DLPRM: Deep Learning based Narrow Passageway Sampler

Adi Album

Introduction:

Sampling-based methods are a fundamental class of algorithms in the field of robotic motion planning. PRM, RRT, and many variants of these cornerstone algorithms are the basis of many algorithms in the field. These algorithms are classified as "Sampling-based", I.e., methods where landmarks are randomly sampled and are used in the attempt of achieving a valid path from start to target.

This class of algorithms has many advantages: Under reasonable conditions this class of algorithms is *probabilistically complete*, they don't require explicitly building the C-space, and can easily be applied to robots with many degrees-of-freedom.

Unfortunately, these algorithms also have many scenes in which they are quite impractical. For example, scenes containing narrow passageways.

Narrow Passageways – The Challenge:

Let $\mathcal{C} \subseteq [0,1]^d$ be the $\mathcal{C}$-space in Euclidean space $\mathbb{R}^d$, and $F \subset \mathcal{C}$ be the free-space. Denote by $s, t \in F$ the start and target configurations and let $\Gamma \neq \emptyset$ be the set of all paths from s to t contained in the free space F.

Let $A \subseteq F$ be a Lebesgue-measurable subset of the free space. We say that A is *critical* if any sampling-based method must sample some point $a \in A$ to construct a valid path from s to t.

As A's volume decreases, the probability of finding a valid path from s to t decreases. Given a sampling-based method that samples N landmarks uniformly at random $\{x_1, \dots, x_N\} \subseteq F$:

$$\Pr(Success) \leq \Pr\left(\bigcup_{i=1}^{N}\{x_i \in A\}\right) \leq \sum_{i=1}^{N} \Pr(x_i \in A) = N \cdot \frac{\mathrm{V(A)}}{V(F)} \xrightarrow[V(A)\to 0]{} 0$$

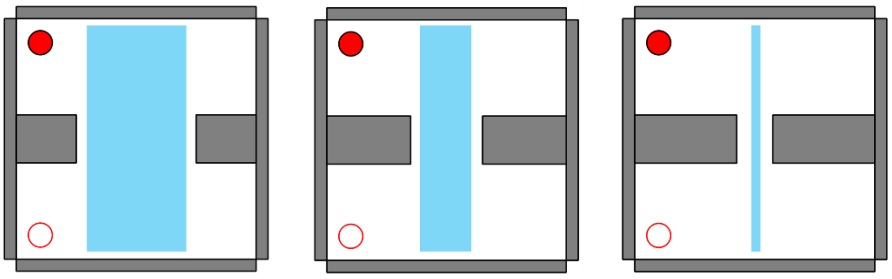


Figure 1: Scenes with critical configuration spaces (in blue) decreasing by volume

Narrow Passageways – Opportunities:

This field has been extensively researched with many methods[1] [2] attempting to solve this challenge. These methods are based on a geometric understanding of the scene, and present different heuristics for increasing the probability of sampling points near narrow passageways. We would like to propose a different method, one based on deep learning.

Motivation:

Using CGAL's simulator, it is clear for us humans where is the critical passageway simply by *looking* at a scene. Could a machine learning based algorithm be trained to meet this task? Many computer vision tasks have made huge breakthroughs using deep learning and particularly the architecture of convolutional neural networks. Using supervised learning techniques, I.e., training neural networks on large input-output training data.

So, the question holds, can a CNN be trained to aid a sampling-based method on the task of finding the critical passageway? In this project, we present DLPRM, an algorithm which shows a proof of concept that this method has potential.

Our project:

Data generation:

Machine learning based algorithm, and in particular deep learning models require large input-output pairs for training. We heavily rely on CGAL's simulator in the processes of generating such data.

1. JSON -> PNG:
   We implemented a simple script which reads as input the JSON file describing the generated scene and outputs a black-and-white image describing the scene's obstacles and free-space.

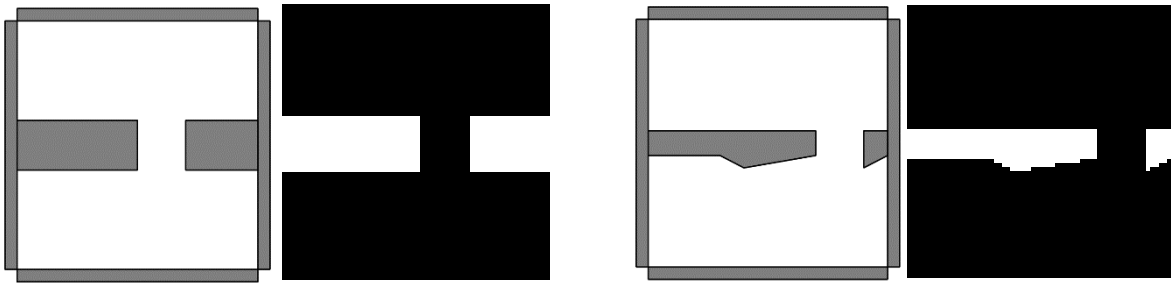


Figure 2: Scene (JSON) and image (PNG) pairs

2. Generating input-output scenes:
   We first generated a base scene describing the scene's boundaries, this can be found under *input_json_obstacles/base.json*. Describing a simple rectangular room with height and width both equal to 20 (in CGAL units)

[1] S. Wilmarth, N. Amato, and P. Stiller. MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space

[2] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners

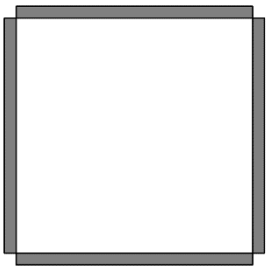
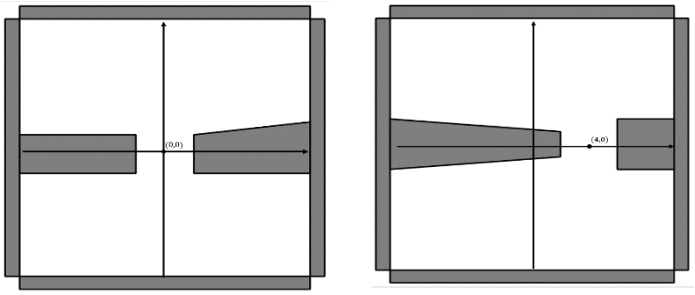


Figure 3: Base scene

Every generated scene will be based on this scene.
Next, we created 10 subdirectories under *input_json_obstacles/* : *0_0/ … 9_0/*. Inside each subdirectory we generated, using CGAL's Scene Designer, a few dozens of scenes each with a narrow passageway centered at $(i, 0)$ in CGAL coordinates for $i \in \{0,1, \dots ,9\}$.



For each such scene, the rectangle describing its narrow passageway is easily maintained.
Unfortunately, a dataset containing a few hundred labeled samples is far from enough for training deep learning models. Next, we used many data augmentation techniques dramatically enriching our dataset.

3. Data augmentations:
   3.1. Uniformly at random translating the scene along the y-axis:

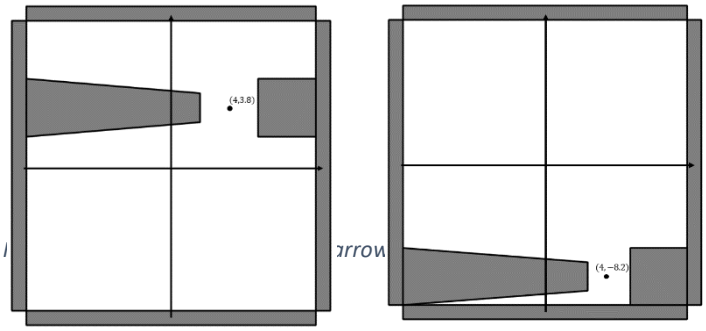


Figure 5: Scenes randomly translated along the y-axis

   3.2. Uniformly at random rotating the scene at $\{0, 90°, 180°, 270°\}$ centered at $(0,0)$:

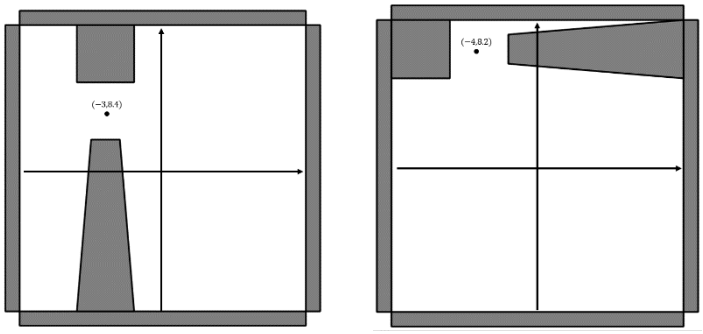


Figure 6: Scenes randomly rotated

3.3. Randomly generating additional obstacles:

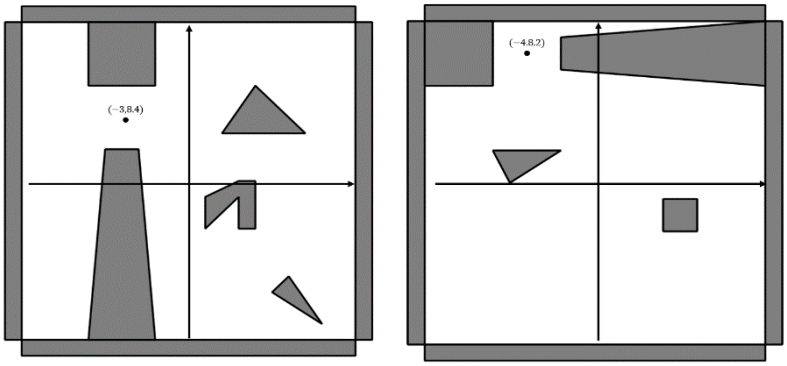


Figure 7: Scenes with randomly generated obstacles

For each scene we sample a random number of obstacles generated by an exponential distribution with a parameter $\lambda = 0.33$. By doing this we promote scenes with less obstacles with allowing occasional scenes with more obstacles. Each such obstacle is randomly translated and rotated and placed in the scene.

We perform these augmentations while carefully maintaining the rectangle defining the scene's narrow passageways:

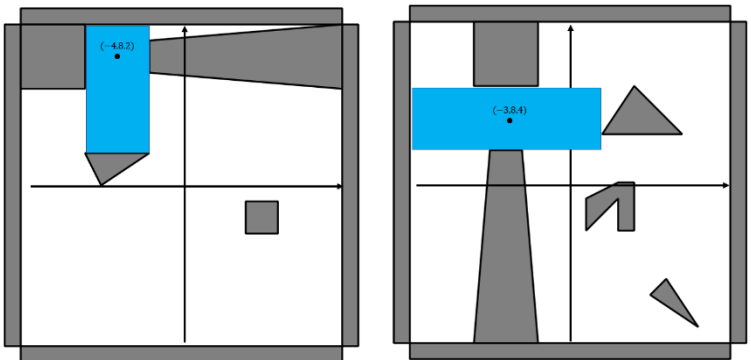


Figure 8: Scenes with narrow passageways in blue

<u>Model selection:</u>

As mentioned above, convolutional neural networks have achieved SOTA results in many image recognition tasks. For this reason, we used a deep learning model with a standard architecture of a convolutional neural network. The architecture can be found in *net.py*. The network inputs a 64×64 grayscale image and outputs a vector of size 4 uniquely determining the bounding box describing the narrow passageway.

The model is trained as a regression task, with a batch size of 32, the standard Adam optimization technique and the Mean Squared Error loss function. For more details, please view *train.py*.

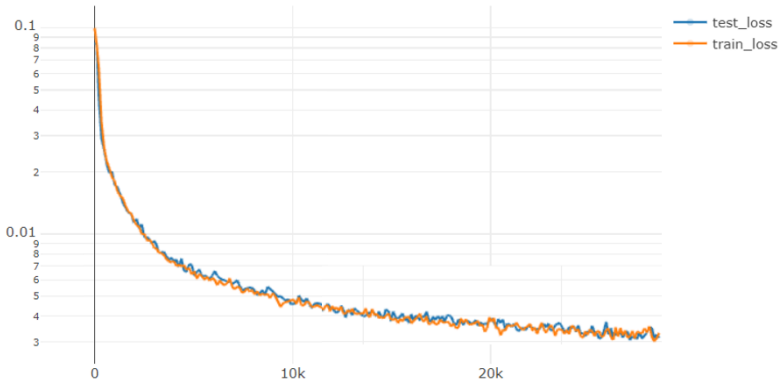


Figure 9: Model's train and validation loss

Results:

The best model obtained a MSE train loss of 0.0033 and a MSE test loss of 0.0030 on a test set of size $1{,}500$ scenes and respective narrow passageways.

This model was integrated in two PRM based solutions: *DiscoPygal/mrmp/solvers/dlprm_disc.py*, and *DiscoPygal/rod/solvers/dlprm_rod.py*.

We'll present some results in this section, and some further results and analysis in the appendix. In this section we shall focus on the landmarks our algorithm samples, compared with a naïve sampling method. In the appendix, we shall give further analysis comparing our PRM implementations for a single disc translating in the plane. We also implemented motion planning DLPRM algorithms for multiple discs translating in the plane, and for a single rod rotating and translating in the plane.

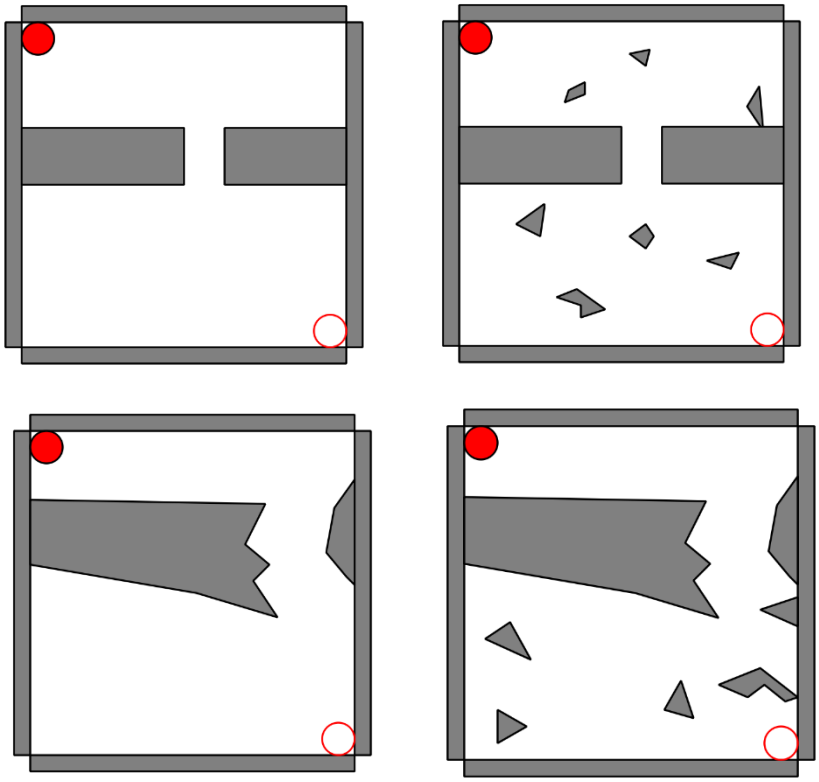


Figure 10: Scene #3 & Scene #4

Scenes #1 & #2 represent scenes with a very narrow passageway, as can be seen in the following figure. Scene #1 consists only of a narrow passageway, where scene #2 contains a similar narrow passageway with additional obstacles.

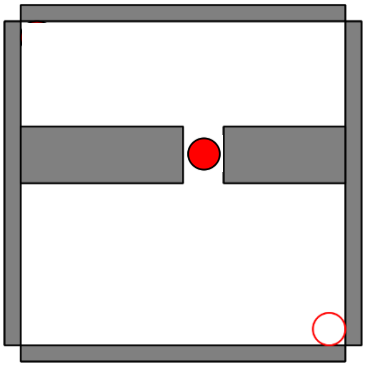


Figure 11: Scene #1 with the disc in the narrow passageway

Scenes #3 & #4 also contain narrow passageways, but this time narrow passageways with non-trivial shapes (i.e., non-rectangular).

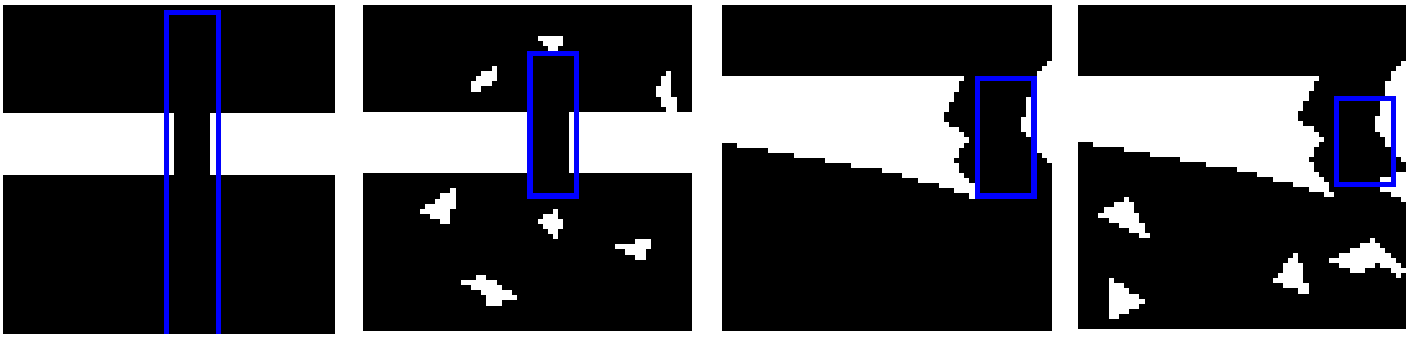


Figure 12: In blue - The bounding box describing the narrow passageway. As given by the CNN

As can be seen in the above figure, the blue rectangles, as obtained as the algorithm's output on these scenes, quite accurately describes the narrow passageway. The next figure shows the difference between using a naïve sampling method and using ours – Based on the above passageways:

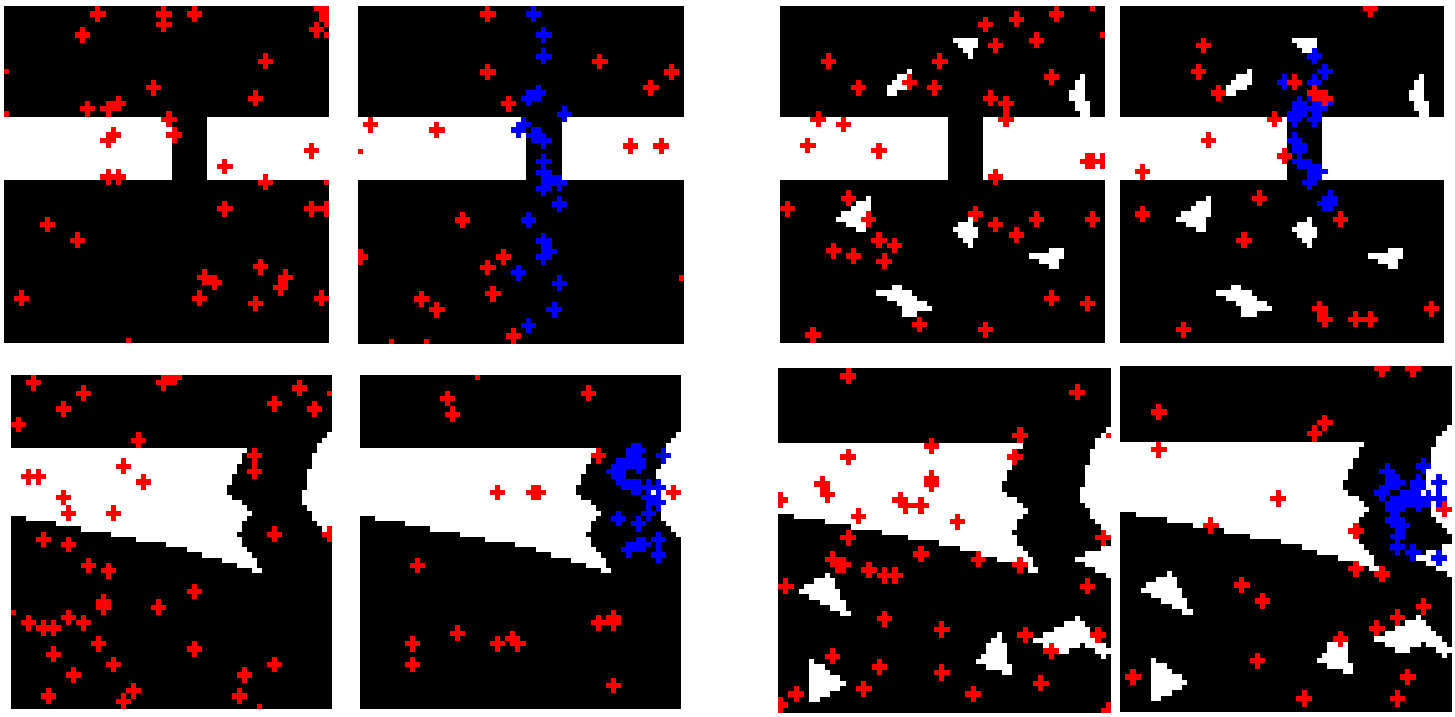


Figure 13:Red: uniformly sampled. Blue: ours

The above figure shows 4 pairs, the figures to the left show uniform sampling of 50 landmarks, yielding with very few (if any) landmarks in the narrow critical passageway. On the right we also have 50 landmarks, of which half were sample from the passageway given by our algorithm, and half are sampled uniformly by random.

Our method dramatically enhances the probability of sampling landmarks in narrow passageways, and by that increasing the probability of finding valid paths from start to target. Further analysis and results will be given in the appendix.

Appendix:

1. DLPRM (our method) vs Naive PRM – Single unit disc:

We'll use the 2 of the scenes presented in the main section of this project:

Scene #1:

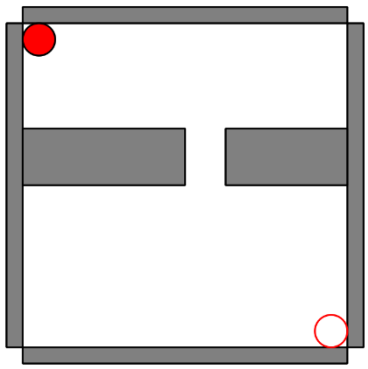
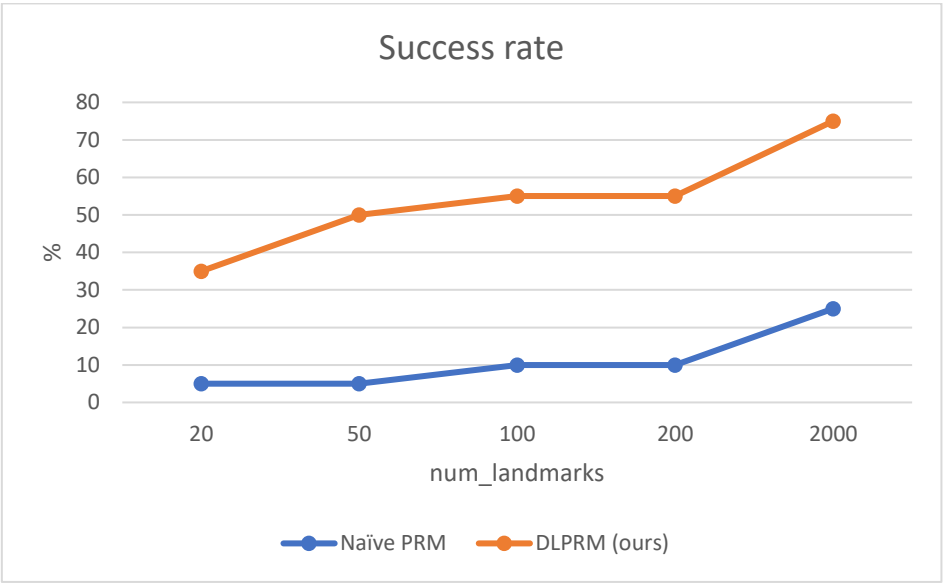


Figure 14: Scene #1

A 'clean' scene, with a single and very narrow passageway.

num_landmarks		20	50	100	200	2000
Naïve PRM	Success rate	5%	5%	10%	10%	25%
	Running time	0.04s	0.08s	0.15s	0.28s	2.62s
DLPRM (Ours)	Success rate	35%	50%	55%	55%	75%
	Running time	0.16s	0.22s	0.29s	0.46s	3.3s



Scene #2:

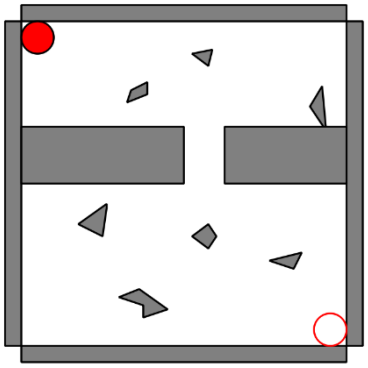
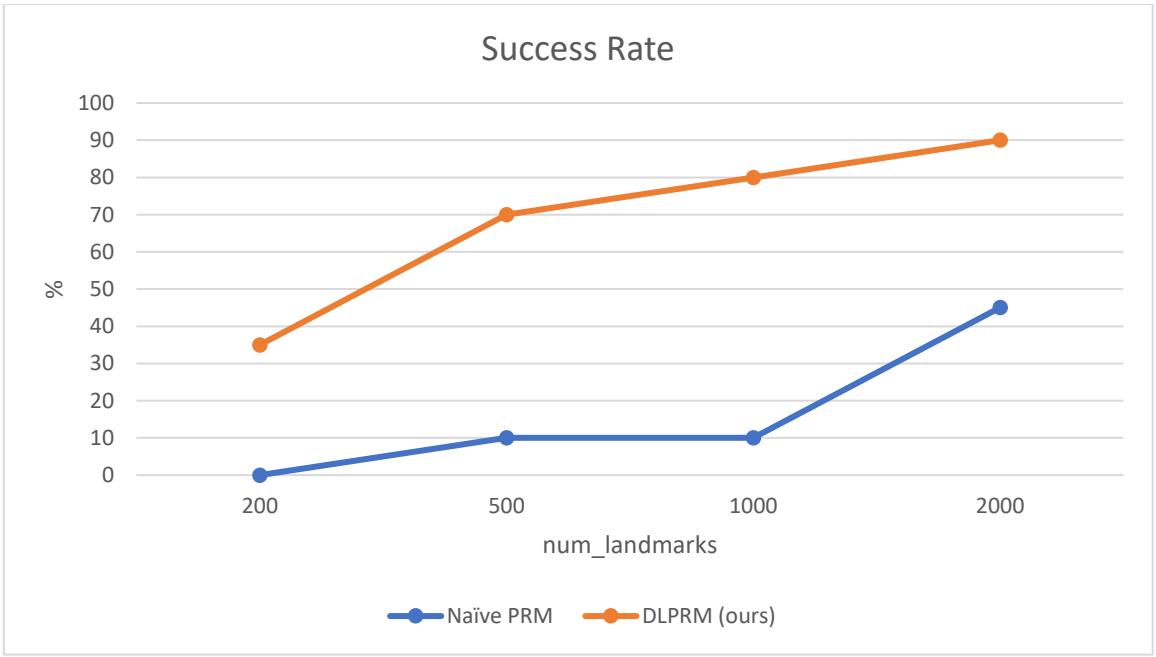


Figure 15: Scene #2

The same narrow passageway as in the scene above, with additional obstacles.

num_landmarks		200	500	1000	2000
Naïve PRM	Success rate	0%	10%	10%	45%
	Running time	0.35s	0.77s	1.61s	3.12s
DLPRM (Ours)	Success rate	35%	70%	80%	90%
	Running time	0.53s	1.04s	2.01s	4.05s



2. Project description:

Github repository:
https://github.com/AdiAlbum1/robotic-motion-planning-final-project

2.1. Easy install:

```
git clone https://github.com/AdiAlbum1/robotic-motion-planning-final-project
cd robotic-motion-planning-final-project
pip install pipenv
pipenv install
pipenv shell
```

2.2. How to run:

- `python generate_base_obstacle_images.py` : Reads as input the JSON files from `input_json_obstacles/` contatining the scene's description, and generates binary images of the scene's obstacles
- `python train.py` : Trains a Deep Neural Network for the above regression task
- `DiscoPygal/mrmp/solvers/dlprm_discs.py` and `DiscoPygal/rod/solvers/dlprm_rod.py` : Our product. An improved PRM implementation where points are sampled using the narrow passageway outputed by the CNN. These are to be used by running CGAL's simulators `python DiscoPygal/mrmp/mrmp_main.py` and `python DiscoPygal/rod/rod_main.py` respectiverly and loading the solvers.

2.3. Using your own trained model:

You can train your own model by using *train.py*. If you wish to deploy your trained model to DLPRM, you must change the parameter *network_path* in *params.py* which indicates the path to the network.
By default, we'll use the model achieved by the training procedure described above in *models/current_best/test_model.pt*.

2.4. Creating your own scenes:

For creating scenes for disc robots translating in the plane use:
python DiscoPygal/scene_designer/scene_designer.py
For creating scenes for a rod robot rotating and translating in the plane use:
python DiscoPygal/scene_designer_rod/scene_deisgner_rod.py

Load *input_json_obstacles/base.json* as the base for your scene. Next, add obstacles creating a scene with a single narrow passageway. This scene can be used in testing our model.