2006

Optimal Sampling In the Space of Paths: Preliminary Results

Colin J. Green
Carnegie Mellon University

Alonzo Kelly
Carnegie Mellon University

Optimal Sampling In the Space of Paths: Preliminary Results

CMU-RI-TR-06-51

Colin J. Green and Alonzo Kelly

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
cjgreen@ri.cmu.edu, alonzo@ri.cmu.edu

Summary. While spatial sampling has received much attention in recent years, our understanding of sampling issues in the function space of trajectories remains limited. This paper presents a structured approach to the selection of a finite control set, derived from the infinite function space of possible controls, which is optimal in some useful sense. We show from first principles that the degree to which trajectories overlap spatially is directly related to the relative completeness that can be expected in sequential motion planning. We define relative completeness to mean the probability, taken over the population of all possible worlds, that at least one trajectory searched will not intersect an obstacle. Likewise, trajectories which are more separated from each other perform better in this regard than the alternatives. A suboptimal algorithm is presented which selects a control set from a dense sampling of the continuum of all possible paths. Results show that this algorithm produces control sets which perform significantly better than constant curvature arcs. The resulting control set has been deployed on an autonomous mobile robot operating in complex terrain in order to respond to situations when the robot is surrounded by a dense obstacle field.

1 Introduction

There is a large body of literature regarding the computation of optimal paths for holonomic robots, and numerous varieties of planners exist which find non-optimal, but executable trajectories, for complex nonholonomic robots. In the latter case, a set of vehicle commands (a control set) is often used to elaborate the search space. Examples include forward simulation planners[5, 11], grid based planners[2], and ego graph planners[7]. In each case the control set has a different meaning, and a different use, but in all cases the selection of the control set has a significant effect on the performance of the planner. Given that there are choices to be made in discrete search space design, it would be prudent to define metrics of performance and try to find designs which optimize them.

This paper concentrates on a completeness metric for finite resolution planning because we are interested in on-line planning for a high speed vehicle operating in complex cluttered environments. While it is useful to know that a planner is complete in the limit of infinite resolution, the more practical problem is that of maximizing the likelihood of solving a query at the resolution actually being used. Indeed, we can distinguish different planners, and different search spaces, based on how well they perform in this regard. While we will apply our techniques to a forward simulation planner, methods to define optimal discrete search spaces should have broad applicability.

In a forward simulation planner, a vehicle model is used to estimate the trajectory that the vehicle will follow in response to given control input. A set of these controls are simulated to give a corresponding set of trajectories which are then evaluated for hazards and propensity to achieve the goal. Finally, a command is selected based on these evaluations and sent to the vehicle. For the remainder of this paper *controls* will refer to the input to the differential equation representing the vehicle model and *trajectories* will refer to the output of this differential equation.

When using this type of planner, a very important consideration is what commands (or, equivalently, trajectories) to evaluate. Each trajectory takes a non-zero amount of time to evaluate, so real-time response requirements allow only a finite number to be searched. In addition, the selection of a trajectory set (given finite resources) has a significant effect on the completeness of the planner. Although such control sets are often used, to the authors knowledge, there has been no effort to evaluate finite trajectory sets in a structured manner with regard to relative completeness. Certainly works such as [10] and recent derivatives have searched the continuum to find the shortest path between any two points in an obstacle free environment. We are concerned here, however, with dense obstacle fields. We are also concerned with meta-level and probabilistic optimality of the search space itself rather than of any searches of it.

In Sect. 2 the work is motivated by a discussion of why constant curvature arcs can be a poor search space for obstacle avoidance planning at high speed. In Sect. 3 the relative completeness of a set of trajectories is defined as the probability that at least one trajectory from the set will not intersect an obstacle in a random field of obstacles. Such a trajectory is deemed to have *survived* the test. This definition is then used in Sect. 3.1 to prove that overlap between trajectories will reduce this probability of survival. Section 3.2 extends this idea of overlap to an environment where there is correlation between obstacles. It suggests that trajectories which are close to each other, even if they do not overlap, will lower the probability that at least one of the trajectories will not intersect an obsticle. Section 4 describes a set of algorithms which generate trajectory sets with the intent to maximize the probability of survival. Finally Sect. 5 presents the results from numerous simulations as well as trials on a outdoor mobile robot, all of which indicate the benefits of these trajectory sets over simple arcs.

2 Motivation

According to Taylor's remainder theorem an arbitrarily smooth curve (for example, a path curvature signal) can always be well approximated by a series of short segments of simple curves like arcs. This seems to suggest that searching a small set of constant curvature arcs often enough (and only executing a very small portion during each cycle) can produce an arbitrary complex solution trajectory. In fact this is far from correct. As illustrated in Fig. 1, finite stopping distance combined with the need to guarantee vehicle safety leads the planner to reject the right turn because it is not safe for a sufficient length. This occurs despite the fact that subsequent planning cycles will have the opportunity to adjust the plan to a more correct path before the obstacle is reached. We observe this behavior on a regular basis in our experimentation. Of course, a planner will not generate a solution if it does not search it and the solution here is not an arc.

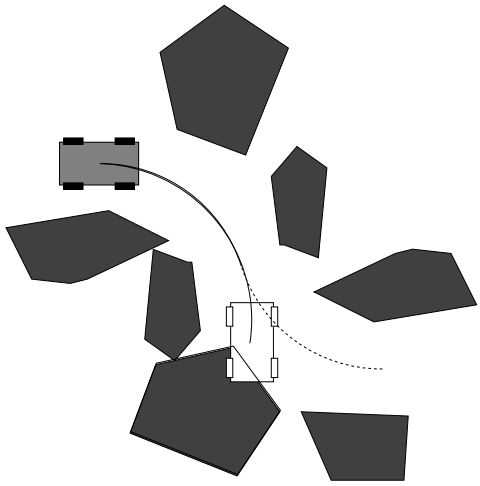


Fig. 1. An example of a situation in which constant curvature controls cannot produce a valid path. A hard right turn must be commanded to to follow the corridor, but it cannot be commanded as it intersects an obstacle within the vehicle's stopping distance.

The issue here is the completeness of the planner. While a forward simulation planner with a finite number of controls will never be entirely complete, some control sets are clearly better than others. A search space composed of only left turns is intuitively a bad choice, as is a space of many 10 meter long trajectories which deviate only slightly from each other. Having shown that arcs are apparently not a *good* control set, we are compelled to ask: what is? Arcs cannot generate the S-curves necessary to navigate in certain situations,

but at the same time S-curves cannot generate the arcs that are also sometimes necessary. From the above perspective, the problem of selecting the best set of controls is one of selecting a finite sample of paths from the function space of possible paths with the goal of maximizing relative completeness. While spatial sampling has received much attention in recent years[3, 8], this type of sampling in the space of paths remains largely unexplored.

3 Theory

Before we can begin to describe a better set of trajectories, we must first have a definition of what makes one trajectory set better than another. A very important characteristic of a set of trajectories, as suggested in Sect. 2, is how complete it renders the local planner. This level of "relative completeness" can be defined as the percentage of the time that the planner will find a path to achieve a random goal in a random configuration of obstacles in bounded time if a path exists. For a fully complete planner this will, of course, be 100% of the time. This is a different notion than resolution completeness[4] or probabilistic completeness[1], neither of which consider the probability of success within a bounded running time. Making the most of the fixed amount of time available is an important consideration on a mobile robot with hard real-time requirements.

For the purposes of this paper, we will consider the relative completeness of a set of trajectories to be the probability that, when they are placed at a random location in a random obstacle field, at least one trajectory does not intersect any obstacles (i.e. is safe). Since the goal can be uniformly anywhere, we can ignore its location in the analysis. Of course, the sequence of planning problems that a real robot encounters are correlated in time as the robot actively tries to avoid the obstacles. Nonetheless, we ignore this matter for the moment and later suggest that even this simplified statement of optimality leads to improved planner performance.

3.1 Computing probability

In an effort to describe a method of computing relative completeness, we must first look at the ways in which an obstacle can intersect a trajectory. In particular it is important to pay attention to the ways in which a single obstacle can intersect multiple trajectories. As shown in Fig. 2(a), two overlapping trajectories generate a partitioning of the world into three different areas (four if space which is not part of any trajectory is included). If an obstacle were to occupy any part of element 01, then trajectory 1 would no longer be safe. Similarly, an obstacle in element 10 implies trajectory 2 is not safe. But, more interestingly, an obstacle in element 11 means *both* trajectories 1 and 2 are not safe. The area of the trajectories, as shown in Fig. 2, represents the convolution of the vehicle body over the trajectory. A obstacle in this area indicates

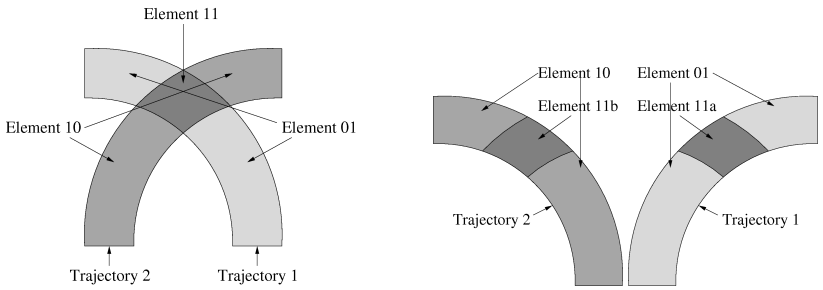


Fig. 2. (a) Shows the overlap of the vehicle convolution for two trajectories and the different areas a partitioning of this space produces. (b) Shows the same two trajectories except in a situation where they no longer overlap.

that the vehicle would intersect that obstacle while traversing the trajectory. There are many methods of finding this area, such as the one suggested by [6].

For two trajectories the partitioning of the space is very simple, but it becomes more complex as more trajectories are involved. Figure 2(a) illustrates a naming convention for elements of the partition that will be used in this paper. Binary numbers are used to represent each subset in the spatial partition. Each bit in these numbers indicates if the trajectory corresponding to that bit number is a member of that subset. For example, 01 indicates that an obstacle in this area will intersect trajectory 1 but not 2. From this numbering scheme it is clear that there are as many as 2^n elements in the partition of the area occupied by all trajectories, where n is the number of trajectories. In most cases there will in fact be fewer than 2^n elements because not all trajectories will overlap in every possible combination.

Now, let the boolean valued function $C(R)$ mean that some portion of region R contains at least one obstacle. Lets assume a function $P(\overline{C(R)})$ which depends on the area of the region (A_R) and returns the probability that R contains no obstacles. In addition, we require that $P(\overline{C(R1)})P(\overline{C(R2)}) = P(\overline{C(R1 \cup R2)})$ which simply says that probabilities depend only on the area of a region, and not on its shape or position. It can be shown that this condition implies a distribution for $P(\overline{C(R)})$ which is exponential in area.

We would like to show that there is a distinct difference between the two pairs of trajectories (in terms of probability that at least one does not intersect any obstacles) shown in Figs. 2(a) and 2(b). The size and shape of the trajectories in both cases are identical. The only difference is the overlapping area (11 in the overlapping case, 11a and 11b in the separated case). At an intuitive level, it is clear that overlap is not optimal because it creates an area

where a single obstacle can hit both trajectories at the same time. A more precise elaboration of the difference between these two cases follows.

There are a total of three cases where at least one trajectory from a set of two will survive in a random configuration of obstacles: trajectory 1 is hit but 2 is not, trajectory 2 is hit but 1 is not, and neither trajectory is hit. For the overlapping case (a), the total probability is

$$P(C(01))P(\overline{C(10)})P(\overline{C(11)}) + P(\overline{C(01)})P(C(10))P(\overline{C(11)})$$
$$+P(\overline{C(01)})P(\overline{C(10)})P(\overline{C(11)})$$

Now, using the multiplicative characteristics defined for the probability function, along with the fact that $P(\overline{C(10)}) = P(\overline{C(01)})$, because both have equal area, and $P(C(01)) = 1 - P(\overline{C(01)})$ we have

$$2(1 - P(\overline{C(01)}))P(\overline{C(10 \cup 11)}) + P(\overline{C(01 \cup 10 \cup 11)}) =$$
$$2P(\overline{C(10 \cup 11)}) - P(\overline{C(01 \cup 10 \cup 11)})$$

The probability for the non-overlapping case (b) can be developed in a similar fashion. For simplicity we will treat the trajectories as a single partions (combining 01 and 11a, as well as 10 and 11b). Also note that $P(\overline{C(11)}) = P(\overline{C(11\text{a})}) = P(\overline{C(11\text{b})})$ because all have equal area.

$$P(C(01 \cup 11\text{a}))P(\overline{C(10 \cup 11\text{b})}) + P(\overline{C(01 \cup 11\text{a})})P(C(10 \cup 11\text{b}))$$
$$+P(\overline{C(01 \cup 11\text{a})})P(\overline{C(10 \cup 11\text{b})}) =$$
$$2(1 - P(\overline{C(01 \cup 11)}))P(\overline{C(10 \cup 11)}) + P(\overline{C(01 \cup 11\text{a} \cup 10 \cup 11\text{b})}) =$$
$$2P(\overline{C(10 \cup 11)}) - P(\overline{C(01 \cup 11 \cup 10 \cup 11)})$$

Notice that the difference between the probabilities for the overlapping and non-overlapping case is that the overlapping area (11) is counted twice in the non-overlapping case. There is a lower relative probability that there are no obstacles in the union of those four areas, than in the three in the corresponding case above, because the probability is assumed to be monotone in area. The lower probability is subtracted in the computation of the total probability that one trajectory survives, giving a higher total probability of at least one one survivor in the case where there is no overlap.

At a higher level, when any two trajectories overlap they have a higher probability of not intersecting any obstacles at all (higher by some amount, β which is a function of the area of overlap) because the union of their areas is reduced. On the other hand, they have a lower probability of having only one survivor. If either trajectory does intersect an obstacle the overlap leads to a lower probability that the other will be safe (lower by, again, β for each of the two ways this could occur). This means that the overlapping trajectories will have a β lower probability of having at least one safe trajectory.

Unfortunately, the computation of probabilities in this manner quickly become computationally impossible. Given that there are $O(2^n)$ subsets in the partition, and each one of them could intersect with one or more obstacles, there are $O(2^{2^n})$ possible disjoint events to evaluate. It quickly becomes impossible to even enumerate all the disjoint events which leave at least one safe trajectory. Instead, approximations (described in Sect. 4.2) or numerical methods (described in Sect. 4.3) must be used.

3.2 Computing probability with correlation

Up to this point our analysis has been lacking in one important aspect: obstacles in the real world are not distributed in a completely random way, but generally have a great deal of correlation. One way to view this correlation is that a location in space has a higher probability of containing an obstacle if any location nearby contains an obstacle. The increase in probability will depend on the distance between the two locations and typically has characteristics of an exponential distribution, high when the distance is small and near zero when the distance is large. In terms of trajectories, this means that proximity becomes akin to overlap. Any pair of trajectories is more likely to overlap an obstacle region, and therefore to both be eliminated in search, if they are merely close to each other and do not overlap. If we consider obstacles to be regions of defined size, moving two trajectories closer together exposes the pair to all of the sizes of obstacles which could overlap both before in addition to the smaller sized obstacles which can now span the reduced space between them. Hence, exposure of the pair must be monotone increasing with proximity. Conversely, the more separated any given pair is, the more likely it is that at least one will survive. This concept of correlation could be used to extend the theory in Sect. 3.1 to include probabilistic partitions, which indicate areas where the presence of an obstacle has some probability of intersecting one or more trajectories.

4 Algorithms

4.1 Reachability tree

For each of the algorithms presented here, a pool of potential trajectories is needed. These potential trajectories are created by generating a reachability tree, where the edges at each node are different curvature commands. Each path through this tree is a list of commands which can be simulated using an arbitrary vehicle model. In this case a model which limits linear and angular velocities and accelerations has been used. This arbitrary vehicle model, combined with relevant initial conditions, such as a non-zero initial curvature, effects the mapping from controls to trajectories. By varying the initial conditions, a pool of potential control sets can be created, from which one can be

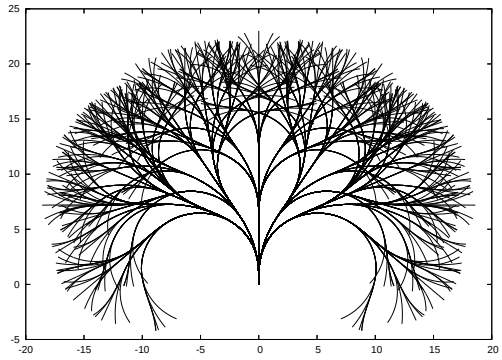


Fig. 3. Shows an example reachability tree with outdegree of 5 and depth of 4, corresponding to 625 trajectories. Simulated for a vehicle with constant linear velocity and limits on curvature and Δcurvature.

selected from at run time based on the current state, which is useful on a mobile robot. An example reachability tree is shown in Fig. 3. We typically use reachability trees with an outdegree of 5-9 and depth of 5-8. There is also an argument that one should use a larger outdegree at the root of the tree, where a small curvature change will have a more significant effect on the shape of the trajectory, and decrease the outdegree towards the leaves. For the purposes of this paper we have maintained a uniform outdegree throughout the tree. The goal of this tree is to quickly create a large sampling of potential trajectories from the infinite number of possible trajectories. Out of this large sample, the algorithms in Sects. 4.2 and 4.3 will select a smaller control set.

4.2 Separation based greedy algorithm

While computing the probability of at least one obstacle being in a given area is assumed to be simple, the computation of the probability that at least one trajectory in a large set will survive (as described in Sect. 3.1) is computationally infeasible because the number of disjoint events that must be enumerated grows doubly exponentially with the number of trajectories. Moreover, adding more realistic obstacle position correlation only worsens matters.

We instead focus on a method which capitalizes on a key point made in Sect. 3.1 and expanded on in Sect. 3.2 : more area between trajectories leads to a planner with higher relative completeness. The area between trajectories is easy to compute and it is related directly to the probability that one trajectory from a set will survive, which makes it an excellent metric to use.

The simplest algorithm to find the best set of trajectories would be to compute the probability of one survivor for all possible trajectory sets of a given size, and select the one with the highest probability of success. This would require checking $\binom{n}{k}$ sets (where n is the number of trajectories in the

reachability tree and k is the desired number of trajectories in the set). For a minimalistic choice of $n = 5^5$ and $k = 25$ this would require checking 1.4e62 combinations, which is not feasible.

Instead a greedy algorithm is used, as shown in Algorithm 1. The set of selected trajectories is seeded with the straight (zero curvature) trajectory, to preserve the capacity to plan a straight path. Next, trajectories are added one by one such that each trajectory added has maximal area between it and its closest neighbor in the set selected so far. It is interesting to note that given a large enough pool of potential trajectories, and once the bounds of the search space have been generated, the trajectory added at any step will have equal area between it and its first and second closest neighbors. Otherwise a different trajectory would have been selected which trades some of the area between it and its second closest neighbor for more area between its first closest, increasing the smallest area. While it is somewhat feasible to compute a separation score in relation to the entire existing set, the non-linear behavior of a typical correlation function makes this less necessary. As the area increases it makes less of a difference, so maximizing pairwise separation is an indication of merit for the entire set.

In maximizing each trajectories separation from its closest neighbor, this algorithm is attempting to minimize the *dispersion*[9] of the selected set of trajectories in the space of trajectories. Dispersion of P in X is given by

$$d(P; X) = \sup_{x \in X} \min_{p \in P} d(x, p)$$

In fact, Algorithm 1 is greedily minimizing the dispersion for the metric space (X, d) where $d(x, y) =$AreaBetween(x, y), X is the space of trajectories in the reachability tree, and P is the selected set of trajectories.

An additional benefit of this algorithm is that it returns a sequence of trajectories in sorted order. This means that given a set of n well separated trajectories, the set of $k, \forall k \leq n$ well separated trajectories is just the first k trajectories from the set of all n. This property is very similar to the property the van der Corput sequence[9] which provides an infinite, dense, sequence of numbers in the interval $[0, 1]$. This is very useful in the context of varying computing time budgets resulting from varying vehicle speeds. Example results from this algorithms application are shown in Fig. 4.

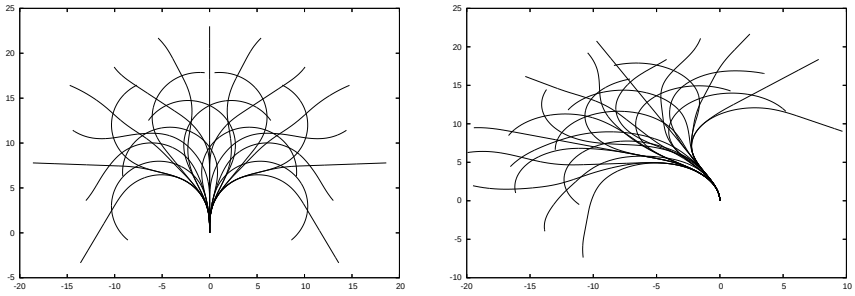
4.3 Monte-Carlo based greedy algorithm

Another method is to compute the probability of survival from a Monte-Carlo simulation. Monte-Carlo algorithms have often been used to overcome combinatorial problems, and have of course been used in non-holonomic planners[1, 9]. In this case, a number of circular obstacles are uniformly placed in a simulated world and given a random size (drawn from a Normal distribution). A set of trajectories is then placed in this world and intersections between trajectories and obstacles are observed. If any trajectories survive,

```
   input  : potentialTrajectories,n
   output: selectedTrajectories

   %Seed the algorithm by adding the zero curvature trajectory
   %to the list of selected trajectories
 1 selectedTrajectories.add(zeroCurvatureTrajectory);
 2 potentialTrajectories.remove(zeroCurvatureTrajectory);
 3 while selectedTrajectories.size() < n do
 4     maxArea ← 0;
       %Evaluate each of the remaining potential trajectories
 5     foreach element x of potentialTrajectories do
 6         minArea ← ∞;
           %Compare the current potential trajectory to each
           %selected trajectories, looking for the minimum area
 7         foreach element y of selectedTrajectories do
 8             if AreaBetween (x,y) < minArea then
 9                 minArea ← AreaBetween(x, y);
           %Compare the current potential trajectory to the best
           %potential trajectory so far
10         if minArea > maxArea then
11             maxArea ← minArea;
12             NextBestTrajectory ← x;
       %Add the NextBestTrajectory to the list of selected
       %trajectories
13     selectedTrajectories.add(NextBestTrajectory);
14     potentialTrajectories.remove(NextBestTrajectory);
```

Algorithm 1: Separation Based Algorithm



(a) (b)

Fig. 4. (a) A set of 25 well separated trajectories. (b) A set of 25 well separated trajectories with a high initial curvature.

success is declared. This process is then repeated for between a thousand and a few million times to compute the probability of survival. For this process, it has proven to be advantageous to create a small region in front of the vehicle which is guaranteed to contain no obstacles. Obstacles in this area have a high probability of taking out all trajectories regardless of which set is being tested, and this case represents the case where the planner has already failed, so it is not interesting from the present perspective. A more complete simulation would take into account the previous decisions made by the planner to steer the vehicle away from oncoming obstacles.

Even though this algorithm can compute a separation score in a fairly fast manner, a greedy algorithm is still required to search the space of all possible combinations of trajectories. The algorithm, which is very similar to the algorithm in Algorithm 1, is to start with the zero curvature trajectory (using the same reasoning as before). Then, evaluate all possible trajectories, which are not a member of the selected set, one at a time by adding it to the selected set, and computing the new sets expected chances of survival. Finally, keep the trajectory that gives the highest probability of survival.

Although this algorithm has its advantages (it is directly computing the probability for the entire set) it is much slower than the algorithm described in Sect. 4.2. In cases where the separation based algorithm takes only a few minutes, this algorithm takes days. Even in this time frame, it is unclear if enough Monte-Carlo steps have been taken to achieve convergence.

5 Results

A series of tests have been performed in simulation. The first set of tests were performed using the Monte-Carlo evaluation method described in Sect. 4.3 to compare different trajectory sets. The results of those tests are shown in Fig. 5. This figure shows that the separation based algorithm produced trajectories that are almost always better than the equivalent set of arcs, over a large range of n (n is the number of trajectories in the set). In addition, the graph shows that the separation based algorithm performed slightly better than the Monte-Carlo simulation algorithm for $n > 25$, this is likely because the number of Monte-Carlo steps had to be reduced to keep the run time under a few days. Note that the separation based set does not perform much better than arcs for a small control set, but the performance benefits increase as the size of the control set increases (correspondingly, as computers get faster, or vehicle speeds are reduce).

These trajectories were also tested in a more complex simulator. In each planning cycle, the vehicle follows the trajectory which was sensed as obstacle free for the longest length for some distance (which corresponds to how far the vehicle would travel during that planning cycle). In the case of a tie, the trajectory which terminates with the largest x coordinate is selected. This has the effect of guiding the vehicle towards $x = \infty$. The world is random and

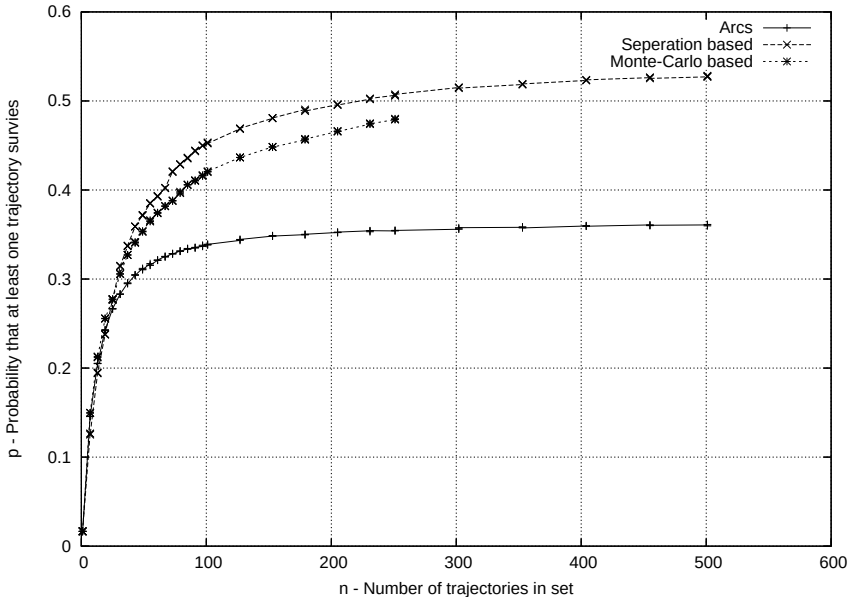


Fig. 5. The results from Monte-Carlo simulations for different trajectory sets.

infinite, so new areas of the world are populated with obstacles as the vehicle drives through them. The obsticles are circular and have a randomly generated size, giving the world the feel of a forest. To help prevent the vehicle from entering infinite cycles, areas where the vehicle has driven over are considered obstacle. In this simulator the effects of the speed of the vehicle, size of the lookahead, and cycle time of the planner can all be tested to compare arcs and well separated trajectories. Finally, this simulator has both a deliberative and reactive region in its trajectories. The reactive region must be obstacle free in order to ensure vehicle safety, while the deliberative region is used to help guide to vehicle away from situations which could compromise the vehicle safety. When running a test, the vehicle cannot slow down or stop. Instead, if the vehicle is in a position where it cannot avoid an obstacle, it runs into the obstacle and that obstacle hit is recorded. Results are computed based on how many times, on average, the vehicle actually hit an obstacle and how many time, on average, the safety of the vehicle could not be ensured because the safest trajectory collided with an obstacle in the reactive region. The results from this investigation are shown in Fig. 6. Overall there is more success in avoiding obstacles than guaranteeing the vehicle safety. This is expected because when vehicle safty is ignored, short segments of arcs can easily be joined to form very complex trajectories, as discussed in Sect. 2.

Using this simulator, a problem with well separated trajectories was noticed. With constant curvature arcs, at each timestep the path which the vehicle was following on the previous timestep is always in the new control

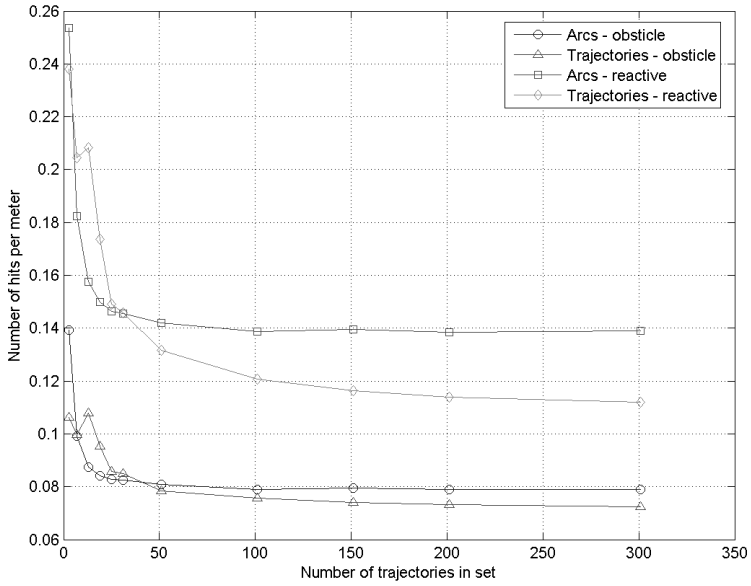


Fig. 6. The results from a dynamic vehicle simulation in an infinite world. Results are shown for the number of obstacles hit per meter and the number of times the safety of the vehicle could not be guaranteed per meter (reactive hits)

set. This allows the vehicle to follow a single trajectory for many cycles. With the well separated trajectories, at each timestep the remaining portion of the trajectory followed on the last cycle might no longer be in the set. This can cause a failure when the vehicle is attempting to navigate around complex obstacles. The simple solution applied to this problem is to always keep the trajectory the vehicle followed last in the current set. The portion of the trajectory before the current location is removed and the end of the trajectory is extended by simulating the vehicle with the last command in that trajectory. Running the same tests from Fig. 6 gives much better results, as shown in Fig. 7. These results follow those shown in Fig. 5 in indicating that well separated trajectories are uniformly better than simple arcs. This supports that the main conclusion is robust to the position correlation of a dynamic vehicle.

Finally, the control sets produced by the separation based algorithm have been tested on a highly mobile outdoor robot, shown in Fig. 8. On this vehicle, our well-separated trajectories are used in a special mode designed to allow the planner to find a path through very complex environments where a simple arc control sets fail. Although it is not possible to quantify the improvements these control sets provide, it is the perception of the field team that they provide a benefit. The difficulty in obtaining a statistically significant amount

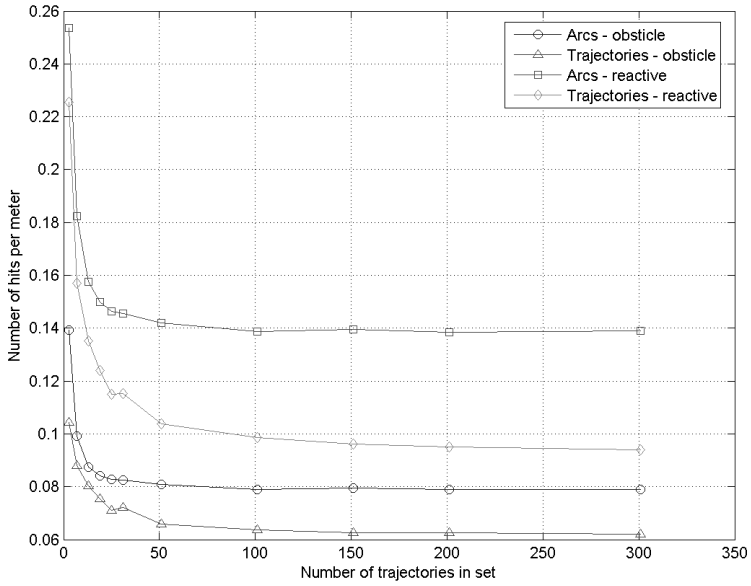


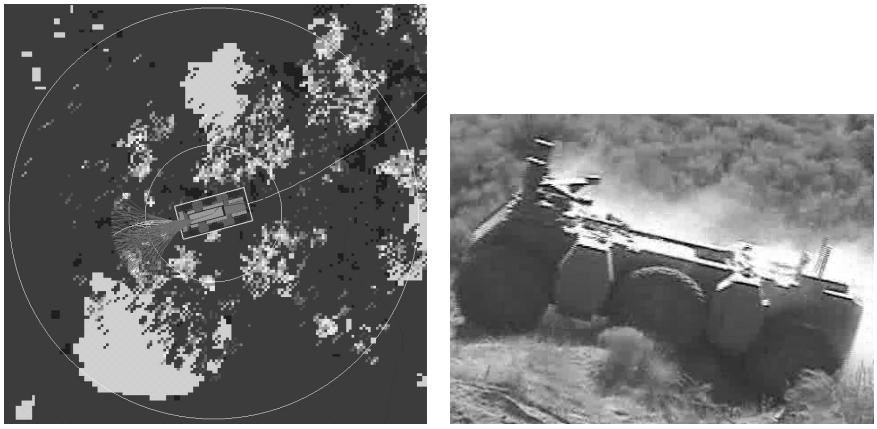
Fig. 7. The results from a dynamic vehicle simulation in an infinite world which always includes, in the current trajectory set, the last trajectory which was executed. Results are shown for the number of obstacles hit per meter and the number of times the safety of the vehicle could not be guaranteed per meter (reactive hits)

of time on the vehicle over a large number of environments using different control sets led us to pursue a simulation approach.

6 Conclusions and Future Work

In this paper we addressed but one of many systems engineering questions that arise in the context of motion planning. Rather than appeal to absolute completeness based on infinite resolution or probabilistic arguments, we have sought instead to begin to answer a fundamental question in real-time motion planning: what search space is most likely to solve a planning query in a given time in any environment.

We have presented a principled definition of relative completeness at finite resolution in terms of the likelihood that at least one safe path will be generated in any world. We have shown that trajectory overlap leads to reduced relative completeness and argued that trajectory separation is equally relevant. We have shown that, in so far as we have formulated the problem, it is intractable to solve exactly in realistic cases. On the other hand, we have also shown that a set of well separated trajectories, even a suboptimal set



(a) (b)

Fig. 8. (a) A screenshot of the planner navigating in cluttered terrain using a well separated trajectory set. (b) The vehicle on which well separated trajectories have been tested.

generated by greedy search, outperforms both a set of arcs and a set based on a more principled Monte Carlo search limited to a few days of computation. We have also shown that significant improvements in obstacle avoidance can be achieved by always ensuring that the vehicle can continue to follow the path which is was previously attempting to follow.

Other fundamental questions of interest include the relative optimality of a finite search space (e.g. the length of the optimal path relative to the continuum solution) and its relative efficiency (e.g. how does runtime trade against optimality). Throughout the field of robotics, investigation into the systems engineering questions of optimal design are still in their relative infancy. Our work is motivated by the quest to achieve the best possible result within the constraints of available computing resources. We felt there was no clear theoretical basis for the choice of one control set over another and therefore sought to construct one. Although we have not answered it exactly, it is clear that the question of which finite search space is best is meaningful, relevant and interesting. We have made one step here toward its solution, and we hope to continue this line of research for other figures of merit and other classes of planning problem.

7 Acknowledgments

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract "Unmanned Ground Combat Vehicle - PreceptOR Integration" (contract number MDA972-01-9-0005). The views and conclusions contained in this document are those of the authors and should not be

interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

1. J. Barraquand and J-C. Latombe. Robot motion planning: a distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, 1991.
2. J. Barraquand and J-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.
3. M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2001*, pages 1481–1487, 2001.
4. P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2002*, pages 1481–1487, 2002.
5. M. Daily, J. Harris, D. Keirsey, K. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong. Autonomous cross-country navigation with the ALV. In *Proceedings of the IEEE International Conference on Robotics and Automation, 1988*, volume 2, pages 718–726, 1988.
6. A. Foisy and V. Hayward. A safe swept volume method for collision detection. In *Proceedings of the Sixth International Symposium of Robotics Research, 1994*, pages 61–68, 1994.
7. A. Lacaze, K. Murphy, N. Declaris, and Y. Moscovitz. Path planning for autonomous vehicles driving over rough terrain. In *Proceedings of the IEEE International Symposium on Intelligent Control, 1998*, pages 50–55, 1998.
8. J. Matoušek. *Geometric discrepancy: an illustrated guide*. Springer-Verlag, Berlin, 1999.
9. H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.
10. J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2), 1990.
11. R. Simmons, E. Krotkov, L. Chrisman, F. Cozman, R. Goodwin, M. Hebert, L. Katragadda, S. Koenig, G. Krishnaswamy, Y. Shinoda, W. L. Whittaker, and P. Klarer. Experience with rover navigation for lunar-like terrains. In *Proceedings of the 1995 Conference on Intelligent Robots and Systems*, pages 441 – 446. IEEE, 1995.