NEURAL NETWORK APPROACH TO FEATURE SENSITIVE MOTION PLANNING

An Undergraduate Research Scholars Thesis

by

JOSE ANDRES MEDINA VARGAS

Submitted to Honors and Undergraduate Research
Texas A&M University
in partial fulfillment of the requirements for the designation as

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Nancy Amato

May 2014

Major: Computer Engineering

TABLE OF CONTENTS

Page

ABSTRACT

Neural Network Approach to
Feature Sensitive Motion Planning. (May 2014)

Jose Andres Medina Vargas
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Nancy Amato
Department of Computer Science and Engineering

Motion planning (MP) is the problem of finding a valid path (e.g., collision free) from a start to a goal state for a movable object. MP is a complex problem with a myriad of applications, ranging from robotics, to computer-aided design, to computational biology. Sampling-based planning deals with MP's complexity by constructing a graph which approximates the planning space. Different sampling based planners have been developed to tackle specific scenarios, but none of these is best for every scenario, e.g., cluttered vs. free space vs narrow passage. Thus, adaptive methods were created to combine different samplers effectively to solve more complex and heterogeneous environments.

Adaptive methods have been proposed that learn the best sampler for the entire space or that partition the space into simple and discrete region types, which are suited for particular samplers. These methods do not solve the problem of environments containing multiple complex areas that are difficult to automatically partition. In this thesis, we propose an alternative approach using neural networks to create an adaptive method that does not require regions. We replace the concept of regions with a visibility distribution, how "free" a node is, allowing our method to work for a wider range of interesting problems. Experiments show significant improvement in speed compared to methods that attempt to use a single sampler for a complex environment.

ACKNOWLEDGMENTS

I would like to thank my faculty mentor Dr. Nancy M. Amato and my graduate adviser Jory Denny for guiding me and helping me through the completion of this thesis. Without their continuous support and help this past year I would have not been able to start, progress, and complete this thesis.

CHAPTER I

INTRODUCTION

Motion Planning (MP) is the problem of finding a valid (e.g., collision free) path for a movable object, referred to as a robot, from an initial state to a goal state. MP is used in multiple fields, including robotics, game design [10], virtual prototyping [8], and bioinformatics [2]. However, the complexity of Motion Planning is exponential in the degrees of freedom, or the number of independent parameters that represent the robot's state [23].

Sampling-based methods [14, 17] have been developed to overcome this exponential complexity, with one common solution being the Probabilistic Roadmap Method (PRM) [14]. PRM creates a graph, called a roadmap, representative of the problem space with the help of a given sampling technique. Many good sampling techniques have been developed, each with a different bias and meant for specific scenarios. For example, while a uniform random distribution works well in free regions, there are others that are better suited to cluttered regions [1], or narrow passages [11]. These samplers all have unique strengths, but none are able to solve every type of scenario efficiently [13]. Furthermore, in heterogeneous environments, i.e., a mix of cluttered, free, and/or narrow passages, no single sampler would work best for the whole environment [21]. In Figure I.1, one can observe a cluttered area in the top center, followed by narrow passages through the middle and a free space on the bottom. This seemingly simple environment cannot be solved efficiently by any single sampler. Adaptive methods were developed to tackle these challenges in MP by combining different sampling methods, and using the sampler best suited for each region.

Current adaptive methods have been limited to learning the "best" method for the entire environment [13] or dividing the map into simple regions [21] and mapping one sampler per region. By learning only one sampler for the entire environment, an assumption is made that the entire environment can be mapped effectively with the use of only a single sampler. In environments similar to Figure I.1, this assumption does not hold, thus making the adaptive

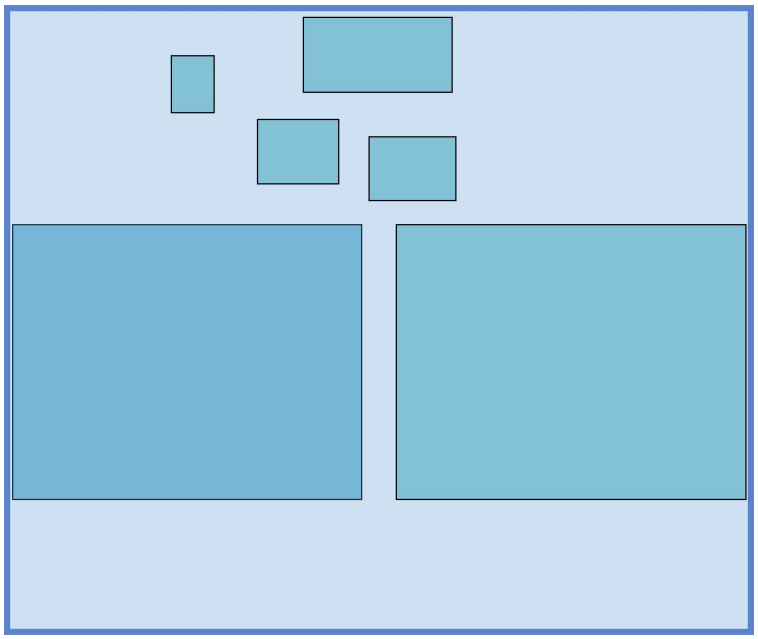


Fig. I.1. Sample environment with cluttered areas, free space regions and narrow passages

method ineffective. Other proposed methods partition the environment into regions. By assuming that every point in an environment can be easily classified as part of a region of a particular region type, these methods end up classifying a significant number of regions into a default non-homogeneous type in which a default sampler is used. Additionally, once the regions are created, another layer of difficulty appears in connecting those disjoint regions. This illustrates that for interesting environments, it is rather hard to define such regions.

Motivated by the above limitations, we propose a neural network approach to feature sensitive motion planning that uses the features of the environment to create a continuous distribution function of the visibility (e.g., connectivity) of each point across the environment. A neural network is an interconnected group of nodes, inspired by the human's nervous system [20]. Neural networks are a powerful tool in pattern recognition and feature extraction applications. By using this more advanced learning technique we can compute a distribution function that appropriately represents the visibility of the environment. Unlike previous attempts to adaptive MP, our approach will not separate the map into discrete

regions. Instead, we will distribute appropriate samplers throughout the environment based on a continuous visibility distribution, thus allowing for more complex environments where regions are not easily determined by alternative approaches. In summary, our contributions are:

- Neuron Feature Sensitive Motion Planning (NeuronFSMP) algorithm. A novel adaptive sampling strategy capable of assigning samplers to configurations, rather than regions, by learning the visibility distribution of the environment
- Experimental analysis shows that neural networks are capable to approximate the visibility distribution of an environment
- Show through experimental analysis that by using a learned visibility distribution one can adaptively combine sampling methods to achieve faster solving of a query, less collision detection calls, fewer nodes required to solve a query, and a higher node creation success rate

In Chapter II, we describe important MP concepts and relevant MP algorithms. In Chapter III, we show our proposed method and explain our algorithm and the training of the neural network. We give an example of how our algorithm would work in a simple environment to better to show how our algorithm works at each step. In Chapter IV, we show the results of our algorithm in different environments and compare it to other methods. In Chapter V, we discuss how our method contributes to the MP field and mention some future work to improve our method.

CHAPTER II

BACKGROUND

In this chapter, we will define relevant terms and concepts to MP as well as give a brief overview of several related adaptive MP algorithms. We present only the work which is most relevant to our method, but note that there are other MP work not mentioned in this paper which are tangentially related to our approach.

Preliminaries

A robot is a movable object whose position and orientation can be fully described through n parameters or *degrees of freedom* (DOFs), each corresponding to an unique factor affecting the robot's state (e.g., object positions, link angles, link displacements). Hence, a robot's placement, or configuration, can be uniquely described by a point $X = \langle x_1, x_2, ..., x_n \rangle$ in an n dimensional space (x_i being the ith DOF). This space, consisting of all robot configurations (feasible or not) is called *configuration space* (C-Space) [19]. The subset of all feasible configurations is the free space, C-free, while the union of the unfeasible configurations is the blocked space, C-obst. Thus, MP becomes the problem of finding a continuous trajectory in C-free connecting the start and goal configurations. Although it is intractable to compute the C-space [23], we can often determine the feasibility of a configuration quite efficiently, e.g., by testing for collision in the *workspace*, the robot's natural space.

Randomized planners exploit these feasibility tests to produce an approximate representation of the connectivity of C-free. One of these, the Probabilistic Roadmap Method [14], builds a roadmap (graph) in C-free. First, collision-free configurations are sampled and added to the roadmap. Then, a simple *local planner* (e.g., straight-line) attempts to connect neighboring nodes; of which only successful connections become edges of the roadmap. Once the roadmap has been built, the roadmap can be queried for a path between start and goal configurations by connecting them to the roadmap and using a simple graph search technique, e.g, A*,

to extract a path. A benefit of PRMs is that once the roadmap is built, you can query it multiple times without having to rebuild your roadmap. Uniformly Random PRMs have a low chance of yielding nodes in narrow passages [12]. This means that a lot of iterations are needed to successfully connect two components separated by a narrow passage due to the low probability of creating nodes in such passage.

Related Work

Many variants of PRM have been introduced that sample using different biases. Unlike uniform sampling, which samples the environment completely randomly, other techniques are developed to sample towards or away from obstacles. For example, Obstacle-based PRM (OBPRM) [1], Uniform OBPRM (UOBPRM) [27], and Gaussian PRM [3] generate samples near C-obst surfaces. OBPRM works by pushing samples toward C-obst surfaces. UOBPRM guarantees uniform node distribution around obstacles by sampling and analyzing random line segments for intersections with C-obst boundaries. Gaussian PRM generates pairs of samples that are a distance d apart according to a Gaussian distribution and if one and only one sample is in C-free it retains the free sample. There are also samplers designed for narrow passages, such as Medial Axis PRM (MAPRM) [26] [18] and Bridge Test PRM [11]. MAPRM works by pushing sampled configurations (free or not) to the medial axis of the free space. MAPRM theoretically guarantees higher density of sampling in narrow passages. Bridge Test PRM creates two samples that are a distance d apart (similar to Gaussian), and if they are both in collision, it checks the validity of their midpoint. If the middle configuration is valid, the midpoint is added to the roadmap. None of these methods are able to efficiently solve every scenario, thus adaptive methods have been introduced to combine these sampling methods.

Adaptive Methods

Hybrid PRM [13] uses reinforcement learning to find the most effective sampler for an entire environment or space. Simple methods (e.g., uniform sampling) initially produce high re-

wards. However, these rewards decrease as the environment becomes oversampled, at which point more complex samplers tend to be preferred as their samples become more important to discovering areas of the environment and merging portions of the map. The downside of Hybrid PRM is that the samples are done globally and thus it tends to converge into a single sampler for the whole environment instead of using the features of the map to use specific samplers in certain areas. Adaptive Rapidly-Exploring Random Trees [6] and the Adaptive Connection Strategy [7] apply this technique to learn the best RRT [17] method or connection strategy respectively.

The concept of workspace importance [15] has been introduced in many methods to exploit the geometric features of the workspace to guide sampling. The workspace is generally a good approximation of C-Space for in many robotic systems such as a simple robot traveling in a room. Thus, using information from the workspace to guide sampling in the C-Space helps as a good starting place for adaptive methods. This was applied to Hybrid PRM in [16] to extract workspace information from cell decompositions to define locations where samplers should be applied.

Feature Sensitive Motion Planning (FSMP) [21] uses machine learning in a divide-and-conquer approach to MP. The planning space is recursively subdivided into regions until the regions can be classified as appropriate for a specific planning strategy from a set of planners. This method allows for multiple samplers to be used in the regions where they are expected to be the most efficient. This method does not adaptively learn which sampler would be the most efficient in which region. Instead, a large and extensive study was done to map samplers to their defined regions. This long study would have to be repeated as new planners are developed instead of the method being able to adapt to the benefits of a new planner. Other subdivision strategies have been proposed in [22].

RESAMPL [24] uses local region information (e.g., entropy of neighboring samples) to decide how and where to sample, which samples to connect, and how to find paths through the

environment. This use of spatial information enables RESAMPL to increase or decrease sampling depending if the regions are narrow or free.

Unsupervised Adaptive Strategy (UAS) [25] combines FSMP and Hybrid PRM by applying Hybrid PRM to regions divided by FSMP. This method uses unsupervised learning to minimize user intervention typically required for manual training and parameter tuning. Although greatly reducing user intervention, UAS is still bounded to the use of discrete regions that cannot always be easily classified into a specific homogeneous type.

Information Theory Approaches. Burns and Brock [4, 5] used the ideas from information theory to guide a sampler towards a region where it is predicted to be useful. Through this guidance, the spatial constraints are explored and an approximate modeling of C-obst helps guide future sampling.

These methods have shown the potential of using adaptive methods for MP. Many of these methods still rely on dividing the environment into specified regions or adapting to one sampler. In the following section, we will explain our approach which does not require dividing an environment into homogeneous regions based on the features of that region.

CHAPTER III

METHODS

We propose a novel approach to adaptive sampling that removes the need of regions and instead assigns samplers based on the learned visibility of each configuration. By removing the regions, each node becomes the centric part of our learning algorithm rather than some region which may or may not be easily classified into a homogeneous class. In this chapter, we will further describe our method, the training of the neural network, and finally show a small example to explain the intuition of our algorithm.

Roadmap Construction

Algorithm 1 *NeuronFSMP*

Input: Environment env, Samplers S

1. NeuralNetwork $NN \leftarrow$ `TrainEnv`(env)
2. Roadmap $R = (V, E) \leftarrow (\emptyset, \emptyset)$
3. **repeat**
4. $\quad c \leftarrow$ `RandCfg`(env)
5. $\quad v \leftarrow NN$.`Predict`(c)
6. $\quad s \leftarrow$ `MapVisibility`(S, v) // Maps v to its respective sampler chosen from S
7. $\quad s$.`ApplySampler`(c)
8. $\quad$ **if** c is `valid` **then**
9. $\quad\quad R$.`AddAndConnect`(c)
10. **until** done

In our algorithm, outlined in Algorithm 1, we start by first training the neural network to our input environment. `TrainEnv`(), outlined in Algorithm 2, will return a Neural Network that has been trained to the features of our environment. From there, we begin to construct a roadmap by retrieving a random configuration and estimating its visibility using the trained neural network. `MapVisibility`() then returns a sample from the Sampler Set S which matches the configuration's visibility. The sampler will then be applied to the configuration and if the resulting configuration passes a validity test it will be added and connected to the

roadmap. The process of getting configurations and applying a sampler to it will be repeated until the problem is completed, e.g., the roadmap reaches a desired coverage or contains a maximum number of nodes.

Training the Neural Network

Algorithm 2 *TrainEnv*

Input: Environment env
Output: NN trained for Environment env

1. Roadmap $R = (V, E) \leftarrow$ `TrainingMap`(Env)
2. NeuralNetwork $NN \leftarrow$ `InitializeNetwork`()
3. **repeat**
4. $error^2 = 0$
5. **for each** $v \in V$ **do**
6. $predicted \leftarrow NN$.`Predict`(v) // Forward-pass to predict visibility
7. $actual \leftarrow$ `ComputeVisibility`(v)
8. $error^2 \leftarrow error^2 + (predicted - actual)^2$
9. NN.`Backpropagation`$(predicted - actual)$
10. NN.`UpdateWeights`()
11. **until** $error^2/||V|| < threshold$
12. **return** NN

Algorithm 2 trains a neural network to guess the visibility value at different locations throughout the input environment. The first step is to create a small training roadmap from the input environment by using a mixture of different samplers. The appropriate size for the of our training roadmap can be approximated by using the features of the workspace, e.g., obstacle vs. free workspace ratio. A larger map will represent the map better but it will also be more computationally expensive as you must do more collision detections, connections, and visibility approximations.

Create the Training Map

The first step to training the neural network is to create the training data, the training roadmap. To create the training map we use a mixture of uniform and obstacle-based sampling. We combine these two sampling methods to get a better approximation of our

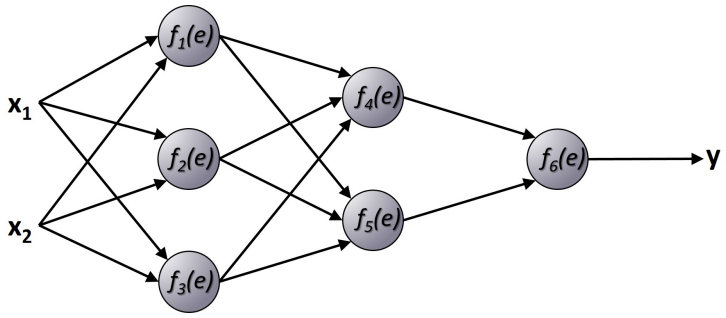


Fig. III.1. A feedforward neural network with 2 inputs, 3 layers and one output.

environment. We start by creating n uniform nodes and calculate the ratio of invalid and valid nodes, multiplied by a constant α:

$$p_{obst} = \alpha \frac{invalidUniformNodes}{totalUniformNodes} \tag{III.1}$$

where p_{obst} is the probability of using an obstacle-based sampler (e.g., OBPRM [1])

We show the effects of α to our visibility distribution in Chapter IV. Using p_{obst} we continue creating nodes using both uniform and obstacle-based samplers. Everytime we create uniform nodes we update p_{obst}. We stop when the desired size of our training map is reached. We connect the valid nodes and return our training map, including the invalid nodes.

Neural Networks

In our algorithm, we train a neural network to approximate the visibility of a configuration based on its features (e.g., DOF values). A neural network is a combination of artificial neurons connected in some way. Figure III.1 shows a *feedworward* neural network. A feedfoward neural network is divided into layers, where the neurons in each layer serve as an input to the next layer, thus sending the information from the first (input) to the last (output) layer.

Each input into the neuron has a weight associated with it, shown in Figure III.2. The following formula describes the summing junction of a neuron, in which the product of each

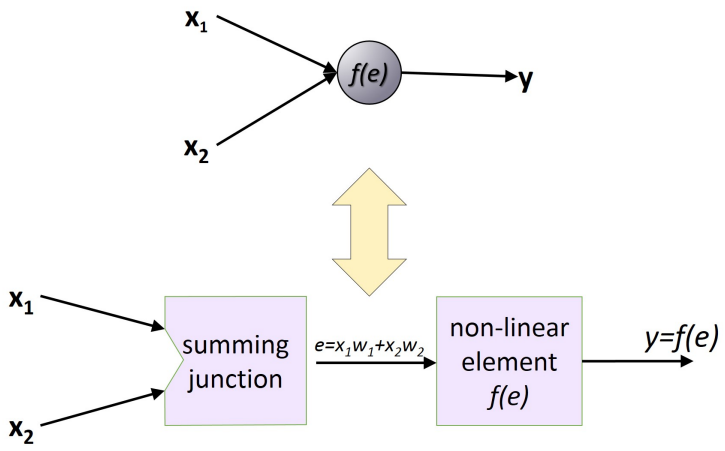


Fig. III.2. A single neuron with two inputs showing its input weights

input and its weight is summed together to create an activation value.

$$activation = \sum_{i=1}^{n} x_i w_i \tag{III.2}$$

where n is the number of inputs, x_i : is input i, and w_i is weight i.

The activation value gets subtracted with the threshold (also called a bias) and fed into a sigmoid function which will output a number between 0 and 1.

$$a = \sum_{i=1}^{n} x_i w_i + (-1)t \tag{III.3}$$

$$output = \frac{1}{1 + e^{-a/p}} \tag{III.4}$$

where t is a threshold value and p determines the curve of the function. A smaller p produces a function more similar to a step function.

In the beginning, we create a neural network with random weights and thresholds. The next step in Algorithm 2 (lines 3-11) will train the neural network by modifying the weights and

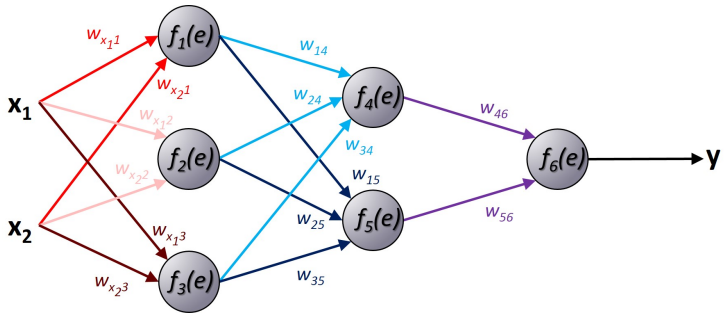


Fig. III.3. Forward propagating from input layer to output layer

thresholds of the neurons until the sum of the squares of all the errors between the predicted and computed visibilities is smaller than some given threshold.

Forward Propagation

We begin training the data by forward propagating the configuration parameters along the neural network which returns a predicted visibility using the current weights. At each step we sum the product of the weights and inputs and subtract a threshold value, Equation III.3 and feed it into the sigmoid function, Equation III.4.

The output of the sigmoid function above is then fed as an input to the next layer in the network. Forward propagating one piece of data with 2 parameters is shown in Figure III.3.

The result of feed forwarding the configuration is saved as a visibility prediction, which will be then compared to the "actual" visibility approximated in the next step.

Visibility Approximation

In order to train our neural network to correctly guess the visibility of configurations based on the configuration's parameters, we must train it on known examples. This means we need to compute the visibility of the nodes in our training roadmap and compare it to what the neural network predicts, and adjust our weights accordingly.

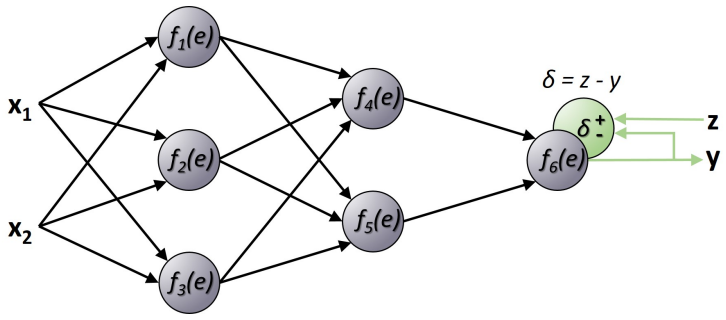


Fig. III.4. Error in output of neural network

For every node in our training roadmap, we attempt a connection to its k-nearest neighboring nodes using a simple local planner (e.g., straight line). The visibility then becomes the ratio between successful connection attempts and total connection attempts.

$$v = \begin{cases} 0 & \text{if not valid} \\ \frac{1+connSuccess}{1+connAttempts} & \text{if valid} \end{cases} \tag{III.5}$$

We use Equation III.5 instead of the simple ratio of successful and total connection nodes to differentiate between invalid nodes and valid nodes with no connections.

Back-propagation and Weight Updates

The next step in the training of our neural network is to back-propagate our error from the output layer to the input layer. Where the error is calculated as,

$$\delta = z - y \tag{III.6}$$

where δ is the error on the output neuron, z is the calculated visibility, and y is the visibility estimated by the neural network.

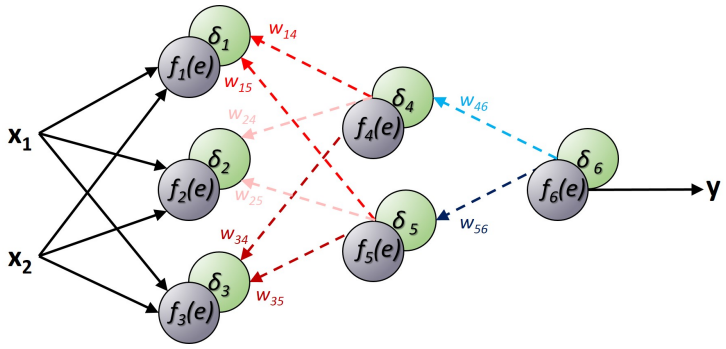


Fig. III.5. Back propagating δ_{out} to calculate the δ_i of each neuron i

Figure III.4 shows the first step in back-propagating the error, which is to calculate said error by subtracting the calculated visibility from the predicted visibility. The error then back-propagates itself in a similar way as forward-propagation except that the error propagates from the output to the input layer, Figure III.5.

$$\delta_i = \sum_{j=0}^{n} \delta_j w_{ij} \tag{III.7}$$

where δ_k is the error in neuron k and w_{ij} is the weight from neuron i to neuron j.

Once all the errors have been back-propagated properly, we update the weights of each neuron:

$$w'_{ij} = w_{ij} + \mu \delta_j \frac{\partial f_j(a)}{\partial a} output_i \tag{III.8}$$

where i is the input neuron, j is the output neuron, w_{ij} is the old weight, w'_{ij} is the new weight, and μ is the learning rate.

It is important to note the effect of μ to the learning speed. μ affects the learning speed by acting as a multiplier to the change in weight at each step. A common method to set μ is to start with a large value and decrease it as the neural network begins to converge. Back-propagation is shown in Figure III.6.

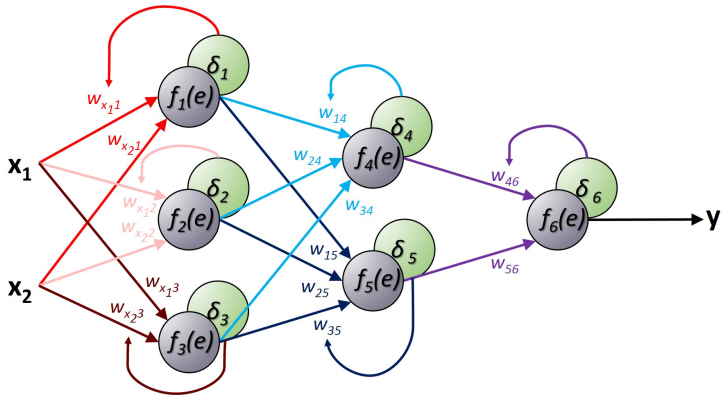


Fig. III.6. Updating the weights coming into neuron i using δ_i

Once the weights have been updated, a new example data is taken from the training roadmap and the process repeats itself until all the data has been used. Once a loop through the data is done, the error is calculated for each example configuration and if the mean squared error (MSE) is less than some given threshold, we allow the program to exit the loop. If the MSE is still too large, we go through all the data again, updating the weights again until the MSE is less than the threshold or a maximum number of iterations is reached.

Choosing a Sampler

With the neural network trained, we are able to predict the visibility of nodes outside our training roadmap by feeding in the features of the configuration (e.g., x,y,z,clearance). In the last step of our algorithm, we select a sampler from our list of samplers using the predicted visibility of a configuration. The visibility is predicted same as we did in the training section, by using forward propagation. Different ranges of visibilities are mapped to certain samplers, i.e., a high visibility implies that this configuration has a fairly large free-space around it, and therefore uniform sampling will work well on it.

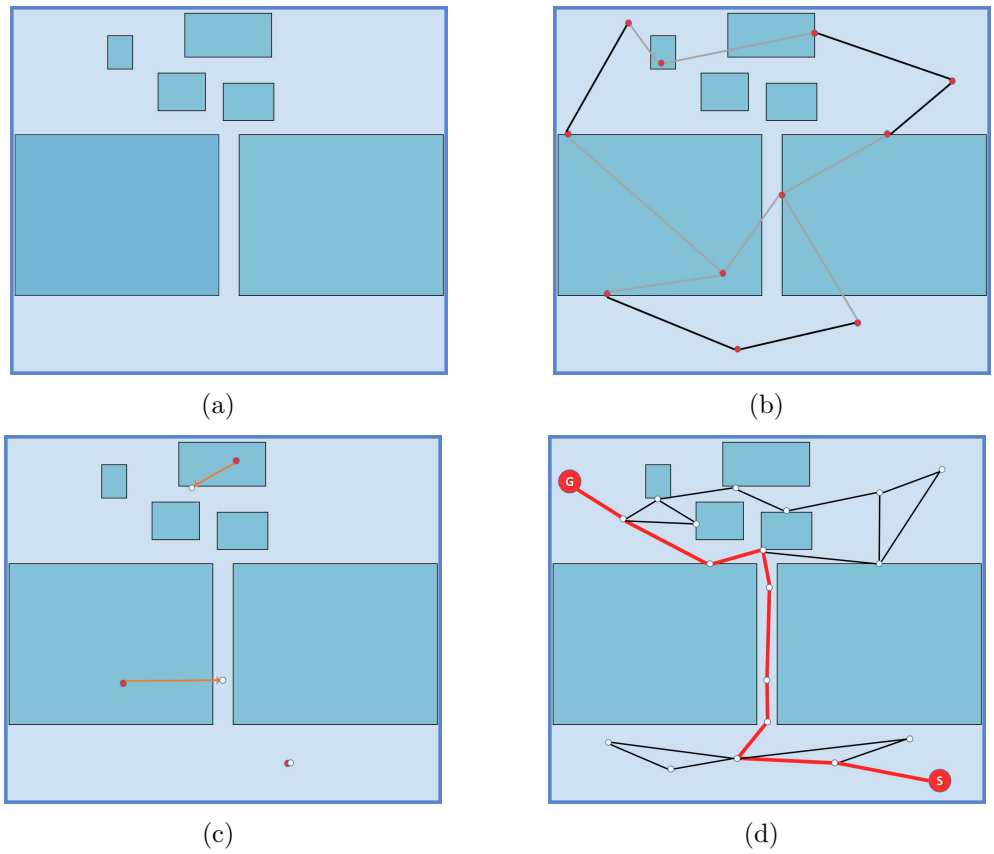


Fig. III.7. Example of our NeuralFSMP algorithm solving a simple environment. (a) Example environment. (b) Small training map. Successful and failed connections are shown in dark and light lines respectively. (c) Original configurations (shown in red) are used to guess the visibility and select a sampler, producing final configurations (shown in white). (d) Roadmap on example environment with start and goal configurations. Extracted path is shown using a red, thick line.

Example

In this section, we will show the intuition of how our algorithm creates a roadmap in a simple environment, Figure III.7(a). In this environment, we can observe a narrow passage connecting the top and bottom areas, and a cluttered area directly above the narrow passage. This simple environment (and a list of samplers) is passed to our main algorithm.

The first step in our algorithm is to train the neural network (Algorithm 2) using a small training roadmap. We calculate the visibility for each node in the training roadmap. In figure III.7(b) we see such a training roadmap as well as the attempted connections done for each node. We use these connections to calculate the approximated visibility of each node, feed the visibility to the neural network, and train it to predict the visibility of future configuration samples.

After a neural network has been trained, we begin the construction of our roadmap by randomly sampling configurations, assigning a sampler to it, and applying the rules of the chosen sampler to the configuration. Figure III.7(c) shows example configurations and how our algorithm will choose a sample depending on the approximated visibility of them. The configuration on the top of the map had a "cluttered" visibility, so our algorithm chooses OBPRM, thus creating a sample near the surface. The configuration in the center-left obstacle had a "narrow passage" visibility, so we choose Bridge sampling, thus creating a sample in the narrow passage. The sample at the bottom has a "free region" visibility which is mapped to Uniform Sampling which keeps the valid configuration.

The process of getting random configurations and selecting a sampler is continued until we reach an end condition, e.g., a query is solved. When our roadmap is queried with start and goal configurations, Figure III.7(d), they are added and connected to our roadmap and using a graph search technique, e.g., A*, we extract a path that solves the query.

CHAPTER IV

RESULTS

In this chapter, we will show through experimental analysis the ability of neural networks to represent the distribution of an environment.

Visibility Distribution using Neural Networks

In this section, we will show the effects of the neural networks hidden units (only 1 hidden layer used) on the learned visibility distribution of the environment in Figure IV.1. We will also show the effects of α, discussed in Chapter III. In table IV.1 you can see the visibility distribution as both of these values change.

All of these experiments were run using a training map of 100, and 1000 neural network learning iterations. These results show how the probability of using an obstacle-based sampler in the training roadmap helps learn a better distribution of the environment. If α is 0, the narrow passage and cluttered areas are not well represented, and if α is too large then it thinks of the cluttered area as a semi-open area as it was able to do many connections. Also a higher number of hidden neurons allows for more complexity in the learned distribution, but a number that is too high can also lead to over-fitting to the training data. These results

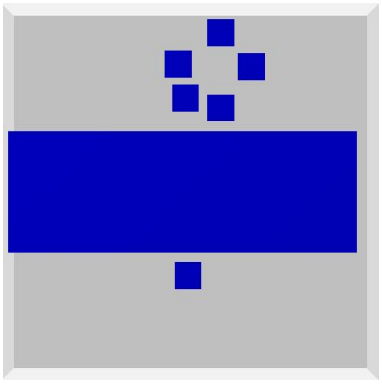


Fig. IV.1. Heterogeneous 2D environment

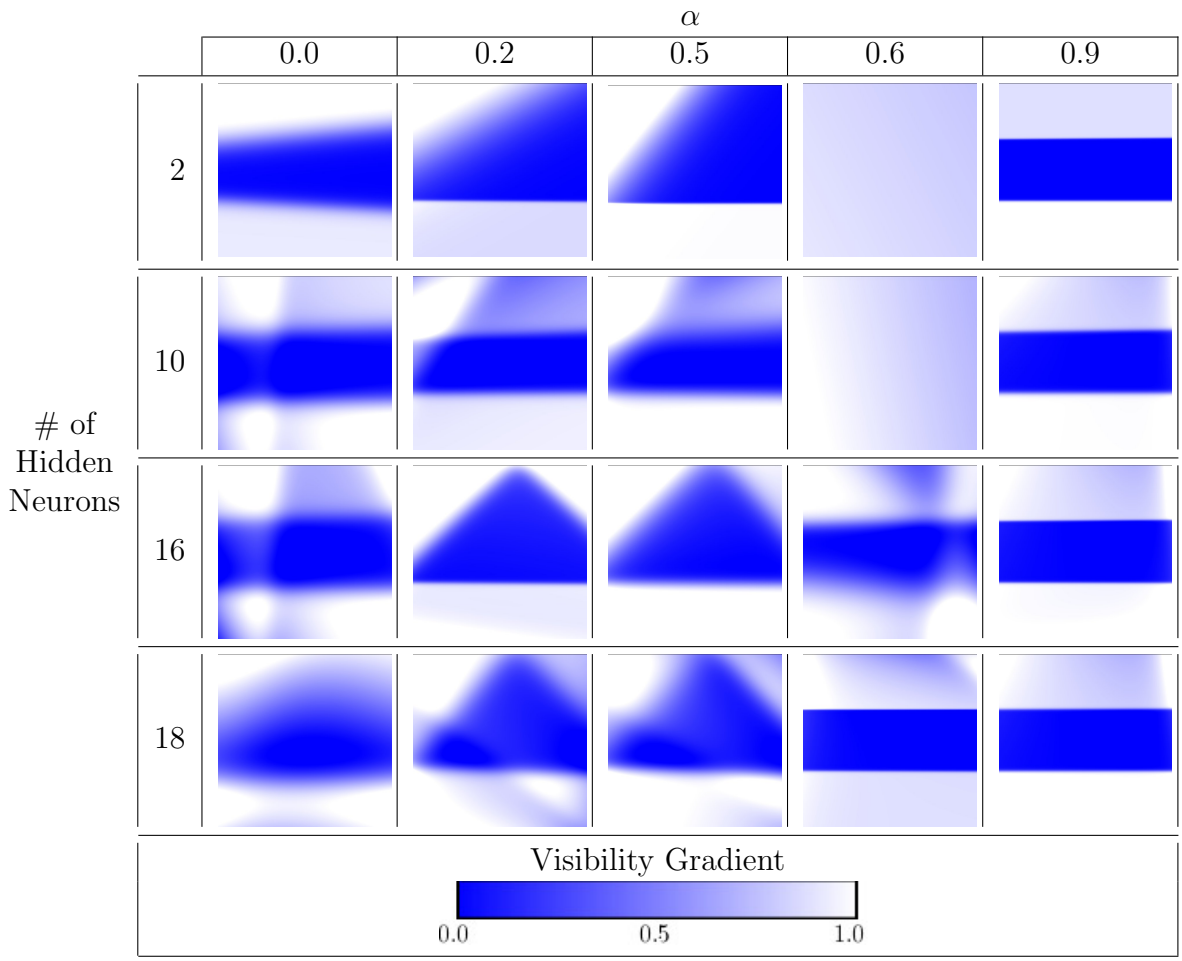


Table IV.1
Learned visibility distribution of environment in Figure IV.1

show that neural networks are able to appropriately learn the visibility distribution of an environment given a small training roadmap. I used these results to guide my experiments in the next section by having a good starting point for α and the number of hidden neurons.

Experimental Setup

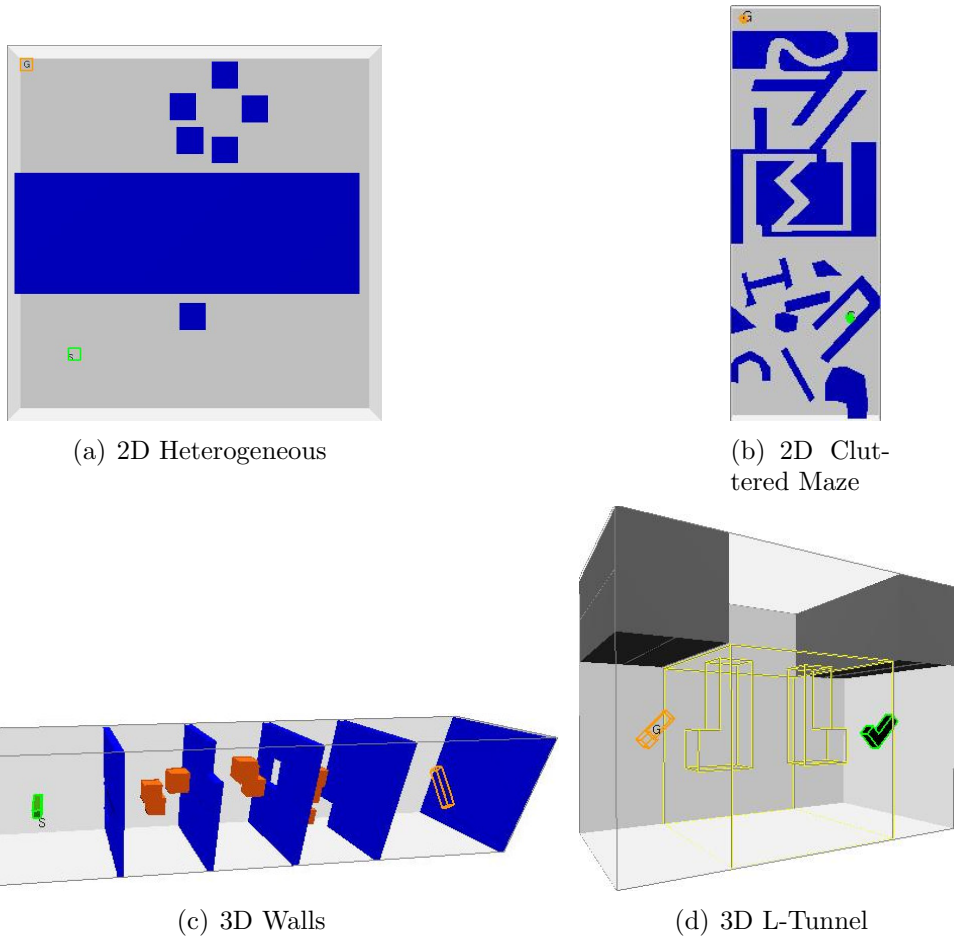
We implemented all the sampling methods using the C++ motion planning library developed by the Parasol Lab at Texas A&M University. The experiments are all conducted on an Intel Core 2 CPU 2.13GHz x 2 processor with 5.8 GiB of physical memory running Linux 3.9.10-

	2D Heterogeneous	**2D Maze**	**3D Walls**	**3D L-tunnel**
α	0.3	0.4	0.4	0.5
Hidden Neurons	$8n$	$6n$	$5n$	$20n$
Training map size	60	200	200	250
Initial Uniform nodes	20	30	50	50

Table IV.2
Environment-specific parameters for NeuronFSMP. n is the robot's number of degrees of freedom

100. RAPID [9] is used for collision detection computations. Connections are attempted between k "nearby" nodes according to some distance metric; here we use k = 7, C-space Euclidean distance, and a simple straight-line local planner. We map visibilities to samplers as follows: for high visibility ($v \geq 0.7$) we use uniform random sampling, elsewhere we use OBPRM. The neural network was allowed to run for up to 500 learning iterations. Figure IV.2 shows the environments used in the experiments (two 2D environments (2 DOFs) and two 3D environments (6 DOFs). Table IV.2 shows some parameters that were set-up differently in each environment. A lot of hidden neurons were used for the 3D L-tunnel because we were already aware of the complexity of the problem and were therefore willing to spend more time training to get a better approximation of the visibility distribution and spend less time building the final roadmap.

We use four different comparison metrics. *Time*, how long does each method take to solve the given query. The smaller the better as it implies the method is faster for the given environment. *Collision Detection (CD) calls*, how many times is the validity function called. Collision detection is usually considered the bottleneck in PRMs. The smaller the better as it implies it is less expensive than other methods when talking about collision detection calls. *Node creation success rate*, what is the probability of a created node to be valid and therefore kept in the roadmap. The higher the better as it means each node is more likely to be kept and therefore waste less time creating invalid nodes. *Nodes generated*, how many nodes did it need to create to solve the query. The smaller the better as it each node was more valuable to the total roadmap.



(a) 2D Heterogeneous

(b) 2D Cluttered Maze

(c) 3D Walls

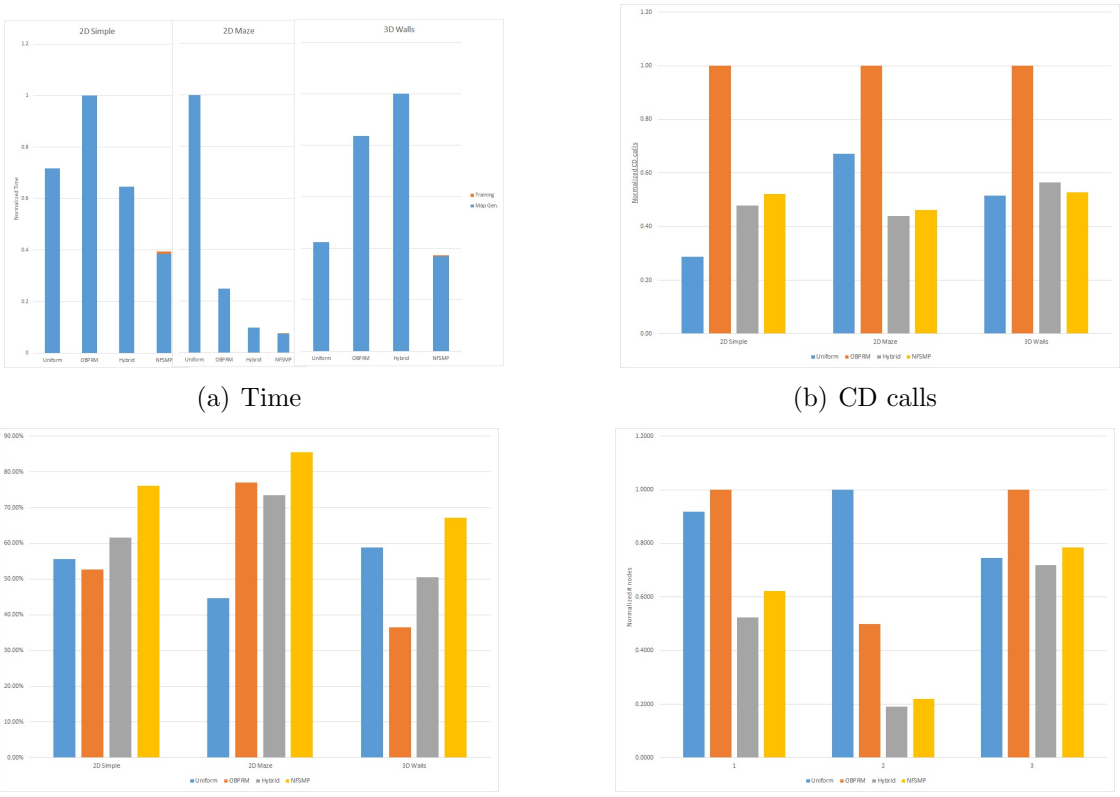
(d) 3D L-Tunnel

Fig. IV.2. Environments for experiments. Green and orange contours are start and goal respectively

Results

The results of the experiments outlined in the previous section are summarized in Figure IV.3 except for the 3D L-Tunnel environment as no method but NeuronFSMP was able to finish it within reasonable time. All of these experiments were run 10 times using different random seeds, and averages (after exclusion of outliers) were calculated.

NeuronFSMP performed faster than all the methods in all the environments. In the 2D cluttered maze, it did not significantly outperform HybridPRM as it is to be expected as it is a mostly homogeneous environment (all cluttered). Another important result is that



(a) Time

(b) CD calls

(c) Node Success Ratio

(d) Nodes generated

Fig. IV.3. Results of the 2D Simple, 2D Cluttered Maze, and 3D Walls environment.

using NeuronFSMP consistently resulted in a higher chance of creating a valid node. This is because we purposefully use samples in the regions we know the sample is likely to succeed and be useful (e.g., use uniform sampling around Cfgs with high visibility).

In the 3D L-tunnel environment, NeuronFSMP took an average of 2.75 hrs to complete and no run took longer than 4.82 hrs. OBPRM was given over 12 hours without finishing. HybridPRM could not complete the environment due to the strictly increasing method of calculating the weights and reaching a float point error. This is a hard problem, thus making the time the weights are increasing for longer than for other problems. The weights eventually reached a number the C++ implementation of the algorithm could not hold, thus crashing the program. This limitation on HybridPRM makes it unusable in this type of environment

without some kind of variable that resets the weights when they get too big or cap the weight at some large value.

CHAPTER V

CONCLUSION AND FUTURE WORK

From the results in the first section of Chapter IV, we can see that neural networks are a good tool to learn the visibility distribution of an environment. By learning the visibility distribution of the environment, the neural network implicitly learn an approximation of the C-Space (which is exponentially hard to compute explicitly). In the future, we would like to run more experiments to see the distribution on different environments, and how the number of hidden neurons and α affect the distribution on different environments with different obstacle/free space ratio. The method used to show the learned visibility distribution works well for 2-dof environments but it would get more difficult as the number of DOF increase. In the future we would like to experiment with other methods of showing the learned distribution that may work for more complex problems.

Our current implementation requires the visibility ranges that are mapped to each Sampler to be an input to the algorithm. This limits the adaptability of the algorithm as the user must already be knowledgable on what sampling method may work best for what visibility numbers on some specific environment. In the future, we would like to implement an adaptive method to learn the appropriate sampling methods for each visibility interval. We would also like to implement more complex neural network structures to learn more complex distributions and compare the results to see the benefits of potentially increasing the training cost for a better approximation of the visibility distribution.

In conclusion, we have shown that neural networks are a good tool to approximate the visibility distribution of an environment, which implicitly also learns an approximation of the C-Space. Using this method we have introduced NeuronFSMP, a novel variant to feature sensitive motion planning, capable of assigning different samplers around different configurations based on the learned visibility of that configuration. It is important to note that once the visibility distribution is learned, it can be applied to multiple parts of the PRM, not only

sampling. Similarly to how HybridPRM was applied to connectivity, you can apply different connection methods (including component connection methods) at different visibility ranges. More experiments are needed to prove how far can we take the idea of a learned visibility distribution but our preliminary results are promising and we plan to continue researching on this topic.

REFERENCES

[1] 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 on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[2] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. *J. Comput. Biol.*, 9(2):149–168, 2002. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2001.

[3] 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.

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

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

[6] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato. Adapting RRT growth for heterogeneous environments. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 1772–1778, Tokyo, Japan, November 2013.

[7] 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, November 2013.

[8] M. Garber and M. C. Lin. Constraint-based motion planning for virtual prototyping. In *Proc. ACM Symp. Solid Phys. Modeling*, pages 257–264, Saarbrücken, Germany, 2002. poster session.

[9] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. Technical Report TR96-013, University of N. Carolina, Chapel Hill, CA, 1996.

[10] J. P. Hespanha, H. J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. In *Proceedings of the IEEE Conference on Decision and Control*, pages 2432–2437, 1999.

[11] 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.

[12] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robot. Res.*, 25:627–643, July 2006.

[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] 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.

[15] H. Kurniawati and D. Hsu. Workspace importance sampling for probabilistic roadmap planning. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, volume 2, pages 1618 – 1623 vol.2, sept.-2 oct. 2004.

[16] H. Kurniawati and D. Hsu. Workspace-based connectivity oracle - an adaptive sampling strategy for prm planning. In *Algorithmic Foundation of Robotics VII*, pages 35–51. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[17] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. J. Robot. Res.*, 20(5):378–400, May 2001.

[18] J.-M. Lien, S. Thomas, and N. Amato. A general framework for sampling on the medial axis of the free space. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 4439 – 4444, sept. 2003.

[19] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[20] J. Mao and A. Jain. Artificial neural networks for feature extraction and multivariate data projection. *Neural Networks, IEEE Transactions on*, 6(2):296–317, 1995.

[21] 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*, pages 361–376. Springer, Berlin/Heidelberg, 2005. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Utrecht/Zeist, The Netherlands, 2004.

[22] M. A. Morales A., L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3114–3119, April 2005.

[23] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[24] 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.

[25] L. Tapia, S. Thomas, B. Boyd, and N. M. Amato. An unsupervised adaptive strategy for constructing probabilistic roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4037–4044, May 2009.

[26] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1024–1031, 1999.

[27] 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.