Offline and Online Evolutionary Bi-Directional RRT Algorithms for Efficient Re-Planning in Dynamic Environments

Sean R. Martin, Steve E. Wright, and John W. Sheppard, *Fellow, IEEE*

***Abstract*— This paper explores the use of evolutionary algorithms (EAs) to formulate additional biases for a probabilistic motion planner known as the Rapidly Exploring Random Tree (RRT) algorithm in environments with changing obstacle locations. An offline EA is utilized to produce a bias in an obstacle filled environment prior to rearranging the obstacles. It is demonstrated that the offline EA finds a bias reflecting the original environment and improves the RRT's efficiency during re-planning in environments with a small number of rearrangements. The Rapidly Exploring Evolutionary Tree (RET) algorithm is introduced as a hybrid RRT algorithm employing an online EA. It is demonstrated that the RET can improve the RRT's performance during re-planning in environments with many rearranged obstacles by exploiting characteristics of a balanced spatial kd-tree.**

I. Introduction

THIS work considers extensions to the Rapidly Exploring Random Tree (RRT) algorithm to more effectively re-plan in the presence of changes in the robot's environment. These changes may arise over time due to uncertainty in the form of sensor noise and sensor bias, or from obstacle movement between sensor updates. Efficient motion re-planning is an important problem in the fields of manipulation and robotic navigation since environments are rarely static in real world applications. Thus, a robot needs to continuously monitor and recalculate plans of motion. Since robots are often constrained by limited computational resources and time, it is important to make the process of re-planning as efficient as possible.

In this work we consider an improvement to the RRT algorithm in the form of an additional bias towards the edge of the explored regions of a given configuration space. This bias takes the form of points in the robot's configuration space that are occasionally sampled instead of the uniform distribution employed in the RRT algorithm. This makes the RRT more efficient since it does not waste time re-exploring areas that have already been examined. More specifically, this bias is found through the use of an evolutionary algorithm (EA). We consider both an offline and online EA and the pros and cons of offline and online approaches.

Manuscript received May 18, 2007.

S. R. Martin is with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (phone: 443-778-2780; fax: 443-778-6896; e-mail: Sean.Martin@jhuapl.edu)

S. E. Wright is with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (e-mail: Steve.Wright@jhuapl.edu)

J. W. Sheppard is an Assistant Research Professor at Johns Hopkins University, Baltimore, MD 21218 USA (e-mail: jsheppa2@jhu.edu)

II. Previous Work

In [7] Kuffner and LaValle present the RRT and bi-directional RRT along with applications to control theory. The RRT is a probabilistic motion planner and satisfies the criteria for probabilistically complete planners: as the number of samples approaches infinity, the probability that the plan connects the start and goal regions approaches 1. This of course assumes that a plan exists from the start to the goal region.

Applying the RRT algorithm to environments requiring efficient re-planning is not a new idea. Veloso and Bruce designed the extended RRT (ERRT) to increase performance through caching previous plans and adaptively biasing search towards older waypoints, the goal, or random points as the environment changes [6]. This work is extended and improved in the DRRT and MP-RRT algorithms [8][10].

In the presence of obstacles, taking samples uniformly distributed over the configuration space is not always the best idea. Often these random sample points will occur inside an object. To compensate, in [4] and [5], the authors propose using an adaptive bias to generate samples for the RRT.

In [2] and [3], the authors propose a "dynamic domain distribution" to address several of the shortcomings of the original RRT algorithm. The dynamic domain distribution is based on the uniform sampling distribution, except that the distribution is restricted to a boundary domain. This is done by restricting samples to be within a radius of the nearest vertex. This helps the RRT to more thoroughly explore the configuration space within a bounded area.

III. Background

A. Bi-Directional RRT Planner

This research employs a variation of the standard RRT algorithm known as the bidirectional RRT to formulate complete plans from the initial orientation to a goal orientation. This approach grows a tree from the initial orientation and a second tree from the goal orientation. In the bi-directional RRT the trees alternate between moving towards random locations in configuration space and moving towards each other. This is helpful in focusing the search from the start to the goal [7]. Pseudo code for the basic bi-direction RRT algorithm is shown in figure 1. The

function *RandomState():state* returns a state from a robot's configuration space drawn from a uniform distribution. *Extend(current:state,target:state)* returns a new state a constant distance from the current state towards the target state. *ExtendToTree(extended:state,T2:rrt-tree)* extends the closest node in tree *T2* to the newly extended node in *T1*. *ExtendToTarget(env:environment,T1:rrt-tree,inc:integer)* extends tree *T1* to a random point in configuration space.

B. Spatial KD-Tree (SKD-tree)

The skd-tree was originally introduced in [11] as a modified binary tree for storing points in a n-dimensional space where layers correspond to dimensions of the space. The RRT does not require the use of a skd-tree; however, a skd-tree often provides a more efficient means of storing tree nodes [6]. For the work in this paper it is necessary to re-balance the skd-tree because the bias introduced shows preference towards the boundary of the subspace C_{sub} of configuration space containing points in the RRT. There are numerous ways of choosing a pivot and dimension during the process of building a skd-tree. We cycle through the dimensions in order and take the median of the data set at each dimension as the pivot. By doing this, nodes at the top of the tree are near the center of C_{sub} and nodes near the boundary of C_{sub} are reached by branching mainly in the same direction away from the center (i.e. always taking the right branch or always taking the left branch). As the tree grows, bias shifts away from the center towards those nodes added on the edge of C_{sub}. This is explained further in the next section.

Rebalancing the skd-tree can be costly. A simple threshold on the number of nodes added before rebalancing was used in this work. However, a more efficient method of rebalancing the tree during addition of new RRT nodes is introduced in [9] and used as part of the ERRT algorithm in [6].

IV. Fitness Functions

EAs are methods of stochastic parallel search that involve evolving a solution to a problem via genetic operations, such as mutation and crossover. These solutions take the form of individuals in a population which are allowed to combine and interact. Individuals are evaluated via fitness functions to determine if they should be represented in future populations. In this paper, individuals are points in configuration space and fitness is based on how efficiently these points cause the RRT to grow.

A. Offline EA Combined with the RRT

As shown in figure 4, the offline EA runs a slightly modified bidirectional RRT algorithm on each individual in the population within a static environment. The individual is sampled in the *BiasState()* function. The EA uses a straight forward measure of the total failed samples as its fitness function, where a failed sample is a sample that results in a collision with an obstacle as a node is being extended. This is indicated by the *EmptyState* variable returned by the Extend function. This fitness value is then assigned to each individual in the population.

B. The Rapidly Exploring Evolutionary Tree (RET)

As shown in figure 5, the online EA uses a fitness function $f()$ related to the number of left and right branches traversed during the insertion of a new node in the skd-tree. This new node is generated from the bias point in the EA population which periodically replaces the uniform sampling distribution in the RRT.

More formally, consider a skd-tree *S*. Let *m* be the dimension of *S*, and let *p* denote a new node being inserted into the skd-tree. Let $L_n(p)$ be the set of layers *l* traversed during insertion for which $n = depth(l) \bmod(m)$. Further divide $L_n(p)$ into two sets $L_{n,L}(p)$ and $L_{n,R}(p)$ where

```
function Bi-DirectionalRRTPlan (env:environment, initial:state,
                                goal:state, T1:rrt-tree,
                                T2:rrt-tree)

    var nearest, extended, target:state;
    var inc:integer;
    inc := 1;
    nearest := initial;
    T1 := initial;
    T2 := goal;

    while(Distance(T1,T2) < threshold)
        extended = ExtendToTarget(env, T1, inc);
        ExtendToTree(extended, T2);

        swap(T1, T2);
        inc = inc + 1;

function ExtendToTarget(env:environment, T1:rrt-tree,
                        inc:integer):state

    var nearest,extended,target:state;
    target = RandomState();
    nearest = Nearest(T1, target);
    extended = Extend(env,nearest,target);

    if extended != EmptyState then
        AddNode(T1, extended);

    return extended;
```

Fig. 1 The basic bi-directional RRT planner

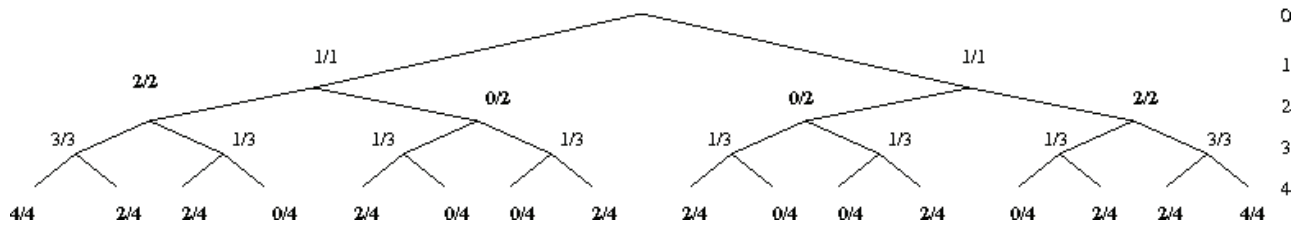

Fig. 2. The values of f() at the first 4 layers of a balanced one dimensional skd-tree.

$L_{n,L}(p)$ contains all layers from which traversal continues down the left branch of the binary skd-tree and $L_{n,R}(p)$ contains all layers from which traversal continues down the right side. The following two functions are defined over the cardinals of these sets.

Definition:

for n < m, $f_n(p) = abs(\left|L_{n,L}(p)\right| - \left|L_{n,R}(p)\right|)$

Definition: $f(p) = \dfrac{\max_n(f_n(p))}{\left|L_n(p)\right|}$

Dividing through by the total layers visited during node insertion normalizes the function such that nodes added to lower layers in the tree do not have an additional bias.

The premise of this paper is to introduce a bias to the RRT algorithm which shows preference to nodes created away from the center of C_{sub}. However, there are multiple C_{sub} regions represented by sub trees in the skd-tree. Thus, there is no straight forward analytical solution. Instead, the bias must change and increase as more nodes are added to a particular C_{sub}. From the above definition, the maximum value of $f(p)$ is 1. Thus, to increase the bias of preferred sample points, the bias on other points must decrease. The following is a simplified proof of this behavior for a skd-tree of a single dimension. We provide empirical evidence for higher dimensions in section V.

Theorem 1: Assuming $f() = 0$ iff n = 0, then for each parent node *p* at layer *n*, and for all even values of *n* >= 2, the average value of $f(nd)$ for child nodes *nd* decreases at layer *n*+2 compared to layer *n* in a single dimensional balanced skd-tree.

Proof:

Since dim(*S*) = 1, $L_n(p)$ = *n*. Proof proceeds by induction on the even layers of the tree.

Base Case: *n = 2*

Consider the change in average value between layer 2 and layer 4. Figure 2 shows all values of $f()$ for every node in a balanced skd-tree. At layer 2 the average value is ½ and at layer 4 the average value is reduced to 3/8. Thus, the base case holds as required.

Inductive Step: *n > 2*

Assume without loss of generality that in the following cases the node *p* at layer *n* is the left child of its parent node from layer *n*-1. If *p* were the right child instead, the values at layer *n*+2 would simply be in the opposite order. Consider the possible parent node values for $f(p)$ divided into the following two cases. Case A is drawn out in figure 3.

Case A:

$f(p) = c/n$ for $0 < c <= n$. For this case:

$avg(f(nd)) = \dfrac{4c}{4(n+2)} = \dfrac{c}{n+2}$. Thus, the average has decreased compared to $f(p)$.

Case B:

$f(p) = 0/n$. This case is the beginning of a new tree. By resetting *n* to 0 for node *p*, then for all child nodes of *nd*, the case is identical to the base case. □

V. ALGORITHMS

A. Offline EA Combined with the RRT

The offline EA runs with the modified version of function *ExtendToTarget* as shown in figure 4 acting as the fitness function as explained in section IV. The final evolved bias is then used in this same version of *ExtendToTarget* to plan through a changed environment. The genotype of the EA consists of a single robot configuration for each tree in the bi-directional RRT algorithm. This configuration is sampled instead of the uniform distribution at every even numbered sample in the RRT. Odd numbered samples are still taken from the uniform distribution.

It follows that the sampled robot configuration acts as a bias to draw the tree in a certain direction in configuration space during growth. Thus, the purpose of the offline EA is to find the best robot configuration for each of the two trees. The fitness function involves running the RRT algorithm with a population of bias robot configurations. The fitness is

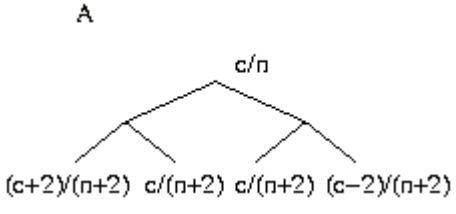

Fig. 3. The values of $f()$ for case A of the inductive step.

the total failed samples (due to collisions with objects) after the RRT completes. We run the EA for 50 generations on a population size of 20 configurations. After each generation, we perform single point crossover with 50% probability and random mutation with 25% probability.

B. The Rapidly Exploring Evolutionary Tree (RET)

The RET runs with another modified version of function *ExtendToTarget* as shown in figure 5. The genotype in the online EA is identical to the offline EA genotype; however, it is used much differently. Further, the RET does not have a set number of generations. It runs continuously during sampling and replaces draws from the uniform distribution in the RRT algorithm with draws from individuals in the EA population. Thus, there is very little overhead aside from sorting the population after all individuals have been sampled and maintaining the skd-tree. References [6] and [9] provide ways of efficiently maintaining the skd-tree. From theorem 1 it is evident that adding a large number of nodes to a balanced skd-tree will result in lower average value over each layer of the tree as depth increases. Thus, points biased by $f()$ will increase in bias as the tree increases in size.

It is important to point out that to maintain a probabilistically complete planner, the planner cannot always sample from the bias points. In other words, the planner must balance exploration with exploitation. With this in mind the online EA was designed with 50% elitism similar to the offline algorithm. However, instead of explicitly setting fitness to the number of failed samples, the fitness is assigned to a new node of the skd tree according to its $f()$ value. Truncation selection was utilized but no crossover was used during generation of offspring. Instead, mutation was 100% on the lower 50% of the population after sorting on fitness level.

```
function ExtendToTarget(env:environment, T1:rrt-tree,
                        inc:integer):state

    var nearest,extended,target:state;

    if IsEven(inc) then
       target = RandomState();
    else
       target = BiasState();

    nearest = Nearest(T1, target);
    extended = Extend(env,nearest,target);

    if extended != EmptyState then
          AddNode(T1, extended);

    return extended;
```

Fig. 4 The modified version of the bi-directional RRT used during and after development of a biased state via an offline EA

VI. Experiments

A. Experimental Setup

The motivation for this work is to find ways of improving the speed of the RRT algorithm. To do this, the number of samples before finding a complete path from a start configuration to a goal configuration must be reduced. The number of failed samples where the extend operation of the RRT algorithm fails to return a node is considered in addition to the number of successful samples that lead to new nodes in the tree structure.

The work space for the following experiments is 1000 by 700 pixels in $\Re^2$. Three different configuration spaces are considered in this workspace corresponding to a point robot, a rod robot, and a linked rod robot. More formally the configuration spaces are: $\Re^2, \Re^2 \times S^1, \Re^2 \times S^2$ where S represents angles for the rod and linked rods. Every configuration / node in the RRT was translated into the work space and any object overlap was considered a failed sample.

The experiments were conducted in an obstacle filled room containing 13 convex objects. To simulate a re-planning scenario in a dynamic environment, the room underwent two types of object rearrangement. These rearrangements were conducted after the offline EA had optimized biases based off the original room configuration. Small amounts of object rearrangement involved picking a single object and moving it a random amount of at most 320

```
function ExtendToTarget(env:environment, T1:rrt-tree,
                        inc:integer):state

    static var p:population;
    var p':population;
    var nearest,extended,target:state;

    if inc > length(p) then
          SortByFitness(p);
          p' = null;
          for each individual i ∈ p
              if i is upper 50% then
                  AddIndividual(i,p');
              else
                   i = RandomState();
                   AddIndividual(i,p');
          p = p';
          inc = 1;

    target = p(inc);
    nearest = Nearest(T1, target);
    extended = Extend(env,nearest,target);
    if extended != EmptyState then
         AddNodeToSKDTree(T1, extended);
         AssignFitness(p(inc), f(extended));
    else
         AssignFitness(p(inc), 0);

    return extended;
```

Fig. 5 The modified version of the bi-directional RRT used in the RET

pixels in both the *x* and *y* direction. This is referred to as local object movement in the following experiments. Global object movement involved large amounts of movement by randomly moving every object in the room at most 320 pixels in both the x and y direction.

In all the experiments each algorithm was run in the same environment. Objects were moved initially, each of the three algorithms was executed, data was collected, and then the objects were returned to their original configuration. This was repeated 1000 times for each type of movement (local and global) in the room. After the objects were moved, a check was performed to determine if either the start or end configuration was overlapped by an object and if necessary the room was reset before continuing. To handle the case where the result of randomly moving objects may enclose either the start or the goal, an upper limit of 20 million samples was set before declaring no solution in the room and resetting. The resets did not count toward the 1000 configurations tested in each environment.

The surrounding lines around the configuration space in figure 6 represent a boundary for the *x,y* coordinate of the robot. This initial point cannot move outside the boundary but the rest of robot can move outside.

VII. Results

Tables I through III show the results of experiments. These values are all averages over the 1000 runs along with the associated standard deviations. Note that the standard deviations in these tables are high. This is likely due to the random obstacle movement which can result in either closely spaced obstacles which require careful planning or widely spaced obstacles which are easier to plan through.

With no object movement, the offline EA produces substantial improvement as seen in table I. However, these experiments are strictly for comparison purposes since more realistic scenarios involve changing environments.

To be more specific about the amount of improvement obtained by using the offline and online EAs, consider the 4D robot. By performing a statistical *z* test on the number of failed samples when comparing the offline bias against the unbiased algorithm over the local movement experiment from table II a *z* value of 51.03 is obtained. This value shows a very high statistical significance in the difference of failed samples. The *z* value obtained when comparing the online bias against the unbiased algorithm is 19.27 which is also extremely high. Both *z* values indicate improvement at a 99% confidence level.

When considering global movement data from table III the *z* value for the offline bias compared to the unbiased algorithm drops to 2.20 indicating improvement at a 97% confidence level. However, the *z* value for the online bias compared to the unbiased algorithm remains relatively high at 7.47. This *z* value indicates improvement at a 99% confidence level.

Although the online bias does not provide as large of an improvement during global movement as the offline bias does during local movement, it requires roughly half the number of samples that the unbiased algorithm requires. This could prove useful in high dimensional environments with drastic changes or platforms with noisy sensors.

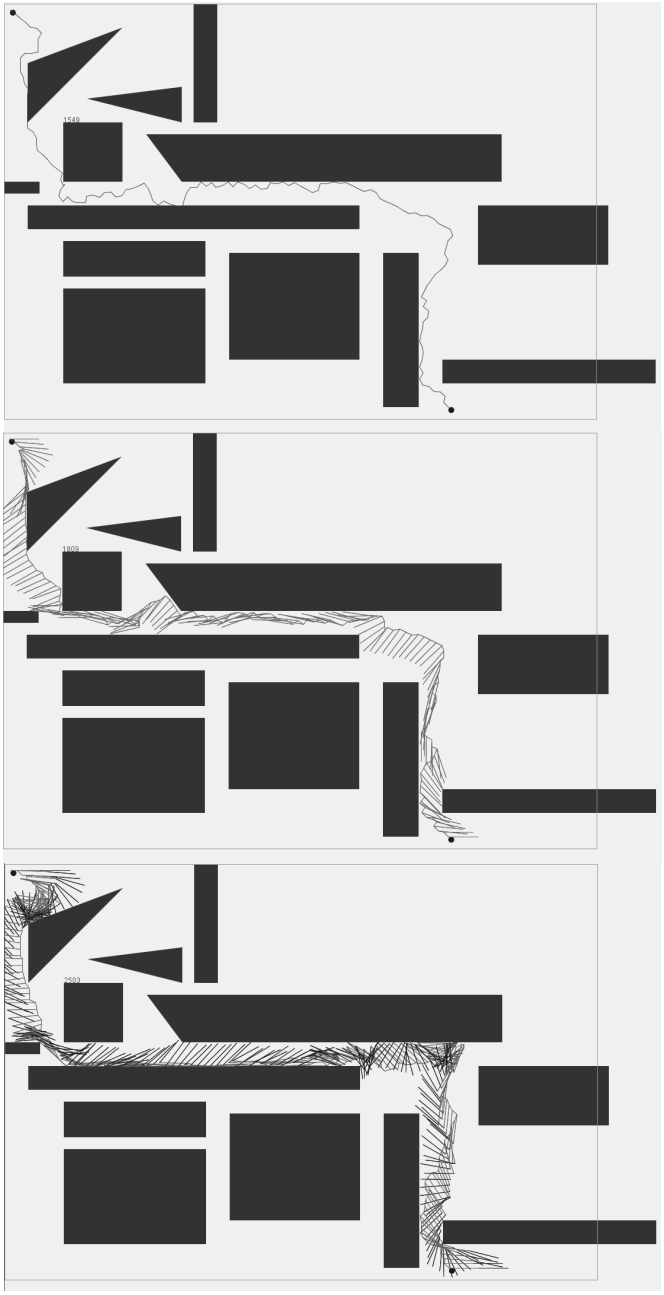

Fig. 6. Completed plans from a start configuration (lower right corner of each image) to a goal configuration (upper left corner of each image). From the top: a 2 dimensional point robot navigating the obstacles, a 3 dimensional rod robot navigating the obstacles, a 4 dimensional linked rod robot navigating the obstacles.

VIII. Conclusion

The experiments in this work suggest that when knowledge about the configuration space is available prior to planning, that information can be exploited and substantially reduce planning time during subsequent re-plans. This is evident in nearly all the experimental results involving the offline bias to the RRT algorithm. However, these results came from room configurations that either didn't change during re-planning or changed a relatively small amount (i.e. local object movement). This is realistic in indoor environments where walls will likely not change during a re-plan, but people or objects could move.

In constantly changing environments, an algorithm such as the RET provides improvement by considering the data structure representing sampled points in an unknown environment. However, this improvement is not a substitute for knowledge about an environment prior to planning.

Additional work needs to be performed in combining the approaches introduced in this paper with efficient routines to maintain a balanced skd-tree such as [6] and [9]. This work also needs to be combined with newer variations of heuristic RRT algorithm such as [8] and [10]. Such combinations could eventually yield a RRT algorithm for planning through dynamic high dimensional configuration spaces in real time on limited computing platforms.

References

[1] J Kim and J. M. Esposito, "Adaptive Sample Bias for Rapidly-exploring Random Trees with Applications to Test Generation," American Control Conference, Portland, OR, June 2005.

[2] L. Jaillet, A. Yershova, S. M. LaValle, and T. Simeon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," Proceedings IEEE International Conference on Intelligent Robots and Systems, 2005.

[3] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," Proceedings IEEE International Conference on Robotics and Automation, 2005.

[4] JM Esposito, J Kim, V Kumar, "Adaptive RRTs for validating hybrid robotic control systems," Proc. Workshop on Algorithmic Foundation of Robotics, 2004.

[5] J Kim, JM Esposito, V Kumar, "An RRT-based algorithm for testing and validating multi-robot controllers," Robotics: Science and Systems Conference, MIT, Cambridge, MA, 2005.

[6] J Bruce, M Veloso. "Real-time Randomized Path Planning for Robot Navigation." Intelligent Robots and System, IEEE/RSJ International. 2002.

[7] S. LaValle, J. Kuffner. "Randomized Kinodynamic Planning."The International Journal of Robotics Research, Vol. 20, No. 5. 2001

[8] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for Rapid Replanning in Dynamic Environments," Proc. IEEE Int. Conf. Robotics and Automation 2007.

[9] A Atramentov and S. M. LaValle. "Efficient Nearest Neighbor Searching for Motion Planning", In Proc. IEEE International Conference on Robotics and Automation 2002.

[10] D. Ferguson, N. Kalra, and A. T. Stentz, "Replanning with RRTs," in Proc. IEEE Int'l Conf. on Robotics and Automation, May 2006.

[11] J.L. Bentley. "Multidimensional Binary Search Trees used for Associative Searching," Communications of the ACM. Vol. 18, No. 9. 1975.

TABLE I
No Obstacle Movement

Robot Dimension	Bias Type	Failed Samples	Number of Tree Nodes
2D	Unbiased	2108.15 +/- 992.80	370.19 +/- 143.02
	Online Bias	1210.56 +/- 669.78	359.63 +/- 132.60
	Offline Bias	626.58 +/- 297.19	400.72 +/- 112.27
3D	Unbiased	14092.20 +/- 7564.27	1616.76 +/- 1094.96
	Online Bias	7881.74 +/- 5569.61	1518.58 +/- 1165.13
	Offline Bias	3380.94 +/- 3962.04	670.40 +/- 837.31
4D	Unbiased	213228.00 +/- 143371.00	26570.90 +/- 19340.70
	Online Bias	124613.00 +/- 104745.00	22451.90 +/- 18468.80
	Offline Bias	6533.09 +/- 13604.90	2382.61 +/- 5253.68

TABLE II
Local Obstacle Movement

Robot Dimension	Bias Type	Failed Samples	Number of Tree Nodes
2D	Unbiased	1994.22 +/-770.57	363.88 +/-104.87
	Online Bias	1175.42 +/-635.42	367.10 +/-136.22
	Offline Bias	723.29 +/-517.72	429.45 +/-179.09
3D	Unbiased	14902.80 +/- 11758.10	1797.37 +/- 1665.56
	Online Bias	8797.85 +/- 9469.49	1721.38 +/- 1887.48
	Offline Bias	8069.06 +/- 15350.00	1946.59 +/- 4659.10
4D	Unbiased	143524.00 +/- 78069.40	17789.10 +/- 10726.10
	Online Bias	95954.40 +/- 72884.20	18076.90 +/- 13625.40
	Offline Bias	17533.20 +/- 42521.20	5969.78 +/- 14807.70

TABLE III
Global Obstacle Movement

Robot Dimension	Bias Type	Failed Samples	Number of Tree Nodes
2D	Unbiased	1411.91 +/- 1010.45	399.75 +/-189.20
	Online Bias	991.27 +/- 794.56	428.69 +/-219.47
	Offline Bias	1490.04 +/- 1082.12	659.73 +/-391.49
3D	Unbiased	6807.67 +/- 12517.40	1538.59 +/- 2649.90
	Online Bias	3354.70 +/- 7155.22	1141.96 +/- 1814.60
	Offline Bias	10089.80 +/- 17255.70	2636.25 +/- 4603.50
4D	Unbiased	22094.50 +/- 51533.20	5366.22 +/- 12807.20
	Online Bias	9924.75 +/- 28338.90	2923.89 +/- 7369.70
	Offline Bias	18513.60 +/- 47586.50	5675.25 +/- 14446.40