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

Particle Filtering for Online Motion Planning with Task Specifications

Berntorp, K.; Di Cairano, S.

TR2016-052 July 2016

Abstract

A probabilistic framework for online motion planning of vehicles in dynamic environments is proposed. We develop a sampling-based motion planner that incorporates prediction of obstacle motion. A key feature is the introduction of task specifications as artificial measurements, which allows us to cast the exploration phase in the planner as a nonlinear, possibly multimodal, estimation problem, which is effectively solved using particle filtering. For certain parameter choices, the approach is equivalent to solving a nonlinear estimation problem using particle filtering. The proposed approach is illustrated on a simulated autonomous-driving example. The results indicate that our method is computationally efficient, consistent with the task specifications, and computes dynamically feasible trajectories.

2016 American Control Conference (ACC)

Particle Filtering for Online Motion Planning with Task Specifications

Karl Berntorp[1] and Stefano Di Cairano[1]

***Abstract*— A probabilistic framework for online motion planning of vehicles in dynamic environments is proposed. We develop a sampling-based motion planner that incorporates prediction of obstacle motion. A key feature is the introduction of task specifications as artificial measurements, which allows us to cast the exploration phase in the planner as a nonlinear, possibly multimodal, estimation problem, which is effectively solved using particle filtering. For certain parameter choices, the approach is equivalent to solving a nonlinear estimation problem using particle filtering. The proposed approach is illustrated on a simulated autonomous-driving example. The results indicate that our method is computationally efficient, consistent with the task specifications, and computes dynamically feasible trajectories.**

I. INTRODUCTION

Research toward increased vehicle autonomy is currently given considerable attention. A key component for autonomy is reliable *motion planning*. The goal of a motion planner is to compute a trajectory profile to safely navigate towards a goal, possibly in presence of moving obstacles and different types of uncertainty. Sampling-based path planners, such as rapidly-exploring random trees (RRTs), are popular [1]. RRTs rely on random exploration of the state space. Each generated sample is checked for collision, typically assuming a static environment; a collision-free sample is added as a node, and connections are made to surrounding nodes for tree expansion. Many path-planning techniques only provide geometric paths. In these cases the considered system is assumed to achieve instantaneous tracking. However, for many applications, such as when considering road vehicles or mobile manipulators, the equations of motion (i.e., the differential constraints) significantly restrict the reachable set and must therefore be accounted for. In addition, autonomous vehicles should be reliable in changing and/or uncertain environments, implying that online replanning is a requirement.

We propose sequential Monte-Carlo methods for online motion planning for vehicles in dynamic environments. First, we formulate an RRT that samples control inputs and includes prediction of obstacle motion, that is, a time stamp is associated with each node. An input-based RRT is motivated by that the state space is often of higher dimension than the input space. The search dimension is therefore decreased with an input-based approach. Furthermore, sampling from the input space generates paths that satisfy differential constraints on the vehicle motion. Explicit motion prediction of obstacles enables reuse of many nodes when replanning, hence achieving fast computations. Second, we leverage that there are often certain task specifications, or objectives, associated with the desired motion. We interpret these specifications as measurements from an idealized motion and formulate the tree expansion in the RRT as a nonlinear, possibly multimodal, estimation problem, which we solve using *particle filtering*. Particle filters (PFs) are sample-based estimators that generate inputs according to a noise model and propagate them through the system model. Hence, complex and system models are trivially incorporated. PFs can achieve arbitrarily good estimates [2], [3] and demonstrations of real-time capabilites of PFs have been shown in, for example, aircraft and automotive applications [4], [5]. The inputs can be biased according to a given goal (e.g., desired location and/or velocity). Depending on certain design choices in our algorithm, both standard input-based RRT and conventional particle filtering can be recovered.

RRTs have also explicitly addressed differential constraints [6]–[8], but equations of motion, which cannot be neglected when considering, for example, nonholonomic or automotive systems, are not trivially incorporated into the traditional RRT framework. To connect two nodes is a two-point boundary value problem, and in general there are no guarantees that a solution exists [9]. Input-based (kinodynamic) RRTs randomly choose a node to expand from, and then generate random inputs and propagate them through the system model [10], [11]. 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 [6]. This is not the case for our approach. In [12], an input-based RRT that samples position references to a tracking controller was developed for autonomous vehicle driving. The approach in [12] assumes goals in terms of position references. Standard RRTs are suboptimal. RRT with asymptotic optimality, RRT*, was introduced in [13], and an application of RRT* to motion planning was made in [8]. However, here we focus on online motion planning with scarce computing capabilites, implying that the main focus of this paper is to quickly find smooth, drivable motion profiles, rather than searching for the optimal motion. PFs in path-planning applications have been considered before, but for uncertainty propagation [14].

II. PARTICLE FILTER FOR ESTIMATION

Throughout, $p(x_k|y_{m:k})$ denotes the conditional probability density function of the variable (state) $x \subset X \in \mathbb{R}^{n_x}$ at time $t_k \in \mathbb{R}$ conditioned on the variable (measurement) $y \in \mathbb{R}^{n_y}$ from time t_m to time t_k, $y_{m:k} := \{y_m, \ldots, y_k\}$. Given mean vector μ and covariance matrix Υ, $\mathcal{N}(\mu, \Upsilon)$

[1]Karl Berntorp and Stefano Di Cairano are with Mitsubishi Electric Research Laboratories, 02139 Cambridge, MA, USA Email:{karl.o.berntorp,dicairano}@ieee.org

and $\mathcal{N}(x|\mu, \Upsilon)$ stand for the Gaussian distribution and probability density function, respectively. The notation $x \sim p(\cdot)$ means x sampled from $p(\cdot)$ and $\propto$ reads proportional to.

Consider the dynamic system

$$x_{k+1} = f(x_k, u_k, w_k), \tag{1a}$$
$$y_k = h(x_k, e_k), \tag{1b}$$

where $f \in \mathbb{R}^{n_x}$ and $h \in \mathbb{R}^{n_y}$ in general are nonlinear functions, $w_k \in \mathbb{R}^{n_w}$ and $e_k \in \mathbb{R}^{n_e}$ are process and measurement noise, respectively, and $u_k \in U \subset \mathbb{R}^{n_u}$ is the deterministic input vector (i.e., the controls). Using a Bayesian framework, (1) can be reformulated as

$$x_{k+1} \sim p(x_{k+1}|x_k, u_k), \quad y_k \sim p(y_k|x_k),$$

where x_{k+1} and y_k are regarded as samples from the respective distributions. Without loss of generality, u_k is set to zero in the following. When (1) is linear with additive Gaussian noise, the optimal estimator is analytically computed and given by the well-known Kalman filter. Most often, however, numerical approximations are required.

A popular approach to the estimation of nonlinear systems is to use PFs [15], [16], in which the probability density function $p(x_k|y_{0:k})$ is estimated. PFs are sequential Monte-Carlo methods that approximate the posterior density with a set of N weighted particles as

$$p(x_k|y_{0:k}) \approx \sum_{i=1}^{N} w_k^i \delta(x_k - x_k^i), \tag{2}$$

where w_k^i is the importance weight for the ith particle and $\delta(\cdot)$ is the Dirac delta function. PFs have been used in a wide range of applications, such as in tracking of automotive systems [4], [5], [17]. Next, we explain the basic version of the PF, but our approach can be extended to other types of PFs (e.g., feedback particle filter [18]). To propagate the particles forward in time, a proposal density

$$x_k \sim q(x_k|x_{k-1}, y_k) \tag{3}$$

is used. The proposal (3) is a key design step in particle filters. The weights w_k^i are recursively computed as

$$w_k^i \propto \frac{p(y_k|x_x^i)p(x_k^i|x_{k-1}^i)}{q(x_k^i|x_{k-1}^i, y_{0:k})} w_{k-1}^i. \tag{4}$$

There is a large set of options for how to choose the proposal density. One obvious choice of proposal density is

$$q(x_k|x_{k-1}^i, y_k) = p(x_k|x_{k-1}^i), \tag{5}$$

that is, prediction of particles is done using the dynamical model (1a). With this choice, the weight update equation is

$$w_k^i \propto p(y_k|x_k^i) w_{k-1}^i. \tag{6}$$

With the proposal (5), the PF is called a bootstrap PF [19].

The PF includes a necessary resampling step. When the effective number of samples $N_{\text{eff}} \approx 1/\sum_i (w^i)^2$ becomes small (e.g., below γN, $\gamma \leq 1$), N particles are chosen (with replacement), where the probability of choosing x_k^i is w_k^i. There is a vast literature on the theoretical properties of PFs—for example, guarantees of convergence both in weak and mean-square sense [2], [3].

III. PARTICLE FILTER FOR ONLINE MOTION PLANNING

Next, we present the proposed motion planner for online motion planning with avoidance of dynamic obstacles.

A. Task Specification

Our approach relies on that there are certain specifications (objectives) associated to the planning problem, which can be used to guide the samples. To exemplify, when considering vehicle motion planning, possible specifications are:

- Stay within some known region
- Maintain constant velocity
- Drive smoothly, that is, prioritize small steer rates
- Keep safety distance to surrounding obstacles

These specifications can be interpreted as desired measurements obtained from an idealized motion. The deviation of the actual velocity v from a desired nominal velocity v_{nom} can be written as $h(v) = v_{\text{nom}} - v$, and ideally we want to enforce $h(v) = 0$. Similarly, desire to maintain a safety distance to obstacles can be formulated using a nonlinear function g that is large for small distances, and ideally $g = 0$.

Assume that the system after discretization with sampling time T_s is described by the stochastic difference equation

$$x_{k+1} = f(x_k) + w_k, \tag{7}$$

where x_k are the states and w_k is additive noise, specified by its (known) probability density function p_w. For notational brevity, u_k is omitted. Now, assume that there are n_y different state-dependent specifications. These are modeled as constraints by introducing desired outputs y_d as functions of the states,

$$y_d = h(x_k) := \begin{bmatrix} h_1(x_k) & \cdots & h_{n_y}(x_k) \end{bmatrix}^{\mathrm{T}}. \tag{8}$$

The desired outputs can directly correspond to some or all of the states. The features in (8) cannot be exactly tracked. We therefore add a probabilistic slack on each desired output, expressed by e_k. The slack determines how large deviations can be tolerated. Imperfect tracking can occur for several reasons, for example, model imperfections. Sometimes the objectives may be in conflict. For instance, in some situations it might be impossible to maintain constant velocity while keeping a safety margin to surrounding obstacles. The result is a relation at each time instant k as in (1b),

$$y_k = h(x_k) + e_k. \tag{9}$$

The measurement noise provides for an intuitive way to accomodate different objectives in the motion planning. Traditionally, the cost function determines which objectives to prioritize. However, when the prioritization is done already in the input sampling, it is more likely that the specifications will be met using fewer samples.

Remark 1: The slack e_k can be set to any distribution. For illustration, we use zero-mean Gaussian with covariance R_k,

$e_k \sim \mathcal{N}(0, R_k)$, since the distribution is then analytically given by R_k. Note that for hard constraints, such as obstacle avoidance, other distributions (e.g., uniform) might be more suitable. This is not explored further here. Examples of distributions are given in [20].

B. Input-Based RRT with Particle Filtering

We use PF as a means to generate trajectories and build the tree. The main steps in the algorithm are outlined next.

1) Online Execution: We recursively compute a trajectory over a horizon constrained by the sensing information, toward a goal $\mathcal{X}_{\text{goal}}$ that is assumed given by a higher-level logic, for example, by road-map information [21].

The computed trajectory is Δt long but is only applied for $\delta t \leq \Delta t$; that is, the computed trajectory is applied in receding horizon. Typical sensors provide better accuracy at short range than long range, but it is still worthwhile to account for the long-range information, similar to what is done in model predictive control. As in [12], we keep a committed tree, where the end coincides with the root node of the next planning phase. The part of the tree that does not originate from the end node is deleted.

2) Collision Checking: Given the vehicle state x_k, `collisionFree` returns `True` if the vehicle position is unoccupied at time t_k and `False` otherwise. Incorporating time implies that a larger portion of the generated tree can be reused in the next planning cycle and suppresses the need for reevaluation of nodes, which is computationally heavy when replanning in dynamic environments [12]. The inclusion of time necessitates prediction of obstacles, given their states at the beginning of every planning cycle, but the number of obstacles in the vicinity of the autonomous vehicle is typically much less than the number of nodes that have to be checked in the reevaluation.

3) State-Space Exploration using Particle Filtering: The procedure `sample` samples a node $\mathcal{V}$ using some heuristics π. A PF executes for T time steps. If particle i ends up in an occupied area, the corresponding weight w^i_{k+1} is set to zero. If all N weights become zero, the PF is terminated and a new node $\mathcal{V}$ is chosen for expansion of the tree $\mathcal{T}$. States are inserted as nodes and the corresponding inputs are edges $\mathcal{E}$ connecting the nodes, forming the tree $\mathcal{T} = \{\mathcal{V}, \mathcal{E}\}$. Each node also contains a timestamp t_s, a cost C reaching that node, and a measure of the cost-to-goal. Because we sample inputs, not nodes, inclusion of time is straightforward. The PF provides N trajectories in each node expansion. To insert all trajectories results in a very large tree and does not utilize the information from the weights. Another option is to choose the particle with largest weight at each time step. However, this can lead to nonsmooth trajectories. We choose the weighted mean of the trajectories, which is the minimum mean-square solution over the computed trajectories.

The choice of proposal (3) is nontrivial. One option is to generate samples using the process model (5). This can be inefficient, because the measurements are ignored. We guide the samples using the conditional distribution

$$q(x_{k+1}|x^i_k, y_{k+1}) = p(x_{k+1}|x^i_k, y_{k+1}). \tag{10}$$

Since the specifications for the whole planning horizon are known beforehand, it is even possible to use future measurements in (10). This will typically add smoothness to the state profiles (compare with filtering/smoothing in state estimation). The choice (10) leads to the weight update

$$w^i_{k+1} \propto p(y_{k+1}|x^i_k) w^i_k. \tag{11}$$

Eq. (11) implies that the weight is independent of the sample x^i_{k+1}. The proposal (10) is optimal in the sense that it maximizes the effective number of samples, but it is generally difficult to sample from, exactly. However, for a linear, Gaussian measurement relation (9) in the form $y_k = Hx_k + e_k$, the expression is analytic. For a nonlinear measurement relation a linearization leads to

$$\begin{aligned} q(x_{k+1}|x^i_{k-1}, y_k) &= \mathcal{N}\left(x_{k+1}|\hat{x}^i_{k+1}, (\Sigma^i_{k+1})^{-1}\right) \\ \hat{x}^i_{k+1} &= f(x^i_k) + L^i_k(y_{k+1} - \hat{y}^i_{k+1}), \\ \Sigma^i_{k+1} &= \left((H^i_k)^{\mathrm{T}} R^{-1}_{k+1} H^i_k + Q^{-1}_k\right)^{-1}, \\ L^i_k &= Q_k (H^i_k)^{\mathrm{T}} (H^i_k Q_k (H^i_k)^{\mathrm{T}} + R_{k+1})^{-1}, \\ \hat{y}^i_{k+1} &= h(f(x^i_k)), \quad H^i_k = \left.\frac{\partial h}{\partial x}\right|_{f(x^i_k)}, \end{aligned} \tag{12}$$

and Q_k is the second-order moment of the process noise. The likelihood in (11) is approximated as

$$p(y_{k+1}|x^i_k) = \mathcal{N}\left(y_{k+1}|\hat{y}^i_{k+1}, H^i_k Q_k (H^i_k)^{\mathrm{T}} + R_{k+1}\right). \tag{13}$$

Algorithm 1 provides the planner and the PF-based exploration is given in Algorithm 2. When $t > t_0 + \delta t$, if a solution exists (Line 13, Algorithm 1) the trajectory with lowest accumulated cost C is chosen for execution (Line 14, Algorithm 1). If no trajectories have converged, the one closest to the goal, measured according to a suitable metric, is chosen (Line 16, Algorithm 1).

Algorithm 1 RRT with PF-Based Input Sampling

1: **Input:** $\{x_0, t_0, \mathcal{X}_{\text{goal}}, \mathcal{T}\}$
2: $t \leftarrow t_0$, $\mathcal{V} \leftarrow x_0$
3: **while** $t \leq t_0 + \delta t$ **do**
4: $\quad \mathcal{V} \leftarrow \texttt{sample}(\mathcal{T}, \pi)$
5: $\quad (x_{0:T}, u_{0:T}, \texttt{Success}) \leftarrow \texttt{Algorithm 2}(\mathcal{V}.x)$
6: $\quad$ **if** `Success` **then**
7: $\quad\quad \mathcal{V} \leftarrow \{\mathcal{V}, x_{0:T}\}$
8: $\quad\quad \mathcal{E} \leftarrow \{\mathcal{E}, u_{0:T}\}$
9: $\quad\quad \mathcal{T} \leftarrow \{\mathcal{V}, \mathcal{E}\}$
10: $\quad$ **end if**
11: $\quad t \leftarrow t + \delta t$
12: **end while**
13: **if** $\texttt{solution}(\mathcal{X}_{\text{goal}}, \mathcal{T})$ **then**
14: $\quad (x_{\text{best}}, u_{\text{best}}) \leftarrow \texttt{constrSolution}(\mathcal{T})$
15: **else**
16: $\quad (x_{\text{best}}, u_{\text{best}}) \leftarrow \texttt{getClosest}(\mathcal{T}, \mathcal{X}_{\text{goal}})$
17: **end if**
18: Apply $(x_{\text{best}}, u_{\text{best}})$ for δt s and repeat from Line 1

Remark 2: When the particle cloud is multimodal, which can happen when, for example, the measurement equations

Algorithm 2 Particle Filter for Tree Expansion

Input: $\{x_0\}$
Initialize: $\{x_0^i\}_{i=1}^N \leftarrow x_0$, $\{w_0^i\}_{i=1}^N \leftarrow 1/N$, `Success` $\leftarrow$ `True`
1: **for** $k \leftarrow 1$ to T **do**
2: **for** $i \leftarrow 1$ to N **do**
3: $(x_k^i, u_k^i) \sim p(x_k | x_{k-1}^i, y_k)$ using (12)
4: **if** `collisionFree`(x_k^i) **then**
5: $\bar{w}_k^i \leftarrow w_k^i p(y_k | x_{k-1}^i)$ using (13)
6: **else**
7: $\bar{w}_k^i \leftarrow 0$
8: **end if**
9: **end for**
10: $N_w \leftarrow \sum_{i=1}^N w_k^i$
11: **if** $N_w = 0$ **then**
12: `Success` $\leftarrow$ `False`
13: Terminate
14: **end if**
15: Normalize: $w_k^i \leftarrow \bar{w}_k^i / \sum_{j=1}^N \bar{w}_k^j$
16: $N_{\text{eff}} \leftarrow 1/(\sum_{i=1}^N (w_k^i)^2)$
17: **if** $N_{\text{eff}} \leq \gamma N$ **then**
18: $(x_k^i, u_k^i) \leftarrow$ `resample`(x_k^i, u_k^i, w_k^i)
19: $w_k^i \leftarrow 1/N, \quad \forall i \in \{1, \ldots, N\}$
20: **end if**
21: $x_k \leftarrow \sum_{i=1}^N w_k^i x_k^i, u_k \leftarrow \sum_{i=1}^N w_k^i u_k^i$
22: $x_{0:k} \leftarrow \{x_{0:k-1}, x_k\}$, $u_{0:k} \leftarrow \{u_{0:k-1}, u_k\}$
23: **end for**
Return: $\{x_{0:T}, u_{0:T}, \texttt{Success}\}$

contain squared entities, one weighted mean can be computed for each mixture component. This implies that decision making is naturally incorporated into the algorithm. Another alternative is to execute several filters, one for each modality, if this can be determined beforehand.

Remark 3: The number of time steps T in Algorithm 2 is typically made adaptive to relate to the desired preview time Δt and the available computation time δt. The desired preview time Δt is dependent on the distance to the waypoint $\mathcal{X}_{\text{goal}}$ and the current vehicle velocity.

C. Relations Between RRT and Particle Filter

It is interesting to investigate the relation of Algorithm 1 to input-based RRT without biasing and conventional PF, respectively. By replacing Line 3 in Algorithm 2 with uniform sampling and performing static collision checking, a recursive version of the input-based RRT in [10] is recovered.

If the heuristics π at Line 4 in Algorithm 1 always chooses the initial state x_0, then the resulting algorithm is a conventional PF, possibly executed several times or once with large N, save for the collision check at Line 4 in Algorithm 2. Otherwise, it is the tree expansion that is done using a conventional PF. We therefore know that, as $N \to \infty$, the algorithm (or tree expansion) produces a set of trajectories that exactly represent the distribution of the task specifications. Furthermore, the weighted mean (Line 21 in Algorithm 2) is the best trajectory, in the mean-square sense.

IV. NUMERICAL STUDY

In the simulation study, an autonomous vehicle travels on a two-lane road. The road includes both straight-line and curved road segments. The desired velocity is $v_{\text{nom}} = 25$ m/s. There are surrounding vehicles maintaining either of the lanes with constant velocity (between 20–23 m/s), on average about 100 m apart. During parts of the simulation, both lanes are blocked by vehicles. The planner must therefore find trajectories that slow down and stay behind until an opening appears. When no obstacles are nearby, the ego vehicle should stay in the right lane. However, when there are obstacles within the planning horizon, the planner computes trajectories for both lanes and choose the best one.

We compare Algorithm 1 for two parameter choices with an input-based RRT [10] (UNIFORM). The heuristics π in Algorithm 1 is as follows: During expansion, with probability 0.5 we choose a random node in the tree, and with probability 0.5 we choose the node closest to the intermediate goal. When a feasible trajectory is found, with probability 0.5 we choose a random node, and with probability 0.5 we choose the initial node (i.e., the current position).

A trajectory has converged when it is located inside a unit ball, centered in the middle of the right lane, $\Delta t v_x$ m down the road (either of the lanes in case of obstacles). The algorithms are implemented in MATLAB. The implementations draw samples and expand the tree for $\delta t = 1$ s to compute a trajectory that is at least $\Delta t = 3$ s of duration, which reflects that onboard sensors can detect long-range obstacles with reasonable accuracy (Δt), but the accuracy is more reliable at shorter distances (δt). 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. The cost function C in the RRTs for each node is a combination of the norm of the mid-lane error and deviations from the nominal velocity. It is chosen to give both good velocity and mid-lane tracking in the RRT using uniform sampling. Already in a nonoptimized MATLAB implementation, Algorithm 1 provides a feasible trajectory faster than real time (i.e., in less than δt s) for the considered scenario and parameters. All implementations use the same computation-time limit.

Vehicle Model: We use an extended kinematic single-track model, which mimics the vehicle behavior well under mild driving conditions. More advanced models can be used in the proposed planner without modification of the algorithms. The motion equations are:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\psi} \\ \dot{v}_x \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\psi + \beta)/\cos(\beta) \\ v_x \sin(\psi + \beta)/\cos(\beta) \\ v_x \tan(\delta)/L \\ u_1 \\ u_2 \end{bmatrix},$$

where $L := l_f + l_r$ is the wheel base and the body slip β is $\beta := \arctan(l_r \tan(\delta)/L)$. In the motion planner, we use an Euler discretization with time step $T_s = 0.1$ s. The inputs to the model are longitudinal acceleration and steer rate, which are sampled from a Gaussian distribution.

Feature Selection: We introduce the following task objectives: (i) follow a nominal longitudinal velocity $v_{\text{nom}} = 25$ m/s; (ii) follow the middle lane, if there are no obstacles in the vicinity of the ego vehicle the goal is to follow the middle of the right lane, but in case of obstacles both lanes are tracked; (iii) maintain a safety margin to vehicles; $(iiii)$ maintain a minimum distance to vehicles in the same lane of d_s m. Assume that there are m obstacles with distances $\{d_j\}_{j=1}^{m}$, which describe the squared distance along each coordinate axis computed in the frame of the obstacle, within some radius from the ego vehicle. Denote the mid-lane error by p_e. Then, the desired outputs and objectives are modeled as $y_d = [v_{\text{nom}}\ 0\ 0]^{\text{T}}$, $h(x) = [v_x\ |p_e|\ \sigma \exp(-d_j^{\text{T}} W d_j)]^{\text{T}}$, where σ determines the safety distance, and W is a weighting matrix. Here, we choose the exponential function for introducing safety margin, but other functions can be used. Requirement $(iiii)$ is solved by manipulating v_{nom}, where we set v_{nom} to maintain a safety margin corresponding to 2 s. Hence, this 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. The allowed deviations from the specifications are set to $R = \text{diag}(4, 4, 2)$, where $\text{diag}(\cdot)$ is the diagonal matrix.

A. Results

1) Obstacle-Free Motion Planning: When there are no predicted obstacles present in the vicinity of the ego vehicle, it should ideally track the middle of the right lane perfectly. Fig. 1 shows computed path, resulting particles (nodes for UNIFORM), and best path for two parameter choices of Algorithm 1 and input-based RRT using uniform input sampling. The nodes are more scattered with uniform sampling, and it is clear that the cost function must be carefully designed to give satisfactory tracking without biased sampling.

Fig. 2 shows the corresponding velocities, which should ideally equal $v_{\text{nom}} = 25$ m/s. Fig. 2 highlights a well-known problem with traditional RRT: when there is a hard computation-time limit, RRT will sometimes not provide a rich enough set of paths to give desired performance. When introducing PFs for guiding the samples, this problem is clearly suppressed. Both parameter setups result in almost perfect lane following.

2) Overtaking: Figs. 3 and 4 show two snapshots of a situation where the autonomous vehicle catches up with a vehicle. In Fig. 3, two sets of trajectories are computed—one where the vehicle maintains the right lane and one where the autonomous vehicle initiates overtaking of the slower moving vehicle. The resulting path is smooth for both $N = 100$ and $N = 10$, although the two branches of the tree are more distinct and the middle lane is better tracked for $N = 100$. This also holds for when reentering the right lane when the overtaking is concluded (Fig. 4). Parts of the trajectories in the respective left lane in Fig. 4 are from the previous iteration, where the aim was to track the left lane. In the lower plot, they are extended toward the right lane.

3) Blocked Lanes: Fig. 5 displays the distance to the preceding vehicle in the same lane (upper plot) and the corresponding velocity profile (lower plot) when both lanes are blocked by vehicles that travel at 20 m/s. The speed of the ego vehicle converges to that of the preceding vehicle, and a safety distance corresponding to the desired 2 s is achieved.

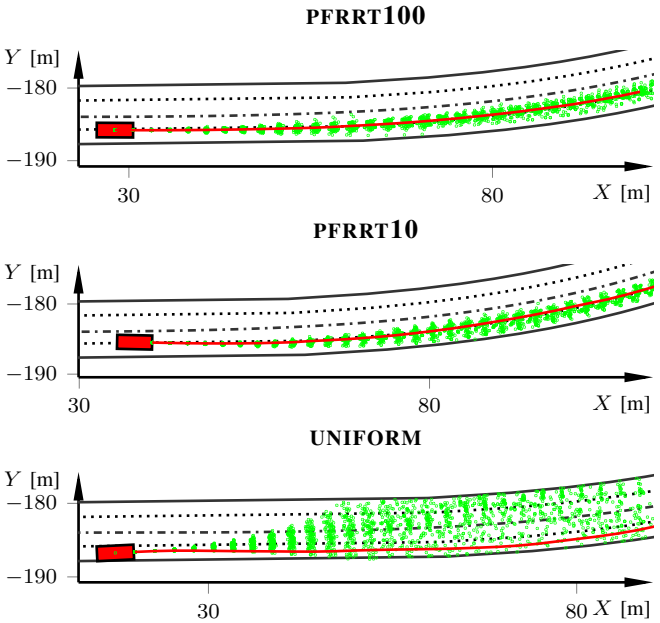

Fig. 1. Snapshots of generated trajectories (green) and best trajectory (red) for motion planning using $N = 100$ particles (upper), $N = 10$ particle in the tree expansion (middle), and RRT with uniform sampling (lower). The cost function is a combination of deviations from the right middle lane (dotted line) and v_{nom}.

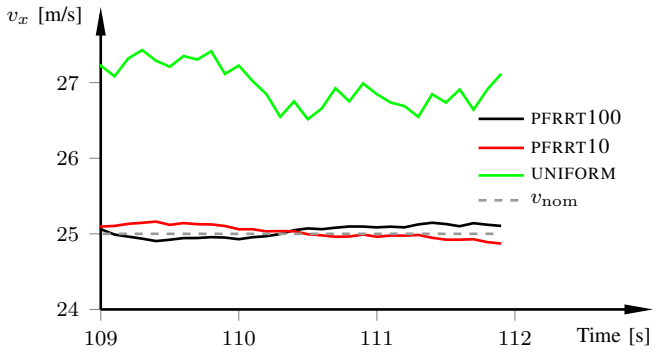

Fig. 2. Generated velocities corresponding to Fig. 1 with $v_{\text{nom}} = 25$ m/s. Lack of biased sampling (UNIFORM) leads to larger error and less smooth velocity profile.

V. CONCLUDING DISCUSSION

We presented an input-based RRT that incorporates particle filtering as a means to probabilistically choose the control inputs, hence achieving an efficient and nonsparse tree in the regions of most interest. An enabler is the introduction of task objectives, which allows a reformulation of the motion planning as a nonlinear estimation problem. Because the objectives are handled by a PF, highly nonlinear and/or conflicting objectives can be modeled. One example of this was the mid-lane deviation (Fig. 3). In a range of applications, there are specifications that the resulting trajectory should aim to fulfill. Our framework is therefore significantly more general than the particular example described in this paper.

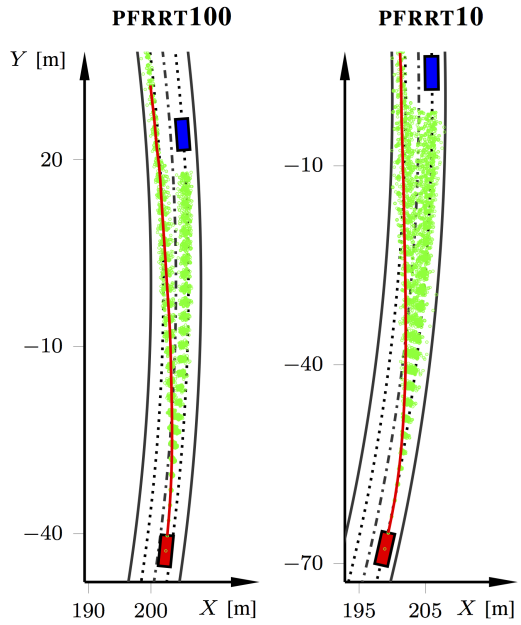

Fig. 3. Snapshot of a situation where the autonomous vehicle catches a slower moving vehicle. The motion planner computes trajectories to either stay in the lane (and slowing down) or initiates overtaking of the slow vehicle.

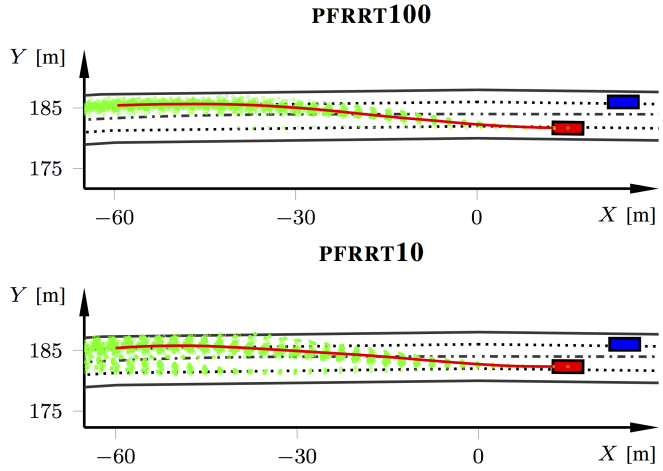

Fig. 4. Snapshot a couple of seconds after initiating overtaking in Fig. 3.

The focus has been on developing an algorithm that quickly and efficiently provides feasible trajectories, rather than finding the optimal. However, with the introduction of PF, there are relations to optimal estimation that can be further explored. It is future work to investigate convergence and optimality properties of our algorithm.

References

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

[2] 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.

[3] 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.

[4] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 425–437, 2002.

[5] K. Berntorp, "Particle filter for combined wheel-slip and vehicle-motion estimation," in *Am. Control Conf.*, Chicago, IL, Jul. 2015.

[6] 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.

[7] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. J. R. Res.*, vol. 26, no. 2, pp. 141–166, 2007.

[8] 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 *Am. Control Conf.*, Washington, DC, Jun. 2013.

[9] R. Vinter, *Optimal Control*. Boston, MA: Birkhäuser, 2010.

[10] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. R. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[11] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. R. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[12] 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.

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

[14] N. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *Int. Conf. Robot. Automation*, Rome, Italy, Apr. 2007.

[15] A. Doucet, N. De Freitas, and N. Gordon, "An introduction to sequential Monte Carlo methods," in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 3–14.

[16] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, 2002.

[17] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 25, no. 7, pp. 53–82, 2010.

[18] K. Berntorp, "Feedback particle filter: Application and evaluation," in *18th Int. Conf. Information Fusion*, Washington, DC, Jul. 2015.

[19] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. F, Radar and Signal Process.*, vol. 140, no. 2, pp. 107–113, 1993.

[20] F. Gustafsson, U. Orguner, T. B. Schön, P. Skoglar, and R. Karlsson, "Navigation and tracking of road-bound vehicles using map support," in *Handbook of Intelligent Vehicles*. Springer, 2012, pp. 397–434.

[21] S. Thrun, M. Montemerlo *et al.*, "Stanley: The robot that won the DARPA grand challenge," *J. Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

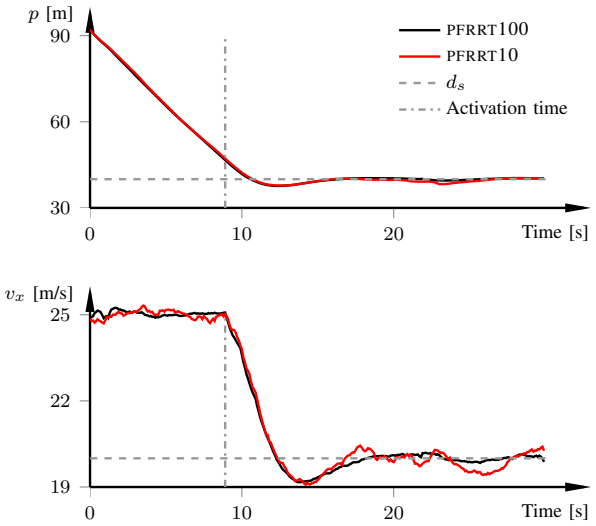

Fig. 5. Distances to preceding vehicle and corresponding velocity profiles. Both lanes are blocked by slow vehicles, so the only possibility is to slow down. A safety margin corresponding to 2 s at obstacle standstill (i.e., 40 m in stationarity) is imposed.