Motion Planning of Autonomous Road Vehicles by Particle Filtering: Implementation and Validation

Berntorp, K.; Inani, P.; Quirynen, R.; Di Cairano, S.

TR2019-062 July 11, 2019

Abstract

Autonomous driving in urban and highway scenarios involves a set of predefined requirements that the vehicle should obey, such as lane following, safety distances to surrounding vehicles, and speed preferences. We have previously shown that by interpreting the motion-planning problem as a nonlinear non-Gaussian estimation problem, we can leverage particle filtering to determine suitable vehicle trajectories. In this paper, we validate our proposed motion planner using scaled vehicles. We show that our motion planner is capable of determining safe and drivable trajectories for a number of challenging scenarios, and that the trajectories can be accurately tracked by a lower-level nonlinear model predictive control scheme.

American Control Conference (ACC)

Motion Planning of Autonomous Road Vehicles by Particle Filtering: Implementation and Validation

Karl Berntorp, Pranav Inani, Rien Quirynen, and Stefano Di Cairano

Abstract— **Autonomous driving in urban and highway scenarios involves a set of predefined requirements that the vehicle should obey, such as lane following, safety distances to surrounding vehicles, and speed preferences. We have previously shown that by interpreting the motion-planning problem as a nonlinear non-Gaussian estimation problem, we can leverage particle filtering to determine suitable vehicle trajectories. In this paper, we validate our proposed motion planner using scaled vehicles. We show that our motion planner is capable of determining safe and drivable trajectories for a number of challenging scenarios, and that the trajectories can be accurately tracked by a lower-level nonlinear model predictive control scheme.**

I. INTRODUCTION

Autonomous vehicles are complex decision-making systems that integrate interconnected sensing and control components. 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.

A discrete decision maker is responsible for determining the local driving goal of the vehicle. A motion planner determines a desired trajectory that the vehicle should follow based on the outputs from the sensing and mapping module and the decision maker. 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 environment. Important requirements are that the trajectory computed by the motion planner is collision free, dynamically feasible, and possible to track by the vehicle controller. The motion-planning problem in autonomous vehicles share many similarities with the standard robotics setup [1]. Exact optimal solutions are in most cases intractable. Approaches relying on model predictive control (MPC) have been developed for specialized scenarios [2]–[4]. However, a typical limiting factor with these approaches is nonconvexity [2]. 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, even to find just a feasible solution. Motion planning in autonomous vehicle research is often performed using either sampling-based methods such as rapidly-exploring random trees (RRTs) [1], [5], [6], graph-search methods [7], [8] such as A* or D* [9], [10], or optimal control, possibly using MPC for tracking the motion plan [11], [12].

We have previously developed a probabilistic method for integrated decision making and motion planning [13], in which 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. 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.

This paper experimentally validates our approach in different driving scenarios. We have developed a test bench consisting of small-scale vehicles [14], and recorded hours worth of driving data for numerous scenarios, such as different types of lane-change, overtaking, collision-avoidance, and traffic-jam situations. We show a relevant set of these results, and we point out implementation aspects that are generally applicable to any autonomous-driving system.

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, $\boldsymbol{y}_{m:k} := \{y_m, \dots, \boldsymbol{y}_k\}$. 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.

II. MODELING

We refer to the automated vehicle as the ego vehicle (EV), whereas other moving vehicles in the region of interest (ROI) of the EV are designated as other vehicles (OV). Note that the OVs can be either autonomous or manual vehicles, as we do not assume any explicit collaboration between different vehicles. Our method handles general discrete-time nonlinear vehicle models for describing the time evolution of the EV,

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

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 assume moderate steering and acceleration maneuvers and use the kinematic single-track model [15], with state

The authors are with Mitsubishi Electric Research Laboratories (MERL), 02139 Cambridge, MA, USA, Email:{karl.o.berntorp,dicairano}@ieee.org, {anani,quirynen}@merl.com.

vector $\boldsymbol{x} = [p_X\ p_Y\ \psi\ v_x\ \delta]^{\mathrm{T}}$, 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, and δ is the steering angle of the front wheel. The inputs are the acceleration and steering rate, 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 and the steering rate $\dot{\delta}$ and acceleration $\dot{v}_x$ are subject to linear constraints, which can compactly be written as

$$\mathcal{U} = \{\boldsymbol{u}_k : \boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max}\}. \tag{2}$$

The road-boundary constraint can be written as

$$\Gamma(p_X, p_Y) \leq 0, \tag{3}$$

where Γ is constructed from point-wise road data.

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 (conservative) ellipsoids. The spatial extent of the collision area of the EV around the o-th OV is denoted with $\mathcal{B}_o$, and the corresponding OV state is $\boldsymbol{x}_o^{\text{OV}} = [p_{X,o}^{\text{OV}}\ p_{Y,o}^{\text{OV}}\ \psi_o^{\text{OV}}\ v_{x,o}^{\text{OV}}]^{\mathrm{T}}$. We define the (deterministic or probabilistic) obstacle set at time step k as $\mathcal{O}_k(\boldsymbol{x}_{o,0}^{\text{OV}}, \mathcal{B}_o)$, which depends on the measured/estimated OV state $\boldsymbol{x}_{o,0}^{\text{OV}}$ at $k = 0$. Denote the planning horizon with T_f. The predicted set of the o-th OV for $k \in [0, T_f]$ is

$$\mathcal{S}_{k,o} = \mathcal{O}_{0:k}(\boldsymbol{x}_{o,0}^{\text{OV}}, \mathcal{B}_o). \tag{4}$$

The area the motion planner should avoid up until time index k is computed as the union over all OV trajectory sets (4),

$$\mathcal{S}_k = \bigcup_{o=1}^{M} \mathcal{S}_{k,o}. \tag{5}$$

A. Discrete Decision Model

Given the route plan, there are behavioral driving decisions to be determined. If we consider a three-lane road and have a high-level route coming from a route planner, the possible modes are either to come to a full stop (S), stay in lane (SL), change lane left (CLL), or change lane right (CLR). The decisions can be modeled as a set of modes

$$\mathcal{M} = \{S, SL, CLL, CLR\} = \{m_1, \ldots, m_4\}. \tag{6}$$

The decision making model is a finite-state Markov process with transition probabilities. These transition probabilities can be determined from the driving context perceived from the sensing and prediction modules, deduced from (5), in combination with the route commanded by the route planner. The transition model between any two modes j and i is

$$m_j \sim p(m_j | m_i), \tag{7}$$

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 (7) can be set to the same value for all transitions.

B. Driving Requirements

The proposed method is based on that nominal driving requirements can be determined a priori. These requirements can be summarized in the vector $\boldsymbol{y}_k \in \mathbb{R}^{n_y}$ for each time step k. We model the driving requirements 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,

$$\boldsymbol{y}_k = \begin{bmatrix} v_{\text{nom},k} & 0 & g_{1,k} & \cdots & g_{M,k} \end{bmatrix}^{\mathrm{T}}, \tag{8}$$

$$g_{o,k} = \begin{cases} 0 & \text{if } d_{o,k} > d_{\min}, \\ f(d_{\min} - d_{o,k}) & \text{if } d_{o,k} \leq d_{\min}, \end{cases} \tag{9}$$

where $d_{o,k}$ is the distance between the EV and the o-th OV and $g_{o,k}(\cdot)$ 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 output equations on the vehicle states as

$$\hat{\boldsymbol{y}}_k = \boldsymbol{h}(\boldsymbol{x}_k, m_j, \mathcal{S}_k, \boldsymbol{x}^{\text{RD}}) + \boldsymbol{e}_k, \tag{10}$$

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 $\{\boldsymbol{x}^{\text{OV}}\}_{o=1}^{M}$), and road information $\boldsymbol{x}^{\text{RD}}$, to the driving requirements. Furthermore, $\boldsymbol{e}_k \in \mathbb{R}^{n_e}$ 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. For the driving requirements in (8), (10) can be written as

$$\boldsymbol{h}(\boldsymbol{x}_k, m_j, \mathcal{S}_k, \boldsymbol{x}^{\text{RD}}) = \begin{bmatrix} v_{x,k} & p_{e,k} & d_{1,k} & \cdots & d_{M,k} \end{bmatrix}, \tag{11}$$

where $p_{e,k}$ is the lateral deviation from the middle of the lane in the road-aligned frame.

III. Decision Making and Motion Planning Using Particle Filtering

We briefly outline our method for decision making and motion planning, but refer to [13], [16] for details.

A. Particle Filtering with Discrete and Continuous States

The objective of the motion planner 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 (2), road constraints (3), and obstacle constraints (5). In addition, we want to minimize deviations from the predefined driving requirements (8).

In a Bayesian framework, by adding process noise $\boldsymbol{w}_k$ to the vehicle model (1), (1) and (10) can be formulated as

$$\boldsymbol{x}_{k+1} \sim p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k), \tag{12a}$$

$$\boldsymbol{y}_k \sim p(\boldsymbol{y}_k | \boldsymbol{x}_k, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_k, m_j), \tag{12b}$$

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

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 (2) such that the resulting trajectory obtained from (1) obeys (3), avoids the obstacle set (5), 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 to be given by a higher-level route planner.

The main idea in the approach is that we determine the state trajectory PDF $p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, m_j, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_T)$, 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.

The PF approximates the state trajectory PDF by a set of N particles $\boldsymbol{x}^i_{0:T}$ and their associated importance weights q^i_T,

$$p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, m_j, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_T) \approx \sum_{i=1}^{N} q^i_T \delta(\boldsymbol{x}_{0:T} - \boldsymbol{x}^i_{0:T}), \quad (13)$$

where q^i_T in (13) is the importance weight for the ith particle and $\delta(\cdot)$ is the Dirac delta mass. To propagate the particles, the PF generates N samples $\boldsymbol{x}^i_k$ from a proposal density $\pi(\cdot)$,

$$\boldsymbol{x}_k \sim \pi(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{y}_k, m_j, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_k) \quad (14)$$

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^i_k according to

$$q^i_k \propto \frac{p(\boldsymbol{y}_k|\boldsymbol{x}^i_k, m_j, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_k) p(\boldsymbol{x}^i_k|\boldsymbol{x}^i_{k-1})}{\pi(\boldsymbol{x}^i_k|\boldsymbol{x}^i_{k-1}, \boldsymbol{y}_k, m_j, \boldsymbol{x}^{\text{RD}}, \mathcal{S}_k)} q^i_{k-1}, \quad (15)$$

The weight update (15) will put low weight to states that are relatively far from obeying the driving requirements. 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 update

$$q^i_k = \begin{cases} (15) & \text{if (3) and (5) are satisfied} \\ 0 & \text{if (3) or (5) are violated} \end{cases}. \quad (16)$$

Using (16), the road constraints are not violated and obstacle set is not entered, provided at any time step there is at least one out of the N particles that satisfies (3) and (5).

IV. Implementation Aspects

In this section we provide implementation details that are important for reliable trajectory planning.

A. Determining the Tree Expansion

It is computationally inefficient to execute the full PF at each planning step, because old, possibly useful, information is discarded [17]. Hence, to allow warm-starting the motion planner, we incrementally expand a tree $\mathcal{T}$ in the following way. Storing the complete representation of the PDF over the planning horizon is problematic given the limited memory capabilities of automotive micro-controllers. Instead, we extract the trajectory and corresponding inputs from (13) by using the minimum mean-square estimates

$$\boldsymbol{x}_{0:T} = \sum_{i=1}^{N} q^i_T \boldsymbol{x}^i_{0:T}, \quad \boldsymbol{u}_{0:T-1} = \sum_{i=1}^{N} q^i_T \boldsymbol{u}^i_{0:T-1}. \quad (17)$$

The tree is extended with the state trajectory in (17) as vertices $\mathcal{V}$ and the input transitions in (17) 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 (5) 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 [6]. Each vertex also contains a timestamp and a cost C for reaching that node. The proposed method not only checks whether an intersection with an OV occurs, but also at what time. The dynamic collision avoidance is because the method generates trajectories (i.e., the solution has the concept of time at generation). At the end of the tree expansion, the lowest-cost trajectory in terms of the cost C is chosen for execution. We sample and connect the driving modes to nodes corresponding to the same driving mode, to avoid several driving-mode changes in one planning phase. In practice we generate the state trajectory in (17) by first generating control inputs, which are then used to simulate the system to obtain the state trajectories [13].

B. Receding-Horizon Implementation

We implement the motion planner in a receding-horizon strategy. 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. 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.

C. Algorithm Summary

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).

V. Experimental Results

This section presents results from multiple scenarios using our scaled vehicle experimental platform. We use the Hamster platform [18] for testing and verifying our control stack before deploying on a full-scale vehicle, see Fig. 1. The Hamster is a 25×20 cm mobile robot for research and prototype development. It is equipped with scaled versions of sensors commonly available on full-scale research vehicles, such as a 6 m range mechanically rotating 360 deg Lidar, an inertial measurement unit, GPS receiver, HD camera,

Algorithm 1 Proposed Planning Method

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

Algorithm 2 Particle Filter for Trajectory Generation

```
    Input: Propagated state $\hat{x}$, driving mode $m_j$, and $\mathcal{T}$.
1: Choose a feasible node in the tree consistent with $m_j$ and extract state $x_0$.
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 $x_k^i$ and input $u_k^i$ from (14).
6:         Update weight $\bar{q}_k^i$ using (16).
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 $q_k^i \leftarrow 1/N, \quad \forall i \in \{1, \dots, N\}$.
16:    end if
17: end for
18: Extract trajectory and inputs according to (17).
    Return: $\{x_{0:T}, u_{0:T-1}\}$
```

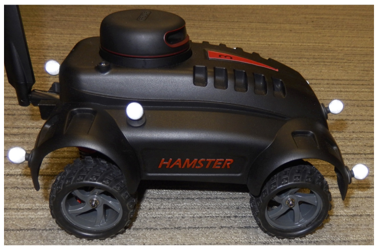

Fig. 1. The Ackermann-steered Hamster mobile robot used in the experiments. The markers (five visible in the figure) are used to track the robot via an Optitrack motion-capture system (Fig. 2).

and motor encoders. It uses two Raspberry PI3 computing platforms, each with an ARM Cortex-A53 processor running Linux Ubuntu for processing. The Hamster has Ackermann steering and is kinematically equivalent to a full-scale vehicle. Its dynamics, such as the suspension system, resembles that of a regular vehicle. The Hamster has a low-level controller with dedicated hardware for power distribution and monitoring, and it is controlled by setting the desired wheel-steering angle and longitudinal velocity. The Hamster has built-in mapping and localization capabilities. The platform is a good proxy for verifying dynamic feasibility and for testing the performance of the system in a realistic setting, with a sensor setup similar to the one expected in full-scale autonomous vehicles. The Hamster connects to external algorithms using the robot operating system (ROS).

To evaluate the control and estimation algorithms in terms of tracking errors and resulting trajectories in a controlled environment, we use an Optitrack motion-capture system [19]. The Optitrack system is a camera-based (see Fig. 2) six degrees-of-freedom tracking system that can be used for tracking drones, ground, and industrial robots. Depending on the environment and quality of the calibration, the system can track the position of the Hamster within 0.9 mm and

Fig. 2. A camera from the Optitrack motion-capture system used for verifying our control and estimation algorithms.

with a rotational error of less than 3 deg. The OptiTrack is connected to the ROS network using the VRPN protocol.

We use three Hamsters in the experimental validation, one acts as the EV and two act as obstacles. The objective is to avoid the obstacles while circulating a two-lane closed circuit in the counter-clockwise direction, with the inner lane as preferred lane. The trajectory, steering angle, and velocity computed by the motion planner are sent to an NMPC using a Real-Time Iteration (RTI) solution scheme (see [20] for details) that tracks the trajectory using the steering angle and velocity as feed-forward terms. The obstacles are commanded to track the middle of either of the lanes, where the preferred lane and vehicle velocities can be changed throughout the experiments. The OVs use PID controllers for tracking the desired lane and velocity. 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 the PID controller, and the OVs are modeled as deterministic and rectangular.

The different parameters in the planner, symmetric input constraints, and symmetric state constraints are shown in Table I. 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 NMPC [20]

TABLE I

The parameter choices for the experimental study.

Parameter	Unit	Value	Meaning
N	-	100	# particles
Δt	s	0.6	Execution time
δt	s	0.1	Allocated computation time
T_s	s	0.3	Sampling period motion planner
T_f	s	5	Planning horizon
h	ms	25	Sampling period NMPC
T	-	T_f/T_s	Prediction 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

implemented using ACADO [21]. The OV controllers operate in their respective Raspberry PI3. From Table I we also see that the vehicle controller tracks the same plan for two consecutive time steps, when an updated plan arrives.

A. Overtaking with Standstill OVs

In this scenario the two OVs drive with varying speeds slightly slower than the EV, with occasional braking maneuvers forcing them to standstill. The motion-prediction module used in the motion planner is only aware of the estimated position and estimated velocity at the beginning of each planning phase, but needs to dead-reckon (i.e., predict) the OV trajectories over the entire prediction horizon. Hence, these braking maneuvers show the motion planners ability to react to unexpected behaviors in the environment.

Fig. 3 displays four snapshots of one situation where one of the OVs suddenly comes to a standstill ($t = 164$ s). The planner reacts to the changed situation and and successfully plans a new trajectory that is subsequently tracked by the NMPC ($t = 165$ s). At ($t = 175$ s), the planner determines to change back to the inner lane.

B. Traffic-Jam Assist

In this scenario there are two OVs, one in each lane, in front of the EV initially driving with considerably slower speed than the EV. The OVs change their respective speeds at time instants unknown to the EV.

Fig. 4 shows snapshots of a 135 s excerpt of an eight minutes long data set. First, a trajectory that urges the vehicle to slow down to satisfy the predefined safety distance is computed ($t = 121$ s). The motion planner computes trajectories for both lanes, but determines that it is better to stay in the inner lane. Between $t = 121$ s and $t = 160$ s, the OV in the outer lane has increased its speed, which makes the motion planner to change lane. At $t = 199$ s the OV in the inner lane speeds up, which leads to that the motion planner determines to again change lane. The motion planner determines that it is safe to overtake the OV in the inner lane ($t = 213$ s). Finally the EV moves back to the preferred lane.

Fig. 5 displays steering and velocity profiles, and the NMPC solution. The measured trajectories closely match the corresponding planned quantities, which shows that the motion planner computes dynamically feasible (i.e., drivable) trajectories. The NMPC profiles mostly match well with the planner profiles, which indicates that the proposed architecture provides reliable driving behavior. There are a few discrepancies, which are mainly due to model differences. First, the vehicle model in the NMPC uses a first-order actuator model, which is not captured in the motion planner. Second, there is a steering offset compensator that affects the control performance, since it is used as a feedforward term to the NMPC. However, the lower-level NMPC can reliably track the trajectories generated by the motion planner.

VI. Conclusion

We evaluated our previously proposed [13], [16] PF based strategy for online decision making and motion planning of autonomous vehicles. Our experimental evaluation was done on a small-scale robotics platform. The results show that the planner provides drivable trajectories that can be tracked with high accuracy, and that the NMPC and motion planner predict similar control inputs (Fig. 5). 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.

References

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] K. Berntorp and S. Di Cairano, "Particle filtering for online motion planning with task specifications," in *Amer. Control Conf.*, Boston, MA, Jul. 2016.

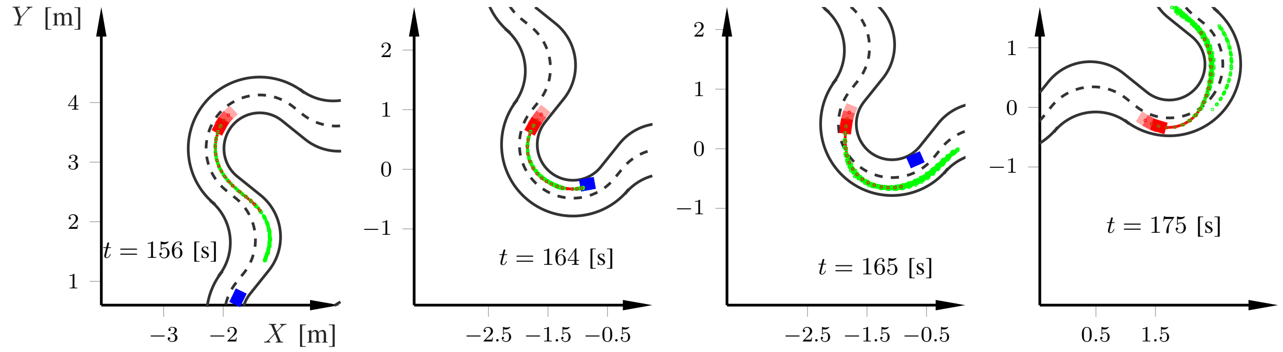

Fig. 3. Four snapshots from a situation where an OV (blue) in front of the EV (red) suddenly comes to a standstill. The EV in red, obstacles in blue, and particles from the motion planner in green. In every figure, snapshots of the EV and OVs are shown every 0.5 s in increasingly darker colors.

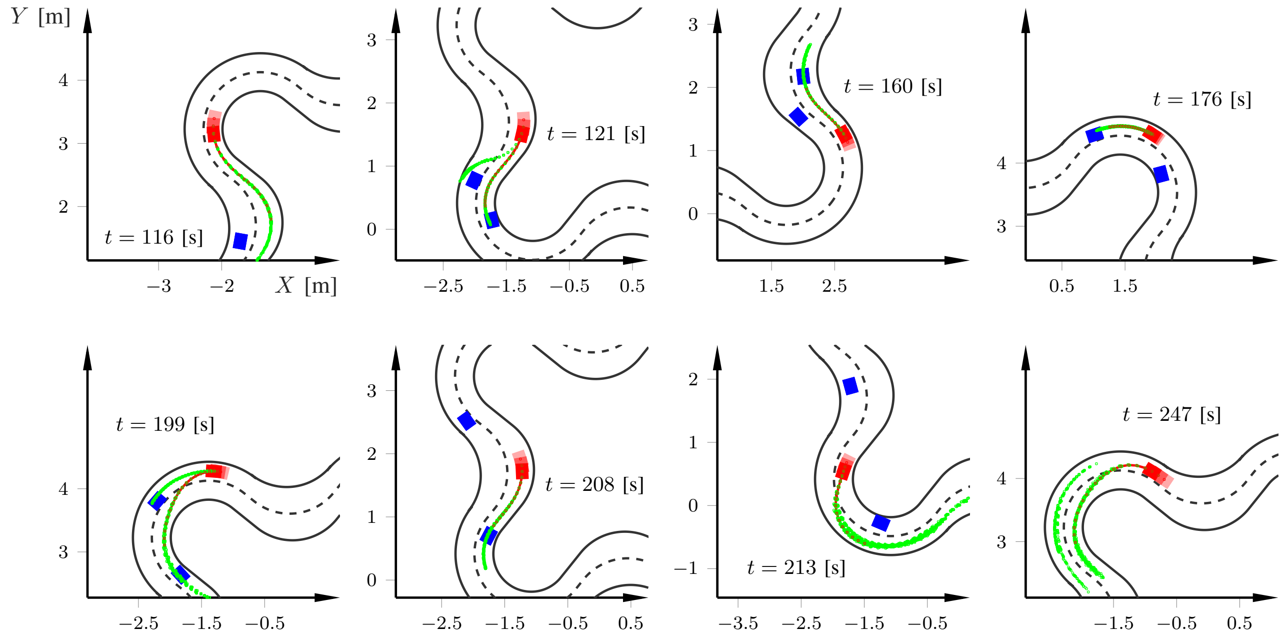

Fig. 4. Eight snapshots from the experimental validation in the traffic-jam scenario. Same notation as in Fig. 3

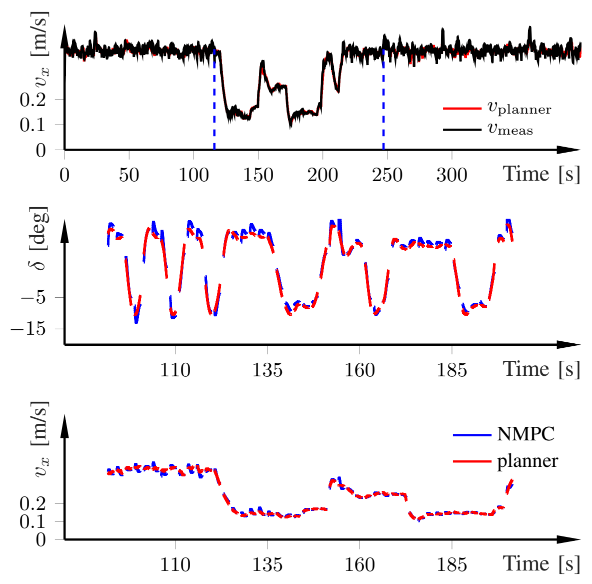

Fig. 5. Steering and velocity references (red) from the motion planner and the corresponding measured quantities throughout the experiment in the upper plot, where the portion of the data set in Fig. 4 is indicated by the blue dashed lines. The two lower plots show the steering and velocity references from the motion planner and the corresponding predicted quantities from the NMPC, at every third planning instant.

[14] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano, "Control architecture design of autonomous vehicles," in *Conf. Control Technol. and Applications*, Copenhagen, Denmark, Aug. 2018, invited paper.

[15] 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.

[16] 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.

[17] 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.

[18] Cogniteam, "The Hamster," 2018, [accessed 8-January-2018]. [Online]. Available: www.cogniteam.com/hamster5.html

[19] Optitrack, "Prime 13 motion capture," 2018, [accessed 23-January-2018]. [Online]. Available: http://optitrack.com/products/prime-13

[20] R. Quirynen, K. Berntorp, and S. Di Cairano, "Embedded optimization algorithms for steering in autonomous vehicles based on nonlinear model predictive control," in *Amer. Control Conf.*, Milwaukee, WI, Jun. 2018.

[21] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating microsecond solvers for nonlinear MPC: a tutorial using ACADO integrators," *Optimal Control Applications and Methods*, vol. 36, pp. 685–704, 2014.