Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns

Noname manuscript No.
(will be inserted by the editor)

Probabilistically Safe Motion Planning to Avoid Dynamic Obstacles with Uncertain Motion Patterns

Georges S. Aoude · Brandon D. Luders · Joshua M. Joseph · Nicholas Roy · Jonathan P. How

Received: date / Accepted: date

Abstract This paper presents a real-time path planning algorithm that guarantees probabilistic feasibility for autonomous robots with uncertain dynamics operating amidst one or more dynamic obstacles with uncertain motion patterns. Planning safe trajectories under such conditions requires both accurate prediction and proper integration of future obstacle behavior within the planner. Given that available observation data is limited, the motion model must provide generalizable predictions that satisfy dynamic and environmental constraints, a limitation of existing approaches. This work presents a novel solution, named RR-GP, which builds a learned motion pattern model by combining the flexibility of Gaussian processes (GP) with the efficiency of RRT-Reach, a sampling-based reachability computation. Obstacle trajectory GP predictions are conditioned on dynamically feasible paths identified from the reachability analysis, yielding more accurate predictions of future behavior. RR-GP predictions are integrated with a robust path planner, using chance-constrained RRT, to identify probabilistically feasible paths. Theoretical guarantees of probabilistic feasibility are shown for linear systems under Gaussian uncertainty; approximations for nonlinear dynamics and/or non-Gaussian uncertainty are also presented. Simulations demonstrate that, with this planner, an autonomous vehicle can safely navigate a complex environment in real-time while significantly reducing the risk of collisions with dynamic obstacles.

G. Aoude, B. Luders, Room 41-105
J. M. Joseph, Room 32-331
N. Roy, Room 33-315
J. P. How, Room 33-326
77 Massachusetts Avenue
Tel.: +1-617-314-4375
E-mail: gaoude@alum.mit.edu, luders@mit.edu, jmjoseph@mit.edu, nickroy@mit.edu, jhow@mit.edu

Keywords Planning under uncertainty · Trajectory prediction · Gaussian processes

1 Introduction

To operate safely in stochastic environments, it is crucial for agents to be able to plan in real-time in the presence of uncertainty. However, the nature of such environments often precludes the existence of guaranteed-safe, collision-free paths. Therefore, this work considers probabilistically safe planning, in which paths must be able to satisfy all constraints with a user-mandated minimum probability. A major challenge in safely navigating such environments is how to properly address the multiple sources of external uncertainty, often classified as environment sensing (ES) and environment predictability (EP) (Lavalle and Sharma, 1997). Under this partition, ES uncertainties might be attributable to imperfect sensor measurements or incomplete knowledge of the environment, while EP uncertainties address limited knowledge of the future state of the environment. This work focuses on addressing robustness to EP uncertainty, a key challenge for existing path planning approaches (Melchior and Simmons, 2007; Fulgenzi et al., 2008; Leonard et al., 2008; Aoude et al., 2010c).

More specifically, this paper considers the problem of probabilistically safe motion planning to avoid one or more dynamic obstacles with uncertain motion patterns. While existing probabilistic planning frameworks can readily admit dynamic obstacles (Thrun et al. (2005); LaValle (2006)), such objects typically demonstrate complex motion patterns in real-world domains, making them difficult to model and predict. For instance, to reliably traverse a busy intersection, an autonomous vehicle would need to predict the underlying intents of the surrounding vehicles (*e.g.,* turning right vs. going straight), in addition to estimating the possible trajec-

tories corresponding to each intent. Even with perfect sensors, accurately predicting possible variations in the long-term trajectories of other mobile agents remains a difficult problem.

One of the main objectives of this work is to accurately model and predict the future behavior of dynamic obstacles in structured environments, such that an autonomous agent can identify trajectories which safely avoid such obstacles. In order to provide long-term trajectory predictions, this work uses pattern-based approaches for modeling the evolution of dynamic obstacles, including clustering of observations (Section 2). Such algorithms group previously-observed trajectories into clusters, with each represented by a single trajectory prototype (Bennewitz et al., 2005); predictions are then performed by comparing the partial path to each prototype. While this reduces the dependence on expert knowledge, selecting a model which is sufficiently representative of the behavior without over-fitting remains a key challenge.

In previous work (Joseph et al., 2010, 2011), the authors presented a Bayesian nonparametric approach for modeling dynamic obstacles with unknown motion patterns. This nonparametric model, a mixture of Gaussian processes (GP), generalizes well from small amounts of data and allows the model to capture complex trajectories as more data is collected. However, in practice, GPs suffer from two interconnected shortcomings: their high computational cost and their inability to embed static feasibility or vehicle dynamical constraints. To address both problems simultaneously, this work introduces the RR-GP algorithm, a clustering-based trajectory prediction solution which uses Bayesian nonparametric reachability trees to improve the original GP prediction. Similar to GPs, RR-GP is a data-driven approach, using observed past trajectories of the dynamic obstacles to learn a motion pattern model. By conditioning the obstacle trajectory predictions obtained via GPs on a reachability-based simulation of dynamically feasible paths (Aoude et al., 2011), RR-GP yields a more accurate prediction of future behavior.

The other main objective of this paper is to demonstrate that through appropriate choice of planner, an autonomous agent can utilize RR-GP predictions to identify and execute probabilistically feasible paths in real-time, in the presence of uncertain dynamic obstacles. This agent is subject to limiting dynamical constraints, such as minimum turning rates, acceleration bounds, and/or high speeds. This work proposes a real-time path planning framework using chance-constrained rapidly exploring random trees, or CC-RRT (Luders et al., 2010b), to guarantee probabilistic feasibility with respect to the dynamic obstacles and other constraints. CC-RRT extends previously-developed chance constraint formulations (Blackmore et al., 2006; Calafiore and Ghaoui, 2007), which efficiently evaluate bounds on the risk of constraint violation at each timestep, to incorporate an RRT-based framework. By applying RRT to solve this risk-constrained planning problem, this planning algorithm is able to rapidly identify probabilistically safe trajectories online in a dynamic and constrained environment. As a sampling-based algorithm (LaValle (2006)), RRT incrementally constructs trajectories which satisfy all problem constraints, including the probabilistic feasibility guarantees, and thus scales favorably with problem complexity.

The proposed planning algorithm tightly integrates CC-RRT with the RR-GP algorithm, which provides a likelihood and time-varying Gaussian state distribution for each possible behavior of a dynamic obstacle at each future timestep. This work shows that probabilistic feasibility can be guaranteed for a linear system subject to such uncertainty. An alternative, particle-based approximation which admits the use of nonlinear dynamics and/or non-Gaussian uncertainty is also presented. Though this alternative can only approximate the feasibility guarantees, it avoids the conservatism needed to establish them theoretically. While this work focuses on intersection collision avoidance, the proposed algorithm can be applied to a variety of structured domains, such as mid-air collision avoidance and military applications.

After Section 2 presents related work, preliminaries are provided in Section 3, which establishes the problem statement, and Section 4, which reviews the GP-based motion pattern modeling approach. Section 5 presents the RR-GP algorithm, with simulation results demonstrating its effectiveness in Section 6. Section 7 extends the CC-RRT framework to integrate RR-GP trajectory predictions, allowing dynamic obstacles with multiple possible behaviors. Finally, Section 8 presents simulation results, which demonstrate the ability of the fully-integrated algorithm to significantly reduce the risk of collisions with dynamic obstacles.

2 Related Work

Modeling the evolution of dynamic obstacles can be classified into three main categories: (1) worst-case, (2) dynamic-based, and (3) pattern-based approaches (Lachner (1997); Mazor et al. (2002); Vasquez et al. (2008)). In the worst-case approach, the dynamic obstacle is assumed to be actively trying to collide with the planning agent, or "host vehicle" (Miloh and Sharma (1976); Lachner (1997)). The predicted trajectory of the dynamic obstacle is the solution of a differential game, where the dynamic obstacle is modeled as a pursuer and the host vehicle as an evader (Aoude et al. (2010a)). Despite providing a lower bound on safety, such solutions are inherently conservative, and thus limited to short time horizons in collision warning/mitigation problems to keep the level of false positives below a reasonable threshold (Kuchar and Yang (2002)).

The dynamic-based approach predicts an obstacle's future trajectory by propagating its dynamics forward in time, based on its current state and an assumed fixed mode of operation. This prediction typically uses a continuous Bayes filter, such as the Kalman filter or its variations (Sorenson (1985)). A popular extension in the target tracking literature is the Interacting Multiple Model Kalman filter (IMM-KF), which matches the obstacle's current mode of operation from among a bank of continuously-updated Kalman filters (Mazor et al. (2002)). Though useful for short-term prediction, dynamic-based approaches tend to perform poorly in the long-term prediction of trajectories, due to their inability to model future changes in control inputs or external factors.

In the pattern-based approach, such as the one used in this work, target obstacles are assumed to move according to typical patterns across the environment, learned via previous observation of the targets. There are two main techniques that fall under this category, discrete state-space techniques and clustering-based techniques. In the discrete state-space technique, the motion model is developed via Markov chains; the object state evolves from one state to another according to a learned transition probability (Zhu (2002)). In the clustering-based technique, previously-observed trajectories are grouped into different clusters, with each represented by a single trajectory prototype (Bennewitz et al. (2005)). Given a partial path, prediction is then performed by finding the most likely cluster, or computing a probability distribution over the different clusters. Both pattern-based techniques have proven popular in solving long-term prediction problems for mobile agents (Fulgenzi et al. (2008); Vasquez et al. (2008)). However, discrete state-space techniques can often suffer from over-fitting for discretizations of sufficient resolution, unlike clustering-based techniques (Joseph et al. (2011)).

Many existing approaches in the literature seek to emulate human-like navigation in crowded environments, where obstacle density is high and interaction between agents and obstacles can significantly influence behavior. Trautman and Krause (2010) use GPs to model interactions between the agent and dynamic obstacles present in the environment. Althoff et al. (2011) use Monte Carlo sampling to estimate inevitable collision states probabilistically, while Henry et al. (2010) apply inverse reinforcement learning for human-like behavior. By contrast, our algorithm considers constrained, often non-holonomic agents operating in structured environments, where encounters with dynamic obstacles are less frequent but more heavily constrained. The proposed algorithm is similar to Fulgenzi et al. (2008), which uses GPs to model moving obstacles in an RRT path planner; however, Fulgenzi et al. (2008) relies solely on GPs for its modeling, which can lead to less accurate prediction, and uses heuristics to assess path safety.

While several approaches have been previously proposed for path planning with probabilistic constraints, the approach developed in this work does not rely on the use of an optimization-based framework (Blackmore et al. (2006); Calafiore and Ghaoui (2007)). While such optimizations have been demonstrated for real-time path planning, they lack the scalability with respect to problem complexity inherent to sampling-based algorithms, a crucial consideration in complex and dynamic environments. For example, MILP-based optimizations – NP-hard in the number of binary variables (Garey and Johnson (1979)) – tend to scale poorly in the number of obstacles and timesteps, resulting in many approaches being proposed specifically to overcome MILP's computational limits (Earl and D'Andrea (2005); Vitus et al. (2008); Ding et al. (2011)). Because sampling-based algorithms such as CC-RRT perform trajectory-wise constraint checking, they avoid these scalability concerns – feasible solutions can typically be quickly identified even in the presence of many obstacles, and observed changes in the environment. The trade-off is that such paths do not satisfy any optimality guarantees, though performance will improve as more sampled trajectories are made available. Extensions such as RRT* (Karaman and Frazzoli (2009)) can provide asymptotic optimality guarantees, with the trade-off of requiring additional per-node computation (in particular, a steering method).

CC-RRT primarily assesses the probabilistic feasibility at each timestep, rather than over the entire path. Because of the dynamics, the uncertainty is correlated, and thus the probability of path feasibility cannot be approximated by assuming independence between timesteps. While path-wise bounds on constraint violation can be established by evenly allocating risk margin across all obstacles and timesteps (Blackmore (2006)), this allocation significantly increases planning conservatism, rendering the approach intractable for most practical scenarios. Though this allocation could also be applied to CC-RRT by bounding the timestep horizon length, it is not pursued further in this work.

This work also proposes an alternative, particle-based approximation of the uncertainty within CC-RRT, assessing path feasibility based on the fraction of feasible particles. Both the approaches of Blackmore et al. (2010) and particle CC-RRT (PCC-RRT) are able to admit non-Gaussian probability distributions and approximate path feasibility, without the conservatism introduced by bounding risk. The optimization-versus-sampling considerations here are the same as noted above; as a sampling-based algorithm, particle CC-RRT can also admit nonlinear dynamics without an appreciable increase in complexity. While the particle-based CC-RRT algorithm is similar to the work developed in Melchior and Simmons (2007), the former is generalizable both in the types of probabilistic feasibility that are assessed (timestep-wise and path-wise) and in the types of uncer-

tainty that are modeled using particles. This framework can be extended to consider hybrid combinations of particle-based and distribution-based uncertainty, though this may limit the ability to assess path-wise infeasibility. Furthermore, by not clustering particles, a one-to-one mapping between inputs and nodes is maintained.

3 Problem Statement

Consider a discrete-time linear time-invariant (LTI) system with process noise,

$$x_{t+1} = Ax_t + Bu_t + w_t, \tag{1}$$
$$x_0 \sim \mathcal{N}(\hat{x}_0, P_{x_0}), \tag{2}$$
$$w_t \sim \mathcal{N}(0, P_{w_t}), \tag{3}$$

where $x_t \in \mathbb{R}^{n_x}$ is the state vector, $u_t \in \mathbb{R}^{n_u}$ is the input vector, and $w_t \in \mathbb{R}^{n_x}$ is a disturbance vector acting on the system; $\mathcal{N}(\hat{a}, P_a)$ represents a random variable whose probability distribution is Gaussian with mean $\hat{a}$ and covariance P_a. The i.i.d. random variables w_t are unknown at current and future timesteps, but have the known probability distribution Eq. (3) ($P_{w_t} \equiv P_w \ \forall \ t$). Eq. (2) represents Gaussian uncertainty in the initial state x_0, corresponding to uncertain localization; Eq. (3) represents a zero-mean, Gaussian process noise, which may correspond to model uncertainty, external disturbances, and/or other factors.

The system is subject to the state and input constraints

$$x_t \in \mathcal{X}_t \equiv \mathcal{X} - \mathcal{X}_{t1} - \cdots - \mathcal{X}_{tB}, \tag{4}$$
$$u_t \in \mathcal{U}, \tag{5}$$

where $\mathcal{X}, \mathcal{X}_{t1}, \ldots, \mathcal{X}_{tB} \subset \mathbb{R}^{n_x}$ are convex polyhedra, $\mathcal{U} \subset \mathbb{R}^{n_u}$, and the $-$ operator denotes set subtraction. The set $\mathcal{X}$ defines a set of time-invariant convex constraints acting on the state, while $\mathcal{X}_{t1}, \ldots, \mathcal{X}_{tB}$ represent B convex, possibly time-varying obstacles to be avoided. Observations of dynamic obstacles are assumed to be available, such as through a vehicle-to-vehicle or vehicle-to-infrastructure system.

For each obstacle, the shape and orientation are assumed known, while the placement is uncertain:

$$\mathcal{X}_{tj} = \mathcal{X}_j^0 + c_{tj}, \ \forall \ j \in \mathbb{Z}_{1,B}, \ \forall \ t, \tag{6}$$
$$c_{tj} \sim p(c_{tj}) \ \forall \ j \in \mathbb{Z}_{1,B}, \ \forall \ t, \tag{7}$$

where the $+$ operator denotes set translation and $\mathbb{Z}_{a,b}$ represents the set of integers between a and b inclusive. In this model, $\mathcal{X}_j^0 \subset \mathbb{R}^{n_x}$ is a convex polyhedron of known, fixed shape, while $c_{tj} \in \mathbb{R}^{n_x}$ is an uncertain and possibly time-varying translation represented by the probability distribution $p(c_{tj})$. This can represent both environmental sensing uncertainty (Luders et al., 2010b) and environmental predictability uncertainty (*e.g.,* dynamic obstacles), as long as future state distributions are known (Section 7.2).

The objective of the planning problem is to reach the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$ in minimum time,

$$t_{\text{goal}} = \inf\{t \in \mathbb{Z}_{0,t_f} \mid x_t \in \mathcal{X}_{\text{goal}}\}, \tag{8}$$

while ensuring the constraints in Eqs. (4)-(5) are satisfied at each timestep $t \in \{0, \ldots, t_{\text{goal}}\}$ *with probability of at least* p_{safe}. In practice, due to state uncertainty, it is assumed sufficient for the distribution mean to reach the goal region $\mathcal{X}_{\text{goal}}$. A penalty function $\psi(x_t, \mathcal{X}_t, \mathcal{U})$ may also be incorporated.

Problem 1. Given the initial state distribution $(\hat{x}_0, P_{x_0})$ and constraint sets $\mathcal{X}_t$ and $\mathcal{U}$, compute the input control sequence $u_t, \ t \in \mathbb{Z}_{0,t_f}, \ t_f \in \mathbb{Z}_{0,\infty}$ that minimizes

$$J(\mathbf{u}) = t_{\text{goal}} + \sum_{t=0}^{t_{\text{goal}}} \psi(x_t, \mathcal{X}_t, \mathcal{U}) \tag{9}$$

while satisfying Eq. (1) for all timesteps $t \in \{0, \ldots, t_{\text{goal}}\}$ and Eqs. (4)-(5) at each timestep $t \in \{0, \ldots, t_{\text{goal}}\}$ with probability of at least p_{safe}. □

3.1 Motion Pattern

A motion pattern is defined here as a mapping from states to a distribution over trajectory derivatives.[1] In this work, motion patterns are used to represent dynamic obstacles, also referred to as agents. Given an agent's position (x_t, y_t) and trajectory derivative $(\frac{\Delta x_t}{\Delta t}, \frac{\Delta y_t}{\Delta t})$, its predicted next position (x_{t+1}, y_{t+1}) is $(x_t + \frac{\Delta x_t}{\Delta t}\Delta t, y_t + \frac{\Delta y_t}{\Delta t}\Delta t)$. Thus, modeling trajectory derivatives is sufficient for modeling trajectories. By modeling motion patterns as flow fields rather than single paths, the approach is independent of the lengths and discretizations of the trajectories.

3.2 Mixtures of Motion Patterns

The finite mixture model[2] defines a distribution over the ith observed trajectory t^i. This distribution is written as

$$p(t^i) = \sum_{j=1}^{M} p(b_j) p(t^i | b_j), \tag{10}$$

where b_j is the jth motion pattern and $p(b_j)$ is its prior probability. It is assumed the number of motion patterns, M, is known *a priori* based on prior observations, and may be identified by the operator or through an automated clustering process (Joseph et al. (2011)).

[1] The choice of Δt determines the time scales on which an agent's next position can be accurately predicted, making trajectory derivatives more useful than instantaneous velocity.

[2] Throughout the paper, a t with a superscript refers to a trajectory, while a t without a superscript refers to a time value.

4 Motion Model

The motion model is defined as the mixture of weighted motion patterns (10). Each motion pattern is weighted by its probability and is modeled by a pair of Gaussian processes mapping (x, y) locations to distributions over trajectory derivatives $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$. This motion model has been previously presented in Aoude et al. (2011), Joseph et al. (2011); Sections 4.1 and 4.2 briefly review the approach.

4.1 Gaussian Process Motion Patterns

This section describes the model for $p(t^i|b_j)$ from Eq. (10), the probability of trajectory t^i given motion pattern b_j. This model is the distribution over trajectories expected for a given mobility pattern.

There are a variety of models that can be chosen to represent these distributions. A simple example is a linear model with Gaussian noise, but this approach cannot capture the dynamics of the variety expected in this work. Discrete Markov models are also commonly used, but are not well-suited to model mobile agents in the types of real-world domains of interest here, particularly due to challenges in choosing the discretization (Tay and Laugier, 2007; Joseph et al., 2010, 2011). To fully represent the variety of trajectories that might be encountered, a fine discretization is required. However, such a model either requires a large amount of training data, which is costly or impractical in real-world domains, or is prone to over-fitting. A coarser discretization can be used to prevent over-fitting, but may be unable to accurately capture the agent's dynamics.

This work uses Gaussian processes (GP) as the model for motion patterns. Although GPs have a significant mathematical and computational cost, they provide a natural balance between generalization in regions with sparse data and avoidance of overfitting in regions of dense data (Rasmussen and Williams (2005)). GP models are extremely robust to unaligned, noisy measurements and are well-suited for modeling the continuous paths underlying potentially non-uniform time-series samples of the agent's locations. Trajectory observations are discrete measurements from its continuous path through space; a GP places a distribution over functions, serving as a non-parametric form of interpolation between these discrete measurements.

After observing an agent's trajectory t^i, the posterior probability of the jth motion pattern is

$$p(b_j|t^i) \propto p(t^i|b_j)p(b_j), \tag{11}$$

where $p(b_j)$ is the prior probability of motion pattern b_j and $p(t^i|b_j)$ is the probability of trajectory t^i under b_j. This distribution, $p(t^i|b_j)$, is computed by

$$p(t^i|b_j) = \prod_{t=0}^{L^i} p\left(\frac{\Delta x_t}{\Delta t}\middle| x^i_{0:t}, y^i_{0:t}, \{t^k : z_k = j\}, \theta^{GP}_{x,j}\right) \cdot p\left(\frac{\Delta y_t}{\Delta t}\middle| x^i_{0:t}, y^i_{0:t}, \{t^k : z_k = j\}, \theta^{GP}_{y,j}\right), \tag{12}$$

where L^i is the length of trajectory t^i, z_k indicates the motion pattern trajectory t^k is assigned to, $\{t^k : z_k = j\}$ is the set of all trajectories assigned to motion pattern j, and $\theta^{GP}_{x,j}$ and $\theta^{GP}_{y,j}$ are the hyperparameters of the Gaussian process for motion pattern b_j.

A motion pattern's GP is specified by a set of mean and covariance functions. The mean functions are written as $E\left[\frac{\Delta x}{\Delta t}\right] = \mu_x(x, y)$ and $E\left[\frac{\Delta y}{\Delta t}\right] = \mu_y(x, y)$, both of which are implicitly initialized to zero for all x and y by the choice of parametrization of the covariance function. This encodes the prior bias that, without any additional knowledge, the target is expected to stay in the same place. The "true" covariance function of the x-direction is denoted by $K_x(x, y, x', y')$, which describes the correlation between trajectory derivatives at two points, (x, y) and (x', y'). Given locations $(x_1, y_1, .., x_k, y_k)$, the corresponding trajectory derivatives $(\frac{\Delta x_1}{\Delta t}, .., \frac{\Delta x_k}{\Delta t})$ are jointly distributed according to a Gaussian with mean $\{\mu_x(x_1, y_1), .., \mu_x(x_k, y_k)\}$ and covariance Σ, where the $\Sigma_{ij} = K_x(x_i, y_i, x_j, y_j)$. This work uses the squared exponential covariance function

$$K_x(x, y, x', y') = \sigma_x^2 \exp\left(-\frac{(x - x')^2}{2{w_x}^2} - \frac{(y - y')^2}{2{w_y}^2}\right) + \sigma_n^2 \delta(x, y, x', y'), \tag{13}$$

where $\delta(x, y, x', y') = 1$ if $x = x'$ and $y = y'$ and zero otherwise. The exponential term encodes that similar trajectories should make similar predictions, while the length-scale parameters w_x and w_y normalize for the scale of the data. The σ_n-term represents within-point variation (e.g., due to noisy measurements); the ratio of σ_n and σ_x weights the relative effects of noise and influences from nearby points. Here $\theta^{GP}_{x,j}$ is used to refer to the set of hyperparameters σ_x, σ_n, w_x, and w_y associated with motion pattern b_j (each motion pattern has a separate set of hyperparameters). While the covariance is written above for two dimensions, it can easily be generalized to higher dimensional problems.

For a GP over trajectory derivatives trained with tuples $(x_k, y_k, \frac{\Delta x_k}{\Delta t})$, the predictive distribution over the trajectory derivative $\frac{\Delta x}{\Delta t}^*$ for a new point (x^*, y^*) is given by

$$\mu_{\frac{\Delta x}{\Delta t}^*} = K_x(x^*,y^*,X,Y)K_x(X,Y,X,Y)^{-1}\frac{\Delta X}{\Delta t}, \tag{14}$$
$$\sigma^2_{\frac{\Delta x}{\Delta t}^*} = K_x(x^*,y^*,X,Y)K_x(X,Y,X,Y)^{-1}K_x(X,Y,x^*,y^*),$$

where the expression $K_x(X, Y, X, Y)$ is shorthand for the covariance matrix Σ with terms $\Sigma_{ij} = K_x(x_i, y_i, x_j, y_j)$,

with $\{X, Y\}$ representing the previous trajectory points. The equations for $\frac{\Delta y}{\Delta t}^*$ are equivalent to those above, using the covariance K_y.

In Eq. (12), the likelihood is assumed to be decoupled, and hence independent, in each position coordinate. This enables decoupled GPs to be used for the trajectory position in each coordinate, dramatically reducing the required computation, and is assumed in subsequent developments. While the algorithm permits the use of correlated GPs, the resulting increase in modeling complexity is generally intractable for real-time operation. Simulation results (Section 8) demonstrate that the decoupled GPs provide a sufficiently accurate approximation of the correlated GP to achieve meaningful predictions of future trajectories.

4.2 Estimating Future Trajectories

The Gaussian process motion model can be used to calculate the Gaussian distribution over trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ for every location (x, y) using Eq. (14). This distribution over the agent's next location can be used to generate longer-term predictions over future trajectories, but not in closed form. Instead, the proposed approach is to draw trajectory samples to be used for the future trajectory distribution.

To sample a trajectory from a current starting location (x_0, y_0), first a trajectory derivative $(\frac{\Delta x_0}{\Delta t}, \frac{\Delta y_0}{\Delta t})$ is sampled to calculate the agent's next location (x_1, y_1). Starting from (x_1, y_1), the trajectory derivative $(\frac{\Delta x_1}{\Delta t}, \frac{\Delta y_1}{\Delta t})$ is sampled to calculate the agent's next location (x_2, y_2). This process is repeated until the trajectory is of the desired length L. The entire sampling procedure is then repeated from (x_0, y_0) multiple times to obtain a set of possible future trajectories. Given a current location (x_t, y_t) and a given motion pattern b_j, the agent's predicted position K timesteps into the future is computed as

$$
\begin{aligned}
&p(x_{t+K}, y_{t+K} | x_t, y_t, b_j) \\
&= \prod_{k=0}^{K-1} p(x_{t+k+1}, y_{t+k+1} | x_{t+k}, y_{t+k}, b_j) \\
&= \prod_{k=0}^{K-1} p\left(\frac{\Delta x_{t+k+1}}{\Delta t}, \frac{\Delta y_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) \\
&= \prod_{k=0}^{K-1} p\left(\frac{\Delta x_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) \\
&\qquad \cdot p\left(\frac{\Delta y_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) \\
&= \prod_{k=0}^{K-1} \mathcal{N}\left(x_{t+k+1}; \mu_{j, \frac{\Delta x_{t+k+1}}{\Delta t}}, \sigma^2_{j, \frac{\Delta x_{t+k+1}}{\Delta t}} \right) \\
&\qquad \cdot \mathcal{N}\left(y_{t+k+1}; \mu_{j, \frac{\Delta y_{t+k+1}}{\Delta t}}, \sigma^2_{j, \frac{\Delta y_{t+k+1}}{\Delta t}} \right), \qquad (15)
\end{aligned}
$$

where the Gaussian distribution parameters are calculated using Eq. (14). When this process is done online, the trajectory's motion pattern b_j will not be known directly. Given the past observed trajectory $(x_0, y_0), ..., (x_t, y_t)$, the distribution can be calculated K timesteps into the future by combining Eqs. (10) and (15). Formally,

$$
\begin{aligned}
&p(x_{t+K}, y_{t+K} | x_{0:t}, y_{0:t}) \\
&= \sum_{j=1}^{M} p(x_{t+K}, y_{t+K} | x_{0:t}, y_{0:t}, b_j) p(b_j | x_{0:t}, y_{0:t}) \qquad (16) \\
&= \sum_{j=1}^{M} p(x_{t+K}, y_{t+K} | x_t, y_t, b_j) p(b_j | x_{0:t}, y_{0:t}), \qquad (17)
\end{aligned}
$$

where $p(b_j | x_{0:t}, y_{0:t})$ is the probability of motion pattern b_j given the observed portion of the trajectory. The progression from Eq. (16) to Eq. (17) is based on the assumption that, given b_j, the trajectory's history provides no additional information about the future location of the agent (Joseph et al., 2010, 2011).

The GP motion model over trajectory derivatives in this paper gives a Gaussian distribution over possible target locations at each timestep. While samples drawn from this procedure are an accurate representation of the posterior over trajectories, sampling N_1 trajectories N_2 steps in the future requires $N_1 \times N_2$ queries to the GP. It also does not take advantage of the unimodal, Gaussian distributions being used to model the trajectory derivatives. By using the approach of Girard et al. (2003) and Deisenroth et al. (2009), which provides a fast, analytic approximation of the GP output given the input distribution, future trajectories are efficiently predicted in this work. In particular, given the inputs of a distribution on the target position at time t, and a distribution of trajectory derivatives, the approach yields a distribution on the target position at time $t + 1$, effectively linking the Gaussian distributions together.

By estimating the target's future trajectories analytically, only N_2 queries to the GP are needed to predict trajectories N_2 steps into the future, and the variance introduced by sampling future trajectories is avoided. This facilitates the use of GPs for accurate and efficient trajectory prediction.

5 RR-GP Trajectory Prediction Algorithm

Section 4 outlined the approach of using GP mixtures to model mobility patterns and its benefits over other approaches. However, in practice, GPs suffer from two interconnected shortcomings: their high computational cost and their inability to embed static feasibility or vehicle dynamical constraints. Since GPs are based on statistical learning, they are unable to model prior knowledge of road

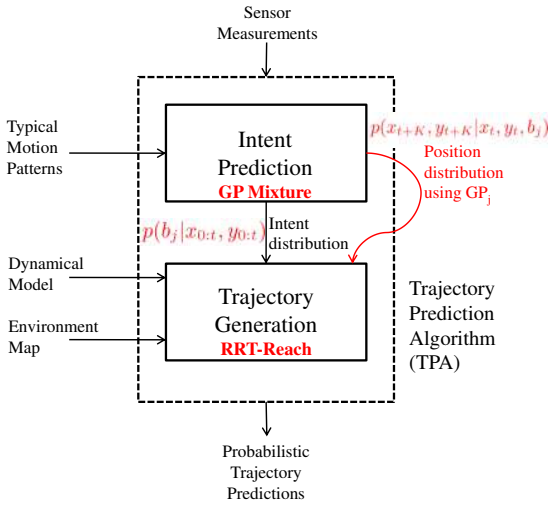


Fig. 1 RR-GP high level architecture

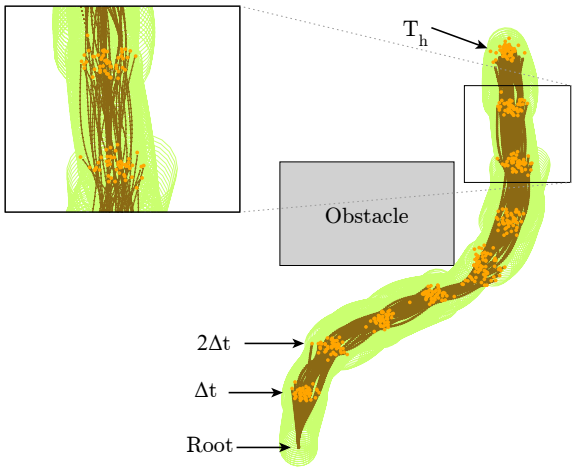


Fig. 2 Simple RR-GP illustration ($M = 1$). RR-GP grows a tree (in *brown*) using GP samples (*orange dots*) sampled at Δt intervals for a given motion pattern. The *green circles* represent the actual size of the target vehicle. The resulting tree provides a distribution of predicted trajectories of the target at $\delta t \ll \Delta t$ increments.

boundaries, static obstacle location, or dynamic feasibility constraints (*e.g.,* minimum turning radius). Very dense training data may alleviate this feasibility problem by capturing, in great detail, the environment configuration and physical limitations of the vehicle. Unfortunately, the computation time for predicting future trajectories using the resulting GPs would suffer significantly, rendering the motion model unusable for real-time applications.

To handle both of these problems simultaneously, this section introduces a novel trajectory prediction algorithm, denoted as RR-GP (Figure 1). RR-GP includes two main components, rapidly-exploring random trees (RRT), and a GP-based mobility model. For each GP-based pattern b_j, $j \in \{1, \ldots, M\}$ as defined in Section 4, and the current position of the target vehicle, RR-GP uses an RRT-based technique to grow a tree of trajectories that follows b_j while guaranteeing dynamical feasibility and collision avoidance. More specifically, it is based on the closed-loop RRT (CL-RRT) algorithm (Kuwata et al. (2009)), successfully used by the MIT team in the 2007 DARPA Grand Challenge (Leonard et al., 2008). CL-RRT grows a tree by randomly sampling points in the environment and simulating dynamically feasible trajectory towards them in closed-loop, allowing the generation of smoother trajectories more efficiently than traditional RRT algorithms.

In contrast to the original CL-RRT approach, the RR-GP tree is used not to create paths leading to a goal location, but instead to grow trees toward regions corresponding to the learned mobility patterns, or intents, b_j (Figure 2). RR-GP does not approximate the complete reachability set of the target vehicle; instead, it generates a tree based on each potential motion pattern, and computes the likelihood of each based on the observed partial path. In this manner, RR-GP conditions the original GP prediction by removing infeasible patterns and providing a finer discretization, resulting in better trajectory prediction.

Algorithm 1 RR-GP, Single Tree Expansion

1: Initialize tree $\mathcal{T}_{GP}$ with node at $(x(t), y(t))$; $gp \leftarrow$ GP motion pattern b; $t_{GP} \leftarrow t + \Delta t$; $K \leftarrow 1$; $n_{\text{success}} \leftarrow 0$; $n_{\text{infeas}} \leftarrow 0$
2: **while** $t_{GP} - t \leq T_h$ **do**
3: Take sample x_{samp} from gp initialized with $(x(t), y(t))$, K timesteps in the future using Eq. (15), and variance heuristics if necessary
4: Among nodes added at $t_{GP} - \Delta t$, identify N nearest to x_{samp} using distance heuristics
5: **for** each nearest node, in the sorted order **do**
6: Extend $\mathcal{T}_{GP}$ from nearest node using propagation function until it reaches x_{samp}
7: **if** propagated portion is collision free **then**
8: Add sample to $\mathcal{T}_{GP}$ and create intermediate nodes as appropriate
9: Increment successful connection count n_{success}
10: **else**
11: Increment infeasible connection count n_{infeas} and if limit is reached **goto** line 18
12: **end if**
13: **end for**
14: **if** n_{success} reached desired target **then**
15: $t_{GP} \leftarrow t_{GP} + \Delta t$; $K \leftarrow K + 1$; $n_{\text{success}} \leftarrow 0$; $n_{\text{infeas}} \leftarrow 0$
16: **end if**
17: **end while**
18: **return** $\mathcal{T}_{GP}$

5.1 Single Tree RR-GP Algorithm

Algorithm 1 details the single-tree RR-GP algorithm, which constructs a tree of dynamically feasible motion segments to generate a distribution of predicted trajectories for a given intent b, dynamical model, and low-level controller. Every time Algorithm 1 is called, a tree $\mathcal{T}_{GP}$

is initialized with a root node at the current target vehicle position $(x(t), y(t))$. Variable t_{GP}, which is used for time bookkeeping in the expansion mechanism of $\mathcal{T}_{GP}$, is also initialized to $t + \Delta t$, where t is the current time, and Δt is the GP sampling time interval.

At each step, only nodes belonging to the "time bucket" $t_{GP} - \Delta t$ are eligible to be expanded, corresponding to the nodes added at the previous timestep t_{GP}. To grow $\mathcal{T}_{GP}$, first a sample x_{samp} is taken from the environment (line 3) at time $t_{GP} + K\Delta t$, or equivalently K timesteps in the future, using Eq. (15). The nodes added in the previous step (*i.e.,* belonging to time bucket $t_{GP} - \Delta t$) are identified for tree expansion in terms of some distance heuristics (line 4). The algorithm attempts to connect the nearest node to the sample using an appropriate reference path for the closed-loop system consisting of the vehicle and the controller. The resulting path is dynamically feasible (by construction) and is then checked for collisions. Since the $\mathcal{T}_{GP}$ is trying to generate typical trajectories that the target vehicle might follow, only a simulated trajectory that reaches the sample without a collision is kept, and any corresponding nodes are added to the tree and t_{GP} time bucket (line 8). When the total number of successful connections n_{success} is reached, t_{GP} and K are incremented, and the next timestep is sampled (line 15).

RR-GP keeps track of the total number of unsuccessful connections n_{infeas} at each iteration. When n_{infeas} reaches some predetermined threshold, the variance of the GP for the current iteration is temporarily grown to capture a more dispersed set of paths (line 3). This heuristic is typically useful to generate feasible trajectories when GP samples are close to obstacles. If n_{infeas} then reaches a second, larger threshold, RR-GP "gives up" on growing the tree, and simply returns $\mathcal{T}_{GP}$ (line 11). This situation usually happens when the mobility pattern b has a low likelihood and is generating a large number of GP samples in infeasible regions. These heuristics, along with the time bucket logic, facilitate efficient feasible trajectory generation in RR-GP.

The resulting tree is post-processed to produce a dense, time-parametrized distribution of the target vehicle position at future timesteps. Since the RR-GP tree is grown at a higher rate compared to the original GP learning phase, the resulting distribution is generated at $\delta t \ll \Delta t$ increments, where δt is the low-level controller rate. The result is a significant improvement of the accuracy of the prediction without a significant increase in the computation times (Section 6).

5.2 Multi-Tree RR-GP Algorithm

This section introduces the Multi-Tree RR-GP (Algorithm 2), which extends Algorithm 1 to consider multiple motion patterns for a dynamic obstacle. The length of the prediction problem is T seconds, and the prediction time horizon is T_h seconds. The value of T_h is problem-specific, and depends on the time length of the training data. For example, in a threat assessment problem for road intersections, T_h will typically be on the order of 3 to 6 seconds (Aoude et al., 2010b). The RR-GP algorithm updates its measurement of the target vehicle every dt seconds, chosen such that the inner loop (lines 6-9) of Algorithm 2 reaches completion before the next measurement update. Finally, the time period of the low-level controller is equal to δt, typically $0.02 - 0.1$ s for the problems of interest.

The input to Algorithm 2 is a set of GP motion patterns, along with a prior probability distribution, proportional to the number of observed trajectories belonging to each pattern. In line 4, the position of the target vehicle is measured. Then, the probability that the vehicle trajectory belongs to each of the M motion patterns is updated using Eq. (11). For each motion pattern (in parallel), line 7 grows a single-tree RR-GP rooted at the current position of the target vehicle using Algorithm 1. The means and variances of the predicted positions are computed for each motion pattern

Algorithm 2 RR-GP, Multi-Tree Trajectory Prediction

1: Inputs: GP motion pattern b_j; $p(b_j(0)) \ \ \forall j \in [1, \ldots, M]$
2: $t \leftarrow 0$
3: **while** $t < T$ **do**
4: Measure target vehicle position $(x(t), y(t))$
5: Update probability of each motion pattern $p(b_j(t)|x_{0:t}, y_{0:t})$ using Eq. (11)
6: **for** each motion pattern b_j **do**
7: Grow a single $\mathcal{T}^j_{GP}$ tree rooted at $(x(t), y(t))$ using b_j (Algorithm 1)
8: Using $\mathcal{T}^j_{GP}$, compute means and variances of predicted distribution $(\hat{x}_j(\tau), \hat{y}_j(\tau)), \forall \tau \in [t + \delta t, t + 2\delta t, \ldots, t + T_h]$
9: **end for**
10: Adjust probability of each motion pattern using dynamic feasibility check (Algorithm 3)
11: Propagate updated probabilities backwards, and recompute $p(\hat{x}(\tau), \hat{y}(\tau)) \ \forall \tau \in [0, \delta t, \ldots, t]$ if any motion pattern probability was updated
12: $p(\hat{x}(\tau), \hat{y}(\tau)) \leftarrow \sum_j p(\hat{x}_j(\tau), \hat{y}_j(\tau)) \times p(b_j(t)|x_{0:t}, y_{0:t})$ $\forall \tau \in [t + \delta t, t + 2\delta t, \ldots, t + T_h]$ using Eq. (17)
13: $t \leftarrow t + dt$
14: **end while**

Algorithm 3 Dynamic Feasibility Adjustment

1: Inputs: GP motion pattern b_j; $\mathcal{T}^j_{GP}$ trees $\ \forall j \in [1, \ldots, M]$
2: **for** each motion pattern b_j **do**
3: **if** $\mathcal{T}^j_{GP}$ tree ended with infeasibility condition **then**
4: $\tilde{p}(b_j(t)) \leftarrow 0$
5: **else**
6: $\tilde{p}(b_j(t)) \leftarrow p(b_j(t))$
7: **end if**
8: **end for**
9: **for** each motion pattern b_j **do**
10: **if** $\sum_j \tilde{p}(b_j(t)) > 0$ **then**
11: $p(b_j(t)) \leftarrow \frac{\tilde{p}(b_j(t))}{\sum_j \tilde{p}(b_j(t))}$
12: **end if**
13: **end for**

at each timestep, using position and time information from the single-tree output (line 8). This process can be parallelized for each motion pattern, since there is no information sharing between the growth operations of each RR-GP tree. Note that even if the vehicle's position has significantly deviated from the expected GP behaviors, each GP prediction will still attempt to reconcile the vehicle's current position $(x(t), y(t))$ with the behavior.

The probability of each motion pattern is adjusted using Algorithm 3, which removes the probability of any pattern that ended in an infeasible region (line 4) due to surpassing the second n_{infeas} threshold. By using dynamic feasibility to recalculate the probabilities (line 11), this important modification helps the RR-GP algorithm converge to the more likely patterns faster, yielding an earlier prediction of the intent of the target vehicle. In the event that all RR-GP trees end with an infeasible condition, probability values are not adjusted (line 10). In practice, this infeasibility case should rarely occur, since the target vehicle is assumed to follow one of the available patterns.

If any of the motion pattern probabilities is altered, line 11 of Algorithm 2 recomputes the probability distribution of the positions of the target vehicle $(\hat{x}(t), \hat{y}(t))$ for times $\tau \in [0, \delta t, 2\delta t, \ldots, t]$. This step is called backward propagation (BP), as it propagates the effects of the updated likelihoods to the previously computed position distributions. Finally, line 12 of Algorithm 2 combines the position predictions from each single-tree RR-GP output into one distribution, by incorporating the updated motion pattern probabilities b_j. This computation is performed for all time τ where $\tau \in [t + \delta t, t+2\delta t, \ldots, t+T_h]$, resulting in a probability distribution of the future trajectories of the target vehicle that is based on a mixture of GPs.

In subsequent results, it is assumed that the target vehicle has car-like dynamics, more specifically a bicycle dynamical model (LaValle, 2006). The target vehicle inputs are approximated by the outputs of a pure-pursuit (PP) low-level controller (Amidi and Thorpe (1990)) that computes a sequence of commands towards samples generated from the GP. This path is then tracked using the propagation function of a PP controller for steering and a proportional-integral controller for tracking the GP-predicted velocity.

Figure 3 demonstrates the dynamic feasibility and collision avoidance features of the RR-GP approach on a simple example consisting of two motion patterns, corresponding to passing a single obstacle on the left or right. (Training and testing procedures of RR-GP scenarios are explained in detail in Section 6.) In this illustration, the test trajectory belongs to the left motion pattern. At each step, Algorithm 2 updates the likelihoods of each motion pattern using Eqs. (11) and (12), and generates two new dynamically feasible trees.

At time $t = 0$ s (Figure 3(d)), the target vehicle is pointing straight at the obstacle. By $t = 1$ s (Figure 3(e)), the vehicle has moved forward and rotated slightly left. Due to the forward movement and left rotation, RR-GP finds more feasible left trajectories than right trajectories (this can be seen by comparing the trajectories as they pass the corners of the obstacle), a behavior GP samples alone cannot capture (Figure 3(b)). At $t = 2$ s (Figure 3(f)), the vehicle has more clearly turned to the left; the RR-GP algorithm returns an incomplete right tree, reflecting that all attempts to grow the tree further along the right motion pattern failed.

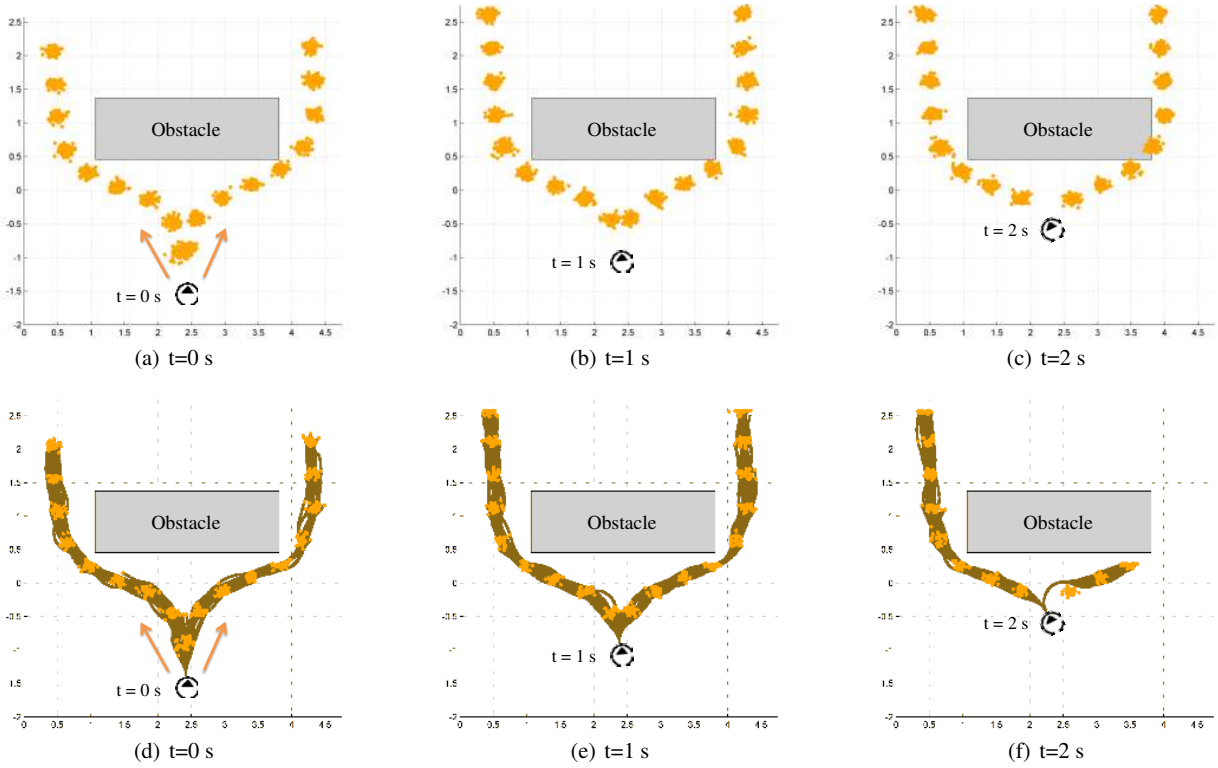
Note that the GP samples for the right pattern are not all infeasible at $t = 2$ s (Figure 3(c)); an algorithm based on GP samples alone would not have predicted the dynamic infeasibility of the right pattern. Furthermore, simple interpolation techniques would not have been able to detect dynamic infeasibility, highlighting the importance of the dynamic model embedded within the RR-GP algorithm. RR-GP also predicts that the left-side trees are dynamically feasible and reaching collision-free regions. This early detection of the correct motion pattern is a major advantage of RR-GP compared to GP algorithms alone.

Remark. *(complexity)* The proposed approach scales linearly with both the number of dynamic obstacles and the number of intents for each. However, both Algorithm 1 and Algorithm 2 can be run in parallel, with a separate process running for each potential behavior of each dynamic obstacle. Assuming the computational resources are available to implement this parallelization, runtime scaling with dynamic obstacle complexity can be effectively eliminated.

While there are no theoretical limitations with respect to the number of dynamic obstacles or motion patterns considered, in practice few are needed at any one time for the structured environments in this work's domain of interest. Complex obstacle environments can typically be broken down into a sequence of interactions with a smaller number of dynamic obstacles, while most of the "decisions" associated with motion intentions can be broken down to a series of consecutive decisions involving fewer branching paths. The trade-off in this case is that the time horizons being considered may need to be reduced.

6 RR-GP Demonstration on Human-Operated Target

To highlight the advantages of the RR-GP algorithm, Algorithm 2 is applied on an example scenario consisting of a single target vehicle traversing past several fixed obstacles. The purpose of this example is to compare the performance (in terms of accuracy and computation time) of the RR-GP approach against two baseline GP mixture algorithms, given either sparse training data (Sparse-GP) or dense training data (Dense-GP). The results below demonstrate that RR-GP, given only the same data as Sparse-GP, matches or



(a) t=0 s (b) t=1 s (c) t=2 s

(d) t=0 s (e) t=1 s (f) t=2 s

Fig. 3 Snapshots of the GP samples (*top row*) and Algorithm 2 output (*bottom row*) on a test trajectory

exceeds the accuracy of Dense-GP while maintaining the runtime benefits of sparse data.

6.1 Setup

Trajectories were manually generated by using a steering wheel to guide a simulated robot through a virtual urban environment described in Aoude et al. (2010c). The vehicle uses the iRobot Create software platform (iRobot (2011)) with a skid-steered vehicle, modified in software to emulate the standard bicycle model

$$\begin{aligned} \dot{x} &= v\cos(\theta), \quad & \dot{y} &= v\sin(\theta), \\ \dot{\theta} &= \frac{v}{L}\tan(\delta), \quad & \dot{v} &= a, \end{aligned} \tag{18}$$

where (x, y) is the rear axle position, v is the forward speed, θ is the heading, L is the wheelbase equal to 0.33 m, a is the forward acceleration, and δ is the steering angle (positive counter-clockwise). The state of the vehicle is $s = (x, y, \theta, v) \in \mathcal{S}$, while the input is $u = (\delta, a) \in \mathcal{U}$, including the constraints $a_{\min} \leq a \leq a_{\max}$ and $|\delta| \leq \delta_{\max}$, where $a_{\min} = -0.7$ m/s^2, $a_{\max} = 0.4$ m/s^2, and $\delta_{\max} = 0.6$ rad. Starting from its initial location, the vehicle is either driven to the left of the obstacle or to the right, for a total of $M = 2$ motion patterns.

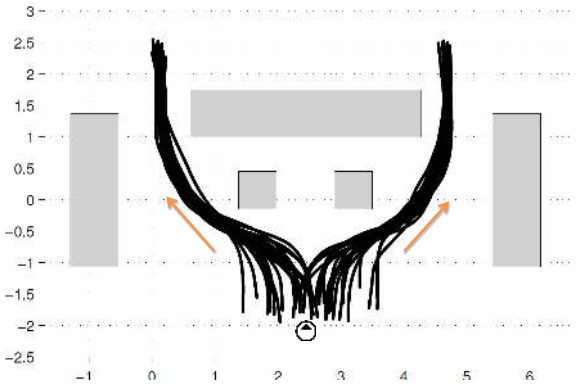


Fig. 4 Training trajectories generated in the simulated road environment according to two motion patterns. The *black circle* (*bottom*) represents the target vehicle, and the *arrow* inside the *circle* represents its heading. The *orange arrows* point in the direction of each pattern (*left* and *right*)

A total of 30 (15 left, 15 right) trajectories were generated for training, with data collected at 50 Hz (Figure 4). Each motion pattern is learned according to Eqs. (11) – (14); both RR-GP and Sparse-GP use data that were downsampled to 1 Hz, while Dense-GP was trained with 2-Hz data. The test data consists of 90 additional trajectories (45 left, 45 right) generated in the same manner as the training data. In testing, the algorithms received simulated measurements from the test trajectories at one-second intervals, *i.e.*, $dt =$

1s in Algorithm 2. For each timestep, Sparse-GP, Dense-GP, and RR-GP are run using the current state of the target vehicle for a time horizon $T_h = 8$s.

In the RR-GP implementation, the control timestep is $\delta t = 0.1$ s, while the GP samples are produced at $\Delta t = 1$s. The limits of successful and infeasible connections per Δt (lines 9 and 11 of Algorithm 1) are 30 and 150, respectively. The approach of Yepes et al. (2007) is followed in calculating prediction error as the root mean square (RMS) difference between the true position (x, y) and mean predicted position $(\hat{x}, \hat{y})$. The mean is computed using Eq. (17) for the Sparse-GP and Dense-GP techniques, and the multi-tree probability distribution for RR-GP. Prediction errors are averaged across all 90 test trajectories at each timestep.

6.2 Simulation Results

This section presents simulation results for the RR-GP algorithm which compare the prediction accuracy and computation time with both Sparse-GP and Dense-GP. Two variations of RR-GP are implemented, one with backward propagation (BP; line 11 of Algorithm 2) and one without.

6.2.1 Motion Pattern Probabilities

Figure 5 and Table 1 show the probability of identifying the correct motion pattern given the observed portion of path followed by the target vehicle, computed as a function of time. While Sparse-GP and Dense-GP only use Eq. (11) to update these probabilities, RR-GP embeds additional logic for dynamic feasibility (see Algorithm 3). Note that the probability corresponding to RR-GP without BP is the same as RR-GP with BP.

At time $t = 0$ s, Sparse-GP and Dense-GP's likelihoods are based on the size of the training data of each motion pattern. Since they are equal, the probability of the correct motion pattern is 0.5. On the other hand, by using collision checks and backward propagation RR-GP is able to improve its "guess" of the correct motion pattern from 0.5 to more than 0.92. At time $t = 1$ s, using observation of the previous target position, all three algorithms improve in accuracy, though RR-GP maintains a significant advantage. After three seconds, the probability of the correct motion pattern has nearly reached 1.0 for all algorithms.

6.2.2 Prediction Errors

Figure 6 shows the performance of each algorithm in terms of the RMS of the prediction error between the true value and the predicted mean position of the target vehicle. At the start of each test, when time $t = 0$ s (Figure 6(a)), the four algorithms are initialized with the same likelihood

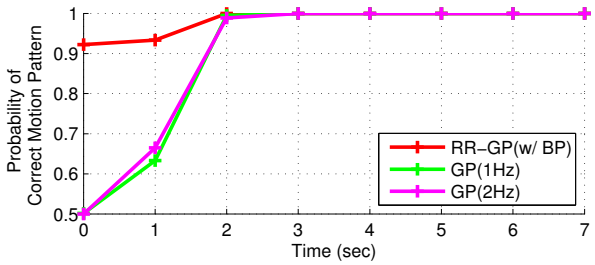
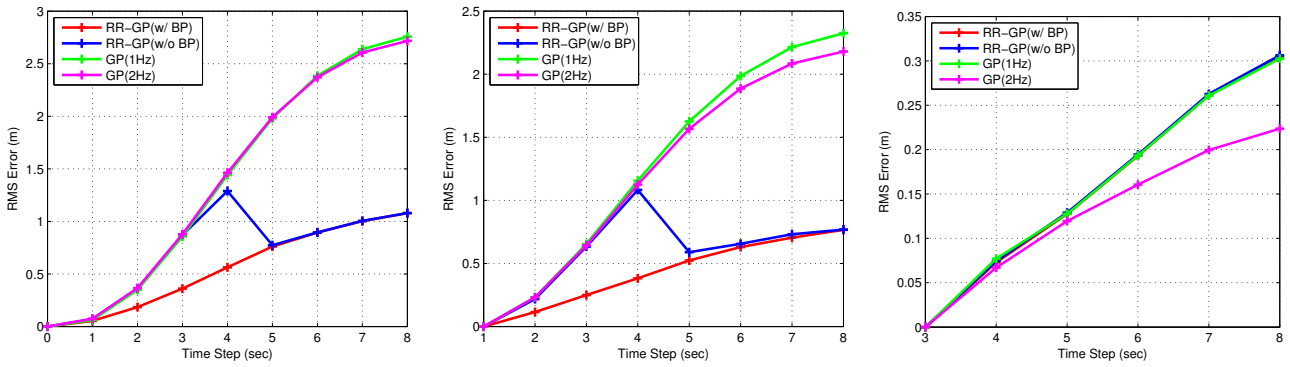


Fig. 5 Average probability (over 90 trajectories) of the correct motion pattern for RR-GP (w/ BP), Sparse-GP (1Hz), and Dense-GP (2Hz) algorithms as function of time. For example, the values at $t = 1$ s represent the probability of the correct motion patterns after the target vehicle has actually moved for one second on its path

values for each motion pattern. This is seen in the Sparse-GP and Dense-GP plots, which are almost identical. RR-GP (w/o BP) also has a similar performance from $t = 0$ s until $t = 4$ s because no dynamic infeasibilities or collisions with obstacles happen in this time range. However, the target first encounters the obstacle around $t = 5$ s, at which point the prediction of RR-GP (w/o BP) improves significantly compared to the GP algorithms since the algorithm is able to detect the infeasibility of the wrong pattern, and therefore adjust the trajectory prediction. The full RR-GP algorithm, denoted as RR-GP (w/ BP) in Figure 6, displays the best performance. Using the backward propagation feature, its prediction accuracy shows significant improvement over that of RR-GP (w/o BP), between $t = 1$ s and $t = 5$ s. This is accomplished by back-propagating knowledge of the future dynamic infeasibility to previous timesteps, which improves the accuracy of the earlier portion of the prediction. This reduces the RMS prediction errors by a factor of 2.4 over the GP-only based algorithms at $t = 8$ s.

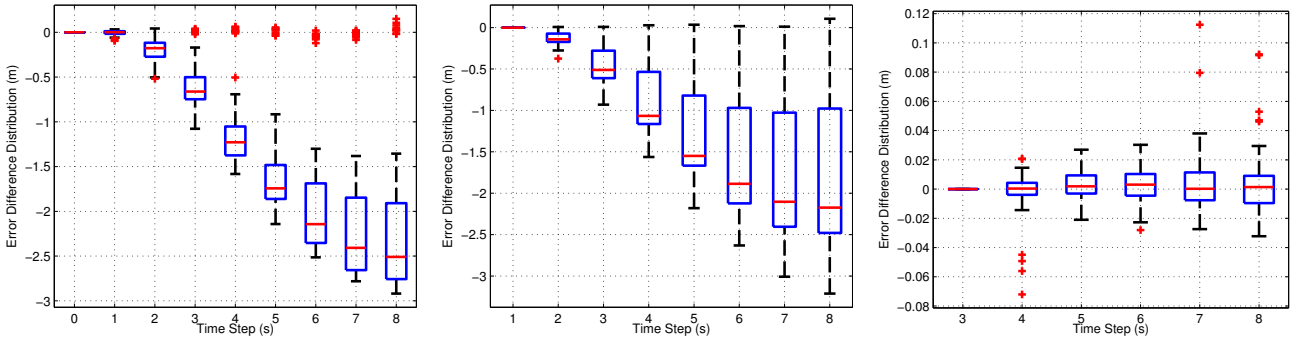
After one second has elapsed (Figure 6(b)), the vehicle has moved to a new position, and the likelihood values of each motion pattern have been updated (Figure 5). The probability of the correct motion pattern computed using Eq. (11) has slightly increased, leading to lower errors for all three algorithms. But as in Figure 6(a), a trend is seen for the four algorithms; the performance of the RR-GP (w/ BP) algorithm is consistently and significantly better than both Sparse-GP and Dense-GP, as well as RR-GP (w/o BP) in the time range prior to the collision detection. Note that Dense-GP performs slightly better than Sparse-GP between $t = 5$ s and $t = 8$ s, due to a more accurate GP model obtained through additional training data.

After three seconds have elapsed (Figure 6(c)), the probability of the correct motion pattern has approached 1.0 (Figure 5), yielding decreased prediction error across all algorithms. The vehicle has moved to a region where dynamic feasibility and collision checks are not significant, due to the negligible likelihood of the wrong motion pattern prediction. Eq. (17) then simplifies to $p(x_{t+K}, y_{t+K}|x_t, y_t, b_{j^*})$,



(a) t = 0 s (b) t = 1 s (c) t = 3 s

Fig. 6 Average position prediction errors (over 90 trajectories) for Sparse-GP (1Hz), Dense-GP (2Hz), and the two variations of the RR-GP algorithm at different times of the example



(a) t = 0 s (b) t = 1 s (c) t = 3 s

Fig. 7 *Box plots* of the difference of prediction errors (for the 90 test trajectories) between Sparse-GP (1Hz) and the full RR-GP algorithm at different times of the example

Table 1 Average probability of correct motion pattern (over 90 tests)

Alg. \ Time(s)	0	1	2	3
RR-GP	0.922	0.935	1.0	1.0
GP(1Hz)	0.5	0.633	0.995	1.0
GP(2Hz)	0.5	0.665	0.988	1.0

where j^* is the index of the "correct" motion pattern, and thus the prediction accuracy is only related to the position distribution of the correct motion pattern. This explains why Sparse-GP and both RR-GP variations show a very similar accuracy, since their position distributions are based on the same sparse data. On the other hand, Dense-GP, due to more dense data, is the best performer, as expected, with RMS errors 1.5 times smaller than the other algorithms at $t = 8$ s.

Figure 7 presents box-and-whisker plots of the average difference between the prediction errors of Sparse-GP (1Hz) and RR-GP (w/ BP), using the same data as Figure 6. The length of the whisker (dashed black vertical line) is $W = 1.5$, such that data are considered outliers if they are either smaller than $Q1 - W \times (Q3 - Q1)$ or larger than $Q3 + W \times (Q3 - Q1)$, where $Q1 = 25^{\text{th}}$ percentile and $Q3 = 75^{\text{th}}$ percentile.

At $t = 0$ s, Figure 7(a) shows the improvement of RR-GP (w/ BP) over Sparse-GP consistently increase, reaching a 2.5 m difference at $t = 8$ s. The outliers in this case are the test trajectories that are dynamically feasible at $t = 0$ s. Here, either of the two motion patterns may be followed, and RR-GP does not have an advantage over Sparse-GP, leading to no error difference. But in the majority of tests, as shown by the whisker lengths and box sizes, RR-GP (w/ BP) significantly reduces prediction error.

Figure 7(b) presents the error difference after 1 s; a similar trend can be seen. The median of the difference grows with the timesteps, but its magnitude slightly decreases compared to the previous timestep, reaching 2.1 m difference at $t = 8$ s. Here the whiskers have increased in length, eliminating many of the outliers from the prior timestep. Finally, after the target vehicle has moved for $t = 3$ s (Figure

Table 2 Average computation times per iteration (over 90 trajectories)

Algorithm	Computation Time (s)
Sparse-GP (1 Hz)	0.19
Dense-GP (2 Hz)	2.32
RR-GP	0.69

7(c)), the differences between RR-GP (w/ BP) and Sparse-GP are not statistically significant.

6.2.3 Computation Times

Table 2 summarizes the average computation times per iteration of the three algorithms over the 90 testing paths. Both RR-GP variations have identical computation times, so only one computation time is shown for RR-GP. Note that the implementation uses the GPML MATLAB toolbox (Rasmussen and Williams, 2005), and these tests were run on a 2.5 GHz quad-core computer.

As expected, Sparse-GP has the lowest computation time, while the RR-GP algorithm ranks second with times well below 1 s, which are suitable for real-time application as shown in Section 7.2. Out of the 0.69 s of RR-GP computation time, an average 0.5 s is spent in the RR-GP tree generation while the remaining 0.19 s is used to generate the GP samples. On the other hand, Dense-GP had an average computation time of 2.3 s, which is significantly worse than the other two approaches and violates the $dt = 1$ s measurement cycle. Such times render the GP mixture model useless for any typical real-time implementation, as expected due to GP's matrix inversions. If dt is changed to 2.3 s for the Dense-GP tests, the prediction accuracy decreases due to slower and fewer updates of the probabilities of the different motion patterns. Additional details on computation time can be found in Aoude (2011).

In summary, this simulated environment with a human-driven target vehicle shows that the RR-GP algorithm consistently performs better than Sparse-GP and Dense-GP in the long-term prediction of the target vehicle motion. It is important to highlight that RR-GP can predict trajectories at high output frequencies, significantly higher than both Sparse-GP (1Hz) and Dense-GP (2Hz). While the GP approach could be augmented with some form of interpolation technique, the solution approach developed in this work systematically guarantees collision avoidance and dynamic feasibility of the trajectory predictions at the higher rates. Another feature of the RR-GP algorithm is its low computation time (Table 2), which is small enough to be suitable for real-time implementation in collision warning systems or probabilistic path planners.

Finally, note that RR-GP was also validated in Aoude (2011) on intersection traffic data collected through the Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) project (Maile et al. (2008)). The obtained results demonstrated that RR-GP reduced prediction errors by almost a factor of 2 when compared to two standard GP-based algorithms, while maintaining computation times that are suitable for real-time implementation.

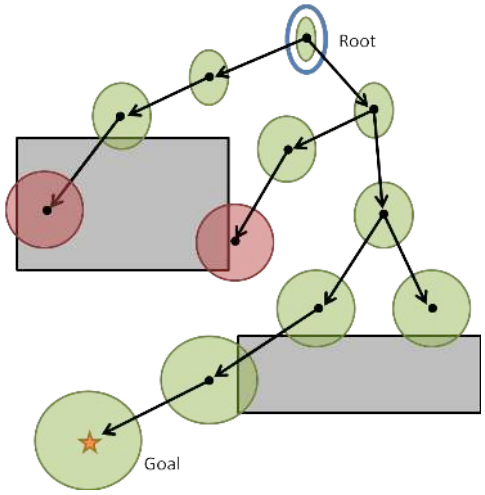


Fig. 8 Diagram of the chance constrained RRT algorithm. Given an initial state distribution at the tree root (*blue*) and constraints (*gray*), the algorithm grows a tree of state distributions to find a probabilistically feasible path to the goal (*yellow star*). The uncertainty in the state at each node is represented as an uncertainty ellipse. If the probability of collision is too high, the node is discarded (*red*); otherwise the node is kept (*green*) and may be used to grow future trajectories

7 CC-RRT Path Planning with RR-GP Predictions

As noted in Section 1, one of the main objectives of this work is to demonstrate that through appropriate choice of planner, an autonomous agent can utilize RR-GP predictions to identify and execute probabilistically feasible paths in real-time, in the presence of uncertain dynamic obstacles. This section introduces a path planning algorithm which extends the CC-RRT framework (Figure 8) of Luders et al. (2010b); Luders and How (2011) to guarantee probabilistic robustness with respect to dynamic obstacles with uncertain motion patterns. These guarantees are obtained through direct use of RR-GP trajectory predictions (Section 5). As these predictions are provided in the form of Gaussian uncertainty distributions at each timestep for each intent, they are well-suited for the CC-RRT framework. After the chance constraint formulation of Blackmore et al. (2006) is reviewed, the CC-RRT framework is presented, then extended to consider dynamic obstacles with uncertain motion patterns. Finally, an alternative particle-based approximation of CC-RRT for nonlinear dynamics and/or non-Gaussian uncertainty is also presented.

7.1 Extension of CC-RRT Chance Constraint Formulation

Recall the LTI system Eqs. (1)-(3); for now, assume that the uncertainty of each obstacle, Eqs. (6)-(7), can be represented by a single Gaussian distribution:

$$c_{jt} \sim \mathcal{N}(\hat{c}_{jt}, P_{c_{jt}}) \; \forall \, j \in \mathbb{Z}_{1,B}, \; \forall \, t. \tag{19}$$

In the context of RR-GP, this implies that the dynamic obstacle is following a single, known behavior, though its future state is uncertain.

Given a sequence of inputs $u_0, \ldots, u_{N-1}$ and the dynamics of Eq. (1), the distribution of the state x_t (represented as the random variable X_t) can be shown to be Gaussian (Blackmore et al. (2006)):

$$P(X_t | u_0, \ldots, u_{N-1}) \sim \mathcal{N}(\hat{x}_t, P_{x_t}) \;\; \forall \, t \in \mathbb{Z}_{0,N},$$

where N is some timestep horizon. The mean $\hat{x}_t$ and covariance P_{x_t} can be updated implicitly using the relations

$$\hat{x}_{t+1} = A\hat{x}_t + Bu_t \;\; \forall \, t \in \mathbb{Z}_{0,N-1}, \tag{20}$$

$$P_{x_{t+1}} = AP_{x_t}A^T + P_w \;\; \forall \, t \in \mathbb{Z}_{0,N-1}. \tag{21}$$

Note that by using Eqs. (20)-(21), CC-RRT can simulate state distributions within an RRT framework in much the same way that a nominal (*e.g.,* disturbance-free) trajectory would be simulated. Instead of propagating the nominal state, the distribution mean is propagated via Eq. (20); Eq. (21) can be used to compute the covariance offline.

As presented in Blackmore et al. (2006), to ensure that the probability of collision with any obstacle on a given timestep does not exceed $\Delta \equiv 1 - p_{\text{safe}}$, it is sufficient to show that the probability of collision with each of the B obstacles at that timestep does not exceed Δ/B. The jth obstacle is represented through the conjunction of linear inequalities

$$\bigwedge_{i=1}^{n_j} a_{ij}^T x_t < a_{ij}^T c_{ijt} \;\; \forall \, t \in \mathbb{Z}_{0,t_f}, \tag{22}$$

where n_j is the number of constraints defining the jth obstacle, and c_{ijt} is a point nominally (*i.e.,* $c_{jt} = \hat{c}_{jt}$) on the ith constraint at timestep t; note that a_{ij} is not dependent on t, since the obstacle shape and orientation are fixed.

It is shown in Blackmore et al. (2006) (for optimization-based frameworks) and Luders et al. (2010b) (for sampling-based frameworks) that to ensure the probability of constraint satisfaction exceeds p_{safe}, the system must satisfy a set of *deterministic* but *tightened* constraints for each obstacle, where the degree of tightening is a function of the degree of uncertainty, number of obstacles, and p_{safe}. These tightened constraints can be applied offline to assure probabilistic guarantees; however, this requires applying a fixed probability bound Δ/B across all obstacles, regardless of how likely they are to cause infeasibility, leading to conservative behavior (Blackmore et al., 2006).

Alternatively, the CC-RRT algorithm leverages a key property of the RRT algorithm – trajectory-wise constraint checking – by explicitly computing a bound on the probability of collision at each node, rather than simply satisfying tightened constraints for a fixed bound (Luders et al., 2010b). In doing so, CC-RRT can compute bounds on the risk of constraint violation online, based on the most recent RR-GP trajectory prediction data.

As shown in Luders et al. (2010b), the upper bound on the probability of collision with any obstacle at timestep t is

$$\Delta_t(\hat{x}_t, P_{x_t}) \equiv \sum_{j=1}^{B} \min_{i=1,\ldots,n_j} \Delta_{ijt}(\hat{x}_t, P_{x_t}), \tag{23}$$

$$\Delta_{ijt}(\hat{x}_t, P_{x_t}) \equiv \frac{1}{2}\left(1 - \text{erf}\left[\frac{a_{ij}^T\hat{x}_t - a_{ij}^T c_{ijt}}{\sqrt{2a_{ij}^T(P_{x_t} + P_{c_{jt}})a_{ij}}}\right]\right),$$

where $\text{erf}(\cdot)$ denotes the standard error function. Thus, for a node/timestep with state distribution $\mathcal{N}(\hat{x}_t, P_{x_t})$ to be probabilistically feasible, it is sufficient to check that $\Delta_t(\hat{x}_t, P_{x_t}) \leq 1 - p_{\text{safe}}$.

Now, suppose the jth obstacle is one of the dynamic obstacles modeled using RR-GP (Section 5) and that it may follow one of $k = 1, \ldots, M$ possible behaviors. At each timestep t, and for each behavior k, the RR-GP algorithm provides a likelihood δ_j^k and Gaussian distribution $\mathcal{N}(\hat{c}_{jt}^k, P_{c_{jt}}^k)$. Thus, the overall state distribution for this obstacle at timestep t is given by

$$c_{jt} \sim \sum_{k=1}^{M} \delta_j^k \mathcal{N}(\hat{c}_{jt}^k, P_{c_{jt}}^k). \tag{24}$$

At each timestep, the probability of collision with dynamic obstacle j can be written as a weighted sum of the probabilities of collision for the dynamic obstacle j under each behavior. With this modification, it can be shown that all existing probabilistic guarantees (Luders et al., 2010b) are maintained by treating each behavior's state distribution as a separate obstacle with the resulting risk scaled by δ_j^k:

$$\begin{aligned} P(\text{collision}) &\leq \sum_{j=1}^{B} P(\text{col. w/ obstacle } j) \\ &= \sum_{j=1}^{B}\sum_{k=1}^{M} \delta_j^k P(\text{col. with obstacle } j, \text{behavior } k) \\ &\leq \sum_{j=1}^{B}\sum_{k=1}^{M} \delta_j^k \min_{i=1,\ldots,n_j} P(a_{ij}^T X_t < a_{ij}^T C_{ijt}^k) \\ &= \sum_{j=1}^{B}\sum_{k=1}^{M} \delta_j^k \min_{i=1,\ldots,n_j} \Delta_{ijt}^k(\hat{x}_t, P_{x_t}), \end{aligned} \tag{25}$$

where C_{ijt}^k is a random variable representing the translation of the jth obstacle under the kth behavior, and Δ_{ijt}^k is used as in Eq. (23) for the kth behavior. By comparison with Eq. (23), the desired result is obtained.

7.2 CC-RRT with Integrated RR-GP

To perform robust planning this work uses chance constrained RRTs (CC-RRT), an extension of the traditional RRT algorithm that allows for probabilistic constraints. Whereas the traditional RRT algorithm (LaValle, 1998) grows a tree of states that are known to be feasible, the chance constrained RRT algorithm grows a tree of state distributions that are known to satisfy an upper bound on probability of collision (Figure 8), using the formulation developed in Section 7.1.

The fundamental operation in the standard RRT algorithm is the incremental growth of a tree of dynamically feasible trajectories, rooted at the system's current state x_t. To grow a tree of dynamically feasible trajectories, it is necessary for the RRT to have an accurate model of the (linear) vehicle dynamics, Eq. (1), for simulation. Since the CC-RRT algorithm grows a tree of Gaussian state distributions, in this case the model is assumed to be the propagation of the state conditional mean and covariance, Eqs. (20)-(21). These are rewritten here as

$$\hat{x}_{t+k+1|t} = A\hat{x}_{t+k|t} + Bu_{t+k|t}, \tag{26}$$

$$P_{t+k+1|t} = AP_{t+k|t}A^T + P_w, \tag{27}$$

where t is the current system timestep and $(\cdot)_{t+k|t}$ denotes the predicted value of the variable at timestep $t+k$.

The CC-RRT tree expansion step, used to incrementally grow the tree, is given in Algorithm 4. Each time the algorithm is called, a sample state is taken from the environment (line 2), and the nodes nearest to this sample, in terms of some heuristic(s), are identified as candidates for tree expansion (line 3). An attempt is made to form a connection from the nearest node to the sample by generating a probabilistically feasible trajectory between them (lines 7–12). This trajectory is incrementally simulated by selecting some feasible input (line 8), then applying Eqs. (26)-(27) to yield the state distribution at the next timestep. This input may be selected at the user's discretion, such as through random sampling or a closed-loop controller, but should guide the state distribution toward the sample. Probabilistic feasibility is then checked using RR-GP trajectory predictions with Eqs. (23) and (25); trajectory simulation continues until either the state is no longer probabilistically feasible, or the distribution mean has reached the sample (line 7). Even if the latter case does not occur, it is useful and efficient to keep probabilistically feasible portions of this trajectory for future expansion (Kuwata et al., 2009), via intermediate nodes (line 10). As a result, one or more probabilistically feasible nodes may be added to the tree (lines 13–16).

A number of heuristics are also utilized to facilitate tree growth, identify probabilistically feasible trajectories to the goal, and identify "better" paths (in terms of Eq. (9)) once at least one probabilistically feasible path has been found.

Algorithm 4 CC-RRT with RR-GP, Tree Expansion

1: Inputs: tree $\mathcal{T}$, current timestep t
2: Take a sample x_{samp} from the environment
3: Identify the Λ nearest nodes using heuristics
4: **for** $m \leq \Lambda$ nearest nodes, in sorted order **do**
5: $N_{\text{near}} \leftarrow$ current node
6: $(\hat{x}_{t+k|t}, P_{t+k|t}) \leftarrow$ final state distribution of N_{near}
7: **while** $\Delta_{t+k}(\hat{x}_{t+k|t}, P_{t+k|t}) \leq 1 - p_{\text{safe}}$ **and** $\hat{x}_{t+k|t}$ has not reached x_{samp} **do**
8: Select input $u_{t+k|t} \in \mathcal{U}$
9: Simulate $(\hat{x}_{t+k+1|t}, P_{t+k+1|t})$ using Eqs. (26)-(27)
10: Create intermediate nodes as appropriate
11: $k \leftarrow k+1$
12: **end while**
13: **for** each probabilistically feasible node N **do**
14: Update cost estimates for N
15: Add N to $\mathcal{T}$
16: **end for**
17: **end for**

Algorithm 5 CC-RRT with RR-GP, Execution Loop

1: Initialize tree $\mathcal{T}$ with node at $(\hat{x}_0, P_{x_0}), t = 0$
2: **while** $\hat{x}_t \notin \mathcal{X}_{\text{goal}}$ **do**
3: Retrieve most recent observations and RR-GP predictions
4: **while** time remaining for this timestep **do**
5: Expand the tree by adding nodes (Algorithm 4)
6: **end while**
7: Use cost estimates to identify best path $\{N_{\text{root}}, \ldots, N_{\text{target}}\}$
8: Repropagate the path state distributions using Eqs. (26)-(27)
9: **if** repropagated best path is probabilistically feasible **then**
10: Apply best path
11: **else**
12: Remove infeasible portion of best path and **goto** line 7
13: **end if**
14: $t \leftarrow t + \Delta\tau$
15: **end while**

Samples are identified (line 2) by probabilistically choosing between a variety of global and local sampling strategies, some of which may be used to efficiently generate complex maneuvers (Kuwata et al. (2009)). The nearest node selection (lines 3-4) strategically alternates between several distance metrics for sorting the nodes, including an exploration metric based on cost-to-go and a path optimization metric based on estimated total path length (Frazzoli et al. (2002)). Each time a sample is generated, $m \geq 1$ attempts are made to connect a node to this sample before being discarded. Additional heuristics include attempting direct connections to the goal anytime a new node is added, and maintaining bounds on the cost-to-go to enable a branch-and-bound pruning scheme (Frazzoli et al. (2002)).

For the real-time applications considered in this work, the CC-RRT tree should grow continuously during the execution cycle to account for changes in the situational awareness, such as updated RR-GP predictions. Algorithm 5 shows how the algorithm executes some portion of the tree while continuing to grow it. The planner updates the current path to be executed by the system every $\Delta\tau$ seconds, using

the most recent RR-GP predictions for any dynamic obstacles as they become available (line 3). During each cycle, for the duration of the timestep, the tree is repeatedly expanded using Algorithm 4 (lines 4-6). Following this growth, some cost metric is used to select the "best" path in the tree (line 7). Once a path is chosen, a "lazy check" (Kuwata et al., 2009) is performed in which the path is repropagated from the current state distribution using the same model dynamics, Eqs. (26)-(27), and tested for probabilistic feasibility (line 8). Due to the presence of dynamic obstacles, it is crucial to re-check probabilistic feasibility at every iteration, even if the agent itself is deterministic. If this path is still probabilistically feasible, it is chosen as the current path to execute (line 10); otherwise the infeasible portion of the path is removed and the process is repeated (line 12) until a probabilistically feasible path is found. In the event that no probabilistically feasible path can be found, mitigation strategies can be implemented to maximize safety, e.g. Wu and How (2012).

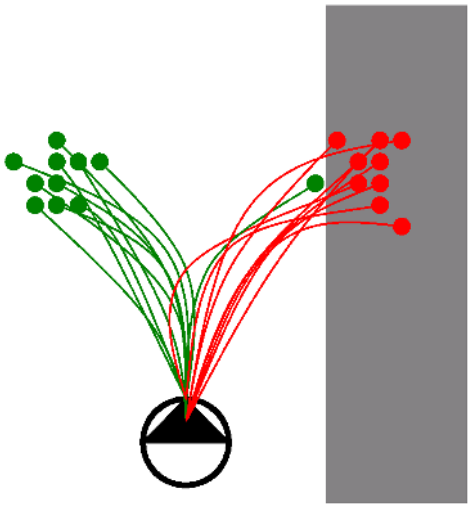


Fig. 9 Diagram of the PCC-RRT algorithm for a single, forward simulation step. Each *green circle*/path represents a simulated particle that terminates in a feasible state, while each *red circle*/path represents a simulated particle that terminates in an infeasible state

7.3 Particle CC-RRT

In the case of nonlinear dynamics and/or non-Gaussian noise, an alternative particle-based framework (Figure 9) can be used to statistically represent uncertainty at a resolution which can be dictated by the user (Luders and How (2011)). Though the generation of particles increases the per-node complexity, the algorithm maintains the benefits of sampling-based approaches for rapid replanning. The particle CC-RRT (PCC-RRT) framework of Luders and How (2011) is generalizable both in the types of probabilistic feasibility that are assessed (timestep-wise and path-wise) and in the types of uncertainty that are modeled using particles. This framework can be extended to consider hybrid combinations of particle-based and distribution-based uncertainty; for example, an agent's dynamics/process noise can be represented via particles, while interactions with dynamic obstacles are modeled using traditional Gaussian distributions. However, this may limit the ability to assess path-wise infeasibility.

Algorithm 6 presents the tree expansion step for the particle-based extension of CC-RRT, PCC-RRT. A set of $P_{\max}$ particles are maintained at each node, each with a position x and a weight w ($\sum w = 1$ across all particles at each timestep). There are two parameters the user can specify to indicate the degree of probabilistic constraint violation allowed: the average likelihood of feasibility at each node cannot exceed $p_{\text{safe}}^{\text{node}}$, while the average likelihood of feasibility over an entire path cannot exceed $p_{\text{safe}}^{\text{path}}$. The latter bound is a key advantage of this particle-based approach, as it is quite difficult to approximate analytically in real-time without introducing significant conservatism. The former likelihood is computed by summing the weights of all feasible nodes, $\sum_p w_{t+k|t}^{(p)}$ (line 7), while the latter is computed iteratively over a path by multiplying the prior node's path probability by the weight of existing nodes that are still feasible (line 15).

One of several resampling schemes may be used for identifying new particles as older ones become infeasible (line 8). In the uniform resampling scheme, all particles are assigned an identical weight at line 12, $w_{t+k+1|t}^{(p)} = 1/P_{\max}$. When this is the case, every particle has an equal likelihood of being resampled. In the probabilistic resampling scheme, each particle is assigned a weight based on the likelihood of that particle actually existing; this is a function of the likelihood of each portion of the uncertainty that is sampled. This can be computed iteratively by node, as

$$w_{t+k+1|t}^{(p)} \propto w_{t+k|t}^{(p)} \cdot P(X_{t+k+1|t} = x_{t+k+1|t}). \tag{28}$$

This requires additional computation, especially as the complexity of the uncertainty environment increases; however, it can provide a better overall approximation of the state distribution at each timestep, for the same number of particles.

8 Results

This section presents simulation results which demonstrate the effectiveness of the RR-GP algorithm in predicting the future behavior of an unknown, dynamic vehicle, allowing the CC-RRT planner to design paths which can safely avoid it. Examples are provided for three scenarios of varying complexity, in terms of dynamics, environment, and possible behaviors. In the first two examples, planning is performed on a vehicle with linear dynamics, such that the

Algorithm 6 PCC-RRT, Tree Expansion

1: Inputs: tree $\mathcal{T}$, current timestep t
2: Take a sample x_{samp} from the environment
3: Identify the M nearest nodes using heuristics
4: **for** $m \leq M$ nearest nodes, in the sorted order **do**
5: $N_{\text{near}} \leftarrow$ current node
6: $\{x^{(p)}_{t+k|t}, w^{(p)}_{t+k|t}\} \leftarrow$ set of feasible particles at N_{near}, with weights
7: **while** $\sum_p w^{(p)}_{t+k|t} \geq p^{\text{node}}_{\text{safe}}$ **and** $P^{\text{path}}_k \geq p^{\text{path}}_{\text{safe}}$ **and** $\hat{x}_{t+k|t}$ has not reached x_{samp} **do**
8: Resample particles up to count of $P_{\max}$, using weights $w^{(p)}_{t+k|t}$
9: Select input $u_{t+k|t} \in \mathcal{U}$
10: **for** each particle p **do**
11: Simulate $x^{(p)}_{t+k+1|t}$ using Eq. (1) and sampled disturbance w_{t+k}
12: Assign weight $w^{(p)}_{t+k+1|t}$ to particle
13: **end for**
14: Remove infeasible particles
15: $P^{\text{path}}_{k+1} \leftarrow P^{\text{path}}_k \cdot \sum_p w^{(p)}_{t+k|t}$
16: $k \leftarrow k+1$
17: **end while**
18: **for** each probabilistically feasible node N **do**
19: Update cost estimates for N
20: Add N to $\mathcal{T}$
21: Try connecting N to $\mathcal{X}_{\text{goal}}$ (lines 5-13)
22: **if** connection to $\mathcal{X}_{\text{goal}}$ probabilistically feasible **then**
23: Update upper-bound cost-to-go of N and ancestors
24: **end if**
25: **end for**
26: **end for**

extended theoretical framework of Section 7 is valid. The final example considers a vehicle with car-like dynamics, using the PCC-RRT algorithm of Section 7.3 to approximate path-wise feasibility. Though a single dynamic, uncertain obstacle (in these examples, a "target vehicle") is present in each case, the approach can be extended to multiple dynamic, uncertain obstacles without further modification, and in fact will scale well under such conditions if parallelization is utilized.

8.1 Infrastructure

Algorithms 2-5 have been implemented using a multi-threaded, real-time Java application, modular with respect to all aspects of the problem definition. All simulations were run on a 2.53GHz quad-core laptop with 3.48GB of RAM.

The software implementation consists of three primary modules, each in a separate thread. The *Vehicle* thread manages the overall simulation, including all simulation objects; it is run in real-time at 10-50 Hz, and operates continuously until a collision has occurred or the vehicle has safely reached the goal. The *RRT* thread implements Algorithms 4-5, growing the CC-RRT tree while periodically sending the current best path in the tree to the Vehicle thread. Finally, the *RRGP* thread maintains predictions on the likelihoods and future state distributions of possible behaviors for each dynamic obstacle by incorporating Algorithm 2, embedded as a MATLAB program.

To ensure the RR-GP algorithm is tested on realistic driving behavior, the target vehicle's motion is chosen from among a set of simulated trajectories, pre-generated for each behavior by having a human operator manually drive the vehicle in simulation. As in Section 6.1, the target vehicle dynamics are based on the iRobot Create platform; the simulated vehicle was driven via a wireless steering apparatus, tuned to emulate traditional, nonlinear control of an automobile. During each trial, one of these paths is randomly selected as the trajectory for the target vehicle.

8.2 Intersection Scenario

Consider a ground vehicle operating in a constrained, two-dimensional environment (Figure 10). This environment is a representative road network designed to emulate a real-world driving environment within the RAVEN testbed (How et al., 2008). The road network is 11.2×5.5 m^2 in size, and is capable of accommodating multiple intersection types.

In this scenario, the objective of the host vehicle is to go straight through the intersection at bottom-center of Figure 10(a), reaching a goal location on the opposite side. However, to get there, the host vehicle must successfully avoid an errant (rule-violating) driver which is traveling through the intersection in the perpendicular direction, and is likely to cross the intersection at the same time as the host vehicle. There are three possible behaviors for the target vehicle as it enters the intersection: (a) left turn, (b) right turn, and (c) straight. The host vehicle is assumed to have a radius of 0.2m, while the target vehicle has a radius of 0.14m; both start at zero velocity.

The host vehicle is modeled as a double integrator,

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v^x_{t+1} \\ v^y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v^x_t \\ v^y_t \end{bmatrix} + \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u^x_t + w^x_t \\ u^y_t + w^y_t \end{bmatrix},$$

where $dt = 0.1$s, subject to avoidance constraints $\mathcal{X}$ (including velocity bounds) and input constraints

$$\mathcal{U} = \{(u^x, u^y) \mid |u^x| \leq 4, |u^y| \leq 4\}.$$

To emphasize the impact of the dynamic obstacle's uncertainty, the host vehicle's own dynamics are assumed deterministic: $w^x_t \equiv w^y_t \equiv 0$. Trajectories are simulated and executed in closed-loop via the controller

$$u^x_t = -1.5(x_t - r^x_t) - 3(v^x_t - r^{v_x}_t),$$
$$u^y_t = -1.5(y_t - r^y_t) - 3(v^y_t - r^{v_y}_t),$$

where (r_t^x, r_t^y) is the reference position and $(r_t^{v_x}, r_t^{v_y})$ is the reference velocity; the reference r_t is moved continuously between waypoints at a fixed speed of 0.35 m/s. The speed of the target vehicle is capped at 0.4 m/s.

In Algorithms 4–5, the tree capacity is limited to 1000 nodes, with a replan time interval (line 14, Algorithm 5) of 0.5s. In Algorithms 1-2, $T_h = 8$s and $\Delta t = 1$s, though the vehicle dynamics are simulated at 10 Hz. The limits of successful and infeasible connections per Δt (lines 9 and 11, Algorithm 1) are 20 and 100, respectively. The RR-GP algorithm is called once every 1.0 seconds, each time giving Algorithm 2 a total of 0.3 seconds to grow its trees. If the time limit is reached, the algorithm is terminated without reaching the time horizon T_h.

A total of 400 trials were performed, consisting of 50 trials each for eight different algorithms:

- Naive RRT: nominal RRT (no chance constraints) in which target vehicle is ignored; this sets a baseline for the minimum expected likelihood of safety
- Nominal RRT: nominal RRT in which target vehicle is treated as a static obstacle at its most recent location
- Velocity-Avoidance RRT: nominal RRT in which the future position of the target vehicle is predicted by propagating its current position based on its current velocity and heading
- CC-RRT (5 cases): Algorithms 4-5 with $p_{\text{safe}} = 0.5$, 0.8, 0.9, 0.99, or 0.999. Note that since p_{safe} is a bound on feasibility at each timestep, rather than over an entire path, it does not act as a bound on the percentage of paths which safely reach the goal.

Each trial differs only in the path followed by the target vehicle and the random sampling used in the RR-GP and CC-RRT algorithms; the sequence of target vehicle paths is consistent across all sets of 50 trials. Four quantities were measured and averaged across these trials: the percentage of trials in which the vehicle safely reaches the goal; the average duration of such paths; the average time to generate an RRT/CC-RRT tree node; and the average time per execution of Algorithm 2.

Table 3 presents the averaged results over the 50 trials for each case. Note that in all five cases using CC-RRT, the host vehicle safely navigates the intersection with a much higher likelihood than any of the cases not using chance constraints. Furthermore, the CC-RRT results demonstrate the clear trade-off between overall path safety (in terms of percentage of trials which reach the goal) and average path duration when using CC-RRT. As p_{safe} is increased from 0.5 to 0.999, the percentage of safe trajectories generally increases (the exception of $p_{\text{safe}} = 0.5$ is not statistically significant), culminating with the host vehicle using CC-RRT with $p_{\text{safe}} = 0.999$ reaching the goal safely in all fifty trials. On the other hand, as p_{safe} is increased and the planner becomes more conservative, the average time duration of the safe trajectories increases.

Figure 10 sheds some light on how different values of p_{safe} affect the types of paths chosen by the planner. In this particular trial, the target vehicle ultimately makes a left turn through the intersection, and would collide with the host vehicle if it did not deviate from an initial straight-path trajectory. The RR-GP algorithm is initially undecided whether the target vehicle is going straight or turning left (as indicated by the shading on the predicted trajectories in Figures 10(a) and 10(b)); by $t = 6$ seconds RR-GP is very confident that the vehicle is turning left (Figures 10(c) and 10(d)). When $p_{\text{safe}} = 0.8$, the planner selects a path with the minimum perturbation needed to avoid the target vehicle's most likely trajectories (Figure 10(a)). As the target vehicle closes in on the intersection (Figure 10(c)), the host vehicle continues to hedge that it can cross the intersection safely and avoid the target vehicle's approach in either direction, and thus does not modify its plan. In contrast, when $p_{\text{safe}} = 0.999$, the planner selects a larger initial perturbation to maintain the host vehicle's distance from the target vehicle (Figure 10(b)). After several RR-GP updates, the host vehicle demonstrates a much more risk-averse behavior, by loitering outside the intersection (Figure 10(d)) for several seconds before making its approach. Ultimately, the host vehicle using $p_{\text{safe}} = 0.8$ reaches the goal (Figure 10(e)) before the host vehicle using $p_{\text{safe}} = 0.999$ (Figure 10(f)). In realistic driving scenarios, the most desirable behavior is likely somewhere between these two extremes.

Table 3 shows that with Naive RRT, by ignoring the target vehicle, the time-optimal path is almost always achieved, but a collision takes place in a majority of trials, with collisions occurring in most instances of the target vehicle going straight or left. In some instances, the Nominal RRT algorithm maintains safety by selecting an alternative trajectory when the target vehicle's current position renders the host vehicle's current trajectory infeasible; however, the overall likelihood of safety is still low. In many cases, the target vehicle collides with the host vehicle from the side, such that a replan is not possible. Of the nominal RRT algorithms, the velocity-avoidance RRT algorithm performs the most competitively with CC-RRT, with 74% of trials yielding a safe trajectory. Since the target vehicle always starts by driving straight toward the intersection (right to left in the figures), the host vehicle responds in nearly all trials by immediately perturbing its own path, based on the assumption that the target vehicle will go straight through the intersection. This contributes to the larger average path duration obtained by velocity-avoidance RRT compared to the other nominal algorithms in Table 3. However, in many instances, velocity-avoidance RRT is still unable to respond to rapid changes in the target vehicle's heading, such as if the target vehicle makes a rapid, left turn inside the intersection.

Table 3 Simulation results, intersection scenario

Algorithm	p_{safe}	% to Goal[a]	Path Duration, s[b]	Time per Node, ms[c]	Time per RR-GP, s[d]
Naive RRT	–	38%	10.01 (0.11%)	0.611	–
Nominal RRT	–	46%	10.90 (8.96%)	0.662	–
Velocity-Avoidance RRT	–	74%	11.14 (11.4%)	0.948	–
CC-RRT	0.5	92%	11.52 (15.2%)	1.610	0.598
CC-RRT	0.8	88%	11.65 (16.5%)	1.620	0.598
CC-RRT	0.9	92%	11.69 (16.9%)	1.620	0.590
CC-RRT	0.99	96%	12.51 (25.1%)	1.537	0.592
CC-RRT	0.999	100%	12.84 (28.4%)	1.492	0.587

[a] Percentage of trials where system executed a path to goal without colliding with any obstacles. (recall that p_{safe} refers to feasibility for a single *timestep*, whereas this entry corresponds to feasibility across the entire *path*)
[b] Percentage is average increase in path duration relative to minimal-time (obstacle-free) path, 10.0s. Only paths which reach goal are included
[c] Cumulative time in Algorithm 4 divided by number of nodes generated
[d] Time spent in Algorithm 2

Finally, note that the average time to either generate an RRT node or call RR-GP is largely independent of p_{safe} for CC-RRT. There is a modest increase in average time per node when moving from naive or nominal RRT to velocity-avoidance RRT (scales by a factor of 1.5) or CC-RRT (scales by a factor of 2.5), though previous work has demonstrated that these factors scale well with environment complexity (Luders et al., 2010b).

8.3 Complex Scenario

In this scenario, the problem complexity is increased, with more obstacles and more possible behaviors for the target vehicle (Figure 11). This environment is the same size as the previous one, with rearranged obstacles; as the target vehicle moves from one side of the environment to the other, it may display as many as six possible behaviors, corresponding to whether each of the four obstacles is passed by the target vehicle on its left or right. Furthermore, by design the target vehicle has a higher maximum speed than the host vehicle, meaning that the host vehicle is at risk of being overtaken from behind if its path is not planned carefully.

Figure 11(b) shows the trajectories generated by RR-GP for the six behaviors in this scenario. This demonstrates the RR-GP algorithm's ability to model complex behaviors for long time horizons, building off available training data in the form of synthetic training trajectories (Figure 11(a)).

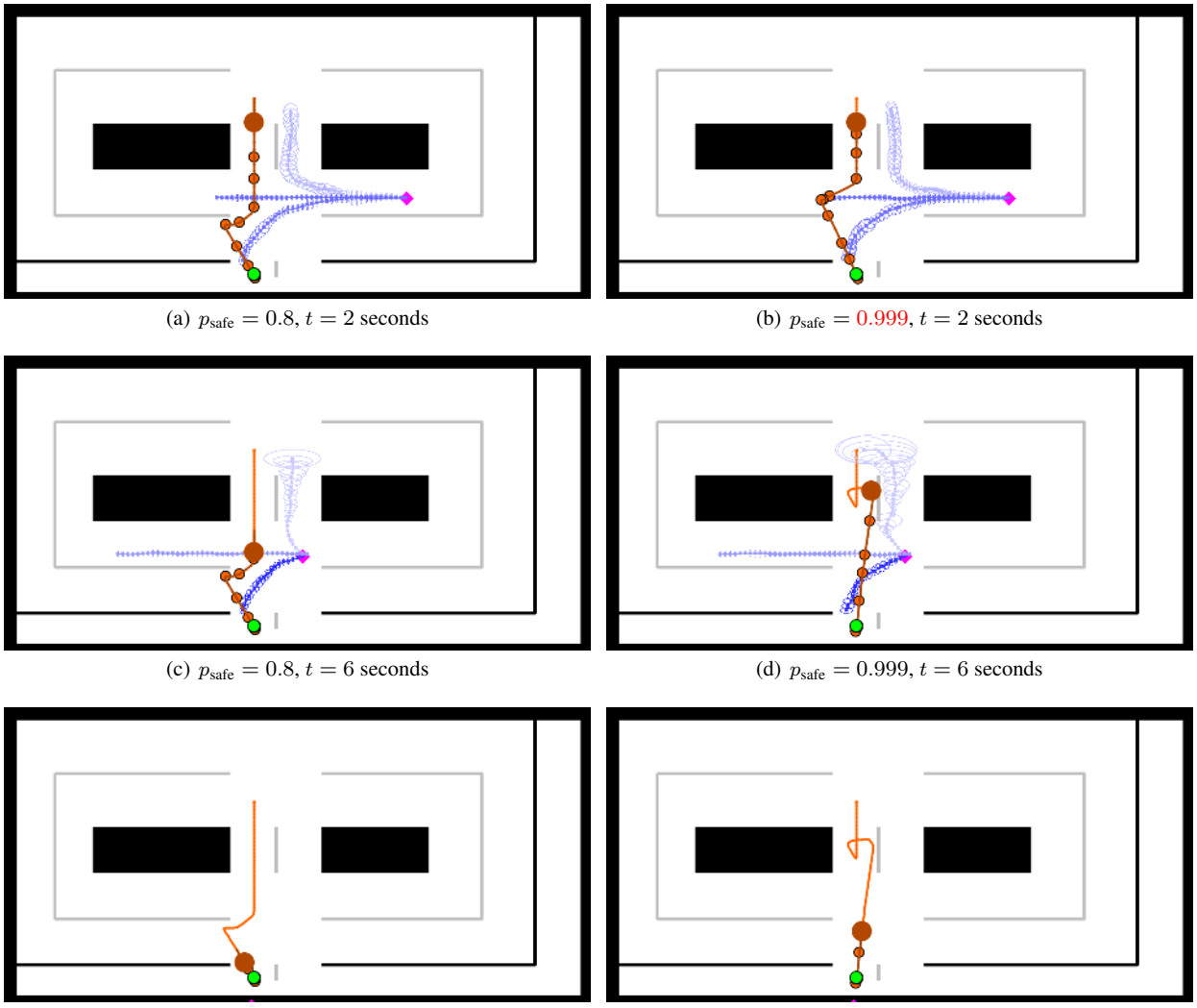
The same double integrator dynamics and controller are used as in Section 8.2, but in this case the reference movement speed is increased from 0.35 m/s to 0.6 m/s. Due to the increased number of behaviors and more complex environment, the Gaussian process formulation is more challenging; as a result, several of the parameters in Algorithms 1-2 have been tuned to improve performance. Here, the limits of successful and infeasible connections per Δt (lines 9 and 11, Algorithm 1) are 10 and 100, respectively. The RR-GP algorithm is called once every 1.25 seconds, each time giving Algorithm 2 up to a full second to grow trees for each behavior, with the time horizon T_h increased from 8 seconds to 12 seconds. Both vehicles have a radius of 0.2 m and start at zero velocity.

The objective of this scenario is to demonstrate the ability of the CC-RRT algorithm, using RR-GP dynamic obstacle predictions, to exhibit safe driving behavior for long-duration missions. The same eight algorithms used in Section 8.2 are again used here; however, rather than performing 50 trials, each algorithm is used to guide the host vehicle through a continuous sequence of 50 waypoints. The host vehicle starts on the left side of the room, and each subsequent waypoint is among a set of four, located near each of the room's corners. The host vehicle is given the next waypoint as soon as the current waypoint is reached; consecutive waypoints are required to be on opposite sides of the room, with respect to the room's long axis. While the host vehicle completes this sequence, the target vehicle moves continuously back and forth between the left and right ends of the room, each time selecting one of the six possible behaviors and one of the four pre-generated trajectories for that behavior (Figure 11).[3] Note that the sequences of waypoints and target vehicle behaviors are both consistent across all algorithms.

If the host vehicle collides with either the target vehicle or an element of the environment, the mission continues; however, the target vehicle is penalized for this collision by being re-set back to its last reached waypoint. This allows the trial for each algorithm to be performed as a single, continuous simulation, while also providing a figure of merit which factors in both path duration/length and collision risk.

Four quantities were measured and averaged for each algorithm: the total time required to reach all 50 waypoints, including collision time penalties; the number of collisions which take place; the average time to generate an RRT/CC-

[3] When the target vehicle moves from left to right, the trajectories shown in Figure 11 are reflected across the room's short axis.



(a) $p_{\text{safe}} = 0.8$, $t = 2$ seconds

(b) $p_{\text{safe}} = 0.999$, $t = 2$ seconds

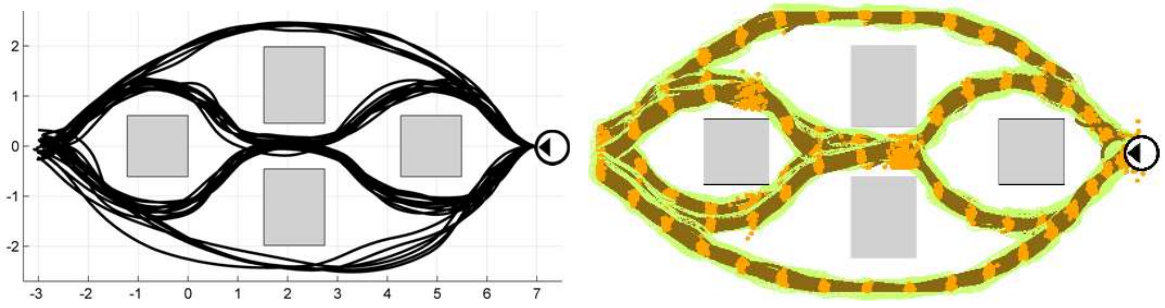
(c) $p_{\text{safe}} = 0.8$, $t = 6$ seconds

(d) $p_{\text{safe}} = 0.999$, $t = 6$ seconds

(e) $p_{\text{safe}} = 0.8$, $t = 11$ seconds

(f) $p_{\text{safe}} = 0.999$, $t = 11$ seconds

Fig. 10 Representative screenshots of the RR-GP and CC-RRT algorithms during trial #25 of the intersection scenario, for two different values of p_{safe}. The host vehicle's path history and current path are in *orange*. The objective of the host vehicle (*large orange circle*) is to reach the goal position (*green circle*) while avoiding all static obstacles (*black*) and the dynamic target vehicle (*magenta diamond*). The *blue paths* indicate the paths predicted by the RR-GP algorithm for each possible behavior, including $2-\sigma$ uncertainty ellipses; more likely paths are indicated with a *brighter shade of blue*. All objects are shown at true size; the *gray lines* are lane markings, which do not serve as constraints



(a) Training trajectories

(b) RR-GP outputs based on target vehicle's starting position

Fig. 11 Environment used in the complex scenario, including possible behaviors for the target vehicle (*at right*)

Table 4 Simulation results, complex scenario

Algorithm	p_{safe}	# Collisions[a]	Mission Duration, s[b]	Time per Node, ms	Time per RR-GP, s
Naive RRT	–	13	737.1	0.731	–
Nominal RRT	–	20	819.5	0.812	–
Velocity-Avoidance RRT	–	4	766.3	1.421	–
CC-RRT	0.5	6	821.2	5.112	0.665
CC-RRT	0.8	8	818.9	5.596	0.654
CC-RRT	0.9	2	811.7	4.343	0.635
CC-RRT	0.99	4	820.5	5.142	0.643
CC-RRT	0.999	6	834.1	4.693	0.639

[a] Number of collisions which took place over a single trial. Collisions which take place within 0.5 seconds of each other are not counted as separate collisions.
[b] Total time required for host vehicle to reach 50 waypoints; note that the vehicle is reset to its last reached waypoint each time a collision takes place.

RRT tree node; and the average time per execution of Algorithm 2.

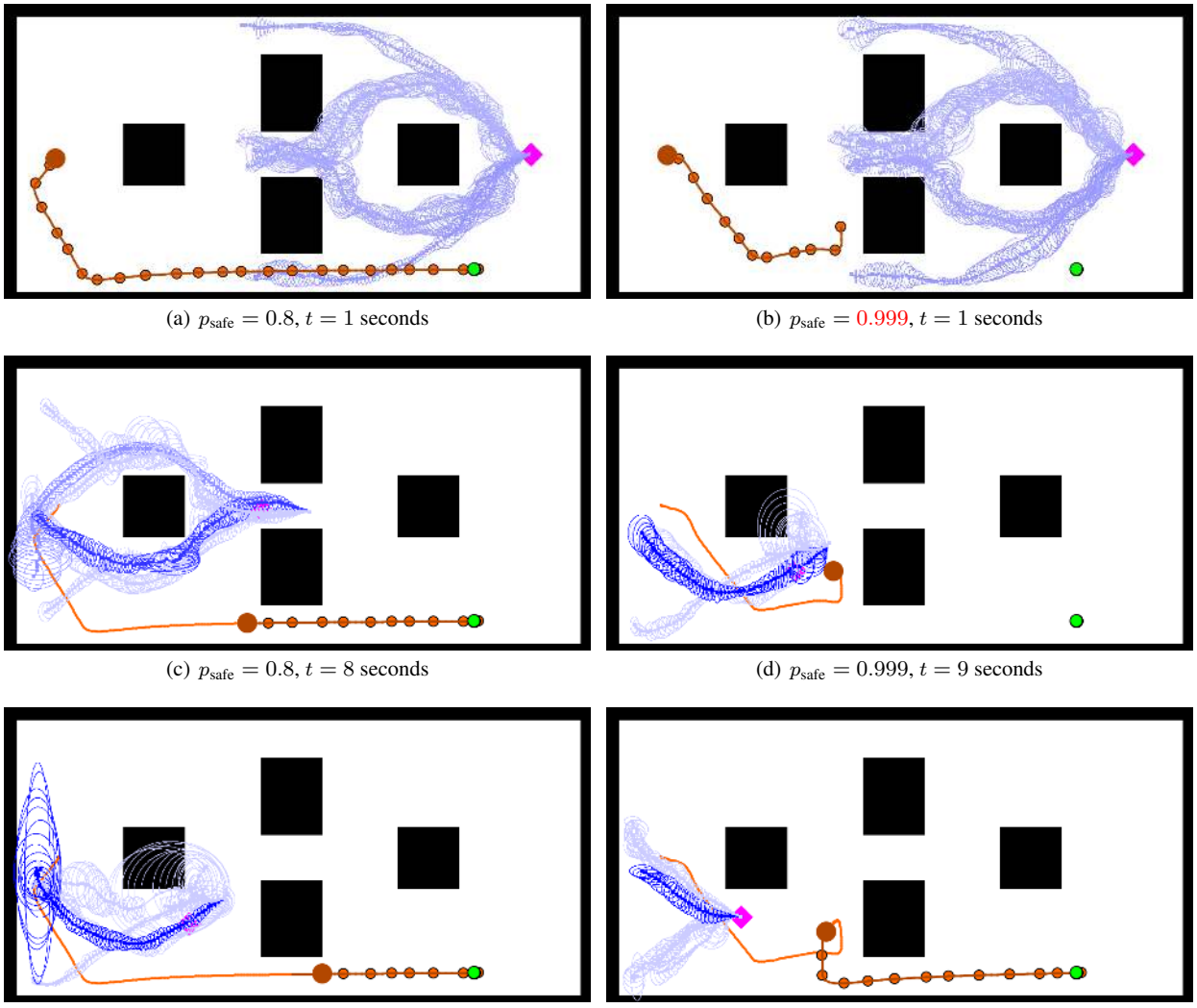
Table 4 presents the results for each algorithm. The most desirable behavior is achieved using CC-RRT with $p_{\text{safe}} = 0.9$, with only 2 collisions taking place over a sequence of 50 waypoints. Two aspects of this scenario tend to increase the likelihood of collision across all algorithms, such that 100% safety becomes unreasonable for this scenario. First, when the target vehicle changes direction, the RR-GP prediction environment changes rapidly, and the host vehicle may not be able to react quickly enough if nearby. Second, the host vehicle may find itself in a corridor being pursued by the target vehicle; since the target vehicle has a larger maximum speed, a collision may become inevitable. Regardless, CC-RRT outperforms both Naive RRT and Nominal RRT for all values of p_{safe} tested, as well as Velocity-Avoidance RRT when $p_{\text{safe}} = 0.9$.

Observing the nominal RRT results in Table 4, it is clear that neither Naive RRT nor Nominal RRT can provide a sufficient level of safety for the host vehicle, with a double-digit number of collisions occurring in each case. On the other hand, Velocity-Avoidance RRT is quite competitive with CC-RRT. Though it does not achieve the minimum number of collisions obtained by CC-RRT for $p_{\text{safe}} = 0.9$, Velocity-Avoidance RRT still only has 4 collisions, as well as a mission duration significantly shorter than any of the CC-RRT trials. Since the trajectories executed by the target vehicle are relatively straight, with few sharp turns (especially compared to Section 8.2), the forward propagation done in this case actually tends to be a good prediction, allowing the host vehicle to make a rapid response when needed.

A notable trend in these results is that as p_{safe} increases, both the number of collisions and mission duration tend to decrease, reach minimum values at $p_{\text{safe}} = 0.9$, then actually increase beyond that value. As p_{safe} is increased, the minimum required probability of feasibility at each timestep is increased, effectively reducing the set of probabilistically feasible paths that may be identified and selected by the CC-RRT algorithm. Thus, assuming the algorithm continues to identify probabilistically feasible paths satisfying the p_{safe} requirement, the performance (in terms of percentage of safe trials) is expected to increase, with a trade-off of additional path conservatism. However, if this assumption is broken, the agent may be unable to find a path at all, and will come to a stop. This "frozen robot" behavior (Trautman and Krause (2010)) actually puts the agent at added risk in a dynamic environment. This is particularly likely to occur in heavily constrained environments for large values of p_{safe}, as is the case here. Many of the collisions in this scenario are caused by the agent being unable to find any safe paths at all to execute. In such cases, it is important for the operator to tune p_{safe} to best meet the needs of the problem being considered.

Figure 12 demonstrates how CC-RRT can exhibit complex, robust avoidance behavior for large values of p_{safe} in order to remain risk-averse. In each trial, the agent's first task (shown in the figure) is to move from the left side of the room to the waypoint at bottom-right; the shortest path is to move along the bottom of the room. The target vehicle's trajectory takes it to the left of the first and last obstacles, and down the central corridor. When the first RR-GP update is performed, there is little data available to infer which way the target vehicle is going, and thus all six behaviors are equally likely. For $p_{\text{safe}} = 0.8$, the planner selects a complete trajectory which reaches the goal (Figure 12(a)). Even though this would lead to a head-on collision for one of the behaviors, the likelihood of that behavior being active is roughly 1 in 6, an acceptable risk for $p_{\text{safe}} = 0.8 \ (< 5/6)$. On the other hand, when $p_{\text{safe}} = 0.999$, the planner is not willing to select a trajectory with crosses the target vehicle's path for *any* possible behavior. Instead, it selects a partial path behind one of the central obstacles, the location which brings it closest to the goal without being in any of the target vehicle's possible paths (Figure 12(b)).

As the mission progresses, the target vehicle's path is revealed to go through the central corridor. For $p_{\text{safe}} = 0.8$,



(a) $p_{\text{safe}} = 0.8$, $t = 1$ seconds

(b) $p_{\text{safe}} = 0.999$, $t = 1$ seconds

(c) $p_{\text{safe}} = 0.8$, $t = 8$ seconds

(d) $p_{\text{safe}} = 0.999$, $t = 9$ seconds

(e) $p_{\text{safe}} = 0.8$, $t = 9$ seconds

(f) $p_{\text{safe}} = 0.999$, $t = 10$ seconds

Fig. 12 Representative screenshots of the RR-GP and CC-RRT algorithms as the host vehicle approaches the first waypoint in the complex scenario, for two different values of p_{safe}

this was not the behavior that risked a head-on collision, so it continues on its initial trajectory (Figures 12(c) and 12(e)), reaching the goal state quickly. On the other hand, when $p_{\text{safe}} = 0.999$, the agent holds its position behind the obstacle until the target vehicle has passed through the central corridor (Figure 12(d)); once the target vehicle has passed by, the agent identifies a new trajectory which reaches the goal (Figure 12(f)), though it arrives at the goal significantly later than if a lower value of p_{safe} were used.

The average runtime needed to generate a tree node using CC-RRT is larger than the average runtime for nominal RRT, by a factor which is larger than the one observed in Section 8.2. Nonetheless, the runtime per node averages only 5ms, meaning many hundreds of nodes can be generated every second. Coupled with the fact that the RR-GP update averages a fraction of a second (about 0.65s), the CC-RRT algorithm with RR-GP is still amenable to real-time implementation for this more complex example. To better demonstrate the operation of this example, a video showing a representative simulation is available at `http://acl.mit.edu/rrgp.mp4`.

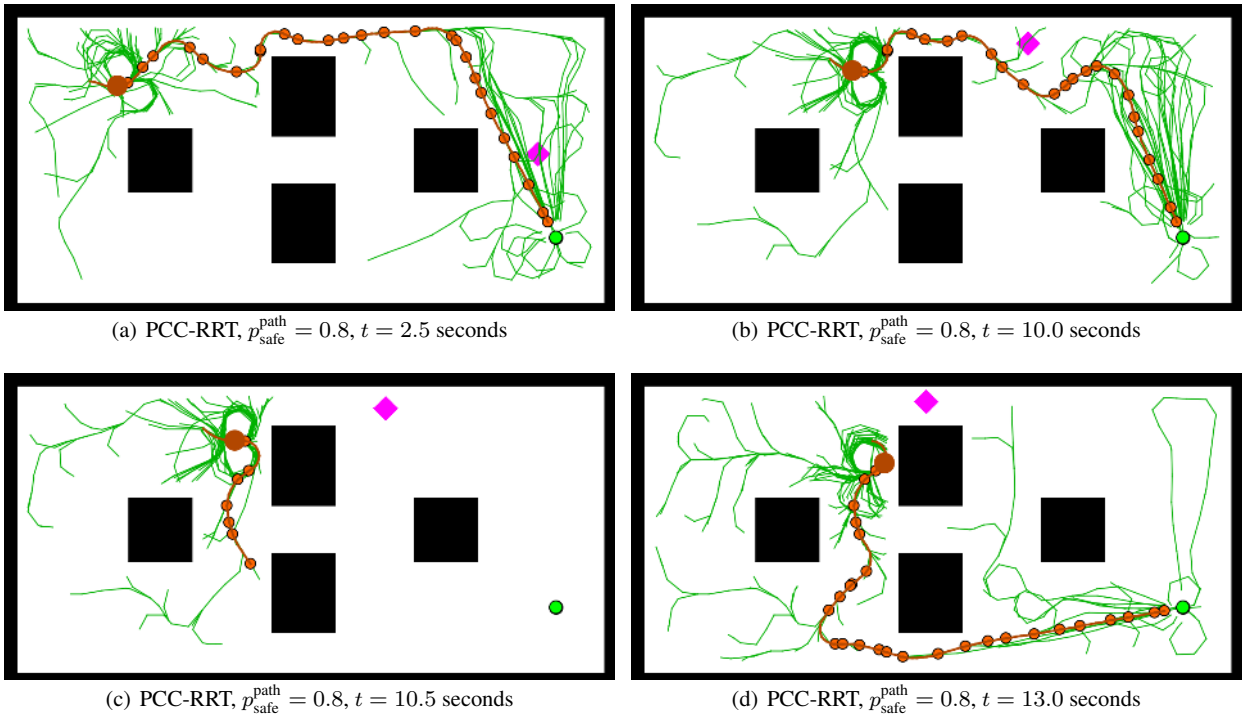
8.4 Nonlinear Dynamics Example

In this final example, the same environment and behaviors are used as in the previous example (Figure 11); however, the host vehicle is now modeled as nonlinear car dynamics

$$
\begin{aligned}
x_{t+1} &= x_t + (dt) v \cos\theta_t + w_t^x, \\
y_{t+1} &= y_t + (dt) v \sin\theta_t + w_t^y, \\
\theta_{t+1} &= \theta_t + (dt) \frac{v}{L_w} \tan\delta_t + w_t^\theta,
\end{aligned}
$$

Table 5 Simulation results, nonlinear dynamics scenario

Algorithm	$p_{\text{safe}}^{\text{path}}$	% to Goal	Path Duration, s	Time per Node, ms
Naive RRT	–	54%	26.91	0.529
Nominal RRT	–	46%	27.46	0.536
Velocity-Avoidance RRT	–	67%	29.29	0.736
PCC-RRT	0.8	83%	32.12	9.422



(a) PCC-RRT, $p_{\text{safe}}^{\text{path}} = 0.8$, $t = 2.5$ seconds

(b) PCC-RRT, $p_{\text{safe}}^{\text{path}} = 0.8$, $t = 10.0$ seconds

(c) PCC-RRT, $p_{\text{safe}}^{\text{path}} = 0.8$, $t = 10.5$ seconds

(d) PCC-RRT, $p_{\text{safe}}^{\text{path}} = 0.8$, $t = 13.0$ seconds

Fig. 13 Representative screenshots of the PCC-RRT algorithm for $p_{\text{safe}}^{\text{path}} = 0.8$ for the nonlinear dynamics scenario. Agent's PCC-RRT tree is shown in *green*; the executed path is shown in *orange* (RR-GP output is suppressed for clarity)

where $v = 0.4$ m/s, $dt = 0.1$s, (x, y) is the vehicle position, θ is the heading, $L_w = 0.2$ m, and $\delta_t \in [-\pi/4, +\pi/4]$ is the steering angle input. As in Section 5, a pure-pursuit low-level controller is applied for steering control. In each trial, the agent's objective is to cross the environment from top-left to bottom-right, while avoiding the dynamic obstacle moving from right to left (as in the previous example).

However, the agent is subject to a non-Gaussian uncertainty: the steering control is subject to an unknown, fixed bias

$$\delta_t = \bar{\delta}_t + \widehat{\delta}_t,$$

where $\bar{\delta}_t$ is the control input and $\widehat{\delta}_t$ is a fixed, unknown offset uniformly sampled on the interval $[-\pi/10, +\pi/10]$. This bias does not change over the course of the mission; however, the agent does not receive observations of its own state, and thus cannot ascertain the value of $\widehat{\delta}_t$. By using closed-loop RRT, this poor mapping will only result in a bounded error (Luders et al. (2010a)), making it still possible to control safely through the environment. Nonetheless, PCC-RRT is appropriate here to address the non-Gaussian uncertainty, as well as the nonlinear dynamics.

The PCC-RRT algorithm, given in Algorithm 6, is structured in this case by generating 20 particles for each trajectory state. Each particle samples on a joint distribution for both $\widehat{\delta}_t$ and the dynamic obstacle placement, which is itself distributed according to the likelihoods generated by the RR-GP prediction. The dynamic obstacle state is assumed to be correlated along each trajectory, modeled as a fixed white noise sample that is transformed relative to each state distribution. A path is deemed probabilistically feasible if $p_{\text{safe}}^{\text{path}} \geq 0.8$ is satisfied.

In this scenario, 24 trials are performed (one for each trajectory in the training data); the same algorithms are compared, but this time only using $p_{\text{safe}}^{\text{path}} = 0.8$ for comparison. Three quantities were measured and averaged across these trials: the percentage of trials in which the vehicle safely reaches the goal; the average duration of such paths; and the average time to generate an RRT/PCC-RRT tree node. Table 5 shows the results for the trials performed. Not only

does PCC-RRT successfully guide the uncertain agent to the goal in more trials than the nominal RRT algorithms, but it also exceeds the desired probabilistic bound of 0.8 across the path. As expected, though, there is a significant trade-off in per-node computational complexity.

Figure 13 shows a typical execution of the PCC-RRT algorithm for this example, as well as a demonstration of typical CC-RRT trees generated in this work (RR-GP output is suppressed for clarity), showing the nonlinearity of the dynamics. The agent initially plans a path above all obstacles to the goal (Figure 13(a)); however, this path becomes infeasible as the planner assesses that the likely trajectory of the dynamic obstacle (magenta) will cause a sufficient number of particles to become infeasible (Figure 13(b)). On the subsequent step, the planner selects an alternative, incomplete path which successfully avoids the obstacle (Figure 13(c)). Within several seconds, a new path to the goal has been found (Figure 13(d)).

9 Conclusion

This paper has developed a real-time path planning framework which allows autonomous agents to safely navigate environments while avoiding dynamic obstacles with uncertain motion patterns. A key contribution of the algorithm is the RR-GP learned motion model, which efficiently identifies predicted trajectories for a dynamic obstacle with multiple behaviors by combining Gaussian processes with a sampling-based reachability computation. As demonstrated, this motion model is capable of developing motion predictions conditioned on dynamic feasibility with runtimes suitable for real-time operation. Further, by integrating these RR-GP predictions within an appropriately modified CC-RRT planning framework, an autonomous agent can identify probabilistically safe trajectories in the presence of these dynamic obstacles. Real-time simulation results have demonstrated the effectiveness of the integrated approach in improving overall vehicle safety for a variety of dynamics, environments, and behaviors.

Future work will focus on ways to potentially increase the complexity of the GP modeling while maintaining real-time suitability. By increasing the GP model complexity, a wide variety of potentially relevant behaviors can be represented, such as correlated position GPs, dynamic obstacles with more complex dynamics, and interaction terms between the agent and dynamic obstacles. However, any such increase in the model complexity can significantly affect real-time performance, requiring a careful trade-off and continued improvements to algorithmic performance. Other future work includes demonstration of RRT* integration (Karaman and Frazzoli (2009)) and simultaneous interaction with multiple dynamic obstacles.

References

Althoff D, Wollherr D, Buss M (2011) Safety assessment of trajectories for navigation in uncertain and dynamic environments. In: IEEE International Conference on Robotics and Automation (ICRA)

Amidi O, Thorpe C (1990) Integrated mobile robot control. SPIE Mobile Robots V pp 504–523

Aoude G, Joseph J, Roy N, How J (2011) Mobile Agent Trajectory Prediction using Bayesian Nonparametric Reachability Trees. In: AIAA Infotech@Aerospace Conference

Aoude GS (2011) Threat Assessment for Safe Navigation in Environments with Uncertainty in Predictability. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge, MA

Aoude GS, Luders BD, How JP (2010a) Sampling-based threat assessment algorithms for intersection collisions involving errant drivers. In: IFAC Symposium on Intelligent Autonomous Vehicles, Lecce, Italy

Aoude GS, Luders BD, Lee KKH, Levine DS, How JP (2010b) Threat assessment design for driver assistance system at intersections. In: IEEE Conference on Intelligent Transportation Systems, Maderia, Portugal, pp 1855–1862

Aoude GS, Luders BD, Levine DS, How JP (2010c) Threat-aware path planning in uncertain urban environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, pp 6058–6063

Bennewitz M, Burgard W, Cielniak G, Thrun S (2005) Learning motion patterns of people for compliant robot motion. International Journal of Robotics Research 24:31–48

Blackmore L (2006) A probabilistic particle control approach to optimal, robust predictive control. In: AIAA Guidance, Navigation, and Control Conference (GNC)

Blackmore L, Li H, Williams B (2006) A probabilistic approach to optimal robust path planning with obstacles. In: American Control Conference (ACC)

Blackmore L, Ono M, Bektassov A, Williams BC (2010) A probabilistic particle-control approximation of chance-constrained stochastic predictive control. IEEE Transactions on Robotics 26(3):502–517

Calafiore GC, Ghaoui LE (2007) Linear programming with probability constraints – part 1. In: American Control Conference (ACC)

Deisenroth MP, Huber MF, Hanebeck UD (2009) Analytic Moment-based Gaussian Process Filtering. In: International Conference on Machine Learning (ICML), Montreal, Canada, pp 225–232

Ding H, Reißig G, Groß D, Stursberg O (2011) Mixed-integer programming for optimal path planning of robotic manipulators. In: IEEE International Conference on Automation Science and Engineering

Earl M, D'Andrea R (2005) Iterative MILP methods for vehicle control problems. IEEE Trans on Robotics 21:1158–1167

Frazzoli E, Dahleh MA, Feron E (2002) Real-time motion planning for agile autonomous vehicles. AIAA Journal of Guidance, Control, and Dynamics 25(1):116–129

Fulgenzi C, Tay C, Spalanzani A, Laugier C (2008) Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, pp 1056–1062

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA, USA

Girard A, Rasmussen CE, Quintero-Candela J, Murray-smith R (2003) Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In: Advances in Neural Information Processing Systems, MIT Press, pp 529–536

Henry P, Vollmer C, Ferris B, Fox D (2010) Learning to navigate through crowded environments. In: IEEE International Conference on Robotics and Automation (ICRA)

How JP, Bethke B, Frank A, Dale D, Vian J (2008) Real-time indoor autonomous vehicle test environment. IEEE Control Systems Magazine 28(2):51–64

iRobot (2011) iRobot: Education & research robots. URL `http://store.irobot.com/shop/index.jsp?categoryId=3311368`

Joseph J, Doshi-Velez F, Roy N (2010) A Bayesian Nonparametric Approach to Modeling Mobility Patterns. In: AAAI

Joseph J, Doshi-Velez F, Huang AS, Roy N (2011) A Bayesian nonparametric approach to modeling motion patterns. Autonomous Robots 31(4):383–400

Karaman S, Frazzoli E (2009) Sampling-based motion planning with deterministic μ-calculus specifications. In: IEEE Conference on Decision and Control (CDC)

Kuchar JK, Yang LC (2002) A review of conflict detection and resolution modeling methods. IEEE Transactions on Intelligent Transportation Systems 1(4):179–189

Kuwata Y, Teo J, Fiore G, Karaman S, Frazzoli E, How JP (2009) Real-time motion planning with applications to autonomous urban driving. IEEE Transactions on Control Systems Technology 17(5):1105–1118

Lachner R (1997) Collision avoidance as a differential game: Real-time approximation of optimal strategies using higher derivatives of the value function. In: IEEE International Conference on Systems, Man, and Cybernetics, vol 3, pp 2308–2313

LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. 98-11, Iowa State University

LaValle SM (2006) Planning Algorithms. Cambridge University Press, Cambridge, U.K.

Lavalle SM, Sharma R (1997) On motion planning in changing, partially-predictable environments. International Journal of Robotics Research 16:775–805

Leonard J, How JP, Teller S, Berger M, Campbell S, Fiore G, Fletcher L, Frazzoli E, Huang A, Karaman S, Koch O, Kuwata Y, Moore D, Olson E, Peters S, Teo J, Truax R, Walter M, Barrett D, Epstein A, Maheloni K, Moyer K, Jones T, Buckley R, Antone M, Galejs R, Krishnamurthy S, Williams J (2008) A perception-driven autonomous urban vehicle. Journal of Field Robotics 25(10):727–774

Luders B, How JP (2011) Probabilistic feasibility for nonlinear systems with non-Gaussian uncertainty using RRT. In: AIAA Infotech@Aerospace Conference, St. Louis, MO

Luders B, Karaman S, Frazzoli E, How JP (2010a) Bounds on tracking error using closed-loop rapidly-exploring random trees. In: American Control Conference (ACC), Baltimore, MD, pp 5406–5412

Luders B, Kothari M, How JP (2010b) Chance constrained RRT for probabilistic robustness to environmental uncertainty. In: AIAA Guidance, Navigation, and Control Conference (GNC), Toronto, Canada

Maile M, Zaid FA, Caminiti L, Lundberg J, Mudalige P (2008) Cooperative Intersection Collision Avoidance System Limited to Stop Sign and Traffic Signal Violations. Tech. rep., midterm Phase 1 Report

Mazor E, Averbuch A, Bar-Shalom Y, Dayan J (2002) Interacting multiple model methods in target tracking: a survey. Aerospace and Electronic Systems, IEEE Transactions on 34(1):103–123

Melchior NA, Simmons R (2007) Particle RRT for path planning with uncertainty. In: IEEE International Conference on Robotics and Automation (ICRA)

Miloh T, Sharma S (1976) Maritime collision avoidance as a differential game. Institut fur Schiffbau der Universitat Hamburg

Rasmussen CE, Williams CKI (2005) Gaussian Processes for Machine Learning. The MIT Press

Sorenson H (1985) Kalman filtering: theory and application. IEEE

Tay C, Laugier C (2007) Modelling Smooth Paths Using Gaussian Processes. In: International Conference on Field and Service Robotics

Thrun S, Burgard W, Fox D (2005) Probabilistic Robotics. MIT Press, Cambridge, MA

Trautman P, Krause A (2010) Unfreezing the robot: Navigation in dense, interacting crowds. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Vasquez D, Fraichard T, Aycard O, Laugier C (2008) Intentional motion on-line learning and prediction. Machine

Vision and Applications 19(5):411–425
Vitus MP, Pradeep V, Hoffmann GM, Waslander SL, Tomlin CJ (2008) Tunnel-MILP: Path planning with sequential convex polytopes. In: AIAA Guidance, Navigation, and Control Conference (GNC), Honolulu, HI
Wu A, How J (2012) Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles. Autonomous Robots pp 1–16
Yepes J, Hwang I, Rotea M (2007) New algorithms for aircraft intent inference and trajectory prediction. AIAA Journal on Guidance, Control, and Dynamics 30(2):370–382
Zhu Q (2002) Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation. IEEE Transactions on Robotics and Automation 7(3):390–397

Georges S. Aoude is a consultant at the Dallas office of the Boston Consulting Group. He received the B.Eng. degree in Computer Engineering from McGill University in 2005 and the S.M. and Ph.D. in Aeronautics and Astronautics from MIT in 2007 and 2011, respectively. On the MIT SPHERES team, he designed spacecraft reconfiguration maneuvers performed onboard the ISS. Research interests include intent prediction, threat assessment, and path planning under uncertainty.

Brandon D. Luders is a Ph.D. candidate in the Department of Aeronautics and Astronautics at MIT, is a member of the Aerospace Controls Laboratory, and participated in the Agile Robotics for Logistics program at MIT from 2008-2010. He received his B.S. in Aerospace Engineering at Georgia Tech in 2006, and his S.M. in Aeronautics and Astronautics at MIT in 2008. Research interests include path planning under uncertainty for autonomous vehicles.

Joshua M. Joseph is a graduate student in the Department of Aeronautics & Astronautics at MIT and is currently a member of the Computer Science and Artificial Intelligence Laboratory. He received a B.S. in Mechanical Engineering and Applied Mathematics from Rochester Institute of Technology and a S.M. from MIT. He is interested in decision-making under uncertainty in data-limited, unknown environments using reduced-order models and Bayesian nonparametrics.

Nicholas Roy is an Associate Professor in the Department of Aeronautics & Astronautics at the Massachusetts Institute of Technology and a member of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. He received his Ph. D. in Robotics from Carnegie Mellon University in 2003. His research interests include mobile robotics, decision-making under uncertainty, human-computer interaction, and machine learning.

Jonathan P. How is the Richard Maclaurin Professor of Aeronautics and Astronautics at MIT. He received a B.A.Sc. from the U. of Toronto in 1987 and his S.M. and Ph.D. in Aeronautics and Astronautics from MIT in 1990 and 1993. Prior to joining MIT in 2000, he was an Assistant Professor at Stanford University. Research interests include robust coordination and control of autonomous vehicles. He is an Associate Fellow of AIAA, and a senior member of IEEE.