The Implementation of RRTs for a Remote-Controlled Mobile Robot

Chi-Won Roh*, Woo-Sub Lee**, Sung-Chul Kang*** and Kwang-Won Lee****

* Intelligent Robotics Research Center, KIST, Seoul, Korea
(Tel : +82-2-958-6816; E-mail: cwroh@kist.re.kr)
** Intelligent Robotics Research Center, KIST, Seoul, Korea
(Tel : +82-2-958-6724; E-mail: robot@kist.re.kr)
*** Intelligent Robotics Research Center, KIST, Seoul, Korea
(Tel : +82-2-958-5589; E-mail: kasch@kist.re.kr)
**** Department of Electronics Engineering, Ajou University, Suwon, Korea
(Tel : +82-031-219-2480; E-mail: lkw@ajou.ac.kr)

Abstract: The original RRT is iteratively expanded by applying control inputs that drive the system slightly toward randomly-selected states, as opposed to requiring point-to-point convergence, as in the probabilistic roadmap approach. It is generally known that the performance of RRTs can be improved depending on the selection of the metrics in choosing the nearest vertex and bias techniques in choosing random states. We designed a path planning algorithm based on the RRT method for a remote-controlled mobile robot. First, we considered a bias technique that is goal-biased Gaussian random distribution along the command directions. Secondly, we selected the metric based on a weighted Euclidean distance of random states and a weighted distance from the goal region. It can save the effort to explore the unnecessary regions and help the mobile robot to find a feasible trajectory as fast as possible. Finally, the constraints of the actuator should be considered to apply the algorithm to physical mobile robots, so we select control inputs distributed with commanded inputs and constrained by the maximum rate of input change instead of random inputs. Simulation results demonstrate that the proposed algorithm is significantly more efficient for planning than a basic RRT planner. It reduces the computational time needed to find a feasible trajectory and can be practically implemented in a remote-controlled mobile robot.

Keywords: RRT, Path Planning, Obstacle Avoidance, Mobile Robot, Bias Technique, Metric Choice

1. INTRODUCTION

The increasing performance of microprocessor and development of various sensors enable mobile robots to be used for various areas. In indoor cases, personal service robots perform the missions of museum tour guide, room cleaning and nursing for the elderly. In outdoor cases, mobile robots have been used for the purpose of patrol, reconnaissance, surveillance and exploring planets, etc. The remote-controlled mobile robots that are operated by the remote operators should be able to navigate with commanded translational and rotational velocities and perceive static and varying environments and avoid dynamic obstacles such as vehicles or pedestrians.

Mobile robots often find themselves in a situation where they must find a trajectory to another position in their environments, subject to constraints posed by obstacles and the capabilities of the robot itself. Rapidly-exploring Random Trees (RRTs) are a recently developed representation on which fast continuous domain path planners can be based and a randomized data structure that is designed for a broad class of a path planning problems [1-3]. The advantage of RRTs is that they can be directly applied to nonholonomic and kinodynamic planning. Moreover, RRTs has been used to solve nonlinear control problems and extended to the case of hybrid systems [4-5].

The original RRT is iteratively expanded by applying control inputs that drive the system slightly toward randomly-selected states, as opposed to requiring point-to-point convergence, as in the probabilistic roadmap approach. It is generally known that the performance of RRTs can be improved depending on the selection of the metrics in choosing the nearest vertex and bias techniques in choosing random states [6].

We designed a path planning algorithm based on the RRT method for a remote-controlled mobile robot. First, we considered a bias technique that is goal-biased Gaussian random distribution along the command directions. Secondly, we selected the metric based on a weighted Euclidean distance of random states and a weighted distance from the goal region. It can save the effort to explore the unnecessary regions and help the mobile robot to find a feasible trajectory as fast as possible. Finally, the constraints of the actuator should be considered to apply the algorithm to physical mobile robots, so we select control inputs distributed with commanded inputs and constrained by the maximum rate of input change instead of random inputs. Simulation results demonstrate that the proposed algorithm is significantly more efficient for planning than a basic RRT planner. It reduces the computational time needed to find a feasible trajectory and can be practically implemented in a remote-controlled mobile robot.

The remainder of this paper is organized as follows. Section 2 introduces a brief formulation of the RRT algorithm. Section 3 considers the kinematics and practical implementation of a RRT for a remote-controlled mobile robot. Section 4 presents the simulation results. Finally, some conclusions are presented in Section 5.

2. RRT ALGORITHM

2.1 Basic RRT Algorithm

2.1.1 Problem Description

The class of problems considered in RRTs can be formulated in terms of six components [3]:

1) State Space: A bounded manifold, $X \subset R^n$
2) Boundary Values: $x_{init} \in X$ and $X_{goal} \subset X$
3) Collision Detector: A function, $D: X \to \{true, false\}$, that determines whether global constraints are satisfied from state x. This could alternatively be a real-valued function that indicates distance from the constraint

boundary.

4) Inputs: A set, U, which specifies the complete set of controls or actions that can affect the state.

5) Incremental Simulator: Given the current state, $x(t)$, and inputs applied over a time interval, $\{u(t') \mid t \le t' \le t + \Delta t\}$, the incremental simulator yields $x(t+\Delta t)$. This usually occurs through numerical integration of a state transition equation, $\dot{x} = f(x,u)$.

6) Metric: A real-valued function, $\rho : X \times X \rightarrow [0,\infty)$ which specifies the distance between pairs of points in X (however, ρ is not necessarily symmetric).

Trajectory planning will generally be viewed as a search in a state space, X, for a control u that brings the system from an initial state, x_{init} to a goal region $X_{goal} \subset X$ or goal state $X_{goal} \in X$. It is assumed that a complicated set of global constraints is imposed on X, and any solution path must keep the state within this set. A collision detector reports whether a given state x satisfies the global constraints. Local, differential constraints are imposed through the definition of a set of inputs (or controls) and an incremental simulator. Taken together, these two components specify possible changes in state. The incremental simulator can be defined by numerical integration of a state transition equation of the form $\dot{x} = f(x,u)$ or can simply be achieved by a simulation software package. Finally, a metric is defined to indicate the closeness of pairs of points in the state space.

2.1.2 Basic RRT Algorithm

The basic RRT construction algorithm is given in Table 1 [1]. A simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected state, $x_{rand} \in X$. The EXTEND function selects the nearest vertex already in the RRT to x. The "nearest" vertex is chosen according to the metric, ρ. The function NEW_STATE makes a motion toward x by applying an input $u \in U$ for some time increment Δt. This input can be chosen at random, or selected by trying to all possible inputs and choosing the one that yields a new state as close as possible to the sample, x_{rand} (if U is infinite, then a finite approximation or analytical technique can be used). NEW_STATE implicitly uses the collision detection function to determine whether the new state and all intermediate states satisfy the global constraints. For many problems, this can be performed quickly ("almost constant time") using incremental distance computation algorithms by storing the relevant invariants with each of the RRT vertices. If NEW_STATE is successful, the new state and input are represented in x_{new} and u_{new}, respectively. Figure 1 shows an RRT grown from the center of a square region in the plane. In this example, there are no differential constraints (motion in any direction is possible from any point). The incremental construction method biases the RRT to rapidly explore in the beginning, and then converge to a uniform coverage of the space. The exploration is naturally biased towards vertices that have larger Voronoi regions. This causes the exploration to occur mostly on the unexplored portion of the state space.

In addition to growing a tree from the starting state, many RRT implementations grow a second tree from the goal tree. Such trees grow in four steps.

1) Grow start-tree towards a random unexplored configuration.

2) Grow goal-tree towards a random unexplored configuration.

3) Grow start tree towards goal tree. At each iteration, select a random vertex in the goal tree to grow towards it.

4) Grow goal tree towards start tree. A solution path is found when the two trees finally connect.

Table 1 The Basic RRT Algorithm

```
Build_RRT ( $x_{init}$ )
    T.init ( $x_{init}$ );
    For   k = 1 to K   do
        $x_{rand}$ = RANDOM_STATE();
        EXTEND (T, $x_{rand}$ );
    Return T
EXTEND (T, $x$ )
    $x_{near}$ = NEAREST( $x$ ,T);
    If   NEW_STATE ( $x, x_{near}, x_{new}, u_{new}$ )
        T.add_vertex ( $x_{new}$ );
        T.add_edge ( $x_{near}, x_{new}, u_{new}$ )
```

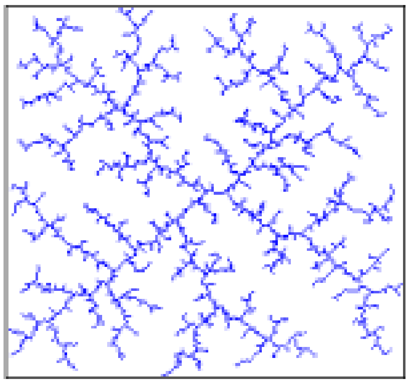

Fig. 1 Example of a basic RRT algorithm

2.2 Other RRTs

If a dual-tree approach offers advantages over a single tree, then it is natural to ask whether growing three or more RRTs might be even better. These additional RRTs could be started at random states. Of course, the connection problem will become more difficult for nonholonomic problems. Also, as more trees are considered, a complicated decision problem arises. The computation time must be divided between attempting to explore the space and attempting to connect RRTs to each other. It is also not clear which connections should be attempted. Many research issues remain in the development of this and other RRT-based planners.

3. RRT FOR REMOTE-CONTROLLED MOBILE ROBOT

3.1 Kinematics

We consider a mobile robot with a 4-wheel differential-drive skid-steering configuration, the two wheels on the same side move in unison, with each pair on opposite side capable of being driven independently. If both pairs are driven forward with the same speed, then the robot moves forward, but if they are driven in opposite directions, the robot will turn in place (i.e., executing a zero-radius turn).

The nominal model equation can be described by the following Eq. (1).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \qquad (1)$$

The state vector $X = [x\ y\ \theta]^T$ represents the robot position and orientation. The control inputs applied to the mobile robot are $u = [v\ \omega]^T$ where v is the translational velocity and ω is the rotational velocity of the robot.

3.2 Implementation of RRT

When we consider not the autonomous mobile robot but the remote-controlled mobile robot (RC-mobile robot), it has to keep the commanded translational and rotational velocities and avoid some obstacles such as static walls and dynamic moving obstacles (e.g., cars, people).

Most existing collision avoidance methods are purely reactive in the sense that they search for safe robot control commands based on the robot’s current proximity sensor data without any projection of the robot’s future state. These methods differ in the way this search is carried out. The Vector Field Histogram method [7] and Potential filed methods [8] do not explicitly take the constraints imposed by the dynamics of the robot into account. Another popular method is the dynamic window approach to collision avoidance [9]. This method searches for the trajectory the robot should take within the next time step based on a local map of the robot’s surrounding built from the latest sensor measurements. In order to reduce the search space, the robot’s dynamic constraints are taken into account by considering only velocities which can be reached within the next time interval. Konolidge [10] uses dynamic programming on the local map to compute the gradient towards the target. To make this approach computationally feasible only the two dimensional space of possible robot positions is considered for planning.

As mentioned in Section 2, RRT algorithms have the advantage that they can be directly applied to nonholonomic and kinodynamic planning. We will suggest a method that effectively modifies the basic RRT to be used for the RC-mobile robot and simultaneously plans collision free paths and takes the kinematic constraints of the robot into account.

There are several issues in the application of RRT algorithms. The main issues that should be considered to improve the performance of algorithms are as followings:

- sampling strategy (bias technique)
- metric choice
- input selection

Sampling strategy is related with how to bias the probability distribution density of random samples. The original RRT has a uniform random distribution and takes a long time to find a path to the goal. Some methods were suggested to improve the computational time. The goal-biased technique simply can be applied by the assignment the probability at the goal point [3]. And also, adaptive sampling bias technique was suggested with applications to test generation [11]. They initially bias the distribution so that states near the unsafe set are selected and monitor the growth of the tree. As the growth rate of the tree declines, the sampling distribution is less biased.

To find a metric that yields good performance can be a very difficult task. The ideal metric is the optimal cost-to-go, which is the optimal cost for the robot to move from one state to another state [6]. Calculating the optimal cost-to-go is at least the same difficulty as the trajectory design problem. In general, a simple Euclidean metric is used. For a particular system, it may be possible to derive a metric from several alternatives, including a Lyapunov function, a steering method, a fitted spline curve, or an optimal control law for a locally-linearized system. In [12], the cost-to-go function from a hybrid controller was used as the metric in an RRT to generate efficient plans for a nonlinear model of a helicopter.

Input selection to make a motion toward a randomly selected state from the nearest neighbor state and minimize the distance between them can be chosen at random, or selected by trying all possible inputs and choosing the one that yields a new state as close as possible to the sample (if the input range is infinite, then a finite approximation or analytical technique can be used). The determined state by the input implicitly uses the collision detection function to determine whether the new state satisfy the global constraints

We consider the practical implementation of a RRT algorithm for the remote-controlled mobile robot. In this case, the situation is so different from the autonomous mobile case. We assume the followings.

Assumptions:

- The mobile robot has a differential drive configuration.
- The translational and rotational velocities are limited.
- The translational and rotational accelerations are also limited.
- The mobile robot receives the target velocity commands every one second from the remote operator.
- The mobile robot knows the environments from various sensors.
- The mobile robot knows its position and attitude.

We need to modify a RRT algorithm in order to adapt the above assumptions in a remote-controlled mobile. These assumptions make the path planning as a local path planning and collision avoidance in some aspects. Next are our modifications on the RRT factors in this paper.

Bias Technique

Most of RRTs have a uniform distribution over the configuration space or a goal-biased Gaussian distribution as selecting a random state. In our case, we assumed that the RC-mobile robot receives the translational and angular velocity commands from the operator every one second (Δt). These commands indicate the direction of the goal region after one second. A simple goal region bias technique is utilized in selecting a random state x_{rand}. It has a Gaussian distribution centered along the commanded direction. The mean and standard deviation of the distribution can be calculated as shown in Eqs. (2) ~ (4).

$$x_{rand} \sim N(\mu, \sigma)\,, \qquad (2)$$

where μ is mean and σ is standard deviation.

$$\mu = \alpha \begin{bmatrix} \cos(\Delta\theta) & 0 \\ \sin(\Delta\theta) & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} + x_{cur} \tag{3}$$

$$\sigma = \alpha \begin{bmatrix} \cos(\Delta\theta) & 0 \\ \sin(\Delta\theta) & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_c \\ \omega_c \end{bmatrix} \tag{4}$$

where,

v_c : commanded translational velocity

ω_c : commanded rotational velocity

α : direction scaling factor

$\Delta\theta = \theta + \omega_c \Delta t$

Choice of Metric

Metrics are used when we choose the nearest vertex and select a control input that produces a new state near a random state. Finding metrics that yield good performance can be a very difficult task. Generally, the same metrics are used in those steps. In this paper, different metrics are used. In case of choosing the nearest vertex, a simple metric based on a weighted Euclidean distance of state is utilized. But in case of selecting a control input, the distance between the nearest state x_{near} and the goal region state x_{goal} that normalized with respect to the distance between x_{near} and x_{new} is added. This improves the performance of exploring to the goal region. Eq. (5) shows the metric that is used in the routine of finding the nearest vertex and Eq. (6) shows the metric that is used in the routine of selecting the control input.

$$\rho_{near} = \left\| x_{near} - x_{rand} \right\|_2 \tag{5}$$

$$\rho_{input} = \left\| x_{near} - x_{rand} \right\|_2 + w_g \left\| x_{near} - x_{goal} \right\|_2 , \tag{6}$$

where, w_g is weighting factor

Input Selection

The control input that drives the system slightly toward randomly-selected points is usually selected randomly. From a practical point of view, it is necessary to consider the constraints of inputs in path planning. We considers the constraints of the maximum velocity $v_{\max}$ and the maximum acceleration $a_{\max}$. Table 2 shows the input selection algorithm that takes these control input constraints and the commanded velocities into account.

Heuristically Pruning the Tree

As performing the input selection algorithm presented in Table 2, the function CALCULATE_MINIMUM_METRIC _INPUT simulates the mobile robot actuated by the selected control input u_t that meets constraints. If the calculated states of the mobile robot are occupied by a wall or other obstacles, This control input is invalid. The number of invalid input that occurred during the Select_Input loop is counted and characterized in the tree vertex. This information is used in the NEAREST routine that finds the nearest vertex from the randomly selected state in Table 1. If the invalid input count is over the predetermined percentage γ of all tried inputs, this vertex is pruned from the tree and will not be selected after then. This strategy that pruning the tree could make the overall computational time fast and prevent the mobile robot from being trapped in the local minimum.

Table 2 Input Selection Algorithm

```
Select_Input ( $x_{rand}, x_{near}, v_c, \omega_c, v_{\max}, a_{\max}$ )
    For k = 1 to K1 do
        $u_{rand}$ = $a_{\max}$ * UNIT_RANDOM();
        $u_t = v_c + u_{rand}$;
        If $u_t \geq v_{\max}$
            $u_t = v_{\max}$;

CALCULATE_MINIMUM_METRIC_INPUT( $u_t$ );
    Return $u$
```

4. SIMULATION RESULTS

We performed various simulations to identify the effect of different factors in designing a RRT algorithm for a remote-controlled mobile robot.

The parameters that used in simulations are shown in Table 3.

Table 3 Simulation Parameters

Parameter		Value
$v_{t\max}$	maximum translational velocity	$1[m/s]$
$v_{a\max}$	maximum angular velocity	$\frac{\pi}{2}\,[rad/s]$
$a_{t\max}$	maximum translational acceleration	$0.5[m/s^2]$
$a_{a\max}$	maximum angular acceleration	$\frac{\pi}{4}\,[rad/s^2]$
Δt	path planning step time	1 [sec]
w_a	angular distance weighting factor	0.7
w_g	weighting factor in ρ_{input}	1
γ	percentage in pruning tree routine	80 [%]
ε	goal region radius	0.5
K	maximum loop count	1500
K1	number of the randomly selected inputs	100
α	goal region direction weighting factor	5

Next subsections show the effect of the factors that have an influence on the algorithm's performance.

4.1 Comparison of Different Bias Techniques

First, we consider the effect of random state bias. The mobile robot initially locates in the state $x_0 = [3,\ 4,\ \pi]^T$. It is assumed that the mobile robot receives a command velocities and calculates the goal region near the state $x_{goal} = [3,\ 8,\ 0]^T$. Fig. 2 shows the result when we choose the goal-biased Gaussian distribution and Fig. 3 shows when we choose random distribution over the configuration space. We adapt the same random input sequences and take the same simulation conditions except the distribution. We can know that the goal-biased distribution gives better performances than the random distribution. First case directly goes to the goal region, but second case explores wide areas and takes much time to perform the algorithm.

4.2 Comparison of Different Metric Choice

To identify the effect of metric choices, we performed the following simulation. We also let the simulation conditions same for both cases except the input selection metric. The mobile robot initially locates in the state $x_0 = [1,\ 8,\ 0]^T$. It is assumed that the mobile robot receives a command velocities and calculates the goal region near the state $x_{goal} = [8,\ 7.5,\ 0]^T$. Fig. 4 shows the results in the case of adding the weighting factor w_g when we choose the control input as you can see in Table 2. We can know that the robot directly moves toward the goal. However, If the weighting factor is removed, as you can see in Fig. 5, the robot produces zigzag trajectory.

4.3 The Effects of Pruning Tree

To reduce the computational time, it could be need to get rid of the invalid vertexes in tree. As we referred in Section 3, we performed simulation to know how the pruning tree can increase the performance. The mobile robot initially locates in the state $x_0 = [4.5, 2, \pi/2]^T$. It is assumed that the mobile robot receives a command velocities and calculates the goal region near the state $x_{goal} = [1, 8, \pi/2]^T$. Fig. 6 shows the simulation result without pruning process. In the center of the configuration space, there are many vertexes tried to expand the tree, but, they failed to extend any more. In order to make the nearest search routine effectively, we removed the vertexes that counted invalid trials over 80 [%]. As you can see in Fig. 7, this process improved the performance of the exploration and reduced the computational time over 20 [%] than the result without pruning tree.

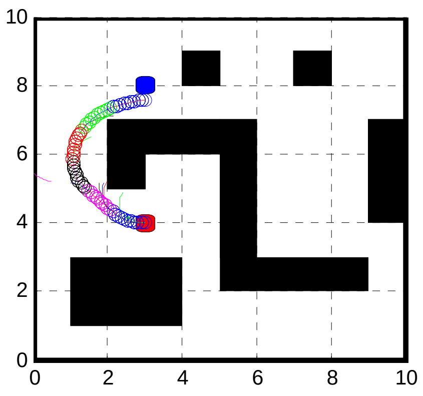
Fig. 2 Simulation result with goal-biased Gaussian distribution

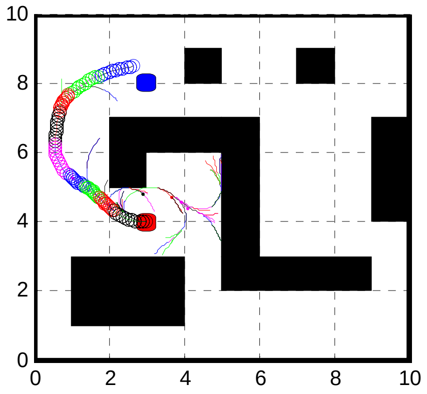
Fig. 3 Simulation result with random distribution

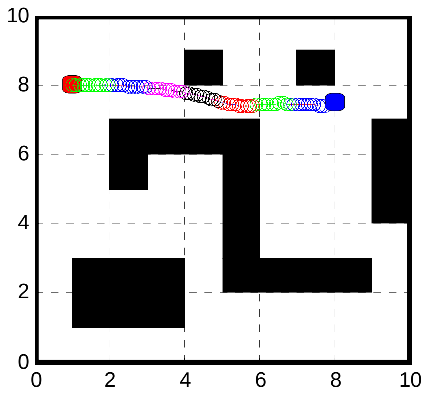
Fig. 4 Simulation result with weighting factor in the input selection

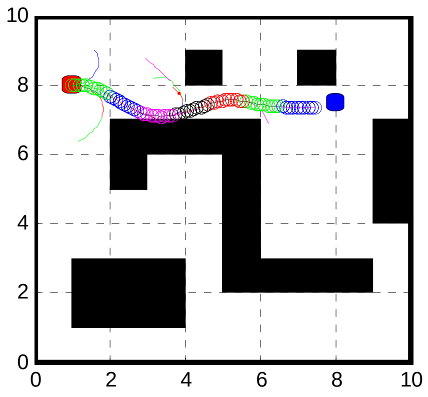
Fig. 5 Simulation result without weighting factor in the input selection

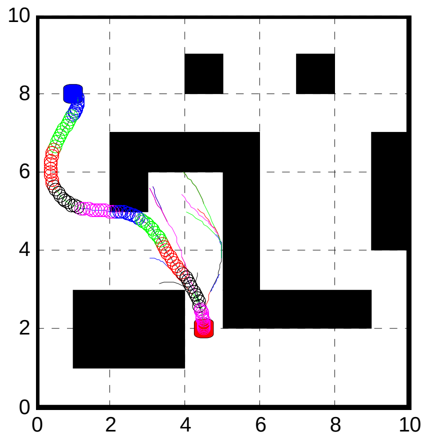
Fig. 6 Simulation result without pruning tree

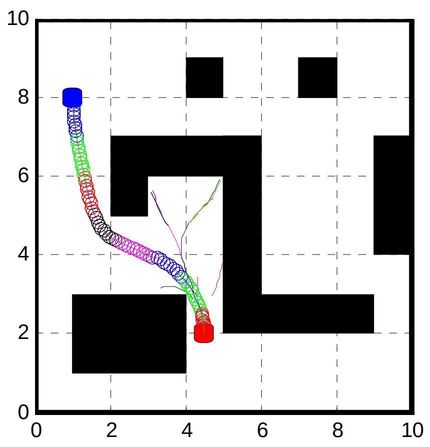

Fig. 7 Simulation result with pruning tree

5. CONCLUSION

We proposed a path planning algorithm based on the RRT method for a remote-controlled mobile robot. First, we considered a bias technique that is goal-biased Gaussian random distribution along the command directions. Secondly, we selected the metric based on a weighted Euclidean distance of random states and a weighted distance from the goal region. It can save the effort to explore the unnecessary regions and help the mobile robot to find a feasible trajectory as fast as possible. Finally, the kinematic constraints of the mobile robot and the constraints of the control inputs were considered in order to apply the algorithm to physical mobile robots. Simulation results demonstrate that the proposed algorithm is significantly more efficient for planning than a basic RRT planner. It reduces the computational time needed to find a feasible trajectory and can be practically implemented in a remote-controlled mobile robot.

In the future, we plan to experiment with a physical mobile robot to identify the performance of the proposed algorithm.

REFERENCES

[1] S. M. LaValle and J. J. Kuffner, “Rapidly- exploring random trees: A new tool for path planning,” *TR 98-11, Computer Science Dept.*, Iowa State University, Oct. 1998.

[2] J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” *Proc. IEEE Int’l Conf. on Robotics and Automation*, pp. 995-1001, 2000.

[3] S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pp. 293-308, A K Peters, Wellesley, MA, 2001.

[4] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan, “RRTs for nonlinear, discrete, and hybrid planning and control,” *Proc. IEEE Conf. Decision and Control*, December 9-12, 2003.

[5] J. M. Esposito, J. Kim, and V. Kumar, “Adaptive RRTs for validating hybrid robotic control systems,” *In Int’l Workshop on the Algorithmic Foundations of Robotics* 2004, Netherlands, 2004.

[6] Peng Cheng, “Reducing metric sensitivity in randomized trajectory design,” *Proc. Of the 2001 IEEE/RSJ Int’l Conf. on Intelligent Robots and Systems*, pp. 43-48, 2001.

[7] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Journal of Robotics and Automation*, Vol. 7, No. 3, pp. 278-288, 1991.

[8] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[9] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics and Automation Magazine*, Vol. 4, No. 1, 1997.

[10] K. Konolige, “A gradient method for real-time robot control,” *In Proc. of IEEE/JSR Conf. on Intelligent Robots and Systems*, 2000.

[11] Jongwoo Kim and Joel M. Esposito, "Adaptive sample bias for rapidly-exploring random trees with applications to test generation," *American Control Conference*, 2005

[12] E. Frazzoli, M.A. Dahleh, and E. Feron,. ”Robust Hybrid Control for Autonomous Vehicle Motion Planning,” In *IEEE Conf. on Decision and Control*, Sydney, Australia, pp. 821-826, 2000