The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

High-Speed Planning and Reducing Memory Usage of a Precomputed Search Tree Using Pruning

Yumiko Suzuki[1,2], Simon Thompson[2] and Satoshi Kagami[2,1]

***Abstract*— We present a high-speed planning method with compact precomputed search trees using a new pruning method and evaluate the effectiveness and the efficiency of our precomputation planning. Its speed is faster than an A* planner in maps in which the obstacle rate is the same as indoor environments. Precomputed search trees are one way of reducing planning time; however, there is a time-memory trade off. Our precomputed search tree (PCS) is built with pruning based on a rule of constant memory, the maximum size pruning method (MSP) which is a preset ratio of pruning. Using MSP, we get a large precomputed search tree which is a reasonable size. Additionally, we apply the node selection strategy (NSS) to MSP. We extend the outer edge of the tree and enhance the path reachability. In maps less than 30% obstacle rates on a map, the runtime of precomputation planning is more than one order of magnitude faster than the planning without precomputed search trees. Our precomputed tree finds an optimal path in maps with 25% obstacle rates. Then our precomputation planning speedily produces the optimal path in indoor environments.**

I. INTRODUCTION

To accept mobile robots in human-inhabited environments, it is necessary for mobile robots to offer services for humans in a timely manner. A* grid-based search is a commonly used algorithm for path planning that can find the optimal path through a search space. A* search algorithms in artificial intelligence [1] have been typically applied to path planning problems using grid-based maps for moving robots. Path planning with grid-based shortest path algorithm using A* or Dijkstra searches all adjacent grid cells of a current node, so it computes the optimal and complete trajectory [2]. However, as the map size grows larger, these algorithms have the problem of spending an exponential amount of time collision checking against static obstacles.

The idea of precomputation of the data and processes that take a long time is especially used in the field of the image processing and the creation of a motion with computer graphics and vision [3], [4], [5], [6], [7]. This idea isn't particularly new; in other words, as preprocessing, it is used to create a model using a huge dataset [8].

In the planning problem area, [9], [10] used the preprocessing of configuration space for path planning for robots with many degrees of freedom and for articulated robots moving in static environments. This method used a preprocessing to speed up planning by paying a preprocessing cost.

1: Faculty of Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma-shi, Nara, 630-0192, Japan. `yumiko-s@is.naist.jp`

2: Digital Human Research Center, National Institute of Advanced Industrial Science and Technology, 2-41-6, Aomi, Koto-ku, Tokyo, 135-0064, Japan. `{simon.thompson, s.kagami}@aist.go.jp`

In this paper, the amount of node expansions is determined by examination. It used a heuristic scheme and tried to add more effective nodes, and this prespecified number reached about six thousand. It was also mentioned that these enhanced precomputations to enlarge the node graph are expensive.

The key idea that to precompute a search tree is a practical way to reduce planning time [11], [12] was implemented in motion generation for animation [3]. One property of precomputed search trees is that collision checking is not greatly required while growing the nodes. Precomputed search trees are one of the ideas for reducing planning time; however, it is a time-memory trade-off. Creating an exhaustive tree built from an FSM [13] with 21 behavior states required 220,000 nodes. However, this huge tree consumes a great deal of memory. A pruning method was used as a way to reduce the memory usage.

Our precomputed search tree is built with pruning based on a rule of constant memory. We have acquired a compact precomputed search tree which is a reasonable size using a new pruning method. Moreover, our precomputed search tree quickly plans the optimal path in indoor environment maps.

This paper describes high-speed planning with a precomputed search tree using a novel technique to reduce planning time and memory. In Section II, we describe the precomputation search with an A* Planner. Section III shows our new pruning technique. Furthermore we define the dispersion in real indoor environments in Section IV. In Section V, we present the effectiveness and high-speed performance of our precomputation planning based on experimental results.

II. PRECOMPUTED SEARCH TREES WITH AN A* PLANNER

This section describes the branches of the extended A* planner which is grown to 5 neighboring gridcells to create our precomputed search tree.

A. Extended A Search*

A* grid-based search is a commonly used algorithm for path planning that can find the optimal path through a search space.

The exhaustive growing of branch nodes will consume exponential time for collision checking the growing branches and use a huge amount of memory to store this exponentially grown tree. It has a problem increasing the time of a collision checking for static obstacles as the search map size becomes bigger. We will reduce the exponential time for collision checking by growing branches using precomputed search trees.

We make a precomputed search tree using a path planning with an efficient version of extended A* grid-based search [14]. The extended A* which is a simple technique is described to speed up optimal path planning on Euclidean-cost grids and lattices. The technique applies to grids and lattices with edges between diagonally-adjacent grid points whose relative costs obey the triangle inequality. Then we use the 5 neighbors branch (Fig.1) based on the extended A* planner to create the precomputed search tree.

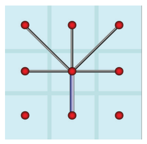

Fig. 1. Branch of Extended A*
we use these 5 neighboring branches based on the extended A* planner to create the precomputed search tree

B. Precomputed Search Trees with A* Planner

We build a tree of the A* search up to a certain depth. If we build the complete tree, its size increases exponentially and its memory usage exceeds the upper limit of the physical memory size. The total number of tree nodes is expressed by Equation (1) with depth level d.

$$The\ total\ number\ of\ nodes = \sum_{k=0}^{d} BranchFactorCount^{k} \quad (1)$$

Whereas Lau et al. [3] is pruned after building the exhaustive tree, we prune the tree at the same time as growing branches automatically. Using our pruning method, it is necessary to manually reduce redundant nodes. We will explain our pruning method in greater detail on Section III. After building the tree, we set a start position in the environment map on the root node of the tree and a goal position on a branch node that is in geometrical relation from the root node. Each node of our precomputed search tree has a position, a link to a parent node, a state flag and cost as the path length up to that node starting from the root node. Each node has only one parent, then the precomputed planner can trace back the path to the root node. We use the tree map to manage the precomputed trees and the nodes that came from every direction. When backtracking the nodes, our planner checks for the collision between a node and the obstacles on the environmental map.

C. Path Finding: Alternate Branch BackTracking

In the paper [3], the path finding algorithm takes a goal position as input, and returns the tree node that represents the solution path. As they trace back towards the root (Fig.2 Top), they mark each node with the blocked flag, if it is not already blocked or not obstructed by an obstacle (the red circle in Fig.2 Top). The one position on the tree has the list of sorted nodes towards the root node. If the node is obstructed the an obstacle (1 in Fig.2 Top), they set the blocked flag all tracked back nodes and select the next node to track in the list of the goal (2 and 3 in Fig.2 Top). But this returning to the list of the goal consumes the time unfortunately, if the tree has a massive number of sorted nodes on each node position.

We introduce new backtracking which is named " Alternate Branch BackTracking (ABBT) " to resolve the time consuming problem above. If the node hits an obstacle (1 in Fig.2 Bottom), we get the next tracking node from the list of neighbors node which is a predecessor of the obstructed node. The main contributions of "ABBT " includes reducing the flag setting time and cutting down on the repetition of returning at the list of the goal for selecting a next node to start fresh.

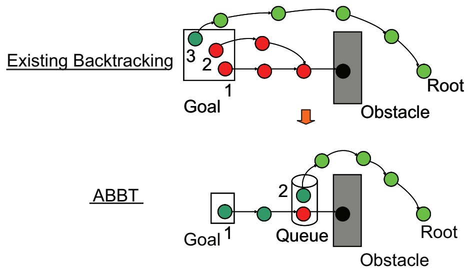

Fig. 2. Alternate Branch BackTracking
In our backtracking method (Bottom), the planner selects the next node from the list of neighbors node which is a predecessor of the obstructed node with an obstacle when starting fresh.

III. Pruning Method

This section describes a method to create a practical precomputed search tree with pruning based on a ratio, maximum size pruning and a technique for search enhancement, the node selection strategy.

A. MSP: Maximum Size Pruning Method

The new pruning method is named "*maximum size pruning (MSP)*," which uses a preset ratio for pruning. Using this preset ratio, it is not necessary to manually reduce redundant nodes after growing branches and to get a large precomputed search tree, which is a feasible memory size in current computers. Our search tree grows in depth first order and this method uses the memory usage ratio for each depth to control the growing node count at each depth. The preset ratio is calculated back based on the system memory size and a node size. We predefine the maximum memory size of each generation. Reducing the total memory usage, the ratio is set for pruning more accumulated points as the node counts of deep depth increase exponentially.

We use a random number to decide the node to prune with the presetting above. Mersenne Twister (MT) ([15], [16]) is used as a random number generator in our pruning method. The MT pseudo-random number generator is a long period random generator and is distributed evenly over high-dimensions. MT brings the same pattern random number, and we constantly get a pruned tree with the same rate and geometry using this random number generator.

This preset is defined as beginning the pruning at the 4th generation and 20 [KB] of memory usage at each generation as the ratio of the maximum memory size in this research. Futhermore, one node of this tree occupies 20 bytes of memory and the tree grows branches for a depth of 40. The length of one branch is about 1 gridcell. One grid cell is 10 cm in a real environment. Figure 3 shows the configuration of a PCS expanded on a 2-dimensional space; the z-axis refers to the number of nodes that reach each position of the grid. The shape of the branch is the extended A* [14] that has five branches in one step. The total memory usage of this tree is about 15 [MB]. If the tree grows all branches without pruning, the memory usage for retaining all node data is 3.6×10^{20} [GB]. Using MSP, we have successfully realized a lightweight tree which is diminished to 4.1×10^{-24} times the size of the fully expanded tree. Our pruning method is excellently efficient at reducing the memory usage compared to Lau's result [3] that reduced the tree to 3% of the original tree size.

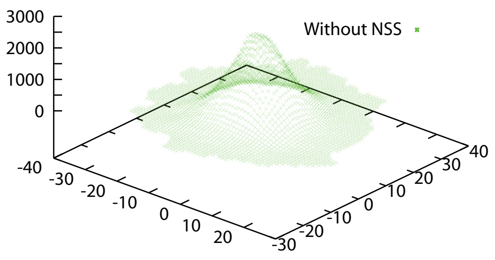

Fig. 3. Precomputed Search Trees with a depth of 40 grown using MSP

B. NSS: Node Selection Strategy

Our search branch spreads radially in five directions. Several nodes grown from different parent nodes reach the same position of the grid. Therefore, the grown tree acquires a heaped central shape (Fig. 3), and the tree starts to extend a branch on the center of the precomputed search tree. These turning-around expansions decrease the amount of nodes that will be far from the root position. Even though it uses the other branches in the grid-based planning, search branches extend 4 or 8 directions on the grid map and the shape of the precomputed search tree will resemble a mountain as in Fig. 3.

To improve this predisposition to heaping, we introduce the Node Selection Strategy (NSS), which is a node selection algorithm to put a node in the open list for precomputation using MSP. The NSS is based on the distance from the root node and the node exchange ratio between pruning and not pruning, which is preset stochastically for nodes in the same generation. The pseudo code of the NSS is Table I.

The node exchange ratio that is used by R in the pseudo code above can be modified arbitrarily before growing the precomputation tree. An exchange ratio of 1:2 means that one node that will turn around will be pruned and two nodes will grow at the far end from the root node. We show the PCS with the MSP using the NSS in **Fig. 4**. The green heap is the PCS with the MSP not using the NSS, and the red one

TABLE I
NODE SELECTION STRATEGY

```
Pseudo code Node Selection Strategy
Performs the Precomputed Search Tree with MSP
Data: n: the nodes of the extending branch
          R: the exchange ratio
Result: a boolean value indicating pruning

Distance = GETDISTANCEFROMROOT(n)
Threshold = n.Generation * n.StraightLength

if STATISTICALPRUNING (n) then
   ifDistance ≥ Threshold then
      if ISEXCHANGE (R) then
         else return false;
   else return ture;
   end
else
   ifDistance < Threshold then
      if ISEXCHANGE (R) then
         else return true;
   else return false;
   end
end
```

is using the NSS with an exchange ratio of 1:2 . The effect of the NSS changes the form of two parts of the trees. First, the heap in the center of the PCS using the NSS (the red one in Fig. 4) decreases compared with not using one (the green in Fig. 4), and its maximum height becomes lower than without the NSS. Second, the verge of the tree using NSS expands farther than the other. Because of the even exchange ratio used by the NSS and five factors of one branch, the memory usage increases; in the case of Fig. 4, its size is 117 [MB]. Using this NSS, the number of nodes in the central heap decreases by about 35,000 total and the distance from root node reaches out about 19 [gridcells] at the maximum. The maximum number of increasing nodes at verge of the tree is about 5,000. The improvement in finding paths to far goals by this strategy will be shown in Section V.

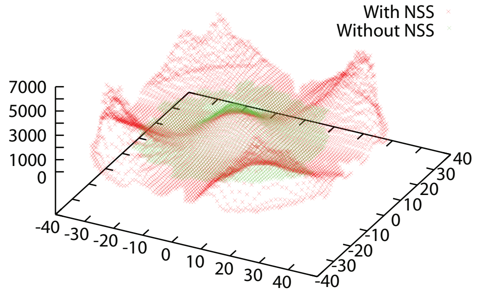

Fig. 4. Precomputed Search Tree with NSS

IV. DISPERSION IN REAL INDOOR ENVIRONMENTS

We defined the dispersion of real indoor environments with a grid based map.

The dispersion is expressed by the "*obstacle rate*," determined by the percentage of the number of obstacle cells divided by the total cell count in a map.

Figure 5 shows the grid based map which is generated by the mapping data using a laser range finder on the robot in one floor of a building. Its grid resolution is 10 [cm] and the map size is 1310×393 [cells].

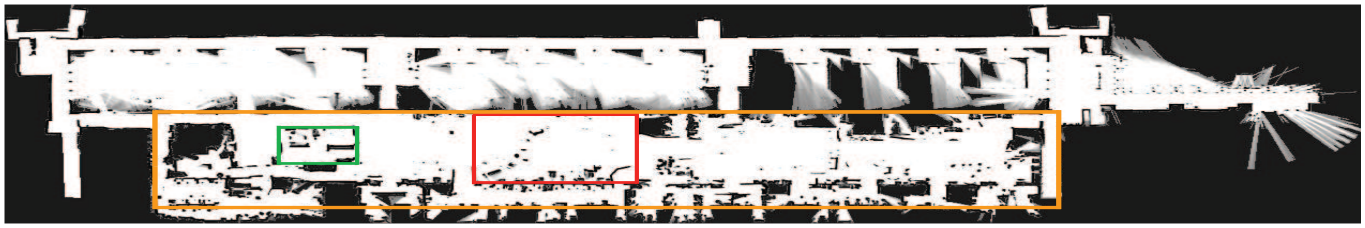

Fig. 5. Floor Map

The area framed by green lines is the map of the living room (Fig.6) and its obstacle rate is 5.1 %. The other area enclosed by a red rectangle is an experimental station and its obstacle rate comes out to 1.8 %, and the obstacle rate of the other area delimited by a big orange rectangle is 4.5 %. The obstacle rate of this total floor map (Fig.5) is 4 %. Therefore, it is likely that the obstacle rate of real indoor environments is less than 5%.

Fig. 6. Living Room and its Grid-Based Map

V. Experiments

In this section, we describe the experimental results to verify the accuracy of A* planner using PCS and the effect of MSP and NSS. Additionally, we confirm the speed-up and path optimality comparing an A* planner with the precomputation planning.

A. Experimental Setup

In this experimental simulation, a PCS is made with forty generations. We put the root node at the center of the map and let trees grow using A* which has five branches. The length of one branch is about 1 gridcells. PCS is applied to the prune processing based on the predefined tree limits of 20 [KB] each generation after the 4th generation. The experimental map is a square map of 55 grids on one side, which is similar to living room size in real environments (ex. Fig.7). The map includes static grid obstacles. The number of obstacles in a map was based on the predefined ratio to the environmental map size. We prepared experimental maps with obstacle ratios from 0 % to 31 % according to the dispersion that we define in Section IV and we placed the obstacles scattered with random numbers on the map. The starting position of planning was specified as the origin point of the experimental map, and we randomly selected about 700 goal positions on the map. We use the ABBT in this experiments.

The CPU was used for these experiments the Intel(R) Core2 X6800 2.93 [GHz] with 4 [GB] of memory and the simulator runs on Linux. All of the experiments below were executed with the same hardware and software.

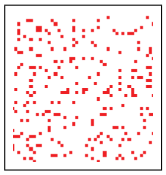

Fig. 7. Random Number Map with 10% Obstacle Rates

B. Success Rate: Effect of MSP

This subsection describes the execution of a precomputed search tree that was pruned using MSP.

Figure 8 shows the result of the path planning using PCS with MSP, giving the percentage of goals reached for random goals. We call this the "*success rate*." In Fig.8, the vertical axis is the "*obstacle rate*" and the horizontal axis is the distance from the start node. The success rate is shown by the difference in color. The bright yellow means 100% of the attained level, and the darker color, the lower success rate. The light blue vertical line drawn at the horizontal axis indicates the position of the beginning of pruning which converts the generation based on MSP settings into the distance of the path.

Even though the precomputed tree is incomplete being pruned by the MSP, the planner reliably generates a path and its success rate is 100% for an obstacle rate of 0% to 31% on a map and up to 27 grids of the distance from root position. As Fig.8 shows, the success rate decreases in the case of a large distance on a map. It is believed that the inefficient spread nodes caused by the turning around expanded branches decrease the nodes at far from the root node.

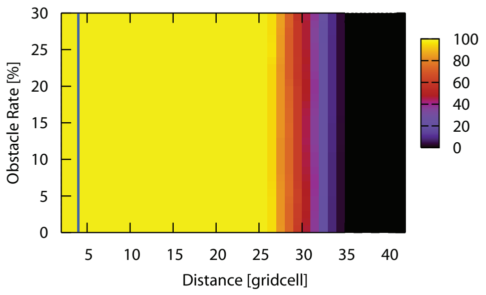

Fig. 8. Success Rate of the Precomputation Planning using MSP

C. *Improved Success Rate: Effect of NSS*

This subsection shows the effect of the strategic selection of nodes NSS applied to MSP.

In this experiment, the exchange ratio of NSS is 1:2. The other settings for goal counts and maps is the same as above. Figure 9 shows the result of the success rate of the path planning using PCS applied NSS to MSP. Comparing the success rate using NSS (Fig.9) with its rate not using NSS (Fig.8), After 27 grids of distance in the Fig.8, the color of the success rate is brightens than its results without NSS. Using NSS to correct the success rate, it especially contribute to improving its performance on the map with much obstacles and on the far goal points. The success rate with NSS is 100% from 0% to 31% of the obstacles rate on a map, whereas the success rate is 0% in a case of the unused NSS. Applying NSS to MSP, therefore, it is useful for upgrading the performance of the path planning using a precomputation tree.

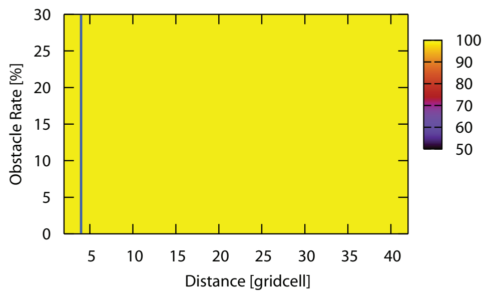

Fig. 9. Success Rate Improved with NSS

D. *Limitation of PCS*

Figure 10 shows the success rate using PCS applied NSS to MSP is the same as in subsection V-A above. In this experiment, the square map is 65 [gridcells] on one side; the goal area is wider than PCS. The reach of the PCS from the root node is 45 [gridcells]. Our planner finds paths to the goals within the reach of the tree.

Under a map obstacle rate of 37% on a map in the current work, the success rate is 100% within the distance of 45 [gridcells]. The obstacle rate of real indoor environments is typically less than 5%; hence the precomputation planning creates a path in the real world and its success rate is 100%.

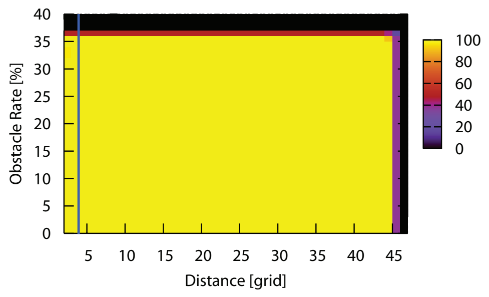

Fig. 10. Success Rate in the case of Big Map

E. *Speeding Up of Searching*

This subsection relates the effect of the speeding up of searching for our precomputation planning.

Figure 11 shows the extracted results of the planning time for the experiment that is executed as a subsection V-C above and the runtime of the path planning without any precomputational techniques. It is a semi-log plot of planner runtime; the vertical axis is planning time in milliseconds and the horizontal axis is the distance. The red and pink lines show the runtime of the A* planning in conditions of 0% and 31% of obstacles rate on a map; the others indicate the precomputed planner runtimes of 0%, 5%, 30% and 31% obstacle rate. The value of the precomputed planner runtime less than or equal to 30% is constantly below the value of the A* planning. In those case the precomputation planner runtime can increase. We found that the planning using our searching method is sped up compared to the A* planning which does not use precomputed trees as seen in Fig.11. In maps with less than a 30% obstacle rate on a map in this experimental condition, the runtime of precomputation planning is more than one order of magnitude faster than an A* planning method without precomputation.

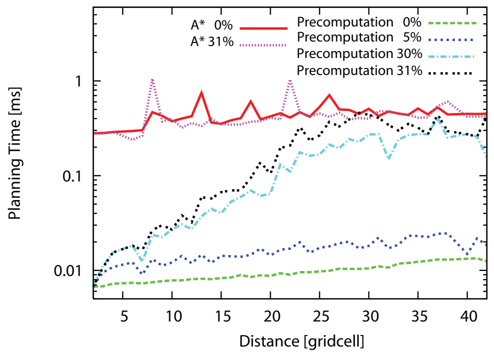

Fig. 11. Computation Time of Planning

F. *Path Optimality*

This subsection describes an evaluation of the optimality of the paths that are provided by the precomputation planning.

The optimality is defined by the percentage of the path length of the precomputation planning as compared to the path length of the existing planning. The extended A* planning [14] produces the shortest path, then we use its path length as the reference value when the optimality value is calculated. The experimental setting is the same as the above V-A and using 700 random goals. Figure 12 shows the path optimality for every obstacle rate from 0% to 30% on the experimental maps. The horizontal axis is the obstacle rate and the vertical axis is the path optimality. The blue dot is the average path optimality and the dotted-line is the standard deviation. The red dot is the mode value.

As shown in the average optimality in Fig.12, the precomputation planning finds a path which is the same in length as the shortest path on the maps with less than 25% obstacle rate. On maps with 26% obstacle rate to less than 29%

obstacle rate, our planner finds a path which is an average less than 115% of the length of the shortest path.

Every mode value is 100% of the length of the shortest path in the all trials.

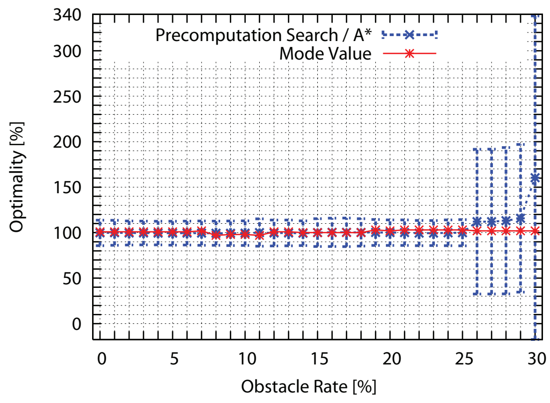

Fig. 12. Path Optimality of the Precomputation Planning with A* Planner

VI. Conclusions and Future Work

In this paper, we have presented a high-speed planning method with compact precomputed search trees using a new pruning method and shown the effectiveness and high-speed performance of this precomputation planning. Its speed is faster than an A* planner in the maps in which the obstacle rate is the same as indoor environments. Our precomputed search tree is built with pruning based on a rule of constant memory, the maximum size pruning method (MSP), which is a preset ratio of pruning. Using MSP, we achieved a large precomputed search tree that is of reasonable memory size. Additionally, by applying the node selection strategy (NSS) to MSP, we extended the outer edge of the tree and enhance the path reachability.

We conducted experiments and analyzed the success rate, the speeding up of path searching and the path optimality. As shown in the experiments, our precomputed search tree finds the paths in the map at an obstacle rate of 31%. When the goal position was even set on an incomplete tree which is pruned, the precomputation planning could successfully finds a path. We evaluated the speed and the path optimality comparing the result of the A* planning to its of the precomputed searching. In the current work, under an obstacle rate at 31% on a map, the runtime of precomputation planning was more than one order of magnitude of faster than the planning without a precomputed search tree. We found paths that were as same in length of the shortest path on the maps with less than 25% obstacle rate. Every mode value was 100% of the length of the shortest path in all the trials. The obstacle rate of real indoor environments is less than 5%; hence the precomputation planning quickly acquires a path in the real world and its path optimality is 100%. Then our precomputation planning produce speedily the optimal path in real indoor environments.

In these experiments, we used a tree with depth of 40 and a square map of 55 gridcells on the each side, which is similar to a living room in a real environments. We made its grid resolution 10 [cm] in our experiment, but the map resolution is variable. Therefore, our new planning may have applicability to a huge environment map. Since we know the performance of the precomputed search tree, we will estimate its runtime and its path optimality in other environments. Also, we will speedily obtain the optimal path in a vast environment not using one big tree but using the compact tree repeatedly.

In future work, we will apply the precomputation planning to a huge environment and implement our planning on the real robot (Fig.13).

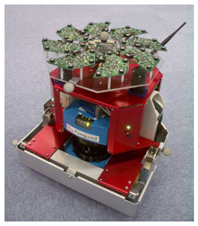

Fig. 13. Real Wheeled Robot

References

[1] N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[2] Steven M. LaValle. *Planning Algorithms*, chapter 5, pp. 185–186. Cambridge University Press, 2006.

[3] Manfred Lau and James J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pp. 299–308, September 2006.

[4] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics (SIGGRAPH 2003)*, Vol. 22, No. 3, pp. 879–887, July 2003.

[5] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. *Graph. Models*, Vol. 68, No. 2, pp. 158–174, 2006.

[6] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph.*, Vol. 22, No. 3, pp. 376–381, 2003.

[7] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. Bi-scale radiance transfer. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pp. 370–375, New York, NY, USA, 2003. ACM.

[8] Henry Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 1, pp. 23–38, January 1998.

[9] Lydia Kavraki and Jean-Claud Latombe. Randomized preprocessing of configuration space for path planning: Articulated robots. In *IEEE/RSJ/GI International Conference on Intelligent Robtots and Systems(IROS)*, pp. 1764–1772, 1994.

[10] L. E. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. pp. 2138–2139, San Diego, CA, 1994. IEEE Press.

[11] Jared Go, Thuc Vu, and James J. Kuffner. Autonomous behaviors for interactive vehicle animations. *International Journal of Graphical Models*, 2005.

[12] Emilio Frazzoli, Munther A. Dahleh Y, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 25, pp. 116–129, 2002.

[13] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pp. 271–280, August 2005.

[14] James J.Kuffner. Efficient optimal search of euclidean-cost grids and lattices. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, September 2004.

[15] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, Vol. 8, No. 1, pp. 3–30, 1998.

[16] Mutsuo Saito and Makoto Matsumoto. *Monte Carlo and Quasi-Monte Carlo Methods 2006*, chapter 2, pp. 607 – 622. Springer Berlin Heidelberg, 2008.