Low Error Path Planning for a Synchro-Drive Mobile Robot

David P. Miller
Prabhakar M. Koushik

TR 86-28

Low Error Path Planning for a Synchro-Drive Mobile Robot

David P. Miller
Prabhakar M. Koushik

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
(703) 961-5605

Abstract

A path-planning system is presented which, given way-points, calculates a low-error path for a synchro-drive robot. The path reduces dead-reckoning errors by using gentle, constant curvature turns for switching directions of the robot's travel — yet direct and minimal length travel are maintained. The system has been implemented and tested with a *Vectrobot* mobile platform.

1 Introduction

In this paper we divide the process of calculating the movements, a robot must execute to get from its present position to its destination, into two steps: route planning and path planning. We will call route planning the process of computing a route to the destination that avoids all major obstacles, and that goes near sufficient landmarks to provide the robot for its navigational needs. We define path planning as the detailed instructions to the robot of where to go straight and where to turn. Path planning calculates the dead-reckoning sections of the robot's route.

The remainder of this section provides further motivation for path planning. The path planning needs for a synchro-drive robot are explored and some past work in path-planning is reviewed. The next section details our path-planning algorithm. An implementation of this algorithm on a synchro-drive robot is then discussed.

1.1 Dead-Reckoning

Getting a mobile robot from one location to another is not a simple task. In non-trivial cases, extensive computation is often required to determine if a feasible route is possible between the starting and destination points. The path must then be plotted. This most often involves some sort of search through a regionalized map of the robot's surroundings [1], [5]. Once the path has been plotted it is still necessary for the robot's instructions for traversing that path to be constructed.

How the above tasks are accomplished, and what their accomplishment actually looks like, depends on many factors; one of prime importance is the robot's ability to do *dead-reckoning*. Dead-reckoning is the ability of a robot to follow a given path using only internal robot sensors such as shaft encoders, slip detectors, and inertial guidance information. The extent of a robot's dead reckoning abilities is also dependent on the environment through which it is traveling. A smooth concrete floor would provide more extensive dead-reckoning capabilities for a rubber-wheeled mobile robot then would a sandy beach or a shag carpet.

There are three classes of dead-reckoning abilities:

- Perfect dead-reckoning, no external sensing is needed
- Useful dead-reckoning, the abilities are sufficient to get the robot somewhere near its destination
- Useless dead-reckoning, no positional information is available without reference to information from external sensors.

The vast majority of mobile robots fall into the second category. Useful dead-reckoning is needed when the robot must traverse an area that is out of range of its sensors [8]. Dead-reckoning abilities are also useful when the robot must make a series of small varied maneuvers in a known and stable environment. For example, consider a robot maneuvering to the other side of a desk (see Figure 1). In these cases, the points in space that the robot should travel through are well known. The greater the accuracy that the robot can traverse these paths, in a predictable manner, the faster the robot will be able to accomplish its tasks (because it need not gather nor process extensive sensory data). This paper describes a method for a syncho-drive robot to traverse a set of points in space that increases the robot's dead-reckoning abilities over the accuracy attainable by going to each point directly.

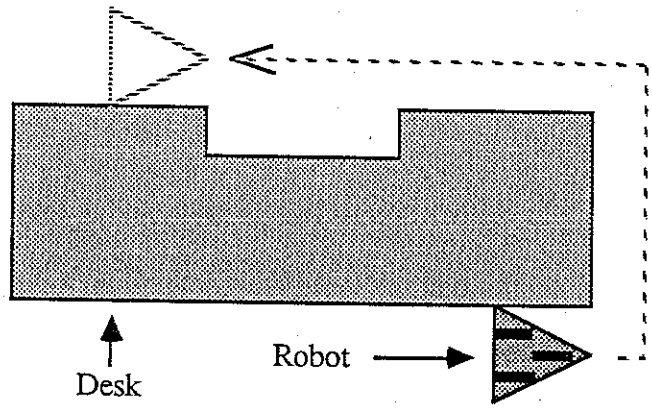

Figure 1: Maneuvering Around a Desk

1.2 The Synchro-Drive System

The synchro-drive system is a relatively new drive scheme. The system consists of three wheels, two motors, and a set of gears and/or belts. All three wheels of a synchro-drive robot are driven. This is done by a single motor mechanically connected to the drive shafts of all the wheels. Since a single motor controls all the wheels, the wheels cannot get out of synch, and there should be little or no drift.

Like the drive, all three wheels in a synchro-drive system steer. Once again, this is performed by a single motor connected to all the wheels. The wheels maintain a constant orientation to one another, but can turn indefinitely with respect to the body of the robot. For the robot to drive in a circle the wheels are turned at a constant rate 360° while they are being driven (see Figure 2). The size of the circle is controlled by the ratio of the speeds between the steering and drive motors.

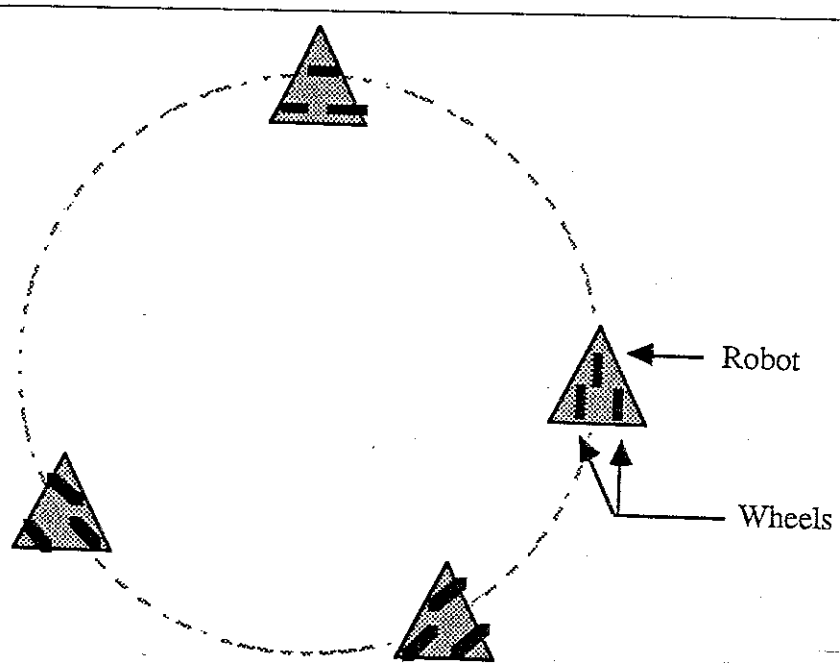

Figure 2: A Synchro-Drive Robot Moving in a Circle

The synchro-drive system allows great manuverability and accuracy of movement. Any position that the robot gets into, it can easily get out of. The synchro-drive also has the unusual property that while the robot "turns", the body of the

robot maintains a constant orientation. All of the commercially available synchro-drive robots have a platform which turns with the wheels. This allows sensors to be mounted in a way such that they are always facing the direction of travel.

1.3 Synchro-Drive Errors

The internal sensors on a typical synchro-drive robot consist of shaft encoders on the drive and steering motors. These sensors allow the robot controller to accurately monitor the amount that the robot's wheels turn and rotate. These sensors allow the robot to catch some types of errors in its execution of movement commands.

The most common type of error in this class of robot is overshoot. When the robot is given a command to move a specified distance, power is applied to the appropriate motor until the sensor indicates that the motor has moved the correct amount; power to the motor is then cutoff. If the speed of the robot's movement is sufficient, the robot's momentum will often overcome the restsistance of the drive system, the robot will coast onward some amount. Some of this overshoot will be registered on the robot's sensors; this amount can be corrected for later. Unfortunately, some of the coasting will involve the robot's wheels slipping over the surface. Depending on the robot, its brake system, and the surface being traversed, the precise amount of this slippage can vary widely.

The other common error in the synchro-drive system comes about as the robot makes a directional change. The wheels on a robot are not infinitely thin. When the wheel is turned to a new heading some slippage of the wheels against the surface of the floor must occur. If this slippage is not perfectly even, some change in the robot's position and orientation may occur. To minimize the robot's precession, each wheel may be rotated slightly off axis. This causes the robot to wiggle rather than precess.

A system for totally eliminating this problem, using dual wheels and differentials has been designed [9]. Unfortunately, such a system is very sensitive to mis-alignment in the wheels and uneveness in the surface being traversed. While very elegant, the dual-wheel solution has not proven practical [10].

2 Planning Low-Error Paths

There are procedural solutions to both the overshoot and precessional errors in a synchro-drive robot. Overshoot is minimized if the robot travels at very low speeds. It can also be reduced by eliminating starts and stops. Precession due to steering changes is minimized the faster the robot is moving when the steering is changed. Caution must be excercised that the robot is not moving too fast when making a sharp turn — or else skidding, and associated errors, may result. The remainder of this section presents a path-planning algorithm which uses these techniques to calculate low-error paths.

2.1 The Path-Planning Algorithm

The algorithm described below is designed to move a synchro-drive robot smoothly through a set of points specified in X, Y coordinates. Figure 3 shows the case of motion between two ponts $A(x_1, y_1)$ and $B(x_2, y_2)$.

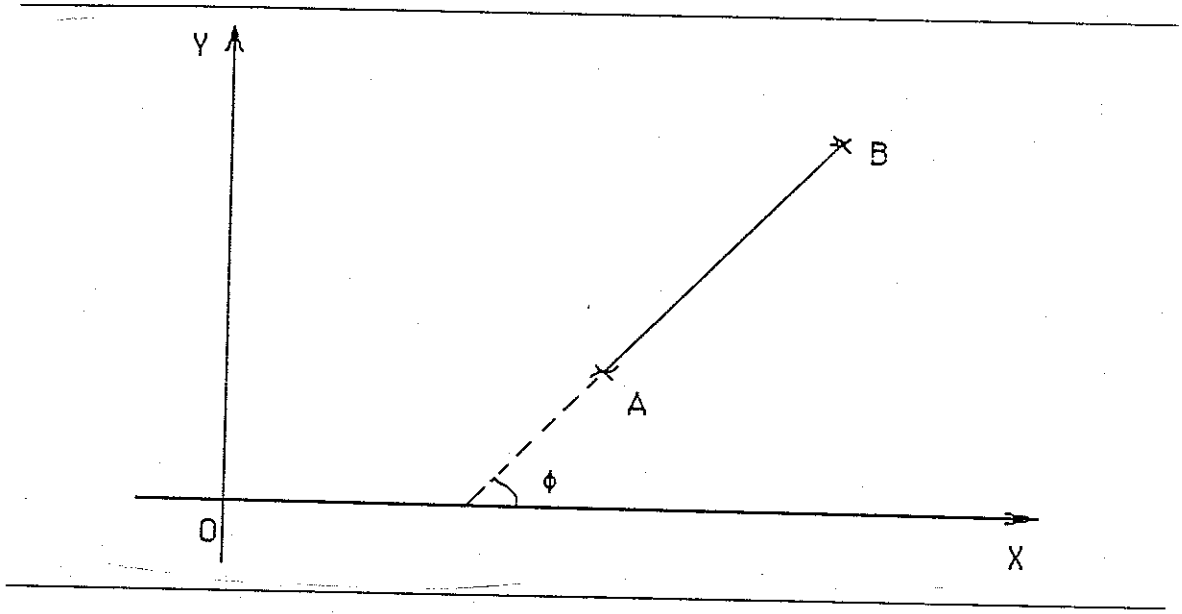

Figure 3: Motion Between Two Points

If the robot is already at point A with its wheels oriented at an angle ϕ with respect to the X-axis then the task of proceeding from point A to point B reduces

to linear motion between the two points. This is achieved by sending the robot a command to move forward an amount $\sqrt{|x_2 - x_1|^2 + |y_2 - y_1|^2}$. The forward motion is achieved by making the robot accelerate to its full speed, and then decelerate at it approaches its destination point. This allows the robot to traverse the distance in a reasonable amount of time while still minimizing the amount of overshoot.

If the robot must travel through two points beyond its current position, the coordinate system is initialized so that the robot's starting position is at the origin with the wheel aligned along the X-axis. (see Figure 4). The driver computes the bisector on $\angle OAB$ and selects a point on the bisector distance $|r|$ from point A. A circle of radius r, which passes through point A is then calculated. The tangent points (which pass most closely to point A) from O to the circle (point L) and from B (point M) are then calculated. The path for the robot then consists of the following segments and arcs: $\overline{OL} \circ LM\overline{MB}$. It should be noted that as the robot traverses the circular arc $\circ LM$ it travels through the point A.

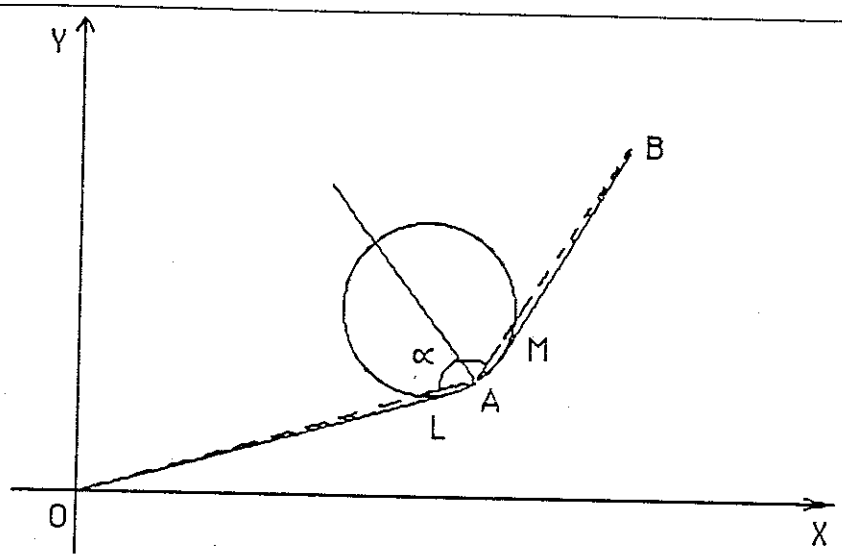

Figure 4: Calculating a Path Along Three Way-Points

If the robot's path involves four or more points (see Figure 5) then the path is calculated as in the three point case described above with one exception. The path segments that do not connect to the starting or destination points are tangent to

the neighboring circles, rather than the route-selected way-points. Thus the robot travels parallel, but slightly to one side, of the line connecting the actual way-points. The displacement of the path from the line connecting the way-points is dependent on the radius of the circles at each way-point.

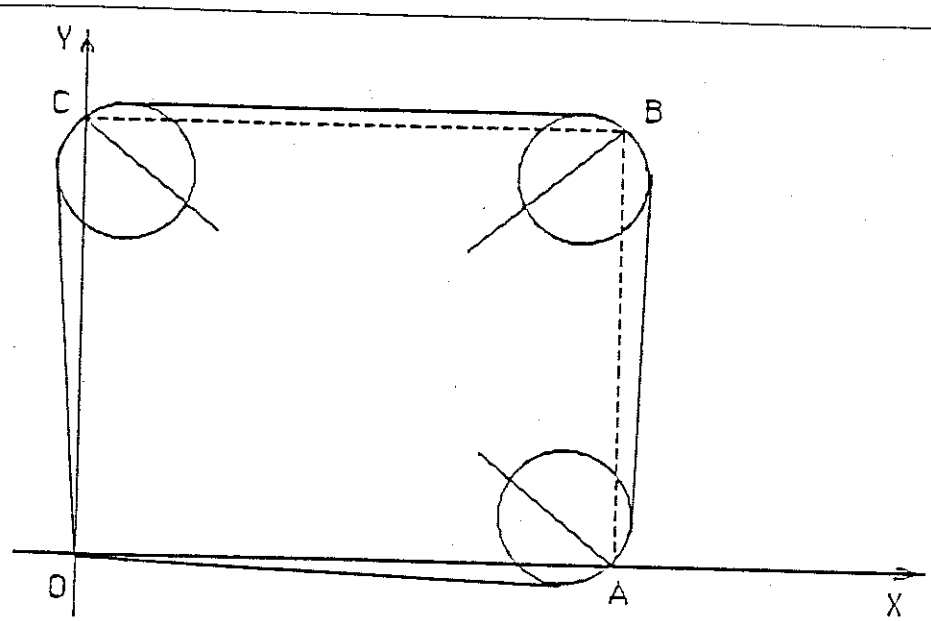

Figure 5: A Path Plan for a Square

3 Previous Work in Path Planning

Much of the work in path planning has dealt with variations of the piano-movers problem [13], [12]. This robot problem (usually referred to as the findpath problem) has concentrated on obstacle growing and travel along the regions of Voronoi diagrams [11], [2], [8]. This body of work has made one of two assumptions: either the robot is circular in shape and can instantly switch direction, or the robot is irregular in shape, but can be rotated at any point on its body. The configuration-space approach to spatial path planning [6] takes the robot's maneuverability into account, but is a more expensive approach to path planning. These systems all merge the process of path-planning with that of route-planning.

The *Smooth-Driver* path-planning system [4], [3] has been used with two different robot drive systems. The drive systems used are two-wheeled independent drive (similar to a wheel-chair) and the Stanford omni-directional robot. The Smooth-Driver system uses *cornu* spirals which have the property of linearly increasing or decreasing curvature. Such a system is very suitable with most drive systems where a robot's rate of turn per distance traveled is independent of the robot's speed. This is not the case for a synchro-drive robot.

4 Uses of a Synchro-Drive Path Planner

Because the body of a synchro-drive robot does not change orientation, no matter what direction the robot travels, the traditional methods of solving the find-path problem are either ineffective or overkill. The space through which the robot can travel can be plotted by obstacle growing. This is accurate since the robot can move in any direction from any position. Once the obstacles have been grown, way-points may be picked through any traditional route-planning scheme.

The path planner described in this paper can then be used to calculate a smooth low error path for the robot to take through the way-points. The path calculated will always have the robot travel the most direct path, or one that is very slightly longer, but includes some extra distance when traveling along convex obstacles (no efficient route planner would have the robot travel along the edge of a concave obstacle).

The path-planner that has been described uses a simple, predictable, and efficient algorithm for computing a low-error path. This system has been implemented in NISP [7], a dialect of LISP, on a Vax-785 computer. The system buffers commands and sends them through a tether to a *Real World Interface* brand *Vectrobot* synchro-drive mobile robot platform, which carries out the instructions. The linear segments of the path include acceleration and deceleration instructions. The circular arcs are run at a moderate drive and turning speed to minimize skidding, precession errors, and keep the turning radius reasonable. Even on difficult surfaces such as carpeting (low pile), the robot is able to maintain positional accuracies of

$\pm 2\%$ in a closed course with 360^o of rotation divided into several smaller turns. This accuracy should prove sufficient for having the robot maneuver about near obstacles via dead-reckoning while it is gathering and processing the sensory data of more distant objects.

References

[1] Rodney A. Brooks.
Find-path for puma class robot.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 40–44, AAAI, Washington D.C., August 1983.

[2] Bruce R. Donald.
Hypothesizing Channels Through Free-Space In Solving the Find-Path Problem.
Memo 736, MIT Artificial Intelligence Laboratory, 1983.

[3] Yutaka Kanayama, David Kriegman, and Soon Yau Kong.
A vehicle controlled architecture - smooth driver.
1986.
Submitted to AAAI 1986.

[4] Yutaka Kanayama and Northisa Miyake.
Trajectory generation for mobile robots.
In *Robotics Research*, MIT Press, 1985.

[5] T. Lozano-Perez and M. Wesley.
An algorithm for planning collision-free paths among polyhedral obstacles.
Communications of the ACM 22, :560–570, 1979.

[6] Tomas Lozano-Perez.
Spatial planning: a configuration space approach.
In *IEEE Transactions on Computing (C-32)*, pages 681–698, 1983.

[7] Drew McDermott.
The NISP Manual.

Technical Report 274, Yale University Computer Science Department, June 1983.

[8] David Miller.
A spatial representation system for mobile robots.
In *Proceedings of the 1985 International Conference on Robotics and Automation*, pages 122–127, IEEE, St Louis, Mo., March 1985.

[9] Hans P. Moravec.
The cmu rover.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 377–380, AAAI, Pittsburg, PN, August 1982.

[10] Hans P. Moravec.
Three degrees for a mobile robot.
In *Proceedings of the 1984 International Computers in Engineering Conference*, pages 274–278, ASME, LasVegas, Nevada, August 1984.

[11] Van-Duc Nguen.
The Find-Path Problem in the Plane.
Memo 760, MIT Artificial Intelligence Laboratory, 1984.

[12] C. O'Dunlaing, M. Sharir, and C. Yap.
Retraction: a new approach to motion planning.
In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 207–220, ACM, Boston, Massachusetts, April 1983.

[13] J. Reif.
Complexity of the mover's problem and generalizations.
In *Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science*, pages 241–247, New York, NY, 1979.