

Learning Humanoid Reaching Tasks in Dynamic Environments

Xiaoxi Jiang
School of Engineering
University of California, Merced
xjiang2@ucmerced.edu

Marcelo Kallmann
School of Engineering
University of California, Merced
mkallmann@ucmerced.edu

Abstract—A central challenging problem in humanoid robotics is to plan and execute dynamic tasks in dynamic environments. Given that the environment is known, sampling-based online motion planners are best suited for handling changing environments. However, without learning strategies, each task still has to be planned from scratch, preventing these algorithms from getting closer to realtime performance. This paper proposes a novel learning-based motion planning algorithm for addressing this issue. Our algorithm, called the Attractor Guided Planner (AGP), extends existing motion planners in two simple but important ways. First, it extracts significant attractor points from successful paths in order to reuse them as guiding landmarks during the planning of new similar tasks. Second, it relies on a task comparison metric for deciding when previous solutions should be reused for guiding the planning of new tasks. The task comparison metric takes into account the task specification and as well environment features which are relevant to the query. Several experiments are presented with different humanoid reaching examples in the presence of randomly moving obstacles. Our results show that the AGP greatly improves both the planning time and solution quality, when comparing to traditional sampling-based motion planners.

I. INTRODUCTION

The ability to plan realistic tasks in realtime is important for many applications, for example: indoor service robotics, human-computer interactions, virtual humans for training, animation, and computer games.

When applied to realistic scenarios, existing sampling-based motion planning algorithms are not yet capable to compute collision-free motions for humanoids in real time. This paper proposes to improve the performance of these planners by integrating the ability to reuse learned skills on new tasks. Our approach is especially beneficial when applied to complex and changing environments.

We provide a method designed to learn from several tasks to be planned in a realistic dynamic environment. Our method learns and reuses attractor points extracted from previously planned reaching paths. The choice of using attractor points is based on evidence gathered from studies with human subjects showing that human arm movements may basically be described as discrete segments between start point, intermediate points, and end point. In addition, these segments appear to be piecewise planar in three-dimensional movements [1]. Inspired by this fact, and based on the observation that sampling-based planners also perform straight connections between landmarks, we propose to select meaningful

landmarks to be reused as attractor points (Figure 1), which are then used to guide subsequent paths in new similar tasks. Note that in most realistic environments, planned solutions for similar tasks often share common structural properties. The proposed method is able to learn such structures by learning and reusing the attractor points.

We apply our Attractor Guided Planner (AGP) to enhance a bidirectional Rapidly-Exploring Random Tree (RRT) planner. It proceeds as follows. Every successful executed reaching task is represented by a group of configurations including the initial point, goal point, attractors, and as well the position of relevant obstacles in the environment. This information is then stored in a task database. When a new reaching task is being queried, AGP searches in the database for a similar task previously planned. Whenever a similar task in the database is found, the random sampling strategy of the original RRT is replaced by a biased sampling process, where samples are selected according to dynamic Gaussian distributions around the group of attractors being reused. The process significantly bias the search to the successful locations in the previous path, preserving similar structures of the solutions. If a partial set of the attractors is no more valid due to a change in the environment, the method gradually changes back to a non-biased random search.

We present experiments simulating several humanoid reaching scenarios. In all the scenarios, the AGP planner shows a significant improvement over the RRT in both computational time and the solution quality. Results also show that even when a good task comparison method is not available, the AGP can still outperform traditional planners. Although all the experiments are performed on humanoids, we believe that the AGP planner will also be beneficial in several other generic motion planning problems.

II. RELATED WORK

Sampling-based motion planning algorithms have proven to be useful for manipulators with several degrees of freedom, such as a human-like structures [2] [3] [4]. These algorithms are mostly based on Probabilistic Roadmaps (PRMs) [5] and/or Rapidly-exploring Random Trees (RRTs) [6], depending on the requirements of the problem. For instance, PRM-like methods are mostly used for multi-query problems but are limited to static environments due to the high cost of building and maintaining roadmaps. In contrast, RRT-like methods are usually efficient single-query planners,

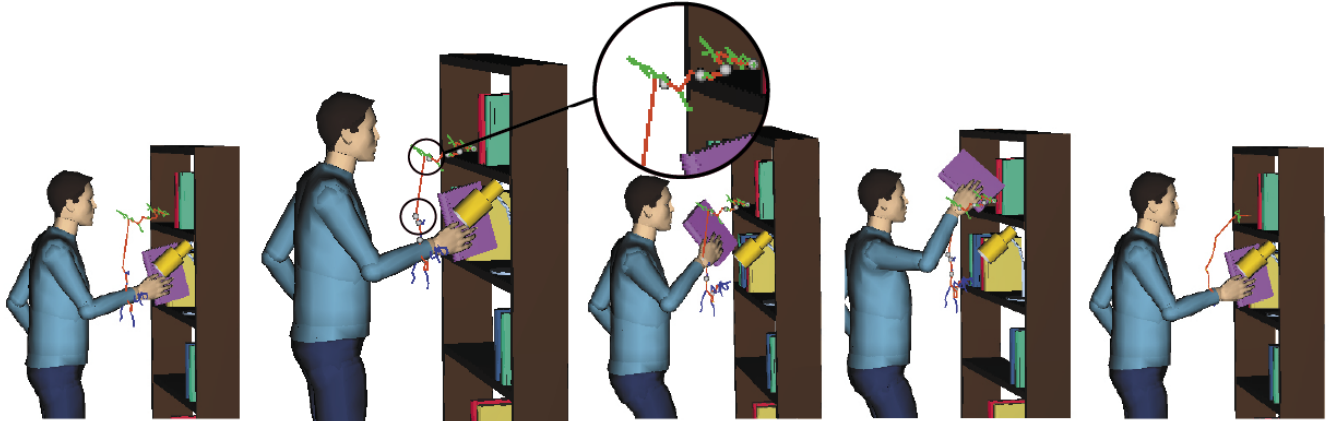


Fig. 1. A sequence of attractors (grey spheres) derived from a solution path (red line) generated by a bidirectional RRT planner (green and blue branches). Note that the attractors specifically represent postures that are trying to avoid collisions, as shown in the third and fourth images. In addition, our Attractor Guided Planning results in much less tree expansions and a smoother solution path (last image).

which construct search trees specifically for each task and environment. Therefore the trees expanded by the RRT are difficult to be reused in a different environment. Due to these limitations, current sampling-based planners scale very poorly to task planning in dynamic environments. We review in the next paragraphs the main approaches for motion planning in non-static environments.

Some works have proposed to combine the advantages of both the multiple-query and single-query techniques in order to obtain efficient solutions for multiple queries in dynamic environments. For instance, dynamic roadmaps have been proposed for online motion planning in changing environments [7] [8], however with limited performance improvement for humanoid reaching tasks.

Other works have explored improvements to single query planners for making them solve multiple queries more efficiently. For instance, the work of [9] uses the RRT as the local planner of PRM, i.e. RRTs are grown between each pair of milestones in the PRM. The RRF planner introduced in [10], builds a random forest by merging and pruning multiple RRTs for future use. A roadmap is then built and maintained for keeping the final set of RRTs concise.

Some other works focus on exploring sampling methods that are more adaptive to the environment [11] [12] [13]. A model-based motion planning algorithm is also proposed in [14]. This method incrementally constructs an approximate model of the entire configuration space using a machine learning technique, and then concentrates the sampling toward difficult regions. While these methods can learn from previous successful experiences to improve future plans, they involve building a model for the entire environment, and thus are not well suited to dynamic environments.

Our method, in contrast, focuses on learning relevant features from successfully planned motions. The learned information

is therefore mainly related to the structure of successful paths, capturing not only the structure of the environment but as well specific properties related to the tasks being solved. It is our own experience that the maintenance of complex structures (such as roadmaps) [8] may not scale well to humanoid reaching tasks and that lighter structures can lead to better results [15]. The work proposed in this paper builds on these previous conclusions.

III. THE AGP PLANNER

A. Problem Formulation and Method Overview

We represent the humanoid character as an articulated figure composed of hierarchical joints. Let the full configuration of the character be $c = (p, q_1, \dots, q_n)$, where p is the position of the root joint, and q^i , $i \in \{1, \dots, n\}$, is the rotational value of the i^{th} joint, in quaternion representation. Assume the character is given a list of random reaching tasks in an environment with a set of moving obstacles $O = (o_1^j, \dots, o_m^j)$. The variable $o_i^j \in O$ here describes the position of the i^{th} obstacle when the j^{th} task is being executed.

A single query motion planner, such as RRT, solves a reaching task T as follows. Let c_{init} be the initial character configuration of the reaching task and let c_{goal} be the target configuration. Two search trees Tr_{init} and Tr_{goal} are initialized having c_{init} and c_{goal} respectively as root nodes, and are expanded iteratively by adding randomly generated landmarks that are free of collision. When a valid connection between the two trees can be concluded, a successful motion is found. We denote this valid path as a sequence of landmark configurations $p = (c_1, \dots, c_n)$.

Different from RRT, our method guides the expansion of the search trees by following a set of attractors derived from a similar previous path. The AGP planner consists with three main parts:

1. **Attractor Extraction.** Everytime a successful path p is obtained, we extract a sequence of attractors $A = (a_1, \dots, a_k)$ from it, where $a_i, i \in \{1, \dots, k\}$, is a configuration point on p . These attractors represent important structural features of p and will be reused for similar tasks.
2. **Task comparison.** Solved tasks are stored in a task database. Each task T is represented by its initial configuration, goal configuration, the attractors, and positions of relevant obstacles, i.e., $T = (c_{init}, c_{goal}, A, O)$. Note that here O is a subset of all the obstacles. Whenever a new task is requested, we find a previously solved task from the database that has the largest similarity to serve as an example. If the value of similarity is lower than a threshold, AGP will decide to operate as a non-biased RRT planner.
3. **Guided planning.** If an example task has been found from the database, its attractor set A is used to guide the planning for the new task. In order to increase the probability that regions around previously successful landmarks are sampled, we form a dynamic Gaussian distribution centered at each attractor point $a \in A$. Samples are then generated according to the Gaussian distribution.

Algorithm 1 summarizes how these three parts take place; their details will be explained in the next sub-sections.

Algorithm 1 AGP Planner($c_{init}, c_{goal}, obstacles$)

- 1: $attractors \leftarrow \text{DBPICKTASK}(c_{init}, c_{goal}, obstacles)$
 - 2: $solution_path \leftarrow \text{GUIDEDPLANNING}(c_{init}, c_{goal}, attractors)$
 - 3: $current_task \leftarrow \text{ATTRACTOREXTRACTION}(solution_path)$
 - 4: $\text{DBINSERTTASK}(current_task)$
 - 5: **return** $solution_path$
-

Line 1 requires selection of an example task from the database that has the largest similarity to the current task. The similarity value is computed by our task comparison function (subsection C). By defining a similarity threshold, we ensure that both the motion query and the environment of the selected example share some common features with the new task. Otherwise, the attractor set A is set to null and the new task will be executed by a non-biased RRT planner.

Line 2 calls the guided planning which returns a solution path. In line 3, the newly generated path is further investigated by fitting it into approximate straight segments divided by a sequence of attractors. These attractors are then considered as a representation of solution path, together with the initial, goal configurations and obstacle positions, stored into the task database, as shown in line 4. To keep the database concise, and also make sure the attractors represent general information of a path, tasks that are generated from scratch always have the priority to be stored in the database. The final step returns the solution path.

B. Attractor Extraction

As stated before, a set of configurations of the character are extracted as attractors from each solution path. Since

the right wrist of the character is the main end effector for reaching tasks, we fit the path of the right wrist approximately into a sequence of piecewise linear segments, and then detect the configuration points as attractors where two segments connect. Given a set of 3D points denoting the global positions of the right wrist, we employ a variation of the LineTracking algorithm [16], which was further experimented and modified by [17]. Although the LineTracking algorithm deals with fitting and extracting lines from 2D Laser data scans, we believe it is also well suited for 3D data sets.

The modified LineTracking algorithm starts by constructing a data window containing the first two points, and fits a line with them. Then it adds the next point to the current line model, and recomputes the new line parameters. If the new parameters satisfy the line condition up to a threshold, the data window incrementally moves forward to include the next point. Otherwise, the new included data point is detected as an attractor point, and the window restarts with the first two points starting from this attractor. Since in our problem the number of configuration points contained by a path is always less than a hundred, we limit the length of the window by ten.

After a list of attractors are detected from LineTracking, we perform a validation procedure in order to ensure all the segments connecting the attractors are valid, i.e., collision free. Suppose a collision is detected on a straight segment (a_i, a_{i+1}) . We find an extra attractor a' , which is the middle point on the original path between a_i and a_{i+1} , and insert it to A between a_i and a_{i+1} . This attractor insertion procedure iterates until all of the straight segments are free of collision. The idea is, in addition to LineTracking, the validation procedure will detect extra attractors in difficult regions around obstacles that always cause collisions. If we apply the sequence of valid straight segments to the same task in the same environment, the solution path will be trivially solved by the attractors.

C. Task Comparison

In order to select a previous path to reuse, We need to define a task comparison metric. Consider the case of comparing the query task T with an example task T' from the database. T' is considered reusable only when its solution path will share similar groups of attractor points with the solution of T . The question at this stage is what information mostly determines the positions of attractor points. One possible approach is to define the metric as the distance between the initial and goal configurations:

$$dist(T, T') = dist(c_{init}, c'_{init}) + dist(c_{goal}, c'_{goal}).$$

The distance between two configurations is computed in the joint space. This naive metric works well with dynamic queries in a static environment, where all the obstacles are

not moving. When it comes to a dynamic environment, the change of obstacle positions may largely affect the structure of a solution path, as illustrated in figure 2. This motivates us to include the changes of obstacle positions in the task comparison metric.

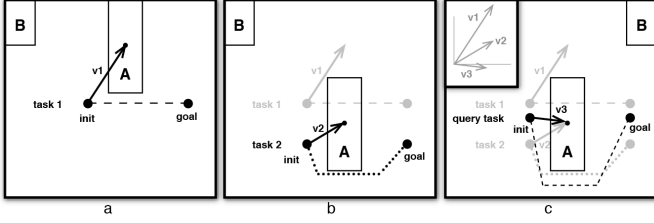


Fig. 2. A planning environment with moving obstacles A and B. Task 1 and task 2 are previously solved (images a and b), and the query task selects an example from the database. v_1 , v_2 and v_3 denote the local coordinates of obstacle A with respect to the three initial configurations. Note that obstacle B is out of the local coordinate system range, thus is not included in the comparison metric. Although the initial and goal configurations of task 1 are closer, the query task chooses task 2 as example because v_2 is closer to v_3 .

When considering obstacles, one important factor is how much influence an obstacle has on the solution path. In figure 2, obstacle B shows a great change of position from task 2 to the query task (images b to c), but since this change has limited impact on the solution path, we should avoid including this obstacle in the comparison. Our comparison technique represents obstacles in local coordinates in respect to the initial and goal configurations.

For each task, we build two coordinate systems with origins respectively located at the initial and goal configurations. Obstacles that are too far away from an origin are not included in the respective coordinate system. Our comparison metric accumulates the distance between each obstacle o from the query task and o' from the example task that is closest to o , computed by their local coordinates. The metric is described as follows.

$$\begin{aligned} dist(T, T') &= h1 * dist(c_{init}, c'_{init}) + h1 * dist(c_{goal}, c'_{goal}) \\ &\quad + h2 * \sum_{o'} \min_o dist(o, o'), \text{ where} \\ &\quad o \in O \text{ from } T \text{ and } o' \in O' \text{ from } T'. \end{aligned}$$

The weights of the motion query distance and the obstacle distance are tuned by heuristics $h1$ and $h2$.

D. Guided Planning

Once a set of attractors is decided to be reused, the sampling strategy in the motion planner is biased toward regions around the attractors. Taking advantage of the attractors is beneficial in two-folds: first, it highly preserves the structure of a successful solution path. Second, whenever a moving obstacle collides with part of the solution path, a large portion of the path is still reusable. Note that since only similar tasks are used by the query, we ensure that the environment does not differ much.

Algorithm 2 summarizes the AGP guided planning algorithm. Suppose a new task T with c_{init} and c_{goal} has selected a task $T' = (c'_{init}, c'_{goal}, A, O)$ from database as example. When planning starts, two search trees Tr_{init} and Tr_{goal} are initialized, and the first and last attractor from set A respectively are selected as sampling guidance, as shown in lines 3 and 4.

Algorithm 2 AGPPLANNER(c_{init} , c_{goal} , A)

```

1:  $attractors \leftarrow A$ ;
2: if  $attractors = null$  then
   return  $RRT\_Planner(c_{init}, c_{goal})$ ;
3:  $a_1 \leftarrow$  first attractor;
4:  $a_2 \leftarrow$  last attractor;
5:  $radius1 \leftarrow 0$ ;
6:  $radius2 \leftarrow 0$ ;
7: while elapsed time  $\leq$  maximum allowed time do
8:    $c_{rand} \leftarrow$  SAMPLE( $a_1$ ,  $radius1$ ).
9:    $c_1 \leftarrow$  closest node to  $c_{sample}$  in  $Tr_{init}$ .
10:   $c_2 \leftarrow$  closest node to  $c_{sample}$  in  $Tr_{goal}$ .
11:  if INTERPOLATIONVALID( $c_1, c_2$ ) then
12:    return MAKEPATH( $root(Tr_{init}), c_1, c_2, root(Tr_{goal})$ ).
13:  end if
14:   $c_{exp} \leftarrow$  NODEEXPANSION( $c_1, c_{rand}, e$ ).
15:  if  $c_{exp} = null$  then
16:    INCREASE( $radius1$ ).
17:    if  $radius1 > threshold1$  then
18:       $radius1 \leftarrow \infty$ .
19:    end if
20:  end if
21:   $c_n \leftarrow$  closest node to  $a_1$  in  $T_1$ .
22:  if DISTANCE( $c_n, a_1$ )  $< threshold2$  then
23:     $a_1 \leftarrow$  next attractor.
24:     $radius1 \leftarrow 0$ .
25:  end if
26:  Swap  $Tr_{init}$  and  $Tr_{goal}$ .
27: end while
28: return failure.

```

We form a dynamic Gaussian distribution around each attractor point in the configuration space. It works as a replacement of the random sampling strategy in the original RRT. The scale of the Gaussian distribution is initialized as zero (line 5 and 6), which means we try to sample on the attractor point as a start (line 8). Lines 9 and 10 require searching for the closet configurations in each tree to the sample. If the interpolation between the two configurations is valid (line 11), Routine MakePath is called to connect the tree branch joining $root(Tr_{init})$ and c_1 to the tree branch joining c_2 and $root(Tr_{goal})$, by inserting the path segment obtained with the interpolation between c_1 and c_2 .

The node expansion in line 14 uses c_{rand} as growing direction and computes a new configuration c_{exp} , as the same as RRT. The difference is, when c_{exp} is returned as void, it means the straight segment leading to the current used attractor is not valid. As a result, in line 16, the scale of the

Gaussian distribution is increased. The increased Gaussian sampler picks samples that are close to the attractor. As time proceeds, if it remains difficult to find a valid sample close to the current attractor, the dynamic Gaussian distribution gradually decreases the bias on the sampling strategy. In line 17, when the scale is larger than a threshold, the sampling changes to non-biased as a RRT planner.

Line 21 finds the closet configuration c_n in each tree to the current attractor. If the distance between c_n and the attractor is small enough, AGP starts to sample on the next attractor point. For the tree rooted at the initial configuration, the order to use the attractors is the same as in the set A , and for the tree started at the goal configuration, the order is opposite.

Deciding the best parameters for the guided sampling is not an easy task. Intuitively, the increase of the sampling radius needs to be small at first and then gradually increase. Also, the two values *threshold1* and *threshold2* need to be tuned such as the non-biased sampling will be activated before *threshold2* switches the current attractor to the next.

IV. ANALYSIS AND RESULTS

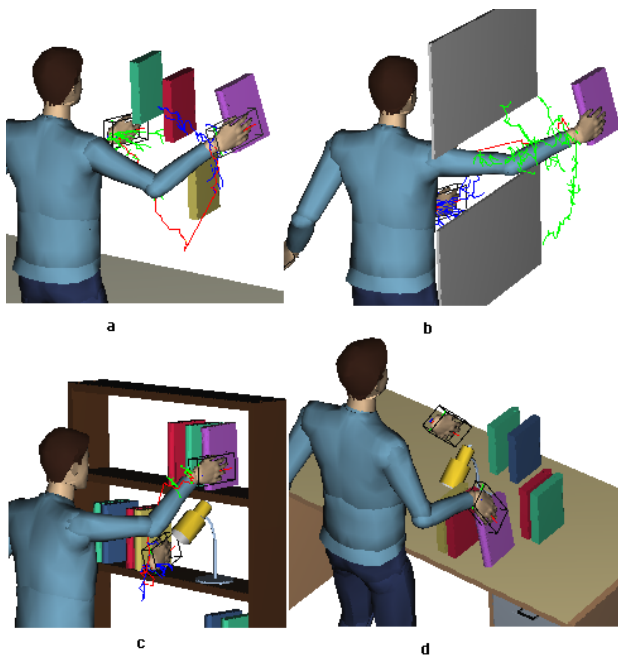


Fig. 3. Reaching tests scenarios.

We tested a number of different humanoid reaching tasks in dynamic environments to evaluate the use of the attractor guided sampling method within the RRT planner. Each test involved planning 200 tasks consecutively. Figure 3 shows snapshots of some of the test scenarios. In Figure 3, scenario (a) represents a single-query problem where every task has the same initial and goal configurations, while the environment is cluttered by three randomly moving obstacles. In scenario (b) we tested dynamic motion queries in a static environment, where tasks are defined as initial and goal

TABLE I

ACCUMULATED COMPUTATION TIME OF AGP (FIRST ROW) AND RRT (SECOND ROW) IN SECONDS FOR 200 CONSECUTIVE TASKS. THE THIRD ROW SHOWS THE NUMBER OF ENTRIES IN THE AGP TASK DATABASE.

3(a)	3(b)	3(c)	3(d)
274.058	341.754	226.879	153.342
518.133	671.404	302.501	221.329
11	11	13	13

configurations randomly generated in the free space on both sides of a narrow passage. In scenario (c) and (d) both the motion queries and the positions of obstacles are dynamic: each task consists of relocating a book already attached to the character's hand to a new target location randomly chosen.

While we can conclude from table 1 that AGP outperforms RRT in all the scenarios, the amount of improvement varies. It is obvious that in some complicated environments where obstacles occlude in most of the problem-solving region, such as the narrow passage scenario in Figure 3(b), AGP shows a better performance than others. On the other hand, for simpler tasks such as relocating books on a shelf or a table, obstacles do not collide with the large portion of free space in front of the shelf, hence the task comparison metric included some information that is not directly relevant to the motion query. This makes the difference between the initial configurations and goal configurations less detectable by the task similarity metric.

Figure 4 shows an example of the solution paths generated by AGP and RRT. It is possible to observe that due to the limited size of the free reachable space, in many cases RRT fails to find a solution in a 5-second time allowance. As revealed by figure 5, AGP successfully solved 98.5% of the tasks, a higher rate than the ratio of success of RRT, which is 94%.

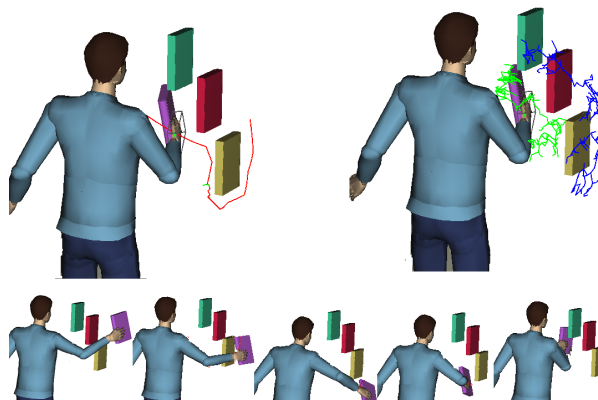


Fig. 4. A reaching problem solved by AGP (left) and RRT (right). The red line denotes a successful path, green and blue branches represent the bidirectional RRT trees. AGP finds solutions within one second with the help of the task database, in comparison, RRT fails to find a solution within the maximum time allowance(5 seconds).

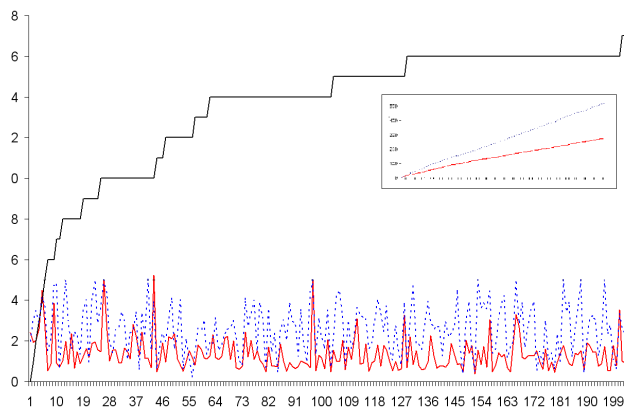


Fig. 5. Computation time (red and blue curved lines) in seconds obtained for solving 200 consecutive reaching tasks, and task database size of AGP (black solid lines). Note that the red solid curved lines (AGP) predominantly stays lower than the blue dashed curved lines (RRT), especially when the database size line stays parallel to the horizontal axis (which means AGP is reusing previous tasks from the database). The accumulated computation time is shown in the small rectangle box.

Note that our algorithm uses attractors extracted from solution paths directly generated by RRT, without any path smoothing being applied. We observed in several experiments that such set of attractors better represent the structure of the task and environment. When attractors are extracted from smoothed paths, besides the extra computation, they are closer to obstacles and therefore more sensitive to environment changes.

V. CONCLUSION

This paper presents an attractor guided planning algorithm (AGP) that learns from previous experiences. The algorithm is able to learn structural features of existing solutions and store them in a task database for later reuse when solving new tasks. Our experiments show that the method is well suited for humanoid reaching problems in changing environments. AGP can also be applied to other generic planning problems, and will show better performance in problems involving the repetition of similar tasks.

In summary, the following main contributions are proposed in this paper:

- the use of a line fitting algorithm for extracting meaningful attractor points from a given path before being smoothed.
- the AGP sampling method which avoids exploring spaces that are irrelevant to the planning query by dynamically biasing the sampling around attractor points.
- a simple but efficient task comparison technique which represents obstacles in local coordinates in respect to the initial and goal configurations.

Many future improvements on our methods are possible. For example, we intend to explore the advantages and drawbacks

of defining attractor points in workspace, instead of in configuration space.

REFERENCES

- [1] S. Schaal, "Arm and hand movement control," *The handbook of brain theory and neural networks*, vol. 2nd Edition, pp. 110–113, 2002.
- [2] J. J. Kuffner and J.-C. Latombe, "Interactive manipulation planning for animated characters," in *Proceedings of Pacific Graphics '00*, Hong Kong, October 2000, poster paper.
- [3] S. Kagami, J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Humanoid arm motion planning using stereo vision and rrt search," in *Journal of Robotics and Mechatronics*, April 2003.
- [4] M. Kallmann, "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.
- [5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, 1996.
- [6] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings of IEEE Int'l Conference on Robotics and Automation*, San Francisco, CA, April 2000.
- [7] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments," in *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 2000.
- [8] M. Kallmann and M. Mataric, "Motion planning using dynamic roadmaps," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, Louisiana, 2004.
- [9] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku, and L. E. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives," in *IROS*, Las Vegas, NV, October 2003, pp. 656–661.
- [10] T.-Y. Li and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management," *Journal of Information Science and Engineering*, pp. 525–538, 2007.
- [11] V. Boor, M. Overmars, and A. Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *Proceedings of ICRA*, Detroit, Michigan, 1999.
- [12] D. Hsu and Z. Sun, "Adaptive hybrid sampling for probabilistic roadmap planning," National University of Singapore, Tech. Rep., 2004.
- [13] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. Amato, "A machine learning approach for feature-sensitive motion planning," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2004.
- [14] B. Burns and O. Brock, "Sampling-based motion planning using predictive models," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment (AIIDE'05)*, Marina del Rey, CA., June 1-3 2005.
- [15] M. Kallmann, "A simple experiment on the effect of biasing sampling distributions for planning reaching motions with rrts," University of California, Merced, Tech. Rep., February 2007.
- [16] A. Siadat, A. Kaske, S. Klausmann, M. Dufaut, and R. Husson, "An optimized segmentation method for a 2d laser-scanner applied to mobile robot navigation," in *Proceedings of the 3rd IFAC Symposium on Intelligent Components and Instruments for Control Applications*, 1997.
- [17] V. T. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics," in *International Conference on Intelligent Robots and Systems (IROS05)*, Edmonton, Canada, 2005.