Fast Sampling-Based Cost-Aware Path Planning with Nonmyopic Extensions Using Cross Entropy

Junghun Suh, Joonsig Gong, and Songhwai Oh

***Abstract*—This paper presents a cost-effective motion planning method for robots operating in complex and realistic environments. While sampling-based path planning algorithms, such as rapidly-exploring random tree (RRT) and its variants, have been highly effective for general path planning problems, it is still difficult to find the minimum cost path in a complex space efficiently since RRT-based algorithms extend a search tree locally, requiring a large number of samples before finding a good solution. This paper presents an efficient nonmyopic path planning algorithm by combining RRT* and a stochastic optimization method, called cross entropy. The proposed method constructs two RRT trees: the first tree is a standard RRT* tree which is used to determine the nearest node in the tree to be extended to a randomly chosen point and the second tree contains the first tree with additional long extensions. By maintaining two separate trees, we can grow the search tree non-myopically to improve the efficiency of the algorithm while ensuring the asymptotic optimality of RRT*. From an extensive set of simulations and experiments using mobile and humanoid robots, we demonstrate that the proposed method consistently finds low-cost paths faster than existing algorithms.**

***Index Terms*—Cost-aware path planning, energy-efficient motion planning, rapidly-exploring random tree**

I. Introduction

IT is envisioned that robots will soon perform complex tasks on behalf of humans, such as search-and-rescue in disaster or hazardous environments. Consider a nuclear power plant accident scenario, in which some regions are contaminated by radioactive materials. If a map of radioactive levels is available, it is desirable to task robots to perform the search-and-rescue operation along the path with the minimum exposure to radioactive materials. Finding the minimum exposure path becomes a difficult problem if the radioactive map shows complex terrains of radioactive levels, in addition to obstacles present in the field. Consider another scenario, in which a humanoid robot is performing a complex manipulation task in a remote area. It is desirable to perform the same task with less energy to extend the operation time of the robot in the field. Finding the most energy-efficient motion plan for a humanoid robot is a difficult problem due to its high dimensional configuration space. Both scenarios present an optimization problem, which has been extensively studied in the area of optimal control [2]. In this paper, we consider a special case of the optimal control problem, in which we search for a motion trajectory with the minimum accumulated cost for a robot, performing a task in a complex terrain or in a high dimensional space.

In order to perform tasks mentioned above, we need to find a trajectory which minimizes the accumulated cost when a robot follows the trajectory over the configuration space. Although sampling-based path planning algorithms, such as a rapidly-exploring random tree (RRT) [3] and the probabilistic roadmap (PRM) [4], are known as efficient approaches for complex path planning problems, they focus mainly on the feasibility of the path with less consideration on the cost of the path. So they are not suitable for above mentioned problems. Some planning algorithms including RRT* [5], a variant of RRT, have been proposed recently considering the importance of the quality of the path [6]–[10]. They use a costmap defined over the configuration space to account for the cost a robot consumes while it navigates over the configuration space.

In this paper, we solve the optimal motion planning problem in a complex configuration space which minimizes the accumulated cost of the path. We propose a cost-aware path planning algorithm for complex configuration spaces inspired by [11], which addresses the position-dependent path planning problem (PDPP). In a PDPP problem, the cost is solely dependent on the position of a robot. The algorithm proposed in [11] constructs an RRT tree by extending the tree using cross entropy path planning [12], which optimizes the trajectory distribution using the stochastic optimization method called cross entropy (CE) [13]. In this paper, we consider a more general problem in which the cost is a function of the internal state of a robot. We refer this problem as a state dependent path planning (SDPP) problem. The proposed method improves upon RRT* by introducing nonmyopic extensions using cross entropy.

The RRT path planning algorithm, as well as RRT*, iteratively samples a random point x_{rand} over the configuration space and makes an attempt to expand a search tree $\mathcal{T}$, which is initialized with a starting point. The tree $\mathcal{T}$ is extended by connecting from $x_{nearest}$, which is the nearest point of the tree from x_{rand}, to a new point x_{new} in the direction of x_{rand}. When RRT* is applied to a complex environment for path planning, it incrementally extends its tree from local search. So it requires a large number of samples to find an optimal solution and the problem gets worse for complex terrains or higher dimensional problems. In order to solve this limitation, a number of extensions have been proposed [14]–[16]. At the core of these approaches lies the modification of the sampling procedure. Especially, Kobilarov [14] performed

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT & Future Planning (NRF-2017R1A2B2006136). A preliminary version of the proposed algorithm applied to motion planning of humanoid robots appeared in the IEEE/RSJ International Conference on Humanoid Robots, 2015 [1].

J. Suh, J. Gong, and S. Oh are with the Department of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul 08826, Korea (e-mail: junghun.suh@cpslab.snu.ac.kr; joonsig.gong@cpslab.snu.ac.kr; songhwai@snu.ac.kr).

efficient sampling using cross entropy when extending an RRT tree. However, an adaptive sampling approach cannot be the ultimate solution in a complex or high dimensional space since it still requires dense sampling.

In order to efficiently search for low-cost paths to the goal region, we need to consider the long-term effect when a node is added to an RRT search tree, such that the search tree is expanded towards the goal region along the direction of low-cost paths. However, if we simply grow the search tree towards the direction of low-cost paths, the tree can be biased and lose its optimality. To address both issues, we maintain two separate RRT trees: a standard RRT* tree $\mathcal{T}$ and an extended tree $\mathcal{T}_e$. $\mathcal{T}$ is used to determine $x_{nearest}$ of any x_{rand} for better exploration. $\mathcal{T}_e$ includes $\mathcal{T}$ and contains additional longer extensions for nonmyopic search over paths with less cost. When a new random state x_{rand} is chosen, the proposed algorithm searches for a path with the minimum cost from $x_{nearest} \in \mathcal{T}$ to x_{rand} using the cross entropy optimization. The path with the minimum cost is inserted to $\mathcal{T}_e$ and x_{new} is selected from the path. The node x_{new} is inserted to $\mathcal{T}$ and then extra nodes in $\mathcal{T}_e$ are added to $\mathcal{T}$ if they allow a low-cost path to x_{new}. By maintaining two separate trees, we can ensure unbiased exploration over the entire configuration space using $\mathcal{T}$ and, at the same time, improves the efficiency of search using long extensions in $\mathcal{T}_e$. To the best of our knowledge, it is the first approach using global tree extensions for RRT*.

We show that the proposed cost-aware path planning algorithm consistently finds low-cost paths against RRT [3] and RRT* [5] from a set of extensive simulations including physics-based simulations using the dynamic model of a two-wheel robot on complex terrains and experiments using a humanoid robot in a high dimensional configuration space.

The remainder of this paper is structured as follows. In Section II, we briefly review cross entropy path planning. A cost-aware path planning problem considered in this paper is defined in Section III and the issues with sampling-based path planning methods for complex terrains or high dimensional spaces are discussed in Section IV. The proposed cost-aware path planning algorithm is presented in Section V and analyzed in Section VI for its probabilistic completeness and asymptotic optimality. An extensive simulation study and experimental results showing the performance and characteristics of the proposed method are provided in Section VII.

A. Related Work

Recently, a number of cost-aware path planning algorithms have been proposed. Suh and Oh [11] presented a sampling-based path planning algorithm which finds a low-cost path with respect to a continuous costmap representing the environmental field. For planning a path, they used RRT and extended the search tree using a stochastic optimization method, called cross entropy (CE) [13]. In [17], a solar-intensity map is used as a costmap and a path which can charge the battery the most is found using dynamic programming. Murphy et al. [18] constructed a costmap representing traversability using aerial images and performed path planning using the A* search algorithm with heuristics to find a less riskier path. Above mentioned algorithms solve instances of PDPP since they consider an instantaneous cost at each location.

On the other hand, the accumulated cost in SDPP is determined depending on how a robot has reached the state. An energy efficient planning can be considered as an instance of SDPP, since the energy consumption at the current state depends on previous states. There are a number of studies on energy efficiency of motion planning for ground mobile robots [19]–[25] and humanoid robots [26]–[30]. In [19]–[21], energy-efficient path planning algorithms were developed for navigating on a 2D plane with no changes in elevation. They determine the velocity schedule of a moving robot by minimizing energy consumption given a predetermined path [19] or finding a path using the A* algorithm [20] or dynamic programming [21]. On the other hand, [22] and [23] considered terrains with different elevations. Sun et al. [22] examined the energy consumed by a robot along the path in terms of friction and gravity, but they did not take into account the optimal velocity profile. In [23], a vehicle velocity profile was considered to minimize the energy consumption by an electric vehicle based on dynamic programming. However, they focused on the efficiency of in-wheel motors to maximize the travel distance on a predetermined road. Kwak et al. [24] minimized the consumed energy along a path when planning on a rough terrain based on particle-RRT (pRRT) [31], which extends a tree by using particles for estimation of the distribution of states at each node of the tree under the environment with uncertainty caused by input. They fused the energy function with the likelihood of successful tree extension in pRRT. Given a costmap representing the terrain with different elevations, Jaillet et al. [25] found a low-cost path using RRT but extending the RRT tree based on the Metropolis criterion.

Unlike the approaches using a ground mobile robot, Michieli et al. [26] performed the energy analysis of a humanoid robotic arm by representing the arm as a complex energy chain of mechanical and electrical components. Kulk et al. [27] proposed a low-stiffness walk of a humanoid robot by manually tuning the parameter on each motor in the robot, through a modification in the low-level controller. They used the electric current data measured with built-in current sensors to evaluate the performance of their work. The design of an energy-efficient and human-like gait for a humanoid was presented in [28] and [29], which focused on a gait with low energy consumption. Kalakrishnan et al. [30] proposed a motion planning method based on a cost function which includes torque costs but without an experimental validation.

Recently, RRT*, which guarantees the asymptotic optimality, and its variants have been introduced [5], [14]–[16], [32]. Given a cost function, RRT* finds the optimal solution as the number of samples increases to infinity. In [33]–[35], modified RRT and RRT* were applied to manipulation tasks in high dimensional spaces. However, these algorithms focus on finding the shortest path without considering the energy consumed by a robot along the path. Hence, they are not suitable for energy-efficient motion planning.

The approach proposed in this paper improves the efficiency in a complex terrain or a high dimensional space while

ensuring the asymptotic optimality of RRT*. It has an anytime flavor in the sense that the proposed algorithm provides a near optimal solution and monotonically improves its solution towards the optimal one as more operations are allowed.

II. Cross Entropy Path Planning

Cross entropy (CE) is a sampling method designed to compute the probability of rare events [13]. It has been also extended to solve combinatorial optimization problems, such as the traveling salesman problem [36]. In [12], the cross entropy method is applied to path planning, where the optimal control law with minimum time is sought for. Algorithm 1 shows the cross entropy path planning algorithm which finds a trajectory for a robot from the start position x_{start} to the end position x_{end}. It samples a trajectory X from distribution $p(\cdot; v)$, where v is a set of controls. The algorithm first generates N random trajectories $X_1, \ldots, X_N$ from $p(X; v_i)$ considering constraints such as obstacles and vehicle dynamics over the configuration space. If $\varrho > 0$ is a small number, the algorithm selects ϱN elite trajectory samples among all trajectory samples that have less cost based on the cost function H, which is defined as:

$$H(X) = \int_0^T C(x(t))dt,$$

where $C = 1 + \beta\|x - x_{end}\|^2$ with a positive constant β and T is the termination time of the trajectory [12]. Then, it updates the parameter v_i (i.e., a set of pairs of control input and its duration) using the elite set. The algorithm iterates until the sampling distribution $p(X; v_i)$ converges to a delta distribution.

Algorithm 1 Cross Entropy Path Planning

Input: 1) Start position x_{start} and end position x_{end}
2) Number of trajectory samples N
3) Coefficients ϱ and β

Output: Shortest time path from x_{start} to x_{end}

1: $i = 0$.
2: Draw N samples $X_1, \ldots, X_N$ from $p(X; v_i)$, where $p(X; v_i)$ is a uniform distribution when $i = 0$.
3: Select ϱN trajectory samples with lower costs among all trajectory samples to the cost function H.
4: Update the parameter v_i using the elite set.
5: $i = i + 1$.
6: Repeat steps 2–5 until $p(X; v_i)$ converges to a delta function.

III. Problem Formulation

Let $\mathcal{Q}$ denote the operation region, in which a robot performs its tasks. Let $\mathcal{X} \subset \mathbb{R}^n$ be the state space of a robot, where $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ and the state $x \in \mathcal{X}$ includes the position $q \in \mathcal{Q}$ of a robot. $\mathcal{X}_{obs}$ represents the space where obstacles are placed or the robot may be in collision due to actuator bounds and $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$ is a free space. We assume that the state of the robot $x(t) \in \mathcal{X}$ is determined by

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

where $u(t) \in \mathcal{U} \subset \mathbb{R}^p$ is the control input applied at time t and f is a class C^∞ or smooth function.

Let $x_0 \in \mathcal{X}$ be the initial state of the robot and $\mathcal{X}_{goal} \subset \mathcal{X}$ be the goal region. Let $\pi_u(t)$ be the trajectory solution to the differential equation (1) for given $u(t)$ from $t = 0$ to $t = T \in \mathbb{R}^+$, where T is the termination time. The trajectory $\pi_u(t)$ can be parameterized by a series of states and control inputs $(x(t), u(t))$ and it connects the initial state and the goal region, such that $x(0) = x_0$ and $x(T) \in \mathcal{X}_{goal}$. We assume in this paper that the trajectory is deterministic given the environment and control inputs. Then we want $\pi_u(t) \in \mathcal{X}_{free}$ for all times. Given the initial state $x_0 \in \mathcal{X}$ and the goal region $\mathcal{X}_{goal} \subset \mathcal{X}$, such that $x(0) = x_0$ and $x(T) \in \mathcal{X}_{goal}$, and a cost function $\mathcal{J}$, the minimum cost path planning problem can be expressed as:

$$\begin{aligned} &\arg\min_{u(t):0\le t\le T} \quad \mathcal{J}\Big(\pi_u(t)\Big) \\ &\text{subject to } \pi_u(t) \in \mathcal{X}_{free} \text{ for all } t \in [0, T] \\ &\qquad \text{and } \pi_u(T) \in \mathcal{X}_{goal}. \end{aligned} \tag{2}$$

The classical path planning problem defines a cost function which only considers the length of a path, so the optimal solution is focused on how fast a robot can reach the goal region. However, the cost function for the minimum cost path planning problem is required to take into account the quality of a path while a robot is moving along the trajectory. This means the cost function $\mathcal{J}$ can depend on a costmap $\mathcal{C}$ which depends on the position of a robot (i.e., $\mathcal{C} : \mathcal{Q} \to \mathbb{R}$) or an energy function E which depends on the state of a robot (i.e., $E : \mathcal{X} \to \mathbb{R}$).

IV. Issues with Sampling-Based Path Planning for Complex Terrains or high dimensional spaces

A sampling-based path planning algorithm, such as RRT and PRM, has some charming properties. First of all, the probabilistic completeness can be guaranteed, meaning that the probability that the algorithm finds a solution increases to one as the number of samples grows to infinity if a solution exists. Such algorithms can also find a solution rapidly for a complex high dimensional space since they incrementally grow a graph to randomly chosen point until they reach the goal region. Moreover, the implementation of a sampling-based method is relatively simple. But, both approaches are not suitable for the problem of cost-aware path planning in a complex terrain or a high dimensional space since they focus on finding a feasible path to the goal region, which is collision-free. A variant of RRT, called RRT*, can be a candidate approach for finding a path appropriate for the mission since it returns a minimum-cost path after an enough number of iterations due to the asymptotic optimality of RRT*. However, finding the minimum-cost path in a complex terrain or a high dimensional space can be considered as an extremely rare event as the dimension of the state space increases, so it requires an extremely large number of samples to find a good solution.

We demonstrate this using a toy example with a valley shown in Figure 1. The goal region is located at a slightly

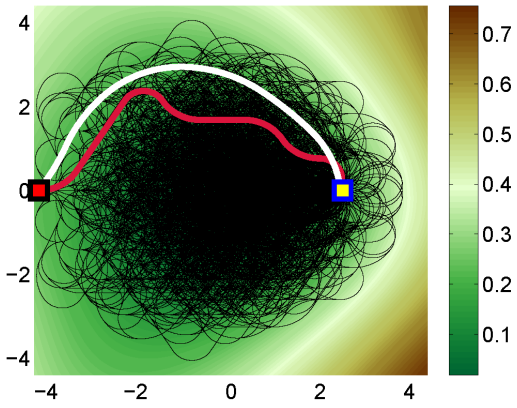


Fig. 1. A simple terrain with a valley. Different colors represent different elevations (see the colorbar). The start position and goal region are marked by red and yellow squares, respectively. Random trajectories from RRT are shown in black. The red trajectory is the minimum cost path found by RRT and the white trajectory is a path found by the proposed algorithm.

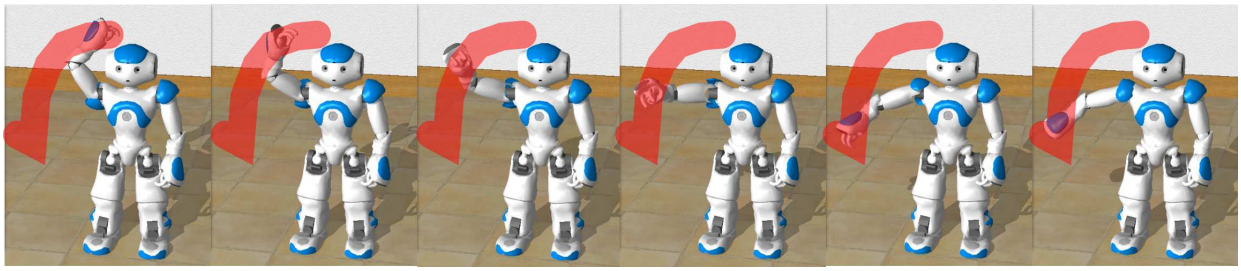
lower altitude than the start position. We assume five angular velocities, $\omega \in \{-\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}\}$, as a set of possible inputs with a constant translational velocity at an interval of 5 seconds. The duration of a trajectory is 60 seconds. The dynamic model and the energy function used in this example are described in Section VII-B. From 5^{12} randomly generated RRT trajectories, 6,038 trajectories reach the goal region (q_{rand}) and they are shown as black lines in Figure 1. The trajectory in magenta is the trajectory with the minimum cost among 6,038 trajectories. For a comparison, a path found by the tree extension step of the proposed algorithm is shown in white. The required energy for the path from the proposed algorithm is 0.3972 J (joule) and the path from RRT requires 6.9577 J, which is more than 17 times larger than a solution found by the proposed algorithm. A trajectory found by RRT* is not shown in Figure 1 since RRT* finds the same path as the proposed algorithm (the white solid line in the figure). However, it took RRT* five times longer to find the solution than the proposed algorithm. When RRT* extends an RRT tree, it selects x_{new} by steering towards x_{rand} without considering the cost of the path to x_{rand} so it extends the tree based on a simple local search. Therefore, RRT* requires an extremely large number of samples to find the minimum cost path.

We consider another toy example, in which a humanoid robot lowers one arm to place its hand at a specific position. Even if the motion is extremely easy, there is a clear difference in energy consumption depending on how to schedule the motion trajectory. Since robot arms has multiple joints and each joint represents a single state, the configuration space has higher dimensionality. We assume five joints for one arm and each joint can move within its limited angle range. When a robot takes an action, the consumed energy is determined according to the torque on each joint and the change of each joint angle. More detailed explanation about a humanoid robot and its energy function used in this work can be found in Section VII. Figure 2(a) and 2(b) show paths obtained from RRT* and the proposed algorithm, respectively, with a deadline of 500 seconds. The red thick lines shown in the snapshots represent the trajectory of an end effector. Since the torque on a single joint of the arm is affected by other joints of the arm, the consumed energy is highly dependent on the state of other joints. That is, there exist certain configurations of the arm which minimize the torque. Therefore, the robot should move while maintaining configurations with low energy consumption. The trajectory obtained from RRT* tends to go straight to the goal region while moving multiple joints simultaneously. On the other hand, the proposed algorithm finds a trajectory which moves each joint of the arm in a more energy-efficient manner. It first lowers the arm by moving the shoulder pitch angle without changing the shoulder roll angle. When the pitch angle becomes zero, i.e., the direction of the arm becomes orthogonal to the body, it moves the shoulder roll joint since this configuration requires less torque on the shoulder roll joint. After moving the shoulder roll joint, the pitch joint moves again to the goal state. The robot following the trajectory obtained from RRT* consumes 420.86 J while the proposed algorithm provides a trajectory requiring only 296.61 J. Since RRT* requires dense sampling in order to find the optimal solution due to its myopic nature, it is computationally expensive in a high dimensional space. Hence, as demonstrated in this example, for solving an energy-efficient path planning problem in high dimensional spaces, we need an efficient nonmyopic approach.

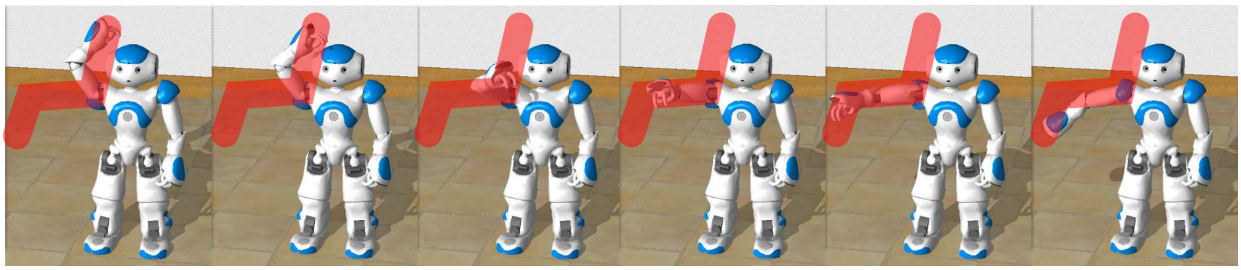
V. Cost-Aware RRT*

We now describe the proposed cost-aware path planning algorithm that generates a trajectory to minimize the accumulated cost along its path. The key idea is to sample from the configuration space and to grow a search tree (or RRT tree) by extending the tree using stochastic optimization. Unlike existing sampling based motion planning algorithms, in which the RRT tree is extended towards the random point based on the distance only, the proposed algorithm finds a path towards the random point with minimal cost in a nonmyopic manner. The proposed algorithm is based on RRT*, which guarantees the probabilistic completeness and the asymptotic optimality, and cross entropy to find cost-efficient controls from a parameterized continuous input space. Thus, the proposed algorithm is referred as cost-aware RRT* (CARRT*) in this paper. We first describe the following procedures of RRT*, which are shared by CARRT*.

- An RRT tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ contains a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$.
- $Sample(\mathcal{X})$ function returns an independent, uniformly distributed sample from $\mathcal{X}$.
- $Nearest\_Neighbor(\mathcal{T}, x)$ returns a vertex in $\mathcal{V}$ which is closest to x in terms of the Euclidean distance.
- $Steer(x, y)$ returns a configuration z in a ball centered around x closest to y, i.e. $\arg\min_z \|z-y\|$ s.t. $\|x-z\| < \rho$, where $\rho > 0$ is a predefined steering parameter.
- $Parent(\mathcal{T}, x)$ returns a unique vertex $x' \in \mathcal{V}$ such that $(x', x) \in \mathcal{E}$.
- $Near(x, r)$ returns a set of all points in $\mathcal{V}$ that are within a ball of radius $r \leq \gamma_{RRT*}(\log(n)/n)^{\frac{1}{d}}$ centered at x,



(a) RRT*



(b) The proposed algorithm

Fig. 2. A simple scenario using a humanoid robot. A robot is assigned to lower its one arm to place its hand at the specific location. Two figures (a) and (b) show the process of following trajectories obtained from RRT* and the proposed algorithm, respectively. The red line represents the trajectory of an end effector of the arm.

where γ_{RRT*} is a constant factor ensuring the optimality of the path, n is the number of all nodes in $\mathcal{V}$, and d is the state dimension.

- $CollisionFree(x, y)$ checks whether the line connecting two points x and y is placed in $\mathcal{X}_{free}$.

The overall structure of CARRT* is given in Algorithm 2. Unlike RRT*, a specifically tailored extension procedure is added to CARRT*. The algorithm constructs two RRT trees: a standard RRT tree $\mathcal{T} := (\mathcal{V}, \mathcal{E})$ and an extended tree $\mathcal{T}_e := (\mathcal{V}_e, \mathcal{E}_e)$. The standard tree $\mathcal{T}$ is used to determine the nearest point to a random point x_{rand} (line 5) for better exploration. The extended tree $\mathcal{T}_e$ includes $\mathcal{T}$, such that $\mathcal{V} \subset \mathcal{V}_e$ and $\mathcal{E} \subset \mathcal{E}_e$. $\mathcal{T}_e$ contains additional branches, which are not present in $\mathcal{T}$, and CARRT* uses $\mathcal{T}_e$ for finding low-cost long paths. The function CE_Extend described below is used for growing $\mathcal{T}_e$.

When performing an extension, if the distance between $x_{nearest}$ and x_{rand} is larger than a threshold $\eta > 0$, i.e. $\|x_{nearest} - x_{rand}\| > \eta$, CE_Extend finds a cost-aware path $\mathcal{P}^*$ from $x_{nearest}$ towards x_{rand} using cross entropy path planning (line 7), where $\eta > \rho$. A set of elite states, X_{CE}, is extracted from $\mathcal{P}^*$ based on the extension unit length ρ (line 8). Note that if we add all points in X_{CE} to $\mathcal{T}$, the tree will be biased towards x_{rand}, resulting poor exploration over the state space. This is the reason why CARRT* constructs two separate trees to tradeoff between exploration and exploitation. The first point x_{ce_1} of X_{CE} is used to plan both trees as x_{new} in function $Plan\_DoubleTrees$ (lines 9–10). All other points in X_{CE} are inserted only to $\mathcal{T}_e$ using $Plan\_SingleTree$ with rewiring (lines 11–13). When the distance between $x_{nearest}$ and x_{rand} is smaller than η, the standard steering function of RRT* is applied (line 15). Since CE_Extend is not effective when x_{near} is close to x_{rand}, the parameter η is used.

The function $Plan\_DoubleTrees$, detailed in Algorithm 3, updates $\mathcal{T}$ and $\mathcal{T}_e$ using the newly selected x_{new} and it is illustrated in Figure 3. Solid circles and thick lines represent vertices and edges of $\mathcal{T}$, respectively. Hollow circles and dash lines represent vertices and edges of $\mathcal{T}_e$, respectively. In line 2 of Algorithm 3, nodes in $\mathcal{T}_e$, except children of x_{new}, which are close to x_{new} are returned as X_{near} using $Near$ (Figure 3(b)). X_{near} are within a ball of radius $r = \gamma_{RRT*}\left(\frac{\log(n)}{n}\right)^{\frac{1}{d}}$ centered at x_{new} as explained in [5], where n is the number of vertices in $\mathcal{T}_e$. In line 3, the best parent node x_{min} of x_{new} is found based on the cost to x_{new} via x_{min} from $Choose\_Parent$ (Figure 3(c)). Here, $Cost(x)$ calculates the cost of the minimum-cost path from x_0 to x. If x_{min} is not included in $\mathcal{T}$, $Update\_Tree$ function (Algorithm 6) is called to insert all ancestor nodes of x_{min} from $\mathcal{T}_e$ to $\mathcal{T}$ (Figure 3(d)). By including nodes in $\mathcal{T}_e$ through $Update\_Tree$, CARRT* gains more chances for extending the tree in a nonmyopic manner. Lines 10–23 are for rewiring. If the rewiring condition (line 11) is satisfied for x_{near}, the parent of x_{near} is shifted to x_{new} like RRT*. However, CARRT* checks whether x_{near} is included in $\mathcal{T}$ for x_{new}. If it is not in $\mathcal{T}$, the node x_{near} and a new edge (x_{new}, x_{near}) are added to $\mathcal{T}$ (lines 13–14). On the other hand, if $x_{near} \in \mathcal{T}$, the rewiring procedure is the same as RRT* (lines 16–18). For x_{new}, $\mathcal{T}_e$ is updated by deleting an edge from the parent of x_{near} to x_{near} (line 20) and including an edge (x_{new}, x_{near}) (line 21). This rewiring case is shown in Figure 3(f). The stopping criterion of CARRT* can be the number of iterations after reaching the goal region $\mathcal{X}_{goal}$, the maximum number of iterations, or a time deadline.

A. CE_Extend

The objective of CE_Extend is to help extending a tree from $x_{nearest}$ with a low-cost path to x_{rand}. A low-cost

Algorithm 2 Cost-Aware RRT*

$\mathcal{V} \leftarrow \{x_0\}, \mathcal{E} \leftarrow \emptyset, \mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$
$\mathcal{V}_e \leftarrow \{x_0\}, \mathcal{E}_e \leftarrow \emptyset, \mathcal{T}_e \leftarrow (\mathcal{V}_e, \mathcal{E}_e)$
while stopping_criterion is false **do**
  $x_{rand} \leftarrow$ Sample($\mathcal{X}_{free}$)
  $x_{nearest} \leftarrow$ Nearest_Neighbor($\mathcal{T}, x_{rand}$)
  if $\|x_{nearest}, x_{rand}\| > \eta$ **then**
    $\mathcal{P}^* \leftarrow$ CE_Extend($x_{nearest}, x_{rand}$)
    $X_{CE} \leftarrow$ Fragment($\mathcal{P}^*, \rho$)
    $x_{new} \leftarrow x_{ce_1} \in X_{CE}$
    $[\mathcal{T}, \mathcal{T}_e] \leftarrow$ Plan_DoubleTrees($\mathcal{T}, \mathcal{T}_e, x_{nearest}, x_{new}$)
    for $\{x_{ce_i}\}_{i=2,\cdots,n} \in X_{CE}$ **do**
      $\mathcal{T}_e \leftarrow$Plan_SingleTree($\mathcal{T}_e, x_{ce_{i-1}}, x_{ce_i}$)
    end for
  else
    $x_{new} \leftarrow$Steer($x_{nearest}, x_{rand}$)
    if CollisionFree($x_{nearest}, x_{new}$) **then**
      $[\mathcal{T}, \mathcal{T}_e] \leftarrow$ Plan_DoubleTrees($\mathcal{T}, \mathcal{T}_e, x_{nearest}, x_{new}$)
    end if
  end if
end while
return $\mathcal{T}, \mathcal{T}_e$

Algorithm 3 Plan_DoubleTrees($\mathcal{T}, \mathcal{T}_e, x_{nearest}, x_{new}$)

$\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{new}\}, \mathcal{V}_e \leftarrow \mathcal{V}_e \cup \{x_{new}\}$
$X_{near} \leftarrow$ Near($\mathcal{T}_e, x_{new}$)
$x_{min} \leftarrow$Choose_Parent($X_{near}, x_{nearest}, x_{new}$)
if $x_{min} \notin \mathcal{V}$ **then**
  $\mathcal{T} \leftarrow$Update_Tree($\mathcal{T}, \mathcal{T}_e, x_{min}$)
else
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{min}, x_{new})\}$
end if
$\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{min}, x_{new})\}$
for $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
  if Cost(x_{new}) + $\mathcal{J}$(Line(x_{new}, x_{near}) < Cost(x_{near}) *AND* CollisionFree(x_{new}, x_{near}) **then**
    if $x_{near} \notin \mathcal{V}$ **then**
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{near}\}$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{new}, x_{near})\}$
    else
      $x_{parent} \leftarrow$ Parent($\mathcal{T}, x_{near}$)
      $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(x_{parent}, x_{near})\}$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{new}, x_{near})\}$
    end if
    $\mathcal{E}_e \leftarrow \mathcal{E}_e \setminus \{$(Parent($\mathcal{T}_e, x_{near}$), x_{near})$\}$
    $\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{new}, x_{near})\}$
  end if
end for
return $\mathcal{T}, \mathcal{T}_e$

Algorithm 4 Plan_SingleTree($\mathcal{T}_e, x_{nearest}, x_{new}$)

$\mathcal{V}_e \leftarrow \mathcal{V}_e \cup \{x_{new}\}$
$X_{near} \leftarrow$ Near($\mathcal{T}_e, x_{new}$)
$x_{min} \leftarrow$Choose_Parent($X_{near}, x_{nearest}, x_{new}$)
$\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{min}, x_{new})\}$
for $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
  if Cost(x_{new}) + $\mathcal{J}$(Line(x_{new}, x_{near}) < Cost(x_{near}) *AND* CollisionFree(x_{new}, x_{near}) **then**
    $\mathcal{E}_e \leftarrow \mathcal{E}_e \setminus \{$(Parent($\mathcal{T}_e, x_{near}$), x_{near})$\}$
    $\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{new}, x_{near})\}$
  end if
end for
return $\mathcal{T}_e$

Algorithm 5 Choose_Parent($X_{near}, x_{nearest}, x_{new}$)

$x_{min} \leftarrow x_{nearest}$
$c_{min} \leftarrow$ Cost($x_{nearest}$) + $\mathcal{J}$(Line($x_{nearest}, x_{new}$))
for $x_{near} \in X_{near} \setminus \{x_{nearest}\}$ **do**
  $c' \leftarrow$Cost(x_{near}) + $\mathcal{J}$(Line(x_{near}, x_{new}))
  if $c' < c_{min}$ *AND* CollisionFree(x_{near}, x_{new}) **then**
    $x_{min} \leftarrow x_{near}$, $c_{min} \leftarrow c'$
  end if
end for
return x_{min}

path is found using cross entropy path planning described in Section II. When applying cross entropy path planning, a path to x_{rand} can be generated in two different ways, similar to the approach described in [14]:

1) Motion primitives: $\{(u_1, t_1), \cdots, (u_m, t_m)\}$, where control input $u_j \in \mathcal{U}$ is applied over time duration of t_j for $j \in \{1, \ldots, m\}$.
2) Waypoint primitives: $\{(x_1, \cdots, x_m)\}$, where $x_j \in \mathcal{X}_{free}$ for $j \in \{1, \ldots, m\}$.

The approach with motion primitives aims to sample a trajectory from the space of control input and time since the cost is affected by the internal state of the robot. Hence, this approach is suitable for SDPP problem. On the other hand, the second approach with waypoint primitives is suitable for PDPP since it chooses waypoints which are representing the shape of a trajectory and the accumulated cost along the trajectory depends on the instantaneous cost at each location. The difference between two approaches is how the cost along the path is determined depending on the cost function. Characteristics of both approaches are explained in detail in Section VII.

Motion or waypoint primitives are sampled from $p(\cdot; v)$, where $p(\cdot; v) = \mathcal{N}(\cdot|\mu, \Sigma)$ and $v = (\mu, \Sigma)$. The Gaussian distribution is used for the ease of computation but other distribution from the natural exponential family can be easily applied [37]. The basic idea behind cross entropy is to optimize v using the Kullback-Leibler divergence, such that the resulting $p(\cdot; v)$ is biased towards the optimal parameter distribution. In our case, we want $p(\cdot; v)$ to become a delta function and represent a minimum cost path.

1) Motion Primitives: The initial value $v_0 = (\mu_0, \Sigma_0)$ of v is set by selecting μ_0 and Σ_0 such that the motion primitive causes the shortest path from x_{near} to x_{rand} and each control is applied at an equal time interval. Since a motion primitive contains a pair of a control input and its time duration, the initial variance is set large enough to generate samples from the space of feasible control inputs and time durations. The initial variance is $\Sigma_0 = [\sigma_u^2\ 0; 0\ \sigma_t^2]$, where σ_u and σ_t are

Algorithm 6 Update_Tree($\mathcal{T}, \mathcal{T}_e, x_{min}$)

$x' \leftarrow x_{min}$
while $x' \notin \mathcal{V}$ **do**
  $x'' \leftarrow$ Parent($\mathcal{T}_e, x'$)
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{x'\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(x'', x')\}$
  $x' \leftarrow x''$
end while
return $\mathcal{T}$

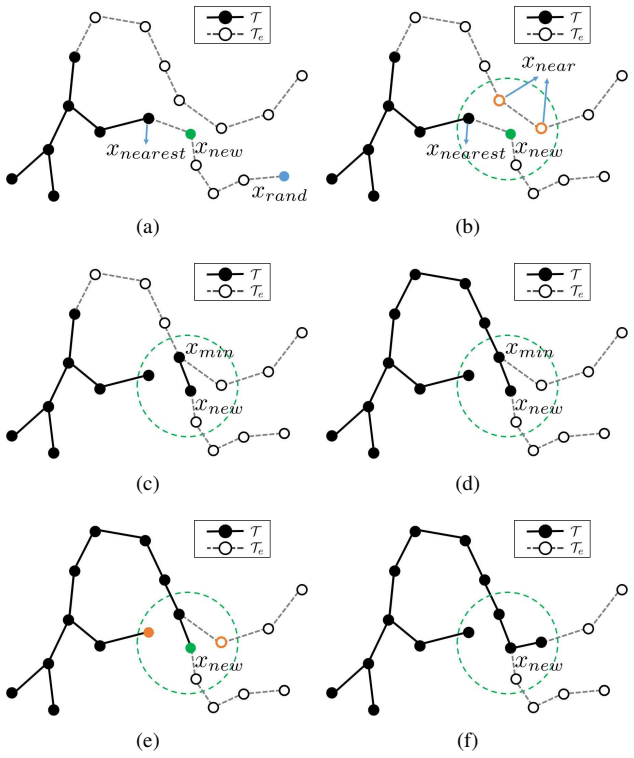


Fig. 3. An illustration of Algorithm 3. The tree $\mathcal{T}$ is represented by solid circles and thick lines while the extended tree $\mathcal{T}_e$ is represented by solid circles and thick lines with additional branches represented by hollow circles and dash lines. (a) $x_{nearest}$ is selected from $\mathcal{T}$ for x_{rand}. (b) X_{near} of x_{new} are chosen from $\mathcal{T}_e$. (c) Among X_{near}, the node with the minimum cost is selected as x_{min}. (d) Once x_{min} is determined from $\mathcal{T}_e$, the ancestor nodes of x_{min} are added to $\mathcal{T}$ through $Update\_Tree$. (e,f) For x_{near}, the rewiring procedure is performed by deleting the edge from the parent of x_{near} to x_{near} and inserting an edge (x_{new}, x_{near}) to $\mathcal{T}_e$ if x_{near} is not included in $\mathcal{T}$.

standard deviations for the control input and time duration, respectively. At the k-th iteration, we generate N_t trajectory samples $\mathcal{P}_1, \cdots, \mathcal{P}_{N_t}$ using motion primitives sampled from $p(\cdot; v_{k-1})$. In order to find a path to reach a region near x_{rand} while having the minimal accumulated cost as stated in the path planning problem (2), we define two parameters $\eta_a > 0$ and $\eta_b > 0$. We first select at most $\eta_a N_t$ trajectories whose terminal positions are close to the goal region and then select at most $\eta_b N_t$ elite trajectories with the lowest costs. After these two layers of filtering, we update $v_k = \{\mu_j^{mp}, \Sigma_j^{mp}\}_{j=1}^m$ from the selected elite samples $\pi_{el} = \{\pi_i\}_{i=1}^{\eta_b N_t}$, where

$$\mu_j^{mp} = \frac{1}{\eta_b N_t}\left[\sum_{l=1}^{\eta_b N_t} u_{lj}, \sum_{l=1}^{\eta_b N_t} t_{lj}\right]^T$$
$$\Sigma_j^{mp} = \frac{1}{\eta_b N_t}\sum_{l=1}^{\eta_b N_t}\left([u_{lj}, t_{lj}]^T - \mu_j^{mp}\right)\left([u_{lj}, t_{lj}]^T - \mu_j^{mp}\right)^T, \tag{3}$$

where (u_{lj}, t_{lj}) is the j-th motion primitive parameterized by the l-th elite sample. In order to guarantee that the update procedure always reduces the cost, we remember the best sample in the previous step and reuse the sample in the current update procedure if it has a lower cost than current elite samples. With these two layers of filtering, we can find a solution to (2) which always arrives at the goal region. The maximum number of iterations in the CE_Extend function is fixed to N_{iter} in our algorithm. After iterations, the minimum cost path $\mathcal{P}^*$ from x_{near} to x_{rand} is selected. In our implementation, a discretized version of the optimal path planning problem is used as described in [11].

2) Waypoint Primitives: The initial value v_0 of v is set such that a direct path from $x_{nearest}$ to x_{rand} is equally covered by each Gaussian component of v_0. Since the cost of a trajectory is determined based on the costmap, not the length of the path, the initial variance should be large enough to cover the entire space. Thus we have set Σ_0 to be $(x_j - x_j^*)^T \Sigma_0^{-1} (x_j - x_j^*) < 2$ for any $x_j \in \mathcal{X}$ and $1 \le j \le m$. At the k-th iteration, we sample N_t waypoints $\{X_i\}_{i=1}^{N_t}$ from $p(\cdot; v_{k-1})$, where $X_i = \{x_{ij}\}_{j=1}^m$. Assuming that there exists the parameterization function $g(\cdot)$ which gives the trajectory $\mathcal{P}(t)$ and control input $u(t)$ given two waypoints x_1 and x_2, i.e., $(\mathcal{P}(t), u(t)) = g(x_1, x_2)$, then for each X_i, we can construct a path $\mathcal{P}_i$ from x_{near} to x_{rand} by connecting x_{near} to x_{i1}, x_{ij} to $x_{i(j+1)}$ for all j, and x_{im} to x_{rand} using the function g. For the case of complex space with multiple obstacles, it is difficult to set the initial value v_0 since the connection from x_{ij} to $x_{i(j+1)}$ cannot be found easily due to obstacles. In such a case, a path obtained from RRT is used as v_0. For the initial variance, we followed the procedure of [12]. For each $\mathcal{P}_i$, we compute the accumulated cost along the path. Unlike the path using motion primitives, all paths can reach x_{rand}, so we use a single-layer filtering without considering the terminal cost. We select $\eta_b N_t$ elite samples with the lowest accumulated costs. From the selected elite samples $X_{el} = \{X_i\}_{i=1}^{\eta_b N_t}$, we update $v_k = \{\mu_j^{wp}, \Sigma_j^{wp}\}_{j=1}^m$, where

$$\mu_j^{wp} = \frac{1}{\eta_b N_t}\sum_{l=1}^{\eta_b N_t} x_{lj}$$
$$\Sigma_j^{wp} = \frac{1}{\eta_b N_t}\sum_{l=1}^{\eta_b N_t}(x_{lj} - \mu_j^{wp})(x_{lj} - \mu_j^{wp})^T. \tag{4}$$

VI. Analysis

A. Probabilistic Completeness

As shown in [3], RRT is probabilistically complete and has an exponentially fast convergence rate for the probability of

finding a solution if one exists as more vertices are added to the RRT tree. Furthermore, it is shown that its variants such as RRT* and the rapidly-exploring random graph (RRG) are also probabilistically complete [5]. Based on the analysis of those works, we extend the proof of the probabilistic completeness of CARRT*. For the proof, we first need the following terms from [38].

Definition 1. *(Local controllability). A robot is defined to be local controllable if and only if, $\forall x \in \mathcal{X}_{free}$, the set of configurations that the robot can reach within a finite time contains a ball centred at x.*

In this work, we assume that a robot with holonomic dynamics is locally controllable as explained in Definition 1. We denote the closed ball of radius ϵ centered at configuration x by $\mathcal{B}_\epsilon(x)$ and the set of all such balls by $\mathcal{B}_\epsilon$.

Definition 2. *(ϵ-reachable set). Let $\epsilon > 0$ and $x \in \mathcal{X}$ be given. The ϵ-reachable set of x, denoted by $R_\epsilon(x)$, is defined by*

$$R_\epsilon(x) = \{x' \in \mathcal{B}_\epsilon(x) |\ \text{A path } \pi \text{ from } x \text{ to } x' \text{ is entirely contained in } \mathcal{B}_\epsilon(x)\}.$$

Definition 3. *(ϵ-free feasible path). A path π is said to be ϵ-free feasible, if the minimal distance between π and the obstacle region is ϵ, i.e., for all $x \in \pi$, $\mathcal{B}_\epsilon(x) \subset \mathcal{X}_{free}$.*

In terms of the notation used in this paper, the notion of probabilistic completeness can be stated as follows.

Definition 4. *(Probabilistic completeness). Suppose that there exists an ϵ-free feasible path from the starting point to the goal region. Then a sampling based motion planning algorithm is probabilistically complete if the probability that the algorithm finds an ϵ-free feasible path from the starting point to the goal region approaches one as the number of samples goes to infinity.*

Unlike RRT and its variants, such as RRG and RRT*, which select a discrete input from a finite set of inputs, CARRT* selects a random input from a continuous input space for tree extension. Therefore, we first show that even if RRT selects an input from a continuous input space, it has the probabilistic completeness property and then show that the proposed algorithm is probabilistically complete.

Lemma 1. *Assume that there exists an input $u \in U = [u_{min}, u_{max}]$, which steers a robot from $x(t) = x \in \mathcal{X}_{free}$ to $x(t+\Delta t) = x' \in R_\epsilon(x)$ for some $\Delta t > 0$ and $\epsilon > 0$. Then, there exists $c > 0$, such that a randomly chosen input from U has the probability at least c of steering a robot from x to a point in $\mathcal{B}_\epsilon(x')$.*

Proof. See Appendix A. □

Suppose that there is an ϵ-free feasible path from the starting point x_{init} to the goal region $\mathcal{X}_{goal}$. Then there exists a sequence of inputs $u_0, \ldots, u_k$, such that $x_0 = x_{init}, x_1, \ldots, x_{k+1}$ and $x_{k+1} \in \mathcal{X}_{goal}$. We can then follow the proof of Theorem 3 in [3] to show the probabilistic completeness of RRT with inputs randomly selected from a bounded continuous set using Lemma 1.

Theorem 1. *An RRT, which selects an input from a bounded continuous input set for tree extension, is probabilistically complete.*

Theorem 2. *CARRT* is probabilistically complete.*

Proof. See Appendix B. □

B. Asymptotic Optimality

In this section, we show that a path found by CARRT* converges to the optimal solution with probability one. Depending on the distance between $x_{nearest}$ and x_{rand}, CARRT* adopts two extension procedures: one is a long extension using CE_Extend when $\|x_{nearest} - x_{rand}\| > \eta$ and the other is a standard extension based on $Steer$. Considering the asymptotic optimality of RRT* [5], we prove the asymptotic optimality of CARRT* by showing the asymptotic optimality for the case when a tree is extended using long extensions. Let $c^* = c(\pi^*)$ be the cost of an optimal path, and c_n be the cost of the minimum-cost solution returned by CARRT* at the end of the n-th iteration. We denotes ζ_d the volume of a unit ball in the d-dimensional space.

Theorem 3. *If $\gamma_{RRT^*} > (2(1 + \frac{1}{d}))^{\frac{1}{d}} (\frac{Vol(\mathcal{X}_{free})}{\zeta_d})^{\frac{1}{d}}$, then CARRT* algorithm is asymptotically optimal, that is, $\mathbb{P}(\{\lim_{n\to\infty} c_n = c^*\}) = 1$.*

Proof. See Appendix C. □

VII. Simulation and Experimental Results

In this section, we discuss results obtained from the proposed algorithm in simulation and experiments. The following planning problems are considered:

(P1) Find a trajectory which minimizes the accumulated position-dependent cost, i.e., the accumulated cost of environmental parameters, when a robot traverses over a field with environmental parameters;
(P2) Find a trajectory which minimizes the accumulated state-dependent cost, e.g., the consumed energy, when a robot traverses over a complex terrain; and
(P3) Find an energy-efficient motion plan for humanoid robots in a high dimensional space.

Note that (P1) is an instance of PDPP while (P2) and (P3) are instances of SDPP. In order to evaluate the performance of the proposed algorithm, we compared the proposed method against the standard RRT [3], RRT* [5][1], and cross entropy path planning algorithms, SCERRT* and TCERRT*, from [14]. All simulations were run on a desktop with a 3.4 GHz Intel Core i7 processor with 16 GB of memory. The proposed method was implemented both in MATLAB and C/C++[2]. For problems (P1) and (P2), all algorithms were implemented in MATLAB. For (P3), the C/C++ implementation of the

[1] While there are a number of extensions to RRT*, including [15], [16], [32], [34], [35], [39], some of them are designed to minimize the travel distance and not applicable to cost-aware navigation problems considered in this paper. In addition, there is no publicly available implementation of their work for fair comparison, hence, only RRT and RRT* are compared.

[2] The software will be made available publicly at http://cpslab.snu.ac.kr/software.

TABLE I
KEY PARAMETERS OF CARRT*.

Parameter	Variable	Range	Value
Tree extend threshold	ρ	> 0	1
CE primitive	m	> 0	6
CE extend threshold	η	$> \rho m$	6
CE elite fraction 1	η_a	[0.1, 1]	0.3
CE elite fraction 2	η_b	[0.01, 0.1]	0.1
CE trajectory sample	N_t	$\eta_b N_t > 2$	100

proposed method was compared to the C implementation of RRT*, provided by the authors of [5][3].

Parameters used by the algorithm are listed in Table I. From a number of different experiments, we have found that performance of the proposed algorithm is not largely affected by the choice of parameters. The tree extend threshold ρ does not affect to the performance of sampling-based algorithms, such as RRT and RRT*, so we have chosen the value according to the size of the state space. Since a motion primitive represents a continuous input sequence in this work, a larger value of m can provide more chance to find a solution. But we have found that $m = 6$ works well for many cases we have tried. The CE extend threshold η is used to balance between exploring the space and exploiting information from the constructed tree. Since the CE method cannot provide long-term effects if the distance between $x_{nearest}$ to x_{rand} is small, we set η to be at least ρm. In order to find a path to reach a target region while having a good quality, we use two CE elite fractions η_a and η_b, unlike the general CE method. We have found that $\eta_b = 0.1$ is highly effective and have set η_a to be larger than η_b. While we can find a better solution by conducting more rounds of sampling, we limit the number of CE rounds to $N_t = 100$ for overall efficiency of the algorithm.

SCERRT* and TCERRT* are RRT* based path planning algorithms using cross entropy. While SCERRT* samples a random point from the state space, TCERRT* samples from a parameterized trajectory space [14]. Additional parameters for SCERRT* and TCERRT* are set to the same values used in [14]. The proposed method has two versions depending on the method used to generate a path in CE_Extend and they will be referred as CARRT*(M) and CARRT*(W). CARRT*(M) uses motion primities (see Section V-A1) while CARRT*(W) uses waypoint primitives (see Section V-A2). For (P3), only CARRT*(W) is applied since a simple dynamical model is assumed in the problem.

A. Cost-Aware Navigation in 2D (P1)

Cost-aware navigation can address the problem of deploying a robot for environmental monitoring, where the environment is detrimental to the robot, for example, the high level of radioactivity, the heat from the fire, or weak wireless connectivity. In such environment, it is highly desirable to make a robot traverse in safer regions. Consider a 2D surveillance region $\mathcal{Q} = [-15, 15]^2$, in which a robot operates. $\mathcal{Q}$ consists of a free space, $\mathcal{Q}_{free}$, and a collection of obstacles, $\mathcal{Q}_{obs}$.

[3]http://sertac.scripts.mit.edu/web/Software.

1) Dynamic Model: For this problem, we use a dynamical model of a two-wheeled robot (Dubins' car). However, the proposed method can be applied to other dynamic models. Let (q_x, q_y) be the position of a robot and θ be its heading. Suppose $v(t)$ is the translational velocity and $u(t)$ is the angular velocity, then the dynamics of the vehicle is as follows:

$$\dot{q}_x(t) = v(t)\cos(\theta(t)), \quad \dot{q}_y(t) = v(t)\sin(\theta(t)), \quad \dot{\theta}(t) = u(t).$$

In simulation, v is fixed at a constant value of $2\,m/s$. A discrete-time version of the above dynamical model is used in simulation where the sampling period is set to $0.01\,s$.

2) Position Dependent Cost Function: As a robot moves from one location to another, the robot is penalized by an instantaneous cost at its current location. The cost of the entire region $\mathcal{Q}$ can be represented as a costmap $\mathcal{C} : \mathcal{Q} \to \mathbb{R}^+$. Given a costmap, we can formulate a position-dependent cost function $\mathcal{J}_{\text{PDPP}}$ as follows:

$$\mathcal{J}_{\text{PDPP}}\Big(\pi_u(t)\Big) = \left(\frac{1}{T}\int_0^T \mathcal{C}(\mathcal{P}_u(t))dt + \epsilon\int_0^T dt\right), \quad (5)$$

where $\mathcal{P}_u(t) \subset \pi_u(t)$ is a path considering only the positions of $\pi_u(t)$, such that $\mathcal{P}_u(t) \in \mathcal{Q}$. The line integral of $\mathcal{C}$ along the path of a robot is the total accumulated cost until the robot arrives at the goal region. The last term with $\epsilon > 0$ is introduced to favor a shorter trajectory for trajectories with the same accumulated cost.

3) Costmap: We assume that a costmap is defined over $\mathcal{Q}$, representing environmental parameters. Since we cannot measure the environmental parameter at every location, we use a nonparametric regression method, namely Gaussian process regression (GPR) [40], to predict the environmental parameter at a site where no measurement is made. We assume that the environmental parameter of interest, which defines the costmap $\mathcal{C}$, follows a Gaussian process. The resulting costmap is defined over the continuous space and this is different from previous approaches where a cost function is defined over a discretized space. If cost values are known at discrete sites, the same method can be applied to smooth the costmap. Hence, by applying GPR, we obtain a costmap of infinite resolution. Note that the appropriate parameters for the kernel function and the variance of the observation model have to be learned from the data before the costmap is applied.

4) Evaluation: We generated four scenarios with different costmaps from a Gaussian process. Scenarios are indexed from S_1 to S_4. We used the Gaussian process MATLAB toolbox [40] for modeling the costmap. Once the costmap is given, we estimate the value $\mathcal{C}(q)$ over $\mathcal{Q}$ through interpolation for computational efficiency. To consider obstacles, we added several obstacles placed at the same locations in all scenarios. The starting position x_0 and the goal region $\mathcal{X}_{goal}$ are randomly chosen for each scenario. Examples of scenarios used in simulation are shown in Figure 4. The left figure in Figure 4 is a scenario without obstacles and the right figure is with obstacles. The costmap is shown as contours with different colors. The high cost region is represented in white and the low cost region is represented in black. The yellow squares represent x_0 and $\mathcal{X}_{goal}$ and they are marked by letter S and

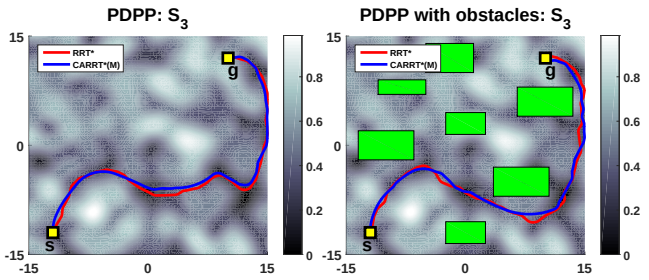


Fig. 4. An example of scenarios used in simulation and low-cost trajectories found by RRT* and the proposed algorithm for PDPP. The color over the field represent the costmap. The darker color represents low cost and the bright color represents high cost. Green rectangles are obstacles.

G, respectively. The obstacle region is represented by green rectangles.

For each scenario, we performed a total of 20 simulations for each algorithm using different pre-specified random seeds and set the time deadline to terminate the algorithm after 2000 seconds. We first compared different algorithms for scenarios without obstacles. The left figure in Figure 4 shows costmaps and the trajectories with the minimum cost found by RRT* and the proposed algorithm.

Figure 5 shows the average trajectory cost found by each algorithm as a function of the running time for 20 trials for each scenario. The error bar represents one standard deviation. Since the cost of RRT was too high, it is not included in Figure 5. Note that all algorithms find the optimal solution given enough running time. Compared algorithms find an initial solution faster than the proposed algorithm, but the initial solution found by the proposed algorithm has significantly low cost compared to other algorithms and converges to the optimal solution faster. The proposed algorithm shows excellent performance in all cases compared to other algorithms.

We also performed the same simulation but with obstacles. Trajectories found by RRT* and the proposed algorithm are shown in the right figure of Figure 4. The average trajectory costs from different algorithms are shown in Figure 6. For all cases, the proposed algorithm converges to the optimal solution faster than other algorithms.

Note that the running times of tested algorithms are relatively high. This is because when we compute the cost of a path, we have to integrate the cost along the path at the expense of additional computation time. The time required for a tree extension of the proposed method is slightly longer than RRT* and other algorithms due to its long extensions using cross entropy. However, when the distance between x_{rand} and $x_{nearest}$ becomes less than the CE extend threshold η, the general steering procedure of RRT* is applied.

Thus, the running time for the tree construction of the proposed algorithm becomes similar to RRT* and other algorithms as the number of nodes increases. For example, when $\eta = 6$, we have noticed that the tree covers the state space such that the long extension procedure no longer performs after a few x_{rand} is sampled as shown in Figure 7. In the beginning, the proportion of long extensions over all extensions is close to one. However, the standard extension becomes dominant for the tree construction after less than 100 tree extensions as shown in Figure 7(a). Furthermore, the number of vertices in a search tree increases steeply due to long extensions in the beginning but grows slowly after 100 extentions (see Figure 7(b)). While the proposed method may require more computation in the beginning, it finds a lower cost path significantly faster than compared algorithm. Figure 8 shows the computational times as a function of the number of nodes in a search tree. It shows that the computation time of the proposed method is similar to RRT* while SCERRT* and TCERRT* require significantly more computation.

For all simulations, the proposed algorithm finds a trajectory with less cost in the beginning compared to other algorithms, thanks to longer extensions for nonmyopic search used in the proposed method.

B. Complex Terrain Navigation (P2)

Since a robot is resource-constrained, it is important to perform a task with the minimum energy consumption, especially in complex terrains, such as the Mars exploration. In order to model a terrain with different elevations, we make a digital terrain model (DTM) or a height map using pairs of location and elevation values. We can display the terrain using the Gazebo simulator [41] based on the DTM. The same dynamic model explained in Section VII-A1 is used.

1) State Dependent Cost Function: Unlike the position dependent cost function used in (P1), we consider a state dependent cost function, $\mathcal{J}_{\text{SDPP}}$, which depends on the energy function $E : \mathcal{X} \to \mathbb{R}^+$. In order to measure the energy consumption by a robot along its path, we use the mechanical work (MW) criterion [25]. The MW criterion evaluates the quality of a path by regarding a nonnegative increment as a penalized cost. In other words, it is assumed that there is no cost loss for the negative slope of the time derivative of the energy function [25]. Using MW, we can define a cost function as follows:

$$\mathcal{J}_{\text{SDPP}}(\pi_u(t)) = \left(\int_0^T \mathbf{I}_{\left\{\frac{\partial E}{\partial t} > 0\right\}} \frac{\partial E(\pi_u(t))}{\partial t} dt + \epsilon \int_0^T dt \right), \tag{6}$$

where $\mathbf{I}_{\{\cdot\}}$ is the indicator function. Hence, the cost is penalized when the time derivative of the energy function has a positive slope along the path of a robot and the cost function $\mathcal{J}$ captures the penalized cost from $t = 0$ to $t = T$. The last term with ϵ plays the same role as the last term in (5).

2) Energy Function: Since we aim to minimize the energy consumption of a robot traversing a complex terrain, we can use the energy function $E : \mathcal{X} \to \mathbb{R}$ along the trajectory of the robot. Given the energy function E, the total energy consumption can be computed by integrating $\partial E/\partial t$ along the path of the robot as shown in (6). The energy at state x can be defined as follows:

$$E(x) = K(x) + P(x), \tag{7}$$

where K is the kinetic energy and P is the potential energy. Since we assume the translational velocity is constant, we

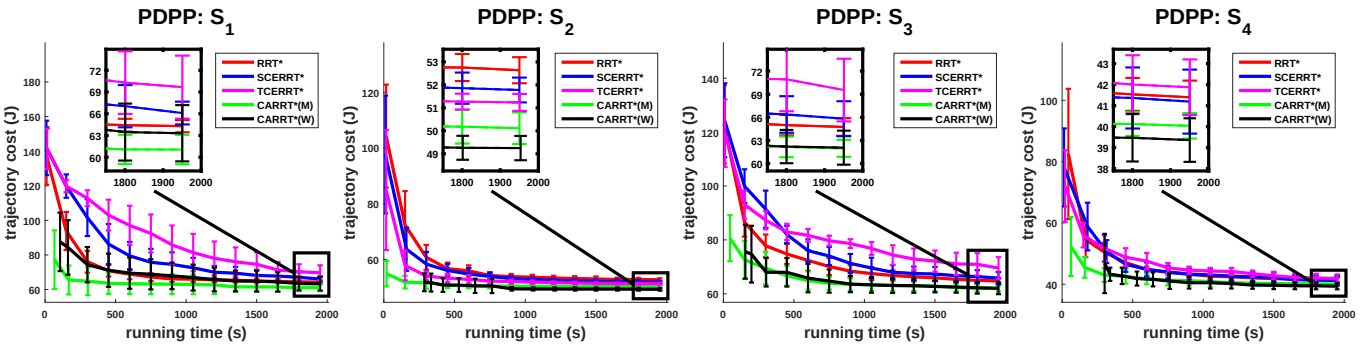


Fig. 5. The average trajectory cost as a function of deadlines for each scenario of (P1) (without obstacles). The average cost of each algorithm is computed from 20 independent trials and one standard deviation is shown as an error bar. The inset in each figure is a magnified cost graph for running times over 1800s. Legend: RRT* (red), SCERRT* (green), TCERRT* (blue), and CARRT* (magenta).

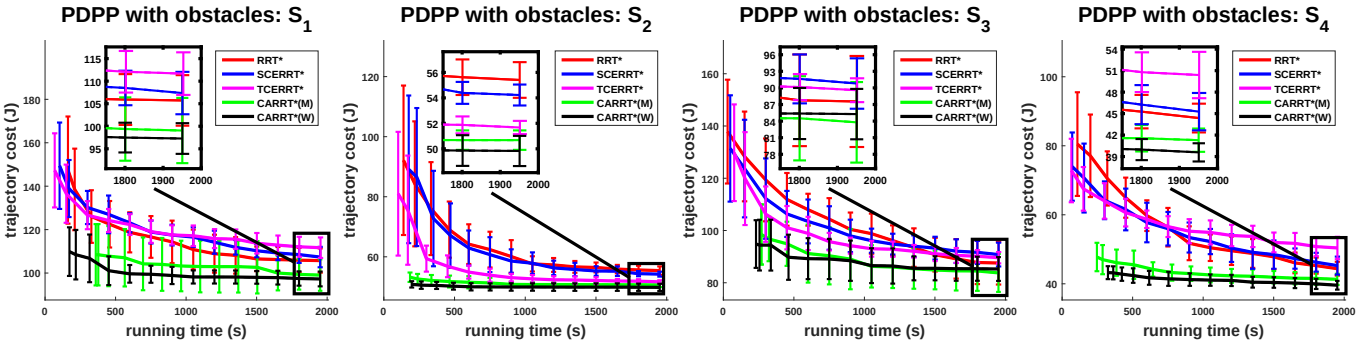
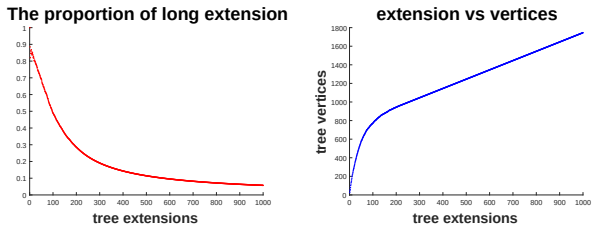


Fig. 6. The average trajectory cost as a function of deadlines for each scenario of (P1) (with obstacles). The average cost of each algorithm is computed from 20 independent trials. The inset in each figure is a magnified cost graph for running times over 1800s. Legend: RRT* (red), SCERRT* (blue), TCERRT* (magenta), and CARRT* (M) (green), CARRT* (W) (black).



(a) (b)

Fig. 7. (a) The proportion of long extensions over all extensions. (b) The number of nodes in a search tree over the number of all extensions.

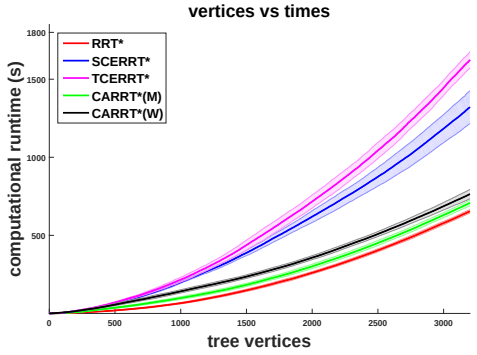


Fig. 8. Computational times over the number of nodes in a search tree. The shaded area indicates twice the standard deviation over 20 runs. Legend: RRT* (red), SCERRT* (blue), TCERRT* (magenta), and CARRT* (M) (green), CARRT* (W) (black).

ignored the kinetic energy in this paper. For the discussion below, the argument x is omitted.

The potential energy P can be defined as:

$$P = P_g + P_f + P_a, \tag{8}$$

where $P_g = m_v g h$ is the potential energy due to gravity, $P_f = \mu m_v g \cos(\phi) \Delta s$ is the potential energy for friction, and $P_a = \frac{1}{2} \rho_f S_a C_d v^2 \Delta s$ is the potential energy for the aerodynamic force. Here, m_v is the mass of the vehicle, h is the height, g is the gravitational acceleration constant, ϕ is the heading angle with respect to the ground plane, μ_f is the coefficient of friction, Δs is the moving distance at velocity v, C_d is the drag coefficient, ρ_f represents the density of the fluid, and S_a represents the front area of the vehicle.

3) Evaluation: The cost function (6) based on the energy function defined in the previous section is applied to all algorithms. All parameters used in this simulation study are set based on a Pioneer 3DX robot as follows: $m_v = 9kg$, $\mu_f = 0.001$, $\rho_f = 1.22Ng^2m^{-4}$, $S_a = 0.087m^2$, $C_d = 0.05$. We assume that there is no obstacle in the terrain but it can be easily introduced.

We generated three simple scenarios (from S_1 to S_3) to study behaviors of different algorithms. The first and second scenario has a valley and a hill, respectively, between x_0 and

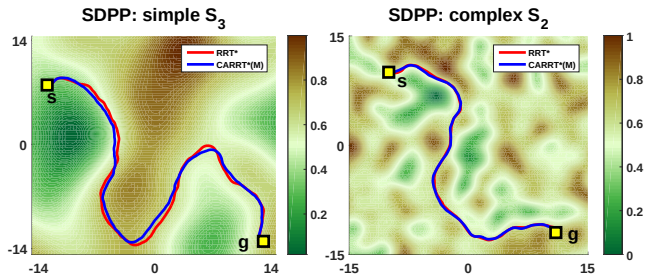


Fig. 9. Examples of simple and complex scenarios used in simulation and low-cost trajectories found by RRT* and the proposed algorithm for SDPP. The color over the field represent the costmap. The brown color represents the high altitude and the green color represents the low altitude.

$\mathcal{X}_{goal}$. The third scenario has a few valleys and hills. The left figure in Figure 9 shows a costmap and trajectories found by RRT* and the proposed algorithm. Each algorithm is run for 20 times to compute the average values. We extended the termination time to 4,000 s, compared to (P1), since it took more time to find the optimal path for these cases. Average energy consumptions for trajectories found by different algorithms are shown as a function of the running time in Figure 10. One standard deviation is shown in an error bar. The proposed algorithm shows the convergence to the optimal low-cost trajectory for all 20 runs for the given deadline of 4,000 seconds, but the deadline is short for other algorithms to converge to the optimal solution. However, given enough time, RRT* and other algorithms find the same solution that CARRT* finds as expected. The optimal trajectory tends to go around local minimum or maximum regions, so it avoids valleys and hills.

We then tested algorithms with four complex scenarios. Each scenario has multiple valleys and hills, unlike the simple scenarios. We randomly set x_0 and $\mathcal{X}_{goal}$ for all scenarios like in P1. The minimum-cost trajectories found by RRT* and the proposed algorithm are shown in the right figure of Figure 9. Figure 11 shows the average minimum energy consumption as a function of times for each algorithm. Once again, for all scenarios, the proposed algorithm shows the best result. Likewise, the proposed algorithm finds a trajectory with less cost at the beginning compared to other algorithms. Furthermore, the final cost of result from the proposed algorithm is much less than other algorithms, since the state space is more complex than the scenarios in (P1).

In order to validate trajectories obtained from simulation, we ran realistic physics-based simulations using the robot operation system (ROS) [42]. The ROS simulator provides a plugin for a differential-drive robot, such as Pioneer 3DX. The path found by a path planning algorithm is used to guide a robot in ROS and Gazebo is used to display how a robot moves in the terrain. Snapshots from the simulation are shown in Figure 12. The arrows represent a path found by an algorithm. As shown before, CARRT*(M) goes around the valley since it is more energy-efficient to move around a local valley.

C. Humanoid Robot Motion Planning (P3)

The major advantage of the proposed method is its efficiency in a high dimensional space. In this section, we apply the proposed method to the problem of efficient motion planning of a humanoid robot with high degree of freedom.

1) State Space: A humanoid robot Nao shown in Figure 13(a) is used for the experimental evaluation of CARRT*. It has a total of 25 degrees of freedom. Since we are interested in manipulating arms of a Nao and each arm has five joints, we limit the state space of Nao to five dimensional space, i.e., $\mathcal{X} \subset \mathbb{R}^5$. Figure 13(b) shows five joints. The configuration space $\mathcal{X}_{free}$ is defined within the joint angle constraints specified in Figure 13(b) and $\mathcal{X}_{obs}$ contains not only obstacles but also the torso and the head of a Nao. We also consider a ten-dimensional space for controlling both arms of a Nao.

2) Energy Function: In this section, we define the energy function suitable for the motion of a humanoid robot. Unlike a ground vehicle, the consumed energy of a humanoid robot can be obtained by computing the joint torques. The time derivative of energy function $\partial E/\partial t$ shown in (6) can be defined as follows:

$$\frac{\partial E}{\partial t} = f(x) \cdot \frac{\partial x}{\partial t}, \tag{9}$$

where $f : \mathcal{X} \rightarrow \mathbb{R}^d$ represents the torque function described in [44]. Joint torque values are obtained from the torque function using humanoid manipulator dynamics [44]. In this work, we ignore the velocity and acceleration of joints and approximate the torque function by considering only joint angles. When we measured the current flow on each joint from a robot in real experiments, the value was too noisy. So we moved arms slow enough along the given trajectory to obtain accurate measurements. Hence, the assumption barely affects the estimation of the consumed energy.

3) Evaluation: We compared the proposed algorithm against the standard RRT and RRT*. Since the general RRT uses the distance to the goal as the cost function when it extends a tree, we applied the energy function to the path obtained from RRT to compute the consumed energy. We generated two scenarios for single-arm and dual-arm manipulations: one is to position both hands to hold a box on a table by moving arms without colliding with the table and the another is to hand over an object from the right hand to the left hand by avoiding obstacles. Both scenarios are shown in Figure 15 and 16. For a single arm case, the same tasks are applied to only one arm and the other arm is kept in the goal region.

For all scenarios, we performed a total of 10 runs of each algorithm using different pre-specified random seeds and set the time deadline to terminate the algorithm as 10,000 seconds for a single-arm case and 30,000 seconds for a dual-arm case, respectively. Figure 14 shows the average cost of 10 trials, along with one standard deviation error bars. The costs of the standard RRT are too high to be included in Figure 14, where the average costs of RRT are 896.6 J and 1882.4 J for single and dual arm cases, respectively, in Scenario 1 and are 553.6 J and 1144.8 J, respectively, in Scenario 2. Even if RRT* converges to the optimal solution given enough

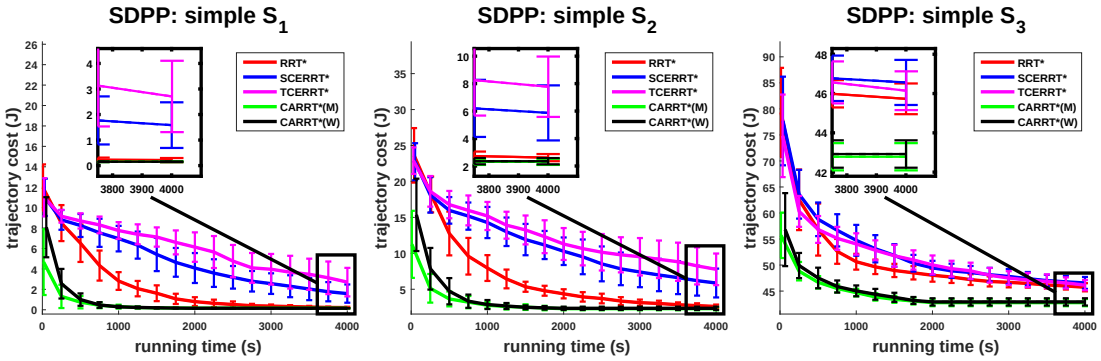


Fig. 10. The average trajectory cost as a function of times for each simple scenario of (P2). The average cost of each algorithm is computed from 20 independent trials and one standard deviation is shown as error bars. The inset in each figure is a magnified cost graph for running times over 3800s. Legend: RRT* (red), SCERRT* (blue), TCERRT* (magenta), CARRT*(M) (green) and CARRT*(W) (black).

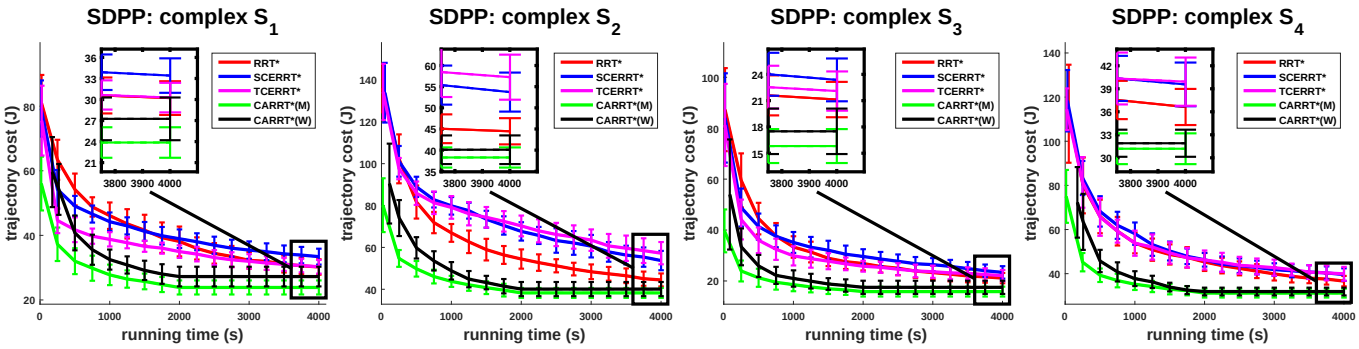


Fig. 11. The average trajectory cost as a function of times for each simple scenario of (P2). The average cost of each algorithm is computed from 20 independent trials. The inset in each figure is a magnified cost graph for running times over 3800s. Legend: RRT* (red), SCERRT* (blue), TCERRT* (magenta), CARRT*(M) (green) and CARRT*(W) (black).

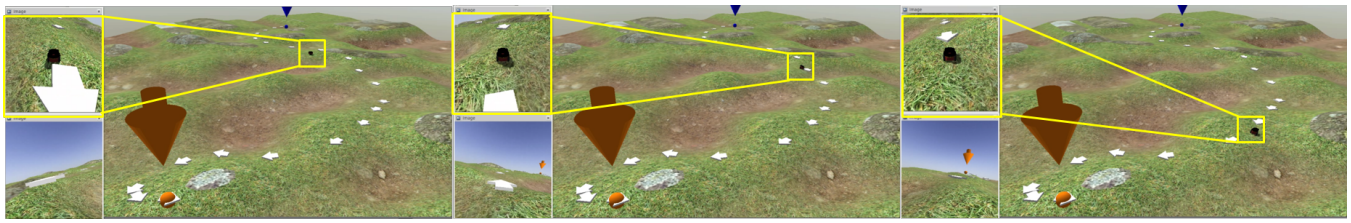
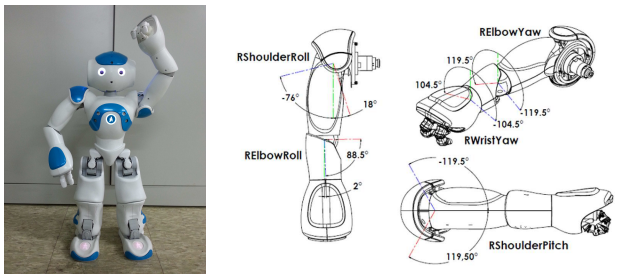


Fig. 12. A sequence of snapshots of a Pioneer 3DX robot following a given trajectory for the proposed algorithm in a ROS+Gazebo simulator. The white arrows are segments of the energy-efficient path found by CARRT*(M).

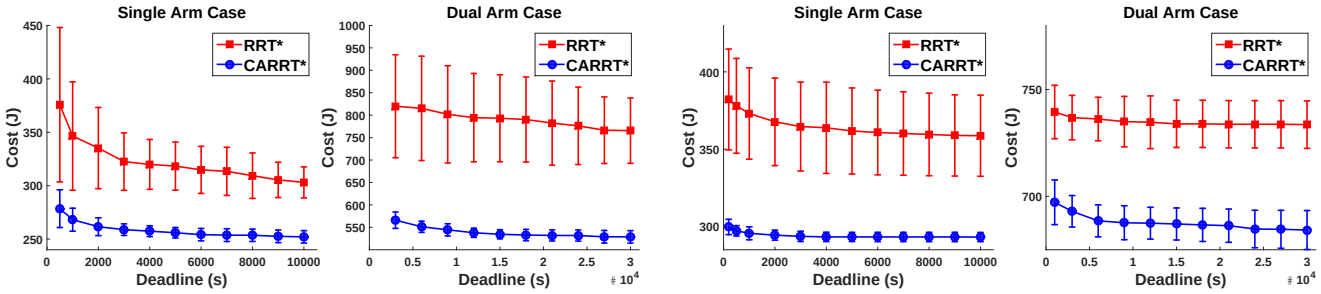


(a) (b)

Fig. 13. (a) A Nao humanoid robot from Aldebaran. (b) Joints of the right arm of Nao and joint angle constraints [43].

time, the cost of its initial solution is relatively high and the convergence rate is extremely slow. This is because RRT* requires dense sampling to improve its solution by refining the tree. The proposed algorithm finds a near-optimal solution fast with fewer samples since it extends the tree by considering the quality of the path. Thus, the proposed algorithm can find a good solution faster. For both scenarios, the proposed algorithm shows the best results.

4) Experiments: The processes of following the trajectory using a real humanoid robot for two scenarios are shown in Figure 15 and Figure 16, respectively. Each figure shows results obtained from RRT* and CARRT*, respectively, for the dual-arm case. The red lines shown in Figure 16 represent the trajectories of end-effectors (i.e., both hands). We demonstrate the results of experiments through the torque value and angle derivative as a function of time shown in Figure 17. As explained in Section IV, the proposed algorithm tends to plan a motion while maintaining a specific configuration of the



(a) Scenario 1 (b) Scenario 2

Fig. 14. The average trajectory cost (in joules) as a function of deadlines for two manipulation scenarios. The average cost of each algorithm is computed from 10 different runs with random seeds and one standard deviation is shown as error bars.

whole arm for energy efficiency. As shown in snapshots of the first column of Figure 15, two hands of the robot are located under the table. In order to place two hands in the position for holding an object, the robot should first raise two arms to its side by moving the RShoulderRoll joint. There is a big difference between two algorithms in terms of the energy consumption between $0\,s$ and $15\,s$. The energy consumption of RRT* is relatively high during this interval. RRT* immediately starts to move the RShoulderRoll joint while keeping the current joints stationary. However, the initial position of the first scenario requires more energy when it moves the RShoulderRoll joint. As shown in Figure 17(a), the torque value of RShoulderRoll is high at the beginning. Since a multiplication of the torque and the angle derivative represents the consumed energy derivative, it is more efficient to move other joints in the beginning, e.g., RShoulderPitch and RElbowRoll, to configurations with less torque values and then move the RShoulderRoll joint as less as possible, which is what CARRT* does.

The results are more distinguishable in the second scenario. While RRT* moves directly to the goal state with less consideration about energy-efficient configurations, the proposed algorithm first lifts both arms vertically and then move arms horizontally to hand over an object (see red lines shown in Figure 16 for the trajectory). It is confirmed from the torque and angle derivative values shown in Figure 17(b). Since all joints, especially RShoulderRoll, have large torque values between $0\,s$ and $18\,s$ as shown in Figure 17(b), the proposed algorithm changes only RShoulderPitch while raising arms. Then, it gradually increases angle derivatives of RSholderRoll and RElbowRoll after the torque value of those joints are reduced.

Even for longer deadlines, the proposed algorithm shows superior performance compared to RRT* as shown in Figure 18. Since dense sampling is required to find the optimal solution in RRT*, much more time is needed in a high dimensional space. Overall, the proposed algorithm tends to move joints with relatively less torque values while maintaining a specific configuration of other joints to reduce the consumed energy.

VIII. Conclusions

In this paper, we have presented a cost-effective motion planning algorithm which finds the minimum cost trajectory in complex and realistic environments. When a costmap with environmental parameters over a configuration space is available, the proposed algorithm finds a suitable trajectory of a robot with the minimum cost. Furthermore, it finds a highly energy-efficient path for traversing a complex terrain with different elevation or performing a manipulation task in a high dimensional space. The proposed method takes advantages of a sampling-based RRT* for exploration and nonmyopic tree extension using a stochastic optimization method, cross entropy (CE). From empirical evidences, including simulation and experiments using a humanoid robot, we show that the proposed algorithm can find a more cost-efficient path in a shorter time than existing algorithms.

Appendix A
Proof of Lemma 1

We first make the following usual assumptions about the dynamical system.

Assumption 1. *(Deterministic system) The dynamical system f is deterministic, i.e., it is uniquely determined for a given set of input and state.*

Assumption 2. *(Smoothness). The dynamical system f is a smooth function, such that derivatives of any order can be taken with respect to both of its arguments.*

Assumption 3. *(Lipschitz continuity). The dynamical system f is Lipschitz continuous in the following sense: There exists some constants $L_1, L_2 \in (0, \infty)$ such that*

$$\|f(x,u) - f(x',u)\| \leq L_1\|x - x'\| \text{ and} \quad (10)$$
$$\|f(x,u) - f(x,u')\| \leq L_2\|u - u'\| \quad (11)$$

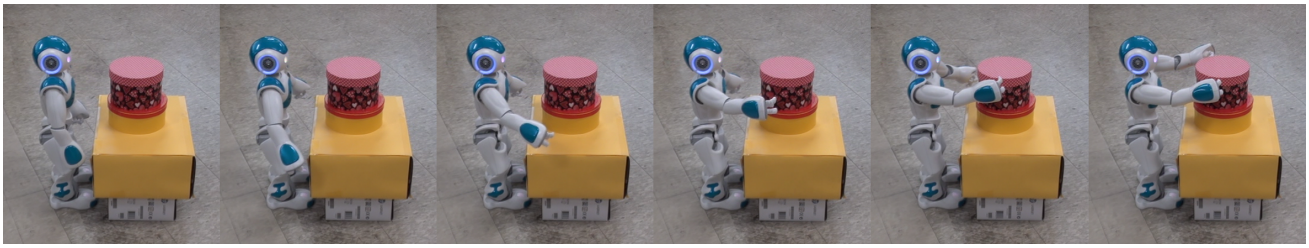
for all $x, x' \in \mathcal{X}_{free}$ and all $u, u' \in \mathcal{U}$.

Let Assumption 1 hold. If the state of the robot is x at time t, the state at time $t + \Delta t$ is uniquely determined as $x(t + \Delta t) = x(t) + \int_t^{t+\Delta t} f(x(t), u(t))dt$ for $u(t)$. Since

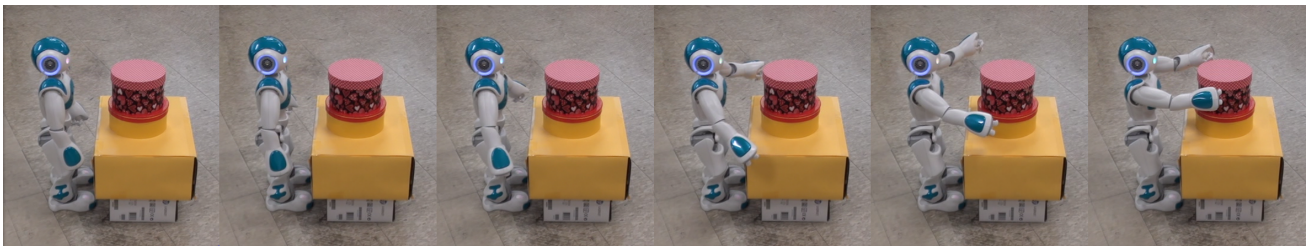
$$\dot{x} = f(x,u) = \frac{dx}{dt} \simeq \frac{x(t+\Delta t) - x(t)}{\Delta t}, \quad (12)$$

for small amount of time Δt, the solution can be written as

$$x(t + \Delta t) = x' = x + \Delta t f(x,u) + \xi_1, \quad (13)$$

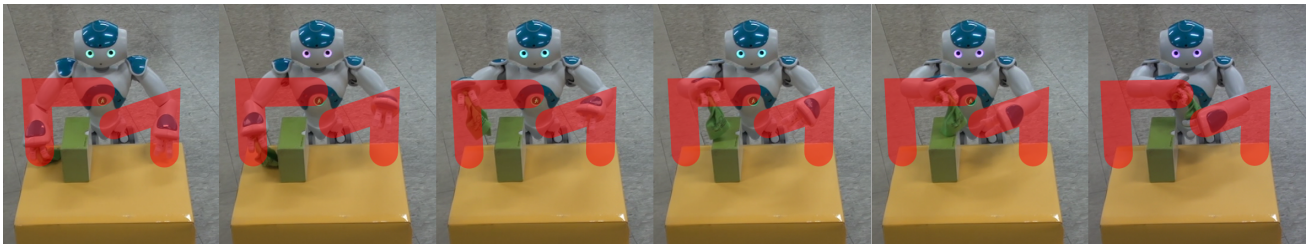


(a) RRT*

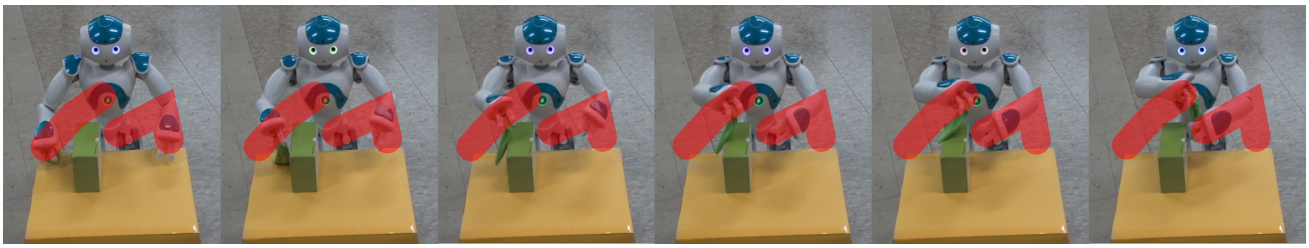


(b) CARRT*

Fig. 15. Scenario 1: a robot moves hands to hold an object on a table. (a) and (b) show the process of following a trajectory obtained from RRT* and CARRT*, respectively from left (initial pose) to right (goal pose).



(a) RRT*



(b) CARRT*

Fig. 16. Scenario 2: a robot hands over an object from the right hand to the left hand while avoiding an obstacle on a table. Red lines in both figures represent trajectories of hands.

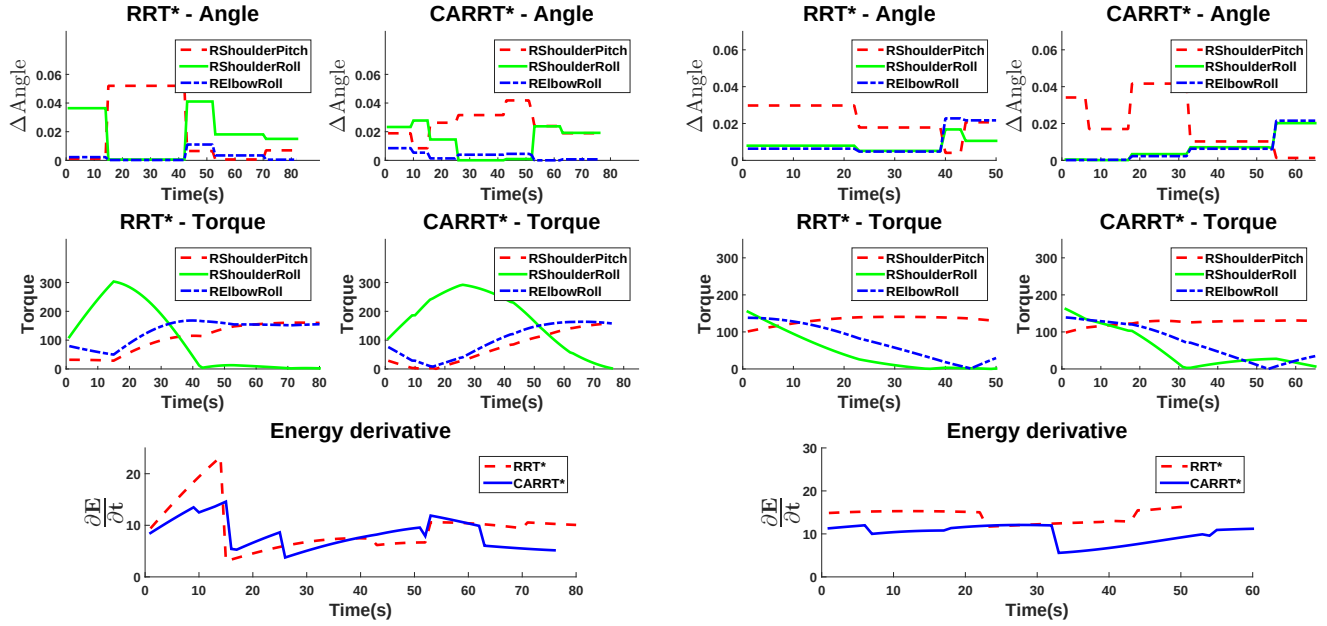
where ξ_1 is the high order term. If a perturbed input, $u+\delta$, is applied to the system for small δ, then the state z at time $t+\Delta t$ is as follows:

$$z(t+\Delta t) = x + \Delta t f(x, u+\delta) + \xi_2. \tag{14}$$

Let Assumption 2 and Assumption 3 hold. Provided that $|\xi_1 - \xi_2|$ is negligible for small Δt, we can find δ such that $\|x(t+\Delta t) - z(t+\Delta t)\| < \epsilon$ Hence, with probability at least $c = |\delta|/(u_{max} - u_{min}) > 0$, we can randomly select an input to steer the robot to $\mathcal{B}_\epsilon(x')$.

Appendix B
Proof of Theorem 2

CARRT* attempts to perform a long extension towards x_{rand} if the distance between $x_{nearest}$ and x_{rand} is larger than η. Hence, if we can show that an ordinary RRT algorithm can construct a tree with a long extension constructed by CARRT*, Theorem 1 can be applied to show the probabilistic completeness of CARRT*. Suppose that there exists an ϵ-free feasible path $\pi(t)$ from $x_{nearest} = \pi(0)$ to $x_{rand} = \pi(T)$. We first construct a set of balls of radius at least ϵ, $\mathcal{B} = \{\mathcal{B}_0, \ldots, \mathcal{B}_M\}$, covering π. The center of $\mathcal{B}_0$ is $x_{nearest}$ and the centers of two consecutive balls are exactly $\|\pi(t) - \pi(t+\Delta t)\|$ apart. Each ball $\mathcal{B}_m$ for $m \in \{1, \ldots, M\}$ has radius ϵ centered at each configuration $\pi(m\Delta t)$ for all $m \in \{1, 2, \ldots, M\}$, where



(a) Scenario 1

(b) Scenario 2

Fig. 17. The angle change and torque value on each joint as a function of time for (a) first scenario and (b) second scenario. Top figures show how much each joints moves and bottom figures show the loaded torque on each joint. The red, green, and blue line represents the values of RShoulderPitch, RShoulderRoll and RElbowRoll, respectively.

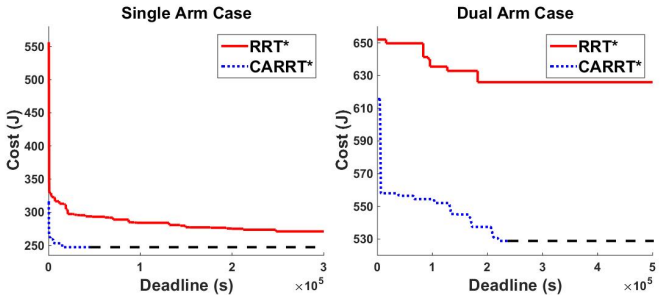


Fig. 18. Costs of paths found by RRT* and the proposed algorithm over longer time deadlines (in seconds) for one trial.

$T = M\Delta t$. Without loss of generality, we assume that all vertices in the RRT tree, except $x_{nearest}$, are at least $(1+i)\epsilon$ away from $\mathcal{B}_i$ for all i.

From Lemma 1, we know that $x_{nearest}$ can reach within $\mathcal{B}_1$ with a positive probability for a random input on some time interval $[0, \Delta t]$, Now, we extend the applied input time to $2\Delta t$. By the Lipschitz continuity of the system and the existence of an ϵ-free feasible path to the goal, there exists $\mathcal{R}_{2\epsilon}(x_{nearest})$ and $\mathcal{R}_{2\epsilon}(x_{nearest}) \cap \mathcal{B}_1$ is measurable. Hence, we can find a nonempty range of inputs to steer from a state in $\mathcal{R}_{2\epsilon}(x_{nearest}) \cap \mathcal{B}_1$ to a state in $\mathcal{B}_2$ on time interval $[\Delta t, 2\Delta t]$ using Lemma 1 again. Thus, we can sample inputs sequentially such that each new vertex added to the RRT tree falls in each ball $\mathcal{B}_m$ sequentially. Since M is finite, there is a positive probability that a long extension from $x_{nearest}$ to x_{rand} can be added by the RRT algorithm.

Appendix C
Proof of Theorem 3

The proof of the theorem is similar to that of Theorem 38 in [5] for holonomic dynamical system and Theorem 5 in [45] for nonholonomic dynamical system. We first review the asymptotic optimality condition of RRT* and show that CARRT* satisfies those conditions. In order to show the asymptotic optimality of RRT*, we begin with the construction of a sequence of paths $\{\pi_n\}$ of ϵ_n-free feasible paths from x_{init} to $\mathcal{X}_{goal}$ and show that π_n converges to the optimal path π^* as $n \to \infty$ by making a set of ball sequences $\{\mathcal{B}_n\}$, such that $\mathcal{B}_n$ covers π_n for all n [5]. Consider any sequence $\mathcal{B}_n = \{\mathcal{B}_{n,1}, \ldots, \mathcal{B}_{n,M}\}$ from $\{\mathcal{B}_n\}$. If each ball contains at least one vertex of RRT* with probability one, then a vertex x in $\mathcal{B}_{n,m}$ is connected to a vertex x' in the consecutive ball $\mathcal{B}_{n,m+1}$ for all m. This is possible due to the choice of γ_{RRT*} and the connection radius $r_n = \gamma_{RRT^*}(\frac{log(n)}{n})^{\frac{1}{d}}$. Let π'_n be a path obtained by RRT* after n iterations. It is shown that π'_n converges to π_n as n increases with probability one. Since π'_n converges to π_n and π_n converges to π^*, RRT* is asymptotically optimal. Hence, the critical condition for the asymptotic optimality is that vertices within radius r_n are connected in a cost-efficient manner.

At the end of every iteration, CARRT* grows by generating the path $\mathcal{P}^*(t)$ from $x = \mathcal{P}^*(0)$ to $x' = \mathcal{P}^*(T)$ towards x_{rand} for any random point x_{rand} like RRT*. However, since CARRT* performs a long extension for $\|x - x_{rand}\| > \eta$, $\mathcal{P}^*(t)$ is obtained by applying sequentially sampled inputs with time interval Δt from x to x' unlike RRT*. In order to insert the whole path $\mathcal{P}^*(t)$ to the tree without losing

information of $\mathcal{P}^*(t)$, we first discretize $\mathcal{P}^*(t)$ into a set of vertices $\{x_0, \ldots, x_K\}$ with time interval Δt, located at $\mathcal{P}^*(k\Delta t)$, where $x_0 = x, \ldots, x_K = x'$ and $T = K\Delta t$. Then, for a set of vertices, $\{x_1, \ldots, x_K\}$, except x_0, CARRT* regards each vertex as a new vertex added to the RRT tree and sequentially inserts x_k and an edge to x_k to the tree. For any vertex x_k inserted to tree, CARRT* performs the rewiring procedure by attempting to create an edge from the vertex x_k to its neighbors within radius $r_n = \gamma_{RRT^*}(\frac{log(n)}{n})^{\frac{1}{d}}$. Note that since CARRT* adds several vertices to the tree in a single iteration unlike RRT*, n denotes the number of vertices in the tree. On this account, we can show the existence of the connection between balls with radius r_n. Using this fact and Lemma 71 in [5], we know that each ball in $\mathcal{B}_n$ contains vertices and every subsequent balls in $\mathcal{B}_n$ are connected via vertices with probability one as n increases. It follows that $\mathbb{P}(\lim_{n\to\infty} \pi'_n = \pi^*) = 1$ due to Lemma 72 in [5]. Hence, CARRT* returns an optimal solution asymptotically as desired.

REFERENCES

[1] J. Suh, J. Gong, and S. Oh, "Energy-efficient high-dimensional motion planning for humanoids using stochastic optimization," in *Proc. of the IEEE/RSJ International Conference on Humanoid Robots (Humanoids)*, Nov. 2015.

[2] R. Bellman, Ed., *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 3, pp. 378–400, May 2001.

[4] L. E. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, Jun. 1994.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[6] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Oct. 2006.

[7] A. Ettlin and H. Bleuler, "Rough-terrain robot motion planning based on obstacleness," in *Proc. of the International Conference on Control, Automation, Robotics and Vision*, Dec. 2006.

[8] J. Lee, C. Pippin, and T. Balch, "Cost based planning with RRT in outdoor environment," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Sep. 2008.

[9] A. Tahirovic and G. Magnani, "Passivity-based model predictive control for mobile robot navigation planning in rough terrains," in *Proc. of the IEEE International Conference on Robotics and Automation*, Oct. 2010.

[10] ——, "A roughness-based RRT for mobile robot navigation planning," in *Proc. of the 18th IFAC World Congress*, Aug. 2011.

[11] J. Suh and S. Oh, "A cost-aware path planning algorithm for mobile robots," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Oct. 2012.

[12] M. Kobilarov, "Cross-entropy randomized motion planning," in *Proc. of the Robotics: Science and Systems*, Jun. 2011.

[13] P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[14] M. Kobilarov, "Cross-entropy motion planning," *International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, Jun. 2012.

[15] J. D. Gammelland, S. S. Srinivasaand, and T. D. Barfoot, "Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic," *arXiv preprint arXiv:1404.2334*, 2014.

[16] D. Kim, j. Lee, and S. Yoon, "Cloud RRT*: Sampling cloud based RRT*," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2014.

[17] P. Plonski, P. Tokekar, and V. Isler, "Energy-efficient path planning for solar-powered mobile robots," *Journal of Field Robotics*, vol. 30, no. 4, pp. 583–601, Jul. 2013.

[18] L. Murphy and P. Newman, "Risky planning on probabilistic costmaps for path planning in outdoor environments," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 445–457, Apr. 2013.

[19] Y. Mei, Y. Lu, Y. Hu, and C. Lee, "Energy-efficient motion planning for mobile robots," in *Proc. of the IEEE International Conference on Robotics and Automation*, Apr. 2004.

[20] S. Liu and D. Sun, "Optimal motion planning of a mobile robot with minimum energy consumption," in *Proc. of the IEEE International Conference on Advanced Intelligent Mechatronics*, Jul. 2011.

[21] G. Wang, M. J. Irwin, H. Fu, P. Berman, W. Zhang, and T. L. Porta, "Optimizing sensor movement planning for energy efficiency," *ACM Transactions on Sensor Networks*, vol. 7, no. 4, pp. 846–894, Feb. 2011.

[22] Z. Sun and J. H. Reif, "On finding energy-minimizing paths on terrains," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 102–114, Feb. 2005.

[23] X. Li, Y. Chen, and J. Wang, "In-wheel motor electric ground vehicle energy management strategy for maximizing the travel distance," in *American Control Conference (ACC)*, 2012.

[24] J. Kwak, M. Pivtoraiko, and R. Simmons, "Combining cost and reliability for rough terrain navigation," in *Proc. of the International Symposium on Artificial Intelligence Robotics and Automation in Space*, Feb. 2008.

[25] L. Jaillét, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, Aug. 2010.

[26] L. D. Michieli, F. Nori, A. A. Pini Prato, and G. Sandini, "Study on humanoid robot systems: An energy approach," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, Dec. 2008.

[27] J. A. Kulk and J. S. Welsh, "A low power walk for the NAO robot," in *Proc. of the Australasian Conference on Robotics and Automation*, Dec. 2008.

[28] S. H. Collins, M. Wisse, and A. Ruina, "A three-dimensional passive-dynamic walking robot with two legs and knees," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 607–615, 2001.

[29] S. H. Collins and A. Ruina, "A bipedal walking robot with efficient and human-like gait," in *Proc. of the IEEE International Conference on Robotics and Automation*, Apr. 2005.

[30] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2011.

[31] N. A. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *Proc. of the IEEE International Conference on Robotics and Automation*, Apr. 2007.

[32] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," CSAIL, MIT, Tech. Rep. MIT-CSAIL-TR-2013-021, Aug. 2013.

[33] F. Burget, A. Hornung, and M. Bennewitz, "Whole-body motion planning for manipulation of articulated objects," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2013.

[34] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011.

[35] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011.

[36] D. P. Kroese, S. Porotsky, and R. Y. Rubinstein, "The cross-entropy method for continuous multi-extremal optimization," *Methodology and Computing in Applied Probability*, vol. 8, no. 3, pp. 383–407, Nov. 2006.

[37] R. Y. Rubinstein and D. P. Kroese, Eds., *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[38] P. Švestka, "On probabilistic completeness and expected complexity of probabilistic path planning," *Technical Report UU-CS-96-20*, 1996.

[39] O. Arslan and P. Tsiotras, "Use of relaxation methods in samping-based algorithms for optimal motion planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, May 2013.

[40] C. E. Rasmussen and C. K. Williams, Eds., *Gaussian Processes for Machine Learning*. Cambridge: The MIT Press, 2006.

[41] "Gazebo." [Online]. Available: http://gazebosim.org/

[42] "ROS." [Online]. Available: http://wiki.ros.org/ROS/Introduction/

[43] "Nao Software Documentation 1.14." [Online]. Available: http://community.aldebaran-robotics.com/resources/documentation/

[44] J. Craig, Ed., *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 2005.

[45] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proceeding of the IEEE Conference on Decision and Control*, Dec. 2010.

Junghun Suh (S'12) received the B.S. degree in electrical engineering from Kwangwoon University, Seoul, Korea in 2009, and the M.S., and Ph.D. degrees in Electrical and Computer Engineering from Seoul National University, Seoul, Korea, in 2011 and 2016, respectively.

He is currently a Senior Research Engineer at LG Electronics, Osan, Korea. In 2016, he was a post-Doctoral Researcher at the Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea. His current research interests include cyber-physical systems, robotics, machine learning, optimization and their applications.

Joonsig Gong (S'12) received the B.S. and M.S. degrees in Electrical and Computer Engineering from the Seoul National University, Seoul, Korea, in 2013 and 2015, respectively.

He is currently an assistant professor in the Department of Electrical Engineering, the Republic of Korea Naval Academy, Changwon, Korea. His main research interest is motion planning of humanoid robots.

Songhwai Oh (S'04-M'07) received the B.S. (with highest honors), M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1995, 2003, and 2006, respectively.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea. Before his Ph.D. studies, he was a Senior Software Engineer at Synopsys, Inc., Mountain View, CA, USA, and a Microprocessor Design Engineer at Intel Corporation, Santa Clara, USA. In 2007, he was a Postdoctoral Researcher in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. From 2007 to 2009, he was an Assistant Professor of electrical engineering and computer science in the School of Engineering, University of California, Merced. His research interests include robotics, computer vision, cyber-physical systems, and machine learning.