Motion Planning with Dynamics

James Kuffner, Jr.

Physiqual
March 1998

Abstract

This paper surveys some recent theoretical work in incorporating the dynamics of robot models in motion planning algorithms. Such work is often referred in the literature as *kinodynamic motion planning*. We first give an overview of the problem in general and how it differs from motion planning with pure kinematics. We then describe three model algorithms, each representing one of three broad categories of existing methods to attack problems involving kinodynamic motion planning. For each algorithm, we review the problem considered, assumptions made, the key ideas, experimental results, and possible extensions. Computational complexity, as well as other issues relevant to practical implementations are discussed, including areas of future research.

1 Introduction

Motion planning algorithms attempt to find a sequence of actions to move a system from an initial to a goal configuration [7]. The seminal paper by Schwarth and Shahir on the *generalized piano movers' problem* sparked a great deal of research efforts aimed at finding efficient solutions for this wide class of problems [9]. Motion planning algorithms must typically operate under two kinds of constraints that restrict the set of feasible motions. *Holonomic constraints* restrict the set of admissible configurations. *Nonholonomic constraints* are constraints on the derivatives, restricting at each configuration the set of admissible velocities.

The vast majority of motion planning algorithms consider only *kinematics*, while ignoring the system *dynamics* entirely. Motion planning that takes into account dynamic constraints as well as kinematic constraints is known as *kinodynamic motion planning*. It can be considered a particular instance of nonholonomic motion planning, since by including the system's derivatives in the configuration parameters (the *state-space* or *phase-space* representation), the dynamic constraints act as either holonomic or nonholonomic constraints. [5]

In general, nonholonomic constraints arise from the presence of one or more rolling contacts between rigid bodies, or from the nature of controls that it is possible to apply to the system in question. When modeling the dynamics of a robotic system using the state-space formulation, nonholonomic constraints usually arise from conservation laws (e.g. angular

	Complete Information (“Piano Movers”)	Incomplete Information (“Sensor-based”)
Kinematics only	Holonomic Non-holonomic	
Kinematics & Dynamics (“Kinodynamic”)	[Bobrow, Dubowsky] [Donald, Xavier, Canny, Reif]	[Lumelsky, Shkel]
	Manipulators or Mobile Robots Typically Off-line	Mobile Robots On-line Execution

Figure 1: Various Categories of Motion Planning Problems. They are divided up into what assumptions are made regarding the completeness of the information available, and what constraints are considered.

momentum conservation). Holonomic constraints in the state-space are typically obstacle-dependent, and map out forbidden regions that are velocity dependent. Configuration-space obstacles ($\mathcal{C}-obstacles$) are used to characterize all configurations of the robot in which it is in collision with one or more obstacles. *State-space obstacles* ($\Phi-obstacles$) include all states of the robot in which a collision with a particular obstacle is inevitable given the constraints on the robot's acceleration and controls. They map out *regions of inevitable collision* in the statespace.

1.1 Problem Statement

Given a kinematic and dynamic model of a robotic system, our task is to design motion planning strategies that move the system from a start state to a goal state while respecting the dynamic constraints. Here, we will consider three methods for kinodynamic motion planning. Each method considered represents one of three broad categories of existing methods to attack problems involving kinodynamic motion planning. For each algorithm, we review the problem considered, assumptions made, the key ideas, experimental results, and possible extensions. Computational complexity, as well as other issues relevant to practical implementations are discussed, including areas of future research.

The example algorithms we will be discussing are categorized in the table in Figure 1. The table divides up motion planning problems into categories based on what assumptions are made regarding the *completeness* of the information available, and what kinds of constraints are considered.

1.2 Motivation

There has been much research on motion planning that deals only with the system kinematics, but very few attempts to incorporate the dynamics. Some researchers feel that there is no need to consider the dynamics, as long as one has access to good controllers. Others suggest that if one is worried about the robot's inertia causing it to deviate from a planned kinematic path, one can just execute the path *very slowly* to minimize the effect of dynamics. These arguments cannot always be justified. It may be the case that the result of a purely kinematic planner will be *unexecutable* by the robot in the environment due to limits on the actuator forces and torques. In addition, one needs to consider dynamics when dealing with imprecise control (a fact of life), when there are time constraints on the execution of the path (cannot move arbitrarily slowly), or one is planning for systems with "significant" dynamics.

The notion of time constraints on the execution of the path naturally leads us to consider the *optimality* of a particular path in terms of the total time it takes to execute. We will discuss this more later on in the paper.

Robots with "significant dynamics" include helicopters, airplanes, wheeled vehicles, submarines, space robots, and even humanoid-type robots. All of these systems can benefit from motion planning algorithms that take dynamics into account, and some even require dynamics to be modeled in order for plans to be useful at all (such as airplanes).

1.3 Incorporating Dynamics

The three categories of algorithms we will consider include strict *two-stage planning*, the *state-space formulation*, and *hybrid methods*. Strict Two-stage Planning essentially means that one first plans kinematically, and then time-parameterizes and/or warps the resulting path to satisfy the dynamic constraints. The state-space (phase-space) formulation, as briefly described earlier, extends the system's configuration parameters to include the derivatives. This nicely unified approach allows one to implicitly take all constraints into account during planning via the state transition equations. Hybrid methods are any attempts to incorporate the dynamics into the planning process without explicitly modeling the state-space. They typically plan in configuration space, but verify dynamic constraints at each step during planning. We will discuss the various assumptions and tradeoffs made by each approach, as well as their relative advantages and disadvantages.

2 Example Two-stage Approach

The central idea behind the two-stage approach is to separate the kinematics-only planning from the dynamics-laden execution. The output of a strictly kinematic motion planner is fed to a controller or other algorithm to incorporate the dynamics as a post-processing step. Here, it is useful to refer to the output of a strictly kinematic planner as a *path*, which has no notion of time or time derivatives. A *trajectory* is a path that has been parameterized by time such that it can be executed in the real world. The job of the second stage in a two-stage approach is to convert the path into a trajectory. One would like to come up with a time-parameterization that not only moves the robot from the start to the goal, but does so in a *time-optimal* fashion. The time-parameterization technique we will examine here produces time-optimal motions, and is presented in detail in papers by Bobrow, Shiller, Dubowsky, and Gibson [1] [10] [11]. We should also note that a similar technique was independently developed by Shin and McKay [12]. It was primarily developed for use by industrial manipulators, but can be applied to any robotic system that has known dynamic equations of motion.

2.1 Problem Statement

The input is a path for the robot's end-effector generated by a strictly kinematic planner. The output is a set of time-parameterized torque functions for all joint actuators, whose execution will follow the given path in a time-optimal fashion, while respecting the system dynamics and torque bounds.

2.2 Relevant Issues

This problem is a highly non-linear, difficult-to-solve optimal control problem. Worse yet, the force and torque bounds one the actuators can be arbitrary functions of the joint angles and their velocities. Also, it has been shown that one cannot simply ignore the non-linearities in the equations of motion as early techniques used to do [1]. Linearizing the dynamics is fundamentally wrong, since the velocity terms have the same significance relative to the acceleration terms for all speeds of movement. Intuitively, this means that time-optimal trajectories are in general, not straight lines, but rather curves in joint space that utilize gravity and non-linear dynamics to help move a manipulator faster to the goal configuration.

2.3 Key Idea

The key idea here is to transform the multi-dimensional optimal control problem into a single-dimensional one by parameterizing the path by a single variable. One possible choice of parameterization is the distance along the path. The single control variable then becomes the acceleration along the path.

2.4 Assumptions

The algorithm assumes that the initial path is continuous, collision-free, and avoids singular configurations. The maximum torque bounds are assumed to be known and that a method exists to compute them from the current joint configuration $q$and their derivatives $\dot{q}$. The robot must have known dynamic equations of motion, and the joint variables $q$must be able to be computed as a function of the path parameter.

2.5 Algorithm Overview

Again, the central idea is to transform the problem into an optimal control problem for a single variable. This reduces the problem to one of finding the *switching points* in the *bang-*

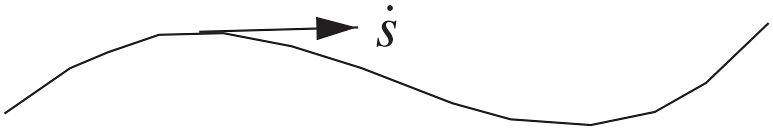

Figure 2: The path can be parameterized by the single varialbe s, the distance along the path. Maximizing the velocity along this path will yield the time-optimal time-parameterization of the path.

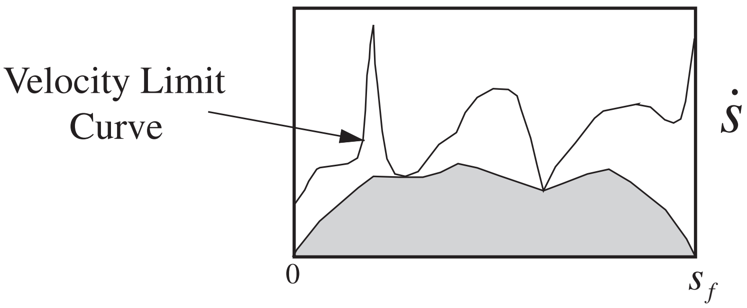

Figure 3: The velocity limit curve as represented on the phase space of the parameter s.

bang optimal control of the single variable. The steps of the algorithm can be broken down into the following:

- Parameterize path by single variable s, the distance along the path.
- Compute the *velocity limit curve* (VLC) based on the maximum torque bounds.
- Using the VLC, compute the switching points along the path for the control variable
- Compute the actuator torques from the control variable using the manipulator dynamics equations.

The time-optimal trajectory will minimize the total travel time, by *maximizing the velocity along the path*, subject to the boundary conditions. Intuitively, this means maximizing the area enclosed under the trajectory curve in the phase space of the parameter s. The velocity limit curve (VLC) is shown in Figure 3. It can be computed from the actuator torque bounds by substitution into the equations of motion of the manipulator. Thus, the actuator torque bounds are transformed into maximum velocity bounds in the plane, and will limit the height of the trajectory curve.

It can be shown that maximizing the velocity essentially means bang-bang control. Thus, the manipulator is either maximally accelerating or decelerating along the path. The problem is reduced to finding the *switching points* between maximum acceleration and maximum deceleration. Standard numerical techniques exist to accomplish this given the velocity limit

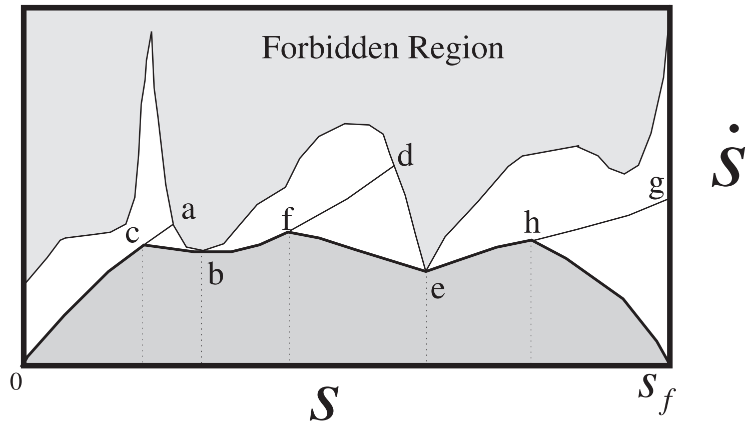

Figure 4: The velocity limit curve is represented on the phase space of the parameter, along with the optimal trajectory with the computed switching points.

curve as a bound on the velocity of the path parameter. In Figure 4, the switching points are shown. Once the switching points are known, one can easily compute the final time-optimal trajectory using the the manipulator dynamics equations, which yield the actuator torques as a function of time.

2.6 Extension to the Basic Algorithm

The basic algorithm returns the time-optimal trajectory for the manipulator only along the *specified* path. It may very well be the case that an even better path lies "nearby" in position space, since a kinematics-only planner was used to find the initial path. This presents the possibility of trying search for a *locally-optimal* trajectory. For example, one could use the control-points of a B-spline representation of the initial path as optimization variables, and the total execution time as a cost function, in order to search for the locally time-optimal trajectory in the neighborhood of the initial path. On can use standard variational techniques here.

2.7 Summary

The algorithm presented is relatively efficient, though still very much an off-line technique. The main problem has to do with the initial path returned by the kinematic planner. Which method for computing the initial path is unclear, and some paths are simply bad candidates to parameterize by time due to the dynamics at particular configurations. Such dynamics singularities, or critical points and *critical arcs* can cause the algorithm to suffer from numerical instabilities. These problems are partially addressed in [11], but the basic computation can become quite slow and complicated. Furthermore, the resulting motion is only time-optimal for the given path or homotopic class. What about the global time-optimal trajectory? This problem is addressed by the kinodynamic planning method described in the next section.

3 Example State-space Approach

The term *Kinodynamic Motion Planning* was popularized by the seminal paper by Donald, Xavier, Canny, and Reif [4], whose method is presented here. Later on, other papers have extended or modified the algorithm, but the fundamental ideas are the same (for example, see [2] [3] [6]. The basic idea is to perform a search on the state-space for a sequence of controls that will move the robot from the start to the goal. Here is the specific instance of the problem that they consider:

3.1 Problem Statement

Given a point mass with velocity and acceleration bounds and a set of polyhedral obstacles in 3D, compute a minimal-time trajectory from the start to the goal, while avoiding obstacles by a safety margin and respecting dynamic constraints.

3.2 Relevant Issues

There is no exact algorithms except for the one-dimensional case, and in 3D, finding an exact solution has been proven to be NP-hard [4]. In addition, true optimal trajectories may graze obstacle boundaries, which is undesirable due to errors in control. We need time-optimal trajectories that are also *safe*, in the sense that they maintain a margin of safety between the robot and the obstacles.

Figure 5 shows three possible kinodynamic solutions to the problem of moving from a starting position and velocity (the start state S), to a goal position and velocity (the goal state G). Path A is the global time-optimal solution, but it is undesirable since it passes to closely to one of the obstacles. Path B represents the global safe time-optimal solution that respects the velocity-dependent margins of safety prescribed (see section 3.5). Path C is an approximately optimal safe path that is within some tolerance of the true optimal safe path B.

3.3 Key Idea

The core idea is to transform the problem into finding the shortest path in a directed graph. The graph is incrementally built by linking discrete points in the state-space by simulating execution of a finite number of bang-bang controls. The graph is searched in a breadth-first manner and stops when it finds the first generated state that lies within some tolerance of the goal state (the goal region). Standard dynamic programming techniques can be applied with the state transition equations coming from the dynamics of the particle, and the cost function involving total time of execution from the start state. Safety is incorporated into the meaning of optimal by a speed-dependent obstacle-avoidance penalty incorporated into the cost.

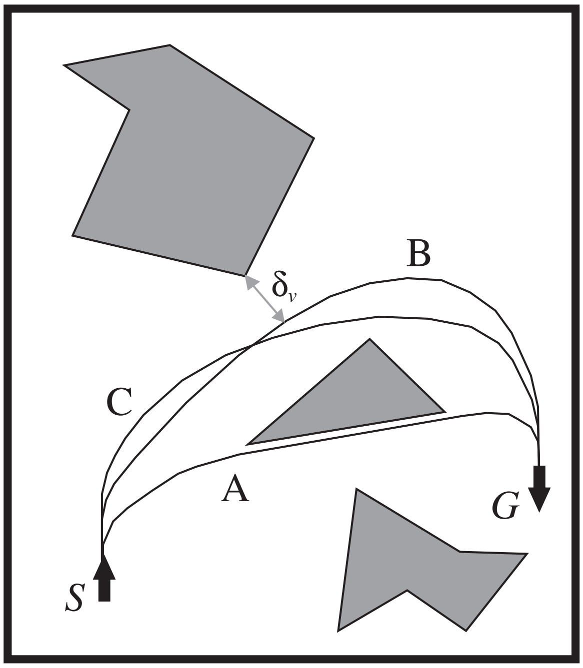

Figure 5: Possible Kinodynamic Solutions. Path A is the global time-optimal solution. Path B is the global *safe* time-optimal solution. Path C is an approximately optimal safe path.

3.4 Assumptions

They assume a point mass moving under Newtonian mechanics in either a 2D or 3D bounded environment of polyhedral obstacles. Thus, a point in the state-space representation is either a 4-dimensional or 6-dimensional vector. The velocity and acceleration bounds are assumed to be known, and are assumed to obey $L_{infinity}$ norms. The algorithm makes us of bang-bang controls in producing the possible states reachable by the start state. A maximal control a is applied for a constant time dt, generating an (a, dt)-*bang motion*. These bang motions trace out quadratic curves in the state-space.

3.5 Safety Margin

Non-zero safety margins are specified through a velocity-dependent penalty function incorporated into the cost. Safety margins are necessary due to imprecision in robot sensing and control. The margin is speed-dependent, as it grows linearly with velocity. Safety margins are both practically motivated, and computationally necessary, since they ensure the existence of a family of approximate solutions nearby the true optimal solution [2].

3.6 Algorithm Overview

The algorithm proceeds as follows:

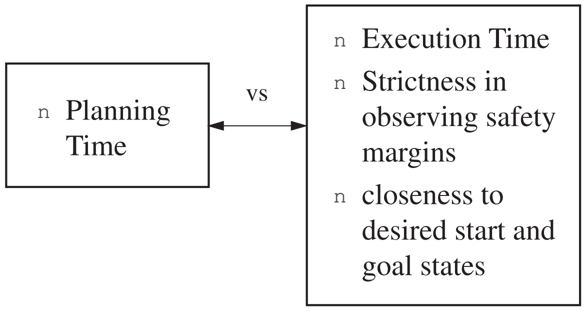

Figure 6: This figure illustrates the tradeoffs between the planning time, the closeness of the computed trajectory to the optimal solution, the strictness in observing the safety margins, and the closeness to the start and goal states.

- Compute the timestep dt, the duration at which to apply (a, dt)-bang controls.
- Initialize the search by letting the initial state S be the "current" state.
- Iteratively apply all controls from the "current" state, check for collisions and safety, and insert the "good" states into the heap of unexplored states.
- Search the graph using breadth-first search until a state within the goal region is found.

3.7 Complexity Issues

The computational complexity is polynomial in both the fineness of the discretization of the state space, and the geometric complexity of the obstacles. All states reachable from the start state under a sequence of bang motions belong to a set of states that lie at the interstices of an underlying, regular grid embedded in the the state space. However, the number of states reachable from a given start state via a sequence of N bang-motions can be exponential in the path length N. Worse yet, the total number of states reachable by an infinite number of such bangs can be unbounded, even in a bounded state-space. Thus, the number of states that need to be examined can be enormous even for small values of N.

3.8 Extensions to Basic Algorithm

The application of the algorithm to a rigid, non-rotating robot is immediate via the C-space formulation. We simply shrink the robot to a point and grow the C-obstacles. Cartesian Robots, or robots with decoupled dynamics bounds can be modeled with minor modifications to the basic algorithm [2]. Open-Chain Manipulators require more work due to revolute joints, though a simplified solution is still possible [3]. Parallel implementations should be possible, as well as employing other kinds of search strategies, such as A^*, or randomized search.

3.9 Summary

The state-space representation allows a straight-forward mapping of the problem to standard dynamic programming techniques. It is proven to return an approximate solution "near" the global time-optimal trajectory (near in total execution time, not necessarily in location). However, this is still very much an off-line technique due to the brute force nature of the search. In 3D, after only 5 levels of search, the number of states to be explored is over 14 million, making the algorithm useless for all but the simplest of problems. There may be ways of pruning the search by not considering states which are very "close" to previously explored states, but in general more research needs to be done towards making the search more efficient. In particular, the use of randomized search strategies to quickly find a solution, even if it is not an optimal solution, would be very useful for many kinds of robotic systems whose dynamics-induced non-holonomic constraints are severe (such as airplanes). This is an open area of research.

4 Example Sensor-based Hybrid Approach

This section describes recent work on motion planning problems of a different flavor that those seen in the previous two sections. In the '*Piano Mover's Problem*, we assume *complete* information about the environment, including the exact location and geometry of all obstacles. In practice, this is rarely the case. Instead, robots typically use sensing to gather information about the local surroundings, possible with the goal of building a complete map of the global environment. Motion planning algorithms that do not assume full knowledge of the environment are referred to here as *sensor-based motion planning*. They operate on *incomplete* information, usually gathered by external sensors such as vision or range-finding.

We describe an algorithm for sensor-based motion planning developed by Lumelsky and Shkel [8] [13], that attempts to incorporate dynamics into a well-known sensor-based planning scheme.

4.1 Problem Statement

Given a point mass robot moving toward a goal in 2D environment amidst finite unknown stationary obstacles of arbitrary shape, we wish to design a controller that operates in real-time, and which guarantees the robot's safety at all times, as well as global convergence towards a goal. Constraints on dynamics, sensing, and control are given as well, and must be taken into account in our solution.

4.2 Relevant Issues

This problem has been referred to as the "Jogger's Problem", since one can think of the problem faced by a jogger navigating in an urban environment [13]. He may turn a corner and be forced to quickly avoid unexpected obstacles, and must decide at each instance what action to perform given the current sensing information and prescribed goals. The authors of the papers reviewed argue that designing a provably-correct algorithm for sensor-based

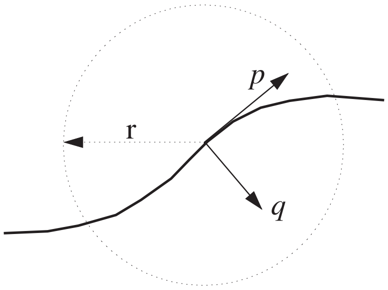

Figure 7: The steering and velocity controls of the robot. p is the forward or backward ("braking") force, and q is the steering force, always orthogonal to p.

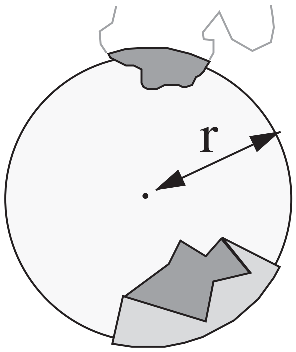

Figure 8: The sensing range, or "radius of vision" of the robot.

motion planning with dynamics requires a single control mechanism. Otherwise, either the convergence properties or the safety of the robot will be compromised. The particular robot controller they designed was constrained to decide which action to perform during the next cycle at a rate of 30-50 Hz.

The robot is modeled as having a limited sensing range, which is reasonable given a robot with range-finders, sonars, or stereo vision. The robot must sense far enough ahead to plan a collision-avoiding maneuver if necessary. There is a fundamental relationship here between the robot's mass, current velocity, and the sensing range that it must maintain in order to guarantee the avoidance of potential collisions. A robot with a larger mass requires better (further) sensing in order to maintain higher velocities. The technique they employ is to locally guarantee the avoidance of all potential collisions in view of the robot's inertia by maintaining an *emergency stopping path*.

4.3 Key Idea

We need to plan the robot's next motion based on the current velocity, control means, and sensing information such that the robot can always resort to a safe, emergency stopping path

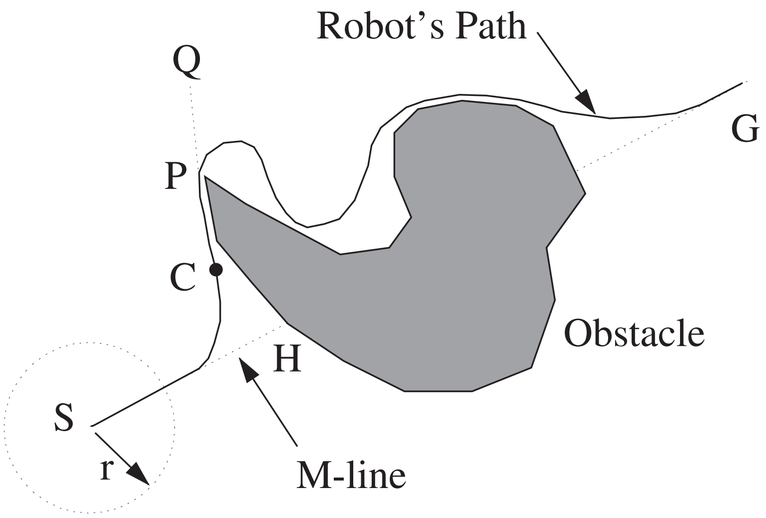

Figure 9: The robot follows the $M - line$ until it senses and obstacle boundary at H. It then follows the contour of the obstacle until the $M - line$ or the goal region is reached. The robot always maintains a safe *emergency stopping path* along the straight line parallel to its current velocity.

in the event of a potential collision with a previously unknown obstacle. The authors do so by always maintaining a straight-line emergency stopping path directly ahead of the robot's current position. Given the robot's current velocity, the robot can be guaranteed to stop at the end of the straight-line segment if needed, by applying a maximal braking force. The general control strategy at each step is to make a motion that will move the robot towards an intermediate target point which lies at the end of the emergency stopping path.

4.4 Assumptions

They assume a Point mass robot moving amongst an unknown 2D environment of unknown, arbitrarily-shaped, *finite* obstacles. The finiteness of the obstacles is important for the global convergence of the VisBug algorithm that they employ. They also assume that the robot knows the relative direction and distance to the goal region at all times. For robots equipeed with GPS or other such sensing devices, this is not too strong of an assumption.

4.5 Algorithm Overview

- Identify intermediate target point T, and make a step in that direction. (T lies on a convergent path, at the edge of the sensing range)

- Maintain a "last resort" emergency stopping path that lies entirely within the visible portion of the robot's sensing range.

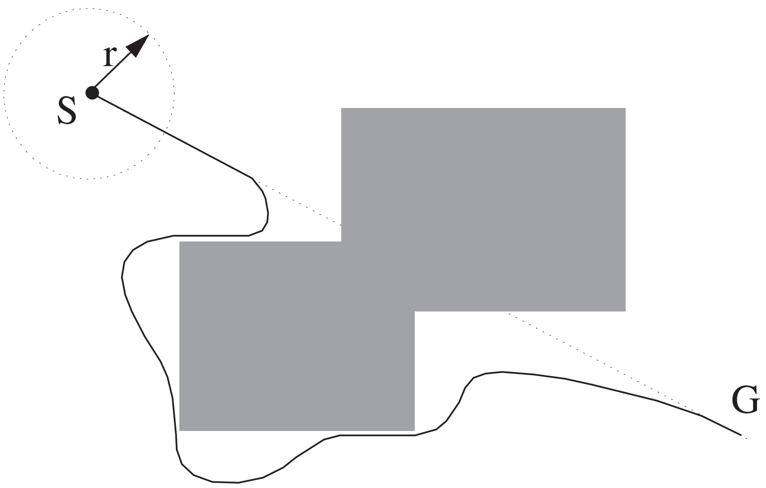

Figure 10: The result of a simulation of the algorithm's performance.

- Employ the "Maximum Turn Strategy", which is essentially bang-bang control in steering.

4.6 Extension of the Basic Algorithm

The algorithm could probably easily be extended to work for polygonal robots in 2D using the C-space formulation. Extending to 3D polyhedral environments seems plausible. A trivial 3D implementation would be to slice the 3D polyhedral environment by a plane containing the start and goal regions and run the 2D algorithm. However, which slice is chosen would radically affect the performance, and it is unclear how to best choose a slice. A more general 3D implementation would likely have better performance.

The additional modeling of more complicated robot dynamics (beyond a point mass) would be nice, such as dealing with robots where angular momentum is an issue. However, with the current algorithm it is unclear how such models could be incorporated. The algorithm could be applied to environments with moving obstacles, but safety or convergence will be no longer guaranteed.

4.7 Summary

This is a relatively efficient on-line algorithm that nicely guarantees global convergence and safety given that the assumptions hold. However, the resulting motion can be far from time-optimal or even reasonable using the VisBug algorithm. Implementing some of the extensions to the basic algorithm and running experiments with real robots would be a promising area for future research.

Figure 11: The result of the simulation with added obstacles appearing around corners.

5 Conclusion

Consideration of a robot's dynamics will be important if we wish to design practical motion planning algorithms that work for systems with time constraints, limited actuator torques, or exhibit significant dynamics. Faster computers with increasingly large amounts of memory will allow more complicated models, including even explicit representation of the state-space. Randomized planning, or other heuristic search methods in the state space, while ignoring the overall optimality could be extremely useful for generating trajectories that implicitly obey dynamics constraints.

References

[1] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators. *IEEE Trans. on Robotics and Automation*, 4(3), 1985.

[2] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14(6):443–479, 1995.

[3] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. *Algorithmica*, 14(6):480–530, 1995.

[4] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993.

[5] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. In *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'96)*, pages 1637–1642, Minneapolis, MN, April 1996.

[6] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 150–155, Cincinnati, OH, 1990.

[7] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[8] V. Lumelsky and A. Shkel. Incorporating body dynamics into the sensor-based motion planning paradigm. the maximum turn strategy. In *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'95)*, pages 1637–1642, Nagoya, Japan, May 1995.

[9] J. Schwarth and M. Shahir. On the 'piano movers' problem, general techniques for computing topological properties of real algebraic manifolds. Technical report, New York University Comp. Sci. Dept., 1982.

[10] Z. Shiller and S. Dubowsky. On computing time-optimal motions of robotic manipulators in the presence of obstacles. 7(7), December 1991.

[11] Z. Shiller and H. Lu. Robust computation of path constrained time-optimal motions. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 144–149, May 1990.

[12] K. Shin and N. McKay. Open-loop minimum-time control of mechanical manipulators and its application. In *Proc. of American Control Conf.*, pages 296–303, San Diego, CA, 1984.

[13] A. Shkel and V. Lumelsky. The jogger's problem : Control of dynamics in real-time motion planning. *Automatica, A Journal of the Int. Federation of Automatic Control*, 33(7):1219–1233, July 1997.