Real-Time Randomized Motion Planning for Multiple Domains*

James Bruce and Manuela Veloso

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
{jbruce,mmv}@cs.cmu.edu

Abstract. Motion planning is a critical component for autonomous mobile robots, requiring a solution which is fast enough to serve as a building block, yet easy enough to extend that it can be adapted to new platforms without starting from scratch. This paper presents an algorithm based on randomized planning approaches, which uses a minimal interface between the platform and planner to aid in implementation reuse. Two domains to which the planner has been applied are described. The first is a 2D domain for small-size robot navigation, where the planner has been used successfully in various versions for five years. The second is a true 3D planner for autonomous fixed-wing aircraft with kinematic constraints. Despite large differences between these two platforms, the core planning code is shared across domains, and this flexibility comes with only a small efficiency penalty.

1 Introduction

Motion planning is problem central to autonomous robotics. As soon as a robot needs to move within a nontrivial environment, the question arises as to *how* it should move to satisfy the constraints posed by its environment. In its general form, the simplest motion planning problem is that of a single query [1]. That is, given some configuration space C that the robot operates within, find a free path from an initial position q_i to a final or goal position q_f. Obstacles pose constraints on valid configurations in C, thus we can subtract them from the full space, leaving the remaining "free" configuration space C_f in which the robot can move without hitting any obstacles. The path planning problem then becomes finding a continuous curve $p(s) \in C_f$ for $s \in [0, 1]$ where $p(0) = q_i$ and $p(1) = q_f$. For some robots, additional constraints are needed due to limitations of the robot itself. These could be kinematic limitations, such as a car-like robot's

* This work was supported by United States Department of the Interior under Grant No. NBCH-1040007, and by Rockwell Scientific Co., LLC under subcontract No. B4U528968 and prime contract No. W911W6-04-C-0058 with the US Army. The views and conclusions contained herein are those of the authors, and do not necessarily reflect the position or policy of the sponsoring institutions, and no official endorsement should be inferred.

G. Lakemeyer et al. (Eds.): RoboCup 2006, LNAI 4434, pp. 532–539, 2007.

steering limitations, or they could be dynamics constraints such as maximum acceleration. These are collectively known as kinodynamic constraints and take the form of additional constraints on $p(s)$.

While the problem of motion planning has been studied extensively, the general trend of research has been concerned with solving successively more difficult problems. While it is important to expand the problems solvable given generous time and computing resources, advances may not translate directly to improvements at the other end of the spectrum. This other end consists of relatively "easy" problems, but with much tighter time bounds and limited available computation. It is these latter problems which abound in mobile robotics, and with which this work is concerned. The vast majority of mobile robots exist on a 2D surface, while those that do move freely in 3D, such as unmanned aerial vehicles (UAVs) typically do not encounter dense obstacles (i.e. C_f occupies a large fraction of C).

In fairly static domains, two-stage "multi-shot" planners such as PRM [2, 3] work well. PRMs separate planning into a learning phase, which builds a finite graph model G of C_f, and the query phase, which maps the problem to graph search on G. In highly dynamic environments however, learned models quickly become obsolete. This encourages the use of "one-shot" planners which only concern themselves with solving a single query given no apriori model of C_f. Among the fastest one-shot planners are the *RRT* family of randomized planners [4, 5, 6]. RRT planners incrementally build a tree in C_f while they search for the solution to a planning problem. A typical RRT search is as follows. First, q_i is added as the root of a tree. Then we iterate the following: Pick a draw a random sample q_r from C, find the closest vertex v in the current tree, then grow the tree toward q_r using an extend operator. The end of the extension is added to the tree with v as its parent. The first few steps of such a tree are shown in Figure 1. As this process is iterated, the RRT grows to fill the free space, tending toward an even distribution. The major variables in the method are in how we draw the random samples and in how the extend operator works. Two adjustments turn the space-filling RRT into a planner. First, we can throw out any extension segment that would hit an obstacle, thus restricting the tree to C_f and guaranteeing that any path from node to node is a valid path in free space. Next, we can alter the random target distribution by picking the goal configuration some fraction of the time, thus biasing the tree to grow toward the goal in a more directed fashion. Once a node is added to the tree that is sufficiently close to the goal configuration, we can trace up the parent pointers in the tree to recreate the path from q_f to q_i, the reverse of which is a plan [4].

1.1 Approach

The ERRT planner developed in previous work [6] builds on RRT and offers a navigation approach for mobile robots using iterated replanning. Each control cycle, a new plan is developed, rather than waiting for an error condition to occur before replanning. This allows the planner to deal with both small and large errors in the same way, and thus is highly tolerant of position jumps and action

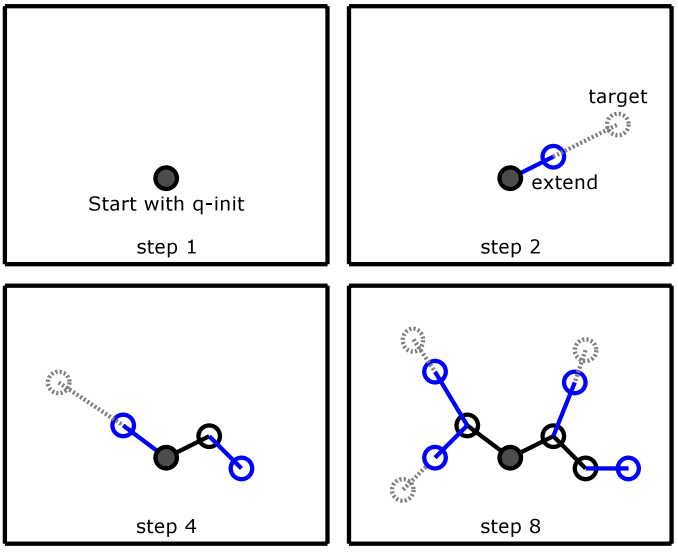

Fig. 1. Example growth of an RRT tree for several steps. At each iteration, a random target is chosen and the closest node in the tree is "extended" toward the target, adding another node to the tree.

error which is invariably present in physical robots. To speed up replanning, and decrease the variance from cycle to cycle in the plans, ERRT also introduces the concept of a waypoint cache. This is a fixed-size bin with random replacement into which states from previous plans are added. During RRT's target point selection, some part of the time waypoints are chosen instead of random configurations or q_f. This biases the planner to search along previously successful plans, both decreasing the running time of the planner and resulting in more stable solutions during successive iterative replans.

In our work, we have built upon ERRT to create a planner for mobile robots which span a wide range of parameters. The first platform are robots built for the RoboCup F180 "small size" league [7]. The field of play is a carpet measuring 4.9m by 3.8m, similar to that shown in Figure 2. Due to its competitive nature, teams have pushed robotic technology to its limits, with the small robots travelling over $2m/s$, accelerations between $3-6m/s^2$, and kicking the golf ball used in the game at up to $10m/s$. These speeds require every module to run in realtime to minimize latency, all while leaving enough computing resources for all the other modules to operate. Since five robots must be controlled at up to 60Hz, this leaves a realistic planning time budget of about 1ms for each robot. The robots themselves are holonomic, and controlled through a local obstacle avoidance mechanism which incorporates dynamics. Thus the planner is free to operate without kinodynamic constraints.

The second platform is an autonomous unmanned air vehicle (UAV), and in particular an autopilot designed for the small UAV shown in Figure 2. The

Fig. 2. Two teams are shown playing soccer in the RoboCup small size league (left), and the RnR RPV-3 unmanned air vehicle (UAV) (right)

UAV and thus the planner must operate in 3D at low to intermediate altitude, avoiding both the terrain and user specified "no-fly" zones as obstacles. The UAV has highly constrained kinodynamics; Despite is small size and $3.5m$ wingspan, the minimum turning radius is $300m$, while climb and descent rates are limited to $5m/s$. The autopilot can accept new sets of waypoints every few seconds, leading to a much less constrained timing schedule compared to the RoboCup robots. However due to the 3D nature of the problem and the constrained kinodynamics, the problem to be solved is much more difficult.

While widely varied, the two domains still offer similarities we may take advantage of. They are both mobile agents operating primarily in ordinary 2D or 3D space, and both can be conservatively but acceptable modelled by a bounding circle or sphere. The remainder of this paper describes the planner we have implemented based off of a generalized extension of the ERRT approach. The next section describes an abstract domain interface that allowed us to share core planning code across the two domains without sacrificing the planners' execution speed. The following section then describes our collision detection approach which takes advantage of bounding spheres to implement exact swept-volume collision checks with high efficiency, while keeping a straighforward implementation to add new types of obstacles.

2 The Domain Interface

The domain interface resulted from an attempt to unify platform interfaces so that common planning code could be developed. In traditional planning work, there are typically three primary modules: Planner, collision detection, and a platform model. While this approach works well for robots of relatively similar type, the communication required between the platform model and collision detection are through the planner, complicating the interface so that the planner needs to know much more about the domain than is really necessary. Thus the

current approach was devised, in which platform model and collision detection are wrapped into a single "domain" module that the planner interacts with. Internally, collision detection and the platform model are implemented separately, but importantly the planner doesn't depend on anything involved in the communication between the two. This allows states in the configuration space to be a wholly opaque type to the planner (denoted by S), which needs only to be copied and operated on by the domain's functions. Additionally, to speed up nearest neighbor lookups, the state must provide bounds and accessors to its individual component dimensions. This allows the planner to build a K-D tree of states so that linear scans of the tree are not necessary for finding the nearest state to a randomly drawn target. While the traditional architecture can be made nearly as flexible, it typically does so at a cost in efficiency. The domain interface approach leaves open the opportunities for improved collision detection speed that can come with constraints or symmetry present in the agent model. The domain operations are as follows:

- *RandomWorldTarget():S* - Returns a state uniformly distributed in C
- *RandomGoalTarget():S* - Returns a random target from the set of goal states
- *Extend(s_0:S,s_1:S):S* - Returns a new state incrementally extending from s_0 toward s_1
- *Check(s:S):bool* - Returns true iff $s \in C_f$
- *Check(s_0:S,s_1:S):bool* - Returns true if swept-sphere from s_0 to s_1 is contained in C_f
- *Dist(s_0:S,s_1:S):real* - Returns distance between states s_0 and s_1
- *GoalDist(s:S):real* - Returns distance from s to the goal state set

Using these primitives, an RRT planner can be built which operates across multiple platforms, and does so without sacrificing runtime efficiency. One could say that it moves most of the important code into the domain itself, making the planner itself simplistic. However, the crucial difference is that the code in the domain is relatively straightforward and self contained, while the intricate interactions and practicality driven fallback cases of planning reside in the core planning code. Thus one could implement a domain with little or no knowledge of path planning, reaching a core goal of general modular programming.

3 Fast Collision Checking

Collision detection is a research area in its own right, and has been extensively studied (for a good survey see [8]). However for our planner we can achieve higher performance than general solutions by taking advantage of some simplifications present in our domains. First, since the mobile robots are bounded by circles or spheres, we need not check two complex shapes against one another to test for collision; We merely need to be able to test a sphere against the possibly complex environment. This results in our planner being pessimistic, but allows us one critical advantage: The ability to model continuous time trajectories in the

collision check. Many implemented planners using general collision checkers represent time trajectories as fixed steps along the path. Unfortunately this creates an uneasy tradeoff between planning time and safety. We take the conservative approach of bounding the agent in a simple shape, but then use exact collision checking for time trajectories given that shape. The result is a planner that does not sacrifice safety in obtaining its fast execution times.

In our originally developed 2D implementation, checking a line-swept circle against various geometries proved easy enough to implement for various obstacles geometries, although adding a new type of obstacle proved quite tedious. Supporting 3D queries for the UAV would have resulted in a much more complex implementation to solve the trajectory-swept-sphere problem, so a different solution was sought. Our new approach required implementing only a single primitive for each obstacle: distance from the obstacle to a point in space. From this primitive, all the other required queries could be derived numerically. In particular, this included the swept-sphere query required for checking trajectories. The method works as shown in Figure 3. For any particular point, the current distance to obstacles, or clearance, determines how far along a trajectory is safe. The checker can then step forward by that distance, and recursively check the remaining swept area. The figure shows a dark blue trajectory to be checked, the light blue is the current clearance, and the red spheres show the steps that can be safely taken each iteration. Normally, few iterations are required, although the algorithm can take many steps if it is very close to an obstacle. This is handled by failing after a certain number of iterations have been exceeded. Though this is yet another pessimistic approximation, long paths running very close to obstacles are not typically desirable for execution by mobile robots anyway.

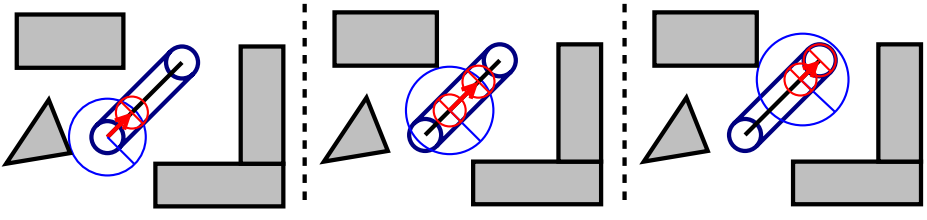

Fig. 3. An example of checking a swept circle using only obstacle distance queries to iteratively step forward along the swept path

While in our current implementation, only line-swept-spheres are supported, the distance query stepping method of checking a swept sphere can be applied to other trajectory functions as well. For some continuous trajectory function $x(t)$ where time $t \in [0, t_f]$, starting from an initial position $x(0)$ with a distance of at least $D(x(0)) = d$ from all obstacles, we need only find the first t such that $\|x(t) - x(0)\|^2 = d^2$, or verify that no such t exists for $t \in [0, t_f]$. For functions with bounded curvature, such as lines and circular arcs, these calculations are straightforward. In particular, for a linear trajectory defined by $x(t) = a + bt$,

then the solution is $t_i = \sqrt{d^s - r^2}/\|b\|$. If $t_i > t_f$, then the trajectory is verified to be free, otherwise a recursive check must be made with a new trajectory starting at time t_i. Thanks to the square root, t_i increases quite rapidly from zero even with very small clearances, resulting in few steps needed for a typical check, and making the approach efficient in practice.

The obstacles implemented by our collision checker range from the obvious simple geometric shapes all the way up to 200x200 terrain meshes for the UAV planner. While the geometric shapes are straightforward to develop a distance metric for, the terrain posed the challenge of efficient distance calculation. Since the terrain is a 2D grid projected into 3D as a low-curvature mesh (i.e. a relatively flat manifold), we broke it up using a 2D K-D tree in grid space, and bounded the individual nodes with an axis-aligned bounding box in the 3D space. To query the distance from a point to the terrain, we follow the branches of the K-D tree nearest first, calculating a distance to the actual terrain once a leaf node is reached. After that, the remainder of the traversals and be compared against this known minimum and pruned if the bounding box is further than the current minimum. This aggressive pruning and the relatively flat nature of actual terrain meshes yields near logarithmic access times for the queries.

4 Results and Conclusion

In the RoboCup domain, we found the planner to work well in practice, helping our team consistently place within the top four teams, with comparatively few penalties for aggressive play. In testing, the planner in the RoboCup achieved execution times below 1ms to meet the tight timing requirement of the small-size system. It averaged just 0.5 ms per run, compared to an average of 0.9 ms for a baseline RRT implementation lacking a waypoint cache. In practice this means that the ERRT implementation can expand more nodes than plain RRT while remaining within the 1 ms planning envelope. Next, while reusing the same code and the same generic collision detection framework, a UAV planner was created by writing a new domain implementation. An example is shown in Figure 4, using actual data for the 12km x 12km area surrounding Reno, Nevada, USA. It has been tested driving a vendor-provided UAV simulator and a real hardware autopilot for an existing UAV. Depending on the problem difficulty, it runs from 0.5s to 2.0s per query on a modern computer. Eight ERRT searches are generated per query, and the shortest plan from a successful search is returned. Due to local minima in the distance metric resulting from kinematic constraints, running several independent runs generated more consistent results than running one large plan. In the RoboCup environment, path consistency is achieved by a high waypoint cache bias in ERRT [6].

Development of a unified planner for multiple mobile robot platforms provided many insights that would be difficult to determine if only one platform or similar platforms were considered for an implementation. However there are still many interesting areas of further work. First and foremost, the relationship between distance metric, kinodynamic constraints, and accelerated nearest-neighbor search

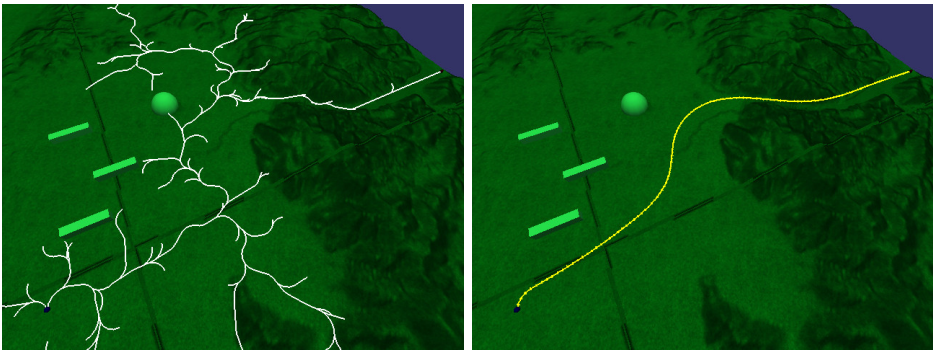

Fig. 4. A kinodynamicly-limited search tree (left), and the corresponding simplified plan (right) for the UAV. The plan length is approximately 13km with waypoints every 100m.

should be explored. Developed good distance metrics for kinematically constrained platforms is possible, but tedious, and more seriously it prevents most forms of accelerating nearest-neighbor search to fail because the triangle inequality is no longer satisfied. Using Euclidean distance worked, but generated local minima that could only be avoided by rerunning the planner several times, which is an inelegant solution. Better approximations which still allow the use of fast geometric data structures most likely exist.

References

1. Latombe, J.-C.: Robot Motion Planning. Kluwer, Dordrecht (1991)
2. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabalistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12, 566–580 (1996)
3. Kavraki, L.E., Latombe, J.-C.: Randomized preprocessing of configuration space for fast path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2138–2145. IEEE Computer Society Press, Los Alamitos (1994)
4. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. In Technical Report No. 98-11 (October 1998)
5. James, J., Kuffner, J., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society Press, Los Alamitos (2000)
6. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS) (2002)
7. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife (1995)
8. Lin, M., Gottschalk, S.: Collision detection between geometric models: A survey. In: Proc. of IMA Conference on Mathematics of Surfaces (1998)