School of Engineering Technical Report TR-2007-001
University of California, Merced - February 2007

A Simple Experiment on the Effect of Biasing Sampling Distributions for Planning Reaching Motions with RRTs

Marcelo Kallmann

mkallmann@ucmerced.edu
Computer Graphics Lab
University of California, Merced
5200 N. Lake Road - Merced CA 95343

Abstract

This paper reports experiments performed with a simple sampling cache mechanism used to bias the configuration sampling of a RRT-based motion planner. The Planner is used for synthesizing reaching motions for a humanlike character performing object relocation tasks in a virtual environment. Experiments are reported showing that the method can significantly reduce the computational time required for planning arm motions.

1 Introduction

Human-like characters able to manipulate objects in interactive virtual environments have direct applications in several domains, such as: Design, Ergonomics, Virtual Reality, Human-Computer Interfaces, Training and Computer Games. Most of the current techniques for animating characters in these applications are still based on the use of pre-defined (captured or designed by hand) motions. Besides being tedious and time-consuming to be created, pre-defined motions cannot be easily reused for manipulating objects that can be arbitrarily located among obstacles in a scene.

Alternatively, motions can be synthesized on-line with the use of motion planners [Lat90]. The difficulty of applying motion planners to interactive characters in virtual environments is mainly due the high, sometimes unpredictable, computational time required for finding valid motions. In addition, it is difficult to achieve realistic humanlike results.

However, when the goal is to achieve characters able to perform given tasks autonomously, motion planners still represent the best approach for computing collision-free reaching motions on-line. In a previous work [DKM04] an on-line bidirectional randomized planner is shown to be generally faster and finding more solutions when compared to using Jacobian-based Inverse Kinematics integrated with collision avoidance.

This paper presents a simple sampling cache mechanism for improving the performance of an arm planner based on Rapidly Exploring Random Trees (RRTs) [LaV98]. The mechanism is presented as an optimization over the RRT sampling strategy: instead of only using random samples, successfull samples from previous plans are reused. This approach is inspired by a similar mechanism previously proposed in the context of mobile

robotics [BV02]. Other learning strategies have also been proposed to motion planners [BB05b] [BB05a], however they do not focus on exploring learning strategies based on succesfull previous experiences, which is the main approach presented here. The experiments reported here demonstrate that the presented method can significantly improve the performance of RRTs.

2 The Cache Optimization Method

The bidirectional planner described in [Kal05] is used as starting point. This arm planner solves reaching tasks in two phases. (1) First, an analytical Inverse Kinematics (IK) [TB96] formulation including collision avoidance is used for computing a goal arm configuration reaching a desired location for the hand. (2) Then, a bidirectional RRT is applied for producing a collision-free arm motion to the goal arm configuration. After a motion is found, a smoothing phase can be optionally employed to improve the quality of the produced motion.

Note that the IK phase is only used to generate goal configurations for the motion planner, and therefore this paper focuses on measuring the effect of the proposed cache optimization on the motion planning phase only. The planning phase is described below.

2.1 Algorithm

Algorithm 1 summarizes the used implementation of the bidirectional RRT. Let c_{init} be the current arm configuration of the character and let c_{goal} be the target configuration given by the IK. Two search trees T_{init} and T_{goal} are initialized having c_{init} and c_{goal} respectively as root nodes, and are sent to the planner. The trees are iteratively expanded by adding valid landmarks. When a valid connection between the two trees can be concluded, a successful motion is found. Otherwise when a given amount of time has passed, the algorithm fails.

Algorithm 1 RRT PLANNER (T_1, T_2)

```
1: while elapsed time ≤ maximum allowed time do
2:    c_sample ← SAMPLECONFIGURATION().
3:    c_1 ← closest node to c_sample in T_1.
4:    c_2 ← closest node to c_sample in T_2.
5:    if INTERPOLATIONVALID (c_1, c_2) then
6:       return MAKEPATH (root(T_1), c_1, c_2, root(T_2)).
7:    end if
8:    c_exp ← NODEEXPANSION (c_1, c_sample, ε).
9:    if c_exp ≠ null and INTERPOLATIONVALID (c_exp, c_2) then
10:      return MAKEPATH (root(T_1), c_exp, c_2, root(T_2)).
11:   end if
12:   Swap T_1 and T_2.
13: end while
14: return failure.
```

The sampling cache optimization is implemented in the sampling routine of Line 2 and is explained in section 2.2. Lines 3 and 4 require searching for the closest configurations in each tree. A linear search suffices as the trees are not expected to grow much. The

metric used is the maximum distance between the positions (in global coordinates) of corresponding joints in each configuration.

Lines 5 and 9 check if the interpolation between two configurations is valid. It is considered valid if the interpolation produces collision-free intermediate postures that respect joint limits. Discrete collision checks along the interpolation between two configurations are performed for checking its validity. In order to promote early detection of collisions, the popular recursive bisection up to a desired resolution is used. Note that continuous tests not requiring a resolution limit are available and can be integrated [SSL02].

Routine MakePath(c_1, c_2, c_3, c_4) (lines 6 and 10) is called whenever a solution is found. It connects the tree branch joining c_1 and c_2 to the tree branch joining c_3 and c_4, by inserting the path segment obtained with the interpolation between c_2 and c_3.

The node expansion in line 8 uses c_{sample} as growing direction and computes a new configuration c_{exp} as follows:

$$c_{exp} = interp(c_1, c_{sample}, t), \text{ where}$$
$$t = \varepsilon/d,\ d = dist(c_1, c_{sample}).$$

Null is returned in case the expansion is not valid, i.e. if the interpolation between c_1 and c_{exp} is not valid. Otherwise c_{exp} is linked to c_1, making the tree grow by one node and one edge. The factor ε represents the incremental step taken during the search. Large steps make the trees grow quickly but with more difficulty in capturing the free configuration space around obstacles. Inversely, too small values generate roadmaps with too many nodes, slowing down the algorithm.

After a solution is found, a final step for smoothing the path is often required and the popular approach of applying several linearization steps is used. Note however that the smoothing phase is not included in the comparison tests of Section 3. The full geometries of the character and the environment are taken into account for checking collisions and the VCollide package [GLM96] is employed for that.

The next section presents the cache mechanism for improving the performance of the RRT planner.

2.2 Configuration Sampling

The sampling routine used by the RRT planner (Line 2 of algorithm 1) has to generate valid configurations for guiding the growth of the two search trees efficiently. The sampling routine guides the whole search and is of extreme importance in determining how fast a solution is found.

A cache is employed for storing and reusing configurations that were part of successful paths in previous calls to the planner. The idea is that when several motions are planned in a similar environment, the solutions may have several similarities. The goal is in fact to learn sampling heuristics that can better guide the search for the specific kind of motions being planned. The cache is updated every time a successful path is found, therefore it adapts to new tasks as they are solved by the planner.

The cache mechanism is parameterized with 3 parameters:

- The cache size n specifies how many configurations are stored in the cache.
- A probability p controls how often sampled configurations come from the cache instead of being randomly generated. This means that, at each iteration, a configuration is taken from the cache with probability p, and randomly generated with probability $1 - p$. Parameter p can be seen as controlling the exploration versus exploitation tradeoff. Random configurations are always sampled inside range limits and using the

appropriate joint parameterization based on swing, twists and Euler angles [Kal05]. Algorithm 2 shows how p is implemented by using function RandomNumber(), which returns a random number in $[0, 1]$.

- The cache update ratio r controls how many nodes of each solution path found should be stored in the cache. For example if r is 1 all nodes in successful paths are stored in random positions in the cache (replacing the previous cached configurations in those positions). Thus, if r is 0.5, only half the nodes are stored, etc. Note that the cache is updated from the original solution found (before any smoothing operation).

Algorithm 2 SampleConfiguration ()

```
1: c_rand ← null.
2: if RandomNumber() < p then
3:    c_rand ← configuration from a random position in the cache.
4: else
5:    while c_rand is null do
6:       c_rand ← RandomConfiguration().
7:       if c_rand is not valid then
8:          c_rand ← null.
9:       end if
10:   end while
11: end if
12: return c_rand.
```

Note that, when the planner is called for the first time the cache is empty and therefore it is not used. The cache is created only after a first path is found, and usually with size smaller than the target size n. During the time the cache has size s smaller than the target size n, an intermediate probability $p * (s/n)$ is used.

Deciding the best parameters for the cache is not an easy task. Intuitively, the size of the cache should be related to the average number of intermediate configurations (i.e. nodes) in the motions being generated by the planner. Probability p should be high if the tasks being solved seem to be very similar, making sense to reuse successful configurations. The choice for parameter r is also related to how similar are the several tasks being solved. If r is small the cache tends to contain configurations contributed from a larger number of solved tasks, otherwise the most recent solutions will have more influence. Several experimental results are presented in the next section.

3 Experiments

Several experiments were performed for evaluating the use of the sampling cache method within the RRT planner.

The task of relocating books in a book shelf was chosen and a routine was developed for automatically generating book relocation tasks. Each task consists of relocating a book already attached to the character's hand to a new target location randomly chosen. The task generation routine specifies random positions for the target location, the initial arm configuration, and also for all the objects in the shelf. The routine ensures that generated tasks are valid by continued sampling until the relocation task is verified to be well defined, i.e., to not have any collisions. Figure 1 shows few examples of generated tasks.

Fig. 1. Example of automatic relocation tasks generated. Each task consists of relocating the grasped book to a new location specified by a target hand location to be reached.

After each task is generated, it is solved two times. First by the bidirectional RRT planner without using the sampling cache mechanism and then with the cache sampling mechanism activated. In the presented tests, the planners were limited to use at most 5 seconds of computation time, after that they were just stopped. Figure 2 shows one found solution as an example. Note that the IK algorithm is used to translate the target hand location into the goal arm configuration for the planner. A small root translation with leg flexion is also being considered by the planner in these experiments.

Fig. 2. Example of one of the solutions obtained (after smoothing).

3.1 Discussion

The book relocation scenario was chosen because it represents a dynamic scenario with some structure. The objects in the shelf all change of position in each new relocation task, however there are no additional moving obstacles outside of the shelf. Also, the relocation motions exhibit some similarity between them, and yet they are always different. Intuitively, it seems there is an optimal subspace for efficiently sampling arm postures specifically for solving the tasks being simulated. In some sense, the goal here is to assess how far the proposed simple cache mechanism can adapt and capture such a subspace.

First, several tests were performed for determining the best parameters for the cache mechanism. Each test consisted of solving 100 relocation tasks consecutively. A total of 600 tests were performed with the possible combinations of parameters from the following sets: $n \in \{20, 40, ..., 400\}$, $p \in \{0.5, 0.6, ..., 0.9\}$, and $r \in \{1, 1/2, 1/3, ..., 1/6\}$. Nearly all the tests (except only 2) demonstrated that the cache optimization improved the performance of the RRT. The average increase of performance (among all 600 tests) was 1.19 and the best results were around 1.50.

The data also revealed that several tests with similar parameters had a significant variation in their performance increase. This indicates the unpredictability of the perfor-

mance gain due to the high randomness involved in the planner. The same sort of variation is observed in the time performance of each test. For instance, Figure 3 illustrates the variation in the planning times obtained in one of the tests, even if the cumulative performance over time demonstrates a stable increase in performance (Figure 4).

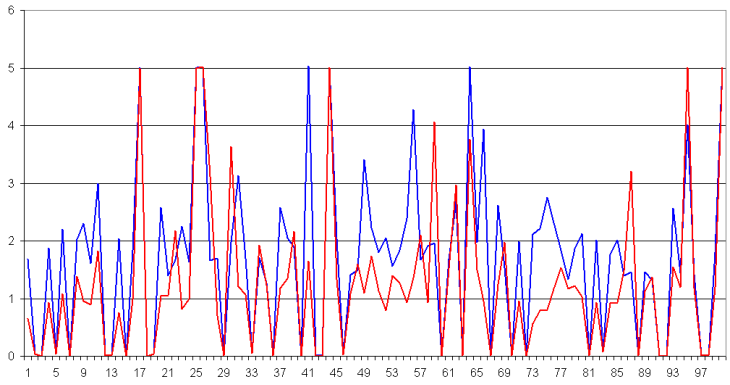

Fig. 3. Computation times in seconds obtained for solving 100 consecutive relocation tasks. Despite the curve variation, it is possible to note that one line (the red one) predominantly stays lower. This line represents the faster performances obtained by using the cache optimization. In the upper line (blue) the cache optimization is not used. Parameters in this test: $n = 200$, $p = 0.9$, $r = 1/6$.

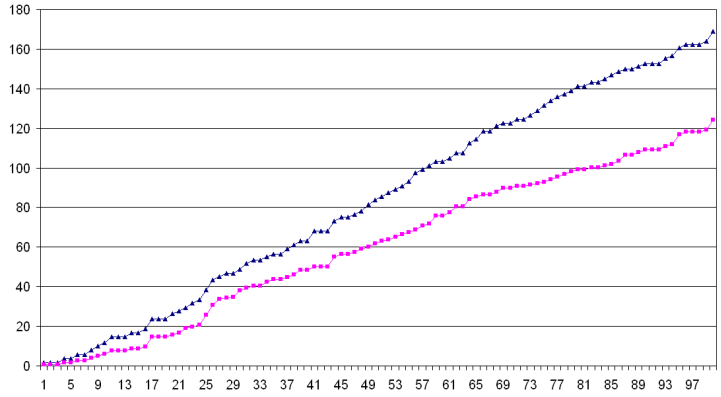

Fig. 4. Cumulative performance in seconds. The lower line (red) shows the faster cumulative performance obtained when solving 100 consecutive relocation tasks using the cache optimization. In the upper line (blue) the cache is not used. In this example the performance gain was of 1.36 and the test data is the same as in Figure 3.

Two sets of parameters were selected for further testing. Table 1 presents additional tests performed with these two sets of parameters. In these new experiments, 100 tests of 100 relocation tasks each were performed for each set of parameters. In these experiments the cache optimization resulted in 1.28 and 1.25 faster results in average. These two sets of parameters were chosen due to their good performance in the first tests. The extensive

tests reported in Table 1 show the best performances that could be obtained with the method.

Table 1. Further tests with two different cache sizes (n). All tests showed some performance increase. The last three columns show the minimum, maximum and average increase in performance.

n	p	r	min	max	average
80	0.8	1/3	1.08	1.53	1.28
120	0.8	1/3	1.07	1.54	1.25

Further Analysis

After analysis and experimentation with the presented relocation tasks, it was possible to manually determine a specific sampling heuristic that also greatly improved the performance of the RRT planner.

This heuristic was found after observing that all the relocations in this example are performed in front of the character and with the elbow flexed. The relocations are also always composed of two distinct phases: bringing the arm closer to the body and then extending it toward the goal location.

By simply sampling random configurations for the arm, but with a constant fixed elbow flexion to almost full flexion it was possible to observe an increase in performance of up to 1.6. This indicates that although the cache sampling mechanism is a generic mechanism that does improve the performance of the RRT planner, it is still not able to capture an optimal sampling heuristic for the specific type of task being solved.

4 Conclusions

A simple technique based on biasing the sampling method of a RRT algorithm was presented for the efficient planning of reaching tasks for humanlike characters in virtual environments. The method leads to a more efficient and practical algorithm for planning reaching motions of average complexity in few seconds.

The proposed cache optimization was extensively tested, demonstrating that learning sampling heuristics can be a powerful approach for easily optimizing sampling-based motion planners. Different sampling heuristics could be learned per task type, and just selected on-line according to the task to be solved.

The presented techniques are simple and robust, therefore suitable to practical applications. Even if not real-time, the presented methods can already be applied to several interactive applications of virtual humans.

References

[BB05a] Burns B., Brock O.: Sampling-based motion planning using predictive models. In *In Proceedings of the IEEE International Conference on Robotics and Automation* (Barcelona, Spain, April 2005).

[BB05b] Burns B., Brock O.: Toward optimal configuration space sampling. In *In Proceedings of Robotics: Science and Systems* (Cambridge, USA, June 2005).

[BV02] Bruce J., Veloso M.: Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002* (Switzerland, October 2002). An earlier version of this paper appears in the Proceedings of the RoboCup-2002 Symposium.

[DKM04] Drumwright E., Kallmann M., Matarić M.: Towards single-arm reaching for humanoid robots in dynamic environments. In *Proceedings of the IEEE-RAS Int'l Conference on Humanoid Robotics (Humanoids)* (Santa Monica, CA, November 2004).

[GLM96] Gottschalk S., Lin M. C., Manocha D.: Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics SIGGRAPH'96 30*, Annual Conference Series (1996), 171–180.

[Kal05] Kallmann M.: Scalable solutions for interactive virtual humans that can manipulate objects. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment (AIIDE'05)* (Marina del Rey, CA, June 1-3 2005), pp. 69–74.

[Lat90] Latombe J.-C.: *Robot Motion Planning*. Kluwer Academic Publisher, December 1990.

[LaV98] LaValle S.: *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. Rep. 98-11, Iowa State University, Computer Science Department, October 1998.

[SSL02] Schwarzer F., Saha M., Latombe J.-C.: Exact collision checking of robot paths. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR'02)* (Nice, decmeber 2002).

[TB96] Tolani D., Badler N.: Real-time inverse kinematics of the human arm. *Presence 5*, 4 (1996), 393–401.