Energy-Efficient High-Dimensional Motion Planning for Humanoids Using Stochastic Optimization

Junghun Suh, Joonsig Gong, and Songhwai Oh

Abstract— **This paper presents a method for planning a motion for a humanoid robot performing manipulation tasks in a high dimensional space such that the energy consumption of the robot is minimized. While sampling-based path planning algorithms, such as rapidly-exploring random tree (RRT) and its variants, have been highly effective for complex path planning problems, it is still difficult to find the minimum cost path in a high dimensional space since RRT-based algorithms extend a search tree locally, requiring a large number of samples to find a good solution. This paper presents an efficient nonmyopic motion planning algorithm for finding a minimum cost path by combining RRT* and cross entropy (CE). 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 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 efficiency while ensuring the asymptotic optimality of RRT*. We first identify and demonstrate the limitation of RRT* when it is applied to energy-efficient path planning in a high dimensional space. Results from experiments show that the proposed method consistently achieves the lowest energy consumption against other algorithms.**

I. Introduction

It is envisioned that robots will soon perform complex tasks on behalf of a human, e.g., search and rescue in hazardous environments. Since a mobile robot carries a limited energy source, it is important to plan a motion of a robot to minimize its energy consumption. This problem is universal in robotics and control and has been studied extensively as optimal control. In this paper, we consider a special case of the optimal control problem, in which we seek for a motion trajectory with the minimum energy consumption for a humanoid robot, performing a manipulation task in a high dimensional space.

Sampling-based path planning algorithms, such as a rapidly-exploring random tree (RRT) [1] and the probabilistic roadmap (PRM) [2], are known as efficient approaches for complex path planning problems. However, they focus mainly on the feasibility of the path with less consideration on the cost of the path. Recently, RRT*, which guarantees the asymptotic optimality, and its variants are introduced [3]–[5]. Given a cost function, it finds the optimal solution as the number of samples increases to infinity. Furthermore, in [6] and [7], modified RRT* are applied to manipulation tasks in high dimensional spaces.

There are a number of studies on the energy efficiency of humanoid robot motion planning. Michieli et al. [8] 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. [9] 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 [10] and [11], which focused on a gait with low energy consumption. Kalakrishnan et al. [12] proposed a motion planning method based on a cost function which includes torque costs but without an experimental validation.

In this paper, we solve the optimal motion planning problem in a high dimensional configuration space which minimizes the energy consumption. The proposed algorithm is inspired by the cost-aware path planning (CAPP) algorithm [13], which utilizes RRT and CE. The CAPP algorithm constructs an RRT tree by extending the tree using CE path planning [14], which optimizes the trajectory distribution using CE. However, since the CAPP algorithm has weaknesses in efficiency and optimality, the proposed algorithm solves those weaknesses based on RRT*. When RRT and RRT* are applied to energy-efficient path planning, they incrementally extend their trees from local search. So it requires a large amount of samples to avoid local minima and the problem gets worse for higher dimensional problems. We address this problem by constructing two RRT trees: a standard tree $\mathcal{T}$ and an extended tree $\mathcal{T}_e$. $\mathcal{T}$ is used to determine the $x_{nearest}$ of any x_{rand} for better exploration. $\mathcal{T}_e$, which includes $\mathcal{T}$, contains additional longer extensions for nonmyopic search over energy-efficient paths. When a new random state x_{rand} is chosen, the proposed algorithm searches for an energy-efficient path from $x_{nearest} \in \mathcal{T}$ to x_{rand} using CE. The most energy-efficient path is inserted to $\mathcal{T}_e$ and x_{new} is selected from the path. The node x_{new} is also inserted to $\mathcal{T}$ and extra nodes in $\mathcal{T}_e$ are added to $\mathcal{T}$ if they allow more energy efficient planning to x_{new}. By utilizing two separate trees, we can ensure unbiased exploration over the state space using $\mathcal{T}$ and, at the same time, improves the efficiency of search using nonmyopic extensions in $\mathcal{T}_e$. We show that the proposed CARRT* algorithm can efficiently schedule the motion of multiple joints of a humanoid robot Nao and achieves the lowest energy consumption. The proposed

This work was in part supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2013R1A1A2009348).

J. Suh, J. Gong, and S. Oh are with the Department of Electrical and Computer Engineering, ASRI, Seoul National University, Seoul 151-744, Korea (e-mail: {junghun.suh, joonsig.gong, songhwai.oh}@cpslab.snu.ac.kr).

algorithm is compared to RRT [1], RRT* [3] and CAPP [13].

The remainder of this paper is structured as follows. In Section II, we provide a formal description of the energy-efficient path planning problem. Section III discusses issues with sampling-based path planning methods in a high dimensional space. Section IV describes the proposed CARRT* algorithm. Results from simulations and experiments are described in Section V.

II. Energy-Efficient Path Planning

We first describe an energy-efficient path planning problem considered in this paper. Let $\mathcal{X} \subset \mathbb{R}^n$ be the state space of a robot, where $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$. We denote $\mathcal{X}_{obs}$ as the space where obstacles are placed or the robot may be in collision with obstacles due to actuator bounds and $\mathcal{X}_{free}$ as a free space where $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$. A feasible trajectory (i.e., a sequence of states) of a robot is defined as a continuous mapping function $\sigma : [0, T] \rightarrow \mathcal{X}_{free}$, where T is the terminating time of the trajectory. Then we want $\sigma \in \mathcal{X}_{free}$ for all times. Let Ω denote the space of all feasible trajectories. Given the initial state $x_0 \in \mathcal{X}_{free}$, the goal region $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$, and a cost function $\mathcal{J} : \Omega \rightarrow \mathbb{R}_{>0}$, an energy-efficient path planning problem can be expressed as:

$$\begin{aligned} &\arg\min_{\sigma \in \Omega} \quad \mathcal{J}(\sigma) \\ &\text{subject to } \sigma(0) = x_0 \text{ and } \sigma(T) \in \mathcal{X}_{goal}. \end{aligned} \tag{1}$$

The cost function for an energy-efficient path planning problem for a humanoid robot is required to take into account the total energy consumed along the path. Thus, the cost function $\mathcal{J}$ depends on the energy function $E : \mathcal{X} \rightarrow \mathbb{R}$. which is determined according to the state of a robot.

In order to measure the energy consumption by a robot along its path, we use the criterion called mechanical work (MW) [15]. The MW criterion evaluates the quality of a path by regarding a nonnegative increment as consumed energy. In other words, it is assumed that there is no energy loss when the slope of the time derivative of the energy function is negative. [15]. Using MW, we define a cost function for a humanoid robot as follows:

$$\mathcal{J}(\sigma) = \left(\int_0^T \mathbf{I}_{\left\{ \frac{\partial E}{\partial t} > 0 \right\}} \frac{\partial E(\sigma(t))}{\partial t} dt + \epsilon \right), \tag{2}$$

where $\mathbf{I}_{\{\cdot\}}$ is the indicator function. Hence, the energy is consumed 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 consumed energy along the whole path. The last term ϵ which represents the length of a path is introduced to favor a shorter trajectory [13].

III. Issues with Sampling-Based Path Planning in High Dimensional Spaces

A sampling-based path planning algorithm, such as RRT and PRM, has some charming properties such as probabilistic completeness, low computational complexity and simplicity. But, both approaches are not suitable for the problem of energy-efficient path planning in 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 an energy-efficient path 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 high dimensional space can be considered as a rare event as the dimension of the state space increases, so it requires a large number of samples to find a good solution.

We demonstrate the defect of RRT* using a 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. We verified more distinguishable difference for complex scenario through experiments (see Section V for more detail). Since robot arms has multiple joints and each joint represents a single state, the configuration space has high dimensions. 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 V. 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 the 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 (joule) 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 intractable 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.

IV. Cost-Aware RRT*

In this section, we present the proposed Cost-Aware RRT* (CARRT*) algorithm performing energy-efficient path planning suitable for high dimensional spaces. CARRT* is based on RRT*, which guarantees the probabilistic completeness and the asymptotic optimality, and the cross entropy

method to find energy-efficient controls from a parameterized continuous input space. We share the following procedures of the RRT*, which are $Sample(\cdot)$, $Nearest\_Neighbor(\cdot)$, $Steer(\cdot)$ $Parent(\cdot)$, $Near(\cdot)$, and $CollisionFree(\cdot)$.

The overall structure of CARRT* is given in Algorithm 1. 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 4) for better exploration. The extended tree $\mathcal{T}_e$ includes $\mathcal{T}$, such that $\mathcal{V} \subset \mathcal{V}_e$ and $\mathcal{E} \subset \mathcal{E}_e$. Since $\mathcal{T}_e$ contains additional branches, which are not present in $\mathcal{T}$, CARRT* uses $\mathcal{T}_e$ for efficient replanning. The function $Find\_Waypoint$ described below is used for growing $\mathcal{T}_e$. $Nearest\_Neighbor$ returns a node $x_{nearest} \in \mathcal{T}$, which is nearest to x_{rand}.

When performing an extension, if the distance between $x_{nearest}$ and x_{rand} is larger than a threshold δ, $Find\_Waypoint$ function finds a minimum-cost path $\mathcal{P}^*$ from $x_{nearest}$ to x_{rand} using CE path planning (line 7). Given $\mathcal{P}^*$, a set of waypoints, X_{CE}, is extracted along $\mathcal{P}^*$ using the extension unit length η through $Fragment$ function (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 exploration and exploitation. Points in X_{CE} are inserted to $\mathcal{T}_e$ (lines 10-13). When the distance between $x_{nearest}$ and x_{rand} is smaller than δ, the standard steering function of RRT* is applied (line 17). The function $BrowseNeighbor$ replans two trees $\mathcal{T}$ and $\mathcal{T}_e$ for the newly selected x_{new}. The stopping criterion can be the maximum number of iterations, or a time deadline.

Two critical functions of CARRT*, $Find\_Waypoint$ and $BrowseNeighbor$, are described below.

A. Find_Waypoint

The objective of $Find\_Waypoint$ is to help finding a point near $x_{nearest}$, which can lead to an energy-efficient path to x_{rand}. We find such point by optimizing the energy efficiency of a proto-path, an energy-aware path, to x_{rand}. An energy-efficient proto-path is found using cross entropy (CE), which is developed for rare event simulation [16]. We assume a proto-path is a $(d \cdot m)$-dimensional Gaussian random vector with distribution $p(\cdot; v)$, such that $p(\cdot; v) = \mathcal{N}(\cdot|\mu, \Sigma)$ and $v = (\mu, \Sigma)$ is the parameter. Here, d is the dimension of $\mathcal{X}$ and m is the length of a proto-path. 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 most energy-efficient proto-path distribution. We optimize v over multiple iterations. The initial value v_0 of v is set such that the direct path from $x_{nearest}$ to x_{rand} is equally covered by each Gaussian component of v_0. At the kth iteration, we sample N_t proto-paths $\{X_i\}_{i=1}^{N_t}$ from $p(\cdot; v_{k-1})$, where $X_i = \{x_{ij}\}_{j=1}^{m}$. For each X_i, we construct a piecewise linear 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

Algorithm 1 CARRT*

1: $\mathcal{V} \leftarrow \{x_0\}, \mathcal{E} \leftarrow \emptyset, \mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$
2: $\mathcal{V}_e \leftarrow \{x_0\}, \mathcal{E}_e \leftarrow \emptyset, \mathcal{T}_e \leftarrow (\mathcal{V}_e, \mathcal{E}_e)$
3: **while** stopping_criterion is false **do**
4: $x_{rand} \leftarrow$ Sample($\mathcal{X}_{free}$)
5: $x_{nearest} \leftarrow$ Nearest_Neighbor($\mathcal{T}, x_{rand}$)
6: **if** $\|(x_{nearest}, x_{rand}\| > \delta$ **then**
7: $\mathcal{P}^* \leftarrow$ Find_Waypoint($x_{nearest}, x_{rand}$)
8: $X_{CE} \leftarrow$ Fragment($\mathcal{P}^*, \eta$)
9: $x_{CE_0} \leftarrow x_{nearest}$
10: **for** $\{x_{CE_i}\}_{i=1,\cdots,n} \in X_{CE}$ **do**
11: $\mathcal{V}_e \leftarrow \mathcal{V}_e \cup \{x_{CE_i}\}$
12: $\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{CE_{i-1}}, x_{CE_i})\}$
13: **end for**
14: $x_{new} \leftarrow x_{CE_1}$
15: BrowseNeighbor($\mathcal{T}, \mathcal{T}_e, x_{nearest}, x_{new}$)
16: **else**
17: $x_{new} \leftarrow$Steer($x_{nearest}, x_{rand}$)
18: BrowseNeighbor($\mathcal{T}, \mathcal{T}_e, x_{nearest}, x_{new}$)
19: **end if**
20: **end while**
21: **return** $\mathcal{T}, \mathcal{T}_e$

Algorithm 2 BrowseNeighbor

1: **if** CollisionFree($x_{nearest}, x_{new}$) **then**
2: $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{new}\}$
3: $X_{near} \leftarrow$ Near($\mathcal{T}_e, x_{new}$), $x_{min} \leftarrow x_{nearest}$
4: $c_{min} \leftarrow$ Cost($x_{nearest}$) + $\mathcal{J}$(Line($x_{nearest}, x_{new}$))
5: **for** $x_{near} \in X_{near} \setminus \{x_{nearest}\}$ **do**
6: $c' \leftarrow$Cost(x_{near}) + $\mathcal{J}$(Line(x_{near}, x_{new}))
7: **if** $c' < c_{min}$ *AND* CollisionFree(x_{near}, x_{new}) **then**
8: $x_{min} \leftarrow x_{near}$, $c_{min} \leftarrow c'$
9: **end if**
10: **end for**
11: **if** $x_{min} \notin \mathcal{V}$ **then**
12: $\mathcal{T} \leftarrow$Update_Tree($\mathcal{T}, \mathcal{T}_e, x_{min}$)
13: **end if**
14: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{min}, x_{new})\}$
15: **for** $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
16: **if** Cost(x_{new})+$\mathcal{J}$(Line(x_{new}, x_{near}) <Cost(x_{near}) *AND* CollisionFree(x_{new}, x_{near}) **then**
17: **if** $x_{near} \notin \mathcal{V}$ **then**
18: $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{near}\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{new}, x_{near})\}$
19: $\mathcal{E}_e \leftarrow \mathcal{E}_e \setminus \{$(Parent($\mathcal{T}_e, x_{near}$), x_{near})$\}$
20: $\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(x_{new}, x_{near})\}$
21: **else**
22: $x_{parent} \leftarrow$ Parent($\mathcal{T}, x_{near}$)
23: $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(x_{parent}, x_{near})\}$
24: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x_{new}, x_{near})\}$
25: **end if**
26: **end if**
27: **end for**
28: **end if**

Algorithm 3 Update_Tree

1: $x' \leftarrow x_{min}$
2: **while** $x' \notin \mathcal{V}$ **do**
3: $\quad x'' \leftarrow \text{Parent}(\mathcal{T}_e, x')$
4: $\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{x'\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(x'', x')\}$
5: $\quad x' \leftarrow x''$
6: **end while**

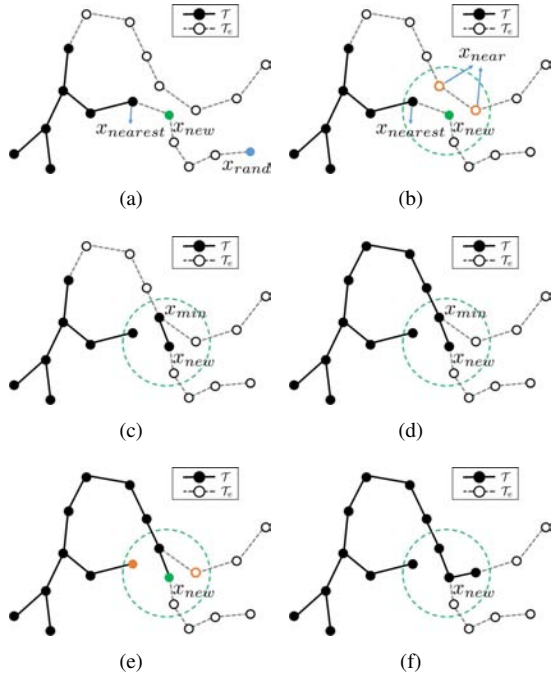


Fig. 1. An illustration of Algorithm 2. The tree $\mathcal{T}$ is represented by solid circles and thick lines while the extended tree $\mathcal{T}_e$ is represented by solid circles and thick lines with additional branches represented by hollow circles and dash lines. (a) $x_{nearest}$ is selected from $\mathcal{T}$ 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 $\mathcal{T}$.

x_{rand}. For each $\mathcal{P}_i$, we compute its energy consumption by summing up costs of its linear segments using (2) assuming a constant speed model. Then, we select η_a elite samples, i.e., η_a samples with the lowest energy consumptions. From the selected elite samples $X_{el} = \{X_i\}_{i=1}^{\eta_a}$, we update $v_k = \{\mu_j, \Sigma_j\}_{j=1}^{m}$, where

$$\mu_j = \frac{1}{\eta_a}\sum_{l=1}^{\eta_a} x_{lj}, \Sigma_j = \frac{1}{\eta_a}\sum_{l=1}^{\eta_a}(x_{lj}-\mu_j)(x_{lj}-\mu_j)^T. \quad (3)$$

After a fixed number of iterations, we terminate and the most energy-efficient piece-wise linear path $\mathcal{P}^*$ is selected.

B. BrowseNeighbor

The function $BrowseNeighbor$, detailed in Algorithm 2, updates $\mathcal{T}$ and $\mathcal{T}_e$ using x_{new} and it is illustrated in Figure 1. Solid circles and thick lines represent vertices and edges of $\mathcal{T}$, respectively. Hollow circles and dash lines represent vertices and edges of $\mathcal{T}_e$, which are not included in $\mathcal{T}$, respectively. In line 2, x_{new} is added to $\mathcal{V}$. In line 3, nodes in $\mathcal{T}_e$, except children of x_{new}, that are close to x_{new} are returned as X_{near} using $Near$ (Figure 1(b)). X_{near} are within a ball of radius $r \propto \left(\frac{\log(n)}{n}\right)^{\frac{1}{d}}$ centered at x_{new} as explained in [3], where n is the number of vertices in $\mathcal{T}_e$ and d is the state dimension. In lines 5–10, the best parent node x_{min} of x_{new} is found based on the cost to x_{new} via x_{min} (Figure 1(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 3) is called to insert all ancestor nodes of x_{min} from $\mathcal{T}_e$ to $\mathcal{T}$ (Figure 1(d)). By including nodes in $\mathcal{T}_e$ through $Update\_Tree$, CARRT* gains more chances for extending the tree in a nonmyopic manner. Lines 15–28 are for rewiring. If the rewiring condition (line 16) 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 $\mathcal{T}$. If it is not in $\mathcal{T}$, the node x_{near} and a new edge (x_{new}, x_{near}) are added to $\mathcal{T}$. Moreover, $\mathcal{T}_e$ 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 1(f). If $x_{near} \in \mathcal{T}$, the rewiring procedure is the same as RRT* (lines 23–25).

The proposed CARRT* algorithm efficiently minimizes the energy consumption by selecting a good next move to extend the RRT tree and replanning the RRT tree based on nonmypotic waypoints. As a result, CARRT* can mitigate the disadvantage of sampling-based methods and find an energy-efficient path more rapidly.

V. Simulation and Experimental Results

A. State Space of a Humanoid Robot

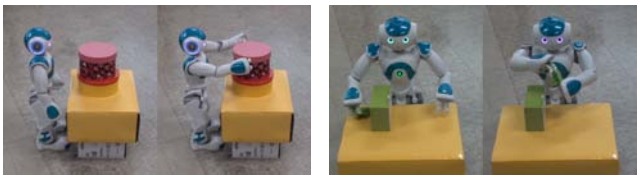
A humanoid robot Nao is used for the experimental evaluation of CARRT*. It has a total of 25 degrees of freedom. Since we are interested in manipulating arms of a Nao and each arm has five joints, we limit the state space of Nao to five dimensional space, i.e., $\mathcal{X} \subset \mathbb{R}^5$. The configuration space $\mathcal{X}_{free}$ is defined within the joint angle constraints 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.

B. Energy Function for 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 (2) can be defined as follows:

$$\frac{\partial E}{\partial t} = f(x) \cdot \frac{\partial x}{\partial t}, \quad (4)$$

where $f : \mathcal{X} \rightarrow \mathbb{R}^d$ represents the torque function described in [17]. Joint torque values are obtained from the torque function using humanoid manipulator dynamics [17]. 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.



(a) (b)

Fig. 2. The manipulation environments of two scenarios (a) and (b) for two arms, respectively. Left and right of each figure represent start and goal configurations.

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.

C. Evaluation

We compared the proposed algorithm against the standard RRT , RRT*[1], and CAPP [13]. All simulations were run on a desktop with a 3.4 GHz Intel Core i7 processor with 20 GB of memory. The proposed algorithm was implemented in C/C++. The implementation of RRT* is based on the C code provided by authors of [3].[2] For all algorithms, the energy function defined in Section V-B is used.

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 2(a) and 2(b). 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 simulations 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. Table I shows the average cost of 10 trials, along with the standard deviation error for each algorithm. Even if RRT* converges to the optimal solution given enough time, the cost of its solution at the fixed final time is relatively high and the converge rate is extremely slow. This is because RRT* requires dense sampling to improve the path by refining the tree. Compared to CAPP, the proposed algorithm finds the near-optimal solution fast with fewer samples since it extends the tree using waypoints found 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.

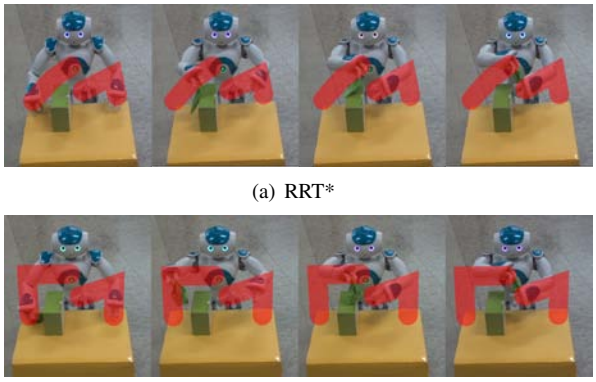
D. Experiments

The processes of following the trajectory using a real humanoid robot for the second scenario is shown in Figure 3. This figure shows results obtained from RRT* and CARRT*, respectively, for the dual-arm case. The red lines 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 4. As explained in Section III, the proposed algorithm tends to plan a motion while maintaining a specific configuration of the whole arm for energy efficiency. As shown in Figure 2(a), two hands of the robot are initially 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 4, 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,

TABLE I

COMPARISON OF ENERGY CONSUMPTION FOR DIFFERENT SCENARIOS.

Algorithm	Scenario 1			
	Single Arm		Dual Arm	
	Cost (J)	$Std.$	Cost (J)	$Std.$
RRT	896.60	321.56	1882.43	254.64
RRT*	303.07	14.53	765.45	72.78
CAPP	265.23	11.45	594.95	6.62
Proposed	**252.05**	**5.81**	**528.70**	**13.85**
Algorithm	Scenario 2			
	Single Arm		Dual Arm	
	Cost (J)	$Std.$	Cost (J)	$Std.$
RRT	553.64	36.67	1144.77	92.32
RRT*	358.71	26.20	733.57	3.63
CAPP	312.75	8.58	717.33	20.76
Proposed	**293.21**	**2.96**	**684.23**	**9.11**



(a) RRT*

(b) CARRT*

Fig. 3. (a) and (b) show the process of following a trajectory obtained from RRT* and CARRT*, respectively, for Scenario 2. Red lines in both figures represent trajectories of hands.

[1]While there are a number of extensions to RRT*, including [4]–[7], they are mainly focused on minimizing the path length. In addition, there is no available implementation of their work for fair comparison, hence, we only compared to RRT and RRT*.

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

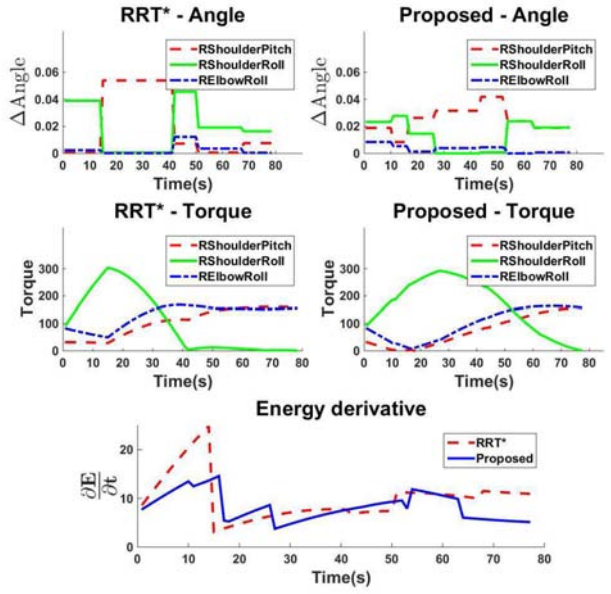


Fig. 4. The angle change and torque value on each joint as a function of time for first scenario. Top figures show how much each joints moves and bottom figures show the loaded torque on each joint. The red, green, and blue line represents the values of RShoulderPitch, RShoulderRoll and RElbowRoll, respectively.

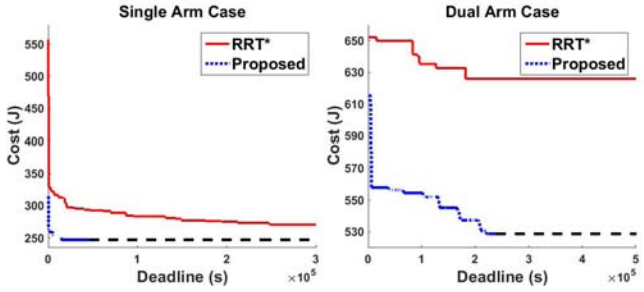


Fig. 5. Costs of paths found by RRT* and the proposed algorithm over longer time deadlines (in seconds) for one trial.

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 3 for the trajectory). It is confirmed from the torque and angle derivative values for second scenario.

Even for longer deadlines, the proposed algorithm shows superior performance compared to RRT* as shown in Figure 5. 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 for relatively less torque values while maintaining a specific configuration of other joints to reduce the consumed energy.

VI. Conclusions

In this paper, we have presented an energy-efficient path planning algorithm which finds a highly energy-efficient path for planning a motion in a high dimensional space. The energy consumption is computed considering the internal state of the robot, since consumed energy of the moving joint is determined according to the states of other joints. When the proposed algorithm plan to perform a manipulation task, it maintains a specific configuration which requires less energy consumption. The proposed method takes advantages of a sampling-based RRT for exploration and nonmyopic tree extension using cross entropy (CE). In simulations and experiments using a humanoid robot, the proposed algorithm finds more energy-efficient paths compared to RRT* in a shorter time.

References

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

[2] 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*, San Diego, CA, June 1994.

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

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

[5] 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.

[6] 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*, San Francisco, USA, Sept. 2011.

[7] 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*, San Francisco, USA, Sept. 2011.

[8] 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*, 2008.

[9] 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*, 2008.

[10] 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.

[11] 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*, 2005.

[12] 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*, 2011.

[13] 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*, Villamoura, Portugal, Oct. 2012.

[14] M. Kobilarov, "Cross-entropy randomized motion planning," in *Proc. of the Robotics: Science and Systems*, Los Angeles, USA, June 2011.

[15] 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.

[16] 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.

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