

GO: Simplified Strategy to Control the Robot in the CiberMouse@RTSS2006 Competition

Jenny Zhinan Han

Song Wei

Albert Cheng

zhan@uh.edu

1983.song.wei@gmail.com

cheng@cs.uh.edu

Computer Science Department, University of Houston, Houston TX 77204, USA

Abstract:

This paper is a brief description of the strategies used in robot "GO" in the Ciber-Mouse robot competition. The challenge we face is: control robot to start from starting point and visit the target area and then return to the starting point. In order to fulfill this task, the paper comes up a non-history fully searching environment exploration to find the target. Besides, we also use virtual Cartesian coordinates to record robot's location in order to do backtrack when returning to the starting point.

1. Introduction

Ciber-Mouse is a competition among virtual robots whose goal is to reach the target area and return to the starting point with no collision. The role of the software is the agent of the robot which provides control of the robot in order to achieve the goal.

In the previous robot contests, there are bunch of accomplishments such as described in [2], [3], [4] and so on.

However, there are some environmental differences in this year's competition. 1. GPS is disabled in the contest which means we cannot get the robot's position in the world. If we could get the robot's starting position in the world, then we can use compass information to update the position of the robot [2]. 2.

There is no map of the labyrinth and we don't know the position of the target area. Moreover, we can get the relative angular position of the target area only if it is in the $-60e+60$ angular range of the robot without any obstacle between them. If we have GPS or know position of the target, we can use strategies described in [2], [3] to achieve the goal.

Because the position information is so limited, we have to nearly fully explore the labyrinth until we find the target. In order to avoiding dead end problem, we use a non-history strategy which means the explored space is only traversed a second time when there is no alternative [2]. After get to the target area, the robot still needs to return to the starting

point which has no beacon. So we record the positions of critical points in the labyrinth based on a virtual Cartesian coordinate which marks the starting point as (0, 0). So that we can get the positions of the critical points like obstacles and target area relative to the starting point. By using these positions information, the robot records the path to the target and then backtracks to the starting point. In this paper, we introduce an overview of the simplified strategies in the second section. Designing details are listed in section 3 (about environment exploration strategy), section 4 (about start point returning strategy) and section 5 (about collision avoiding and exception handling). Finally we conclude our work in Section 6.

2. Strategy Overview

At the beginning the robot may not even know the direction of the beacon relative to the starting point, because there probably are some obstacles between the starting point and the beacon. So the robot starts to run fully exploration.

We set a virtual Cartesian coordinate on the grid with origin (0, 0) on starting point and X axis facing the virtual North.

For each cell in the grid, we assign a status "free" or "obstacle" or "unknown" to it and when the robot traverses a cell, it will be given a "free" or "obstacle" mark. If the mark of a cell is not "unknown", the robot intends not to traverse it again except that there is no other way to go.

At each step, it should return around to detect the beacon in all possible

direction, because at each pose of the robot, it can only detect beacon in range of -60° to 60° .

If it cannot find the beacon in any direction, the beacon must hide behind some obstacle. So the robot sets the cell "free" and checks the mark of next cell in direction of west, north, east and south. If there is a cell's mark is "obstacle", the robot will go following the walls. Or else, it will go to the "unknown" cell in the priority of west, north, east and south which means if the next western cell of the current cell is "unknown", the robot will go west one step, otherwise, it checks the next cell to the north of it and keeps doing that until it find a "unknown" cell to go or none of cells in four directions is "unknown". Keep doing this, the robot will sometime reach the wall or obstacle. And then by following the wall method, it will round the obstacles between starting point and beacon. So that it finally can detect the beacon.

After the beacon being detected, the robot walks directly to the target area and turns on the beacon led in the target area.

The second part should be the returning trip. Now we already get the position information of the starting point and target area, the compass information and the status information of each cell in the way coming to the target area. With this information we can build a maze map and derive the returning path from it. The work of part is still in hand.

3. Environmental exploration

The environment exploration is based on dynamically updating a

map along the exploration process. In order to deal with different scenarios and adaptively evolve our algorithms, we decide to use a state machine to choose different path finding algorithms. When there is no detected beacon signal and no knowledge about the maze, following wall is of course a good point to start with, and we'll tuning it a little bit in terms of the turning at the end of the wall and the wall scanning algorithm to cope with most of the test maze. With information recorded in the map, the best point that can lead to the beacon on the boundary will be calculated. If our wall following based algorithm is good and we're lucky, the best point will be not far away from where we are. In this case, we just need a reactive approach to explore new areas from there. If we find that we're far away from a good location, we can use path finding algorithm to navigate to the point. After we got the beacon signal, we need to change the algorithm (switch to another state), go straight to the beacon. We think the most important issue in this process is scanning: By carefully tuning the scanning algorithm, we can determine the surroundings in a few turns. And also that we haven't received beacon signal is also information for us. Combined with obstacle information, we can conclude that the beacon doesn't exist in a certain area. This enable us to focus on the remaining area. However, other robot might be scanned as obstacles also. There are two ways to validate our scan, one is to combine the obstacle context together to make our decision, the other is to scan the same location multiple times at

different time, since the other robots are moving most of the time. The latter require us to identify the change in scanned scenario. However, combining information scanned at different location together is also an effective way to quickly build an accurate map.

4. Start point returning

the start point is the origin of our coordination system. We'll find the shortest path based on the path we've build. In this process, however, since maybe we'll choose a better way than the way we got to the beacon, we can add more information to the map and continuously update it and better the path. Like in the beacon finding phase, we also need to take care of collision with other robots.

5. Collision avoiding and exception handling

Most part of the game is to solve a maze, and advanced techniques for this are planning ones. However, I think the basic solution for collision avoiding and exception handling can only be reactive. The difficulty lies in combining the reactive handling with planned strategy. Also, when doing collision handling we need to identify if we're going to hit walls or we're going to hit robots. If we're going to hit walls, we'll stop in our direction and update the map with the new information. If we are going to hit robots, then possibly we are on the right path, and different strategies could be applied here. For simplicity,

I think we'd better not try to avoid others when we don't have to, since others will try to avoid us. And avoiding others will break your original plan for at least a while. This is different in exception handling, since we need to handle exceptions as actively as possible, by first invoking special behavior routines to save us from immediate danger and then reevaluating the whole plan.

6. Conclusion

Our basic strategy is : follow wall when we can find the way to the beacon, build path with maximum utilization of scanning we can achieve, and apply appropriate path finding algorithm to find the shortest path, and deal with everything else on a case by case basis.

However, we realize that we should be always learning and adopting new techniques in the development. So we'll use the state machine model and component model to build an extensible framework, and develop different component and algorithms along the way, trying to go as far as we can.

Reference:

[1] "Ciber-Mouse Rules and Technical Specifications", RTSS, July, 2006,
http://www.ieeta.pt/~lau/web_ciberRTSS/docs/ciberRTSSrules.pdf

[2] L. Seabra Lopes, N. Lau, L.P. Reis (2000). "Intelligent Control and Decision-Making demonstrated on a Simple Compass-Guided Robot". IEEE SMC'2000, USA.

[3] Santos, V. Silva, L. Almeida (2002). "A robust self-localization system for a small mobile autonomous robot". ISRA 2002, Int. Symp on Robotics and Automation, Toluca, Mexico.

[4] E. Martins, P. Pedreiras (1999). "Jerry: A jolly ElectRic Robot, Yo!" Revista do DETUA, 2(6)