Robotics and Autonomous Systems 60 (2012) 1579–1591

Contents lists available at SciVerse ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Real-time navigation using randomized kinodynamic planning with arrival time field

I. Ardiyanto *, J. Miura

Department of Computer Science and Engineering, Toyohashi University of Technology, Toyohashi, Aichi, 441-8580, Japan

ARTICLE INFO

Article history:
Received 9 January 2012
Received in revised form
10 July 2012
Accepted 23 September 2012
Available online 8 October 2012

Keywords:
Path planning
Arrival time field
Randomized tree
Mobile robot
Kinodynamic constraints

ABSTRACT

In this paper, we propose a novel path planning algorithm for a mobile robot in dynamic and cluttered environments with kinodynamic constraints. We compute the *arrival time field* as a bias which gives larger weights for shorter and safer paths toward a goal. We then implement a randomized path search guided by the arrival time field for building the path considering kinematic and dynamic (kinodynamic) constraints of an actual robot. We also consider path quality by adding heuristic constraints on the randomized path search, such as reducing unstable movements of the robot by using a heading criterion. The path will be extracted by backtracking the nodes which reach the goal area to the root of the tree generated by the randomized search, and the motion from the very first node will be sent to the robot controller. We provide a brief comparison between our algorithm and other existing algorithms. Simulation and experimental results prove that our algorithm is fast and reliable to be implemented on the real robot and is able to handle kinodynamic problems effectively.

1. Introduction

In many robot applications, a path planner plays an important role for making the robot fulfill the given tasks, such as approaching a destination and avoiding collision with obstacles. The usual sequence of the path planning algorithm is as follows: get the environment data using sensors, generate the path, and control the robot according to the generated path. There are several things which have to be considered to develop a path planning algorithm for real implementation of the robot: path optimality, path safety, and applicability to the real robot and environments.

Several parameters can be used to measure the path optimality, e.g. distance metrics, time cost, and other cost functions. For example, if we use a distance metric as the measurement, it means that the path which gives the shortest distance toward a destination will be considered as the optimal path. The path safety means the algorithm must ensure that the robot has a "good" interaction with its surrounding environment, such as not colliding with other objects. The safety of the path also means that the generated path does not harm the robot itself, e.g. the algorithm does not generate a path with a very drastic change of velocity, which may harm the motor of the robot.

In the real application, the robot has kinematic and dynamic restrictions, such as speed, acceleration, and possible motion. These problems are often addressed as kinodynamic constraints of motion planning. Calculation time is also very critical for a real implementation of path planning algorithms.

1.1. Related works

There are a lot of works which have been presented and discussed to address the problem of path planning. The randomized technique is one among many approaches which is used by many researchers [1,2]. A randomized path planner such as RRT (Rapidly-exploring Random Tree) [3] is widely accepted because of its ability to explore the tree in a vast area. A problem in RRT is that it produces a path with many branches over the space due to its natural behavior of using the randomized technique. Some studies have been conducted to overcome this problem. Urmson and Simmons [4] proposed a heuristic technique based on a probabilistic cost function to optimize generated trajectories. Another approach is presented by Bruce and Veloso [5] by introducing additional waypoint caches to improve the performance of the random tree algorithm.

Rodriguez et al. [2] presented a variant of the RRT algorithm which can explore narrow passages. Vonasek et al. [6] propose an iterative scaling approach for RRT search, based on an iterative refinement of the guiding path using a scaled model of the robot. There is also research on RRT using kinodynamic constraints by LaValle et al. [7] and Plaku et al. [8].

Several researchers [9,1,10] also worked on sampling-based path planners. Karaman and Frazzoli [9] designed an incremental sampling-based path planner and proved its optimality in a static

* Corresponding author.
E-mail addresses: iardiyanto@aisl.cs.tut.ac.jp (I. Ardiyanto), jun.miura@tut.jp (J. Miura).

0921-8890/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2012.09.011

environment. Jaillet et al. [1] proposed a sampling-based method on a configuration-space cost map. Zucker et al. [10] introduced a workspace-biased sampling to be applied on a bidirectional RRT.

Another work proposed by Hassouna et al. [11] does not use the randomized technique, but instead uses a potential function generated by the Level Set Method over the free space. The path is extracted using sequences of the best value of the field between the initial position of the robot and the goal.

1.2. Our approach

Most algorithms either only consider static environments (e.g. [11,3,4,7,9,1,10,2]) or need a long calculation time thereby making it hard to be applied to the real robot (e.g. [8]). Some other algorithms do not consider kinodynamic restrictions of the robot in their simulations (e.g. [11,9]). That means it will need much effort and modification to apply those algorithms to the real robot.

We introduce an algorithm called Heuristic Arrival Time Field-biased Random Tree (HeAT-RT) to overcome all of those problems. This paper is an extended version of [12] which explains the HeAT-RT algorithm. Basically, it is a real-time path planning algorithm that takes advantage of the high-exploration ability of a randomized tree combined with an arrival time field and heuristics to achieve the path optimality, safety, and applicability to the real robot. We use the arrival time field to give a bias, and guide the randomized tree expansion in a favorable way. Together with heuristics, the arrival time field effectively ensures that the robot chooses the path in the tree expansion with considerable clearance from any obstacle (safety) and has an optimum cost to reach the destination. These costs include the length, the time to reach the goal, and the smoothness of the path.

1.3. Paper outline

This paper is organized as follows. We explain the arrival time field and its properties used by our algorithm to perform the randomized search in Section 2. Section 3 describes the main algorithm of the random tree and how we optimize the trajectory generated by our path planner using a heuristic method. We then show simulation results, a comparison with other existing path planner algorithms, and experimental results in Section 4. The rest is the conclusion of our work and possible future work.

2. Arrival time field

We first describe the basic idea of the *arrival time field*. Let us consider the environment $\mathbb{C}$ as a two dimensional space that holds information as follows:

- Non-passable area, which holds information about walls and static obstacles, denoted by $\boldsymbol{O} \subseteq \mathbb{C}$.
- Free space area, which defines observed areas where the robot will not collide with walls as well as static obstacles, denoted by $\boldsymbol{F} \subseteq \mathbb{C}$.
- Unknown area, which defines areas that have never been observed by the robot, e.g. an area behind an obstacle which cannot be observed by any sensor, denoted by $\boldsymbol{U} \subseteq \mathbb{C}$.

2.1. Definition

We define the *arrival time field* as a space containing the information about the time needed by each point in the space for approaching a determined goal point. Let us first consider a basic kinematic equation in the one dimensional case which correlates speed V, position x, and time T as

$$\Delta x = V \Delta T, \tag{1}$$

or we can rewrite it as

$$\frac{\Delta T}{\Delta x} = \frac{1}{V}. \tag{2}$$

In a higher dimension, (2) can be expressed as

$$|\nabla T| = \frac{1}{V}. \tag{3}$$

Let a monotonic wave front originating from a determined source point move across a space; then the arrival time of the wave front in every point can be calculated using (3). The arrival time of a point depends on the distance from the source point and the speed used by the wave front traveling toward that point. This problem is known as the *eikonal equation* problem which can be solved by Godunov approximation [13]. In 2D space, for example, the equation is given by

$$\sqrt{\max\left(D_{i,j}^{-x}T, -D_{i,j}^{+x}T, 0\right)^2 + \max\left(D_{i,j}^{-y}T, -D_{i,j}^{+y}T, 0\right)^2} = \frac{1}{V_{i,j}}; \quad (i,j) \in \boldsymbol{F} \tag{4}$$

where

$$\begin{aligned} D_{i,j}^{+x} &= T_{i+1,j} - T_{i,j}, \\ D_{i,j}^{-x} &= T_{i,j} - T_{i-1,j}, \\ D_{i,j}^{+y} &= T_{i,j+1} - T_{i,j}, \quad \text{and} \\ D_{i,j}^{-y} &= T_{i,j} - T_{i,j-1}. \end{aligned} \tag{5}$$

$T_{i,j}$ is the arrival time value of cell (i,j), and $V_{i,j}$ denotes the speed function of cell (i,j). The solution of (4) can be retrieved using a solver such as Fast Marching Method [11,13], Fast Sweeping Method [14], or Fast Iterative Method [15].

We use Fast Marching Method (FMM) to solve (4). According to [11], (4) can be approximated by the first order finite difference scheme

$$\max\left(\frac{T_{i,j} - T_1}{\Delta x}, 0\right)^2 + \max\left(\frac{T_{i,j} - T_2}{\Delta y}, 0\right)^2 = \frac{1}{V_{i,j}^2} \tag{6}$$

where

$$\begin{aligned} T_1 &= \min\left(T_{i+1,j}, T_{i-1,j}\right) \\ T_2 &= \min\left(T_{i,j+1}, T_{i,j-1}\right). \end{aligned} \tag{7}$$

The solution[1] of (6) is given by

$$T_{i,j} = \begin{cases} T_1 + \dfrac{1}{V_{i,j}} & \text{for } T_2 \geq T \geq T_1 \\ T_2 + \dfrac{1}{V_{i,j}} & \text{for } T_1 \geq T \geq T_2 \\ \text{quadratic solution of (6)} & \text{for } T \geq \max\left(T_1, T_2\right). \end{cases} \tag{8}$$

Eq. (4) indicates that the arrival time of each point depends on its speed function $V_{i,j}$. That means we can set the influence of walls and static obstacles by adjusting the speed function, so that the areas near obstacles have larger arrival times. For that purpose, we implement a monotonic function denoted by

$$V_{i_1,j_1} = \begin{cases} n^{\left\| x_{i_1,j_1} - x_{i_2,j_2} \right\|} & \text{for } x_{i_1,j_1} \in \boldsymbol{F},\ x_{i_2,j_2} \in \boldsymbol{O} \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

[1] The quadratic solution in this equation means a quadratic equation $ax^2 + bx + c = 0$ has the solution $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

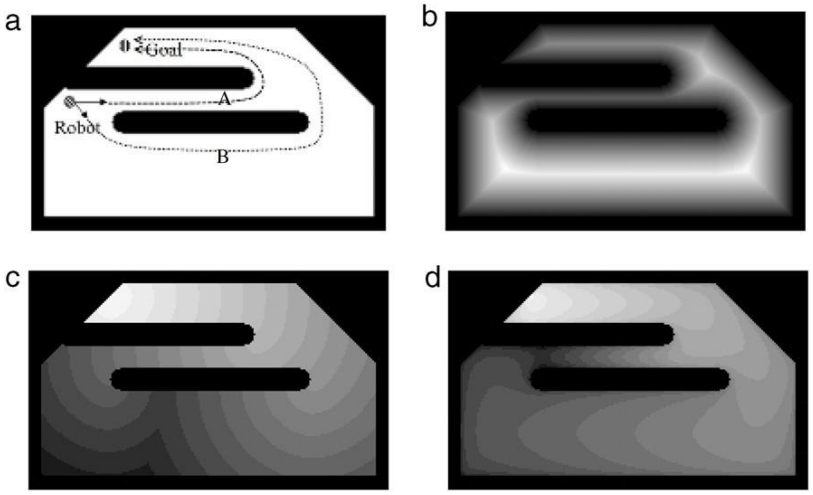

Fig. 1. Advantages of using monotonic velocity function: (a) An environment with two possible paths toward the goal. (b) Monotonic velocity field of (a). (c) Arrival time field with uniform velocity function. (d) Arrival time field with monotonic velocity function. Darker color means longer arrival time. By occupying monotonic velocity function, taking path B is better according to (d).

where V_{i_1,j_1} is the speed on the point x_{i_1,j_1}, x_{i_2,j_2} is the nearest point of an obstacle to x_{i_1,j_1}, and n is a constant for adjusting the monotonic function's value, to give more differences on each cell. We use the monotonic function's result for the speed function for calculating the arrival time field, which will make the speed near obstacles smaller. These definitions make the robot keep some distance from walls and obstacles.

The emphasis point of the arrival time field is that it provides the minimum predicted arrival time for each point rather than the shortest distance to the goal. This predicted arrival time depends on its speed function. We can exploit the speed function by inserting information about several possibilities that the robot may face in the environment. For example, in a plain environment with obstacles, we can determine that it is better for the robot to move slower at a narrow space, a corridor, or an area next to the obstacles. We set a smaller speed on the area near the obstacle; then a shorter travel time will be through an area far from the obstacle (see Fig. 1). This example shows that the safety factor is also taken into account in the arrival time field calculation.

The result of the arrival time field calculation is normalized and inverted so that the goal point has the highest weight. We then use that result as a bias for guiding the tree expansion.

3. HeAT Random Tree

It is easy to extract an optimal path for a point robot, i.e. the robot can freely move in all directions, using the result of the *arrival time field* by backtracking the path along the fastest field from the start to the goal [11]. In the case of considering kinematics and dynamics of robot as constraints, as well as dynamic environments, it is very difficult to apply such approaches. We therefore propose a randomized kinodynamic path planner algorithm utilizing the arrival time field bias as its guidance and heuristic search to optimize the path, called *Heuristic Arrival Time Field-biased (HeAT) Random Tree*.

3.1. Definition

Our randomized tree is constructed by collections of reachable states $\mathbb{S}$ called nodes. Every node is defined by the tuple $S = \{x, y, \theta, v, w, t\} \in \mathbb{S}$, representing the robot position in xy-axis and its heading θ, current translational velocity (v) and rotational velocity (w) of the robot in that node, and time t for reaching that node from the current state.

We give a predefined set of possible motions to the path planner. Each motion in the set consists of a translational and a rotational velocity as robot control denoted by $u_i = \{v_i, w_i\}$, $(i = 1, 2, \ldots, K)$, where K is the total number of motions, which satisfies kinematic constraints of the robot. Based on experiments, we currently use K = a set of 85 motions, which combines translational velocity (in mm/s) $v \in \{-100, -50, 0, \ldots, 600\}$ and rotational velocity (rad/s) $w \in \{-\frac{\pi}{2}, \ldots, \frac{\pi}{2}\}$.

Let S_t be the current state and S_{t+1} be the next state reached from S_t using a chosen motion $u = \{v, w\}$. We define this action of extending state S_t as a function

$$S_{t+1} \leftarrow g(S_t, u), \tag{10}$$

and according to [16], the new robot pose in state S_{t+1} can be obtained by the following equation:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \frac{v_1}{w_1} \begin{pmatrix} -\sin\theta_t + \sin(\theta_t + w_1\Delta t) \\ \cos\theta_t - \cos(\theta_t + w_1\Delta t) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ w_1\Delta t \end{pmatrix}, \tag{11}$$

where Δt is the time difference between S_t and S_{t+1}.

3.2. Short-time dynamic obstacle motion model

In a dynamic environment, we consider a state as a collision-free state if the state does not hit any static and dynamic obstacles. We need to perform a collision checking of every new state. Collision with static obstacles is checked using the information of non-passable area $\boldsymbol{O}$. For dynamic obstacles, we make a short-time dynamic obstacle motion model using constant speed for motion prediction of dynamic objects. Let $D'_x(t)$ and $D'_y(t)$ be the predicted positions of an obstacle at time t in x and y coordinates. We predict the position of each dynamic obstacle by

$$D'_x(t) = D_x(0) + v_{D_x}t \tag{12}$$

$$D'_y(t) = D_y(0) + v_{D_y}t \tag{13}$$

where $D_x(0)$ and $D_y(0)$ are the current positions of the obstacle,

and v_{D_x} and v_{D_y} are the speed of the obstacle on the respective x and y coordinates.

We assume that the motion prediction of a moving obstacle is effective for a short range of time, due to its uncertain behavior. We currently use fixed 10 time slices with a cycle time of 500 ms, starting from the time of the current state of the robot. For each time slice, we predict both the position of each moving obstacle and that of the robot in order to see whether the robot and dynamic obstacles will cause a collision in that time slice.

3.3. Random tree algorithm

We expand the tree from the current position of the robot, using a similar approach to basic RRT [3] to take advantage of its random exploration ability. Unlike the basic RRT algorithm which chooses a random point from the entire space, we select a random point using the bias from the *arrival time field* so that the tree grows in a favorable direction toward the goal. We iteratively choose a random point P_{target} which has a higher value of arrival time field than a threshold value (note that we invert the result of the arrival time field calculation, so that the goal point has the highest value). The threshold Th is determined by

$$Th = \text{bias}(S_{\text{init}}) + K_{\text{th}}(\text{bias}(S_{\text{far}}) - \text{bias}(S_{\text{init}})) \tag{14}$$

where $\text{bias}(S_{\text{init}})$ is the arrival time value at the current robot position, $\text{bias}(S_{\text{far}})$ is the arrival time value at the node which has the highest value of the arrival time in the current iteration, and K_{th} is a constant between 0 and 1. The threshold value starts from the value of the arrival time field of the initial state (current robot position), and will grow when a new created node has a higher bias value than all of the current existing nodes. We then choose the nearest node S_{near} to the P_{target} among all of the nodes in the tree.

Every time S_{near} is chosen and is eligible to be expanded, we will calculate a new state S_{new} of that node (S_{near}) by evaluating all of the possible motion controls

$$S_{u_i} \leftarrow g(S_{\text{near}}, u_i), \quad \text{for } i \in \{1, 2, 3, \ldots, K\} \tag{15}$$

where S_{u_i} is the extension of S_{near} using motion control u_i, and K is the total number of motions, which satisfies kinematic constraints of the robot, and is free from any collision. Let $\{v_i, w_i\} \in u_i$ be $\{v_2, w_2\}$ and $\{v, w\} \in S_{\text{near}}$ be $\{v_1, w_1\}$; then possible motion controls which meet kinematic constraints can be chosen if the motion satisfies

$$\begin{aligned} v_2 &\le v_{\max}, \\ w_2 &\le w_{\max}, \\ \frac{(v_2 - v_1)}{\Delta t} &\le a_{\max}, \\ \frac{(w_2 - w_1)}{\Delta t} &\le \alpha_{\max}, \end{aligned} \tag{16}$$

where $v_{\max}$, $w_{\max}$, $a_{\max}$, and $\alpha_{\max}$ respectively denote the maximum allowable translational velocity, angular velocity, translational acceleration, and angular acceleration, and Δt is the cycle time of calculation, currently 500 ms.

We then pick the best motion control

$$u_{\text{best}} = \arg\min_i \text{cost}(S_{u_i}), \tag{17}$$

which gives the best cost function, to get the new node

$$S_{\text{new}} = S_{u_{\text{best}}} \leftarrow g(S_{\text{near}}, u_{\text{best}}). \tag{18}$$

$\text{cost}(S)$ is a cost function for evaluating a defined motion as follows:

$$\alpha M_1 + \beta M_2 + \delta M_3, \tag{19}$$

$$M_1 = \text{bias}(S), \tag{20}$$

$$M_2 = \text{dist}(P_{\text{target}} - S), \tag{21}$$

$$M_3 = \left|\theta_S - \theta_{S_{\text{near}}}\right|, \tag{22}$$

where $\text{bias}(S)$ is the arrival time value at predicted point (x_S, y_S), $\text{dist}((x_{P_{\text{target}}}, y_{P_{\text{target}}}) - (x_S, y_S))$ is the distance between destination point $(x_{P_{\text{target}}}, y_{P_{\text{target}}})$ and the predicted point (x_S, y_S). $|\theta_S - \theta_{S_{\text{near}}}|$ is the heading difference of the robot between the current state and the predicted state, and α, β, and δ are constants.

The algorithm above is summarized as follows (see also Fig. 2 and Algorithm 1):

1. Determine the threshold of the region for picking a random point by (14).
2. Pick a random point P_{target} which has a better bias value than the threshold (see Algorithm 2).
3. Choose the nearest node S_{near} to the random point P_{target}.
4. Evaluate all possible motions using (15); and
5. Extend the tree (S_{new}) by choosing the best motion (see Algorithm 3).

Algorithm 1: HeAT Random Tree Planner

Properties :
$\mathbb{S}$ = collection of nodes
u = motion control
$bias(S)$ = arrival time value of node S
$threshold$ = bound the area for choosing random point

Function : HeAT_RANDOM_TREE_PLANNER()
$\mathbb{S} \Leftarrow S_{init}$
$threshold \Leftarrow bias(S_{init})$
while *time_is_available* **do**
  $S_{near} \Leftarrow$ CHOOSE_STATE(P_{target}, $\mathbb{S}$)
  $\mathbb{S} \Leftarrow \mathbb{S} \cup$ EXTEND_TREE(S_{near}, u)
  $Update(threshold)$ [1]
end while

[1] update the threshold value, according to (14).

Algorithm 2: Choose State

Properties :
$\mathbb{S}$ = collection of nodes

Function : CHOOSE_STATE(P_{target}, $\mathbb{S}$)
while *time_is_available* **do**
  P_{target} = random point from F
  if $bias(P_{target}) \ge threshold$ **then**
    return nearest_node(P_{target}, $\mathbb{S}$) [1]
  end if
end while

[1] return the nearest node in $\mathbb{S}$ to P_{target}.

The constant K_{th} in (14) gives us control over how fast the tree will grow and how disperse the tree will be. A high value of K_{th} means the threshold will increase rapidly, which then makes the tree grow fast and focus toward the goal. A low value of K_{th} means the threshold will increase slowly; then we will get a more disperse tree. In the current implementation, we use $K_{\text{th}} = 0.25$. Fig. 3 shows the growth of the threshold.

The basic randomized planner always tends to make a disperse path due to its natural behavior. We need to determine a proper criterion for extending every node chosen by the randomized planner to reduce inefficient and disperse motions. In this case, we want to reduce unstable movements that are often found in the path created by the randomized planner. We use the previous heading criterion as defined in (22) to ensure that we will not choose a very large difference of heading on each pair node causing unstable movements.

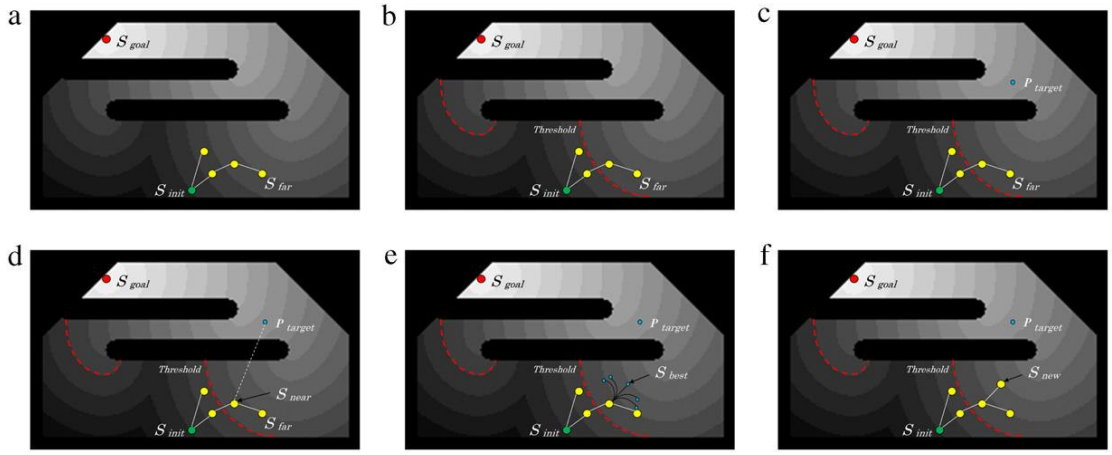

Fig. 2. Random tree algorithm: (a) Initial condition before extending the tree. (b) Determine the threshold. (c) Pick a random point (P_{target}). (d) Choose the nearest node (S_{near}). (e) Evaluate all possible motions. (f) Extend the tree (S_{new}).

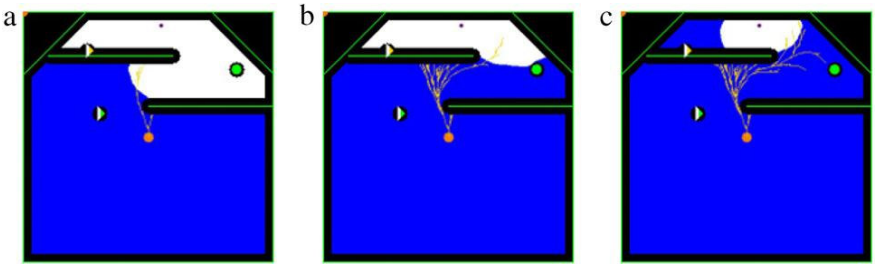

Fig. 3. Growing the threshold from (a) to (c). The white area is the region for choosing a random point.

Algorithm 3: Extending Tree

Properties :
u = set of motion controls
$temp\_cost$ = temporary variable for storing cost value
S' = temporary variable for storing the information of a node

Function : EXTEND_TREE(S_{near}, u)
for all u **do**
  $S' \Leftarrow (S_{near}, u)$
  if $cost(S')$[1] $\leq temp\_cost$ **then**
    $S_{new} \Leftarrow S'$
    $temp\_cost = cost(S')$
  end if
end for
return S_{new}

[1] cost value is calculated according to (19).

3.4. Restarting tree algorithm

We can expect that growing the tree from the initial point to the goal using the arrival time field bias takes a small amount of time. We will take advantage of this fact to construct more possible paths. Once a node in the tree reaches the goal area, the threshold for choosing a random point is set back to the value of arrival time field of the robot's current state, and we repeat the process of expanding the tree. We call these processes "restarting tree algorithm". We run the tree expansion algorithm for a fixed amount of time, currently 200 ms. These restrictions are implemented in order to keep the computation time as fast as possible to be recognized as a real-time path planner.

3.5. Directing initial robot heading

The utilization of kinodynamic constraints to the robot, i.e. the robot cannot freely move in all directions, may lead the randomized planner to make a large curve in the path when the target position is in the opposite direction of the robot. In this case, it is better to direct the robot in a certain heading; pointing the robot directly to the goal position is not the best choice, because with the appearance of obstacles, it may lead the robot to wrong trajectories. We overcome this problem by adding a heuristic that is to direct the initial robot heading to the most promising area using a small frame of the arrival time field.

A small frame of the *arrival time field*, centered on the robot's initial position, is divided into four regions (Fig. 4). We calculate the total weight of each region and choose the region which has the largest weight. When the region behind the initial position of the robot has a larger weight, we will rotate the robot to that region and set that rotation as the initial expansion of the tree. We can see a comparison of the algorithms with and without this initial heading in Fig. 5.

Fig. 5(a) shows a long arch in the path when the robot does not use the initial heading. On the contrary, we can reduce the cost of the path of the robot when we apply the initial heading at the beginning of the tree, as shown in Fig. 5(b).

3.6. Path extraction

HeAT Random Tree will provide several feasible paths of the robot from the initial state to the goal due to our *restarting tree algorithm*, satisfying the kinodynamic constraints and is free of collision with any static and dynamic obstacles. We examine all

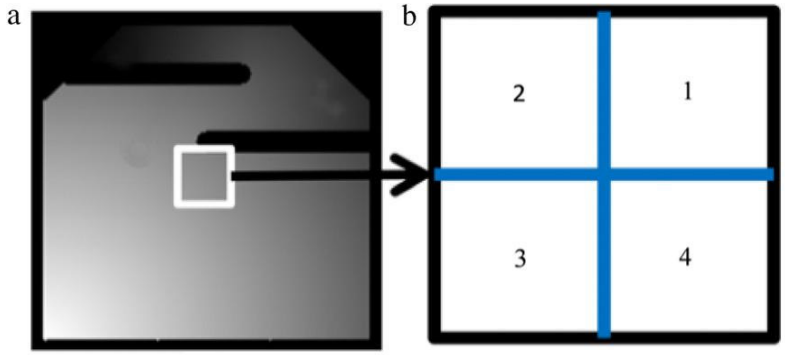

Fig. 4. Making a small frame of the arrival time field: (a) the arrival time field, (b) a part of (a) centered at the robot position, divided into four regions.

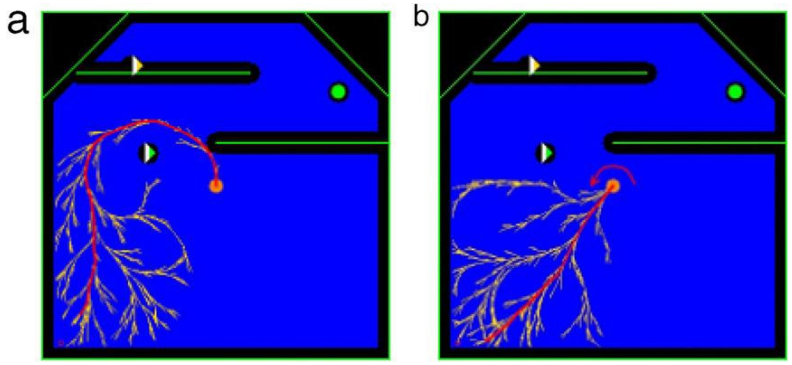

Fig. 5. Effect of directing the initial heading of the robot: (a) without and (b) with the initial heading. The goal is at the lower left corner.

of the feasible paths by backtracking from the nodes which reach the goal area to the root of the tree and select the path which is the fastest one and has the most *Minimum Work (MW)* cost (see Eq. (23)). The term *fastest* here means the path which has the minimum time to reach the goal, and there are possibilities that several paths will have the same minimum time because of the restarting tree algorithm.

MW cost is computed by

$$\mathrm{MW} = \sum_{t=\mathrm{start}}^{\mathrm{goal}} |w(t+1) - w(t)| \tag{23}$$

where $w(t)$ is the angular velocity/steering radius (in rad/s) on the node S_t. The path, the steering radius of which often changes, i.e. a non-smooth path, will have a higher value of MW.

MW cost ensures that the chosen path is the smoothest one among all of the fastest paths. The first motion of the path is sent to the robot controller.

3.7. Reusability of path

We use a very fast time cycle (currently, 500 ms per cycle) for calculating the entire algorithm, i.e. updating map information, static and dynamic obstacles, and performing calculations of HeAT Random Tree. We assume that the environment has not changed very much during that cycle. The path generated by the previous calculation is still expected to be feasible for the current cycle. The previous path is examined from the root to the longest collision-free state of the path and is used as the initial tree for the current calculation.

4. Experimental results

We test the HeAT Random Tree path planner both in simulation and experiment with the real robot. All implementations were done using a laptop PC (Core2Duo, 2.1 GHz, 2 GB memory, Windows XP). We implement our path planner algorithm as an RT-component (see Figs. 6 and 7) which is a software module running on RT-middleware[2] environment [17] for reusability (cooperates with other modules in the real implementation, e.g. sensor modules, mapping modules, etc.).

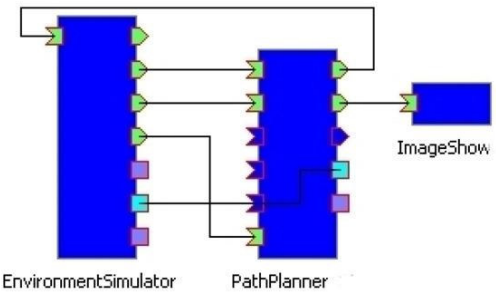

Fig. 6. RT-component connection for the simulations.

4.1. Simulation: local planner

We use an Environment Simulator [18] to perform the simulation of our path planner as a local planner. The simulator generates a 200 × 200 cell map consisting of free space and static obstacles as the local map for the robot, mimicking the canteen

[2] RT-middleware is a specification on a component model and infrastructure services applicable to the domain of robotics software development, authorized by OMG (Object Management Group).

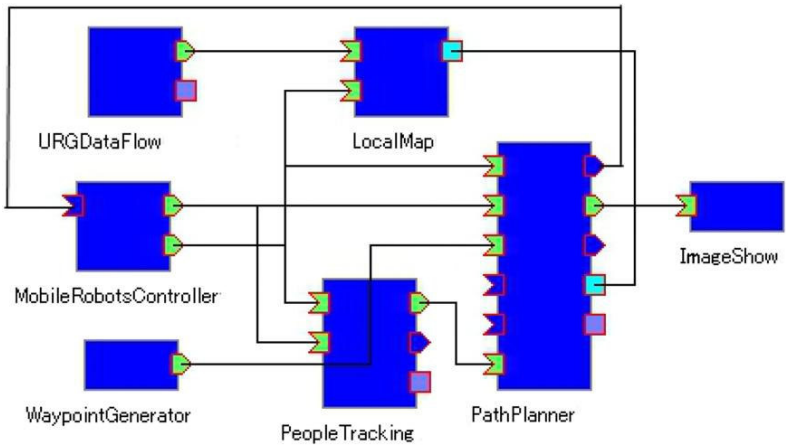

Fig. 7. RT-component connection for the experiments.

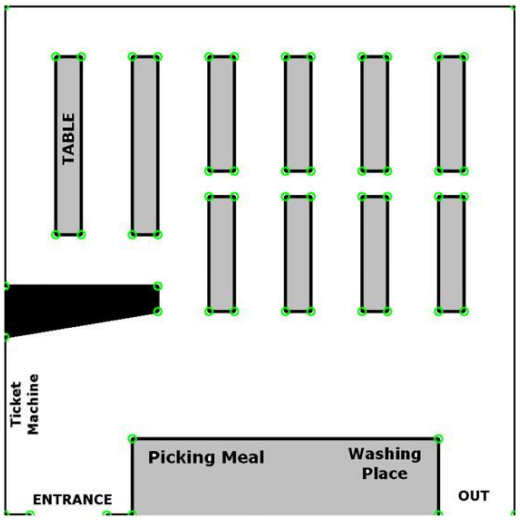

Fig. 8. Modeled environment for simulation.

of our university. This simulator also provides information about people movement to the path planner. The flow of the simulated people behavior is as follows: people enter the canteen, queue up for tickets at the ticket machine, take meals using a tray, go to a free seat, remain in the seat for eating, bring the tray to the washing place, and go to the exit (see Fig. 8).

We apply the HeAT Random Tree path planner to people tracking and waypoint following problems. Fig. 9 shows the simulation result on both problems, where the blue area denotes free space, the green line is wall and static obstacles, the orange circle denotes the robot position, the red circle is the goal, the triangles represent people movement, and the black area is the extending space for obstacles. Here, we want to show that our path planner is fast enough to run as a local planner, and can deal with both static and dynamic obstacles. Especially for the people tracking problem, we want to show that our path planner can deal with a dynamic goal.

In the people tracking simulation, the robot has to follow one of the people while avoiding static obstacles, walls, and other people. We will say that the robot has succeeded in solving the people tracking problem when the robot can follow the tracked person from the entrance until that person sits at a table for eating (see Fig. 9(a) and (c)).

In the waypoint following simulation, the robot is given a sequence of waypoints to follow where the goal lies on the very last waypoint of the sequence. The simulator gives several conditions of environment like corridors, intersections, and open space with wandering dynamic obstacles. The task of waypoint following problem will be judged a success if the robot can arrive at the goal safely (see Fig. 9(b) and (d)).

Table 1
Statistics of simulation result.

Statistics	People tracking	Waypoint following
Arrival time field calculation (max) (ms)	40	40
Random tree calculation (max) (ms)	250	250
Number of nodes (avg.)	1500 nodes	3000 nodes
Maximum speed (mm/s)	500	600
Number of simulations	10 times	10 times
Successful runs (%)	100	100

Table 1 shows the robustness of the HeAT Random Tree algorithm. The simulation of each problem is done 10 times successfully. Overall calculations need less than 500 ms. The path planner produces fewer nodes in the people tracking problem because the robot keeps close to the goal (i.e. the followed person), so that the area for expanding the tree becomes narrower than that on the waypoint following problem.

4.2. Simulation: global planner

We then use our algorithm as a global planner (see Fig. 10). Fig. 10(a) and (b) show the benefit of *the arrival time field* with a safety profile as we had explained in Section 2, where the safest path will be chosen if applicable for the robot. Fig. 10(d) and (f) show how our path planner will act in the environment with several possible paths. By using the *arrival time field*, our algorithm will try to find the fastest and smoothest path for the robot while maintaining safety. For example in Fig. 10(f), there are many possible paths and the *the arrival time field* nicely guides the tree expansion.

4.3. Comparison with other algorithms by simulation

We provide a brief comparison between the HeAT Random Tree algorithm and other existing RRT-based path planning algorithms. We use the original RRT algorithm by [3] and the hRRT algorithm by [4] for comparison. The original RRT algorithm chooses a random point uniformly from the entire space, and then extends the tree from the nearest node to that random point. The hRRT algorithm uses a heuristic method based on a probabilistic cost function. This algorithm selects a random point for extending the

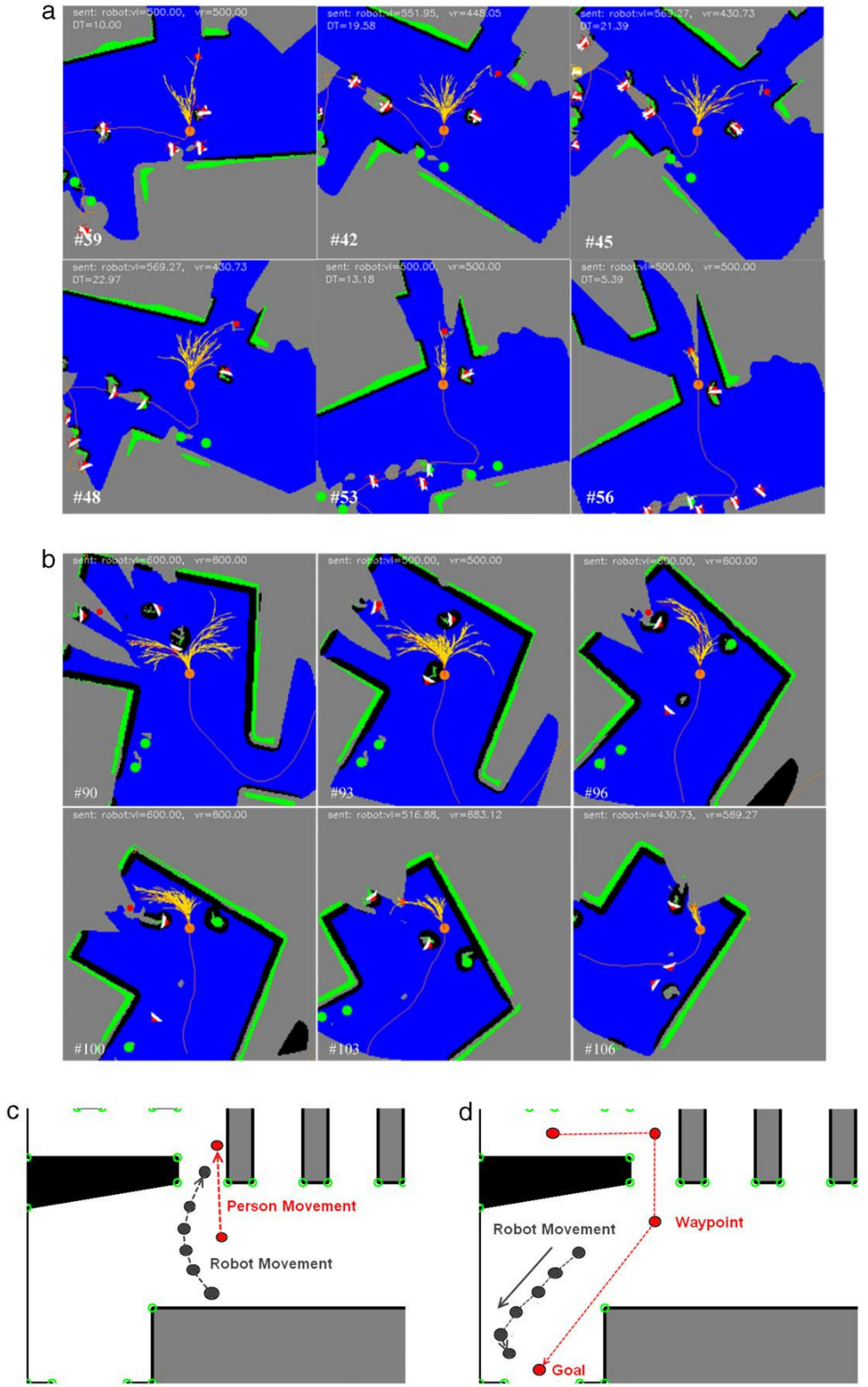

Fig. 9. Screenshot sequences of simulation using Environment Simulator: (a) people tracking problem, (b) waypoint following problem, (c) global map view of (a), (d) global map view of (b). People are the goal in (a). The blue area denotes free space, the green line is wall and static obstacles, the orange circle denotes the robot position, the red circle is the goal, the triangles represent people movement, and the black area is extending space for obstacles. Gray circles in (c) and (d) are global positions of the robot with respect to local positions in (a) and (b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

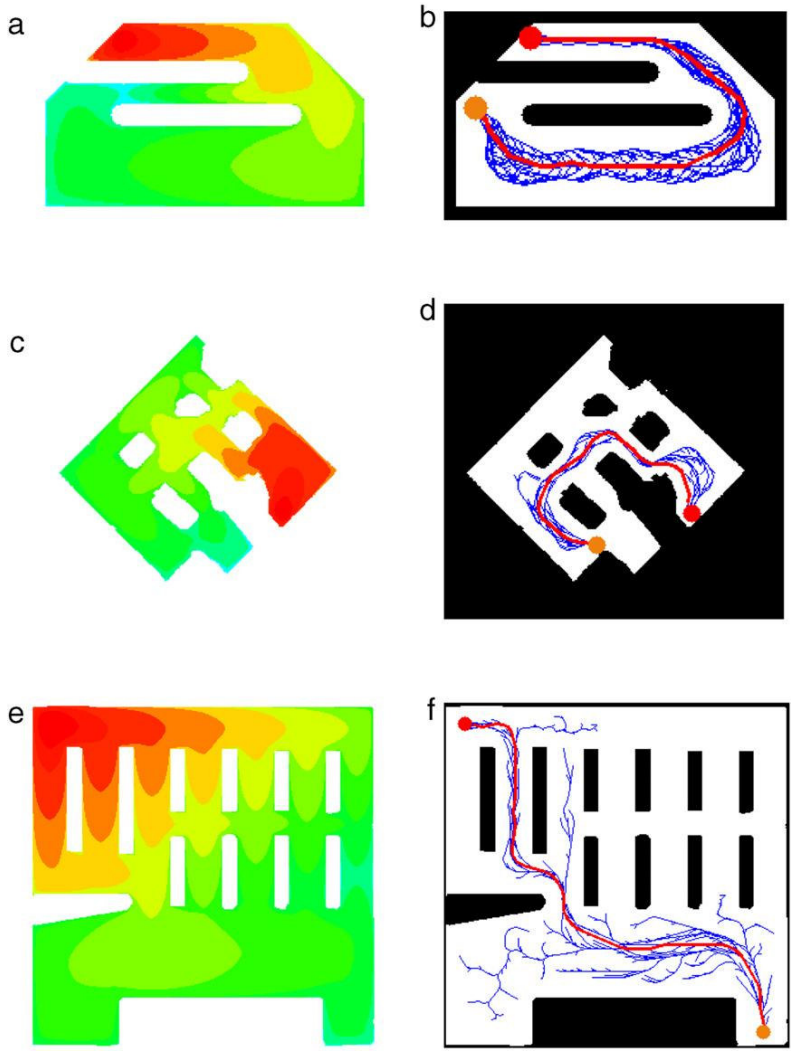

Fig. 10. Simulation of HeAT Random Tree as global planner using various maps, its *arrival time field* (left), and respecting tree expansions (right). The blue lines denote trees, the red line is robot path, the orange circle denotes robot position, the red circle is the goal, the white area is free spaces, and the black area is obstacles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

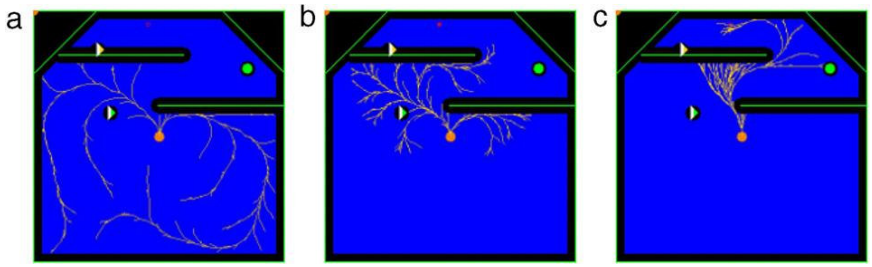

Fig. 11. Comparison of tree expansion on the dynamic environment: (a) original RRT, (b) hRRT, and (c) HeAT Random Tree.

tree using a *distance cost function* to reduce the dispersion of the tree produced by the original RRT algorithm.

For fairness of comparison, we use the same environment. We also apply the same kinodynamic constraints to all algorithms, even if each original algorithm did not deal with them. We set the maximum time to 300 ms for performing the calculation of each algorithm, to be considered as a real-time algorithm.

Fig. 11 can give us a good illustration about how the tree will expand in each algorithm on a dynamic environment. The original RRT algorithm expands the tree in a disperse way (Fig. 11(a)). hRRT gives a better result than the original RRT by occupying heuristic method of cost function, but this heuristic method based on distance cost only considers the free spaces and the static obstacles; therefore it cannot handle dynamic environments and finds it difficult to get out of obstacles in front of the robot as shown in Fig. 11(b). On the contrary, HeAT Random Tree algorithm is nicely guided by the arrival time field and heuristics, and expands in a favorable way as shown in Fig. 11(c).

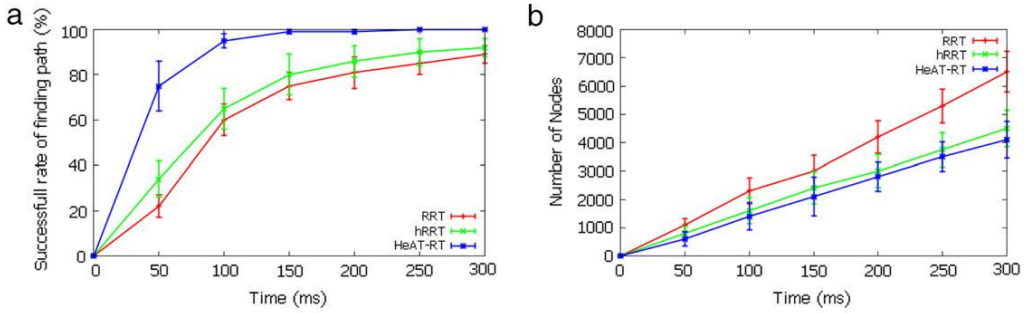

Fig. 12. Comparison of RRT, hRRT, and HeAT-RT algorithms: (a) successful rate of finding the path, (b) number of nodes.

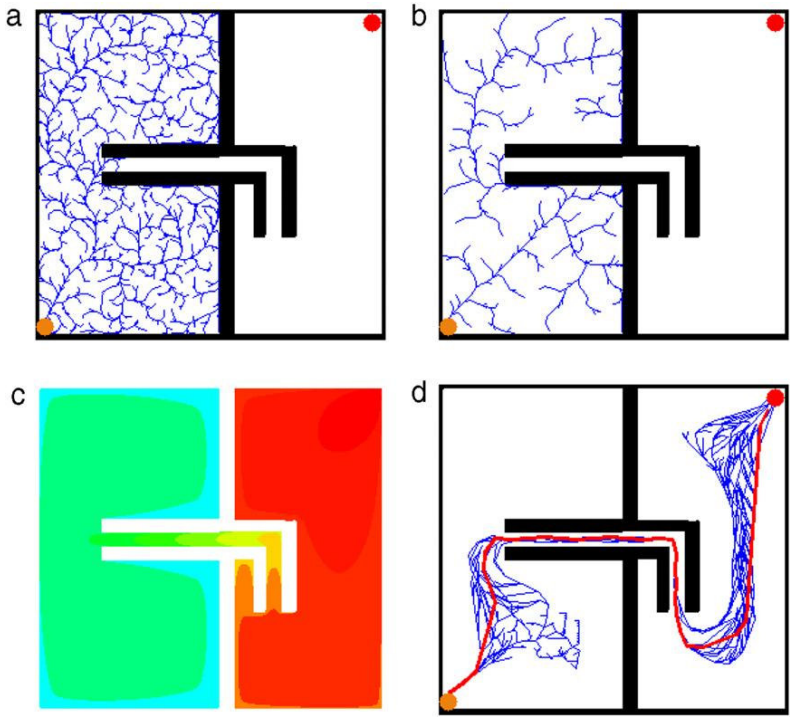

Fig. 13. Comparison of tree expansion on the environment with narrow passage: (a) original RRT, (b) hRRT, (c) arrival time field, and (d) HeAT Random Tree.

Fig. 12 shows the effectiveness of HeAT Random Tree against the original RRT and hRRT to find the path, based on the experiment shown in Fig. 11. HeAT Random Tree can rapidly find and generate the path, even though RRT and hRRT can produce more nodes at the same time. A high-rate of successful path generation is needed to ensure the robustness of the algorithm. HeAT Random Tree is expected to have a longer calculation time because of *arrival time field* generation, but surprisingly it is just a slight difference of node numbers among the three algorithms. This slight difference happens because the original RRT and hRRT face many collision states that will slow down the computation time due to kinodynamic constraints, whereas HeAT Random Tree is well-guided by the arrival time field so that the tree will be expanded to the safe area.

Figs. 13 and 14 show the comparison of each algorithm on the environment with narrow passages and bug-traps. Our algorithm effectively expands the tree, produces a nice path, and maintains safety, while RRT and hRRT fail to find the path. Both figures show the benefit of *the arrival time field* with a safety profile which guides the tree expansion in a favorable way. Note in Fig. 13(d) that, the path lies in the middle of a narrow passage, which means our path planner can maintain the safety of the path even in a difficult environment.

We then compare our HeAT Random Tree algorithm with RRT and hRRT algorithms using the Environment Simulator for solving the waypoint following problem. The robot is given the same initial position and goal to reach using each algorithm. Along the way in the simulator, the robot runs through several situations like narrow corridors, intersections, and open spaces with several moving obstacles surrounding the robot (see Fig. 15). We aim to test each algorithm using different kinds of environments and situations, to get better illustrations of performance of each algorithm. We run the simulation 10 times for each algorithm. Fig. 15 shows that our algorithm can keep the path as smooth as possible on the narrow corridor and intersection, and effectively grows the tree toward the goal in all situations, compared to other algorithms.

Table 2 shows the simulation results of each algorithm for the waypoint following problem. The results are taken and averaged from successful runs (i.e. the robot reaches the goal). Most of the collisions occur because of moving obstacles. Averaged MW-cost results show the smoothness of the path generated by HeAT

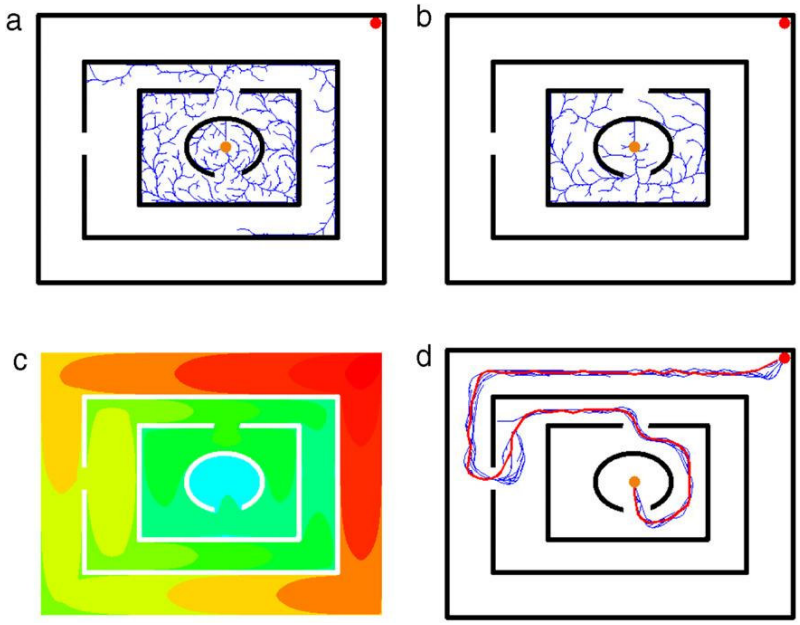

Fig. 14. Comparison of tree expansion on the environment with multiple bug-traps: (a) original RRT, (b) hRRT, (c) arrival time field, and (d) HeAT Random Tree.

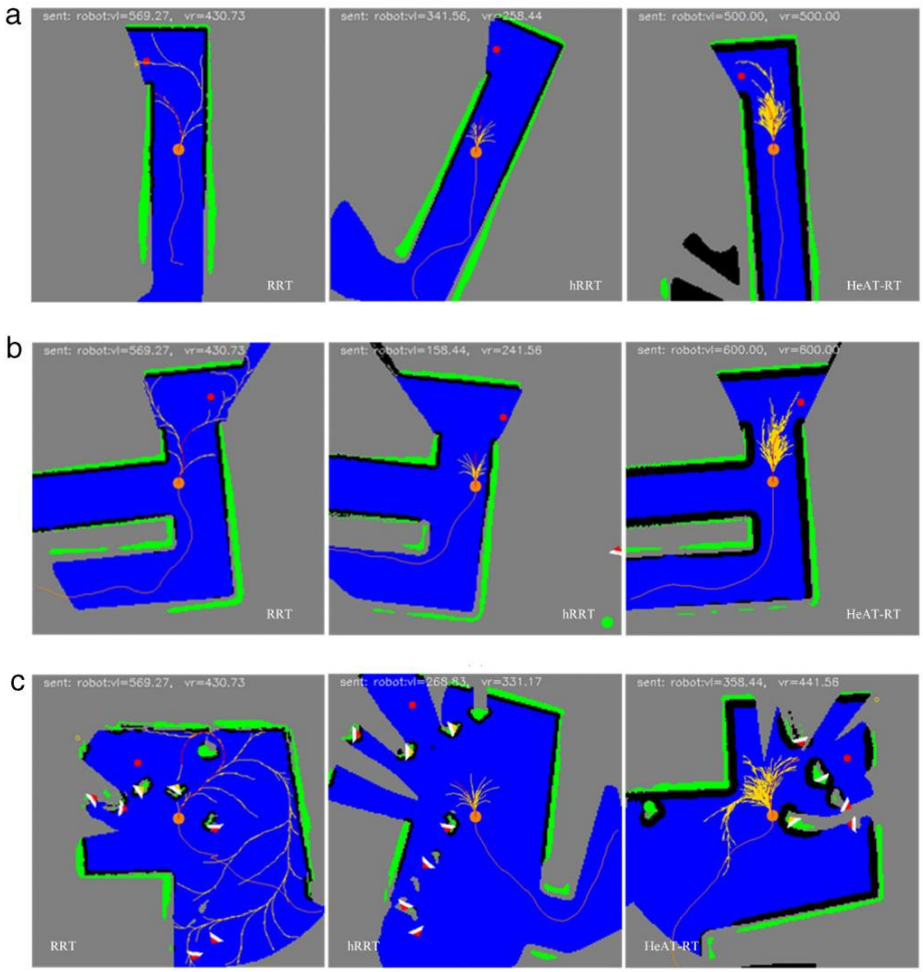

Fig. 15. Comparison of three algorithms at (a) narrow corridor, (b) intersection, and (c) open space with several moving obstacles.

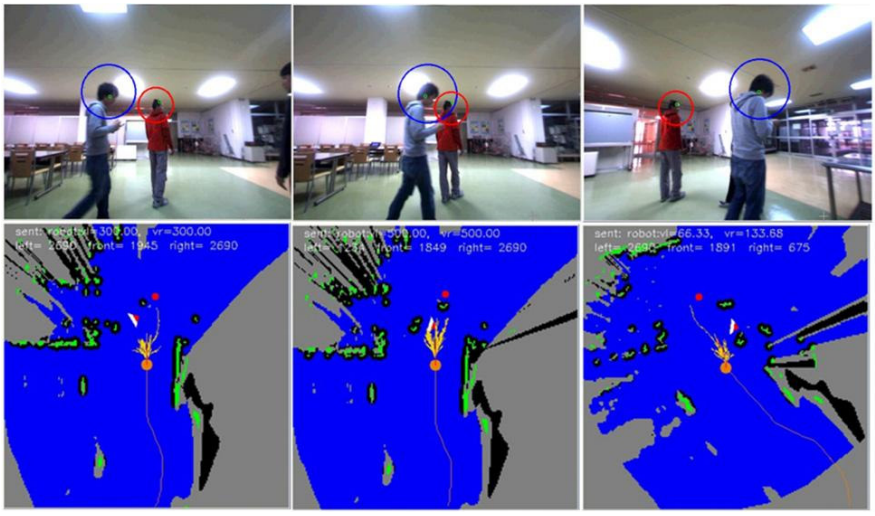

Fig. 16. Experiment of people tracking in complex environment (canteen of our university): (from left to right) sequences of the real world (top) and local map scene (bottom). The robot has to follow the person in the red circle. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

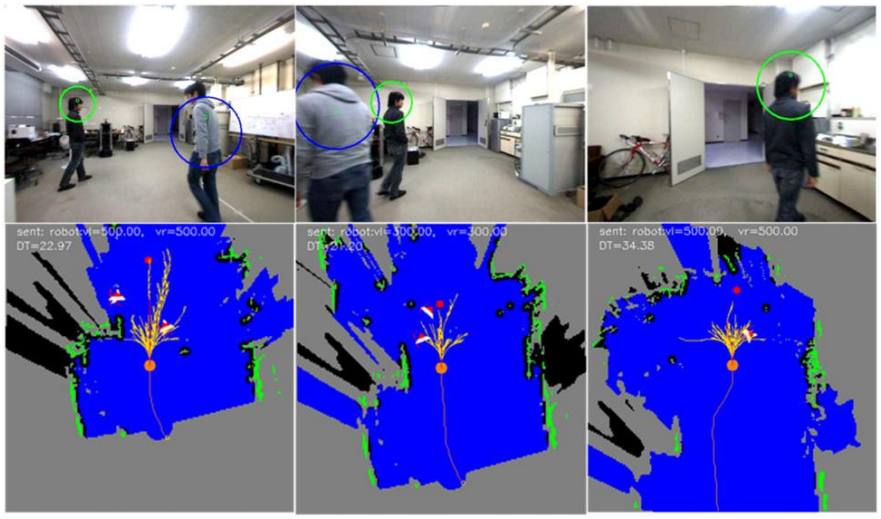

Fig. 17. In-room experiment of following waypoints: (from left to right) sequences of the real world (top) and local map scene (bottom). Circles in the top images represent detected people.

Random Tree, computed by

$$\text{Averaged MW} = \frac{\sum_{\text{step=start}}^{\text{goal}} |w(\text{step}+1) - w(\text{step})|}{\sum \text{step}} \tag{24}$$

where $w(\text{step})$ is the angular velocity/steering radius control (in rad/s) in each step, and $\sum \text{step}$ is the total number of steps from the start to the goal, and collision states are not included in the computation. A lower value of Averaged MW means the robot rarely changes direction. Overall, these results show the superiority of our algorithm.

4.4. Experiment on real robots

We use a Patrafour robot by Kanto Auto Works Ltd. and ENON by Fujitsu Ltd. for experiments, equipped with a stereo camera (Bumblebee2 by Point Grey), a laser range finder (UTM-30LX by Hokuyo), and laptop PC. We test the HeAT Random Tree path planner using two problems: people tracking and waypoint following problems.

Table 2
Comparison of three algorithms using Environment Simulator.

	RRT	hRRT	HeAT RT
Time to goal (avg.) (s)	157	113	58
Number of collisions (avg.)	10	9	2
MW-cost (avg.)	0.57	0.33	0.15
Successful runs (%)	50	70	100

The path planner utilizes a 200 × 200 grid of probabilistic local map with an actual size of 10 × 10 m. In the people tracking problem, the robot has to follow a person using a people tracking algorithm [19] while examining its environment. We use the canteen of our university representing a more complex environment (see Fig. 16). The robot follows one person while avoiding any collision with surrounding people.

In the waypoint following problem, we determine several waypoints which have to be reached by the robot in the real world. We use an in-room environment for doing a waypoint following experiment. The robot has to reach the goal which is located in front of the door while avoiding any collision with obstacles and wandering people (see Fig. 17).

Both following waypoints and people tracking tasks are successfully done within 500 ms of computation time per cycle and using a maximum speed of 500 mm per second. These experiments show the ability of the HeAT algorithm to be directly implemented on the real robot using a real environment.

5. Conclusions

We have presented a novel path planning algorithm which uses the *arrival time field* as a bias for a randomized tree search. Heuristic approaches of our algorithm are proved to be effective for handling a dynamic environment and kinematic constraints of the robot. We have shown that our algorithm is superior to other existing path planner algorithms. Simulation and experimental results also show that our algorithm is applicable to the real robot, and can be used in real-time.

Several improvements are possible to be applied. One is to integrate the moving obstacle model to the arrival time field so that we will get a *3D time–space* model of moving obstacles for a more reliable and efficient path. By using the 3D time–space model of moving obstacles, we can obtain several possibilities of routes and examine them to extract the most promising region for generating the path.

Acknowledgments

We would like to thank Atsushi Shigemura for developing the environment simulator component and Junji Satake and Masaya Chiba for developing the people tracking component. This work is supported by NEDO (New Energy and Industrial Technology Development Organization, Japan) Intelligent RT Software Project.

References

[1] L. Jaillet, J. Cortes, T. Simeon, Sampling-based path planning on configuration-space costmaps, IEEE Transactions on Robotics 26 (4) (2010) 635–646.
[2] S. Rodriguez, X. Tang, J.M. Lien, N.M. Amato, An obstacle-based rapidly-exploring random tree, in: Proc. of IEEE Int. Conf. on Robotics and Automation, ICRA, 2006, pp. 895–900.
[3] S.M. LaValle, J. Kuffner, Rapidly-exploring random trees: progress and prospects, in: Proc. of 4th Int. Workshop on Algorithmic Foundations on Robotics, WAFR'00, 2000.
[4] C. Urmson, R. Simmons, Approaches for heuristically biasing RRT growth, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems, IROS, 2003, pp. 1178–1183.
[5] J. Bruce, M. Veloso, Real-time randomized path planning for robot navigation, in: Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems, IROS, 2002, pp. 2383–2388.
[6] V. Vonasek, J. Faigl, T. Krajnik, L. Preucil, A sampling scheme for rapidly exploring random trees using a guiding path, in: Proc. of 5th European Conf. on Mobile Robots, 2011, pp. 201–206.
[7] S.M. LaValle, J. Kuffner, Randomized kinodynamic planning, in: Proc. of IEEE Int. Conf. on Robotics and Automation, ICRA, 1999, pp. 473–479.
[8] E. Plaku, L.E. Kavraki, M.Y. Vardi, A motion planner for a hybrid robotic system with kinodynamic constraints, in: Proc. of IEEE Int. Conf. on Robotics and Automation, ICRA, 2007, pp. 692–697.
[9] S. Karaman, E. Frazzoli, Incremental sampling-based optimal motion planning, in: Robotics: Science and Systems (RSS), 2010.
[10] M. Zucker, J. Kuffner, J.A. Bagnell, Adaptive workspace biasing for sampling-based planners, in: Proc. of IEEE Int. Conf. on Robotics and Automation, ICRA, 2008, pp. 3757–3762.
[11] M.S. Hassouna, A.E. Abdel-Hakim, A.A. Farag, PDE-based robust robotic navigation, Image and Vision Computing 27 (2009) 10–18.
[12] I. Ardiyanto, J. Miura, Heuristically arrival time field-biased (HeAT) random tree: an online path planning algorithm for mobile robot considering kinodynamic constraints, in: Proc. of IEEE Int. Conf. on Robotics and Biomimetics, ROBIO, 2011, pp. 360–365.
[13] J. Sethian, A fast marching level set method for monotonically advancing fronts, Proceedings of the National Academy of Sciences 93 (1996) 1591–1595.
[14] H. Zhao, A fast sweeping method for Eikonal equations, Mathematics of Computation 74 (2004) 603–627.
[15] J. Won-Ki, W. Ross, A fast iterative method for Eikonal equations, SIAM Journal on Scientific Computing 30 (2008) 2512–2534.
[16] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics, The MIT Press, 2005.
[17] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, W.K. Yoon, RT-middleware: distributed component middleware for RT (robot technology), in: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS, 2005, pp. 3555–3560.
[18] A. Shigemura, Y. Ishikawa, J. Miura, J. Satake, An RT component for simulating people movement in public space and its application to robot motion planner development, Journal of Robotics and Mechatronics 24 (1) (2012) 165–173.
[19] J. Satake, J. Miura, Robust stereo-based person detection and tracking for a person following robot, in: Proc. of ICRA Workshop on Person Detection and Tracking, Kobe, Japan, 2009.

I. Ardiyanto received the B.Eng. degree from Gadjah Mada University, Indonesia, and the M.Eng. degree from Toyohashi University of Technology (TUT), Japan. He joined the TUT–NEDO (New Energy and Industrial Technology Development Organization, Japan) research collaboration on service robots, in 2011. He is interested in planning and control systems for mobile robotics and computer vision.

J. Miura received the B.Eng. degree in mechanical engineering in 1984, and the M.Eng. and Dr.Eng. degrees in information engineering in 1986 and 1989, respectively, all from the University of Tokyo, Tokyo, Japan. In 1989, he joined the Department of Computer-Controlled Mechanical Systems, Osaka University, Suita, Japan. Since April 2007, he has been a Professor at the Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Japan. From March 1994 to February 1995, he was a Visiting Scientist at the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. He received the Best Paper Award from the Robotics Society of Japan in 1997. He was also selected as one of the six finalists for the Best Paper Award at the 1995 IEEE International Conference on Robotics and Automation. Prof. Miura has published over 100 papers in international journals and conferences in the areas of intelligent robotics, computer vision, and artificial intelligence.