Robots and Search Methods: Abstraction Through Application

Carl W. Steidley, R. Stephen Dannelly, Mario A. Garcia, and Sreevani Pelala,

Texas A&M University Corpus Christi

Abstract

Frequently in the introductory computer programming sequence we introduce topics in an abstract fashion in which the abstraction seems perfectly straightforward as well as easily implementable to students. Such is the topic of search. Generally, the topic of search is introduced to students as early as the data structures course where the student is introduced to various algorithms for the search of tree structures. In this light, students understand and are able to implement search as an abstract method of information retrieval. However, in this paper we describe a project in which students implement one kind of goal-based agent called a problem-solving agent. The development and implementation of this agent required the construction of a search process to find solutions to real-world physical problems. [4,8]

Introduction

A main element in analyzing a problem is the state space in which the problem exists, which describes all the possible configurations of the environment. In robotics, the environment includes the body of the robot itself. Both the configuration of the robot's body and the locations of objects in physical space are defined by real-valued coordinates. It is therefore impossible to apply standard search algorithms in any straightforward way because the numbers of states and actions are infinite. Much of the work in robot planning has dealt with ways to tame these continuous state spaces.

The question of moving a robot around successfully can be considered as problems of motion in a configuration space. Algorithms that handle a configuration space directly assume that an exact description of the space is available, so they cannot be used where there is significant sensor error and motion error. In some cases, no description of the space is available until the robot starts moving around in it. Russell and Norvig [5] identify five major classes of navigation and motion planning algorithms. We arrange them below roughly in the order of the amount of information required at planning and execution times:

Cell decomposition-these methods break continuous space into a finite number of cells, yielding a discrete search problem.

Skeletonization- these methods compute a one-dimensional "skeleton" of the configuration space, yielding an equivalent graph search problem.

Proceedings of the ASEE Gulf-Southwest Annual Conference
The University of Texas at Arlington

Bounded-error planning- these methods assume bounds on sensor and actuator uncertainty, and in some cases can compute plans that are guaranteed to succeed even in the face of severe actuator error.

Landmarked-based navigation – these methods assume that there are some regions in which the robot's location can be pinpointed using landmarks, whereas outside those regions it may have only orientation information.

On-line algorithms – these methods assume that the environment is completely unknown initially, although most assume some form of accurate position sensor.

A Cell Decomposition Problem-Discrete Search as a Function Maximization

Any search problem that is used to search blindly through a space will, generally, grow exponentially. In exploring a large search space, the trick is to use additional information about the problem being considered to avoid examining most of the nodes that might conceivably lead to solutions. Instead of selecting the node to expand next in a domain-independent way, we need to use domain-specific information for the same purpose.

Of course, the problem of finding the best node to expand next is generally no easier than the search problem that we are trying to solve. So, what is typically done is to apply some sort of heuristic, or "rule of thumb," [6] to decide what to do. For example, given a list of nodes to be expanded, we might simply guess how far each is from a goal node and expand the one thought to be closest.

Of course, things may not always work out so easily because our heuristic function estimating the distance to the goal may make mistakes in some cases. Another problem is that we need to limit the amount of time spent computing the heuristic values used in selecting a node for expansion.

There is no real "solution" to these problems of inaccurate or computationally expensive heuristics; the bottom line is that search problems sometimes will take an exponential amount of time to solve, and there is simply no way around this. But there is a third problem with this approach that is more tractable [3].

This problem can be best be understood by considering an example. Suppose that we are looking for a solution to a maze, where our estimate of the value of a node is simply the straight line distance from the current position and the exit from the maze. Thus, at any given point we do our best to move closer to the exit.

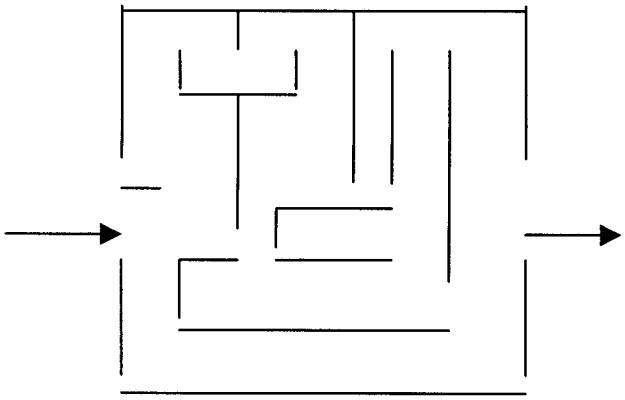

Figure 1. A maze with two solutions

Now consider the maze of figure 1, where a robot enters the maze on the left and exits on the right. As shown there are two solutions; 1. a simple one that begins with a step downward and 2. a much more complex one that begins with a step directly toward the exit but is then deflected in other directions. The algorithm that we have described will find the longer of these two paths, since it will begin by moving toward the goal and will then be committed to this choice.

Let us consider a heuristic search from another perspective, that is, from a formal point of view. Search problems can be simple function maximization problems: we have function f(p) that measures the interestingness of a point p on the surface and need to find that value of p for which the function's value is a maximum.[2]

The functional analog then is to attempt to find the global maximum of a function of many variables by always moving in the direction in which the rate of change is the greatest. This technique is known as hill climbing or steepest ascent for the obvious reason. The search version is presented below.

1. Set L to be a list of the initial nodes in the problem, sorted by their expected distance to the goal. Nodes expected to be close to the goal should precede those that are farther from it.
2. Let *n* be the first node on L. If L is empty, fail.
3. If *n* is a goal node, stop and return it and the path from the initial node to *n*.
4. Otherwise, remove *n* from L. Sort *n*'s children by their expected distance to the goal, label each child with its path from the initial node, and add the children to the front of L. Return to step 2.

In this version of the algorithm, we always take a step from a child of the previously expanded node when possible; this gives hill climbing a "depth first" flavor. If we drop this restriction we get an algorithm known as best-first search. That is:

1. Set L to be a list of the initial nodes in the problem.

2. Let *n* be the node on L that is expected to be closest to the goal. If L is empty, fail.
3. If *n* is a goal node, stop and return it and the path from the initial node to *n*..
4. Other wise, remove n from L and add to L all of n's children, labeling each with its path from the initial node. Return to step 2.

Odometry As A Dead-Reckoning Navigation Method

Odometry is the most widely used navigation method for mobile robot positioning. Odometry is based on simple equations that are easily implemented and that utilize data from inexpensive incremental wheel encoders. It is well known that odometry provides good short-term accuracy, is inexpensive, and allows very high sampling rates. However, the fundamental idea of odometry is the integration of incremental motion information over time, which leads inevitably to the accumulation of errors. Particularly, the accumulation of orientation errors will cause large position errors which increase proportionally with the distance traveled by the robot. Despite these limitations, most researchers agree that odometry is an important part of a robot navigation system and that navigation tasks will be simplified if odometric accuracy can be improved. Odometry is used in almost all mobile robots, for various reasons:

- Odometry data can be fused with absolute position measurements to provide better and more relieable position estimation.
- Odometry can be used in between absolute position updates with landmarks. Given a required positioning accuracy, increased accuracy in odometry allows for less frequent absolute position updates. As a result, fewer landmarks are needed for a given travel distance.
- Many mapping and landmark matching algorithms assume that the robot can maintain its position well enough to allow the robot to look for landmarks in a limited area and to match features in that limited area to achieve short processing time and to improve matching correctness.
- In some cases, odometry is the only navigation information available; for example: when no external reference is available, when circumstances preclude the placing or selection of landmarks in the enviroment, or when another sensor subsystem fails to provide usable data. [7]

Robotic Platform

Autonomous robot navigation provides an excellent method of providing students with interesting search problems. In the instance described in this paper students are provided with an ActivMedia Pioneer 2 Mobile robot platform.

An ActivMedia Pioneer 2 mobile robot is a versatile platform built on the client-server model. The robot acts as a server and uses "Saphira" resources. Saphira is a mobile application development environment. The robot server handles the low-level details of the sensors and drive management, such as, collecting range finding information from on-

board sensors, maintaining individual wheel speeds, positioning, heading and so on requires the client to guide it.

The Pioneer's on-board systems include: An 8 transducer sonar system, that is, 5 in the front providing 180 degree coverage, one on each side, and one in the back; A fast-track vision system; A radio modem; Electronic compass; and high precision drive wheel encoders and decoders.

Implementing Best-First As An Autonomous Robot Navigation Algorithm

The problem that was assigned was to have the Robot navigate over a course with starting point S and ending point G using a Best-First Search algorithm. The course may contain any number of intermediate obstacles (I_x). The Best-First algorithm, described above, is a goal-directed and knowledge-based algorithm, that is, both S and G are defined prior to navigation of the course. The objective being to move from S to G through a series of obstacles without colliding with the obstacles. The algorithm is implemented by having the robot traverse the Y component of its goal vector first. Reaching the Y component of its goal the robot then traverses the X component of it goal vector. [1]

In order to reset and initiate all systems the Robot is placed at S and is pointed to the east or along , what has been designated, the positive X axis. In order to simplify the computation time and effort it was decided that the Robot would, after initializing itself, rotate to the north or positive Y axis and then proceed to move toward G. Upon encountering an obstacle it will either pass by the obstacle or go around the obstacle depending upon the distance and orientation of the robot with respect to the obstacle. The robot will then perform a search for the next obstacle before it progresses in the positive Y direction.

The algorithm was implemented using the Robot's sonar system to determine the distance to the nearest obstacle. The robot keeps track of both its Y and X coordinates by tracking its precision wheel encoders and comparing with its goal X and Y components. The robot rotates in 10 degree increments allowing for obstacle orientation recording. Having surveyed its surroundings, the robot orients itself toward the nearest obstacle. The robot then, based on the heuristic that at each position of the robot it will take the shortest path by progressing towards the obstacle nearest to it, attempts to go around the obstacle.

When the robot discerns that it is within 200 mm of the Y coordinate of the goal (to provide a space buffer for the robot to pass by) it will compare its X coordinate with that of the goal. If the X coordinate is greater than the X coordinate of the goal G, the robot will move toward the west (negative X direction) until it reaches its goal. Utilizing this approach helps in finding the exact goal of the robot with the deviations discussed below.

Of course, the robot will reach its goal with the greatest precision, if the X path is unrestricted by obstacles. If, however, there happens to be obstacles in the path of the robot on its way to reaching the X coordinate of its goal, G, there could be considerable deviation in the path of the robot from the Y coordinate of G, as depicted in Figures 2, 3, and 4.

In some instances a deviation is inevitable, since, if the goal happens to be a point within or under the obstacle, it cannot be reached (Figure 2). In other situations, (Figure 3 & 4), a possible solution to refining this deviation would be integrating the algorithm with mapping wherein the robot is actually aware of its goal at a certain distance so that it could change its path and head toward the correct goal.

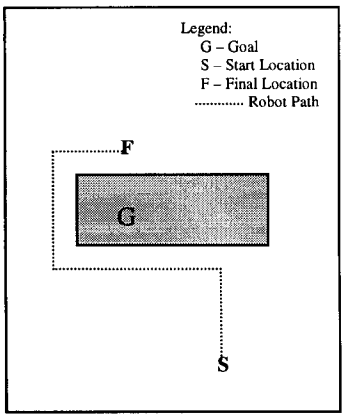

Figure 2. Sample Path #1

While heading towards the X coordinate of G the robot moves in straight line as long as there are no obstacles in its path. When the robot encounters an obstacle, it tries to move around the obstacle rather than stop, hence in moving around the obstacle it arrives at the correct X coordinate of the goal, but sacrifices its correct Y coordinate.

At this point the algorithm is then recursively repeated until the goal is reached or it is determined to be unobtainable for the reasons given above.

Implementation Difficulties

Although the algorithm described above appears straightforward, robust, and easy to develop, when implemented in the real world a number of problems where encountered. The first problem to overcome was to simply not run into anything. Trial and error using

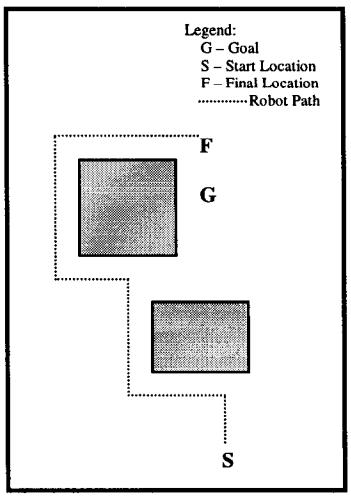

Figure 3. Sample Path #2

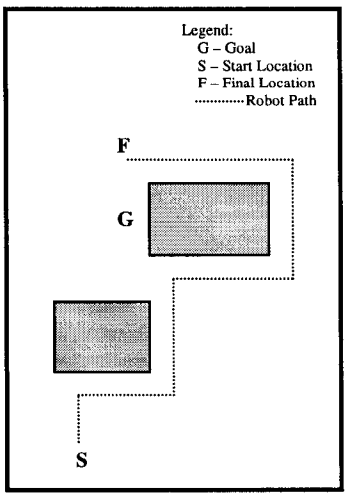

Figure 4. Sample Path #3

several different hallways as well as a maze-like obstacle course in a large lab space indicating that it was necessary to keep the robot at least 200mm away from walls and other obstacles. This buffer zone was needed for several reasons. First, the mobile robot in question is equipped with a gripper on its front, which projects several centimeters in front of the sonars.

Second, the real world is full of small pebbles, bumps in carpet padding, and so forth that cause a moving robot to wobble. Third, sometimes an object encountered is actually not a chair, but is instead a human leg
that moves quickly when startled by a robot.

Moving around an obstacle appears to be a simple process - turn left, move along object until it disappears from the side sonar, move a distance beyond the object to allow space to turn, turn right, etc. However, in practice obstacle avoidance is quite difficult. The first problem to solve is how to avoid obstacles while avoiding an obstacle. A recursive algorithm was the obvious solution. However, while avoiding object A, if object B is found and is avoided leading to object C whose avoidance leads back to object A, the algorithm will infinitely recurse. One innovative solution was to randomly traverse clockwise or counterclockwise when an object was encountered. A second obstacle avoidance problem is that is very possible to encounter the point G while traversing an obstacle. Also, traversing clockwise may more likely lead to the goal, while counterclockwise traversal may lead the robot even farther away from the goal. The complexity of these interrelated problems, to say the least, caused students to realize the importance of problem analysis and algorithm testing over their usual source code development technique of immediately sitting down and writing code.

Systematic and Non-Systematic Odometry Errors

Odometry is based on the assumption that wheel revolutions can be translated into linear displacement relative to the floor. Students must learn that this assumption is only of limited validity. One extreme example is wheel slippage: if one wheel was to slip on, say, an oil spill, then the associated encoder would register wheel revolutions even though these revolutions would not correspond to linear displacement of the wheel.

Along with the extreme case of total slippage, there are several other more subtle reasons for inaccuracies in the translation of wheel encoder readings into linear motion. All of these errors fit into one of two categories: systematic errors and non-systematic errors.

Systematic Errors

- Unequal wheel diameters.
- Average of actual wheel diameters differs from nominal wheel diameter.
- Actual wheelbase differs from nominal wheelbase.
- Misalignment of wheels.

- Finite encoder resolution.
- Finite encoder sampling.

Non-Systematic Errors

- Travel over uneven floors.
- Travel over unexpected objects on the floor.
- Wheel slippage due to:
  - ❖ slippery floors.
  - ❖ overacceleration.
  - ❖ fast turning (skidding).
  - ❖ external forces (interaction with external bodies).
  - ❖ internal forces (castor wheels).
  - ❖ non-point contact with the floor.

Students must also learn that the clear distinction between systematic and non-systematic errors is of great importance for the effective reduction of odometry errors. For example, systematic errors are particularly grave because they accumulate constantly. On most smooth indoor surfaces systematic errors contribute much more to odometry errors than non-systematic errors.

Conclusions

While we have used a very sophisticated robot system provided by a National Science Foundation grant, the advent and use of inexpensive, hobbyist robot systems, such as the Lego MindStorms robot systems, makes the application of previously abstract computing algorithms come to life.

References

1. Aoki, T., Matsuno, M., Suzuki, T, & Okuma, S. (1994), "Motion Planning for multiple obstacles avoidance of autonomous mobile robot sing hierarchical fuzzy rules", Proceedings of the IEEE International Conference on Multi-sensor Fusion and Integration for Intelligent Systems, 1994.
2. Cho, D.K., & Chung, M..J. (1991), "Intelligent Motion Control Strategy for a Mobile Robot in the Presence of Moving Obstacles", Proceedings IROS '91, 1991.
3. Li, W. Fuzzy "Logic-Based 'Perception-action' Behavior Control of a Mobile Robot in Uncertain Environments", Proceedings IROS '94, 1994.
4. Latombe, J.C., "Robot Motion Planning", Kluwer Academic, New York, 1993.
5. Russell, S and Norvig, P, "Artificial Intelligence: A Modern Approach", Prentice Hall, Upper Saddle River, NJ, 1995.
6. Feigenbaum, E.A. and Lenat, D.B., "On the Thresholds of Knowledge", Artificial Intelligence, 47:139-159, 1991.
7. Borenstein, J., Everett, H.R., Feng, L., Lee, S.W., and Byrne, R.H., "Where Am I? Sensors and Methods for Mobile Robot Positioning", Department of Energy Report for the Oak Ridge National Laboratories, April 1996.
8. Dannelly, R. S., Steidley, C.W., "A Student Laboratory Environment for Real-Time Software Systems Development", The Journal of Computing in Small Colleges, Vol. 16, No. 3, pp. 132-137, March 2001.

CARL STEIDLEY is Professor of Computer Science and Chair of Computing and Mathematical Sciences. His interests are in the applications of artificial intelligence, real-time computing, and robotics. His most recent extra-university research and development appointments have been with NASA Ames Research Center, Oak Ridge Natl. Labs, and Electro Scientific Industries in Portland, OR

R. STEPHEN DANNELLY is Associate Professor of Computer Science at Texas A&M University – Corpus Christi. Dr. Dannelly's interests are in software engineering, environmental modeling, and involving undergraduates in research. Dannelly's most recent funding has come from NASA, NSF, and the U.S. Army.

MARIO GARCIA is an Assistant Professor of Computer Science at Texas A&M University-Corpus Christi.. His interests are artificial intelligence, expert systems, neural networks, robotics and software engineering. He has implemented industrial applications of expert systems in several cities in Mexico, and in Houston, Tex. He is also interested in the use of technology to improve teaching.

SREEVANI PELALA is an Adjunct Professor of Computer Science at Texas A & M University, Corpus Christi. Her interests are in the applications of artificial intelligence and robotics and web based dynamic database applications in business information systems.