Motion control for fast mobile robots: a trajectory-based approach

Andrew Howard, Alan Blair, Dariusz Walter, Ed Kazmierczak
Dept. of Computer Science and Software Engineering
University of Melbourne 3010, AUSTRALIA
{andrbh,blair,dwalt,ed}@cs.mu.oz.au

Abstract

This paper presents an approach to motion control for fast mobile robots. The approach is based around the notion of a *trajectory*, which describes the time evolution of the robot's position, orientation and velocity over some finite interval. The use of trajectories allows us to take into account a wide range of *dynamic* effects, including (but not limited to) finite accelerations and wheel slip. As a result, this trajectory-based approach is able to produce robust behaviour at high speeds. This paper develops the basic formalism for the trajectory-based approach, and presents some preliminary results from simulation.

1 Introduction

This paper presents an approach to motion control for fast mobile robots. The approach is motivated by the desire to solve problems of the form:

- Move from A to B as quickly as possible.
- Move from A to B, arriving at a particular time.
- Move from A to B, arriving at a particular time, with a particular orientation and velocity.

While there are a wide range of techniques that can be employed to solve such problems, the vast majority suffer from a common weakness: they fail to take account of the robot's *dynamics*. In particular, they tend to ignore the fact that the robot is capable of only finite accelerations. Consequently, while these techniques work well for slow-moving robots, they become increasingly unreliable at higher speeds [Koren and Borenstein, 1991; Fox *et al.*, 1997]. Our approach, on the other hand, accounts for dynamic constraints in an explicit manner, leading to robust high-speed behaviour.

The key innovation in our approach is the fact that it considers robot *trajectories*, rather than individual robot *states*. A trajectory can be thought of as a path through the robot's state space; it describes the time evolution of the robot's position, orientation and velocity over some finite interval. The use of trajectories allows us to take a wide range of dynamic constraints into account, including (but not limited to) finite accelerations and wheel slip effects. As a bonus, the use of trajectories also allows us to implement time-dependent behaviours, such as reaching a goal at a specific time.

The approach has three components: a set of constraints, a set of objectives, and a planner.

- The set of *constraints* determines which trajectories are physically possible. The maximum speed and acceleration of the robot, for example, are expressed as constraints. Each constraint is expressed in the form of a *constraint function* that assigns *costs* to trajectories; trajectories that satisfy the constraints have low costs, trajectories that violate the constraints have high costs.
- The set of *objectives* determines which trajectories are desirable. If we want the robot to reach a particular location, for example, we can define a 'move-to-goal' objective. Each objective is expressed in the form of an *objective function* that assigns costs to trajectories; trajectories that meet the objective have low costs, trajectories that do not meet the objective have high costs.
- The *planner* attempts to find an optimal trajectory by searching the space of all possible trajectories. It looks for a trajectory that minimises a weighted sum of the constraint and objective functions. The planner uses a fast gradient descent method.

There are a number of features of this approach that should be noted.

Firstly, there is the ability to 'fuse' objectives. Through an appropriate weighting of the individual objective functions, it is possible to combine objectives such as 'move-to-goal' and 'avoid-static-obstacle'. It is also easy to add new objectives, making the system very flexible.

Secondly, constraints and objectives are treated in a uniform manner – both are expressed in terms of a function to be optimised. As a result, during the optimisation process, there may be a conflict between finding a trajectory that meets the objective, and finding a trajectory that can actually be executed. In an ideal world, we would eliminate this tension by incorporating the constraints into the problem in a more fundamental way. For example, if the constraints where *holonomic*, they could be used to eliminate one or more of the components in the state space.[1] In the resultant *generalised* state space, trajectories would be guaranteed to satisfy the constraints, eliminating the need for a separate constraint function. Unfortunately, all of the important constraints in this problem (such as the maximum velocity and maximum acceleration constraints), are nonholonomic. The constraint function, then, is a pragmatic mechanism for capturing such constraints.

The final feature that should be noted is the manner in which the planner searches for optimal trajectories. Clearly, the space of all possible trajectories is huge, even for trajectories with a relatively short duration (one or two seconds, for example). An exhaustive search is therefore out of the question. Instead, the planner makes use of a simple (and fast) gradient descent method. Unfortunately, as with all gradient descent methods, this introduces some potential pitfalls:

- The solution may not converge if the function being optimised is not suitably smooth.
- The solution may converge to a local minimum that is far from optimal.
- The algorithm may be too slow to run in real-time.

In practice, we have found that all of these pitfalls can be avoided or mitigated through careful implementation of the algorithm. We will discuss these issues in more detail in Section 4.

The remainder of this paper is structured as follows: in Section 2 we discuss some related approaches to motion control; in Section 3 we present a general formalism for trajectory-based motion control, and apply that formalism to a simple differential-drive robot; finally, in Section 4, we describe some preliminary experiments that demonstrate the plausibility of the approach.

2 Review

Motion control is a well-studied area; there are, it seems, almost as many techniques are there are mobile robots. Rather than attempting to present an exhaustive survey of these techniques, we will simply note the sources of inspiration for the approach described in this paper. Specifically, we will discuss potential field approaches [Khatib, 1986; Arkin, 1989], and the dynamic window approach (DWA) [Fox *et al.*, 1997; Thrun *et al.*, 1998].

Potential field techniques are based on an analogy with electrostatics: the robot becomes a charged particle that is repelled from obstacles and attracted to the goal. Our technique also makes use of this analogy, but it is the robot's *trajectory* that is repelled from obstacles and attracted to the goal. In a sense, our potential fields exist in the (high-dimensional) space of all possible robot trajectories, rather than the (two-dimensional) space of all possible robot positions. As a result, our approach has two advantages over potential fields: it does not suffer from the oscillation effects observed with potential fields (particularly for robots travelling at high speed) [Koren and Borenstein, 1991]; and it is able to express time-dependent behaviours (such as reaching a goal at a particular time). The key disadvantage of our technique with respect to potential fields is that it is computationally intensive.

The dynamic window approach (DWA) attempts to control the robot in *velocity space*, i.e. the space made up of the robot's linear and angular velocity. Since the robot is capable of only finite accelerations, the DWA approach defines a 'dynamic window' in the robot's velocity space. This window includes all of the velocities that can be reached by the robot within the next time interval. An 'objective' function is then applied to all the velocities in this window to determine the optimal velocity; the objective function considers such issues as the distance to the nearest obstacle and the progress being made towards the goal. Our approach is similar, in that we define a *control space* for the robot (which may or may not correspond to the robot's velocity space), and attempt to find optimal control values. Unlike the DWA approach, however, we consider *trajectories* in the control space; i.e. time sequences of control values. This allows us to produce much more complex behaviours from much less complex objective functions (albeit at the cost of extra computation). To some extent, our approach can be thought of as a generalisation of the DWA approach.

3 Formalism

The formalism we use can be summarised as follows:

- Define the *control space* for the robot.

  The control space captures those values that can be *directly* controlled in software. The control space may consist of a set of voltages or currents supplied to the robot's motors, or it may consist of a set of target velocities for the robot's wheels (which are

[1]Holonomic constraints take the form of simple relationships between components of the state space. Non-holonomic constraints, on the other hand, usually take the form of inequalities. See [Goldstein, 1980] for a detailed discussion.

sent to some low level controller).

- Define the *state space* for the robot.

  The state space captures the important kinematic and dynamic values for the robot, such as its position, orientation and velocity. These values cannot be directly controlled in software.

- Determine the *equations of motion*.

  The equations of motion describe the time evolution of the robot's state as a function of the control values. That is, for any given trajectory in the control space, the equations of motion can be used to determine the corresponding trajectory in the state space.

- Write down the *constraint functions*.

  In determining the equations of motion, one must make certain assumptions (that the robot does not slip or topple, for example). In some cases, we can ensure that these assumptions are satisfied by adding constraints to the control space. Each constraint is expressed in terms of a constraint function that assigns low costs to trajectories that satisfy the constraints, and high costs to trajectories that violate the constraints.

- Write down the *objective functions*.

  The desired robot behaviour can be described in terms of a set of objective functions. These functions assign a low cost to 'good' trajectories (where the robot meets or comes close to meeting the objective) and a high cost to 'bad' trajectories (where the robot fails to meet the objective).

- Find the *optimal trajectory* using gradient descent.

  The optimal trajectory is defined to be one in which the robot meets all of the objectives whilst simultaneously satisfying all of the constraints. The optimal trajectory is found using gradient descent to minimise a weighted sum of the constraint and objective functions.

3.1 Example: Differential Drive

As an example, consider a robot with a simple differential drive mechanism. The robot has a pair of independent drive wheels and a pair of passive caster wheels. The motion of the robot is determined by the relative speeds of the two drive wheels, each of which has a PID controller that will (to a very good approximation) maintain a given wheel speed.

The *control space* for this robot has two components: the target left and right wheel speeds (l, r). The *state space*, on the other hand, has five components: the robot's linear and angular velocity (v, ω), and the robot's position and orientation (x, y, θ). Let ζ denote a trajectory through the control space, and let η denote the corresponding trajectory through the state space. In principle, both ζ and η are continuous multi-dimensional functions describing the time evolution of the control and state values, respectively. In practice, we discretize these functions by dividing time into discrete intervals Δt. We can then define ζ to be a *sequence of points* in control space, and η to be the corresponding sequence of points in state space. We write this as:

$$\begin{aligned} \zeta &= \{\zeta_i\} = \{(l_i, r_i)\} \\ \eta &= \{\eta_i\} = \{(v_i, \omega_i, x_i, y_i, \theta_i)\} \end{aligned} \tag{1}$$

where $\zeta_i = (r_i, l_i)$ denotes the control values (the target wheel speeds) at time $t = i\Delta t$, and $\eta_i = (v_i, \omega_i, x_i, y_i, \theta_i)$ denotes the corresponding state. The number of points n in the trajectory is given by $n = T/\Delta t$ where T is known as the *trajectory duration*. When choosing the trajectory duration, one must make certain trade-offs: long durations will tend to make behaviours more robust, but they will also increase the time required to optimise the trajectory. Ultimately, the duration T must be chosen experimentally; see [Howard, 2000] for a more detailed discussion.

If we assume that the robot's wheels do not slip, and that the robot is travelling on a flat surface, simple geometry dictates that the discrete equations of motion for this robot are:

$$\begin{aligned} v_i &= (l_{i-1} + r_{i-1})/2 \\ \omega_i &= (l_{i-1} - r_{i-1})/W \\ x_i &= x_{i-1} + v_{i-1}\Delta t \cos\theta_{i-1} \\ y_i &= y_{i-1} + v_{i-1}\Delta t \sin\theta_{i-1} \\ \theta_i &= \theta_{i-1} + \omega_{i-1}\Delta t \end{aligned} \tag{2}$$

where W is the effective wheel separation. Note that this is a zero-th order approximation to the continuous case. Given a trajectory ζ through the control space, these equations allow us to determine the corresponding trajectory η through the state space.

3.2 Constraints

The *constraints* on the robot's trajectories come in two forms. Firstly, there are constraints that arise from the physical construction of the robot. There will, for example, be a *max-speed* constraint, since the robot's motors are capable of only finite speeds. Secondly, there are constraints that we *impose* in order to ensure that trajectories are predictable. In deriving the equations of motion, certain assumptions were made. Specifically, that the robot's wheels do not slip and that the surface over which the robot is travelling is smooth and flat. While it is relatively easy to guarantee the validity of the second assumption (by restricting the robot to an appropriate environment), the first assumption is somewhat more problematic. It is, for example, relatively

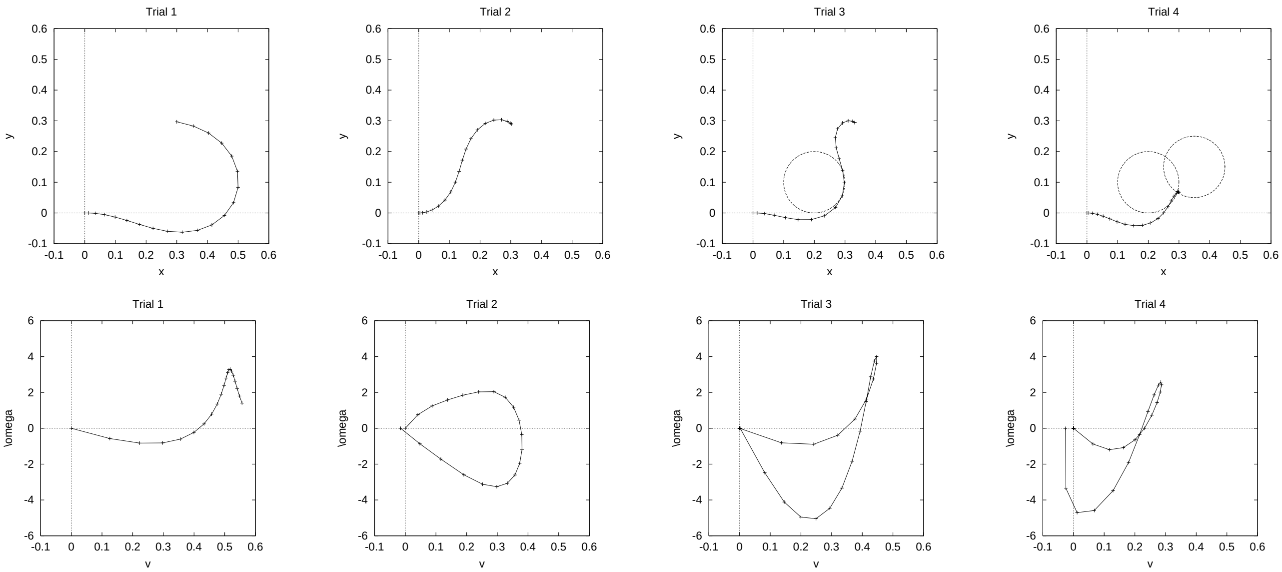

Figure 1: Results for the simulated trials. The top row shows the trajectories in position (x, y) space. The bottom row shows the trajectories in velocity (v, ω) space.

easy to build a robot capable of generating theoretical accelerations of as high as 1g (approximately 10m/s/s). In practice, however, the robot's wheels will usually start to slip at around 0.2g. Once this happens, the equations of motion will no longer hold, and the robot's motion will cease to be predictable. The solution, clearly, is to impose a *max-accel* constraint that limits the robot's acceleration.[2]

Both the max-speed and max-accel constraints can be expressed in the form of *constraint functions*. For the max-speed constraint, we can write:

$$C_{\mathrm{max-speed}}(\eta) = \sum_i \left\{ \begin{array}{ll} (v_i - v_{\mathrm{max}})^2 & \text{if } \mid \mathrm{v_i} \mid < \mathrm{v_{max}} \\ 0 & \text{otherwise} \end{array} \right\} \quad (3)$$

where v_{max} is the maximum velocity. Note that this expression is continuous, and has continuous derivatives; these properties are shared by all of the constraint and objective functions we define, and helps to ensure the success of the gradient descent method used by the planner to find optimal trajectories. For the max-accel constraint, we can write:

$$C_{\mathrm{max-accel}}(\eta) = \sum_i \left\{ \begin{array}{ll} (a_i - a_{\mathrm{max}})^2 & \text{if } \mid \mathrm{a_i} \mid < \mathrm{a_{max}} \\ 0 & \text{otherwise} \end{array} \right\} \quad (4)$$

[2]Note that this analysis of wheel slip effects is extremely is crude, considering as it does only linear accelerations. More complex effects arise when one considers motion around an arc, where centrifugal forces will tend to make the robot slide or topple. These effects can be modelled, and appropriate constraint functions designed. They are, however, beyond the scope of this paper.

where the acceleration a_i is defined as $a_i = (v_i - v_{i-1})/\Delta t$, and a_{max} is the maximum allowed acceleration. Similar expressions can be derived for angular velocities and accelerations.

3.3 Objectives

There are two basic capabilities that any self-respecting robot must have: the ability to reach a goal and the ability to avoid obstacles. Both of these can be framed in terms of simple objective functions.

Move-to-goal

In its simplest form, the move-to-goal objective expresses the desire to reach a particular position *at a particular time*. We can generalise this somewhat to say that the robot should also arrive with a particular orientation and velocity. In other words, the move-to-goal objective can express the desire to move to a particular robot *state*. We can express this in terms of an objective function as follows:

$$O_{\mathrm{goal}}(\eta) = \sum_i \left\{ \begin{array}{ll} [(\eta_i - \eta_{\mathrm{goal}}) \cdot \eta_w]^2 & \text{if } \mathrm{i\Delta t = t_{goal}} \\ 0 & \text{otherwise} \end{array} \right\} \quad (5)$$

where η_{goal} is the goal state and t_{goal} is the time at which we wish to reach the goal. The η_w term is a weight vector expressing the relative importance of the various components of the goal state. For example, if we want the robot to reach a certain pose, but don't care about the velocity, the weight term would be $\eta_w = (0, 0, 1, 1, 1)$. Alternatively, if the velocity is important, but the orientation is not, the weight term would be $\eta_w = (1, 1, 1, 1, 0)$.

Avoid-static-obstacle
The avoid-static-obstacle objective is used to avoid collisions with static obstacles (such as walls). We can express this in terms of an objective function as follows:

$$O_{\text{avoid}}(\eta) = \sum_i \left\{ \begin{array}{cc} (d_i - d_{\min})^2 & \text{if } \mathrm{d_i} < \mathrm{d_{min}} \\ 0 & \text{otherwise} \end{array} \right\} \quad (6)$$

where d_i is the distance between the robot and the nearest point on the obstacle at time $t = i\Delta t$, and $d_{\min}$ is the ideal (minimum) distance. The objective function $O_{\text{avoid}}(\eta)$ is applied for each and every obstacle, and the results summed to produce.

The Planner
The role of the planner is to find a trajectory that meets all of the objectives, whilst simultaneously satisfying all of the constraints. In practice this means looking for a trajectory that minimises a weighted sum of the various constraint and objective functions. We call this weighted sum the *cost* function and define it as follows:

$$U(\eta) = (1-\kappa)\sum_j w_j C_j(\eta) + \kappa \sum_k w_k O_k(\eta) \quad (7)$$

where the first sum is over all constraints and the second is over all objectives. The w_k's are weight terms expressing the relative importance of the different objectives (and constraints). For example, we would generally set the weight associated with the avoid-static-obstacle objective to be large compared to the weight associated with the move-to-goal objective. If the avoid-static-obstacle weight is set too small, the robot may start to 'cut corners' in an attempt the reach the goal. These weights must be set experimentally.

The κ is also a weight term. It expresses the importance of the constraints relative to the objectives. For large values of κ, the objective functions will dominate, resulting in trajectories that meet the objectives, but may not satisfy the constraints. For low values of κ, the constraint functions will dominate, resulting in trajectories that satisfy the constraints, but may not meet the objectives. As with the w_k's, this weight term must be set experimentally.

To find the optimal trajectory, the planner makes use of a simple gradient descent method. The algorithm is as follows:

1. Select an initial control space trajectory ζ.
2. Compute the corresponding state space trajectory η using the discrete equations of motion.
3. Update the trajectory ζ according to the rule:
$$\zeta \longleftarrow \zeta + \sigma \nabla_\zeta U(\eta) \quad (8)$$
where $\nabla_\zeta U(\eta)$ denotes the gradient of the cost function with respect to ζ, and σ is the step size.
4. Compute the new state space trajectory η.
5. Repeat steps 3 and 4 until some termination condition is met. The termination conditions we use are: (1) stop when the cost falls below some threshold; or (2) stop when we run out of time.

Note that there is a significant complication present in this algorithm: we are optimising the *control space* trajectory ζ, but the cost function is defined with respect to the *state space* trajectory η. Consequently, the computation of the gradient term $\nabla_\zeta U(\eta)$ is somewhat involved. See [Howard, 2000] for a complete derivation.

4 Experiments

Figure 1 summarises the results for a group of four trials conducted in simulation. In each trial, the robot was given a certain task, such as reaching a goal or avoiding obstacles. The planner then generated an optimal trajectory for this task. The trajectories have a duration of 2s and a step size of 100ms. All of the trajectories presented in this section were generated using a planner written in C++, running on a standard 450Mhz PC. Each trajectory was subject to 2000 optimisation steps, requiring approximately 1 second of CPU time.

In the first trial, the robot was given the task of reaching the goal position $(0.3, 0.3)$ at time $t = 2s$; no goal was set for the robot's orientation or velocity. The first two plots in Figure 1 show the trajectory generated by the planner, in both position (x, y) and velocity (v, ω) space. The plots clearly indicate that the planner was able to find a trajectory that reaches the goal. Note, however, that there is a residual velocity (since the final velocity was not specified), and that the trajectory does not represent the shortest path to the goal (since this was also not specified). Given the under-constrained nature of the task, there are many optimal trajectories; from these, the planner has chosen a somewhat unusual one.

In the second trial, the robot was given the task of reaching the goal position $(0.30, 0.30)$ with an orientation of $-45°$ and zero velocity. Once again, the plots indicate that the planner was able to generate a satisfactory solution. Given the more constrained nature of this task, the trajectory corresponds more closely to our intuitive expectations; it is quite different from that generated in the first trial.

In the third trial, a single obstacle was introduced; the goal remained the same as in the previous trial. Once again, the trajectory generated by the planner is not the one we might expect – it would seem more natural to go around the other side of the obstacle. Nevertheless, the trajectory *is* optimal, since the cost of the final trajectory is very close to zero. Clearly, this trial demonstrates the sensitivity of the final trajectory to the initial 'guess' used to seed the search process. In this case, the initial guess corresponded to forward motion in a straight line; if we had chosen the initial guess to correspond to a sharp left turn, followed by straight-line motion, the result would be quite different.

In the final trial, a second obstacle was introduced. This obstacle was deliberately placed in the most awkward possible position; as a result, the planner was unable to find a good trajectory. Note that a good solution does exist – the robot could loop around the other side of the obstacles, for example. However, the initial guess was such that the trajectory converged to a local, rather than a global, minimum; this is one of the weaknesses of gradient descent as a search method. We are still investigating the impact of local minima effects, and are considering a range of mitigation strategies. One possible strategy is simply to seed the search process with a number of different (but carefully selected) initial guesses.

Overall, these trials serve to establish the following:

- The objective functions described in Section 3.3 do generate the desired behaviour.
- The planner is able to find optimal solutions, but only if it is given an appropriate initial guess.
- The planner can run in real time on currently available hardware.[3]

Unfortunately, since the trajectories generated in this trials were never actually executed the trials tell us nothing about the accuracy of the equations of motion, nor of the appropriateness of the constraint functions. They also tell us nothing about the impact of local minima in complex environments. For this, we need to conduct trials on a real robot, in a real environment; such trials are currently underway.

5 Conclusion and further work

This paper describes the basic formalism for a trajectory based approach to motion control, and presents some preliminary experimental results. Clearly, much work remains to be done.

Firstly, the experiments described in this paper serve only to establish the *plausibility* of the approach. Many more experiments are required to validate the approach for real robots, in real environments.

Secondly, if we want our robots to move at *very* high speeds, the formalism may need to be extended. In particular, as noted in Section 3.2, the analysis underlying the wheel-slip constraint is extremely crude, and may fail under certain conditions. We may also need to choose a different control space; while PID control has a number of advantages, it has it also has a number of limitations. It is very difficult, for example, to control accelerations accurately (due to controller overshoot and oscillation effects).

Finally, we would like to fully exploit the possibilities of the trajectory-based formalism by experimenting with much more complex dynamic objectives, such as avoid-moving-obstacle or intercept-moving-obstacle.

[3]Given that the planner has not yet been optimised, we are confident that the implementation can be improved to the point where the computation time is less than the step time.

References

[Arkin, 1989] Ronald C Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[Fox *et al.*, 1997] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[Goldstein, 1980] Herbert Goldstein. *Classical mechanics.* Addison-Wesley, 1980.

[Howard, 2000] Andrew Howard. Motion control for a fast mobile robot – a trajectory-based approach. Technical report, Department of Computer Science and Software Engineering, University of Melbourne, 2000.

[Khatib, 1986] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[Koren and Borenstein, 1991] Y Koren and J Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 1398–1404, 1991.

[Thrun *et al.*, 1998] Sebastian Thrun, Arno Bücken, Wolfram Burgard, Dieter Fox, Thorsten Fröhlinghaus, Daniel Hennig, Thomas Hofmann, Michael Krell, and Timo Schmidt. Map leaning and high-speed navigation in rhino. In David Kortenkamp, R Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 21–50. MIT Press, 1998.