Path Planning for Autonomous Vehicles Driving Over Rough Terrain

A. Lacaze, Y. Moscovitz, N. DeClaris, K. Murphy

Computational Intelligence Research Laboratory

University of Maryland,College Park, MD 20742

and

Intelligent Systems Division

National Institute of Standards and Technology, Gaithersburg, MD 20899

Abstract

This paper presents a multiresolutional architecture of planners for obstacle avoidance in outdoor mobility. The planner makes use of off-line dynamical simulations results to efficiently find paths that avoid obstacles. The trajectories are generated by set of steering velocity commands. An implementation of the planner was developed and tested in a vehicle at the National Institute of Standards and Technology (NIST) with promising results. This approach of building the search space for the planner results realistic cost and trajectories for the vehicle.

I. Introduction

The problem of path planning has been explored over the years by many researchers [1, 2, 3, 4, 5, 6]. In the past hardware could not handle the computational requirements involved in on-line path planning causing the systems to be unrealistic for outdoor mobility. Thus, many researchers opted for rule based subsumption architectures that gave good results in well defined environments where the developers of the system could generalize rules for the motion. Unfortunately, motion in the outdoors is a very unstructured environment where good rules that will apply in different circumstances are very hard to come by and extremely expensive to test. However, we are now at a point where faster hardware can allow for planning: the investigation of alternative decisions to achieve a goal before committing. In the future, we predict that these planning systems will also learn from experiences as to reduce these trees of alternatives and further improve their capabilities for finding correct paths.

We propose a planner architecture which is designed to be embedded in the behavior generation (BG) module of the RCS/NASREM architecture designed at NIST [7]. The architecture assumes that there exists goal state that the systems needs to achieve. A multiresolutional world model allows the planners at different levels of resolution to create and execute sequences of commands (plans). World models (WM) at different levels contain knowledge of different coarseness with respect both to space and time. Therefore, since planners investigate possible futures utilizing the language supplied by the WM of the level, their result is also a hierarchy of strings of commands that propagates from the top of the hierarchy (most coarse) to the bottom until the commands are sent to the actuators.

We will concentrate in this paper on one level of this hierarchy. The level of interest is what RCS calls the "subsystem level" because at this level trajectories for the different subsystems of the vehicle are planned. This level receives goals in the form of sequences of points separated approximately by 50 meters and a envelop around within which the subsystem planner must restrict its movement. These lower resolution plans are found by the upper level using a similar planning algorithm.

II. Goal of the Subsystem Planner

The goal of the subsystem level planner is to:

1. follow the envelop assigned by the upper level
2. minimize a cost criterion given by the upper level (i.e. time, distance, energy, etc.).
3. avoid obstacles that will the vehicle put the vehicle at risk.
4. create sequences of points and envelops that the lower lever (higher resolution) can follow.
5. Create this plans fast enough so that plans do not became stale due to changes in the sensed obstacles or changes on the predicted state of the car.
6. send emergency instructions to lower levels in case no path are found.

7. request re planning of the higher level if changes in the state of the environment cause this level planner to do something that was not predicted by the upper level. Some example of these circumstances are: the only paths available are outside of the envelop given by the upper level, or the cost of achieving the goal was under or overestimated by the upper level.

One of the biggest challenges are the timing constraints. The reason for this are the following:

- In order to make an accurate decision for the correct course of action at the present time, the planner must predict (and simulate) the consequences of that this action will have in the future. Prediction and simulations are computationally costly procedures
- because of the limited range that sensors have in this level and the change of position of the car, the information available to the planner changes fast. So, the planner must react in a timely fashion to these changes to assure the safety of the vehicle.

We have some tools that we use in the design of the planner at this level to warranty a quick planning cycle.

- we compute and tabularize the results of off-line simulations so that the most time consuming parts of the prediction of costs will be completed before the vehicle starts its operation.
- RCS has guidelines for the timing cycles at all levels of resolution. If these timing constraints are followed, experience shows that all levels will have sufficient time to re-plan.

III. General Description

We assume that a map is available. This is a Cartesian grid that represents features of the terrain that are or could be significant in the calculation of the cost of traversing each cell. These may include obstacles and other features of the terrain like directional traversability. These features may have time tags associated so that moving obstacles can be represented. They may also have certainty measures.

We have found that the most efficient reference point for the map is the predicted position of the car when the planning cycle is finished. In other words, the car will be at the center of that map looking forward at the end of the planning cycle. The reasons are the following:

1. This allows the off-line simulations to directly apply to this map without coordinate transformations.
2. It is possible to build a hash table that efficiently map properties of pixels into properties of trajectories. We will see this with more detail when we describe the planner.
3. It is simple and efficient to transform and fuse sensor readings into this "local" map.

The planner can be divided into two different parts:

1. All the tasks that need to be calculated and tabulated before the vehicle has started its operation. We will call these algorithms "off-line."
2. The algorithms used during the normal operation of the vehicle. These algorithms use results found by the off line algorithms. We will call these algorithms "on-line."

Since the planner in our example must have 0.5 second or faster response in order to drive at the specified speed, special effort was made to make as much of the computation off-line, and make the on-line calculations as fast as possible.

IV. The Off-line Algorithms

IV.A. Building an ego-graph

The planner builds a graph that shows possible states that the car may visit and the connectivity between these states to be traversed by the vehicle. The trajectories found by the on-line algorithms will be a subset of these graph. We will call this graph which is an overlay on the local map, the ego-graph. An example of the ego-graph used in our example can be seen on Figure 1. In this example the graph shows only points that are within sensor range and horizontal view scanning angle. The Ego-graph is built from 5 layers of 17 points concentrated to the center. This graph can be easily extended outside of the sensor range. In that case, the local map must have permanent obstacles that get shifted as the vehicle moves and certainty associated with the features. In this graph the predicted location of the car at the end of the planning cycle is at the vertex of the inverted cone, facing up.

IV.B. Calculating Cost

In order to accurately calculate the on-line cost of traversing the different edges in the graph, two aids can be built off line: a hash table that transforms local map cell features into features of the trajectories of the wheels of the car at each segment of the graph (stripes), and a cost of traversing that trajectory at different speeds.

In order to develop these two aids an off-line planner of a higher resolution was built. We are interested in having accurate stripes and cost at throughout the ego-graph, but especially closer to the vehicle because:

1. these are the decisions that we have to make now. The decisions further away in time will go

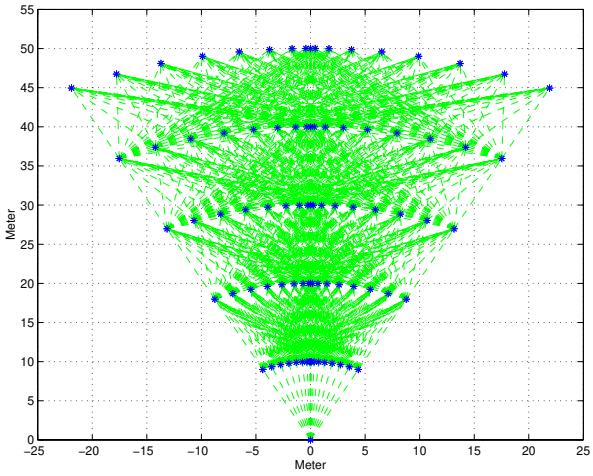

Figure 1: An example of an ego-graph

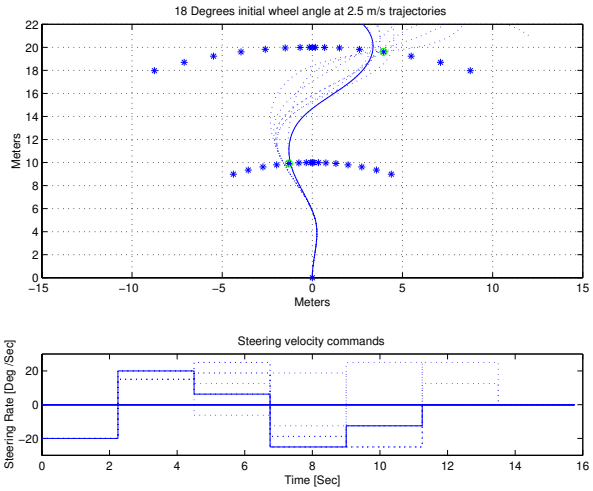

Figure 2: An example of trajectories and their steering velocities commands (the best is solid line)

through more planning cycles before being sent to the lower level.

2. sensors provide more accurate information about the environment closer to the vehicle.

We would like to be able to store the dynamically simulated trajectory for each segment in the ego-graph, unfortunately, this will require very large storage capabilities which are not available, nor will they be available in the foreseeable future. The reason for these is that since the ego-graph is a directed acyclic graph, it has multiple ways to achieve a position in the local map. In other words, there are multiple initial conditions for each segment.

We decided to do a more accurate dynamic simulation of the first two layers and leave the rest of the graph which is further away from the current state in space and time as linear approximations of the trajectories. The dominant affects on the vehicle trajectories are speed, initial steering (wheel angle), and maximum turn rate of the vehicle and maximum velocity to turn the wheels. We found that the initial steering and the limited steering velocity enable the vehicle to reach only part of the front area.

So for the first two layers of the ego-graph (the two rows of points closer to the vehicle) the following steps taken:

1. assuming a starting vehicle velocity and a initial wheel angle, a tree that includes all the possible sequences of finite length (we used 7) of actuator steering commands are applied and simulated using a dynamical model.Example can be seen on Figure 3. Wheel steering resolution was sixth of the maximum velocity. We calculate the trajectories for 10 positive starting wheel angles. Example of trajectories and their steering commands can be seen on Figure 2.
2. for each point in the first layer and for each point in the second layer all trajectories that traverse (within a threshold) those points are separated. Example can be seen on Figure 4
3. the best trajectory is selected according with energy cost function. Additional cost is calculate base on the trajectory angle, from the front, near the second point in order to maintain velocity forward. Example of trajectories and their steering commands can be seen on Figure 2. The two point in the example of Figure 4 demand hard maneuver in low velocity (5 MPH) because of high initial angle of the wheels to the right. We observed that the best trajectory is the one that minimizes the steering velocity/energy when it pass a point. The highest steering velocity is in the beginning of the trajectory and after the first point.
4. the process is repeated for all starting velocities and starting angles.
5. a tree is built where for each point in each of the two layers. The selected trajectory is one in distance of less then 0.4 Meter from the first point and less then 0.8 Meter from the second point. In the example on Figure 2 the trajectories are going through point 5 in the first layer and point 14 in the second layer. The reason for the larger distance in the second point is that we wanted to included more trajectories in the test and until vehicle reach the second point there will be at least 4 replanning cycles to adjust the trajectory. The best trajectory with the highest speed is selected and the speed is recorded. Example can be seen on Figure 5. For points that are not reachable at 20 MPH a lower velocity (15, 10,

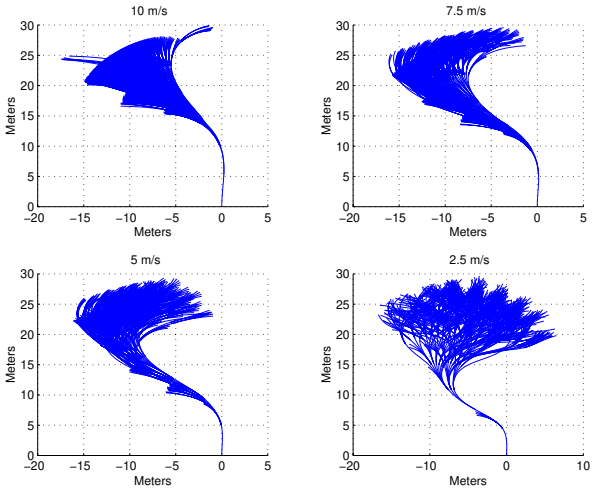

Figure 3: An example of trajectories at four speeds for the same steering velocities commands

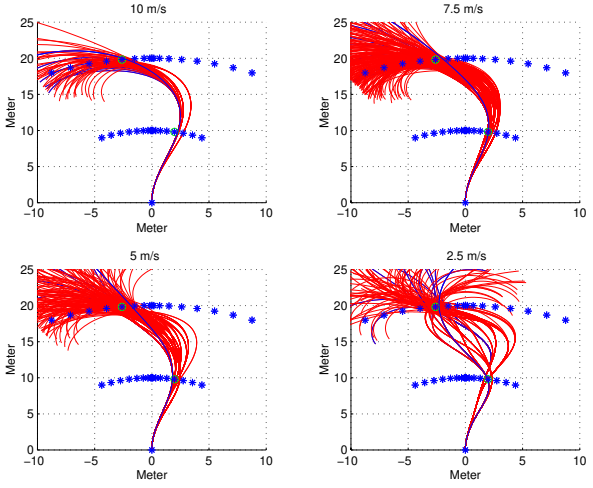

Figure 4: An example of all the trajectories that goes through two points on the ego-graph

5 MPH) trajectories are selected. The basic assumption is that the vehicle can keep trajectory that was simulated in higher speed.

The equations of motion are as follows:

$$P'_N = P'_X = \cos\theta \cos\phi \; u_1 \tag{1}$$

$$P'_W = P'_Y = \sin\theta \cos\phi \; u_1 \tag{2}$$

$$\phi' = u_2 \tag{3}$$

$$theta' = \frac{Kunder}{L} \sin\phi \; u_1 \tag{4}$$

$$v = \cos\phi \; u_1 \tag{5}$$

P_X , P_Y and P'_X , P'_Y are vehicle coordinates and velocities. θ is the vehicle heading and ϕ is the wheel angle limited to ϕ_{Max}. u_1 and u_2 are velocity of the front wheel and steering velocity. u_2 is limited to

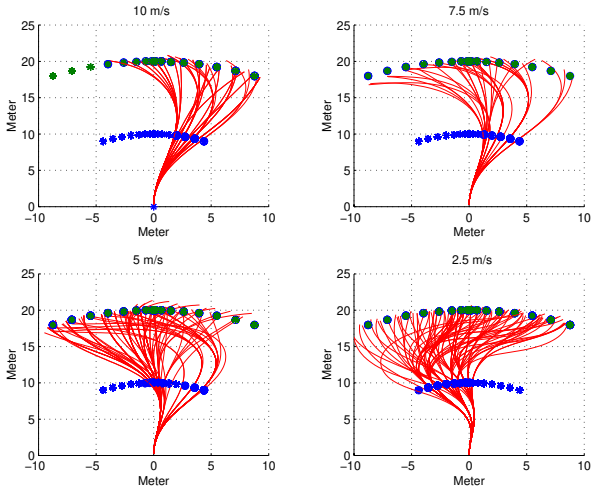

Figure 5: An example of whole set of trajectories at four speeds for a given initial steering

u_2Max so it takes about three second to steer the wheels from one side to another. v is the vehicle speed. The systems equations are based on [8]and Understeer is found in [9]

$$K_{under} = \frac{L}{L + Kv^2} \tag{6}$$

Where L is the distance between the front and rear wheels. v is the vehicle speed in MPH and k=0.0019 for the NIST HMMWV. The output of the offence algorithm:

1. An egograph that show dynamically feasible connections among states in the local map
2. A velocity range for the trajectories.
3. a cost of traversing the segments
4. a hash table that maps cells in the local map into trajectory indices

V. On-line Algorithms

The on-line routines perform the following task:

1. Allocate space for the ego-graph
2. Load the ego-graph. See Figure 6.
3. Allocate space for the cell to trajectory hash table
4. Load cell to trajectory table (created off line)
5. Assign the cost of traversing each of the segments in the graph based on the value calculated off line.
6. For each planning cycle
   (a) Fetch the goal area from the upper level planner
   (b) Identify the points within the ego-graph that fit the constraints set by the upper level goal

(c) Fetch the local map showing the current obstacles centered and oriented on the predicted position of the vehicle at the end of the planning cycle

(d) Using the cell to trajectory table and the local map, See Figure 7 and Figure 8.

i. Temporarily mark the trajectory pieces that are not traversable given the obstacles in the map.

ii. Map other features that can affect the cost of the trajectories (i.e. uncertainty).

iii. Assign time tags to these trajectories if the features are moving as a function of time.

iv. Update the cost of the trajectory pieces using the other features available in the map.

v. Eliminate trajectories that do not fit the current starting state. Two examples: trajectories that starting with different wheel angle, and trajectories that whose speed range is outside of the current velocity

(e) search the graph for the optimal solution that achieves the goal assigned by the upper level. The search procedure is described in the next subsection. See Figure 9.

(f) send result to the executor. The planner output includes the best trajectory for the current speed range and the recommended speed. Speed is adjusted based of the costs of speed depended trajectories.For example the planner might found only low speed trajectories to avoid obstacles in the pathway.

Figure 6 to Figure 9 show an example of planning steps with a real obstacle map. The ego-graph shown is build, initially, only from connection between the points. Later on the first two rows were replaced by the simulated trajectories.

V.A. Search Procedure

1. put the start node, s, on a list called OPEN of unexpected nodes
2. exit if OPEN is empty
3. Remove from OPEN a node, n, at which $f = g + h$ is minimum and place it on a list called CLOSED to be used for expanded nodes. Where g is the cost of traveling to n and h is the predicted cost of traveling to the goal node.
4. If n is a goal node, exit successfully tracing back the solution
5. Expand node n, generating all its successors with pointers back to n

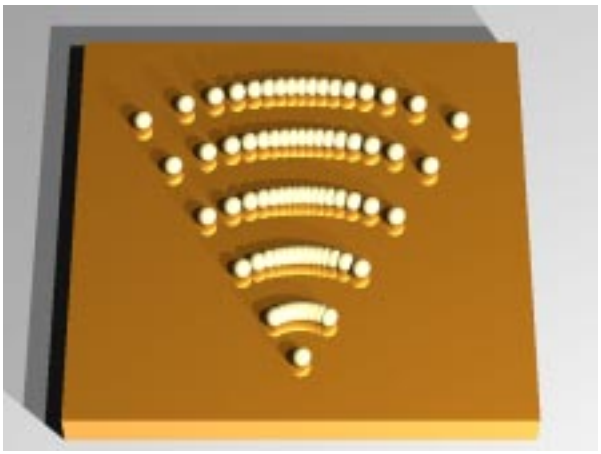

Figure 6: Ego graph points on the sensor footprint

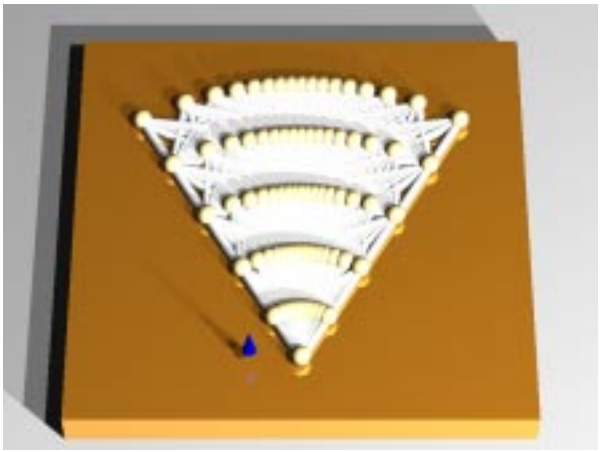

Figure 7: Connected Ego graph not affected by single obstacle out of the path way

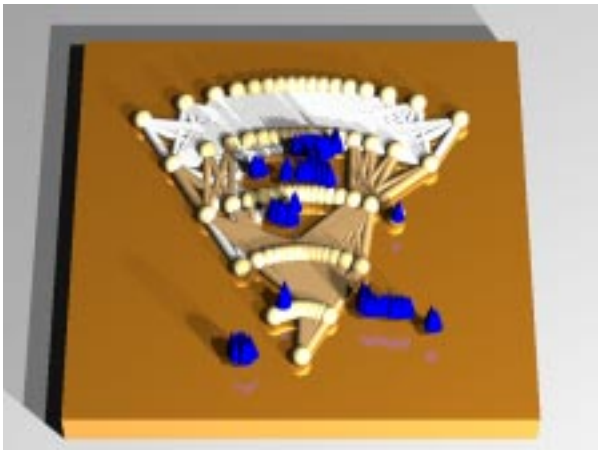

Figure 8: Ego graph affected by multiple obstacles in the path way

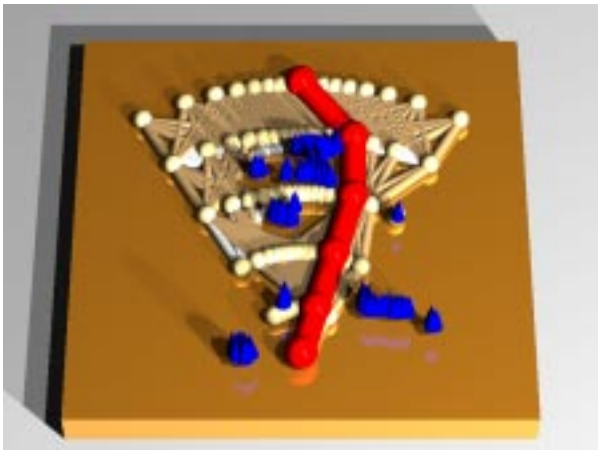

Figure 9: Ego graph with optimum path

Figure 10: The NIST HMMWV

6. For n' successor of n
   (a) Load the pre-calculated, off line,cost $f(n')$.
   (b) If n' was neither OPEN nor CLOSED, then add it to OPEN, assigning $f(n')$ to node n'
   (c) if n' already resided in OPEN or CLOSED, compare the newly computed f(n') with that previously assigned to n'. If the new value is lower substitute it for the old one changing the pointer. If the matching node resided in CLOSED move it to OPEN
7. goto 2

VI. Discussion and Conclusions

The path planner presented in this paper has significant advantages over traditional A* implementations based on the following important ideas:

1. The assumption that a Cartesian grid map, attached to the vehicle front, which includes terrain features and obstacles critical to path planning, is available for off-line computations before the vehicle starts its operation in a particular mission.
2. The implementation of a computing approach that use as map reference point the predicted position of the vehicle when the planning cycle is finished, combines off-line and on-line algorithms (thus overcomes timing constraints by efficiently transforming and fusing sensor data, and hash tables based on realistic trajectories).
3. The use of stored computations that take into consideration vehicle dynamics, (including current vehicle speed, wheel angle and rate of steering, for avoiding obstacles).
4. Dealing with modified data resolution based on distance in front of the vehicle. The two first layers are made of realistic trajectories while the rest of the ego-graph is made from simple connections. The accuracy that the trajectories cross points is best at the first layer (at 10 Meters) and lower in the second layer (at 20 Meters). This concept is based on the assumption that continuos replanning take care for the accuracy near the vehicle.

These ideas enabled us to achieve a hierarchical RCS/NASREM architecture incorporating a subsystem level planner that works in the 50-Meter range which:

1. Provides trajectory and speed control to the lower level for ensuring that the vehicle is kept on a smooth path towards its ultimate goal, and avoid obstacle scanned by the sensor. See NIST HMMWV at Figure 10.
2. Avoids obstacles while operating within the operative range of the vehicle dynamic and computation timing constraints.
3. Vehicle behavior is smoothly and the turning wheel not jumpy.

For path planning of autonomous vehicle over rough terrain the conclusions are:

1. Realistic trajectories that map the vehicle ability to pass through obstacle area are essential.
2. Trajectories must take into consideration vehicle dynamics including current wheel angle and rate of steering.
3. Planner with A* searcher for optimal trajectory is applicable for predefine set of trajectories and an obstacles pixels to trajectory table.
4. The concept of the planning in modified data resolution representation works very well.
5. The NIST RCS hierarchical architecture system provides the basis for determining the range of operation, the planning timing period and the resolution the of data.

6. In order to achieve smooth driving the planner must provide a proper (at least kinematics based) trajectory and speed recommendations to the subsystem (traveler) that keep the vehicle on the chosen route.

VII. References

[1] R. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.

[2] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEE Trans. System Science and Cybernetics*, 4(2):100–107, 1968.

[3] H. P. Moravec. *Robot Rover Visual Navigation*. UMI Res. Press, 1981.

[4] T. Perez-Lozano. Spatial planning: A configuration space approach. *IEEE Trans. Computers*, 32, 1983.

[5] A. Meystel. *Autonomous Mobile Robots*. World Scientific, 1991.

[6] J. Albus, A. Meystel, and S. Uzzaman. Nested motion planning fon an autonomous robot. In *IEEE International Conference on Aerospace Systems*, Westlake Village, CA., 1993.

[7] J. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:473–509, 1991.

[8] G. Pappas and K. Kyriakopoulos. Modeling and feedback control of nonholonomic mobile vehicles. In *Proceedings of the 31st Conference on Decision and Control*, Tucson, Arizona, December 92.

[9] K. Murphy. Analysis of robotic vehicle steering and controller delay. In *Fifth International Symposium on Robotics and Manufacturing*, Wailea, HI, August 1994.