2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Randomised MPC-based Motion-Planning for Mobile Robot Obstacle Avoidance

Alex Brooks and Tobias Kaupp and Alexei Makarenko

Abstract— **This paper presents an algorithm for real-time sensor-based motion planning under kinodynamic constraints, in unknown environments. The objective of the trajectory-generation algorithm is to optimise a cost function out to a limited time horizon. The space of control trajectories is searched by expanding a tree using randomised sampling, in a manner similar to an RRT. The algorithm is improved by seeding the tree using the best control trajectory from the previous iteration, and by pruning branches based on a bound to the cost function and the best trajectory found so far. Performance of the algorithm is analysed in simulation. In addition, the algorithm has been implemented on two kinds of vehicles: the Segway RMP and a four-wheel-drive. The algorithm has been used to drive autonomously for a combined total on the order of hundreds of hours.**

I. INTRODUCTION

A fundamental competency for an autonomous mobile robot is the ability to plan and execute collision-free paths through unknown environments in real-time. Historically, research on obstacle avoidance has focussed on indoor robots whose small sizes and low speeds allow momentum to be ignored with negligible consequences. However, as robotics moves out of laboratories, it becomes necessary to consider kinodynamic constraints: a controller cannot change the pose of a robot directly, but rather applies controls (*e.g.* accelerations) which modify the pose through some process model.

The approach taken in this paper is to apply model-based receding horizon control, using sampling-based methods to produce trajectories. Figure 1 provides an overview of the controller. The planning module is provided with *a priori* models of the dynamics of the vehicle and the dynamics of the environment. At each iteration, the planner uses sensor information to update estimates of the position and velocity of the vehicle, and of obstacles in the world. Using these estimates, plus a set of goals provided externally, a trajectory generator plans a feasible collision-free path over the next few seconds. The first step of this plan is executed, then the entire process is repeated. Replanning at every iteration allows the controller to adjust its plan in response to errors in its predictions.

This paper focusses on the trajectory generation step. The problem is challenging: the time allowed for a single iteration of online planning is tightly constrained, however the space of possible command sequences over a period of several seconds is enormous. Randomised sampling-based methods, particularly RRTs [9], have proven extremely successful at tackling path-planning problems in continuous high-dimensional spaces. Relative to classical methods such as A* [8], sampling-based methods avoid the state explosion associated with discretisation of the state space. The ability to deal with high-dimensional state spaces is particularly important when dealing with differential constraints, because the addition of velocities to the state space increases the dimension of the state.

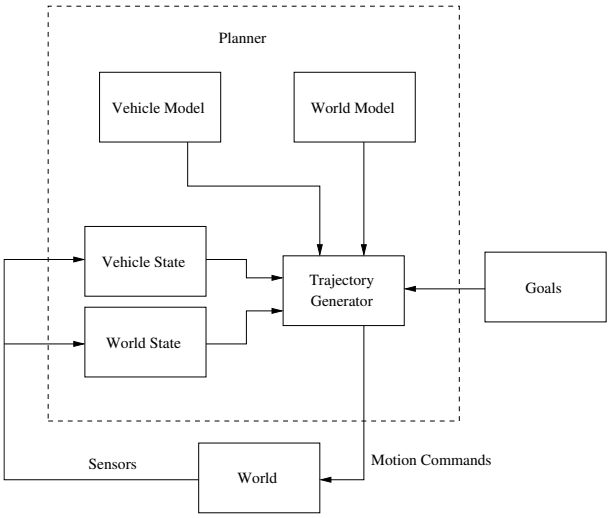


Fig. 1. Overview of the planning loop.

Relative to the existing literature on RRT-based motion planning, this paper focusses on kinodynamic problems using local information only. The lack of global information means that the planner cannot in general compute a plan all the way to a goal location, since the goal may be outside sensor range and the robot's view of areas of the configuration space may be occluded by obstacles. Instead, the planner's task is to optimise a cost function over a limited time-horizon which will safely bring it closer to the goal. We will show that by bounding this cost function, existing parts of the search tree can be pruned and unexplored parts of the state-space can be ignored. This can lead to substantial improvements in terms of performance and computation time.

Kinodynamic planning involves non-holonomic vehicles and differential constraints [3]. While several papers have shown how re-use of plans from previous iterations can dramatically improve performance [2][13][4], this strategy

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government.

The authors are with the Australian Centre for Field Robotics, University of Sydney, NSW 2006, Australia {a.brooks,t.kaupp,a.makarenko}@acfr.usyd.edu.au

is complicated in kinodynamic problems by the fact that it is non-trivial to compute the controls required to connect two arbitrary points in state-space. In particular, it is non-trivial to connect the current state (which will not exactly match any of the previous plan's predicted states, due to control errors), to any state from the previous plan.

Instead, we propose the computation of subtrees consisting of states similar to the states from the previous planning iteration, by remembering the controls which were used to generate those subtrees. A reasonable plan can usually be produced by predicting forwards from the current state according to the best control trajectory from the previous iteration. While the trajectory through state-space will not be identical, the plan will be good as long as the goal is relatively stable and sensors do not detect any drastic changes in obstacles. We will show that seeding the search tree with the best plan from the previous iteration can be of particular benefit in that it provides a tight initial bound on the cost function, allowing large areas of the state-space to be ignored.

The algorithm has been applied to both a team of Segway RMPs navigating in an urban environment, and the Sydney-Berkeley team's DARPA Grand Challenge vehicle. Collectively, the Segways have accumulated on the order of hundreds of vehicle-hours of autonomous driving using the algorithm described in this paper.

The remainder of this paper proceeds as follows. Section II discusses related work. Section III then formulates the problem and presents the trajectory generation algorithm. Section IV describes the experimental setup, and Section V presents the results. Section VI concludes.

II. Related Work

Motion-planning problems can be divided into *global* problems based on *a priori* information, and *local* problems based on *sensory* information [10]. A solution to a global problem is generally a complete collision-free path from a start configuration to a goal configuration, whereas local problems require the planner to continuously produce a control input under strict time constraints, with incomplete and possibly uncertain information.

For a review of global motion-planning techniques, see [9]. For kinodynamic problems, RRTs in particular have recently become enormously successful. While the pass/fail criteria for a path in a global problem is usually binary (based on whether or not it reaches the goal), there has also been work on incorporating continuous cost functions to coerce RRT algorithms into producing higher-quality paths [12][5]. Another common approach is to use an RRT to generate an admissable path in the absence of cost considerations, then subsequently optimise using a different algorithm to improve path quality [9].

A great deal of work in local mobile-robot obstacle avoidance has focussed on reactive methods, in which explicit models are not required (*e.g.* [11][10]). Instead, motion commands are generated directly from sensor data. While these methods have been successful (particularly in indoor scenarios), and are fast and easy to apply, they are more difficult to apply to problems with dynamic constraints. Furthermore, they suffer from a lack of lookahead, which can lead to highly suboptimal paths and problems with oscillation.

Model-predictive methods improve on reactive methods by allowing some degree of lookahead. A common approach is to search the control-space deterministically by projecting the vehicle forward along a fixed set of elemental paths such as lines, circles, and clothoids (*e.g.* [7][6]). Benefits of randomised sampling include reduced chances of becoming trapped in systematic "worst-case" scenarios [1], and the ability to focus computation on promising areas of the state-space, for example using the pruning strategy proposed in this paper.

III. Trajectory Generation

A. Problem Formulation

The problem can be posed as follows. Let $\mathbf{x}(t) \in \mathbf{X}$ and $\mathbf{o}(t) \in \mathbf{O}$ denote the state of the vehicle and of obstacles, respectively, at time t. $\mathbf{X}$ will generally consist of the vehicle's pose in configuration-space plus its velocity. The behaviour of obstacles is assumed known and independent of the vehicle's motion. At time t the controller applies a control input $\mathbf{u}(t) \in \mathbf{U}(\mathbf{x}(t))$. Planning occurs over a fixed horizon of h seconds.

Without loss of generality, we assume that planning always begins at $t = 0$. The dynamics of the vehicle are specified by a state transition equation $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, such that the vehicle's state can be predicted using

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^{\Delta t} f(\mathbf{x}(t), \mathbf{u}(t))dt \tag{1}$$

Let $\tilde{\mathbf{u}}_{t_1:t_2}$ denote a control trajectory for the time interval $[t_1, t_2]$. The planner must produce a complete control trajectory $\tilde{\mathbf{u}}_{0:h}$ for the time interval $t \in [0, h]$. Let $c(\mathbf{x}(0), \tilde{\mathbf{u}}_{0:h})$ denote a cost function which assigns a scalar cost to any control trajectory and initial vehicle state, given the behaviour of obstacles. The trajectory generator's task is to compute the control trajectory $\tilde{\mathbf{u}}^*_{0:h}$ such that $c(\mathbf{x}(0), \tilde{\mathbf{u}}^*_{0:h})$ is minimised.

B. Basic Trajectory Generation Algorithm

The core of the planning algorithm is similar to an RRT. We maintain a tree T in which each node q consists of the tuple $< \mathbf{x}_q, t_q, \mathbf{u}_q >$: a point in state-space, a time, and the control input required to reach that node (or the null control input $\mathbf{u}_0$ at the root node). The algorithm iteratively expands the tree by selecting an existing node q and a control input $\mathbf{u}$, then adding new nodes by integrating $\mathbf{u}$ forwards in time from q. This process is repeated N times, where N is a free parameter.

Let $c_T(q_f)$ denote a function which computes the cost of the control trajectory $\tilde{\mathbf{u}}_{0:h}$ from the root node to a final leaf node q_f at $t = h$. During tree expansion, we keep track of the lowest-cost final node q_{best}. The best control trajectory is the one required to traverse the tree from root to q_{best}. The entire procedure is shown in Algorithm 1.

Algorithm 1 Basic trajectory generation.

```
1   initialise: T ← q_root = < x(0), 0, u_0 >, c_min ← ∞
2   for i = 1 ... N
3       q ← SELECT_NODE(T)
4       u ← SELECT_CONTROL(T, q)
5       t ← t_q, x ← x_q
6       while t < h
7           Δt ← SELECT_DT(x, u, t)
8           x ← x + ∫_t^{t+Δt} f(x(t), u)dt
9           t ← t + Δt
10          if IS_INFEASIBLE(x) then break
11          add q_new =< x, t, u > to T
12          if t = h then
13              c_q ← c_T(q_new)
14              if c_q < c_min then
15                  c_min ← c_q, q_best ← q_new
16              end if
17          end if
18      end while
19  end for
```

Algorithm 1 requires a number of subroutines. The usual RRT implementations of **SELECT_NODE** and **SELECT_CONTROL** sample a point in state-space, select the nearest node, then select the control which grows the tree from the node towards the sampled point [9]. The situation is complicated in this case, however. Firstly, the state-space is unbounded, so it is unclear how to sample a state. Secondly, computation of the nearest neighbour requires a distance metric which considers both vehicle pose and velocity. Thirdly, it is non-trivial to compute a control which moves from one state to another. Our current implementations of **SELECT_NODE** and **SELECT_CONTROL** avoid these issues by simply sampling both from a uniform distribution. Our implementation of **SELECT_DT** adjusts the time-step dynamically, selecting a time-step which ensures that collisions will not be missed and that the time-horizon will not be exceeded. **IS_INFEASIBLE** checks for infinite-cost states (*e.g.* collisions with obstacles).

C. Trajectory Generation with Pruning

Expansion can occur in a more directed manner if a bound can be placed on the cost attainable from a partial control trajectory. Let $c^o_T(q)$ denote a function which returns a lower bound on the cost of any control trajectory which includes the node q. In other words, $c^o_T(q)$ is an optimistic estimate of the cost attainable from q for the entire time horizon.

If, during expansion, the optimistic cost for a node q is greater than the best cost found so far, c_{min}, then q cannot possibly contribute to a better control trajectory than the best one already found. Therefore, q should not be added to the tree. If it is already part of the tree, then q and all of its children can be pruned. Since $c^o_T(q)$ is strictly a lower bound, pruning in this way only eliminates trajectories which are provably worse, and therefore does not break any probabilistic completeness guarantees.

Rather than searching the entire tree for prunable nodes whenever c_{min} improves, we take a lazy approach to pruning. When a node q is selected as a candidate for expansion, it is either expanded or pruned depending on $c^o_T(q)$. Algorithm 2 specifies the procedure.

Algorithm 2 Trajectory generation with pruning. The pruning step (line 11) is highlighted.

```
1   initialise: T ← q_root = < x(0), 0, u_0 >, c_min ← ∞
2   for i = 1 ... N
3       q ← SELECT_NODE(T)
3       if c^o_T(q) > c_min then
3           PRUNE(T, q); continue
3       end if
4       u ← SELECT_CONTROL(T, q)
5       t ← t_q, x ← x_q
6       while t < h
7           Δt ← SELECT_DT(x, u, t)
8           x ← x + ∫_t^{t+Δt} f(x(t), u)dt
9           t ← t + Δt
10          if IS_INFEASIBLE(x) then break
11          if c^o_T(q) > c_min then break
12          add q_new =< x, t, u > to T
13          if t = h then
14              c_q ← c_T(q_new)
15              if c_q < c_min then
16                  c_min ← c_q, q_best ← q_new
17              end if
18          end if
19      end while
20  end for
```

D. Heuristics for Finding Good Trajectories

The pruning strategy described in the previous section allows potentially large areas of the state-space to be ignored if a provably-better trajectory has already been found. During the tree expansion process, the earlier a good trajectory can be found, the less time will be wasted exploring branches which will subsequently be pruned. This motivates strategies for finding reasonable trajectories quickly.

One strategy is to identify heuristics which are likely to produce reasonable trajectories, and to apply these heuristics to the root node prior to random expansion of the tree. An obvious heuristic is the plan from the previous iteration, assuming that the world remains relatively static. This plan can be re-generated in state-space by applying the control trajectory from the previous iteration, shifted back in time by the interval since the last plan was generated. Section V shows that using the previous control trajectory in this way is extremely effective for improving plan quality.

Another interesting alternative might be to use a simple and fast planner, such as Nearness Diagram [10], to generate an initial solution. This idea is left for future work.

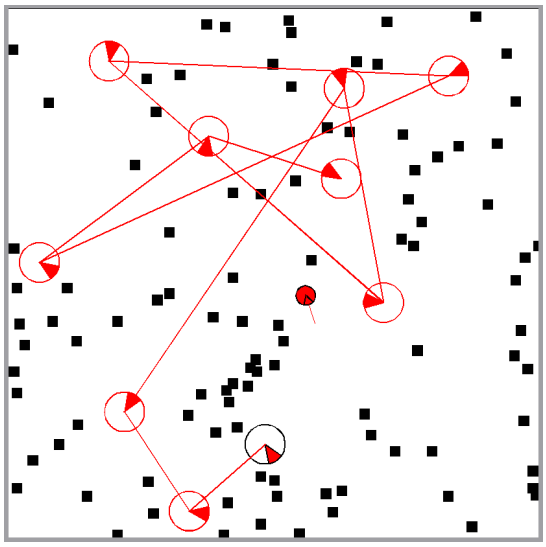


Fig. 2. An example simulated world used for algorithm evaluation. Black squares are 1m×1m obstacles, open red circles are waypoints (the filled sector indicates the required heading), and the filled red circle indicates the vehicle. The positions of obstacles and waypoints are random. The size of the world is 40m×40m.

IV. Experiments

Experiments were conducted using a simulated differential-drive robot, navigating in randomly generated two-dimensional 40m×40m worlds such as the one shown in Figure 2. The vehicle state-space $\mathbf{X}$ is 5-dimensional, consisting of the the pose of the robot (x,y,θ) plus its linear and rotational velocities (v,ω). The control-space $\mathbf{U}$ consists of linear and rotational accelerations, $\dot{v}$ and $\dot{\omega}$.

The robot can sense obstacles using a simulated 2D laser scanner, with 80m range and a 180° field of view. Its task is to navigate from the origin through a series of 10 randomly-generated waypoints scattered throughout the world. Algorithms are assessed based on the wall-clock time taken to negotiate each course, and the CPU time required to select control inputs.

A. Cost Function

The planner needs a cost function with which to compare potential trajectories. The following weighted sum was used:

$$\begin{aligned} c(\mathbf{x}(0), \tilde{\mathbf{u}}_{0:h}) = & w_o \max_t \left[c_o\left(\mathbf{x}(t)\right)\right] + \\ & w_a \min_t \left[c_a\left(\mathbf{x}(t)\right)\right] + \\ & w_r \min_t \left[c_r\left(\mathbf{x}(t)\right)\right] + \\ & w_m \min_t \left[c_m\left(\mathbf{x}(t)\right)\right], \quad t \in [0, h] \end{aligned}$$

where w_o, w_a, w_r, and w_m are weights which sum to one. The terms are as follows:

- c_o is a velocity-dependent term which penalises proximity to obstacles (the vehicle must allow more clearance when moving faster):

$$c_o(\mathbf{x}(t)) = \begin{cases} 1 - sgm\left[\frac{\psi - \psi_{min}}{\psi_{min}}\right] & \text{if } \psi > \psi_{min} \\ \infty & \text{if } \psi \leq \psi_{min} \end{cases}$$

where sgm denotes the sigmoid function, ψ denotes the clearance from the nearest obstacle, and ψ_{min} denotes the minimum clearance.

- c_a rewards approaching goals. The planner maintains a goal-horizon of G goals. The cost for the first goal is calculated using

$$c_{a,0}(\mathbf{x}(t)) = \begin{cases} g_l(\mathbf{x}(t))/g_l(\mathbf{x}(0)) & \text{if } g_l(\mathbf{x}(t)) > \tau_l \\ 0 & \text{if } g_l(\mathbf{x}(t)) \leq \tau_l \end{cases}$$

where $g_l(\mathbf{x}(t))$ is the linear distance from the vehicle to the centre of the goal, and τ_l is the linear tolerance used when assessing whether the goal has been reached (*i.e.* the radius of the goal). The cost for the n^{th} goal, $c_{a,n}(\mathbf{x}(t))$, is assessed similarly, but with $g_l(\mathbf{x}(0))$ replaced by the distance from the $(n-1)^{\text{th}}$ goal. c_a is a weighted sum over all G goals, with the cost for any goal set to 1 unless the trajectory has reached previous goals.

- c_r rewards reaching goals in both position and orientation. For the n^{th} goal:

$$c_{r,n}(\mathbf{x}(t)) = \begin{cases} 1 & \text{if } g(\mathbf{x}(t)) > \tau \\ 0 & \text{if } g(\mathbf{x}(t)) \leq \tau \end{cases}$$

where $g(\mathbf{x}(t))$ and τ denote a distance and tolerance, respectively, which account for both linear position and heading. Again, reaching subsequent goals only lowers the cost if previous goals have been reached.

- c_m rewards motion, helping the planner escape local minima:

$$c_m(\mathbf{x}(t)) = 1 - sgm\left[d(\mathbf{x}(t), \mathbf{x}0) - 0.1\right]$$

where d denotes cartesian distance.

The weights are free parameters which determine how aggressively or conservatively the vehicle behaves. For the experiments in this section, they were set to $w_o = 0.37$, $w_a = 0.36$, $w_r = 0.26$, and $w_m = 0.01$. A goal horizon of $G = 2$ was used. Compared to a heuristic algorithm such as VFH [11], the process of parameter selection is relatively simple because each paramater has a clear physical meaning with obvious consequences.

In order to predict the proximity of points along a given trajectory to obstacles, the planner maintains a local map of the environment, built up over a series of scans. This allows the use of information about obstacles behind the vehicle which are not presently visible. This local map is kept small by forgetting about obstacles which the vehicle cannot reach in the time horizon.

Since the cost function consists of $\min$ and $\max$ operators, it is simple to construct the function $c_T^o(q)$ which provides an optimistic cost estimate. For a node q at time $t_q < h$, c_o can only increase, and c_a, c_r, and c_m can be bounded using the maximum straight-line distance the vehicle could move in $h - t_q$ seconds in the absence of obstacles.

V. Results

Results were obtained in 50 randomly generated worlds, using a time horizon of 7 seconds. In each world, experiments were conducted with varying amounts of time spent

generating plans. This was controlled by varying N, the number of expansions made to the tree. All variations of the algorithm were able to navigate through all worlds without collisions.

Figure 3 shows the effects of seeding the tree with the trajectory from the previous planning iteration. This clearly leads to a significant reduction in path completion time. Figure 4 illustrates an example of pruning. Without pruning, the planner wastes considerable time exploring parts of the state-space which cannot lead to better trajectories than the one already computed. The effects are analysed in Figure 5. For a given number of expansions, pruning results in a non-trivial improvement in plan quality and an approximate halving of the time required for planning. If the planning time were held constant instead of the number of expansions, plan quality could be further improved.

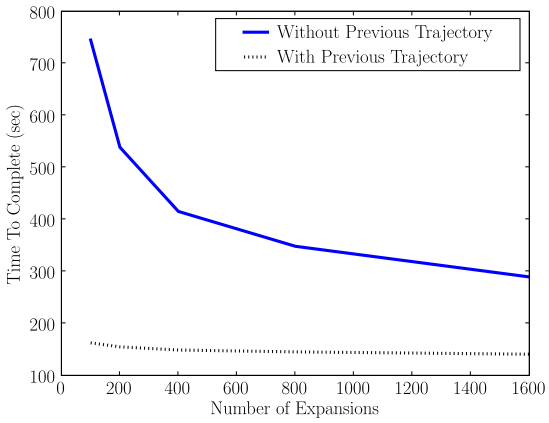
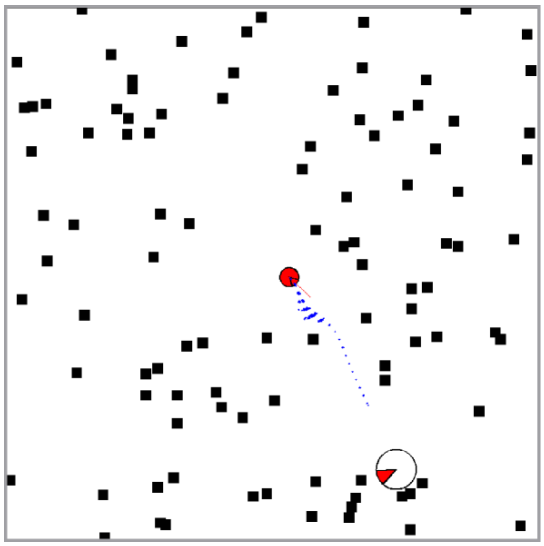


Fig. 3. Mean wall-clock time taken to complete each test course, with pruning enabled, both with and without seeding the planner with the trajectory from the previous planning iteration.

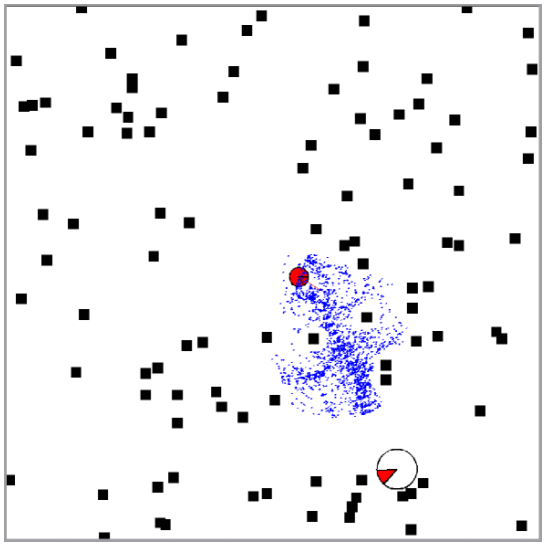
The algorithm has been implemented on the vehicles and in the environments shown in Figure 6. Firstly, the algorithm has been implemented on a team of Segway RMPs, using the cost function described in Section IV. The RMPs operate in an urban environment, requiring them to navigate at speeds up to 3.5m/s in large open spaces, in buildings, and through doorways. The cost function has been extended to consider the plans of nearby Segways, represented as dynamic obstacles. The Segways have driven autonomously for a combined total on the order of hundreds of hours using this algorithm. Secondly, the algorithm was used on the Sydney-Berkeley team's DARPA Grand Challenge vehicle, with a modified cost function which accounted for proximity to the centre-lines of road lanes.

VI. Conclusion

This paper presented an algorithm for real-time sensor-based motion-planning under kinodynamic constraints, in environments for which there is no *a priori* information. The approach explores the space of possible control trajectories by iteratively expanding a tree using randomised sampling. Performance can be dramatically improved by (1) seeding the planning tree using the best control trajectory from the previous iteration, and (2) pruning branches based on a bound to the cost function and the best cost found so far. The algorithm was implemented on two kinds of vehicles, and has been used for autonomous driving for a combined time on the order of hundreds of hours.



(a) Planning With Pruning



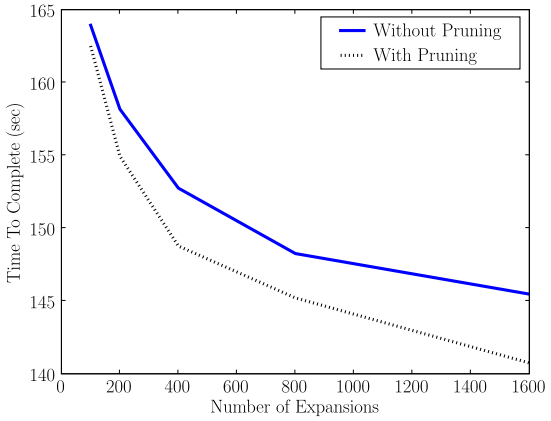
(b) Planning Without Pruning

Fig. 4. Trajectory generation with and without pruning. The trajectory tree is displayed using blue dots: each dot corresponds to a possible future state. Without pruning, considerable effort is wasted exploring areas of the state-space which cannot lead to better trajectories. The images were generated using N=1600 expansions. At the time of planning, the vehicle is moving forwards and turning to the right.

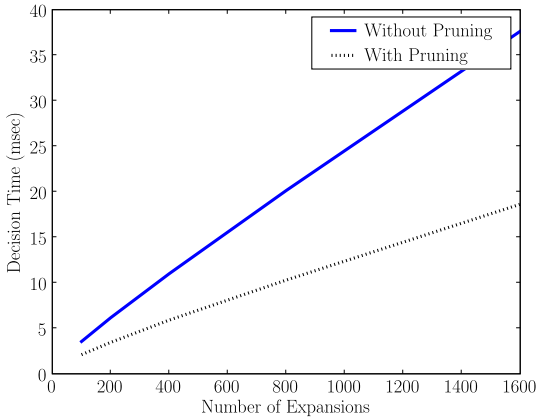
References

[1] M. Branicky, S. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 1481– 148, 2001.

[2] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2002.



(a) Mean wall-clock time to complete course



(b) Mean CPU time to generate plan

Fig. 5. The effects of pruning. For all tests, trees were seeded with the trajectory from the previous planning iteration.

(a) A Segway RMP (a second RMP is in the background).

(b) The environment in which the Segways have been operated. Eight vehicles are visible in the scene.

(c) The Sydney-Berkeley team's DARPA Grand Challenge vehicle.

Fig. 6. Vehicles and environments where the algorithm has been implemented and used.

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

[4] D. Ferguson, N. Kalra, and A. Stentz. Replanning with rrts. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 1243 – 1248, 2006.

[5] D. Ferguson and A. Stentz. Anytime RRTs. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 5369–5375, 2006.

[6] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *Intl. Journal of Robotics Research*, 25(5-6):449–483, 2006.

[7] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains. *Intl. Journal of Robotics Research*, 21(10-11):917–942, 2002.

[8] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[9] S. LaValle. *Planning Algortihms*. Cambridge University Press, 2006.

[10] J. Minguez and L. Montano. Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.

[11] I. Ulrich and J. Borenstein. VFH+: Reliable obstacle avoidance for fast mobile robots. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 1572–1577, 1998.

[12] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2003.

[13] M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for rapid replanning in dynamic environments. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, 2007.