# Anytime RRBT for Handling Uncertainty and Dynamic Objects

Hyunchul Yang[1] Jongwoo Lim[2] Sung-eui Yoon[3]

***Abstract*— We present an efficient anytime motion planner for mobile robots that considers both other dynamic obstacles and uncertainty caused by various sensors and low-level controllers. Our planning algorithm, which is an anytime extension of the Rapidly-exploring Random Belief Tree (RRBT), maintains the best possible path throughout the robot execution, and the generated path gets closer to the optimal one as more computation resources are allocated. We propose a branch-and-bound method to cull out unpromising areas by considering path lengths and uncertainty. We also propose an uncertainty-aware velocity obstacle as a simple local analysis to avoid dynamic obstacles efficiently by finding a collision-free velocity. We have tested our method with three benchmarks that have non-linear measurement regions or potential collisions with dynamic obstacles. By using the proposed methods, we achieve up to five times faster performance given a fixed path cost.**

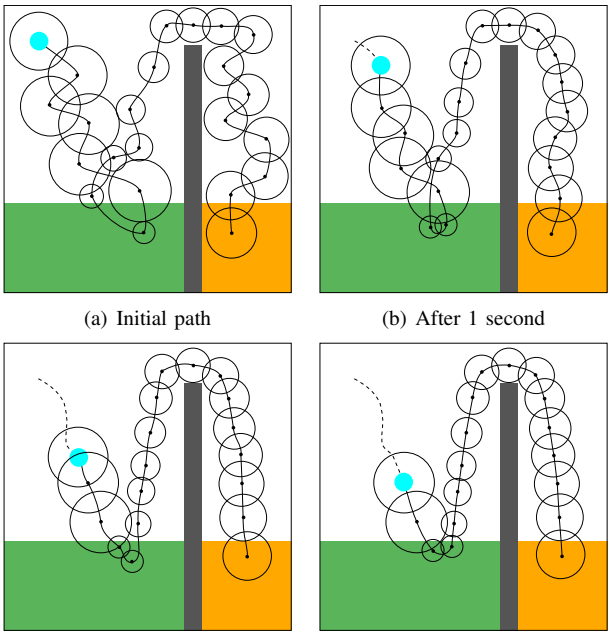
## I. INTRODUCTION

The mobile robot navigation problem in dynamic environments poses crucial challenges to motion planning research. One of main issues is to predict potential collisions against other dynamically moving objects and to find collision-free paths in an efficient manner. Moreover, various noises from low-level controllers and sensors are generated and aggravated with external disturbances (e.g., friction) in the real world. The motion planner, therefore, should take into account the uncertainty. On top of these two issues of avoiding collisions against other moving objects and considering uncertainty, it is very important to compute an initial path efficiently and to improve it to the optimal one as we have more computational time.

Many prior techniques have been proposed for the motion planning problem. Most recent ones are based on sampling approaches such as RRT [1], PRM [2], and EST [3]. Recently, Rapidly-exploring Random Belief Tree (RRBT) method has been proposed for handling uncertainty with a useful feature of computing the optimal path [4]. While RRBT has such useful features, it can generate an excessive amount of belief nodes considering different paths and uncertainty levels. As a result, it tends to be too slow to be used in practice.

**Main contributions.** To address the aforementioned issues, we propose a novel, anytime RRBT method that considers uncertainty and generates collision-free paths, which are constantly improved toward the optimal one during execution (Sec. IV-B). We also use an efficient branch-and-bound approach to cull out unpromising belief nodes to improve the overall performance of our method, while preserving the optimality. We also propose to use an Uncertainty-aware Velocity Obstacle (UVO) as a simple but effective local analysis for computing a collision-free velocity of our robot against other dynamic obstacles.

To demonstrate the anytime and optimality properties of our method, we have applied it to two different problems in simulation environments. By using our branch-and-bound method, our method achieves up to five times faster performance over the prior RRBT method given a fixed path length, since it can effectively cull out unpromising belief nodes and focus more on exploring new areas. Furthermore, our method with the proposed UVO achieves shorter paths in a less computational time over our method w/o UVO, since a velocity avoiding collisions against other moving objects can be computed quickly. These results indicate that our anytime RRBT takes one step forward towards efficient optimal path planning considering uncertainty in dynamic environments.



(a) Initial path (b) After 1 second

(c) After 3 seconds (d) After 5 seconds

Fig. 1. Runtime results of our anytime RRBT method. This simulation environment has the space of $5 \times 5(m^2)$. We show an initially computed path for the robot (a). (b), (c), and (d) show updated trajectories while following the prior path. The robot can sense the environment only in the green region. Dotted lines are committed trajectories, and the goal is shown as the yellow region. The circles represent the uncertainty; bigger ones mean higher uncertainty.

[1]Hyunchul Yang is at Robotics Program, KAIST, Daejeon, South Korea. [2]Jongwoo Lim is at Division of Computer Science and Engineering, Hanyang University, Seoul, South Korea. [3]Sung-eui Yoon is a corresponding author at School of Computing, KAIST, Daejeon, South Korea. hcyang06@gmail.com, jongwoo.lim@gmail.com, sungeui@gmail.com

## II. RELATED WORK

In this section, we discuss prior work on sampling-based motion planning, and its extensions to support anytime, uncertainty, and dynamic environments.

### A. Sampling-based Motion Planning

Sampling-based algorithms (e.g., RRT[1], PRM[2], and EST[3]) have been used to solve motion planning problems in high-dimensional configuration spaces. Among these techniques, Rapidly-exploring Random Tree (RRT) has received much attention over the last decade. RRT operates by growing a tree in the configuration space, takes random samples iteratively, and then expands the tree towards the new samples from the closest nodes in the tree.

RRT has many advantages including the probabilistic completeness and scalability in high dimensional spaces. These advantages triggered numerous extensions including RRT$^*$ computing optimal solutions [5].

**Anytime extension.** RRT can find an initial solution quickly, but it does not always find the optimal solution. To overcome this issue, RRT$^*$[5], Cloud RRT$^*$[6], and RRT-X[7] algorithms have been proposed. Furthermore, its anytime extension has been proposed to support the real world robotic systems. Notable examples are anytime RRT [8] and anytime RRT$^*$ [9], which are execution-time replanning algorithms for RRT and RRT$^*$, respectively. These studies, however, are not designed for handling uncertainty arising from real sensors and robot motions.

### B. Uncertainty

The uncertainty arises due to various errors of control and sensors. One of most popular methods is Linear Quadratic Gaussian Motion Planning (LQG-MP) [10] that models uncertainty of various system states using Gaussian noise and propagates the uncertainty using the Kalman filter iteratively. Patil et al. [11] improves accuracy in critical cases involved with collisions based on a truncated Gaussian distribution considering that a prior condition is collision-free. Xu et al. [12] addressed trajectory planning of car-like robots using the LQG framework. Liu et al. [13] combined RRT with LQG-MP to handle uncertainty.

Recently, Bry et al. [4] extended the prior RRT method to Rapidly-exploring Random Belief Tree (RRBT) considering uncertainty. RRBT contains multiple belief nodes with different uncertainty level and path lengths in a graph. It can preserve individual uncertainty estimates along multiple trajectories passing a particular location. While this method can handle many general scenarios, the benefit comes with a cost; a slow reasoning speed. Motion planners considering uncertainty and real-time performance together has been less studied. Our method aims for handling both uncertainty and dynamic objects based on the anytime extension and uncertainty-aware velocity obstacle model.

### C. Dynamic Environments

Different approaches in motion planning for handling dynamic obstacles have been developed. To predict trajectory of other moving robots, Gaussian-based ones [14], [15] have been commonly used. These studies work well for environments with many occluded zones. They, however, tend to be slow, and do not consider to compute the velocity of the robot avoiding collisions.

Velocity Obstacle (VO) [16] represents a robot using its radius and velocity to avoid potential collisions. The VO method has been applied to numerous applications such as multiple robot planning and driver assistance [17], [18], [19].

The VO method has been extended into Reciprocal Velocity Obstacle (RVO) [20] and Hybrid Reciprocal Velocity Obstacle (HRVO) [21] for considering reciprocity between robots. HRVO was adopted for recent motion planning [22], [23], [24]. We also adopt its idea and apply to our RRBT based planner to consider uncertainty and dynamic objects in an efficient way.

## III. BACKGROUND

In this section, we describe our problem formulation, and a brief review on the RRBT algorithm [4].

### A. Problem Description

Let $S$ and $U$ be the state space and control space, respectively. The robot is described by a discrete time description of its dynamics and sensors:

$$s_t = f(s_{t-1}, u_{t-1}, q_t), \qquad q_t \sim N(0, Q_t) \tag{1}$$

$$z_t = h(s_t, r_t), \qquad r_t \sim N(0, R_t) \tag{2}$$

where $s_t \in S$ is the state of the robot, $u_t \in U$ is the control input, $q_t$ is a random process disturbance drawn from a zero-mean Gaussian distribution with variance $Q_t$, $z_t$ is the measurement of the robot's state, and $r_t$ is a measurement noise drawn from a zero-mean Gaussian with variance $R_t$. Let $S_{obs}$, $S_{goal} \subset S$ to represent obstacles and goal regions, respectively. The obstacle free regions can be then denoted by $S_{free} = S \backslash S_{obs}$.

The robot's state can be described by $s_t \sim N(\hat{s}_t, \Sigma_{R_t})$ at time $t$, and then its initial state is given as follows:

$$s_0 \sim N(\hat{s}_0, \Sigma_{R_0}), \tag{3}$$

where $\hat{s}$ and $\Sigma_R$ are the mean and covariance of $s$, respectively.

The state of dynamic obstacles is described by

$$o_t^i \sim N(\hat{o}_t^i, \Sigma_{o_t}^i) \tag{4}$$

where $\hat{o}^i$ and $\Sigma_o^i$ are the mean and covariance of the $i$-th obstacle, respectively.

Let $\pi_{0:t}$ denotes a path consisting of a series of states and control inputs: $(s_0, u_0, ..., s_t, u_t)$. The optimal path planning problem is then defined to minimize the following cost function [10]:

$$\mathbb{E}\left[cost(\pi_{0:T})\right], \tag{5}$$

$$P(s_T \notin S_{goal}) < \delta, \; P(s_t \in S_{obs}) < \delta, \; \forall t \in [0, T], \tag{6}$$

where $T$ is the total time to reach the goal and $\delta$ is a user defined threshold for the chance-constraint [25]. As

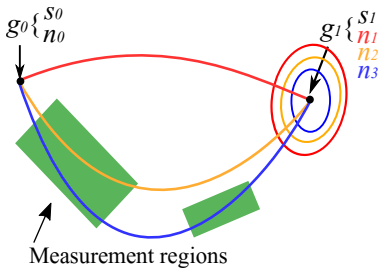



Fig. 2. Each vertex (e.g., $g_1$) in our method can have multiple belief nodes (e.g., $n_1, n_2$, and $n_3$) depending on their uncertainty levels (drawn in ellipses) and path distances. The red belief $n_1$ has a low distance cost, but with a high uncertainty. On the other hand, the blue belief has a long distance cost, but with a low uncertainty value. The yellow belief node has in-between values for the uncertainty and distance.

$\delta$ gets smaller, we try to find paths that have smaller probability to have collisions. For our tests, we set $\delta$ to be 0.159, corresponding to the one sigma value of the normal distribution. A cost function used in our method is discussed later (Sec. IV-A).

### B. Rapidly-exploring Random Belief Tree (RRBT)

RRBT is an incremental sampling based motion planning algorithm that provably converges to the optimal path, taking the uncertainty into consideration [4]. This algorithm incrementally constructs a graph with random sample strategy. The algorithm builds a graph with a set of vertices, $G$, and each vertex can have multiple belief nodes. More precisely, let $g \in G$ denote a vertex, then $g.s$ and $g.N$ denote the state of vertex and the set of belief nodes associated with the vertex $g$, respectively. Each belief node $n \in g.N$ stores its parent belief node in the associated path.

A series of belief nodes in parent-child relationship defines a path that starts from the initial vertex. Since a vertex can have multiple belief nodes, different paths passing the vertex can be defined. For example in Fig. 2 the vertex $g_1$ has three different belief nodes, $n_1, n_2$, and $n_3$, representing different uncertainty levels acquired from their associated trajectories from the vertex $g_0$.

RRBT can generate an excessive amount of belief nodes to consider different uncertainty levels and path lengths, and thus can be very slow. To ameliorate this issue, RRBT uses a partial path ordering method:

$$n_a < n_b \Leftrightarrow (n_a.\Sigma < n_b.\Sigma) \wedge (n_a.dist < n_b.dist), \qquad (7)$$

where $n_a$ and $n_b$ represent two different partial paths to the same vertex associated with the node $n$. $\Sigma$ and $dist$ associated with each belief node $n$ are a state estimate of the covariance and the trajectory distance, respectively. When $n_a < n_b$, we treat that the path passing $n_a$ is better than the other, and we can prune the other path passing $n_b$.

This approach guarantees not to prune the optimal path. It, however, has a low culling ratio, and can generate an excessive number of belief nodes in the same vertex, resulting in unsuitable performance for real-time robotics systems.

**Algorithm 1** Anytime RRBT

**Require:** Vertices $G$, State $s_{init}$, Belief $n_{init}$, State $s_{goal}$

$g_{init}.s \leftarrow s_{init}$
$g_{init}.N \leftarrow \{n_{init}\}$
$G \leftarrow \{g_{init}\}$
**loop**
  $[s_{robot}, \Sigma_{robot}, O] \leftarrow$ UpdateMeasurement()
  **if** $s_{robot} \in s_{goal}$ **then**
    **return**
  **end if**
  ReinitializeGraph($G$, $s_{robot}$, $\Sigma_{robot}$, $O$)
  $g_{new} \leftarrow$ RandomVertex($G$, $O$)
  $G_{near} \leftarrow$ Near($G$, $g_{new}.s$)
  $[G, G_{new}] \leftarrow$ Wire($G$, $G_{near}$,$\{g_{new}\}$)
  $[G, G_{new}] \leftarrow$ Wire($G$, $G_{new}$, $G_{near}$)
**end loop**

We propose anytime extensions with an effective branch-and-bound method for RRBT to handle uncertainty in an efficient way.

## IV. The Proposed Method

In this section, we explain our anytime RRBT method with an efficient branch-and-bound method, followed by our uncertainty-aware velocity obstacle method.

### A. Cost Function

For path planning we need to define a cost function that associates a cost with a path (e.g., Eq. 5). Various cost functions that model observation and control uncertainty have been studied in the field. Among available cost functions, we focus on balancing between the travel distance of the robot and the amount of uncertainty. In general, the robot should move around in environments to collect sensor data. We use a simple, monotonic form for our cost function:

$$cost(\pi_{0:t}) = \alpha Distance(\pi_{0:t}) + \beta ||\Sigma_{R_t}||_2, \qquad (8)$$

where $Distance(\pi_{0:t})$ is the travel distance following the trajectory $\pi_{0:t}$, $\Sigma_{R_t}$ is the uncertainty at time $t$, and $\alpha$ and $\beta$ are the weights. We use the L2 norm of the covariance matrix of $\Sigma_{R_t}$, since it has been known to return the maximum variance among all the possible vector directions.

### B. Anytime Extensions

We present an anytime RRBT method, inspired by anytime RRT$^*$[9]. Our anytime RRBT finds the asymptotically optimal path, and the planned path is constantly updated while a robot is following the path. Our method has two extensions over the prior RRBT. The first is a graph re-initializing method for making the RRBT reusable. The second is a branch-and-bound method for accelerating the estimation process. A pseudocode of our anytime RRBT is shown in Alg. 1.

*1) Reinitialize the graph:* Suppose that we compute a trajectory $\pi_{0:T}$ up to the goal position at the last time of path planning, $t_p$, we maneuver our robot to follow the trajectory, and are about to re-run our anytime RRBT at the current time, $t_c$. In this case, the portion in the planned trajectory up to $t_c$, $\pi_{0:t_c}$, is executed and already committed.

To accommodate dynamic changes occurred in-between $t_p$ and $t_c$, we first update estimates on states (e.g., positions and velocities) of other dynamic obstacles using the Kalman filter. We also delete already committed nodes and update the underlying graph data structure to refine the trajectory. We then make a new vertex and a belief node that represents the robot's current state, $x_{t_c}$, and connect it with the graph as the new source in the graph for the planning process. We also perform wire operations to compute better paths in the similar manner to those of RRT* [9]. Finally, we update each belief node starting from the new source and check whether it collides with updated dynamic obstacles. When belief nodes are in collisions, we prune them. Specifically, we use our uncertainty-aware velocity obstacle for the collision checking process (Sec. IV-C). We also perform our branch-and-bound for each belief node, described in below.

*2) Branch-and-bound:* The branch-and-bound technique has been used to accelerate various search problems including graph and tree based search. We apply this concept to our anytime RRBT. We define a *CostToGo* function like the heuristic function of A* techniques [26].

Let $n^*$ be an arbitrary belief node, $c^*$ be the cost of the optimal path of connecting $n^*$ and $n_{goal}$, which is a node generated for the goal position. It has been proven that when a heuristic function satisfies three admissibility conditions, it is guaranteed that the $A^*$ algorithm finds the optimal solution [27]. These admissibility conditions are that the graph has finite edges, the heuristic function returns positive values, and it is conservative; it returns a value that is equal to or less than the optimal cost.

Unfortunately, when we designed our heuristic function to be conservative, we found that it is ineffective in terms of culling for our branch-and-bound method. This phenomenon is occurred mainly when the heuristic function returns too conservative values to the actual cost, as discussed in the literature [27].

For achieving efficient performance, we propose to use the following, approximate heuristic function:

$$\begin{aligned} CostToGo(n^*) = \alpha EstDist(n^*, n_{goal}) \; + \\ \beta \max(0, EstUncertain(n^*, n_{goal}) - \varepsilon), \end{aligned} \tag{9}$$

where $\alpha$ and $\beta$ are the same weights used in our cost function (Eq. 8).

For the conservative estimation on maneuvering distance of our robot, we assume that the robot moves toward the goal straightly, and use the estimate distance for $EstDist(\cdot)$. For the estimated uncertainty of $EstUncertain(\cdot)$, we also assume that the robot follows the straight line and the uncertainty associated with $n^*$ increases according to the Kalman filter by following the straight line. To consider the non-linear nature of uncertainty and design a conservative heuristic function, $EstUncertain(\cdot)$ should return the zero uncertainty, but this setting does not result in effective culling. Instead, we choose to reduce the linearly increasing uncertainty with a small, but randomly generated value, $\varepsilon$. We guarantee that the randomly generated value can make $EstUncertain(\cdot)$ to be zero in a probabilistic manner, to satisfy the admissibility conditions [28].

We then cull out belief nodes which satisfy the following condition:

$$N' = \left\{ n \in N | Cost(n) + CostToGo(n) \geq Cost(n_{goal}) \right\}. \tag{10}$$

Note that our final heuristic function is not conservative always, but is conservative probabilistically. This guarantees that our method can compute optimal paths in a probabilistic sense. We found that this works well in practice, while achieving high performance with the probabilistic optimal guarantee.

We also tested different places to perform our branch-and-bound method. We found that evaluating costs and performing the branch-and-bound at the end of the reinitialization process shows the best performance in our tested benchmarks.

**Algorithm 2** RandomVertex

**Require:** Graph $G$, Obstacles $O$

**repeat**
  $s_{rand} \leftarrow$ RandomState()
  $g_{nst} \leftarrow$ Nearest($G$, $s_{rand}$)
  **for all** $n \in g_{nst}.N$ **do**
    $[s_{rand}, n_{rand}] \leftarrow$ getCollisionFree($O$, $g_{nst}.s$, $s_{rand}$, $n$)
    **if** $n_{rand} \neq nobelief$ **then**
      *break*
    **end if**
  **end for**
**until** $n_{rand} \neq nobelief$
**return** Vertex($s_{new}$, $n_{new}$)

**Algorithm 3** getCollisionFree

**Require:** $O$, $s_a$, $s_b$, $n_a$

$s_{new} \leftarrow s_b$
$n_{new} \leftarrow$ Propagate($s_a$, $s_b$, $n_a$)
**if** $n_{new} \neq nobelief$ **then**
  $s_{new} \leftarrow$ UVOCheck($O$, $s_b$, $n_{new}$)
  $n_{new} \leftarrow$ Propagate($s_a$, $s_{new}$, $n_a$)
**end if**
**return** $[s_{new}, n_{new}]$

### C. Collision Checking using UVO

As we generate random samples and attempt to connect them to nodes in the graph, we need to check collisions along the computed path. Once we have collisions, we could reject those random samples and try other samples. Instead of this naive approach, we perform a simple local analysis and compute a velocity for our robot avoiding collisions against other dynamic objects within our anytime RRBT. As a simple

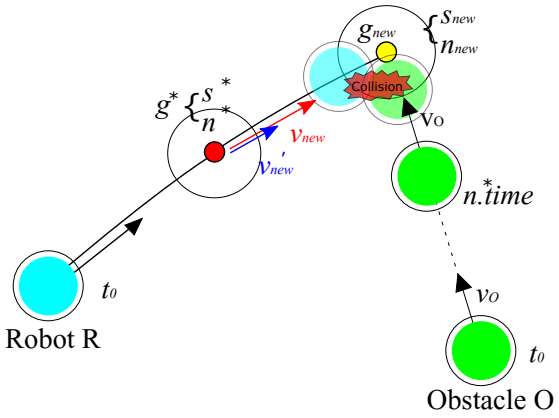


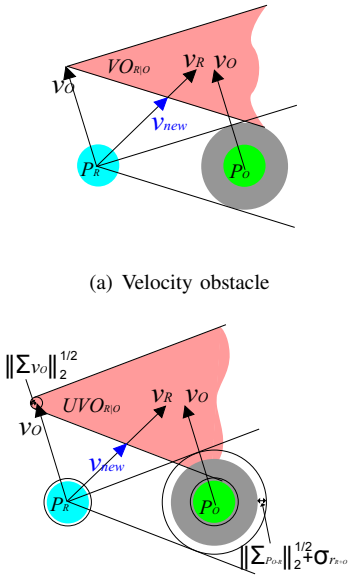

Fig. 3. We generate a new vertex $g_{new}$ that has a speed $v_{new}$ represented in its state $s_{new}$, and try to connect the vertex to $g^*$. We use our UVO to check collision and compute a new velocity $v'_{new}$.

local analysis, we propose to use an Uncertainty-aware Velocity Obstacle (UVO) method. Pseudocodes of generating a random vertex, checking collisions, and updating velocities are shown in Alg. 2 and 3.

During the process of our anytime RRBT, we generate a new sample state, $s_{new}$, that has random position and speed (Fig. 3). Suppose that $n^*$ is chosen as a belief node that is one of nearest neighbors to the new sample state. We then attempt to connect the node $n^*$ to the state $s_{new}$. For this, we check whether the robot collides with static obstacles, while propagating a belief from $n^*$ to the state $s_{new}$ using a *Propagate* function. The *Propagate* function returns a new belief node $n_{new}$ at the state $s_{new}$, while estimating the belief using the Kalman filter. If the chance-constraint (Eq. 6) is violated, the function then returns no belief node.

Once the propagation succeeds, we then check whether the robot collides with dynamic obstacles (Fig. 3). Given $i$-th dynamic object, $o_i$, let $P_{o_i}$, $r_{o_i}$, $v_{o_i}$ denote its measured position, radius of a circle bounding the object, and velocity, respectively. We estimate position and uncertainty of the obstacle $o_i$ at time $n^*.time$ using the Kalman filter. After estimating the obstacle position, we use our UVO to check whether there are collisions between the robot and $o_i$ until the time of the propagation, $n_{new}.time$. Furthermore, when there are collisions, we use UVO to compute a new, positive velocity for the new sample state $s_{new}$, instead of blindly generating random states.

Velocity obstacle has been used for local collision avoidance and navigation of a robot surrounded by multiple moving obstacles in two dimensions [16]. Let $R$ be a circle bounding our robot and $O$ be another circle bounding $i$-th dynamic obstacle with radii $r_R$ and $r_O$, respectively. Let $P_R$ and $P_O$ denote the center points of those circles, respectively. We also use $v_R$ and $v_O$ denote the velocities of the robot and obstacle, respectively. The velocity obstacle for $R$ induced by $O$, $VO_{R|O}$, is a set of velocities for the robot $R$ that result in collision with the obstacle $O$ in near future (Fig. 4.(a)).




(a) Velocity obstacle

(b) Uncertainty-aware velocity obstacle

Fig. 4. The upper figure shows velocity obstacle $VO_{R|O}$. The lower figure shows that our uncertainty-aware velocity obstacle $UVO_{R|O}$ has a wider region than the $VO_{R|O}$ for conservatively handling uncertainty. The robot $R$ will collide with the obstacle $O$ in near future, because the velocity $v_R$ is the UVO area. The new velocity $v_{new}$calculated to avoid collision is shown.

Precisely, $VO_{R|O}$ is defined as follows:

$$VO_{R|O} = \{v | \exists t > 0 : t(v - v_O) \in Disc(P_O - P_R, r_R + r_O)\}, \tag{11}$$

where $t$ represents a time, and $Disc(P,r)$ is a disc of radius $r$ centered at $P$. This concept was extended for considering both reciprocity and uncertainty [20]. For our purpose, we do not need to consider the reciprocity, since we cannot control trajectory of other dynamic obstacles. As a result, we design our uncertainty-aware velocity obstacle tailored to our problem.

The uncertainty-aware position, radius, and velocity of our robot $R$ and a dynamic obstacle $O$ used for our method are defined as the following:

$$\begin{aligned} P_R &\sim N(\hat{P}_R, \Sigma_{P_R}),\ r_R \sim N(\hat{r}_R, \sigma^2_{r_R}),\ v_R \sim N(\hat{v}_R, \Sigma_{v_R}), \\ P_O &\sim N(\hat{P}_O, \Sigma_{P_O}),\ r_O \sim N(\hat{r}_O, \sigma^2_{r_O}),\ v_O \sim N(\hat{v}_O, \Sigma_{v_O}), \end{aligned} \tag{12}$$

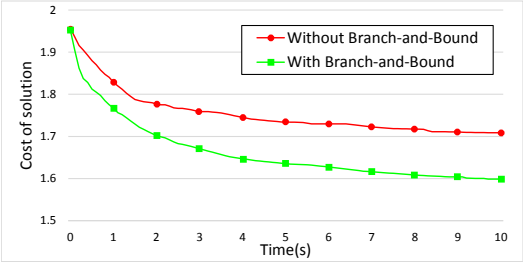
where means, variances, and covariances (e.g., $\hat{P}_O$ and $\Sigma_{P_O}$) are acquired from sensors.

For simplicity and efficiency, we define the relative position, $P_O - P_R$, and the relative radius, $r_O + r_R$, as follows:

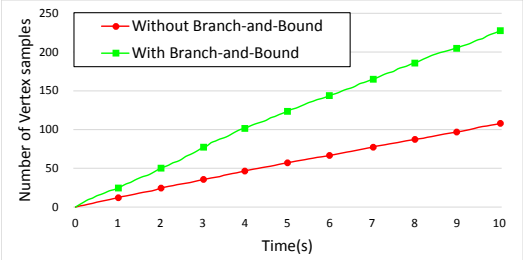
$$P_{O-R} \sim N(\hat{P}_O - \hat{P}_R, \Sigma_{P_{O-R}}),\ r_{R+O} \sim N(\hat{r}_R + \hat{r}_O, \sigma^2_{r_{R+O}}). \tag{13}$$

Our uncertainty-aware velocity obstacle, $UVO_{R|O}$, is then computed as the following:

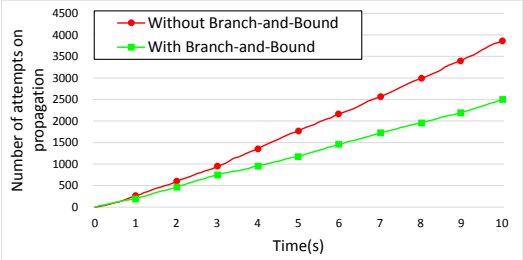
$$\begin{aligned} UVO_{R|O} = \{v | \exists t > 0 : t(v - Disc(\hat{v}_O, ||\Sigma_{v_O}||_2^{1/2})) \in \\ Disc(\hat{P}_O - \hat{P}_R, \hat{r}_R + \hat{r}_O + ||\Sigma_{P_{O-R}}||_2^{1/2} + \sigma_{r_{R+O}})\}. \end{aligned} \tag{14}$$




(a)




(b)




(c)

Fig. 5. These graphs show the average results of our method w/ and w/o our branch-and-bound method over 30 runs on the problem in Fig. 1. (a) shows costs of solutions as a function of computation time. (b) shows the number of generated vertices. (c) plots the number of attempts for propagation.

Fig. 4 shows visual illustration of velocity obstacle and our uncertainty-aware velocity obstacle. As can be in the figure, the region of velocity obstacle has the triangular shape, while the region of our uncertainty-aware velocity obstacle is an extended triangle for conservatively considering the uncertainty. To avoid collision, we need to compute a new velocity that is not in that region. In this case, we create a new state, $s'_{new}$, that has an adjusted velocity, $v'_{new}$, at the same position of $s_{new}$, to avoid the collision. We finally append the updated, new vertex $g'_{new}$ to the underlying graph of our anytime RRBT.

Note that we achieve the optimality of our anytime RRBT even with our UVO, since our method keeps to generate other random samples that can avoid collisions. Our UVO method generates samples avoiding collisions more effectively based on its simple local analysis in terms of velocity. Nonetheless, the UVO method applied only to dynamic obstacles can violate the chance-constraints locally. This could be avoided by expanding UVO more, but computing a tight bound is desirable and requires further study.

## V. Experimental Results

The proposed algorithm is implemented and tested on a system with a 3.4 GHz Intel Core i7 CPU and 16GB of memory running MAC OSX. To ensure that our approach applies well to real robots, we use the V-REP simulator [29] for the robot simulation, since it supports various dynamics related to low-level controls (e.g,. motors), noisy environments (e.g., motor control errors), and asynchronous invocations of low-level controller and our motion planning modules.

To demonstrate benefits of our methods, we design three scenes with a disc-shape mobile robot. For all the experiments, we set the weights $\alpha = 0.1$, $\beta = 0.9$ in the cost function Eq. 8. The distance value tends to be much bigger than the uncertainty value, and we thus use a small alpha value to balance them out. For $\varepsilon$ used in our heuristic function, we generate a random number uniformly in a range between 0 and 0.01 that can make $EstUncertain(\cdot)$ zero probabilistically in our tested settings.

### A. Anytime Extension Experiments

We adopt robot dynamics with various errors in the 2-D case, as suggested by Bry et al. [4],

$$x_t = x_{t-1} + u_{t-1} + q_t, \qquad q_t \sim N(0, 0.01I) \tag{15}$$

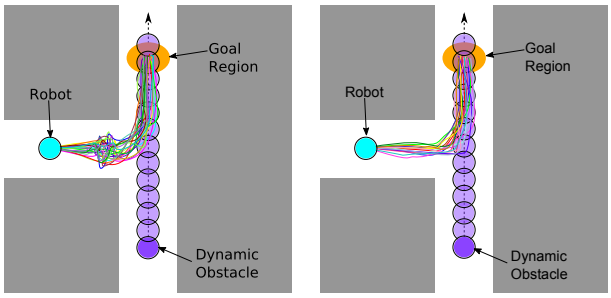
$$z_t = x_t + r_t, \qquad r_t \sim N(0, R), \tag{16}$$

where we use the same notations used for the problem definition (Sec. III-A). The $R$ is set according to locations in the environment. We assume that the robot can update sensor data only on the green region in Fig. 1. Therefore, $R$ is set 0.01 in the green regions, while $R$ is set $\infty$ in the rest of regions.

In Fig. 1, we plot committed and computed trajectories of the robot. The robot starts with a high uncertainty on states. To gather exact location information, the robot should visit the green regions at least once. The robot starts to follow an initial path after calculating the initial path. When the initial path is computed, the robot starts following it, and thanks to the anytime feature of our method, the robot computes better paths while executing the prior path, and reaches the goal in an efficient and effective way.

We measure benefits of our Branch-and-Bound (BB) method accelerating our anytime RRBT method. To focus on the BB method, the robot is kept static throughout the planning process to exclude the effect of re-initialization for the test; we let our planner to keep to generate samples and improve paths without moving. Overall, our method w/ BB culls out 15% belief nodes on average compared to those generated w/o BB.

We then look at the cost of our method w/ and w/o BB (Fig. 5-a)). We achieve lower costs given the same time budget over our method w/o BB. In a different perspective, our method w/ BB achieves less computational time given a fixed cost over our method w/o BB, and its benefits are getting bigger as we achieve lower costs. For example, at the cost of 1.8, our method achieves two times faster




(a) Our method w/o UVO (b) Our method w/ UVO

Fig. 6. Intersection scene with $3 \times 3(m^2)$ size. A purple obstacle moves toward top direction straightly with the constant velocity. The robot encounters the obstacle after a few seconds, and finds paths to the goal region (orange), while avoiding the obstacle. Our method w/o UVO avoids the obstacle with zig-zag paths (a), while ours w/ UVO reduces the speed to avoid collisions, resulting in smooth paths and better runtime performance (b).

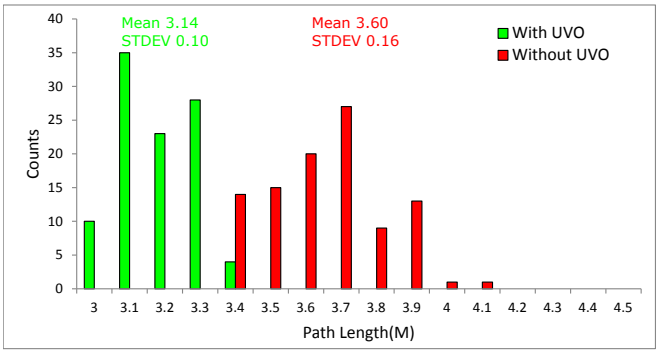
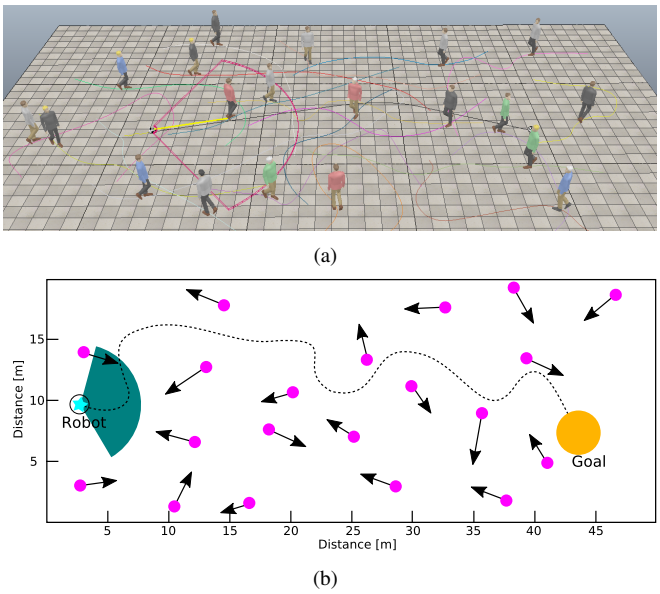



Fig. 7. Histograms of path lengths in the scene of Fig. 6. Our method w/ UVO achieves shorter path lengths over our method w/o UVO.




(a)

(b)

Fig. 8. A scene with many moving humans. (a) shows randomly generated paths for 20 agents in the simulator. (b) shows an example, robot trajectory that successfully avoids multi-obstacles using our method. The magenta dots are human with their current velocity vectors.

performance, while at the cost of 1.7, we achieve about five times faster performance.

Since our BB method culls out belief nodes with higher costs and uncertainty, our method focuses more on exploring new space by generating more vertices, resulting in computing paths with lower costs. As you can see in Fig. 5-b), our method w/ BB generates two times more vertices over our method w/o BB. On the other hand, our method w/o BB attempts to connect newly generated vertices with existing belief nodes that may have high cost and uncertainty (Fig. 5-c)), since we do not cull out such nodes. Overall, according to the BB method, we are able to cull out unpromising paths and explore new or promising paths in a more effective way.

### *B. UVO Experiments*

To test our method w/ UVO, we use a $3 \times 3(m^2)$ scene with an intersection (Fig. 6). In this scene, our cyan-colored robot has a 1 $m$ range sensor that can measure the location, velocity, and radius of other robots within the 1 $m$ distance. We set uncertainty parameters, $\Sigma_{P_O} = 0.05I$, $\sigma_{r_O} = 0.01$ and $\Sigma_{v_O} = 0.1I$, for our UVO method (Eq. 12). A purple-colored dynamic obstacle moves to the north straightly with a constant velocity. Our robot aims to reach the orange goal region, while avoiding the dynamic obstacle.

To demonstrate benefits of our method w/ UVO, we compare it against our method w/o UVO. For clear demonstration between these two different methods, we use a constant velocity for our robot instead of randomly generating velocities during our anytime RRBT; we achieve similar results by using random velocities. Especially, we look at executed trajectories of these two different methods (Fig. 6). Once we have collisions, our method w/o UVO reject the sample with the collision and continues to generate new random samples avoiding such collisions. On the other hand, our method w/ UVO performs the local analysis and modifies the velocity within the same sample to avoid collisions.

As a result, the UVO method returns an adjusted velocity, and thus its computed trajectories tend to be more smooth and shorter than those generated by our method w/o UVO (Fig. 7). Furthermore, we compute such better paths in faster performance. Specifically, our method w/ UVO makes 110 vertices per second on average with 100 trials, while our method w/o UVO makes 93 vertices per second. Overall, our method w/ UVO achieves better paths in faster running performance.

### *C. Multi-Obstacles Scene*

We also apply our method to a scene with 20 human agents, where each human agent follows his or her own path, which is generated randomly (Fig. 8). We use circles of $0.3m$ radius as bounding disks for our UVO method. We assume that our robot has a $5m$ range sensor, and uncertainty parameters are same to ones reported in Sec. V-B. We run this test 30 times, and our method is able to find collision-free paths that avoid moving agents, while moving toward the goal region in all the tested experiments. We show a video segment of this test in the accompanying video

## VI. CONCLUSION

We have presented a novel anytime RRBT method for handling uncertainty and dynamic environments in an efficient manner. Our method continuously improves collision-free paths toward the optimal solution, as we get more computational time. We have proposed an efficient branch-and-bound technique based on our heuristic function. We have also proposed our uncertainty-aware velocity obstacle to effectively perform collision detection and find an adjusted velocity avoiding collisions without rejecting samples. We have tested our methods in three different benchmarks and verified that our method has useful benefits, while effectively handling uncertainty and dynamic objects.

There are many interesting avenues for future research directions. While our method supports anytime replanning, its performance may not be enough for robots with high velocity. At a worse case, our robot may have collisions against such fast moving objects, even if we consider uncertainty within our anytime RRBT method. To achieve better performance, we would like to design a massively parallel version utilizing recent GPUs. We currently use 2D circles to bound robots for our UVO analysis. This can be applied to other shapes of robots and higher dimensions by projecting them into 2D. Nonetheless, we would like to design anisotropic shapes (e.g, ellipsis) to support elongated robots and extend it to 3D cases for more accurate analysis.

## ACKNOWLEDGMENTS

We would like to thank SGLab members and anonymous reviewers for constructive comments. This work was supported in part by IT R&D program of MSIP/KEIT [R0126-16-1108] and MSIP/NRF (2013-067321). Sung-eui Yoon is a corresponding author of the paper.

## REFERENCES

[1] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[4] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 723–730.

[5] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[6] D. Kim, J. Lee, and S.-e. Yoon, "Cloud rrt: Sampling cloud based rrt," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2519–2526.

[7] M. Otte and E. Frazzoli, "Rrt-x: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478.

[8] D. Ferguson and A. Stentz, "Anytime rrts," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5369–5375.

[9] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *ICRA*, 2011, pp. 1478–1483.

[10] J. Van Den Berg, P. Abbeel, and K. Goldberg, "Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[11] S. Patil, J. Van den Berg, and R. Alterovitz, "Estimating probability of collision for safe motion planning under gaussian motion and sensing uncertainty," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3238–3244.

[12] W. Xu, J. Pan, J. Wei, and J. M. Dolan, "Motion planning under uncertainty for on-road autonomous driving," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2507–2512.

[13] W. Liu and M. Ang, "Incremental sampling-based algorithm for risk-aware planning under motion uncertainty," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2051–2058.

[14] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1056–1062.

[15] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Autonomous Robots*, vol. 35, no. 1, pp. 51–76, 2013.

[16] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 560–565.

[17] Z. Shiller, R. Prasanna, and J. Salinger, "A unified approach to forward and lane-change collision warning for driver assistance and situational awareness," SAE Technical Paper, Tech. Rep., 2008.

[18] E. Prassler, J. Scholz, and P. Fiorini, "Navigating a robotic wheelchair in a railway station during rush hour," *The international journal of robotics research*, vol. 18, no. 7, pp. 711–727, 1999.

[19] P. Fiorini and D. Botturi, "Introducing service robotics to the pharmaceutical industry," *Intelligent Service Robotics*, vol. 1, no. 4, pp. 267–280, 2008.

[20] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1928–1935.

[21] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *Robotics, IEEE Transactions on*, vol. 27, no. 4, pp. 696–706, 2011.

[22] J. Van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "Lqg-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 346–353.

[23] Z. Shiller, O. Gal, and A. Raz, "Adaptive time horizon for online avoidance in dynamic environments," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 3539–3544.

[24] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, "Collision avoidance under bounded localization uncertainty," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1192–1198.

[25] L. Blackmore, H. Li, and B. Williams, "A probabilistic approach to optimal robust path planning with obstacles," in *American Control Conference, 2006*. IEEE, 2006, pp. 7–pp.

[26] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[27] N. J. Nilsson, *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[28] S. Yoon, S.-E. Yoon, U. Lee, and D. H. Shim, "Recursive path planning using reduced states for car-like vehicles on grid maps," *Intelligent Transportation Systems, IEEE Transactions on*, 2015.

[29] M. F. E. Rohmer, S. P. N. Singh, "V-rep: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.