Motion Planning for Skateboard-like Robots in Dynamic Environments

by Salik Syed

Introduction

Motion planning is a branch of computer science concentrating upon the computation of paths for robots or digital actors through complex spaces subject to various constraints. Examples of motion planning include path computation for multilimbed robots in automobile assembly plants, path computation for climbing or walking robots, or even dosage optimization for radiosurgery. In this article, I present new techniques that better compute paths for vehicular robots, specifically those that move through terrain that affects the dynamics of vehicle motion to a large degree. An excellent illustration of motion planning is a skateboarder moving through a series of ramps in order to reach a goal location. In this situation, reasoning is essential—a skateboarder may need to take a circuituous route and take ramps in order to reach a target location.

Unfortunately, computers have very limited "reasoning" abilities. However, they have vast computational power. Due to this, the space of motions of a robot must be searched until a motion that leads to the goal location is reached. Unfortunately, the dimensionality of this space is often sufficiently large that a typical search or even a probabilistic roadmap-based approach (see "Finding Paths"), would not be computationally feasible. In order to overcome this fact, human-defined heuristic functions must be used to more efficiently explore the space defining a robot's set of available actions and their resulting motion. This article will present a generalized overview of the motion planning problem, as well as heuristic methods that improve motion planning for vehicles in dynamic environments.

Formulating the Problem

In order to conduct motion planning, the robot's location in space must first be represented. In order to do this, the motion of an individual robot is generalized to a space known as the **configuration space**. The simplest example of a configuration space is that of a robot that moves on flat land, whose position is parameterized by the vector, $[x,y]$. The configuration space, C, for this robot is the plane, $R2$. Each point in C corresponds to a unique configuration of the robot in the work space, W, which is either 2-D or 3-D Euclidean space. More degrees of freedom in a robot increases the dimensionality of the configuration space, but not the work space. If an orientation to this robot is added, a 3-D configuration space is now created, because the parameterization of the robot is now $[x,y,\Theta]$.

Obstacles in the work space can also be transformed to points in the configuration space. By doing this, the configuration space is split into two regions, free space and obstacle space. The free space, F, is all points, $p \in C$, such that the robot's configuration at point p does not result in collision. The obstacle space is then simply the set of points not in F (see Figure 1).

The obstacle space may be used to represent configurations of the robot that result in self-intersection or unstable configurations and need not represent physical obstacles. It is important to note that the topology and dimensionality of the configuration space may be vastly different from that of the work space. The configuration space is often a high-dimensional space with complex topology. In contrast, the work space is almost always 2-D or 3-D Euclidean space. To demonstrate this, examine the example of the two-jointed arm (Figure 2).

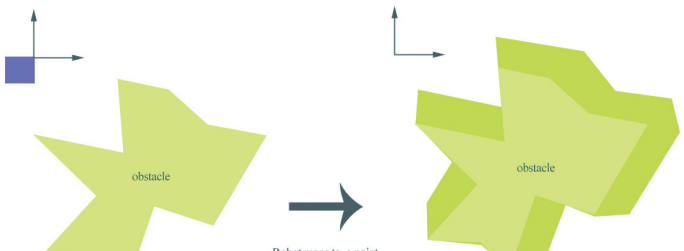


Figure 1: Work space and configuration space of a disc-shaped robot parameterized by [x, y]. Note that the obstacle in the configuration space maps to the set of configurations, which results in collision in the work space—this corresponds to the swept area of the robot about the obstacle. The robot is now reduced to a simple point.

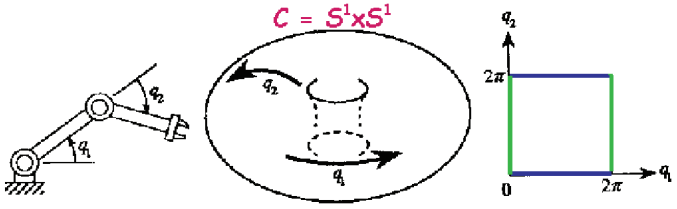


Figure 2: Work space and configuration space of a two-joint robot parameterized by joint angles, [q1,q2].
(Drawing courtesy of Prof. Jean-Claude Latombe)

Topologically, the configuration space of this robot is a torus. This is easy to see because the orientation, 2π, is equivalent to the orientation, 0, for both joints. If this torus is "unwrapped," the correspondence between the work space and configuration spaces can be seen (see Figure 3).

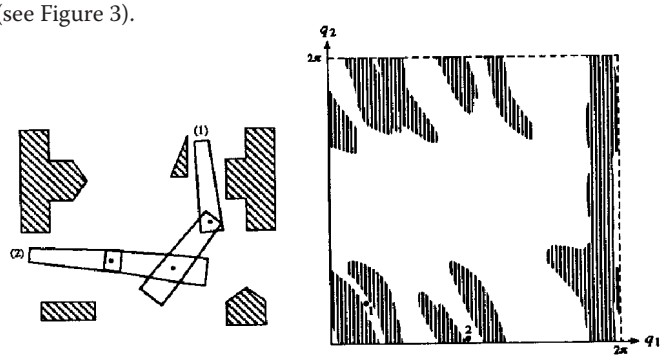


Figure 3: Work space and configuration space of a two-joint robot parameterized by angles, [q1,q2]. Note that the mapping from work space to configuration space is not as intuitive as the disc robot. (Drawing courtesy of Prof. Jean-Claude Latombe)

Moving obstacles in the work space can be elegantly represented by simply introducing another dimension, time, to the configuration space. The configuration space provides a simple mathematical representation of the robot's possible states, given the present environment. The generalized planning problem now reduces to finding a path fully contained in the free space that connects two configurations, p, p'. It is easy to see that while all robot states actually occur in 2-D or 3-D Euclidean space, there are many parameters that define those states. Because there are many such parameters (to define a human's exact state can take hundreds of parameters), this space is often high dimensional and of complex topology.

Finding Paths

The free space can be analyzed using standard search methods, such as gradient descent, or even by explicitly computing and partitioning the free space and obstacle geometry. However, these techniques generally work only for low-dimensional configuration spaces. Explicit computation of the geometry of configuration spaces is extremely difficult for spaces of more than three dimensions. Gradient descent works well for slightly higher dimensions but fails to find solutions when the dimensionality further increases, because of its tendency to get "trapped" in local minima [5].

Because standard algorithms fail to produce results in high-dimensional spaces, a new method known as the **probabilistic roadmap** was introduced. A probabilistic roadmap chooses configurations at random in the configuration space. Configurations that belong to the free space and are within a certain metric of each other are connected via an arc. This arc must also satisfy the property that all points on the arc are in the free space. The primary cost of building the roadmap is checking this arc to ensure that it is collision free. After many samples, this method creates a graph approximating the connectivity of the free space (see Figure 4).

This graph is known as the probabilistic roadmap. The nodes of this graph are often referred to as **milestones**. In order to find a path, the roadmap is simply searched to see if a configuration can be

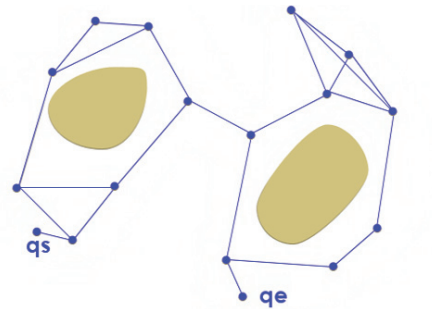


Figure 4: Probabilistic roadmaps of a simple configuration space. Nodes are connected to approximate connectivity of free space.

reached within some metric of the goal configuration. This can be done using standard graph-search algorithms. It is important to note that the path returned by the roadmap may be far from optimal, due to the random nature of milestone selection. The running time of the roadmap, however, is bounded, given that the configuration space satisfies a geometric property known as **expansiveness**. Expansiveness, in the simplest sense, is the lack of "narrow passages" in the space being sampled. To picture this concept, consider two rooms connected by a long hallway. As the width of the hallway decreases, so does the expansiveness of the space. If this expansiveness is sufficiently high, the probability of the planner finding a path grows exponentially with the number of configurations sampled. Thus, this type of planner is said to be **probabilistically complete** [4].

Various improvements can also be implemented to better characterize the connectivity. One simple method is to oversample regions where failed nodes and successful nodes occur within a certain distance of each other—the intuition being that more information is needed near obstacle boundaries than near large empty portions of free space [1].

Planning with Non-Holonomic or Dynamic Robots

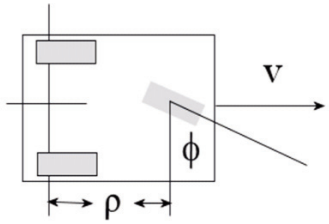
The probabilistic roadmap approach assumes that given two configurations whose connecting arc is in the free space, the robot can travel from one to the other. This works well for multijoint arm type robots or omnidirectional robots, which can move along any arc in free space. This is not the case if there is a robot with a **non-holonomic constraint**. A non-holonomic robot's motion constraints are described by a set of nonintegrable differential equations. The theory behind these constraints is quite complex. For the purposes of this paper, the main point is that such constraints limit the robot from travelling along an arbitrary path in the free space. Given constraints such as these, a path found in the free space may not be feasible because the vehicle cannot transition between two milestones. Because of this limitation, a new planning method was created that extends the basic roadmap; this method is known as **kinodynamic roadmap-based planning** [6]. Instead of randomly sampling configurations in the configuration space, the initial position, p, is the starting point. From p, new configurations are expanded based on a sampling of the **control space**. The control space is simply the set of possible actions the robot can take from the present state. For a car, this may correspond to turning the wheels left or right or accelerating. The application of a control results in new configuration that is reachable from p. Thus, the roadmap takes the form of a tree, with the initial position being the root node. Any node can be chosen to expand

and generate new nodes, typically sampling new nodes where there is a lower density of milestones in a region of the configuration space. Given that the space is sampled uniformly and that the configuration space, or state-time space, in the case of the dynamic robot, still satisfies the property of expansiveness, kinodynamic planning has also been shown to converge exponentially with the number of nodes [3].

To illustrate kinodynamic planning, consider the simple example of a point robot moving in a plane, subject to an acceleration constraint. The state at time, $t = 0$, is the beginning, when the robot is at the start position. From here, the control space is randomly sampled for actions. These actions are integrated forward in time to create nodes at time, $t = 0 + dt$. This process is continued until one of the trees nodes reaches the goal region. The nodes are chosen based on the density of samples in a given region of the state-time space, which is parameterized by velocity, position, and time.

The planner will now be examined. It was implemented to plan for a "skateboard robot" moving along complex terrain, consisting of ramps, walls, and other obstacles. The kinodynamic planning method shown above is the basis for the planner. The planner is implemented for vehicle motion subject to dynamic terrain. The key difference between the point robot and this skateboard robot is that the environment actively affects the dynamics of the robot, in the case of the skateboard. For instance, if the skateboard is moving at a given velocity, that velocity changes when the skateboard goes up or down a ramp. Exploitation of dynamics such as these is essential for path planning.

Planner for the Skateboard Robot



$$\delta\theta/\delta t = (v/\rho)\tan\phi$$

Figure 5: The model used to simulate a car's motion. The turn rate of the body is simply a function of the velocity, v, and steering angle, ϕ.

The initial approach was to first implement a basic flat land planner for a vehicular robot moving about obstacles. To parameterize the skateboard, a simple "bicycle" model was used for motion (see Figure 5). Acceleration was also limited to the forward direction. The root milestone was chosen and then expanded. Controls were chosen at random when creating new nodes. The process for creating a new node was as follows:

1. Choose steering angle uniformly at random (*+/– 35 deg*).
2. Choose a push force uniformly at random.
3. Choose an integration time uniformly at random.

Once these factors were chosen, the parent milestone was expanded by integrating the board state forward, given the set of controls.

To ensure that the state time space was uniformly sampled, a grid-based approach, similar to the method presented in "Randomized Kinodynamic Planning with Moving Obstacles" [3], was used.

The space was subdivided into a four-dimensional grid, $[x, y, z, \Theta]$. Note that neither velocity nor time is taken into account. This means that the sampling of the space is not completely uniform. It is just a fairly good approximation. Next, a cell, or **bucket**, that contains at least one milestone is chosen uniformly at random. Then, from that bucket, a milestone is chosen, again, uniformly at random. This corresponds roughly to sampling based on the density of milestones in a given region.

Using this basic method, the planner was able to find simple paths through a limited number of obstacles. When there were dense sets of obstacles that required careful maneuvering, the planner often failed. Failure was defined as failing to complete the path in a set threshold of milestones in the roadmap. Secondly, if the goal location was extremely far from the start location, the planner would often fail because the integration step when creating milestones was too small. Increasing the average time step was not a sufficient answer, as the robot would now not be able to carefully maneuver in tight spaces. To combat these problems, several new heuristics were introduced, which together are called **complexity-based sampling**.

Complexity-based Sampling

The desired behavior of the robot is that in large free spaces there is a large integration time and a larger average push force. In complex, obstacle-cluttered regions, it is desirable to minimize acceleration (pushing) and have a fairly small integration step, so that any given action is fairly small. A simple way to approximate this behavior is to base the integration time step and push force probability distribution upon the complexity of the configuration space that is in the vicinity of the current node being expanded. The method used to implement this heuristic is to assign each grid bucket a complexity metric, which is based upon the number of failed milestones within this bucket—the idea being that buckets that have many failed milestones are probably in a region with many obstacles. Unfortunately, this metric alone is not sufficient to capture the intricacies of the problem. Take, for instance, a very large free space that eventually leads to a very dense space of obstacles. The problem in this case is that there will be a very high push force in the area that is free, despite the fact that the robot will eventually hit an area with a high density of obstacles. The desired behavior of the robot is that it sees that it will eventually hit obstacles from its current state and should slow down even before it reaches a dense region. In order to do this, communication is introduced between parent and child milestones and between a milestone and its bucket, which represents its course representation in space (see Figure 6).

The tree-shaped structure of the roadmap lends itself easily to this model; a bi-directional tree is created where each node knows its course representation in state space by storing a reference to its bucket. Whenever a milestone is failed, the complexity is increased recursively, going up the chain of nodes from child to parent. If a certain node fails, the bucket of that node has its complexity metric increased. But, in addition to this, the complexity metric of its parent node is also increased. This continues up the chain with a decaying magnitude up to a certain depth threshold. This method allows quick adaptations of the selection distributions for pushing force and integration time for vastly different environment types in a generalized fashion.

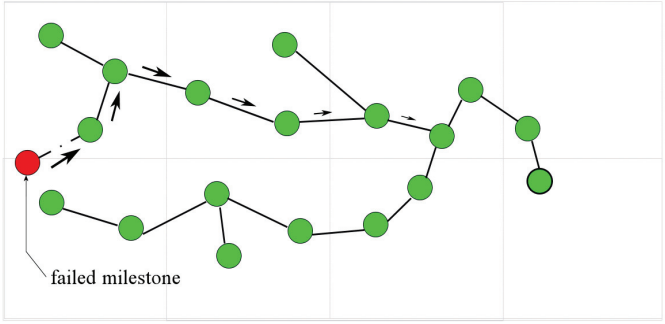


Figure 6: An illustration of a tree-based roadmap. Nodes are placed in buckets, which correspond roughly to their location in state-time space. This is a 4-D space, but for illustration purposes, 2-D is shown. Complexity-based sampling allows inter-node communication. The failure of the last expanded node (red) causes a traversal up the levels of the tree (arrows), causing each parent to change its control selection strategy. The effect is decayed proportionally to the distance in the hierarchy from the failed node.

Goal Distance Heuristic

Another problem often faced is that the search tree grows randomly. In complex spaces, this is not necessarily a bad thing, as a solution often requires a large breadth to find a solution. Unfortunately, in extremely simple situations such as a straight-line path in an open environment, (unnecessarily) random paths may manifest because of the stochastic nature of tree growth. In order to prevent this, a ranking heuristic was also implemented. Each grid bucket was given a ranking. Then, the set of all buckets was placed in a queue (similar to best-first search). The ranking was based on the distance to goal of the average milestone within the bucket, and it was inversely proportional to the number of milestones in the bucket. This metric corresponded roughly to over-sampling regions leading towards the goal, but decaying the rank with each additional expansion. Using this technique, a good balance between expanding likely candidates was achieved. Yet, it did not over sample a single region of the state-time space. In more difficult terrain, this technique alone was not sufficient. Therefore, a probability, *P*, of choosing a random node versus a shortest-path node was introduced instead. This probability was changed based on the total number of failed milestones. This made the planner act randomly in complex spaces. Yet, when in an empty space, it planned straighter paths using the distance heuristic.

Ramp Heuristic

The last problem that was faced was that regions where there were inclines in terrain were often improperly sampled or under-sampled. One major problem is that generally the robot would not have the huge push necessary to build enough speed to clear a ramp. The preparation for a ramp has to be done well in advance (see Figure 7).

In regions where there is a ramp, the robot generally wants to be moving at a fairly high velocity in order to clear the ramp. Furthermore, the robot should be oriented in an angle such that the robot will not veer or fall when on the ramp. To fix this, a similar approach to complexity-based sampling mixed with goal distance sampling was used. By examining changes in robot position in the work space, which milestones correspond to the skateboard being on ramps can be found. Using this information, the control-selection parameters of milestones that eventually lead to a ramp can be changed. In practice, the same approach as complexity-based sampling was used. Milestones that lead to ramps had their control selection parameters changed to favor a higher push force. The priority of milestones that lead to ramps was also increased, meaning that such milestones would be sampled more frequently. These heuristics led to much better coverage of the roadmap on the ramps (see Table 1).

Results

The planner was tested on various types of terrain (see Table 1). The easiest map, *Narrow Hall*, consisted of an L-shaped bend along which the robot had to navigate. The hardest map, Escape, required the skateboard to climb up a ramp and jump off in order to move from one walled area to another. This required the skateboard to move away from the goal in order to have enough room to build-up speed, then push quickly to clear the wall, using the ramp. The combination of all three heuristics found a solution much faster than no heuristic in all cases.

These results demonstrate that heuristic methods are quite useful in many practical cases in problems where the search space is extremely vast. However, in planning problems that involve multiple dynamic maneuvers—jumping over walls or utilizing ramps—executed in

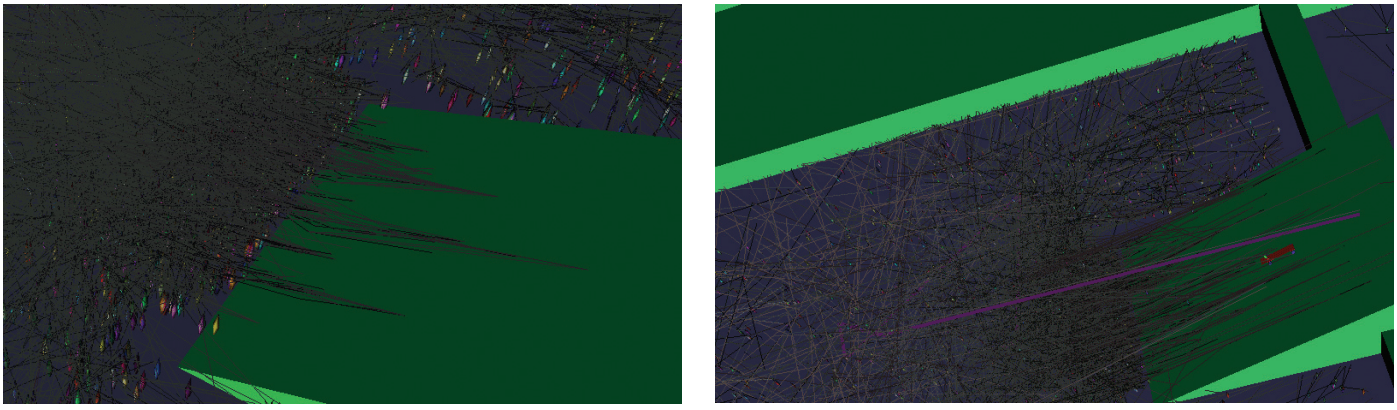


Figure 7: Left: The roadmap fails to climb up the ramp enough to reach the top. Speed needs to be built before getting to the ramp. Right: Improved coverage of uphill areas in the roadmap by using new sampling heuristics. The gray lines represent edges on the tree. The colored objects are failed milestones. Pink lines represent a link between milestones that reach a goal location.

	Planner Tests (Time in seconds/Number of Milestones)				
	None	Distance Heuristic Only	Complexity Sampling	Distance + Complexity	Distance + Complexity + Z reasoning
Obstacle Course	Did not finish	Did not finish	Did not finish	31.0/5232	11.0/2457
Escape	Did not finish	Did not finish	Did not finish	Did not finish	11.5/617
Ramp Climb	4.4/340	12.3/843	34.4/4033	25.2/1701	<1/100
Half-Pipe Climb	Did not finish	Did not finish	Did not finish	18.5/1917	~1.3/140
Narrow Hall	11.2/220	3.8/105	9.8/1277	<1/210	<1/173

Table 1: Results from running the planner on various terrain. All tests were done on a dual-core 1.8 GHz machine with 1 GB of memory. Tests that did not finish exceeded either the time or memory limit.

sequence, this method quickly breaks down. Like many planning heuristics, the approach taken has environments in which the heuristics misguide the search. Fortunately, in most practical cases, these heuristics work well. To improve performance, a multistage method similar to the one used for climbing robots [2] may be useful in finding paths quickly.

Conclusion

After implementing the bulk of the methods, the planner worked fairly well in simple dynamic environments. The skateboard robot was able to properly go up and down a half-pipe and skate up and down ramps. The planner was able to adapt quickly between regions that required complex navigation and those that could be solved easily. For a final test, the board was placed in a room where the only way to get out was by climbing a ramp and jumping out, which the planner was able to do, albeit with a highly non-optimal path. In the future, more robust methods will be needed, as the planner failed to produce results with problems that required utilizing more than two to three ramps. A multistage planning approach may work better. For more information, see [2]. I have presented only a glimpse at the field of motion planning as well as a peek at simple methods for improving planner performance in constrained dynamic environments. There is a vast array of research on the subjects of planning for humanoid robots, climbing robots, and more.

Interested readers can find more information at this Web page: **http://robotics.stanford.edu/~latombe/cs326/2007/links.htm**.

References

1. Boor, Overmars, M. H., and van der Stappen, A. F. 1999. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics and Automation*. 1018–1023.

2. Bretl, T. 2006. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *Int. J. Robotics Resear. 2*, 4. 317–342. **http://robotics.stanford.edu/%7Elatombe/cs326/2007/class14/bretl.pdf**.

3. Hsu, D., Kindel, R., Latombe, J. C., and Rock, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *Int. J. Robotics Resear. 21*, 3. 233–255.

4. Hsu, D., Latombe, J.C., and Kurniawati, H. 2006. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Resear. 25*, 7. 627–643.

5. Latombe, J. C. Probabilistic roadmaps: Basic techniques lecture. **http://robotics.stanford.edu/~latombe/cs326/2007/class6/class6.ppt.**

6. LaValle, S. M. and Kuffner, J. J. 2001. Randomized kinodynamic planning. *Int. J. Robotics Resear. 20*, 5. 378–400.

Biography

*Salik Syed (**ssyed@stanford.edu**) is a sophomore at Stanford University majoring in computer science. His interests are robotics, computer graphics, and artificial intelligence. He would like to thank Doug Green and Jean-Claude Latombe for introducing him to robotics.*