Implementation of Dynamic Obstacle Avoidance on the CMU NavLab

Dai Feng
Dept. of Electrical Engineering
Grove City College
Grove City, PA 16127

Sanjiv Singh
Field Robotics Center
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Bruce H. Krogh
Dept. of Electrical
and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

1 Introduction

This paper presents the design and implementation of local obstacle avoidance algorithms on the CMU NavLab. The CMU NavLab is a modified Chevy van designed as a test bed for autonomous navigation [1], [2]. It carries various computing systems including three Suns, one Warp systolic array processor, and several Intel 386 real-time processors for the purpose of processing sensor information and generating vehicle motion commands. The position and velocity feedback information are obtained from an inertial guidance system mounted on the vehicle. Obstacle maps are generated from laser range scanner. The motion of the NavLab is controlled by sending steering and drive commands to the servo controllers which regulate the steering wheel angle and the vehicle speed.

The obstacle avoidance algorithm we present in this paper is part of a reflexive path tracking scheme. In this scheme, the NavLab is normally under the control of a path tracking program which drives the NavLab along a predefined path use dead-reckoning techniques. Path tracking cannot cope with the situation where obstacles are present on the predefined path. To handle such situations, an obstacle avoidance algorithm is added to the reflexive path tracking scheme whereby when an obstacle is detected on the predefined path, the vehicle control is transferred to the obstacle avoidance algorithm which guides the vehicle around the obstacle. This combination results in a scheme that reacts to the changing and uncertain environment.

In our obstacle avoidance problem formulation, we assume the NavLab servo-controller is capable of tracking a reference trajectory with specified accuracy, provided the trajectory satisfies geometric, kinematic and dynamic constraints determined by the NavLab's physical characteristics. This permits the obstacle avoidance problem be formulated in terms of a simplified model of the underlying servo-controlled system. We apply a previously proposed navigation scheme in which approximations for the complex nonlinear system dynamics and operating constraints (including obstacles) admit a computationally feasible solution, generating reasonable trajectories based on local feedback information [3]. As in our original formulation, the obstacle avoidance problem is decomposed into two subproblems, namely, the *subgoal selection* and the *dynamic steering* problems. The subgoal selection algorithm (SSA) is designed to provide a temporary sequence of subgoals that lead the NavLab around the obstacles and back to the predefined path the NavLab was tracking. Once a visible subgoal is generated, the steering control algorithm (SCA) dynamically steers the NavLab to the subgoal. Both the SSA and the SCA have been presented in detail in earlier papers [4], [5].

2 Processing the Scanner Data

The obstacle avoidance system uses Cyclone, a laser range scanner capable of obtaining a single axis of range data at up to 8 scans/second. The scanner is mounted on the front of the Navlab to collect obstacle information while the Navlab is in motion. The scanner has a range of 30 meters and a field of view of 120 degrees. The scanner data are processed to generate a polygonal representation of the obstacles within the scanner range. During each sampling interval (typically 0.5 seconds), range data from the laser scanner is processed to produce a polygonal representation of the local obstacles. Range measurements are segmented into polygonal shapes in two steps.

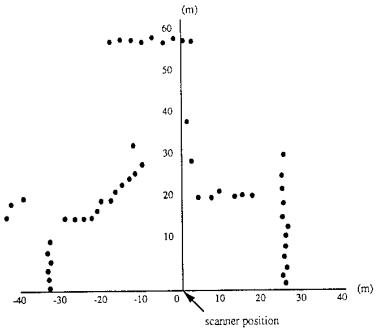

Figure 1: Raw data obtained from the Range Scanner

Figure 1 shows range data from one scan of the scanner in a hypothetical world. In the first step, range data is converted from polar to Cartesian coordinates and the (x,y) points are clustered into contiguous blocks. That is, if two consecutive range measurements are separated by more than a preset threshold (s_1), a new cluster is started. Figure 2 shows the data from Figure 1 in its segmented form. It should be noted that s_1 is chosen based on accuracy and desired level of detail required in segmentation. In any case it is not useful to set this threshold any lower than the accuracy or resolution of the range sensor. The Cyclone scanner has a resolution of 30 cm and we found that a value of 1m worked well in the terrains that the NavLab was navigating.

In the second pass, a common "polyline" algorithm (PLA) is used to find a set of straight lines to fit each cluster [6]. This scheme uses a recursive method that ensures that the fitted lines are never more than a preset threshold (s_2) away in lateral distance from the range data points. To start with, a single line is fitted between the first and the last line in the segment. For every point in each segment, the perpendicular distance to the fitted line is calculated. If this distance is larger than s_2, the procedure recursively picks the point farthest from the straight line (of distance (m)), makes it a new breakpoint and replaces the current fitted line with two new line segments, until none of the points are more than the threshold s_2 away from the fitted line. This process is illustrated in Figure 3 (a),(b),(c) by fitting a polyline to segment 4 from Figure 2.

Relaxing the threshold s_2 has the effect of a coarser fit with fewer

CH2872-0/90/0000-0208 © $1.00 1990 IEEE

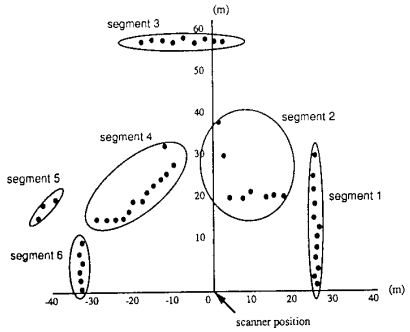

Figure 2: Segmented range data

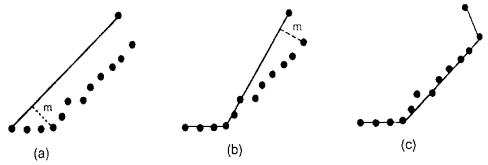

Figure 3: A polyline fit to segment 4

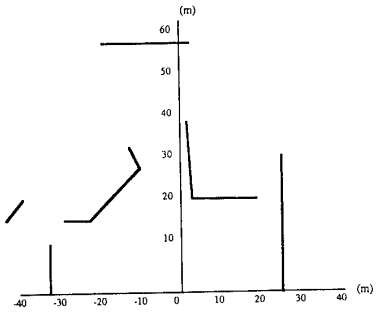

Figure 4: The polygonal shapes extracted from data of Figure 1

lines to approximate the cluster. Tightening it has the effect of a finer fit, albeit with more line segments.

Finally, Figure 4 shows the extracted lines obtained by application of the PLA to the data from Figure 1.

3 The SSA

The obstacle avoidance algorithm on the NavLab is invoked when the NavLab is in the process of tracking the predefined path and a cluster of obstacles is detected by the Cyclone scanner. The obstacle avoidance algorithm must steer the NavLab around the obstacles and guide it back onto the predefined path at some point beyond the obstacles. We call this point the *final goal* for the obstacle avoidance algorithm. Most often the final goal is not visible from the initial position of the NavLab. Subgoals are then required to guide NavLab around the obstacles until the final goal is visible. The subgoals are generated by the SSA. Each time the SSA is invoked, it first determines if the final goal is blocked by an obstacle. In determining whether an obstacle blocks a goal, we expand the obstacle by an amount equal to $\rho+\delta$ where ρ is the radius of the circle enscribing the NavLab and δ is the tolerance parameter. If the line of sight to the goal from the NavLab position intersects the expanded obstacle, we say the goal is blocked.

If the final goal is blocked by an obstacle, we mark this obstacle as an initial obstacle and the first subgoal must be generated to lead the NavLab around this initial obstacle. A subgoal can be place on the right side or the left side of the obstacle. In the SSA we choose to go around the side of the obstacle that is closest to the final goal. The subgoal is generated next to the obstacle edge at a distance of $\rho+\pi$ ($\pi>\delta$). The parameter π can be chosen to modify the characteristics of the NavLab trajectory, with larger π generally resulting in smoother trajectories. The subgoal is placed at the right (when subgoal direction is right) of the obstacle's right edge so that the lines from the obstacle edge to the subgoal and from the NavLab position to the subgoal are perpendicular to each other.

Once a subgoal for the initial obstacle is generated, it is checked to ensure that it is not blocked by, or too close to, another obstacle. New subgoals are recursively generated until the last subgoal is not blocked by nor too close to an obstacle. For the last subgoal, a *free space* is generated. A free space is a triangular region which does not contain any obstacles. Two of the tree vertices of the triangle are the subgoal and the NavLab position. The third vertex is chosen to exclude the obstacles from the free space while allowing the free space to contain a large area for the NavLab to maneuver. We refer the readers to reference [5] for detailed rules for generating the free space.

Each subgoal is tested for *feasibility*. When a subgoal is not feasible, it means either the subgoal is outside of the region covered by the scanner, or the location of the subgoal requires the NavLab to change its trajectory in a way that is kinematically or dynamically impossible (e.g., to make a turn with a turning radius smaller than the minimum turning radius of the NavLab). The first case may happen if either the initial obstacle is already very close to the edge of the scanner's range or a cluster of closely bundled obstacles next to the initial obstacle extends all the way to the edge of the scanner's range. When such condition occurs, we simply change the subgoal direction and try to generate subgoal to lead the NavLab around the other way. When that also fails to give a feasible subgoal, the SSA returns the information that there is no feasible subgoal and the *default maneuver* (defined in section 4) is executed to bring the NavLab to a stop. A higher level program must then select a new final goal. On the other hand, if a feasible subgoal is selected, it is passed along with its free space to the SCA which will determine the proper steering/drive commands.

4 The Steering Control Algorithm

Given the subgoal and free space, the SCA uses a model of the NavLab to generate the steering/drive commands. The NavLab is a conventionally steered vehicle(CSV) for which steering is achieved by means of pivoting the front wheels and propulsion is achieved by applying torque to the rear wheels. For simplicity, the NavLab can be modeled by the "bicycle" model with each pair of wheels collapsed into a single wheel [7]. The wheel base, denoted by l, is the distance between the axes of the front and rear wheels of the NavLab. The local coordinate frame for the NavLab is assigned at a distance of l_r from the rear wheel on the center line of the NavLab body. We refer to the velocity of the local coordinate frame on the NavLab as the NavLab velocity and the speed of NavLab at the rear wheel as the rear wheel speed, denoted by $\mathbf{v}$ and v_r respectively. The orientation of the NavLab, which is defined by the orientation, θ, of the local coordinate frame in the global frame, is critical in determining the future state of the NavLab.

For the purpose of generating steering/drive commands, we consider the NavLab as a point concentrated at the origin of the robot coordinate system. The finite size of the NavLab can be

compensated for by means of expanding the obstacle regions in the scanner map. For the ease of generating a reference trajectory to be tracked by the NavLab servo, we choose the steering angle of the NavLab front wheel $\delta(t)$ and the speed of the rear wheel $v_r(t)$ as two of the reference states and their derivatives as the steering vector.

The task of the SCA is to generate a piecewise constant steering vector at each sample interval. The steering vector is applied to the reference model to generate the steering/drive commands which results in a NavLab trajectory that avoids the obstacles. For our NavLab model, we have selected the angle of the steering wheel δ and the speed of the rear wheel v_r as the state variables. For the ease of considering the obstacles when generating a reference trajectory, we choose two other state variables to be the x and y positions of the NavLab in the global coordinate frame, p_x, p_y. To specify a unique state of the NavLab, the orientation of the NavLab, θ, is required as the fifth state variable. We note that for purpose of driving the NavLab only δ and v_r are required as the output of the NavLab model since these are the steering/drive commands that must be issued to the NavLab's servo controllers.

The overall structure of the SCA is illustrated in figure 5. As shown in the figure, the steering vector $\mathbf{u}(t_n)$ is selected from a constraint set which is the intersection of the *system constraint* set $U_s(t_n)$ and the *environment constraint* set $U_e(t_n)$. The system constraint set $U_s(t_n)$ represents the kinematic and dynamic operating limits of the NavLab. When the steering vector is within this set it is guaranteed the resulting reference trajectory can be followed by the servoed NavLab within a known tolerance. The environmental constraint set $U_e(t_n)$ results from mapping the obstacle constraints into constraints on the steering vector using a representation of the free space, $C(t_n)$. When the steering vector $\mathbf{u}(t_n)$ is within this set, it is guaranteed that the state of the NavLab will be safe at the next decision time, t_{n+1}. Figure 5 indicates that if the constraint set $U_e \cap U_s$ is empty, the SDA generates a *default maneuver* which is a predetermined reference trajectory generated to bring the NavLab to a stop before running into obstacles. When $U_e \cap U_s$ is nonempty, the steering vector $\mathbf{u}(t_n)$ is chosen from $U_e \cap U_s$ to approximate a desired steering vector $\mathbf{d}^*(t_n)$ called the *objective vector*. There is great freedom in choosing the objective vector and performing the approximation. For the NavLab, we used a heuristic method which has been reported in [3].

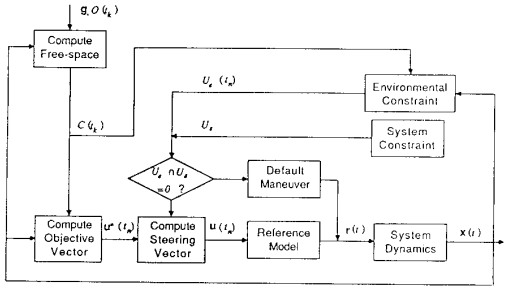

Figure 5: The feedback structure of the steering control algorithm.

5 Implementation of the SSA and SCA

At the time of the implementation, the real time system hardware on the NavLab was not completely ready. Therefore the SSA and SCA was implemented on a pseudo real-time system with a Sun 3 providing the main computation power. The scanner, SSA and SCA are synchronized through the clock on the initial guidance system. The cyclone scanner is regulated by its own servo- controller to spin at 2 revolutions per second, which generates a local obstacle map every 0.5 seconds. The inertial guidance system sends a data structure containing the NavLab states including position, velocity and yaw rate to our obstacle avoidance program at 0.1 second intervals. Each NavLab state structure has a time stamp. The obstacle avoidance program checks each incoming state structure. At every 0.5 seconds, the SSA is invoked to generate new subgoals using the latest scanner map. At every 0.1 seconds, the SCA is invoked to generate new steering/drive commands.

The structure of the obstacle avoidance program is shown in figure 6. The inputs to the obstacle avoidance program are the reference signal (final goal) from a higher level program, NavLab state feedback from the inertial guidance system, and the local map from the cyclone scanner. Each time a stamped state feedback is available, it is checked to see if 0.5 seconds have elapsed since last time the SSA was executed. If so, a new local coordinate is established with the current NavLab position as the origin and the current NavLab heading as the positive x-axis direction. The location of the final goal is then transformed from the global coordinates to this new local coordinate system. The SSA is then invoked to generated new subgoals, using the local obstacle information generated by the PLA. Notice that local maps are naturally generated in the local coordinates. If no feasible subgoals can be generated by the SSA, emergency breaking, that is, the default maneuver is applied. The higher level program which generates the final goal is also notified to modify the final goal. When a feasible subgoal is generated, the SCA is executed to generated the steering vectors . The steering vector is integrated using the NavLab model to obtain the steering/drive commands which are sent to the NavLab servo controller. The SCA is executed each time a state feedback is available, ie, at 0.1 second intervals.

6 Experimental Results

The obstacle avoidance algorithm was implemented in the C programming language in a Sun work station. This allows for easy development and porting of the program to the Sun computer on the NavLab. The program can be run in simulation mode using a NavLab dynamics model developed by the FastNav team at the Field Robotics Center of Carnegie Mellon University was used to simulate the NavLab response. In figure 7, the simulated obstacle environment and scanner data are shown in the global coordinate frame. The initial position of the NavLab is at the (20m 10m) location. The free space is shown as the triangular region with subgoal and NavLab position as two of its vertices. Figure 8 shows the processed obstacle map and the free space (in the local coordinates). Figure 9 shows the complete trajectory around the obstacle as generated by the obstacle avoidance algorithm. This simulation shows the obstacle avoidance algorithm is capable of generating smooth trajectories around the obstacles. In actual runs, the obstacle avoidance algorithm can drive the NavLab at a speed up to 2 /s. Above this speed strong oscillation occurs. The oscillation was caused by the delay in the NavLab control structure where a steering/drive command issued to the NavLab is not executed until 0.5 seconds later. Similar results have been obtained in experimental runs with the actual system.

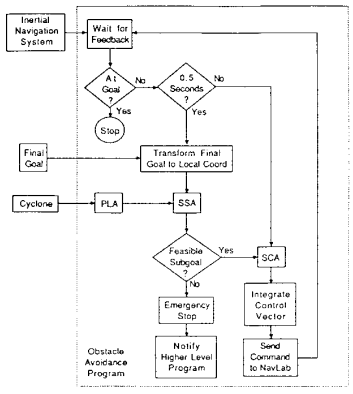

Figure 6: The structure of obstacle avoidance program for the NavLab.

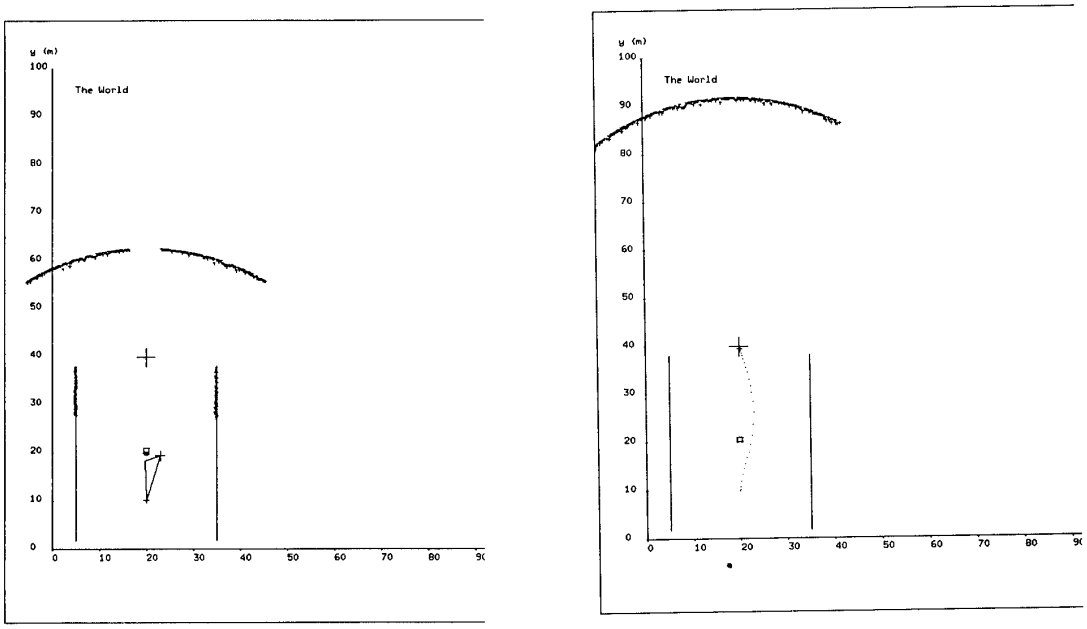

Figure 7: The simulated obstacle environment as viewed in global coordinate frame

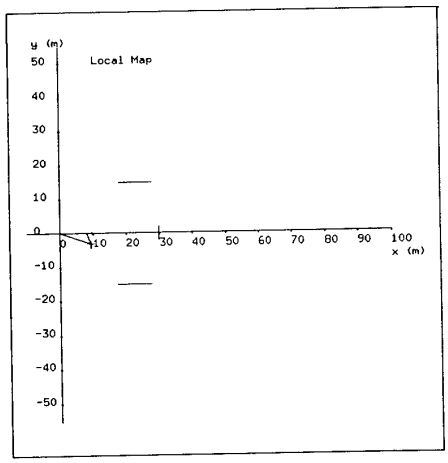

Figure 8: The simulated obstacle environment as viewed in local coordinate frame

References

1. C. Thorpe (editor), *Vision and Navigation: The CMU Navlab,* Kluwer Press, 1990.
2. Dowling, et.al., "Navlab: An Autonomous Navigation Testbed", Tech. report CMU-RI-TR-87-24, Carnegie-Mellon University Robotics Institute, November 1987, Carnegie-Mellon University Robotics Institute Technical Report.
3. D. Feng, *Satisficing Feedback Strategy for Local Navigation of Autonomous Mobile Robot,* PhD dissertation, Carnegie Mellon University, May 1989.
4. B.H. Krogh and D. Feng, "Dynamics Generation of Subgoals for Autonomous Mobile Robots Using Local Feedback Information", *IEEE Trans. on Automatic Control,* May 1989.
5. D.Feng and B.H. Krogh, "Dynamics Steering Control of Convetionally-Steering Mobile Robots", *IEEE International Conference on Robotics and Automation,* IEEE, May 1990.
6. D.H. Ballard, C.M. Brown, *Computer Vision,* Prentice Hall, 1982.
7. T.J. Graettinger and B.H. Krogh, "Evaluation and Time-Scaling of Trajectories for Wheeled Mobile Robots", *ASME Journal of Dynamic Systems, Measurement and Control,* To appear.