Studying learning techniques in different phases of PRM construction

Chinwe Ekenna, Diane Uwacu, Shawna Thomas, and Nancy M. Amato

Abstract— **Probabilistic Roadmap Methods (PRMs) solve the motion planning problem in two phases by sampling free configurations and connecting them together to build a map that is used to find a valid path. Existing algorithms are highly sensitive to the topology of the problem, and their efficiency depends on applying them to a compatible problem. Reinforcement learning has been applied to motion planning and rewards the action performed by planners during either sampling or connection, but not both.**

Previous work computed a global reward and action scheme, which saw a setback when heterogeneous environments were concerned. Local learning (connection) was recently introduced to offset this weakness identified during global learning, and there was some improvement in planner performance. These different learning schemes (global and local) have shown strengths and weaknesses individually.

In this paper, we investigate local learning for sampling. We study what type of learning to apply when, and how the two phases of PRM roadmap construction interact, which has not been investigated before. We show the performance using each scheme on a KUKAyouBot, an 8 degree of freedom robot, and analyze what happens when they are all combined during roadmap construction.

I. Introduction

Motion planning problems consist of finding collision-free paths between a given start and goal position for an object. To simplify these computationally hard problems, most algorithms assume that the start and goal are known by the agent. Probabilistic Roadmap Methods (PRMs) [15] are a category of algorithms that solve motion planning problems in two phases. During the sampling stage, valid configurations of the robot in the environment are generated, and during the connection stage those sampled nodes are connected together with edges to construct a roadmap that is used to find the valid path.

There are certain motion planning problems, e.g., planning for deformable robots [11], [21], [22], manipulation planning [14] and computational biology search problems [18], [24], where the efficiency of the solution depends on applying the best method to the corresponding problem. However, it is hard to predict an optimal technique for every environment. Furthermore, heterogeneous environments, which comprise the majority of problems of interest in motion planning, would need more than one algorithm choice applied to different sections of the space.

Machine learning approaches with a cumulative reward approach to actions performed have been applied to PRMs [6]–[8], [13] with some measure of success, such as better roadmap quality and less time to solve a query. Such previous work looked into rewarding the actions of methods during the sampling and connection stage of PRM roadmap construction while using a global approach, i.e., rewarding the actions of these methods based on their performance in the whole environment. However, these methods need the environments partitioned ahead of time in order to ensure maximum performance. This is a major drawback since partitioning the environment doesn't guarantee homogeneous or adequate partitions and in some cases is challenging to do.

This learning approach was taken a step further by exploring the idea of local learning during the connection phase of PRM roadmap construction, i.e., rewarding methods based on the actions of methods within a dynamically determined region [9]. This is helpful in heterogeneous environments where the narrowness of the environment varies and the density of obstacles differs from point to point. This local learning method eliminated the need to explicitly partition the environment.

In this work, we introduce local learning to the sampling stage of PRM roadmap construction and study how beneficial it is to use learning methods during both the sampling and connection phases of PRMs. We discuss and show what stages it would be useful to have these learning methods applied to. If we have to learn at all stages, we need to know the overhead in terms of time to construct these roadmaps compared to non-learning approaches and the quality of roadmaps produced.

We run experiments on a KUKAyouBot and analyze the impact of applying local learning to either one or both phases of the PRM roadmap construction. Our results show that local learning is important during sampling and connection phases with less time needed to solve a given query.

II. Related Work

In this section we discuss work related to this research including adaptive sampling, connection and overall planning for PRM methods.

A. Adaptive Sampling

Many techniques use machine learning to improve the performance of methods during the sampling phase of PRM roadmap construction. In this section we briefly highlight some of these methods.

1) Feature Sensitive Motion Planning [19] uses machine learning to help partition and characterize planning problems. Here, the planning space is subdivided in a recursive manner, then each region is classified and assigned an appropriate planning method. One

Chinwe Ekenna, Diane Uwacu, Shawna Thomas, and Nancy M. Amato are with the Parasol Lab, Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA, {cekenna, duwacu, sthomas, amato}@cse.tamu.edu.

main strength of this approach is its ability to map workspace/C-Space topologies for a particular planner to generate configurations in. However, it is not able to adapt sampling methods over time.

2) HybridPRM [13] employs a reinforcement learning approach to select a node generation method that is expected to be the most effective at the current time in the planning process. However, these samplers are applied globally over the whole problem, and the features of the planning space, such as topology, are not used when deciding where to apply the selected method.
3) Utility Guided Sampling [6], [7] uses information from previous experiences to guide sampling to more relevant areas of C-Space. Every exploration of C-Space provides information to the motion planner. They construct an approximate model of C-Space. Their model captures and maintains information from each configuration and predicts the state of unobserved configurations to reduce collision detection calls.

B. Adaptive Connection

This section focuses on the learning methods applied during the connection phase of PRM roadmap construction.

1) Adaptive Neighbor Connection (ANC): The work in [8] adaptively selects the appropriate connection method to use over time (i.e., a combination of a local planner and neighbor finding approaches based on some distance metric). It does so by maintaining a selection probability for each method. The main weakness of this approach is that it bases its decisions on the performance of connection methods over the entire environment in a global approach.

2) Spatial Adaptive Neighbor Connection (ANC_Spatial): In this work, learning was customized based on the past performance in dynamically determined regions. It made use of spatial information continuously obtained from dynamically determined regions in the environment based on the different connection method's performance. It stored the reward and cost of each previous connection attempt during the roadmap construction process and used this history to learn the appropriate connection method for a given node in a dynamically determined region. This saves the user from partitioning the environment, a trade-off that was needed in [8] to yield good results.

C. Adaptive Planning

This section discusses the adaptive methods that focus on the overall approach, i.e., sampling and connection during PRM roadmap construction.

1) RESAMPL [23] uses local region information (e.g., entropy of neighboring samples) to make decisions about both how and where to sample, and which samples to connect together. This use of spatial information about the planning space enables RESAMPL to increase sampling in regions identified as narrow and decreases sampling in regions identified as free. These approaches do not consider the topology that is discovered within the explored space.
2) Learning from Experience [4] proposes a framework called Lightning that is able to learn from experience. Lightning consists of two modules that run in parallel: a planning from scratch module and a module that retrieves and repairs paths stored in the path library. Any path that is generated for a new query is checked by a library manager to decide how expensive the path is and how similar it is to previously generated paths. However, as the size of the library gets bigger, it becomes impractical to add new paths.
3) Apprenticeship Learning [1] uses inverse reinforcement learning and presents a refined algorithm that compares the trajectories with a more accurate metric and uses the algorithm in the context of apprenticeship learning. It solves problems within the context of motion planning by observing how expert agents behave i.e., learn from demonstration.
4) Curiosity Driven PRM [10] utilizes reinforcement learning to enhance PRM planners for humanoids. To enhance time overhead of PRM as it plans (thinks) before executing actions, the authors created a modular behavioral environment (MoBeE) that implements a model-based reinforcement learner on planners. They assign probabilities to all possible actions from a given state and use them to identify interesting versus non interesting actions. This helps explore least visited areas, thus speeding up the planning stage of PRM. However this work is modeled to work for humanoids and it is not a general PRM method.

III. Local Learning Approach

The local learning approach focuses on the performance of the learning methods within a dynamically determined region. Using reinforcement learning, each method is evaluated in terms of the cost and reward of previous attempts in that region. A method is rewarded for every successful addition it performs. The cost is expressed in terms of the number of collision attempts made.

A. The Reinforcement Learning Model

To give some insight into how we apply reinforcement learning within our framework, the different states refer to the steps of building the roadmap. We take an action every time we choose a connection method or a sampler, and applying the chosen method is the function that gets us to the following state. The reward and cost are then computed after observing the performance of the chosen method, and the agent saves that method with its reward and cost, to be used and updated in the following round. Our reinforcement learning model then consists of:

- Environment states: process of building the roadmap,
- Actions: choosing the next method,
- Functions: applying the chosen method to move between states,

- Rules: computing the reward and cost based on performance.

Algorithm 1 describes the spatial learning algorithm. This is a general algorithm that can be used for sampling and connection. We initialize all the methods M to uniform probability and determine the local learning region as defined by the set of nearest neighbors using NF_{local} in D, where D is a tuple containing the method, reward and cost. For each determined neighbor, we update the probability using the UpdateProbability function in Algorithm 2. We determine the next method to perform an action based on the updated probabilities and call the PerformAction functions (Algorithm 3 and 4) which updates the cost and reward and also adds a configuration to the roadmap (sampling) or add an edge (connection) based on required specifications.

Algorithm 1 Spatial Learning(D, M, NF_{local})

1: Let P_q be a set of probabilities initialized to the uniform distribution, D be data containing tuples (m, reward, cost), NF_{local} be a neighbor finding method, and M be a set of learning methods such that $\mid P_q \mid = \mid M \mid$.
2: Let L be the learning region defined as the set of nearest neighbors to q given by NF_{local} in D.
3: **for** each $n \in L$ **do**
4: P_q = UpdateProbability(n.method, n.reward, n.cost)
5: **end for**
6: Select m based on P_q.
7: (reward,cost) = PerformAction(m)
8: D $\leftarrow$ (m, reward, cost)

The UpdateProbabilty function (Algorithm 2) is used to continually calculate and update the probabilities of the methods. This is important because this is where learning and keeping tabs on their performance is done.It shows the reinforcement learning calculations performed to obtain the probabilities determined for the methods M.

Algorithm 2 UpdateProbability(m,$reward$,$cost$)

1: $w \leftarrow$ Update Weight using reward and m in Equation 1
2: $P_{nc} \leftarrow$ Calculate Probability without cost using w in Equation 2
3: $P_q \leftarrow$ Calculate Probability using P_{nc} , m and cost in Equation 3
4: **return** P_q

First, methods are rewarded according to the number of their returned configurations that are successful. The reward is updated on the cost insensitive probability because it should be independent of the accrued cost.

After finding the updated reward, the weight is calculated as a function of the updated reward:

$$w_i(t+1) = w_i(t) \exp \frac{\gamma x_i^*}{m}, i = 1, 2, ..., m, \quad (1)$$

where x_i^* is the updated reward found by dividing the reward by the cost insensitive probability. For the weights to adapt quickly, we use an exponential factor.

We then find the probability p_{nc}^* for each method m_i ignoring the cost:

$$p_{nc}^* = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{m} w_j(t)} + \gamma\frac{1}{m}, i = 1, 2, ..., m, \quad (2)$$

where $w_i(t)$ is the weight of m_i in step t, t is the number of connection attempts made by the planner, γ a fixed constant that represents the randomness of the method choice and m is the number of methods in the set. We set $gamma$ as 0.5 to ensure all methods have equal chances of being utilized. This formula computes the probability p_{nc}^* as a weighted sum of the relative weight of the m_i and the uniform distribution, which ensures that each method gets a chance to be selected.

We calculate a cost sensitive probability as a function of the cost insensitive one and the cost of connection attempts:

$$p_q = \frac{\frac{p_{nc}^*}{c_i}}{\sum_{j=1}^{m} \frac{p_j^*}{c_j}}, i = 1, 2, ..., m. \quad (3)$$

B. Perform Action during Sampling

Algorithm 3 describes local learning during the sampling stage. We sample using the learnt sampling method m from the set M, create a configuration q, and if is invalid, return the reward and cost as 0 and 1 respectively. Otherwise, we connect the configuration q to the roadmap G. We return a reward of 1, if the current connected component $curr.count$ is greater than or equal to the previous connected component count $prev.count$ where $curr$ = current and $prev$ = previous. Otherwise, we make a calculation on how visible the configuration generated is.

Algorithm 3 Sampling
PerformAction(m)

1: Sample configuration q using m
2: **if** q is not valid **then**
3: **return** (0,1) where 1 is the sampling collision calls
4: **else**
5: Connect configuration to G.
6: **end if**
7: **if** $curr.count \geq prev.count$ **then**
8: $reward = 1$
9: **else**
10: $visibility = curr.succ/curr.att$
11: $reward = \epsilon^{-\gamma * visibility^2}$
12: **end if**
13: cost = # of collision calls after connection + # of collision call after sampling
14: **return** (reward, cost)

As defined in [20], a configuration q is visible to q' if there exists a path (e.g. a straight line) from q to q' that is entirely valid. In our analysis, a method that creates a configuration that increases the visibility of its connected component is more rewarded than one that adds a random

configuration that oversamples the connected component. we determine visibility as a function of current success recorded by the method divided by all the current attempts so far. The reward is thus an exponential function determined by the methods visibility. We determine the cost as the number of collision calls made after the connection has been made including the collision call recorded after the configuration q has been sampled.

C. Perform Action during Connection

Algorithm 4 describes the local learning connection stage for PRM. We connect the configurations based on m from the set M and reward the methods based on the number of successful connections in ratio to the total connection attempts. We calculate the cost as the number of collision calls made after the connection has been made.

Algorithm 4 Connection
PerformAction(m)

1: Connect configuration q to G using m
2: reward = # of successful connections /Total connection attempts
3: cost = # of collision calls after connection
4: **return** (reward,cost)

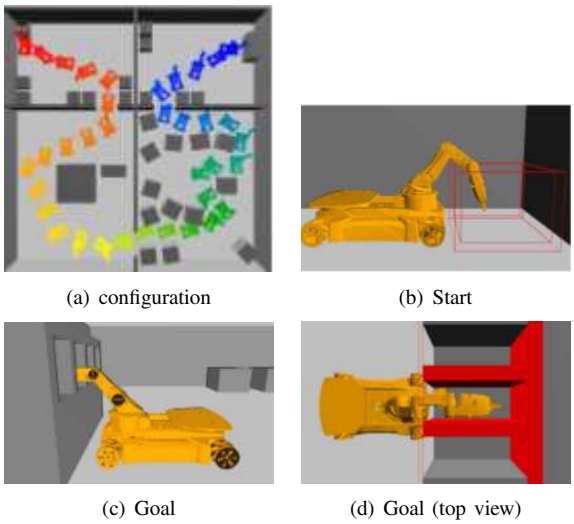
IV. Experimental Results

We begin by performing experiments using individual sampling methods, global and local learning using the K-Closest connection method. Subsequently, we pick the best individual sampling method to run experiments using connection methods including our global learning and local learning connection method. Finally, we apply learning to both the sampling and connection phase of PRM roadmap construction.

A. Experimental Setup

Our tests were run in a C++ motion planning library developed in the PARASOL lab at Texas A&M. All experimental results are averaged over 10 runs. We ran our experiments on a KUKAyouBot robot [16] as shown in Figure 1. It is an 8 DOF robot in an environment with four different rooms. Its base has 5 DOFs that allow it to move forward, backward and rotate, and its arm has 3 DOFs. The robot moves through different rooms with narrow passages and arrives at a destination where it performs an action (grasps or puts an object down). We perform single query experiments and record the time needed to solve the query.

The sampling methods we compare to are Uniform sampling [15], OBPRM [2], Gauss [5], Bridge Test [12], and UOBPRM [25]. For connection methods, we use K-Closest, K-Closest,K-Rand and R-Closest,K-Rand with $k = 10$ and $k_2 = 3k$ as determined in [17], local learning and global learning.



(a) configuration (b) Start (c) Goal (d) Goal (top view)

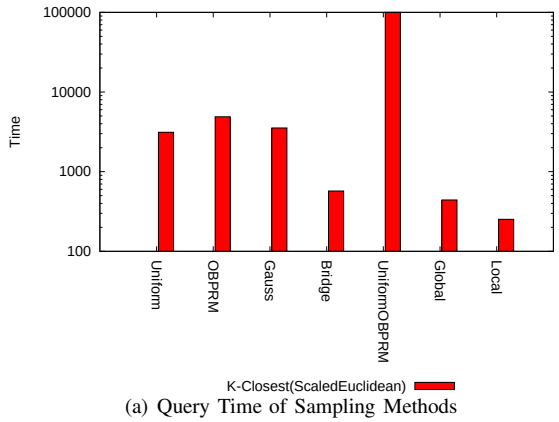
Fig. 1: KUKAyouBot. (a) The KUKAyouBot space configuration is composed of narrow passages that are hard to sample and connect. (b) Figures show the starting (a) and ending (b) positions of the robot and (c) shows an up view of the goal position.

1) Learning in the Sampling Phase: Figure 2(a) shows the time needed to solve the query in the KUKAyouBot environment (see Figure 1(a)) using the different sampling methods listed. Here we determine how each of the sampling methods perform including global and local learning during the sampling phase. We use the K-Closest(ScaledEuclidean) connection method because experiments have shown that it is the simplest and most commonly used [3] which is also the best performing method from our previous experiments [8].

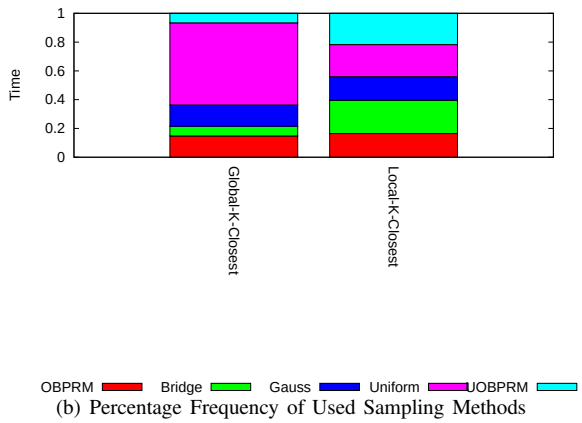
Both local and global learning methods are better than all the individual methods. The local learning sampling method performs better than the global method in this experiment which indicates that learning is important during the sampling phase. The Bridge Test performs better in terms of time to solve the query than the other sampling methods where learning is not applied.

Figure 4(b) shows the frequency of usage of the different sampling methods when learning is applied to the connection phase alone (i.e., global and local results). We see that global learning in most cases learns to use Uniform sampling which is the simplest algorithm and thus would record a smaller cost which the global method leverages on. However it does not learn the Bridge Test sampling method which from our results (see Figure 2(a)) is the better performing individual sampling method. Local learning utilizes all the available sampling methods efficiently and records the smallest time needed to solve the query.

2) Learning in Connection: The results in Figure 2(a) helps us select the sampling method (Bridge Test) to utilize for this experiment. Figure 3(a) shows time need to solve the query using the Bridge Test as a sampling method and the different connection methods including our global and local

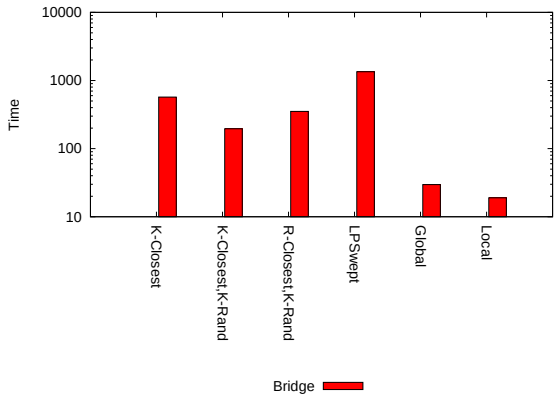


(a) Query Time of Sampling Methods

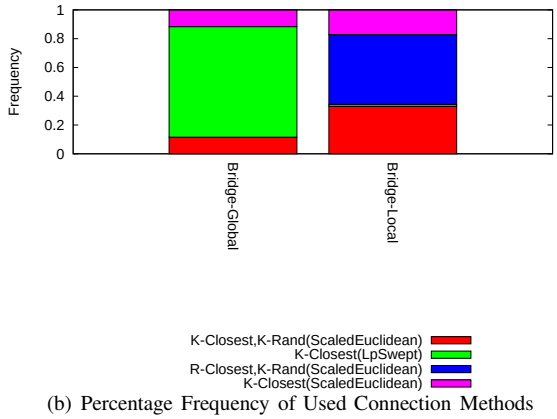


(b) Percentage Frequency of Used Sampling Methods

Fig. 2: Query Time and Frequency of Usage for Different Sampling Methods



(a) Query Time of Connection Methods



(b) Percentage Frequency of Used Connection Methods

Fig. 3: Query Time and Frequency of Usage for Different Connection Methods using Bridge Test Sampling

learning connection methods. We perform this experiment to determine if applying learning during the connection stage is beneficial.

From the plots, the local connection method outperforms all the other methods. It outperforms the other methods by a magnitude of 10 as is the case with the LpSwept method. This result indicates that learning is indeed important during the connection phase. The same trend of learning benefits seen in section IV-A.1 appears here.

Figure 4(b) shows the performance the frequency of usage of the connector methods for learning employed during the connection stage. Here we see that global connection learning learns LpSwept which is not one of the better connection method. Global learning connection's performance as earlier discussed in [8] is as result of the need to partition the environment to get good results which we did not do in these experiments. The local connection method however, utilizes the methods more which is an important feature of the local learning approach, i.e., their ability to utilize resources in a more intelligent way, which in this case is being able to use methods available as the need arises.

3) Learning in Both Phases: Figure 4(a) shows the time needed to solve the query when learning (global and local) is applied to both the sampling and connection phase. Our results show that applying local learning to sampling and global learning to connection solves the query in the least time.

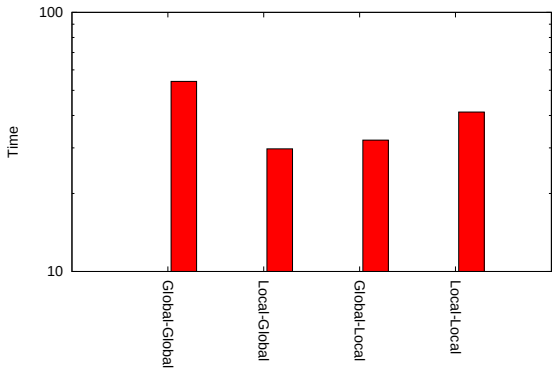
We see that applying local learning to sampling and global learning to connection is the best performing combination, followed closely by a global sampling and then local connection approach. Figure 4(b) shows the frequency of usage during learning both for sampling and connection and we see that the methods utilize all available methods in it list better than other methods.

Global sampling learning in most cases learns to use Uniform sampling which is the simplest algorithm and thus would record a smaller cost which the global method would leverage on. However, global connection learning in most cases picks the LpSwept method which is a method that spans the volume of the space when identifying neighbors which is more effective but tends to be more expensive.

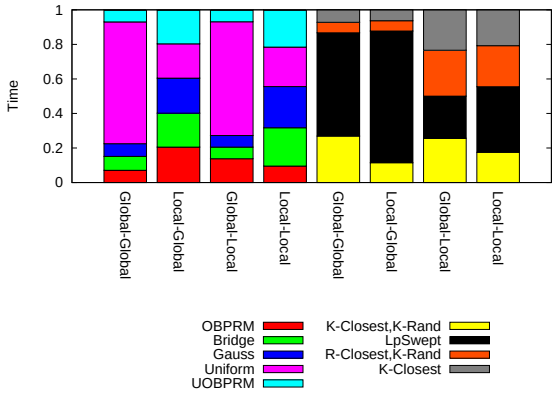
In general however, we have shown that being able to have sampling and connection methods available to learn from during PRM roadmap construction improves in the overall time taken to solve a given query.

V. Conclusion

We have analyzed the impact of using global and local reinforcement learning strategies on the two phases of planning for PRMs. We performed experiments on a KUKAyouBot



(a) Query Time with Learning Applied to both Sampling and Connection



(b) Percentage Frequency of Usage during the Sampling and Connection Phase

Fig. 4: Query Time and Frequency of Usage during both Phases of PRM Construction

robot in a heterogeneous environment, we were able to show that learning is important to solving motion planning problems. We discussed the benefits of applying local learning during all phases of PRM roadmap construction. In the future, we plan to analyze the performance of our learning approach on problems with more degrees of freedom and more complex environments. We also plan to continue investigating on fine tuning our local learning algorithms to increase optimality of runtime.

REFERENCES

[1] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. September 2008.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: an obstacle-based PRM for 3d workspaces. In *Proceedings of the third Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd. (WAFR '98).

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. Automat.*, 16(4):442–447, August 2000.

[4] D. Berenson, P. Abbeel, and K. Goldberg. A robot path planning framework that learns from experience. In *In the proceedings of the International Conference on Robotics and Automation (ICRA)*, 2012.

[5] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1018–1023, May 1999.

[6] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3313–3318, 2005.

[7] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Proc. Robotics: Sci. Sys. (RSS)*, pages 105–112, 2005.

[8] C. Ekenna, S. A. Jacobs, S. Thomas, and N. M. Amato. Adaptive neighbor connection for PRMs: A natural fit for heterogeneous environments and parallelism. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 1–8, Tokyo, Japan, November 2013.

[9] C. Ekenna, D. Uwacu, S. Thomas, and N. M. Amato. Improved roadmap connection via local learning for sampling based planners. Technical Report TR15-006, Texas A&M, April 2015.

[10] M. Frank, J. Leitner, M. Stollenga, A. Forster, and J. Schmidhuber. Curiosity driven reinforcement learning for motion planning on humanoids. *Frontiers in Neurorobotics*, 7(25), 2014.

[11] R. Gayle, M. C. Lin, and D. Manocha. Constraint-based motion planning of deformable robots. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1046–1053, April 2005.

[12] D. Hsu, T. Jiang, J. Reif, and Z. Sun. Bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4420–4426, 2003.

[13] D. Hsu, G. Sánchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3885–3891, 2005.

[14] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motion for interactive object manipulation and grasping. *Computer Graphics Forum*, 22(3):313–322, Sept. 2003.

[15] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[16] KUKA. http://www.youbot-store.com.

[17] T. McMahon, S. Jacobs, B. Boyd, L. Tapia, and N. M. Amato. Local randomization in neighbor selection improves prm roadmap quality. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 4441–4448, 2012.

[18] K. Molloy and A. Shehu. Biased decoy sampling to aid the selection of near-native protein conformations. In *BCB*, pages 131–138. ACM, 2012.

[19] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, Springer Tracts in Advanced Robotics, pages 361–376. Springer, Berlin/Heidelberg, 2005. (WAFR '04).

[20] M. A. Morales A., R. Pearce, and N. M. Amato. Metrics for analyzing the evolution of C-Space models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1268–1273, May 2006.

[21] T. Morita, J. Takamatsu, K. Ogawara, H. Kimura, and K. Ikeuchi. Knot planning from observation. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3887 – 3892 vol.3, sept. 2003.

[22] S. Rodriguez, J.-M. Lien, and N. M. Amato. Planning motion in completely deformable environments. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2466–2471, May 2006.

[23] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. (RESAMPL): A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[24] S. Thomas, X. Tang, L. Tapia, and N. M. Amato. Simulating protein motions with rigidity analysis. In *Proceedings of the 10th annual international conference on Research in Computational Molecular Biology*, RECOMB'06, pages 394–409, Berlin, Heidelberg, 2006. Springer-Verlag.

[25] H.-Y. C. Yeh, S. Thomas, D. Eppstein, and N. M. Amato. UOBPRM: A uniformly distributed obstacle-based PRM. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 2655–2662, Vilamoura, Algarve, Portugal, 2012.