation that the driving requirements, such as staying on the road, right-hand traffic, and obstacle avoidance, are known ahead of planning, we formulate the driving requirements as measurements generated by an ideal system. An interpretation for our approach is that the PF determines decisions and corresponding trajectories, and scores them according to how likely they are to obey the driving requirements. In each planning phase, the PF approximates the joint PDF of the state trajectory conditioned on the decision and driving requirements. Our method solves the decision-making and motion-planning problems in an integrated fashion, and it samples both decisions (modes) and trajectories within the particle-filter framework. By relying on the PF framework, we can utilize the structure of the environment to only search in areas that are likely to give good trajectories.

PFs can achieve arbitrarily good estimates [5], [6]. PFs are simulation based, which implies dynamic feasibility. There exist methods for biasing the state trajectories toward the driving requirements by choosing proper control inputs, which significantly reduce the computations and hence ensure real-time feasibility of the approach. Because of the biasing our method is fast in determining dynamically feasible trajectories that are intuitive with respect to the driving requirements, at the price of being slightly less efficient in finding trajectories that are counter-intuitive with respect to the driving requirements. Therefore, our approach is suitable for normal driving situations where the trajectories typically executed by human drivers are intuitive, and hence the trajectories generated by our method will provide confidence to passengers of the autonomous vehicle.

[1] The authors are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139, USA. Email: karl.o.berntorp@ieee.org.

[1] A preliminary version of this work was presented in [3], [4]. The current, elaborated, version contains an extension to discrete decision making, thorough description of the method and algorithms, and experimental validation.

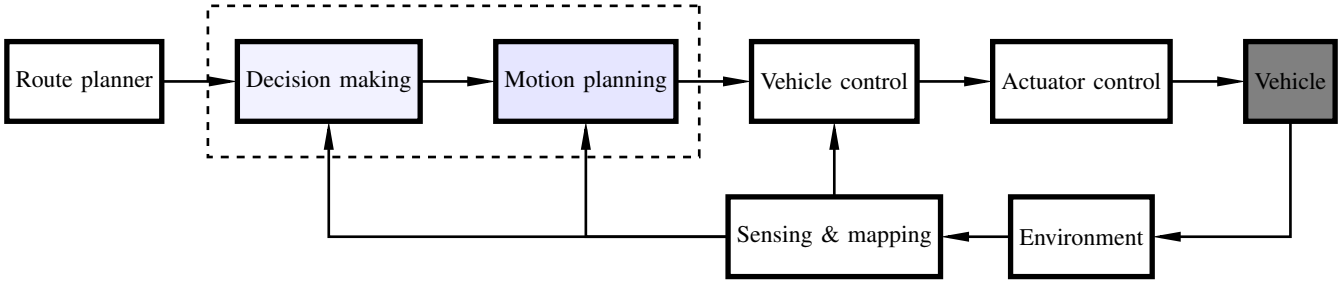

Fig. 1. A high-level system architecture of an autonomous vehicle. The different blocks can be interconnected in various ways but the main building blocks typically remain the same. In our approach, we consider the joint decision-making and motion-planning problem.

Notation: Throughout, $p(\boldsymbol{x}_{0:k}|\boldsymbol{y}_{0:k})$ denotes the conditional probability density function of the state trajectory $\boldsymbol{x} \subset \mathcal{X} \in \mathbb{R}^{n_x}$ at time $t_k \in \mathbb{R}$ conditioned on the variable (measurement) $\boldsymbol{y} \subset \mathcal{Y} \in \mathbb{R}^{n_y}$ from time t_0 to time t_k, $\boldsymbol{y}_{m:k} := \{y_m, \ldots, \boldsymbol{y}_k\}$. Given mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ stand for the Gaussian distribution and PDF, respectively. The notation $\boldsymbol{x} \sim p(\cdot)$ means $\boldsymbol{x}$ sampled from $p(\cdot)$ and $\propto$ reads proportional to.

Outline: This paper is outlined in the following way. Sec. II gives an overview of some of the previous work related to autonomous vehicle motion planning. Sec. III provides the models, and Sec. IV gives the scope of the motion-planning problem. Sec. V develops the proposed method. In Sec. VI we provide simulation study, which is followed by an experimental evaluation of our approach in Sec. VII. Finally, conclusions are drawn in Sec. VIII.

II. Previous Work

The motion-planning problem in autonomous vehicles has many similarities with the standard robotics setup [7], and exact optimal solutions are in most cases intractable. Approaches relying on model predictive control (MPC) are available [8]–[13]. However, a limiting factor with these approaches is typically nonconvexity [9]. This results in achieving only a locally optimal solution, which may be significantly far from the globally optimal one, and possibly in a very large computational load and time. Also, while MPC efficiently converges for convex problems, using MPC for highly nonconvex problems can result in getting stuck in local infeasible solutions, with nonconvergence as a result. Instead, motion planning in autonomous vehicle research is often performed using either sampling-based methods such as rapidly-exploring random trees (RRTs) [7], graph-search methods [14], [15] such as A* [16] or D* and variations of it [17], [18], or optimal control, possibly using MPC for tracking the motion plan [19], [20].

RRTs rely on random exploration of the state space. RRT-type methods are very suitable for unstructured environments, due to their random sampling leading to guarantees of finding feasible solutions whenever a solution exists. Optimal variants of the suboptimal RRT have been recently developed [21], [22]. RRTs can solve autonomous vehicle motion-planning problems assuming a target region is given by the decision maker. This provides for a decomposition of the decision-making and motion-planning problem, which simplifies the problem, but can lead to inconsistencies between the decision maker and motion planner, since the route planner typically does not know the limits of the motion planner. Furthermore, RRTs are generally applicable but can be inefficient for non-trivial and/or underactuated dynamics due to the computational cost or limitations in computing steering laws to connect points sampled in the configuration space [23], [24]. Furthermore, the quality of the path in RRT can vary heavily between any two planning instants, which is highly undesirable for autonomous driving. Hence, a tailored implementation is typically needed to achieve the performance requirements set for autonomous vehicles.

Sampling-based methods have been applied to autonomous vehicles. For instance, [25] proposed a kinodynamic RRT handling the dynamic feasibility issue, and [26] considered time-optimal motion planning by leveraging differential flatness in the single-track vehicle model. Kinodynamic RRTs randomly choose a node, generate random inputs, and propagate them through the system model [27]. This reduces the tree expansion to integration of the system model and by construction generates drivable paths, provided a realistic model. However, it introduces other potential problems, such as sparse coverage of the reachable set [25], and the generated trajectory is of varying quality. For efficiency, [28] proposed closed-loop RRT (CL-RRT), which uses closed-loop prediction for trajectory generation. The optimal version is in [29]. The method in [30] uses a dynamic programming phase for a coarse trajectory solution, which is then refined by sampling. Table I gives an overview of some different approaches for motion planning and their properties.

Our approach borrows concepts from RRT in that we also perform random sampling to construct a tree expansion of reachable locations. However, to overcome some of the limitations of RRT-like methods, we do not sample the state space, but rather the input space. First, this implies a reduced search dimension, since the input space is usually of lower dimension than the state space. Second, the trajectories satisfy the differential constraints (i.e., the dynamics of the vehicle) by construction, thus avoiding the need for steering algorithms. Third, sampling the input space produces trajectories, rather than paths, which is beneficial in scenarios with dynamically moving obstacles. We also add an additional term correcting the sampled input based on the driving requirements, hence generating trajectories such that we minimize the deviations from the driving requirements in the probabilistic sense. This

TABLE I

DIFFERENT METHODS FOR MOTION PLANNING AND THEIR PROPERTIES. RRT-TYPE METHODS WORK UNDER GENERAL CONDITIONS BUT MAY BECOME INEFFICIENT IN CERTAIN CASES. OPTIMIZATION-BASED METHODS (E.G., MPC) ARE TYPICALLY VERY EFFICIENT FOR (CLOSE TO) CONVEX PROBLEMS, BUT CAN BE COMPUTATIONALLY HEAVY AND GET STUCK IN LOCAL INFEASIBLE SOLUTIONS FOR NONCONVEX PROBLEMS.

RRT-type	
Pros	**Cons**
feasibility	randomness
optimality	inefficiency
global solution	dynamic feasibility
Optimization	
Pros	**Cons**
optimality (convex)	numerical problem
efficient solvers	computational cost
dynamic feasibility	

results in finding relatively quickly trajectories that are intuitively good, although it may slow down the computation of the optimal trajectory with respect to any of the RRTs for optimal motion planning [21], [22]. However, our rationale is that in automated driving under normal conditions, we seek to quickly find trajectories that are effective, but not necessarily optimal.

III. MODELING

In this section we introduce the different discrete-decision model and the vehicle model used in the problem solution. We refer to the automated vehicle as the ego vehicle (EV), whereas other moving entities in the region of interest (ROI) of the EV are designated as other vehicles (OV). That the OVs can be either autonomous or manual vehicles, as we do not assume any explicit collaboration between different vehicles. In the following, we model the vehicle with respect to the global inertial frame. Although throughout the paper, for illustration we will use a specific vehicle model, our method handles general discrete-time nonlinear vehicle models for describing the time evolution of the EV that can be written in the form

$$\boldsymbol{x}_{k+1} = \bar{\boldsymbol{f}}(\boldsymbol{x}_k) + \bar{\boldsymbol{g}}(\boldsymbol{x}_k)\boldsymbol{u}_k, \tag{1}$$

with EV state $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ and EV input $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$, and k is the time index corresponding to time t_k.

We introduce the following assumptions.

Assumption 1: Positions and velocities of the OVs relative to the EV at the current time are known.

Note that the quantities involved in Assumption 1 can be measured and estimated by onboard sensors such as cameras, Lidars, radars, and/or ultrasound sensors attached to the EV. The future states of the OVs over the planning horizon are not assumed to be known a priori in the method we propose in this paper, but the estimation of the future state is incorporated into the planning method.

Assumption 2: The road geometry, number of lanes, and the direction of travel in each lane is known.

The quantities involved in Assumption 2 are usually known over the ROI from maps and onboard cameras. We collect the road information over the ROI in the road state $\boldsymbol{x}^{\mathrm{RD}}$.

A. Discrete Decision Model

Given the route plan, when driving on road networks such as multiple-lane highways or city driving, there are behavioral driving decisions that need to be determined. For instance, if we consider a three-lane road and have a high-level route coming from a route planner (Fig. 1), the possible decisions are either to come to a full stop (S), stay in lane (SL), change lane left (CLL), or change lane right (CLR). The possible decisions can be modeled as a set of modes

$$\mathcal{M} = \{S, SL, CLL, CLR\} = \{m_1, \ldots, m_4\}. \tag{2}$$

We model the decision making model as a finite-state Markov process with transition probabilities associated with each decision. These transition probabilities can be determined, for instance, from the driving context perceived from the sensing and prediction modules, in combination with the route commanded by the route planner. We write the transition model between any two modes j and i as

$$m_j \sim p(m_j|m_i), \tag{3}$$

which is determined from the transition probabilities. When there is no a priori information from the sensing and prediction modules, the transition probabilities encoded in (3) can be set to the same value for all transitions.

B. Vehicle Model for Motion Planning

We introduce the following assumption on the driving behavior assumed in the motion planner.

Assumption 3: The planner operates regular driving maneuvers, while emergency braking and aggressive evasive maneuvers are handled by a separate control system.

Based on Assumption 3, the motion planner can be based on a single-track model [31]. Assumption 3 is reasonable, since the decision maker and motion planner typically execute at considerably slower update rates than the vehicle controller , and dynamic effects such as wheel slip are handled by the lower-level control layers [2]. However, note that the planner we present is not limited to the driving behavior imposed by Assumption 3. When Assumption 3 does not hold, more complex vehicle models may be included in (1).

A model based on force-mass balances is generally more accurate than a kinematic model, but for regular driving the differences are small [32], and remaining model mismatches are compensated by lower-level control logics (Fig. 1). In this paper we use the discretized version of the kinematic single-track model [32],

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{p}_X \\ \dot{p}_Y \\ \dot{\psi} \\ \dot{v}_x \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\psi + \beta)/\cos(\beta) \\ v_x \sin(\psi + \beta)/\cos(\beta) \\ v_x \tan(\delta)/L \\ u_1 \\ u_2 \end{bmatrix}, \tag{4}$$

where p_X, p_Y are the longitudinal and lateral position in the world frame, respectively, ψ is the heading (yaw) angle of the EV, $\dot{\psi}$ is the yaw rate, v_x is the longitudinal velocity of the EV, δ is the steering angle of the front wheel, $L := l_f + l_r$ is the wheel base, and $\beta := \arctan\left(l_r \tan(\delta)/L\right)$ is the (kinematic)

body-slip angle. The inputs u_1, u_2 are the acceleration and steering rate, respectively, which allows to obtain smooth velocity and steering profiles and to constrain the rate of changes of the velocity and steering angle, respectively.

We impose various state and input constraints on the vehicle. The steering angle δ is subject to linear constraints, and linear input constraints on the steering rate $\dot{\delta}$ and acceleration $\dot{v}_x$ are introduced. These constraints can be compactly written as

$$\mathcal{U} = \{\boldsymbol{u}_k : \boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max}\}. \tag{5}$$

The road boundaries impose constraints on the position vector $\boldsymbol{p}$, as do obstacles in the region of interest (ROI) of the vehicle. These constraints are in general nonconvex, but as it will be explained later in Sec. V, they are directly handled in our approach. The road-boundary constraint can be written as

$$\Gamma(p_X, p_Y) \leq 0, \tag{6}$$

where the function Γ is typically constructed from point-wise data of the road and lane boundaries.

In general the constraints due to the OVs can take any shape. For instance, if the motion of the OVs is estimated by means of Kalman filters, a natural choice is to model the OVs as ellipsoids. Alternatively, the OVs can be modeled as having rectangular shape [9]. The spatial extent of the collision area of the EV around the oth OV is denoted with $\mathcal{B}_o$, and the corresponding OV state is

$$\boldsymbol{x}_o^{\mathrm{OV}} = [p_{X,o}^{\mathrm{OV}}\ p_{Y,o}^{\mathrm{OV}}\ \psi_o^{\mathrm{OV}}\ v_{x,o}^{\mathrm{OV}}]^{\mathrm{T}}.$$

The spatial extent of the EV and the OVs, as well as measured positions and velocity at the beginning of the planning phase, result in additional time-varying constraints on the states of the EV. We define the (deterministic or probabilistic) obstacle set at time step k as $\mathcal{O}_k(\boldsymbol{x}_{o,0}^{\mathrm{OV}}, \mathcal{B}_o)$. Denote the planning horizon with T_f. The predicted set of the oth obstacle for each $k \in [0, T_f]$ is

$$\mathcal{S}_{k,o} = \mathcal{O}_k(\boldsymbol{x}_o^{\mathrm{OV}}, \mathcal{B}_j). \tag{7}$$

The collision-avoidance area at time index k is computed as the union over all OV trajectory sets (7),

$$\mathcal{S}_k = \bigcup_{o=1}^{M} \mathcal{S}_{k,o}. \tag{8}$$

C. Driving Requirements

The method we propose is based on the understanding that nominal driving requirements can be determined beforehand. For instance, when driving on a structured road network a set of driving requirements may be:

- Stay on the road
- Obey right-hand traffic rules
- Maintain a certain velocity (e.g., from the speed limits)
- Stay in the middle of a certain lane
- Drive smoothly, that is, prioritize small steer rates
- Keep safety distance to surrounding obstacles

The driving requirements for each time step k can be summarized in the vector $\boldsymbol{y}_k \in \mathbb{R}^{n_y}$. The driving requirements will be dependent on the different modes in the mode set $\mathcal{M}$. For instance, the modes CLL and CLR will result in different desired lanes to follow. The driving requirements can directly correspond to some or all of the states, and typically the requirements are not possible to fulfill exactly. For instance, it might be impossible to maintain a certain velocity while keeping a specified safety distance to other vehicles. In this paper, we model the driving requirements assuming that we want to maintain a (possibly time varying) nominal velocity v_{nom}, be positioned in the middle of the lane corresponding to the driving mode, that is, to have zero deviation from the middle of the lane, and ideally keep the distance larger than $d_{\min}$ from the surrounding vehicles.

Hence, for M obstacles in the ROI of the EV,

$$\boldsymbol{y}_k = \begin{bmatrix} v_{\mathrm{nom}} & 0 & g_1 & \cdots & g_M \end{bmatrix}^{\mathrm{T}}, \tag{9}$$

where

$$g_o = \begin{cases} 0 & \text{if } d_o > d_{\min}, \\ f(d_{\min} - d_o) & \text{if } d_o \leq d_{\min}, \end{cases} \tag{10}$$

in which d_o is the distance between the EV and the oth OV and $g_o(\cdot)$ is a monotonically increasing function.

The resulting trajectory generated by the motion planner will not exactly track $\boldsymbol{y}_k$, due to, for instance, conflicting requirements, input constraints, the vehicle kinematics limiting the drivable space, sensing and modeling errors, or limited computing time. The driving requirements are modeled as constraints on the vehicle states as

$$\hat{\boldsymbol{y}}_k = \boldsymbol{h}(\boldsymbol{x}_k, m_j, \mathcal{S}_k, \boldsymbol{x}^{\mathrm{RD}}), \tag{11}$$

where $\boldsymbol{h}$ is a nonlinear function relating the EV state $\boldsymbol{x}_k$, mode m_j, OV obstacle set $\mathcal{S}_k$ (hence also $\{\boldsymbol{x}^{\mathrm{OV}}\}_{o=1}^{M}$), and road information $\boldsymbol{x}^{\mathrm{RD}}$, to the driving requirements. For the driving requirements in (9), (11) can be written as

$$\boldsymbol{h}(\boldsymbol{x}_k, m_j, \mathcal{S}_k, \boldsymbol{x}^{\mathrm{RD}}) = \begin{bmatrix} v_x & p_e & d_1 & \cdots & d_M \end{bmatrix}, \tag{12}$$

where p_e is the lateral deviation from the middle of the lane in the road-aligned frame corresponding to the driving mode. Fig. 2 shows the driving requirements and the notation.

IV. Problem Statement

This paper focuses on the motion-planning problem on road networks, such as highway driving or city driving. In our proposed approach, we consider a discrete-time vehicle model of the EV that can be written in the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{g}(\boldsymbol{x}_k)\boldsymbol{w}_k, \tag{13}$$

where $\boldsymbol{f} \in \mathbb{R}^{n_x}$ and $\boldsymbol{g} \in \mathbb{R}^{n_x \times n_u}$ in general are nonlinear functions, and $\boldsymbol{w}_k \in \mathbb{R}^{n_u}$ is the input (process) disturbance. The input disturbance is Gaussian distributed according to $\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k)$, where $\boldsymbol{Q}_k$ is the covariance matrix. Compared to the deterministic vehicle model (1), we use an input disturbance, which is used to generate the control input in our approach as will be explained later, instead of directly using the deterministic control input. In general, the input disturbance can depend on both the EV state $\boldsymbol{x}_k$ and the driving mode m_j, but we do not make that dependence explicit

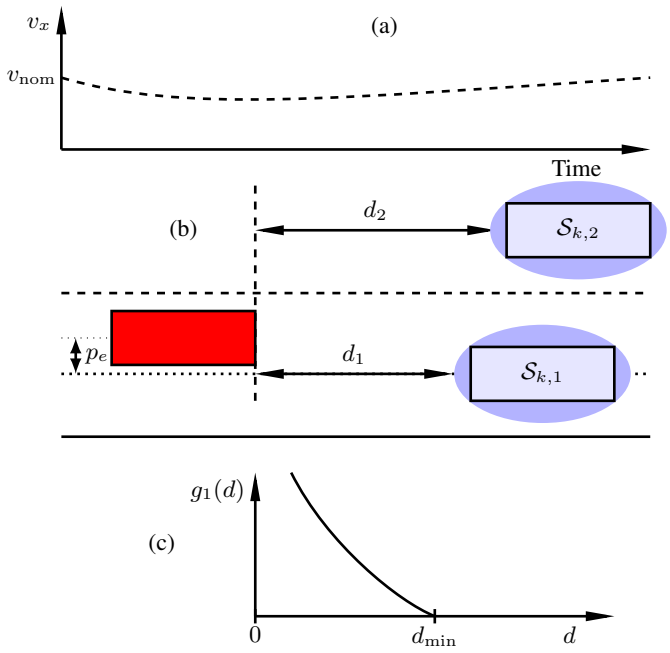

Fig. 2. Illustration of the driving requirements used in this paper; (a) nominal velocity profile v_{nom} requirement, which in general can be time varying; (b) mid-lane tracking requirement p_e and safety distance d_j to OV requirements; (c) possible shape of $g_j(\cdot)$ for separation distance requirement. In the figure the obstacle sets are modeled as ellipsoidal sets obtained, for example, from a Kalman filter, including uncertainty around the spatial extent of the rectangular OVs.

in the following. We model the EV behavior with respect to the driving requirements as

$$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, m_j, \mathcal{S}_k, \boldsymbol{x}^{\text{RD}}) + \boldsymbol{e}_k, \tag{14}$$

where $\boldsymbol{e}_k \in \mathbb{R}^{n_e}$ is the slack, which results in the probabilistic cost, on the driving requirements. We model $\boldsymbol{e}_k$ as a stochastic Gaussian disturbance with covariance $\boldsymbol{R}_k$ that can be dependent on the vehicle and driving mode. Compared to the ideal driving requirements (9), we have added the noise term $\boldsymbol{e}_k$. Eq. (14) models the outputs of the system (13) on which driving requirements are imposed. The term $\boldsymbol{e}_k$ is important for several reasons. For example, due to sensor noise from estimation algorithms responsible for the vehicle state and road map and to avoid infeasibility in trying to fulfill all driving requirements exactly. For convenience, we define

$$\mathcal{Z}_k = \{\boldsymbol{x}^{\text{RD}}, \mathcal{S}_k\} \tag{15a}$$
$$\mathcal{Z}_k^{\text{G}} = \{\boldsymbol{x}_k, \mathcal{Z}_k\}, \tag{15b}$$

where (15a) groups the external time-varying entities and (15b) groups the global time-varying information. In a Bayesian framework, using the notation (15), (13) and (14) can be reformulated as

$$\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k), \tag{16a}$$
$$\boldsymbol{y}_k \sim p(\boldsymbol{y}_k|\mathcal{Z}_k^{\text{G}}, m_j), \tag{16b}$$

where $\boldsymbol{x}_{k+1}$ and $\boldsymbol{y}_k$ are regarded as samples from the respective distributions.

Given the vehicle dynamics (1), the goal of the motion-planning method is to generate an input trajectory $\boldsymbol{u}_k$, $k \in [0, T_f]$ over the planning horizon T_f satisfying the input constraints (5) such that the resulting trajectory obtained from (1) obeys (6), avoids the obstacle set (8), and reaches the goal region, that is, $\boldsymbol{x}_{T_f} \in \mathcal{X}_{\text{goal}}$, where goal region $\mathcal{X}_{\text{goal}}$ is assumed given by a higher-level route planner.

V. Decision Making and Motion Planning Design

In this section we present the method for decision making and motion planning. First, we show how PF can be used to simultaneously generate decisions and corresponding collision-free dynamically feasible motion plans. Then, we outline the full algorithm for integrated decision making and motion planning formulated as a tree-expansion algorithm.

A. Particle Filtering with Discrete and Continuous States

According to Sec. IV, the aim is to determine an input trajectory and corresponding motion plan over the planning horizon T_f that navigates the road safely while satisfying input constraints (5), road constraints (6), and obstacle constraints (8). In addition, we want to minimize deviations from the predefined driving requirements (9).

In determining what combination of driving mode and state trajectory is preferable, we assume that the driving mode (3) is sampled less frequently than the states. That is, for a given sample m_j we execute a PF for a predefined time T. The main idea in the approach is that we determine the state trajectory PDF $p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, m_j, \mathcal{Z}^{\text{G}})$, conditioned on the driving requirements $\boldsymbol{y}_{0:T}$, the driving mode m_j, and the global information as a finite weighted sum over the planning horizon, and then extract the trajectory from the PDF. By doing this iteratively, we construct a trajectory $\boldsymbol{x}_{0:T_f}$ based on the driving requirements and modes. In our approach, the driving requirements are the equivalent of sensor measurements in a traditional estimation problem.

According to the PF approach, we approximate the state trajectory PDF by a set of N particles $\boldsymbol{x}_{0:T}^i$ and their associated importance weights q_T^i as

$$p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, m_j, \mathcal{Z}_{k-1}^{\text{G}}) \approx \sum_{i=1}^{N} q_T^i \delta(\boldsymbol{x}_{0:T} - \boldsymbol{x}_{0:T}^i), \tag{17}$$

where q_T^i in (17) is the importance weight for the ith particle and $\delta(\cdot)$ is the Dirac delta mass. To propagate the particles forward in time, at any time step k in the planning phase, the particle-filter approach generates N samples $\boldsymbol{x}_k^i$ from a proposal density $\pi(\cdot)$,

$$\boldsymbol{x}_k \sim \pi(\boldsymbol{x}_k|, \boldsymbol{y}_k, m_j, \mathcal{Z}_k) \tag{18}$$

by using the particles from the previous time step $k-1$. After sampling the state at time index k, which amounts to the prediction step, the measurement step consists of updating the weights q_k^i according to

$$q_k^i \propto \frac{p(\boldsymbol{y}_k|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_k) p(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i)}{\pi(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_k, m_j, \mathcal{Z}_k)} q_{k-1}^i, \tag{19}$$

where $p(\boldsymbol{y}_k|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_k)$ is the likelihood (16b) and $p(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i)$ is the vehicle model (16a). It is natural to choose the vehicle model (16a) as proposal, that is,

$$\pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_k) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i). \tag{20}$$

This reduces the weight update (19) to

$$q_k^i \propto p(\boldsymbol{y}_k|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_k) q_{k-1}^i. \tag{21}$$

However, using only the prediction model to construct the proposal density leads to inefficient exploration of the state space, since the information about the driving requirements and the driving mode are only used in the weight update, when the particles have already been generated. This would lead to a large spread of the particles, which implies that the particles are not used efficiently.

To maximize the usage of each particle when planning the motion, we incorporate the driving requirements and driving mode when choosing the inputs to the system. Since the driving requirements $\boldsymbol{y}_{0:T_f}$ and the predicted obstacle set $\mathcal{S}_{T_f}$ are known beforehand, we can even use requirements $\boldsymbol{y}_s$, where $k \leq s \leq T$, to generate samples $\boldsymbol{x}_k^i$ by choosing as proposal density

$$\pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_s, m_j, \mathcal{Z}_s) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_s, m_j, \mathcal{Z}_s). \tag{22}$$

The proposal (22) leads to the modified weight update

$$q_k^i \propto \frac{p(\boldsymbol{y}_s|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_s) p(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i)}{p(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_s, m_j, \mathcal{Z}_s)} q_{k-1}^i, \tag{23}$$

Inserting (22) into (23) and using

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_s, m_j, \mathcal{Z}_s) = \frac{p(\boldsymbol{y}_s|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_s) p(\boldsymbol{x}_k^i|\boldsymbol{x}_{k-1}^i)}{p(\boldsymbol{y}_s|\boldsymbol{x}_{k-1}^i, m_j, \mathcal{Z}_s)}, \tag{24}$$

where the likelihood is obtained from

$$p(\boldsymbol{y}_s|\boldsymbol{x}_k^i, m_j, \mathcal{Z}_s) = \int p(\boldsymbol{y}_s|\boldsymbol{x}_s, m_j, \mathcal{Z}_s) p(\boldsymbol{x}_s|\boldsymbol{x}_k^i) \, \mathrm{d}\boldsymbol{x}_s, \tag{25}$$

leads to the weight update

$$q_k^i \propto p(\boldsymbol{y}_s|\boldsymbol{x}_{k-1}^i, m_j, \mathcal{Z}_s) q_{k-1}^i. \tag{26}$$

With the proposal (22) we are generating particles using a fixed-point smoothing step with smoothing (prediction) time t_s, that is, we are leveraging knowledge about the driving requirements in the future when determining the motion plan for the entire planning horizon. Eq. (26) indicates that the weight update is independent of the sample $\boldsymbol{x}_k^i$, that is, (22) is optimal in the sense that it maximizes the effective number of samples. On the other hand, it is generally difficult to sample from (22) exactly.

If the driving requirements are encoded in such a way that the function $\boldsymbol{h}(m_j, \mathcal{Z}_k^{\mathrm{G}})$ in (14) is differentiable, which is the case for this paper as long as $d_o \neq d_{\min}$ in (10), a first-order approximation of the optimal proposal can be used as follows. The proposal (22) is in a recursive estimation context equivalent to a measurement update using the measurement $\boldsymbol{y}_s$. At time $k-1$, each particle is by itself a state estimate with no uncertainty, but with a propagation covariance $\boldsymbol{Q}_{k-1}$. Hence, using a first-order approximation leads to an extended Kalman filter update as

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^i, \boldsymbol{y}_s, m_j, \mathcal{Z}_s) \approx \mathcal{N}\left(\boldsymbol{x}_k|\hat{\boldsymbol{x}}_k^i, (\boldsymbol{\Sigma}_k^i)^{-1}\right), \tag{27}$$

where

$$\begin{aligned}
\hat{\boldsymbol{x}}_k^i &= \boldsymbol{f}(\boldsymbol{x}_{k-1}^i) + \boldsymbol{K}_s^i(\boldsymbol{y}_s - \hat{\boldsymbol{y}}_s^i), \\
\boldsymbol{\Sigma}_s^i &= \left({\boldsymbol{H}_s^i}^{\mathrm{T}} \boldsymbol{R}_s^{-1} \boldsymbol{H}_s^i + \boldsymbol{Q}_{s-1}^{-1}\right)^{-1}, \\
\boldsymbol{K}_s^i &= \boldsymbol{Q}_{s-1} {\boldsymbol{H}_s^i}^{\mathrm{T}} (\boldsymbol{H}_s^i \boldsymbol{Q}_{s-1} {\boldsymbol{H}_s^i}^{\mathrm{T}} + \boldsymbol{R}_s)^{-1}, \\
\hat{\boldsymbol{y}}_s^i &= \boldsymbol{h}(\hat{\boldsymbol{x}}_s^i, m_j, \mathcal{Z}_s), \quad \boldsymbol{H}_s^i = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_s^i},
\end{aligned} \tag{28}$$

and $\hat{\boldsymbol{x}}_s^i$ is the prediction using the motion model (1) using zero input of particle i from time index $k-1$ to s. Therefore, the control input for each particle is

$$\boldsymbol{u}_k^i = \boldsymbol{K}_s^i(\boldsymbol{y}_s - \hat{\boldsymbol{y}}_s^i) + \boldsymbol{\sigma}_s^i, \tag{29}$$

where $\boldsymbol{\sigma}_s^i \sim \mathcal{N}(\boldsymbol{0}, (\boldsymbol{\Sigma}_s^i)^{-1})$. Furthermore, by setting

$$\boldsymbol{Q}_k = \boldsymbol{B} \bar{\boldsymbol{Q}}_k \boldsymbol{B}^{\mathrm{T}}, \tag{30}$$

we can design $\boldsymbol{Q}_k$ such that the control inputs only act on the longitudinal acceleration and steering rate, thereby ensuring dynamic feasibility. The likelihood in (26) is approximated as

$$p(\boldsymbol{y}_s|\boldsymbol{x}_{k-1}^i, m_j, \mathcal{Z}_s) \approx \mathcal{N}\left(\boldsymbol{y}_s|\hat{\boldsymbol{y}}_s^i, \boldsymbol{H}_s^i \boldsymbol{Q}_{s-1} (\boldsymbol{H}_s^i)^{\mathrm{T}} + \boldsymbol{R}_s\right). \tag{31}$$

The proposal density (27) will make the particles tend to the driving requirements, and if $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are chosen wisely, the risk of violating the road constraints (6) and enter the obstacle set (8) will be kept to a minimum. The reason is that the weights (26) corresponding to the particles that violate, or are close to violating, the constraints, will become small and therefore be rejected in the resampling procedure of the particle filter. However, this only provides statistical guarantees on constraint satisfaction. Hence, to make sure we do not violate the road constraints and enter the obstacle set, we use the modified weight update

$$q_k^i = \begin{cases} p(\boldsymbol{y}_s|\boldsymbol{x}_{k-1}^i, m_j, \mathcal{Z}_s) q_{k-1}^i & \text{if (6) and (8) are satisfied} \\ 0 & \text{if (6) or (8) are violated} \end{cases}. \tag{32}$$

With the weight update (32), which in practice performs collision and out-of-road checking, we ensure there is no risk of violating the road constraints and entering the obstacle set, provided at any time step there is at least one out of the N particles that satisfy (6) and (8).

Using the PF update equations (27)–(32) results in a more efficient exploration of the state space than an input-based RRT. As an example of this, Fig. 3 shows two snapshots of samples generated by the PF updates (upper plot) and a kinodynamic RRT (lower plot), when the EV (red) tries to maintain the right lane of a two-lane road. The samples are more scattered with the kinodynamic RRT, but by using the PF updates, we can direct the samples toward a particular lane.

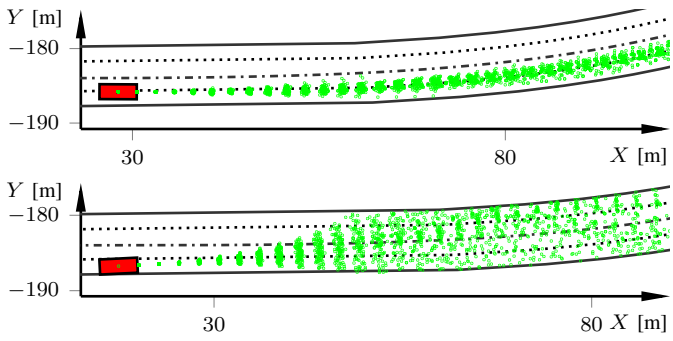

Fig. 3. Snapshots of generated samples (green) for motion planning using the proposed PF-based sample generation (upper) and a kinodynamic RRT (lower).

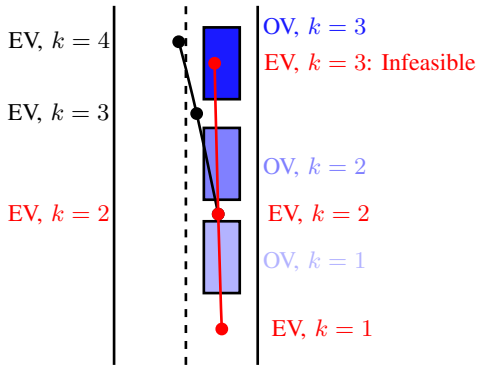

Fig. 4. The incorporation of time in the tree expansion enables reuse of nodes in the tree. For simplicity the EV is modeled as a point mass in the figure. The tree splits into two branches at $k = 2$. The node corresponding to $k = 2$ intersects with an OV, but since the OV is associated with $k = 1$, the node is collision free. However, the red node at $k = 3$ intersects with an OV associated with $k = 3$, which implies that a collision is detected and this branch is not extended beyond $k = 2$, as opposed to the branch in black.

B. Determining the Tree Expansion

To allow warm-starting the motion planner, we incrementally expand a tree $\mathcal{T}$ in the following way. At the first iteration of the algorithm when the tree is empty, we recursively execute the PF equations (27)–(32) to get (17). Storing the complete representation of the PDF over the planning horizon would imply a large memory requirement, which is problematic given the limited memory capabilities of automotive microcontrollers. Instead, we extract the trajectory and corresponding inputs from (17) by using, for instance, the minimum mean-square estimate

$$\boldsymbol{x}_{0:T} = \sum_{i=1}^{N} q_T^i \boldsymbol{x}_{0:T}^i, \tag{33a}$$

$$\boldsymbol{u}_{0:T-1} = \sum_{i=1}^{N} q_T^i \boldsymbol{u}_{0:T-1}^i. \tag{33b}$$

The tree is appended with the state trajectory (33a) as vertices $\mathcal{V}$ and the corresponding input transitions (33b) as edges $\mathcal{E}$. In the next iteration, we start from a state corresponding to a vertex in the tree and again execute the PF. Since the time is incorporated into every vertex in the tree, as long as the obstacle prediction (8) is reliable the tree can be reused in the next planning cycle and there is no need for the reevaluation of nodes, which is computationally heavy when replanning in dynamic environments [28]. Each vertex also contains a timestamp and a cost C for reaching that node. As illustrated in Fig. 4, the proposed method not only checks whether an intersection with an OV occurs, but also at which time. This is due to the way we construct the tree using the particle filter. At the end of the tree expansion, the lowest-cost trajectory in terms of the cost C is chosen for execution.

The proposed method is similar to RRT in that it builds a tree of vertices and edges, but there are several differences. The driving modes (2) and the subsequent sampling according to (3) ensure that safe stopping is incorporated; samples are drawn according to a proposal density (27), which generates dynamically feasible trajectories (33) consistent with the motion model of the vehicle; and dynamic collision avoidance is due to the PF based motion planning ensured by (32). Note that in practice we generate the state trajectory (33a) by first generating control inputs according to (29), which are then used to simulate the system forward in time to obtain the state trajectories. Hence, our approach avoids the need of steering control laws for connecting the states as usually done in RRT, which is complicated for systems with nonholonomic kinematic constraints. Furthermore, the input space is typically of much lower dimension than the state space, implying reduced computational burden for our method when compared to sampling the state space.

C. Implementation Aspects

Because of sensing errors and unpredicted changes in the environment, for instance due to new obstacles (7) entering the ROI, we implement the motion planner in a receding-horizon strategy. That is, the computed trajectory is T_f long but is only applied for $\Delta t \leq T_f$, and the maximum allowed (allocated) computation time for finding the motion plan is δt. Fig. 5 illustrates how the tree is reused between planning iterations. Similar to [28], we keep a committed tree, which is the part of the tree that will be executed. In the beginning of a planning phase, the measured EV position is obtained, and the EV position over the allocated computation time δt is predicted, compared, and matched with a node being the closest node in the tree. This node becomes the root node of the planning phase, and the part of the tree that is not a descendant of the end node is deleted.

As for the driving mode, we sample and connect it to nodes corresponding to the same driving mode, to avoid several driving-mode changes in one planning phase. That is, when sampling the driving mode m_j, we start the planning from a vertex $\mathcal{V}$ in the tree with the same driving mode m_j.

The vehicle dynamics is discretized assuming a sampling time T_s, which is typically determined by the update rate of the sensor and/or the available computing power. The covariance matrices $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are two important design parameters. Typically, $\boldsymbol{R}_k$ is determined from insights on which driving requirements are most important, whereas $\boldsymbol{Q}_k$ can be tuned for a given $\boldsymbol{R}_k$. However, it is also possible to learn the parameters from recorded driving data. Here, recent advancements in

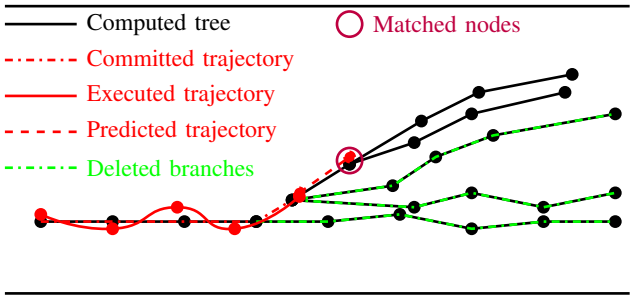

Fig. 5. A part of the generated tree is committed for execution. In the next iteration, the measured EV state is used to account for the planner computation time, by matching the predicted node to a node in the tree. The branches that do not originate from the updated root node are deleted.

particle Markov Chain Monte Carlo methods seem particularly interesting to explore, since the proposed motion planner can be easily incorporated in such a learning framework [34], [35].

D. Algorithm Summary and Properties

Algorithm 1 describes the planner and the PF-based exploration is given in Algorithm 2. When the computation time exceeds δt, the safe trajectory with lowest accumulated cost C is chosen for execution (Line 15, Algorithm 1).

Algorithm 1 Proposed Planning Method

1: **Input:** State estimate $\hat{\boldsymbol{x}}$, goal region $\mathcal{X}_{\text{goal}}$, tree $\mathcal{T}$.
2: Propagate $\hat{\boldsymbol{x}}$ with the allocated time slot δt.
3: Set root node of $\mathcal{T}$ corresponding to $\hat{\boldsymbol{x}}$.
4: Delete part of $\mathcal{T}$ that is not a descendant of the root node.
5: Update obstacle set (8) and road constraint (6) to compute allowed region $\mathcal{X}_{\text{free}}$.
6: Set $t_{\text{CPU}} \leftarrow 0$
7: **while** $t_{\text{CPU}} \leq \delta t$ **do**
8: Generate driving mode m_j from (3).
9: Determine $\{\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T-1}\}$ using Algorithm 2.
10: **if** $\boldsymbol{x}_{0:T}$ is obstacle free **then**
11: Add $\boldsymbol{x}_{0:T}$ as vertices $\mathcal{V}_{\text{new}}$ to $\mathcal{T}$.
12: Add $\boldsymbol{u}_{0:T-1}$ as edges $\mathcal{E}_{\text{new}}$ to $\mathcal{T}$.
13: **end if**
14: **end while**
15: Determine lowest-cost safe state trajectory $\boldsymbol{x}_{\text{best}}$ and corresponding controls $\boldsymbol{u}_{\text{best}}$.
16: Apply $\{\boldsymbol{x}_{\text{best}}, \boldsymbol{u}_{\text{best}}\}$ for time Δt and repeat from Line 1.

In terms of algorithm properties, the key question in PFs is how well a function $g(\boldsymbol{x}_k)$ can be approximated by $\hat{g}(\boldsymbol{x}_k)$ compared to the conditional expectation $\mathbb{E}(g(\boldsymbol{x}_k))$, where

$$\mathbb{E}(g(\boldsymbol{x}_k)) = \int g(\boldsymbol{x}_k) p(\boldsymbol{x}_{1:k}|\boldsymbol{y}_{1:k}) \, \mathrm{d}\boldsymbol{x}_{1:k}, \tag{34a}$$

$$\hat{g}(\boldsymbol{x}_k) = \sum_{i=1}^{N} q_k^i g(\boldsymbol{x}_k^i). \tag{34b}$$

There are several convergence proofs of PFs [5], [36], for instance, almost sure weak convergence,

$$\lim_{N\to\infty} \hat{p}(\boldsymbol{x}_{0:k}|\boldsymbol{y}_{0:k}) = p(\boldsymbol{x}_{0:k}|\boldsymbol{y}_{0:k}), \tag{35}$$

Algorithm 2 Particle Filter for Trajectory Generation

Input: Propagated state $\hat{\boldsymbol{x}}$, driving mode m_j, and $\mathcal{T}$.
1: Choose a feasible node in the tree consistent with m_j and extract state $\boldsymbol{x}_0$.
2: Set $\{x_{-1}^i\}_{i=1}^N \leftarrow x_0$, $\{w_{-1}^i\}_{i=1}^N \leftarrow 1/N$,
3: **for** $k \leftarrow 0$ to T **do**
4: **for** $i \leftarrow 1$ to N **do**
5: Generate state $\boldsymbol{x}_k^i$ and input $\boldsymbol{u}_k^i$ using (27).
6: Update weight $\bar{q}_k^i$ using (31).
7: **end for**
8: **if** $\sum_{i=1}^N \bar{q}_k^i = 0$ **then**
9: Terminate and return to Algorithm 1, Line 7.
10: **end if**
11: Normalize: $q_k^i \leftarrow \bar{q}_k^i / \sum_{j=1}^N \bar{q}_k^j$
12: Set $N_{\text{eff}} \leftarrow 1/(\sum_{i=1}^N (q_k^i)^2)$
13: **if** $N_{\text{eff}} \leq \gamma N$ **then**
14: Resample particles with replacement.
15: Set $q_k^i \leftarrow 1/N, \quad \forall i \in \{1, \ldots, N\}$.
16: **end if**
17: **end for**
18: Extract trajectory and inputs according to (33).
Return: $\{x_{0:T}, u_{0:T-1}\}$

in the sense that $\lim_{N\to\infty} \hat{g}(\boldsymbol{x}_k) = \mathbb{E}(g(\boldsymbol{x}_k))$. As a consequence, if the driving requirements are modeled in such a way that the PDF of the state trajectory conditioned on the driving requirements and accounting for the obstacle set (8), $p(\boldsymbol{x}_{0:k}|\boldsymbol{y}_{0:k})$, is nonzero for all $k \in [0, T]$ and reaches the target region $\mathcal{X}_{\text{goal}}$, the PF approximation (33a) due to the weight update (26) will be collision free, and will reach the target region $\mathcal{X}_{\text{goal}}$. The method lacks optimality in the traditional RRT sense. However, it finds the minimum mean-square estimate (33a) of the state trajectory contained in the resulting approximate PDF, which from (35) is asymptotically optimal.

VI. SIMULATION STUDY

In this section we evaluate the proposed decision maker and motion planner in simulation for different relevant scenarios.

A. Problem Setup

In the simulation study, an autonomous vehicle travels on a one-way two-lane road. The road includes both straight-line and curved road segments. The road coordinates are data from the outer ring test track of the Japanese Automobile Research Institute proving ground in Shirosato, Japan, and the vehicle parameters used in the simulation study are obtained from a real mid-size SUV, from data-sheet, precision testbenches, and experimentally recorded data analysis. There are surrounding vehicles maintaining either of the lanes with constant velocity. In the simulation, the obstacle set is predicted based on an OV vehicle model in closed-loop with a lane-tracking controller that controls the OVs assuming a fixed lane over the planning horizon T_f.

The goal region $\mathcal{X}_{\text{goal}}$ is chosen such that a trajectory is considered to have reached the region if the trajectory

TABLE II
THE PARAMETER CHOICES FOR THE SIMULATION STUDY.

Parameter	Value	Meaning
N	50	# particles
Δt [s]	1	Execution time
δt [s]	0.1	Computation time
T_s [s]	0.1	Discretization time
T_f [s]	5	Planning horizon
T [steps]	T_f/T_s	Prediction time
t_s [s]	1	Smoothing time

is collision free and at least T_f long. The road boundary constraint (6) is determined by linear interpolation of the data points of the road, and checking for feasibility of (6) amounts to checking if the path, with the vehicle geometry taken into account, intersects any point on the interpolated lines. The driving modes are sampled in the following way: First, we iterate over all possible driving modes in $\mathcal{M}$ to ensure that we exhaustively test all possibilities. Then, we sample the modes with a uniform probability distribution. Because we test the method on a two-lane road, only one of the options CLL and CLR are possible when located in a given lane.

The different parameters used in the planner are shown in Table II. Algorithm 1 is implemented in MATLAB on a 2014 i5 laptop, with Algorithm 2, where the main bulk of the computations reside, implemented via C-coded `mex`-functions. The PF prediction horizon T is dependent on the timestamp of the node it expands from, to not predict too far ahead in the future, and the nominal value is shown in Table II. Algorithm 1 provides a feasible trajectory faster than real time (i.e., in less than δt s) for the considered scenario and parameters.

Driving Requirements: The driving requirements are the same as introduced in (9), that is, (i) track the reference velocity v_{nom}; (ii) follow the middle lane, where the lane is determined from the driving mode; and (iii) maintain a minimum distance $d_{\min}$, to vehicles in the same lane. The minimum distance requirement is activated whenever the velocity at the beginning of a planning phase will lead to a safety distance smaller than d_s m in less than δt s. We assume left-hand traffic, and the cost function C for each node is a combination of the norm of the mid-lane error from the left lane, the distance to obstacles, and deviations from the reference velocity.

B. Results

We illustrate the method on two scenarios. The first illustrative example concerns a situation where there is a slower moving vehicle ahead of the EV. The second example concerns a situation when the road is blocked, and the motion planner must slow down to avoid a collision.

1) Overtaking: In the overtaking situation, the desired velocity is set to $v_{\text{nom}} = 25$ m/s and the minimum distance corresponds to 3 s headway time, $d_{\min} = 3v_x$. However, because of the smoothing time t_s used in the particle prediction (22), the vehicle will start to slow down before the minimum distance is reached.

Figs. 6 and 7 show snapshots of a situation where the autonomous vehicle (red) catches up with a vehicle (blue)

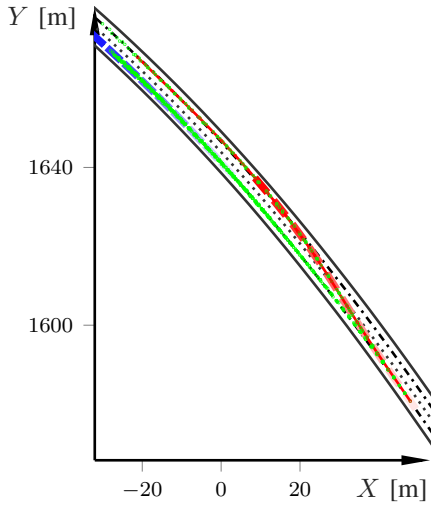

Fig. 6. A situation where the EV (red) must decide whether to switch lane or to stay behind the obstacle (blue). The particles generated in the planning phase are in green, and the chosen nodes are indicated in red. Snapshots of the EV and OV trajectories every third discretization step are shown in increasingly darker color.

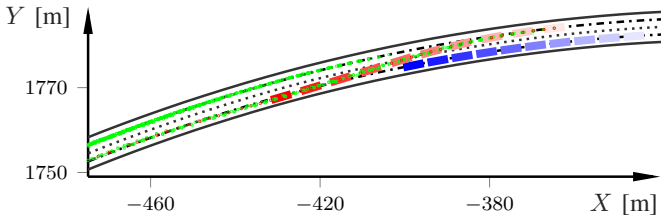

Fig. 7. An overtaking situation, see Fig. 6. The vehicle reenters the left lane and concludes the overtaking maneuver.

and eventually overtakes it. The particles generated within one planning phase are shown in green, and the lowest-cost trajectory is in red. There are two sets of trajectories computed, one where the vehicle maintains the left lane and one where the autonomous vehicle initiates overtaking of the slower moving vehicle. These two sets correspond to driving mode m_2 (stay in lane), m_3 (change lane left), and m_4 (change lane right). In Fig. 6, to stay in the left lane would demand a decreased velocity, which is penalized in the cost function. Hence, the motion planner prefers to change lane, given that it can maintain a velocity close to the speed limit. In Fig. 7, the EV reenters the left lane since that lane is again obstacle free.

The corresponding determined velocity profiles are shown in Fig. 8. In this figure there is a tendency to compute velocities below the reference velocity. This will usually be the case unless the value in the covariance matrix $\boldsymbol{R}_k$ related to the reference velocity is set to a very small number. The reason is that is usually easier to closely track the middle lane if the velocity is decreased. Since in our approach the particle filter computes the asymptotically optimal PDF, the minimum mean-square estimate (33) is extracted from that PDF, and the mass of the PDF will be slightly concentrated to a lower velocity, the result is the observed behavior.

2) Blocked Lanes: In this driving situation the road is blocked by two slower moving vehicles driving at approx-

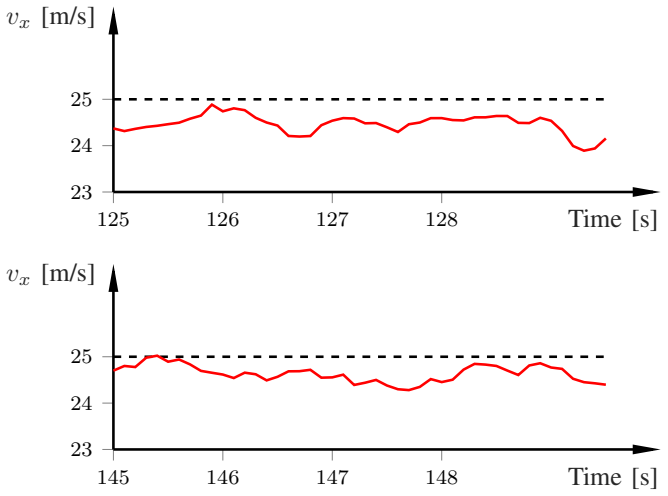

Fig. 8. Planned velocities (red) corresponding to Fig. 6 (top) and Fig. 7 (bottom) with $v_{\text{nom}} = 25$ m/s (dashed black).

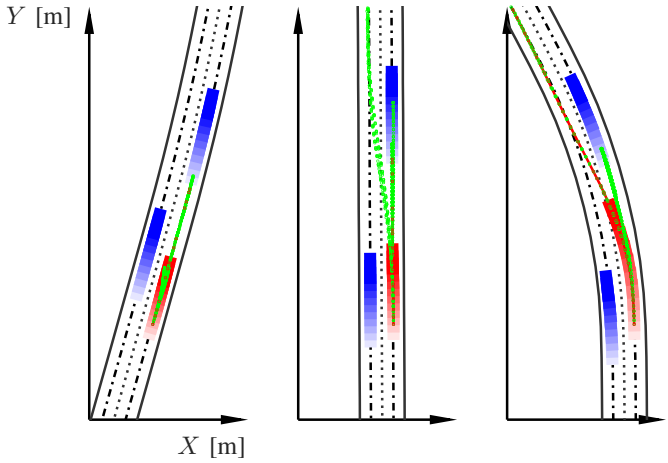

Fig. 9. Snapshots when both lanes are blocked by slow vehicles, so the only possibility is to slow down. The left plot corresponds to 90 s, the middle plot to 125 s, and the right plot to 135 s in Fig. 10. See Fig. 6 for notation.

imately 20 km/h, one in each lane, so the motion planner must decrease the velocity and wait for an opening to become available to overtake the vehicles. In this scenario, we set the desired velocity to $v_{\text{nom}} = 13.89$ m/s, corresponding to 50 km/h and the minimum distance corresponds to 3 s, $d_{\min} = 3v_x$. However, because of the smoothing time t_s used in the particle prediction (22), the vehicle starts slowing down before the minimum distance is reached.

Fig. 9 displays three snapshots of the situation and Fig. 10 shows the planned velocity profile. For Fig. 9, in the left plot there is no safe path to change lane. In the middle plot it is possible to change lane, which would imply better tracking of the reference velocity. However, the motion planner considers the risk, encoded in the cost function, too high so it chooses to maintain the right lane. In the right plot, there is enough space between the two cars to safely overtake the vehicle in the right lane. The right plot corresponds to the time instant marked lane change in Fig. 10.

In the simulations we set the allocated computation time to 0.1 s. However, this choice was quite arbitrary and not limited

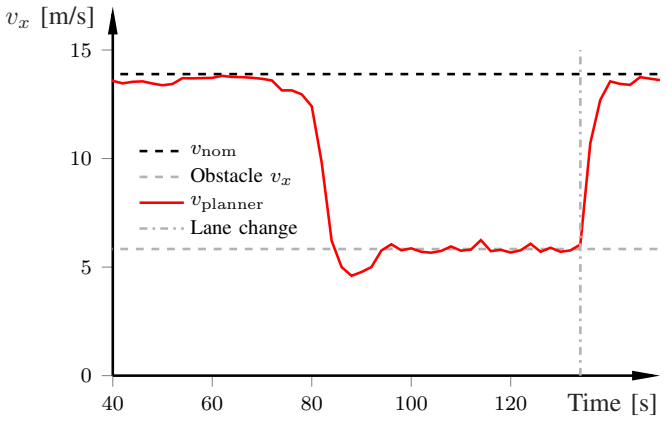

Fig. 10. Planned velocity (red) for the blocked-lane scenario in Fig. 9. Both lanes are blocked by slow vehicles, so the only possibility is to slow down and maintain the velocity of the leading OV until an opening appears. A safety margin 3 s is imposed.

TABLE III
AVERAGE SOLUTION TIME, NUMBER OF NODES THAT CAN BE GENERATED IN THE ALLOCATED COMPUTATION TIME, AND TIME FOR GENERATING A MINIMUM REQUIRED NUMBER OF NODES, FOR 100 ITERATIONS.

N	Mean CPU time	# nodes	50 nodes
20	0.032 s	160	0.029 s
50	0.037 s	115	0.035 s
100	0.051 s	87	0.044 s
200	0.068 s	61	0.073 s
500	0.145 s	35	0.154 s
1000	0.276 s	30	0.234 s

by the required computation time for the proposed method using the parameters in Table II. We conclude the simulation study with a computation-time analysis. To generate a motion that lasts for T_f s with a discretization time T_s s, a minimum of $T_f/T_s = 50$ nodes need to be generated. Table III displays the average computation time for finding a solution to the motion-planning problem when using the parameters in Table II with a varying number of particles. Also shown are the number of nodes that can be generated in the maximum allowed computation time $\delta t = 0.1$ s with a varying number of particles, and how long time it takes to generate the minimum required number of nodes (for the parameter setup in Table II) with a varying number of particles. When analyzing the results, it should be remembered that what we report is an over-estimate of the algorithm execution time, due to the overhead in the MATLAB interpreter, the overhead introduced by the context switch and the transfer of the variables from MATLAB to C, and due to the overhead introduced by the time-measuring function (`tic-toc`). Nevertheless, Table III indicates that the proposed method can indeed generate a feasible trajectory within the allocated computation time slot, and that $N = 50$ is not an upper limit. Note also that the method in the presented examples finds valid solutions also when $N < 20$.

VII. EXPERIMENTAL RESULTS

In this section we evaluate the proposed motion planner on an experimental setup. We use the Hamster platform [37] in

Fig. 11. The Ackermann-steered mobile robot used in the experiments. A camera from the Optitrack motion-capture system is in the top right.

TABLE IV
THE PARAMETER CHOICES FOR THE EXPERIMENTAL STUDY.

Parameter	Unit	Value	Meaning
N	-	100	# particles
Δt	s	0.6	Execution time
δt	s	0.1	Computation time
T_s	s	0.3	Discretization time
T_f	s	5	Planning horizon
T	-	T_f/T_s	Prediction time
t_s	s	1.8	Smoothing time
$\delta_{\max}$	deg	15	Maximum steering angle
$\dot{\delta}_{\max}$	deg/s	-10.5	Maximum steering rate
$\dot{v}_{x,\max}$	m/s^2	0.2	Maximum acceleration

combination with an Optitrack motion-capture system [38] to track the robot, see Fig. 11. The Hamster is a 25×20 cm mobile robot for research and prototype development. It is equipped with sensors commonly available on full-scale research vehicles, such as Lidar, inertial measurement unit, GPS receiver, camera, and motor encoders. It uses two Raspberry PI3 for processing. The Hamster is robot operating system (ROS) compatible, hence allowing to be integrated in a ROS network. The robot uses Ackermann steering and is therefore kinematically equivalent to a full-scale vehicle, and its dynamics, such as the suspension system, resembles that of a regular vehicle. Hence, it presents itself as a good platform for verifying dynamic feasibility and for testing the performance of the motion planner in a realistic situation. The Optitrack system is a camera-based (see top right of Fig. 11) motion-capture system that tracks markers attached to the robot with below centimeter-level accuracy. When scaled according to the size of the Hamster, its precision is approximately equivalent to those of high-precision localization systems used in autonomous vehicle research such as Trimble Applanix or OXTS RT-3003.

We use three Hamsters in the experimental validation, one acts as the EV and two of them act as obstacles. The objective is to avoid the obstacles while circulating a two-lane closed circuit, with the left lane as preferred lane. The inputs to the robot are the desired steering angle and velocity. We use the steering angle and velocity computed by the motion planner and feed them to PID-controllers that track the velocity and lateral position error from the motion plan, given the respective reference commands. The obstacles are commanded to track the middle of either of the lanes with a constant velocity, and also use PID controllers for this task. The current position of the OV is obtained from the Optitrack while the velocity is estimated. The OV prediction model is based on a simple lateral controller that approximates, but is not exactly equal to, the PID controller.

We have implemented a high-level target-point generator to produce the goal region $\mathcal{X}_{\text{goal}}$. The target point is determined by checking for a feasible point in either of the lanes $T_f v_{\text{nom}}$ m ahead of the current position of the EV, by taking the dynamic obstacles into account. Preference is given to the left lane in case both lanes are unoccupied. The goal region is a circle with radius 0.1 m from the target point. The road boundary constraint (6) is given by an analytic function expressed as super-ellipses, and checking for feasibility of (6) amounts to checking inequalities. The driving modes in this scenario are chosen to be consistent with the lane in which the target point occurs, that is, the motion planner only plan trajectories in the lane of the target point.

The different parameters in the planner, symmetric input constraints, and symmetric state constraints are shown in Table IV. Algorithm 1 is implemented in MATLAB, with Algorithm 2 embedded as C-coded `mex`-functions. MATLAB acts as a ROS node executed from a standard Linux desktop and sends the reference command to the EV vehicle controllers executing on the built-in Raspberry PI3. The OV controllers operate in their respective Raspberry PI3. The PF prediction horizon T is dependent on the timestamp of the node it expands from, not to predict too far ahead in the future, and the nominal value is shown in Table IV. From Table IV we also see that the vehicle controller tracks the same plan for two consecutive time steps, by which time an updated plan arrives.

The driving requirements are the same as introduced in Sec. VI, with the nominal velocity $v_{\text{nom}} = 0.4$ m/s and with a minimum headway distance $d_{\min} = 3v_x$ to vehicles.

A. Results

In the experimental evaluation we use a six minutes long data set. There are two obstacles, one in each lane, both with the constant reference velocity $v_{\text{nom}} = 0.2$ m/s. Because one of the OVs is in the inner lane, they have different lap times and both regular lane change situations and situations when both lanes are blocked occur.

Fig. 12 shows snapshots of a situation when the two obstacles block both lanes. The sequence of snapshots lasts for about 100 s. The EV (red) positions correspond to the true positions as measured by the Optitrack system and the green dots are the particles generated for the planning phase. First, a trajectory that overtakes the OV in front of the EV is computed ($t = 24$ s). Then, the EV stays in the right lane behind the OV until an opening appears ($t = 74$ s), and finally the EV moves back to the preferred lane ($t = 82$ s).

Inspection of Fig. 12 gives an indication of how well the tracking controllers follow the path and velocity profile. Given a well-designed tracking controller, the tracking errors quantify how good the motion planner is at generating dynamically

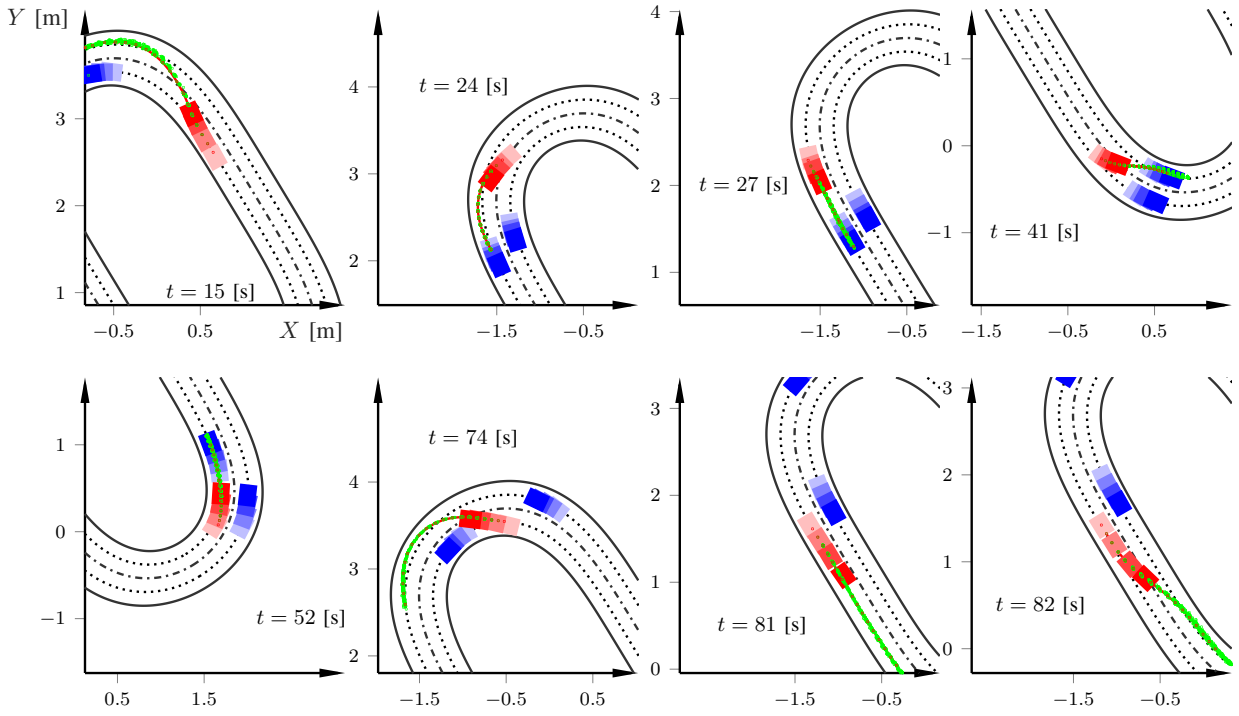

Fig. 12. Eight snapshots from the experimental validation. The EV in red, obstacles in blue, and particles from the motion planner in green. In every figure, snapshots of the EV and OVs are shown every 0.5 s in increasingly darker colors.

feasible trajectories. To give a measure of this, Fig. 13 displays the lateral control and velocity error for the whole experiment. The corresponding desired velocity $v_{\text{nom}} = 0.4$ m/s from the driving requirements, the velocity profile v_{planner} from the motion planner, and the measured velocity v_{meas} obtained from the motor encoder, as well as the reference steering profile and resulting steering angles, are in Fig. 14. Throughout, the position error is within a few centimeters and the velocity error is mostly within 2 cm/s. There are a few occasions where the errors spike. For instance, at approximately 75 s both the position and velocity error increase sharply. The reason for this can be found in the plot in Fig. 12 corresponding to $t = 74$ s, where the planner finds a gap between the two obstacles. Here, the planner initiates a sharp turn as well as an acceleration. By looking at the corresponding velocity and steering references to the controller in Fig. 14, it is clear that there are significant changes in reference commands, which the decoupled lateral and longitudinal PID-controllers cannot track precisely, given also model approximations. Furthermore, from the lower plot in Fig. 14 we see that the steering angle saturates several times throughout the experiments, because of the sharp turns in the track. This also contributes to the position errors. Further improvements are possible by coordinating longitudinal and lateral control by more advanced control methods, such as MPC. However, the results in Figs. 13 and 14 show that the controller can track the trajectories from the motion planner, that is, that the planner computes dynamically feasible and realistic trajectories, with good instantaneous tracking results even using decoupled PID-controllers. Fig. 15 shows the path of the EV for the entire experiment.

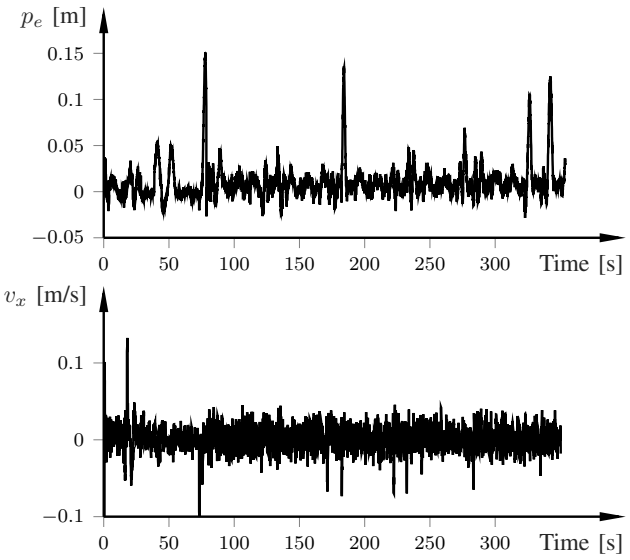

Fig. 13. The lateral control error (upper) and velocity error (lower) for the whole experiment corresponding to Fig. 12.

VIII. Conclusion

We proposed a particle-filtering based strategy for real-time decision making and motion planning of autonomous vehicles. An enabling observation was that driving requirements typically can be formulated beforehand. By formulating the allowed deviations from the requirements as a probability distribution, our method uses the requirements to guide the particles to the statistically preferable regions in the state space, which gives an efficient implementation. The sampling-

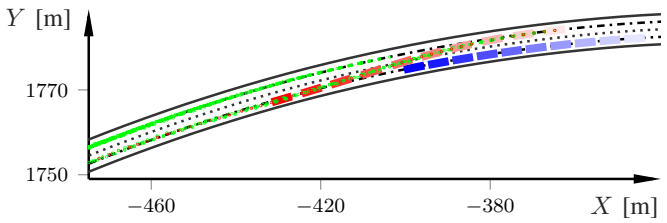

Fig. 14. Velocity and steering profiles for the experimental validation, which are the inputs to the low-level controllers. The upper and mid plots show the velocities for the whole experiment and the time span corresponding to the snapshots in Fig. 12, respectively. The velocity tracking controller stays close to the velocity profile determined by the motion planner (red), indicating that the motion planner computes realistic target velocity and acceleration profiles. For the lower plot, the sharp turns in the track results in that the planner (red) at times stays close to the steering-angle bounds, or even saturate, hence the robot steering angle (blue) will also saturate.

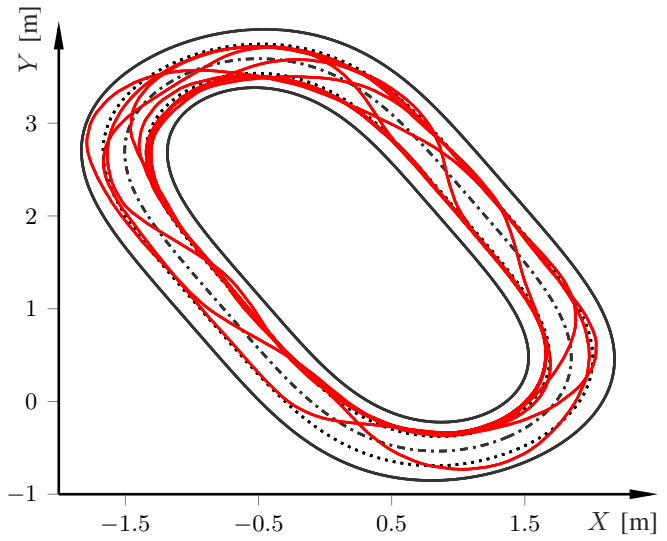

Fig. 15. Measured path for the entire experiment.

based formulation allows to discard particles that have a risk of collision. The implementation is based on a receding-horizon strategy to provide feedback in the planning and for warm starting the next iteration.

An observation from our simulation study is that the planner is able to provide a motion lasting several seconds within fractions of a second, which hints at real-time feasibility of the approach. Although our experimental evaluation was done on a small-scale robotics platform, it shows that the planner provides drivable trajectories that can be tracked even with simple decoupled PID controllers. These results reinforce that the planner is able to provide drivable trajectories for a number of different scenarios, such as lane following, lane change, obstacle avoidance, and traffic-jam situations. It is future work to also show applicability to city scenarios, such as in intersections.

References

[1] S. M. Broek, E. van Nunen, and H. Zwijnenberg, "Definition of necessary vehicle and infrastructure systems for automated driving," Eur. Commission, Tech. Rep. 2010/0064, Jun. 2011.

[2] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016.

[3] K. Berntorp and S. Di Cairano, "Joint decision making and motion planning for road vehicles using particle filtering," in *IFAC Symp. Advances in Automotive Control*, Kolmården, Sweden, Jun. 2016.

[4] ——, "Particle filtering for online motion planning with task specifications," in *Amer. Control Conf.*, Boston, MA, Jul. 2016.

[5] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Trans. Signal Process.*, vol. 50, no. 3, pp. 736–746, 2002.

[6] R. Douc, E. Moulines, and J. Olsson, "Long-term stability of sequential Monte Carlo methods under verifiable conditions," *The Annals of Applied Probability*, vol. 24, no. 5, pp. 1767–1802, 2014.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[8] J. Nilsson, P. Falcone, M. Ali, and J. Sjöberg, "Receding horizon maneuver generation for automated highway driving," *Control Eng. Pract.*, vol. 41, pp. 124–133, 2015.

[9] N. Murgovski and J. Sjöberg, "Predictive cruise control with autonomous overtaking," in *Conf. Decision and Control*, Osaka, Japan, 2015.

[10] J. Funke, M. Brown, S. M. Erlien, and J. C. Gerdes, "Collision avoidance and stabilization for autonomous vehicles in emergency scenarios," *IEEE Trans. Control Syst. Technol.*, vol. PP, no. 99, pp. 1–13, 2016.

[11] V. Turri, A. Carvalho, H. E. Tseng, K. H. Johansson, and F. Borrelli, "Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads," in *Int. Conf. Intell. Transp. Syst.*, The Hague, Netherlands, 2013.

[12] J. Ji, A. Khajepour, W. W. Melek, and Y. Huang, "Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints," *IEEE Trans. Veh. Technol.*, vol. 66, no. 2, pp. 952–964, 2017.

[13] H. Guo, C. Shen, H. Zhang, H. Chen, and R. Jia, "Simultaneous trajectory planning and tracking using an MPC method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle," *IEEE Trans. Ind. Informat.*, 2018.

[14] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the Urban challenge," *J. Field R*, vol. 25, no. 8, pp. 425–466, 2008.

[15] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The Stanford entry in the Urban challenge," *J. Field R.*, vol. 25, no. 9, pp. 569–597, 2008.

[16] N. J. Nilsson, *Principles of Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1980.

[17] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Int. Joint Conf. on Artificial Intelligence*, Montreal, Quebec, Canada, 1995.

[18] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, 2005.

[19] S. Anderson, S. Peters, T. Pilutti, and K. Iagnemma, "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios," *Int. J. Vehicle Autonomous Systems*, vol. 8, no. 2/3/4, pp. 190–216, 2010.

[20] K. Berntorp, "Path planning and integrated collision avoidance for autonomous vehicles," in *Amer. Control Conf.*, Seattle, WA, May 2017.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[22] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *Int. Conf. Robotics and Automation*, Karlsrühe, Germany, May 2013.

[23] O. Arslan, E. A. Theodorou, and P. Tsiotras, "Information-theoretic stochastic optimal control via incremental sampling-based algorithms," in *Symp. Adaptive Dynamic Programming and Reinforcement Learning*, Orlando, FL, Dec. 2014.

[24] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman *et al.*, "A perception-driven autonomous urban vehicle," *J. Field R.*, vol. 25, no. 10, pp. 727–774, 2008.

[25] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[26] J. Hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *Amer. Control Conf.*, Washington, DC, Jun. 2013.

[27] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.
[28] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, 2009.
[29] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *Int. Conf. Robotics and Automation*, Singapore, May 2017.
[30] T. Gu and J. M. Dolan, "On-road motion planning for autonomous vehicles," in *Int. Conf. Intell. Robotics and Applications*. Springer, 2012, pp. 588–597.
[31] T. Gillespie, *Fundamentals of vehicle dynamics*. Society of Automotive Engineers, Inc., 1992.
[32] A. Carvalho, S. Lefévre, G. Schildbach, J. Kong, and F. Borrelli, "Automated driving: The role of forecasts and uncertainty - a control perspective," *Eur. J. Control*, vol. 24, pp. 14–32, 2015.
[33] R. Rajamani, *Vehicle Dynamics and Control*. Springer-Verlag, 2006.
[34] F. Lindsten, M. I. Jordan, and T. B. Schön, "Particle Gibbs with ancestor sampling." *J. Machine Learning Res.*, vol. 15, no. 1, pp. 2145–2184, 2014.
[35] K. Berntorp and S. Di Cairano, "Particle Gibbs with ancestor sampling for identification of tire-friction parameters," in *IFAC World Congress*, Toulouse, France, Jul. 2017.
[36] X. L. Hu, T. B. Schön, and L. Ljung, "A general convergence result for particle filtering," *IEEE Trans. Signal Process.*, vol. 59, no. 7, pp. 3424–3429, July 2011.
[37] Cogniteam, "The Hamster," 2018, [accessed 8-January-2018]. [Online]. Available: www.cogniteam.com/hamster5.html
[38] Optitrack, "Prime 13 motion capture," 2018, [accessed 23-January-2018]. [Online]. Available: http://optitrack.com/products/prime-13

Karl Berntorp received the M.Sc. degree in Engineering Physics in 2009 and the Ph.D. degree in Automatic Control in 2014, from Lund University, Lund, Sweden. In 2014 he became a research scientist at the Mitsubishi Electric Research Laboratories in Cambridge, MA. His research is on statistical signal processing, sensor fusion, and optimization-based control, with applications to automotive, aerospace, transportation systems, and communication systems. His work includes design and implementation of nonlinear filtering, constrained control, and motion-planning algorithms. Dr. Berntorp is the author of more than 35 peer-reviewed international papers and several patents, and he is a founder/co-founder of two engineering consultancy companies.

Tru Hoang received the Bachelors degree in Mechanical Engineering from Boston University in 2015. After receiving his Masters degree in the Aerospace Engineering from the University of Michigan in 2017, he joined MERL for an internship in the Mechatronics group. He is currently working as an Algorithm Engineer at Aptiv. His research interests are optimal planning and control of intelligent vehicles.

(SM'08) Stefano Di Cairano received the Master (Laurea), and the PhD in Information Engineering in '04 and '08, respectively, from the University of Siena, Italy. He has been visiting student at the Technical University of Denmark and at the California Institute of Technology. During 20082011, he was with Powertrain Control R&A, Ford Research and Adv. Engineering, Dearborn, MI. Since 2011, he is with Mitsubishi Electric Research Laboratories, Cambridge, MA, where he is now the Senior Team Leader for Optimization-based Control, and a Senior Principal Researcher in Mechatronics. His research is on optimization-based control strategies for complex mechatronic systems, in automotive, factory automation, transportation systems and aerospace. His research interests include model predictive control, constrained control, networked control systems, hybrid systems, optimization.

Dr. Di Cairano has authored/co-authored more than 130 peer reviewed papers in journals and conference proceedings and 25 patents. He was the Chair of the IEEE CSS Technical Committee on Automotive Controls 2012-2015, is the Chair of IEEE Standing Committee on Standards since 2016, and an Associate Editor of the IEEE Transactions on Control Systems Technology.