Proceedings of the 2011 IEEE
International Conference on Robotics and Biomimetics
December 7-11, 2011, Phuket, Thailand

Heuristically Arrival Time Field-Biased (HeAT) Random Tree: An Online Path Planning Algorithm for Mobile Robot Considering Kinodynamic Constraints

Igi Ardiyanto and Jun Miura

Abstract— **This paper presents a new path planning algorithm for mobile robot in dynamic environments. We calculate the *arrival time field* as a bias which gives larger weights for shorter and safer paths towards a goal. We apply a randomized path search guided by the arrival time field for constructing the path considering kinematic and dynamic (kinodynamic) constraints of actual robot. We also consider path quality by adding heuristic constraints such as directing the initial heading of the robot and 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. We provide a brief comparison between our algorithm and other existing algorithms. Simulation and experimental results prove that our algorithm is fast enough to be applied to the real robot and show the effectiveness of the algorithm for handling kinodynamic problems.**

I. Introduction

Path planning is one of developing problems in the field of robotics. Path planning algorithm is used to make the robot fulfill given tasks, such as approaching the destination or avoiding collision with obstacles. A good path planner must be able to make a path reaching the destination efficiently and safely, can deal with both static and dynamic obstacles in the real environment, consider about path quality, and can be applied online in the real robot.

Many approaches have been presented and discussed to address the problem of path planning. Randomized technique is one among many approaches which is used by a lot of researchers ([7], [11]). Randomized path planner such as RRT (Rapidly-exploring Random Tree) [3] is widely accepted because of its ability to explore the tree in the vast area. A problem in RRT is that it produces a very spreading path over the space due to its natural behavior of using 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 [1] by introducing additional waypoint caches to improve the performance of the random tree algorithm. Rodriguez, *et al.* [11] presented a variant of RRT algorithm which can explore narrow passages. There are also researches on RRT using kinodynamic constraints by La Valle, *et al.* [5] and Plaku, *et al.* [10].

I. Ardiyanto and J. Miura are with Department of Computer Science and Engineering, Toyohashi University of Technology, 1-1 Hibarigaoka, Tenpaku-cho, Toyohashi, Aichi, 441-8580, Japan {iardiyanto, jun}@aisl.cs.tut.ac.jp

Several researchers ([6], [7], and [9]) also worked on sampling-based path planners. Karaman and Frazzoli [6] designed an incremental sampling-based path planner and proved its optimality on a static environment. Jaillet, *et al.* [7] proposed a sampling-based method on configuration-space costmap. Zucker, *et al.* [9] introduced a workspace-biased sampling to be applied on a bidirectional RRT.

Another work proposed by Hassouna, *et al.* [2] does not use 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.

Most of those algorithms are either only consider static environment (e.g. [2], [3], [4], [5], [6], [7], [9], and [11]) or need a long calculation time thereby making it hard to be applied in the real robot (e.g. [10]). Some other algorithms do not consider kinodynamic restrictions of the robot in their simulations (e.g. [2] and [6]). That means it will need much efforts and modification to apply those algorithms to the real robot.

Based on those problems, we propose a novel approach by introducing the *arrival time field* to guide a randomized path search. Basically, we want to take advantages of the randomized search method which can widely explore the area, then combine it with the arrival time field bias which can ensure the convergence of the path. We also consider kinodynamic constraints and the path quality of the robot so that our algorithm is applicable to the real robot.

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 II. Section III describes the main algorithm of random tree and how we optimize the trajectory generated by our path planner using heuristic method. We compare our algorithm with existing path planner algorithms in section IV. We then show the simulation and experimental results in section V. The rest is conclusion of our work and possible future works.

II. Arrival Time Field

A. Notation

Let us consider the environment $\mathbb{C}$ as a two dimensional space that holds information as follows:

- Non-passable area, holds information about walls and static obstacles, denoted by $\boldsymbol{O} \subseteq \mathbb{C}$;

- Free space area, defines visible areas where robot will not collide walls as well as static obstacles, denoted by $\boldsymbol{F} \subseteq \mathbb{C}$;
- Unknown area, defines the area that is not visible by the robot, e.g. area behind obstacle which cannot be observed by any sensor, denoted by $\boldsymbol{U} \subseteq \mathbb{C}$.

We use unknown area only for real implementation purpose (see section V, experiments on the real robot), because in the simulation we assume that every region can be classified either as free space or non-passable area.

B. Arrival Time Field

An *arrival time field* is used to provide a bias to drive expansion of the tree in a favorable direction toward the destination. We want the tree expands through a safe place within the fastest time. To fulfill such requirements, we apply the distance transform over free space $\boldsymbol{F}$ to give more weights on the cells near static obstacles and walls. We then perform calculation of the arrival time field originated from a goal point to give less weight on the closer position towards the goal.

The distance transform of point x_1 is calculated by

$$f(x_1) = \begin{cases} \sqrt{\|x_1 - x_2\|^2} & \text{for } x_1 \in \boldsymbol{F} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$f(x_1) \leftarrow n^{f(x_1)} \tag{2}$$

where x_2 is the nearest obstacle to x_1.

We implement Level Set Method as described in [2] to generate the arrival time field. For every point (*i,j*) in $\boldsymbol{F}$, the arrival time is given by

$$\sqrt{\max\left(D^{-x}_{i,j}T, -D^{+x}_{i,j}T, 0\right)^2 + \max\left(D^{-y}_{i,j}T, -D^{+y}_{i,j}T, 0\right)^2} = \frac{1}{f_{i,j}}; (i,j) \in \boldsymbol{F} \tag{3}$$

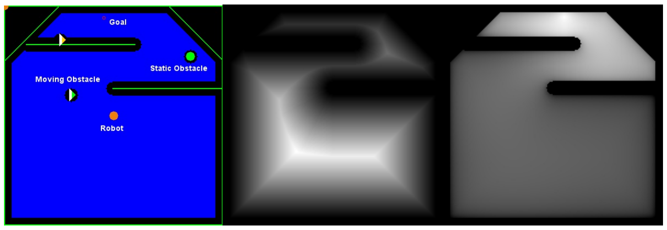
where

$$\begin{aligned} D^{+x}_{i,j} &= T_{i+1,j} - T_{i,j}, \\ D^{-x}_{i,j} &= T_{i,j} - T_{i-1,j}, \\ D^{+y}_{i,j} &= T_{i,j+1} - T_{i,j}, and \\ D^{-y}_{i,j} &= T_{i,j} - T_{i,j-1}. \end{aligned} \tag{4}$$

$T_{i,j}$ is the arrival time value of cell (*i,j*), and $f_{i,j}$ denotes speed function of cell (*i,j*).

Equation (3) indicates that the arrival time of each point depends on its speed function *f*. That means we can set the influence of walls and static obstacles by adjusting the speed function, so that areas near obstacles have larger arrival time. For that purposes, we implement a monotonic function to the distance transform as described in (2) to give more differences on each cell and use the distance transform's result as speed function for calculating the arrival time field, which will make the speed near obstacle become smaller. These definitions make the robot keep some distances from the walls and obstacles (see Fig. 1).

The value of each point is normalized and inverted so that the goal point has the highest weight. We consider the normalized arrival time value of every cell produced by those calculation above as a bias for guiding the tree expansion.



(a) (b) (c)

Fig. 1: (a) Environment map, where blue area denotes free space, green line is wall, the orange circle denotes robot position, the red circle is destination, the triangles are moving obstacles, and the black area is extending space for obstacles. (b) Distance transform of (a). (c) Arrival time field, brighter area means higher value.

III. HeAT Random Tree

It is easy to extract an optimal path for a point robot, i.e. the robot can freely move to all direction, using the result of arrival time field by backtracking the path along the fastest field from start to the goal [2]. In the case of considering kinematic and dynamic 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.

Heuristically Arrival Time Field-biased (HeAT) Random Tree is constructed by collections of reachable states called node. Every node is defined by the tuple $N = \{x, y, \theta, v, w, t\}$, representing robot position in xy-axis and its heading of 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.

A. Predefined Motion Set

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 notated by $u_i = \{v_i, w_i\}, (i = 1, 2, \ldots, K_u)$, where K_u is the total number of the motions, which satisfies kinematic constraints of the robot.

B. State Extension

Let N_t be the current state and N_{t+1} be the next state reached from N_t using a chosen motion set $u_1 = \{v_1, w_1\}$. We define this action of extending state N_t as a function

$$N_{t+1,u_1} \leftarrow g(N_t, u_1), \tag{5}$$

and according to [8], the new robot pose of N_{t+1} can be obtained by the following equation:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \frac{v_1}{w_1} \begin{pmatrix} -\sin\theta_t + \sin(\theta_t + w_1\Delta t) \\ \cos\theta_t - \cos(\theta_t + w_1\Delta t) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ w_1\Delta t \end{pmatrix} \tag{6}$$

where Δt is the time difference between N_t and N_{t+1}.

C. Short-time Dynamic Obstacle Motion Model

In a dynamic environment, we consider a state as 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 position of an obstacle at time t in x and y coordinate. We predict the position of each dynamic obstacle by

$$D_x^{'}(t) = D_x(0) + v_{D_x} t \tag{7}$$

$$D_y^{'}(t) = D_y(0) + v_{D_y} t \tag{8}$$

where $D_x(0)$ and $D_y(0)$ are the current position of each obstacle, and v_{D_x} and v_{D_y} are speed of the obstacle on the respective x and y coordinate.

We assume that motion prediction of moving obstacle is effective for a short range of time, due to its uncertain behavior. We currently use fixed 10 time slices with cycle time of 500 milliseconds, started 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 to see whether the robot and dynamic obstacles will cause a collision in that time slice.

D. Directing Initial Robot Heading

The utilization of kinodynamic constraints to the robot, i.e. the robot can't freely move to all directions, may lead the randomized planner to make a large curve of 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, but pointing the robot directly to the goal position is not best choice, because in appearance of obstacles, it may lead the robot to wrong trajectories or even local minima. 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 arrival time field.

A small frame of the *arrival time field*, centered on the robot initial position, is divided into four regions (Fig. 2). 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 larger weight, we will rotate the robot to that region and make 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. 3. Fig. 3a shows a long arch of path when the robot doesn't use the initial heading. Contrary, we can reduce the cost of the path when we apply the initial heading at the beginning of the tree, as shown in Fig. 3b.

E. Best-first Strategy for Simple Situation

We aim at the fastest calculation time for making an on-line path planner. In a simple situation (e.g., the goal is in front of the robot with little number of obstacles), we try to generate a simple motion using kinodynamic constraints.

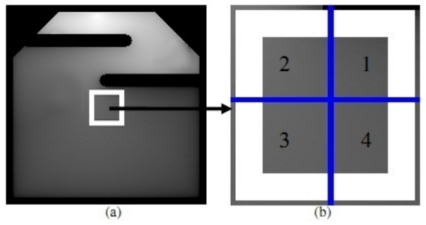
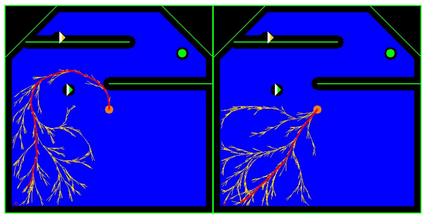


Fig. 2: (a) the arrival time field, (b) a part of (a) centered at the robot position, divided into four region



(a) (b)

Fig. 3: Effect of directing the initial heading of the robot, (a) without and (b) with the initial heading

Basically, we try to generate the path by repeatedly adding the best motion in the motion set to the current state N_t and get the predicted state N_{t+1}

$$N_{t+1,u_{best}} \leftarrow g(N_t, u_{best}), \tag{9}$$

where u_{best} is obtained by examining all of possible motions

$$N_{t+1,u} \leftarrow g(N_t, u), \text{for } u \in \{u_1, u_2, u_3, \ldots, u_{K_u}\} \tag{10}$$

$$u_{best} = \arg\min_{u} cost(N_{t+1,u}) \tag{11}$$

where K_u is the total number of the motion set, which satisfies kinematic constraints of the robot. $cost(N_{t+1,u})$ is a cost function for evaluating a motion, defined by

$$\alpha * M_1 + \beta * M_2 + \delta * M_3, \tag{12}$$

$$M_1 = bias(x_{t+1}, y_{t+1}) \tag{13}$$

$$M_2 = dist((x_{goal}, y_{goal}) - (x_{t+1}, y_{t+1})) \tag{14}$$

$$M_3 = |\theta_{t+1} - \theta_t| \tag{15}$$

where bias(x_{t+1}, y_{t+1}) is the arrival time value at predicted point(x_{t+1}, y_{t+1}) generated by a motion u, $dist((x_{goal}, y_{goal}) - (x_{t+1}, y_{t+1}))$ is the distance between destination point (x_{goal}, y_{goal}) and predicted point (x_{t+1}, y_{t+1}). $|\theta_{t+1} - \theta_t|$ is the heading difference of the robot between the current state and the predicted state, and α, β, and δ are constants.

This heuristic motion planner is very fast when it finds a feasible path, so we make a time limit for performing this action. If this heuristic cannot find a path or collide with an obstacle within the time limit, then we invoke the randomized tree search.

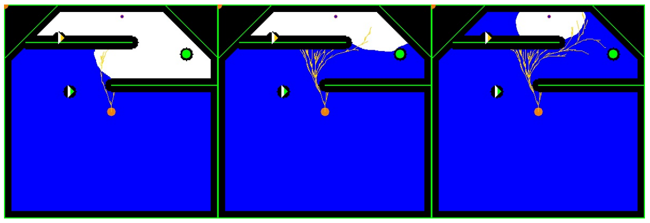


Fig. 4: Expanding the tree using bias (see algorithm (1)). The white area is the region for choosing a random point.

F. Random Tree Algorithm

We expand the tree from the current position of the robot, using a similar approach to basic RRT [3] to take the advantages 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 towards the goal. We choose a random point which has higher value of arrival time field than a threshold value. This threshold value is started from the value of the arrival time field of the initial state, and grows when a new created node has a better value than this threshold (see Fig. 4). This heuristic is performed as algorithm 1.

Every time a node is chosen and eligible to be expanded, we will calculate a new state of this node using all of possible motion controls. We then pick one of the motion controls by repeating (9), (10), and (11), which is free from any collision and gives the best cost function evaluated by (12), (13), (14), and (15), where (14) is slightly modified to $dist((x_{target}, y_{target}) - (x_{t+1}, y_{t+1}))$, that is the distance between chosen random point(x_{target}, y_{target}) and the predicted point (x_{t+1}, y_{t+1}).

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 randomized planner. We use the previous heading criterion as defined in (15) to ensure that we will not choose a very large difference of heading on each pair node causing unstable movements.

G. 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 the advantage of this fact to construct more possible paths. Once a node in the tree reach the goal area, the threshold for choosing random point is set back to value of arrival time field of robot's current state, and we repeat the process of expanding the tree. We call that processes as "restarting tree algorithm". We run the tree expansion's algorithm on a fix amount of time, currently 200 ms, or we stop the algorithm if the maximum number of node is reached (currently 3000 nodes). These restrictions are implemented in order to keep the computation time as fast as possible to be recognized as a real-time path planner.

Algorithm 1: HeAT Random Tree

```
Properties :
$N$ = collection of nodes
$u$ = motion control
$bias(x)$ = arrival time value of point $x$

Function : HeAT_RANDOM_TREE_PLANNER()
$N \Leftarrow x_{init}$
$temp\_bias \Leftarrow bias(x_{init})$
while $time\_is\_available$ do
    $x_{near} \Leftarrow$ CHOOSE_STATE$(x_{rand}, N)$
    $N \Leftarrow N \cup$ EXTEND_TREE$(x_{near})^1$
    if $bias_(x_{new}) \geq temp\_bias$ then
        $temp\_bias \Leftarrow bias(x_{new})$
    end if
end while

Function : CHOOSE_STATE$(x_{rand}, N)$
while $time\_is\_available$ do
    $x_{rand}$ = random point from $F$
    if $bias_(x_{rand}) \geq temp\_bias$ then
        return BEST_NODE$(x_{rand}, N)^2$
    end if
end while
```

^{1}make an extension node by evaluating every possible u according to (10).
^{2}return the nearest node of N to x_{rand}.

H. Path Extraction

HeAT Random Tree will provide several feasible paths of the robot from the initial state to the goal satisfying the kinodynamic constraints and is free of collision with any static and dynamic obstacles. We select the fastest path by backtracking the nodes which reach the goal area to the root of the tree. The first motion of the path is sent to the robot controller.

I. Reusability of Path

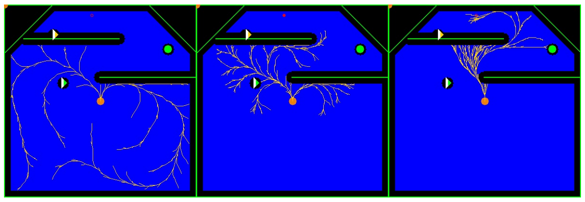
We use a pretty fast time cycle (currently, 500 milliseconds per cycle) for calculating the entire algorithm i.e., updating map information, static and dynamic obstacles, and performing calculation of HeAT Random Tree. We assume that the environment is not so much changed during that cycle. The path 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 used it as initial tree for the current calculation.

IV. Comparison of HeAT Random Tree and Other Existing Algorithms

We provide a brief comparison between HeAT Random Tree algorithm and other existing RRT-based path planning algorithms. We use RRT algorithm by [3] and hRRT algorithm by [4] for comparison. RRT algorithm chooses a

TABLE I: Comparison of Path Planning Algorithms

	Success Rate of 20 times Run (%)	Average of Execution Time of Successful Runs (ms)
Basic RRT	65	75
hRRT	70	95
HeAT Random Tree	95	110



(a) (b) (c)

Fig. 5: Comparison of tree expansion, (a) Basic RRT, (b) hRRT, and (c) HeAT Random Tree

random point uniformly from the entire space, then extends the tree from the nearest node to that random point.

hRRT algorithm uses a heuristic method based on a probabilistic cost function. This algorithm selects a random point for extending the tree using a distance cost function so that dispersion of the tree is expected to be reduced.

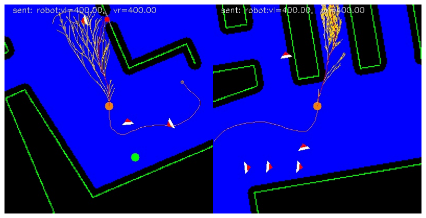
For fairness of comparison, we use the same environment including presences of the static and dynamic obstacles. We also apply the same kinodynamic constraints to all of algorithms, even if each original algorithm did not concern about it. We set the maximum number of nodes to 1000 and the maximum time to 200 milliseconds for performing the calculation of each algorithm. Table I and Fig. 5 show the comparison of each algorithm.

Table I shows the effectiveness of HeAT Random Tree against Basic RRT and hRRT. High-rate of successful path generation is needed to ensure the robustness of the algorithm. HeAT Random Tree is expected to have longer time of calculation because of *arrival time field* generation, but surprisingly it is just a slight difference of the execution time among the three algorithms. This slight difference happens because basic 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.

Fig. 5 can give us a good illustration about how the tree will expand in each algorithm. Basic RRT algorithm expands the tree in a disperse way (Fig. 5a). hRRT gives better result than Basic 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 environment and difficult to get out of obstacles in front of the robot as shown in Fig. 5b. 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. 5c.

TABLE II: Statistic of Simulation Result

Statistic	Value
Arrival Time Field calculation	40 ms
Best-first Path calculation	10 ms
Random Tree calculation	250 ms
Number of nodes	3000 nodes
Maximum speed	400 mm/second
Number of simulation	10 times
Successful runs	100%



(a) (b)

Fig. 6: Screenshot of simulation using Environment and People Movement Simulator

V. Result

We test HeAT Random Tree path planner in both of simulation and experiment with the real robot. All of implementations were done using a laptop PC (Core2Duo, 2.1 GHz, 2GB memory, Windows XP). We implement our path planner algorithm as an RT-component which is software module running on RT-middleware[1] environment [14] for reusability.

A. Simulation

We use an Environment and People Movement Simulator [12] to perform simulation of our path planner. The simulator generates a 200x200 cells of map consist of free space and static obstacles mimicking canteen of our university. This simulator also provides dynamic objects information to the path planner. These dynamic objects represent the movement of people. We apply HeAT Random Tree path planner to people tracking problems (see Fig. 6).

The robot has to follow one of dynamic obstacles considered as a person while avoiding static obstacles, walls, and other people moving inside the simulator. Simulation's flow is as follows: people enter the canteen, queue the ticket at ticket machine, take the meals using a tray, go to a free seat, stay on the seat for eating, bring the tray to the washing place, and go to the exit. We will say that the robot has succeeded in solving people tracking problem when the robot can follow the tracked person from the entrance until that people stay at the table for eating.

Table II shows the robustness of HeAT Random Tree algorithm. All of 10 times simulations have been done successfully. Overall calculations need less than 500 milliseconds, it means HeAT Random Tree can be run online.

[1]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).

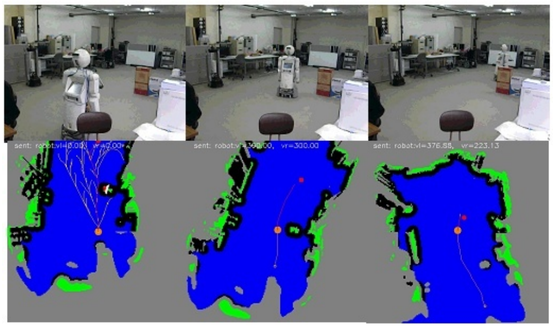


Fig. 7: Experiment of following waypoints, (from left to right) sequences of the real world (top) and local map scene (bottom)

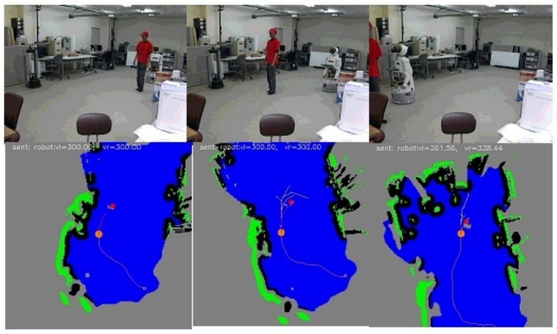


Fig. 8: Experiment of people tracking, (from left to right) sequences of the real world (top) and local map scene (bottom)

B. Experiment on The Real Robot

We use an ENON robot by FUJITSU for experiments, equipped with a stereo camera (Bumblebee2 by Point Grey), a laser range finder (by Hokuyo), and laptop PC. We test HeAT Random Tree path planner using two problems; following waypoints and people tracking problems.

In the first problem, we determine several waypoints which have to be reached by the robot on the real world. For people tracking problem, the robot has to follow a person using a people tracking algorithm [13].

We utilize a 200x200 grids of probabilistic local map with actual size of 10x10 meter. For real implementation, we introduce Unknown Area $\boldsymbol{U}$ as mentioned in section II. This area (grey area in Fig. 7 bottom and 8 bottom) appears due to the limitation of the sensors.

Fig. 7 is a screenshot of the robot following waypoints. The robot has to reach the sequence of destinations safely by avoiding any obstacle. Fig. 8 is a screenshot of the people tracking problem. In this problem, the robot has to follow a person while checking its environment. Both following waypoints and people tracking tasks are successfully done within 500 milliseconds of computation time per cycle and using maximum speed of 400 millimeters per second. Those experiments show the ability of HeAT algorithm to be directly implemented on the real robot using a real environment.

VI. Conclusion

We have presented a novel path planning algorithm which uses the *arrival time field* as 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 against other existing path planner algorithms. Simulation and experimental results also show that our algorithm is applicable to the real robot, and can be used real-time.

Several improvements are possible to be applied. One of them is to integrate the moving obstacle model to the arrival time field so that we will get *3D time-space* model of moving obstacles for a more reliable and efficient path. By using 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.

VII. Acknowledgment

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] J. Bruce and M. Veloso. "Real-Time Randomized Path Planning for Robot Navigation", in Proc. IEEE/RSJ Conf. on Robotics and Systems, 2002.
[2] M. S. Hassouna, A. E. Abdel-Hakim and A. A. Farag., "PDE-based robust robotic navigation," Image and Vision Computing, vol. 27, pp. 10-18, 2009.
[3] S. M. LaValle and J. Kuffner. "Rapidly-exploring random trees: Progress and prospects". In Proc.of Fourth Intl. Workshop on Algorithmic Foundations onRobotics (WAFR'00), 2000.
[4] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS), 2003.
[5] S. M. LaValleand J. Kuffner. "Randomized kinodynamic planning". In Proc.IEEE Int. Conf. Robotics and Automation (ICRA), pages 473-479, 1999.
[6] S. Karaman and E. Frazzoli. "Incremental sampling-based optimal motion planning". In Robotics: Science and Systems (RSS), 2010.
[7] L. Jaillet, J. Cortes, and T. Simeon, "Sampling-Based Path Planning on Configuration-Space Costmaps," IEEE Transactions on Robotics,vol. 26, no. 4, pp. 635-646, Aug. 2010.
[8] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, The MIT Press, 2005.
[9] M. Zucker, J. Kuffner, and J. A. Bagnell, "Adaptive workspace biasing for sampling-based planners". In Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 3757-3762 (2008).
[10] E. Plaku, L. E. Kavraki, and M. Y. Vardi. "A Motion Planner for a Hybrid Robotic System with Kinodynamic Constraints". In Proc. of the 2007 IEEE Int. Conf. on Robotics and Automation, pp. 692-697.
[11] S. Rodriguez, X. Tang, J.M. Lien, and N.M. Amato. "An obstacle-based rapidly-exploring random tree". In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 895-900, 2006.
[12] A. Shigemura, Y. Ishikawa, J. Miura , and J. Satake, "People Movement Simulation in Public Space and Its Application to Robot Motion Planner Development", Proc. 2010 Int. Conf. on Advanced Mechatronics, pp. 504-509, Osaka, Japan, Oct. 2010.
[13] J. Satake and J. Miura , "Robust Stereo-Based Person Detection and Tracking for a Person Following Robot", Proc. ICRA-2009 Workshop on Person Detection and Tracking, Kobe, Japan, May 2009.
[14] Ando, N., Suehiro, T. , Kitagaki, K., Kotoku, T. , Yoon, W.K.: RT-middleware: Distributed component middleware for RT (robot technology). In: Proceedings of 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 3555-3560, 2005.