Path Planning and Navigation for Geeves: A Tour-Guide/Greeter Robot

Viraporn Vongsoasup
Bucknell University
Lewisburg, PA 17837
Email: vvongsoa@bucknell.edu

Maja J. Matarić
Computer Science Department
University of Southern California
Los Angeles, CA 90089
Email: mataric@usc.edu

Abstract

This paper focuses on path planning and navigation for Geeves, a tour-guide/greeter robot at the University of Southern California Interaction Lab/Center for Robotics and Embedded Systems. First, we begin with a brief description of the path planning and navigation problems in mobile robotics. After that, we focus on how Geeves navigates around the robotics lab. Specifically, Geeves uses a previously implemented path planner, containing a *wavefront* controller combined with the *vector field histogram* (VFH), to move around obstacles in the environment. The path planner ensures that the robot maintains a safe distance away from obstacles, while finding an optimal path from each destination to the next. Simulated experiments reveal that this system safely and effectively navigates Geeves around the Interaction Lab.

1. Introduction

Path planning and navigation is a frequently studied problem in mobile robotics. Planning algorithms can be grouped into two categories, *local* and *global*. The *local* navigation problem deals with navigation over short distances of a few meters, where the main problem is obstacle avoidance. *Local* navigation allows the robot to move in its immediate environment and to reach local goals without colliding into obstacles. A well-known solution to this problem involves creating an occupancy grid of the immediate surroundings around the robot. The robot uses this grid to determine its navigation direction towards a goal [1], [2].

The *global* navigation problem, on the other hand, deals with navigation on a larger scale in which the robot cannot observe the goal state from its initial position. It detects significant landmarks and, in most cases, uses a global map to determine its location in the environment. *Global* navigation allows the robot to reach distant goals specified according to the global map. One solution to the *global* navigation problem is to create a pre-specified map or a map on the fly. Other approaches rely on some technique of localization, such as Monte carlo localization [3] and Markov localization [4].

We apply both *local* and *global* navigation on Geeves, a tour-guide/greeter robot at the University of Southern California Center for Robotics and Embedded Systems. Geeves performs local navigation using an occupancy grid of the robotics lab and the *vector field histogram* (VFH) [5]. Geeves navigates globally by implementing a wavefront algorithm in Player [6] and Adaptive Monte Carlo Localization (AMCL). The next sections describe all of these applications in more detail.

2. Local Navigation

A file, containing the locations (x, y) and orientation (θ) of various stops along the tour, is fed as input into our program, which is a client to the Player server [6] (more information about Player is included at the end of this report). The program also takes as input an occupancy grid of the lab. The occupancy map, shown in Figure 1, only displays the main pathway that Geeves traverses on the tour, which does not include any cubicles and empty spaces between each cubicle. The reason for this is because Geeves only makes stops on destinations along the main pathway constructed on the grid.

In addition to the file and map, Geeves uses VFH [5] to detect unknown obstacles while simultaneously steering itself towards each stop on the tour. Briefly, the VFH method uses a two-dimensional Cartesian histogram grid as a world model. The histogram grid is reduced to a one-dimensional polar histogram that is constructed around the robot's momentary location. Each sector in the polar histogram contains a value representing the polar obstacle density in that direction. The algorithm selects the most suitable sector among all polar histogram sectors with a low polar obstacle density, and the robot is aligned in that direction.

3. Global Navigation

Geeves navigates globally around the lab by implementing a wavefront algorithm. The wavefront algorithm creates a grid of squares each with a value of zero after it is given a map of the surrounding area. Next, all obstacles are marked on the grid as occupied squares. Any squares near the occupied ones are given a high number depending on how close they are to the obstacle. The wavefront driver then places a value on every square adjacent to the goal and then a slightly higher value on all squares adjacent to the first squares, until all the squares have a value determined by their distance to the goal and any nearby occupied squares. The algorithm then selects the lowest square near the robot and moves there, which simply creates the shortest path to a goal. An example of wavefront is illustrated in Figure 2.

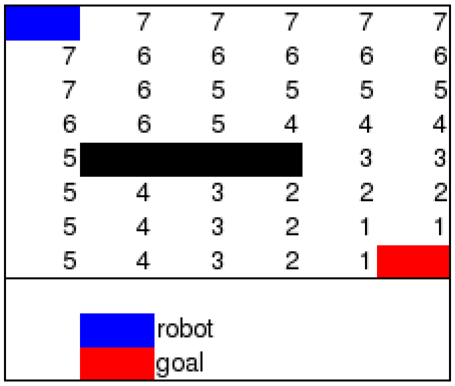

Figure 2: Wavefront grid with lower numbers surrounding the goal, and higher numbers surrounding the robot.

While Geeves is steering around the lab, it determines its location in the lab through AMCL [3]. A series of blobs, or landmarks, on the ceiling of the lab are an arrangement that is known to the robot beforehand. A camera mounted on the robot detects these blobs on the ceiling, and the robot determines its location in the lab through trigonometry and probability.

4. Experimental Design

We conducted experiments using the Stage simulator [6] to determine if the robot navigates safely and effectively to each destination on the tour. Safely means the robot should do no harm to itself or to other people as it moves throughout the room. Effectively means that the robot should get to within a short distance of the goal, preferably several centimeters. This navigation should also occur in a reasonable amount of time so that it does not appear clunky. All of our experiments were conducted with ideal localization rather than our own implementation. Furthermore, we were unable to do experiments under real conditions because our own implementation of localization was not functional at the time. If we conducted experiments under real conditions, we would have used the Pioneer 2DX mobile robot with a camera for the landmark detector and Esra Head for the speech.

5. Experimental Results and Discussion

Our experiments revealed that Geeves can navigate to each stop on the tour safely; Geeves did not collide into walls or cubicles that are indicated as dark shapes on the occupancy grid. Our results also showed that Geeves can steer around the lab effectively, for the most part. Geeves did not enter any forbidden areas on the map and did not get stuck in corners of the lab. The robot was able to arrive at each destination, but it sometimes took a longer path to move to each point. In other words, the robot did not always take the shortest path to each destination. Additionally, the robot traveled past its next location several times, and it would have to turn around to move to that location. Despite these issues, Geeves was able to effectively navigate to each destination most of the time.

6. Conclusion and Future Work

Although Geeves was able to navigate safely and effectively in our simulated experiment under ideal localization, our experiments do not guarantee that Geeves will perform the same way under real conditions. Additionally, the simulated environment did not contain other obstacles, such as chairs and trash bins, that could have deterred the robot’s performance. However, since our simulated environment was similar to the actual conditions in the lab, the robot will probably perform the same under real conditions.

Further work consists of testing the robot under real conditions after we fix the ACML driver. We also plan to test the entire tour-guide project, complete with interaction, after we are satisfied with the robot’s navigation around the lab.

7. Acknowledgement

This work is supported by the CRA-W Distributed Mentoring Program for the summer of 2006. Additionally, we thank David Feil-Seifer for his assistance with our project.

References

[1] I. Ulrich and J. Borenstein, “Vfh+.reliable obsstacle avoidance for fast mobile robots,” in *IEEE Int. Conf. on Robotics and Automation*, May 1998, pp. 1572-1577.

[2] “Vfh*:local obstacle avoidance with look-ahead verification,” in *IEEE Int. Conf. on Robotics and Automation*, April 2000, pp. 2505-2511.

[3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Proc. Of ICRA-99*, 1999, pp. 1322-1328.

[4] D. Fox, “Markov localization: A probabilistic framework for mobile robot localization and navigation,” Ph.D. dissertation, Institute of Computer Science III, University of Bonn, 1998.

[5] J. Borenstein and Y. Koren, “The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots,” in *IEEE Journal of Robotics and Automation*, 7(4): June 1991, pp. 278-288.

[6] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems,” in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, pages 317-323, Coimbra, Portugal, June 2003.

About Player

Player provides a network interface to a variety of robot and sensor hardware. Player provides an interace to the robot's sensors and actuators over an IP network. Our navigation program communicates with Player over a TCP socket, reading data from sensors, writing commands to actuators, and configuring devices while running.

We used *libplayerc++*, a C++ client library for Player, to develop our navigation program. The core of *libplayerc++* is based around the following classes:

1. *PlayerCc::PlayerClient*: Controls each connection to a Player server
2. *PlayerCc::ClientProxy*: Defines an interface for all proxy devices in Player
3. *PlayerCc::PlayerError*: Detects errors from *libplayerc++* and throws an exception

Our program used the path planner in *PlannerProxy* from *ClientProxy* to control the robot's motion through the lab. *PlannerProxy* provides an interace to a 2D motion planner. We used the following member functions in *PlannerProxy* for the robot's navigation:

- *PlannerProxy(PlayerClient *aPc, uint aIndex=0)*: Constructor
- *~PlannerProxy()*: Destructor
- *void SetGoalPose(double aGx, double aGy, double aGa)*: Set goal pose
- *void RequestWaypoints()*: Get the list of waypoints
- *void SetEnable(bool aEnable)*: Enable/disable the robot's motion
- *uint GetPathValid() const*: Returns 1 if the planner found a valid path, else returns 0
- *uint GetPathDone() const*: Returns 1 if the robot arrived at the goal, else returns 0

For more information about Player, visit http://playerstage.sourceforge.net/doc/Player-2.0.0/player/index.html.