Trajectory Prediction by Generative Models for Optimization Based Planners

Thibault Barbié

Kyutech

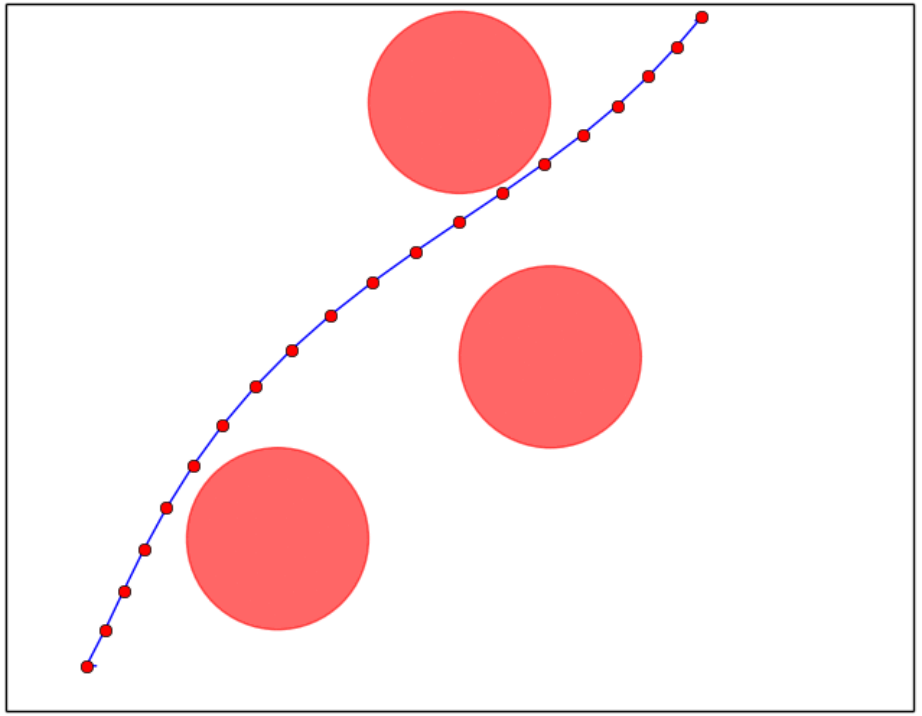
Kyushu Institute of Technology

Optimization Based Planners (OBP)

Optimization of a given initial trajectory. It is a **prior** guess of a valid solution.

Problem: most OBP use a naive **straight-line** through configuration space as a prior.

They do not use previous **knowledge**.

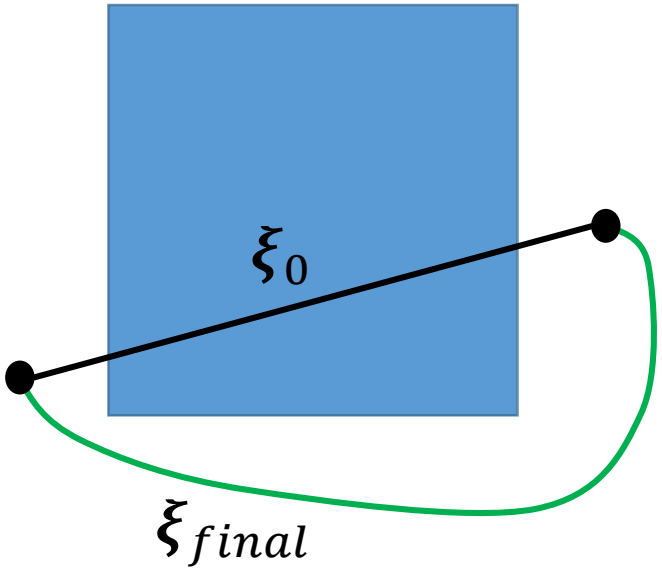


Optimization based planner

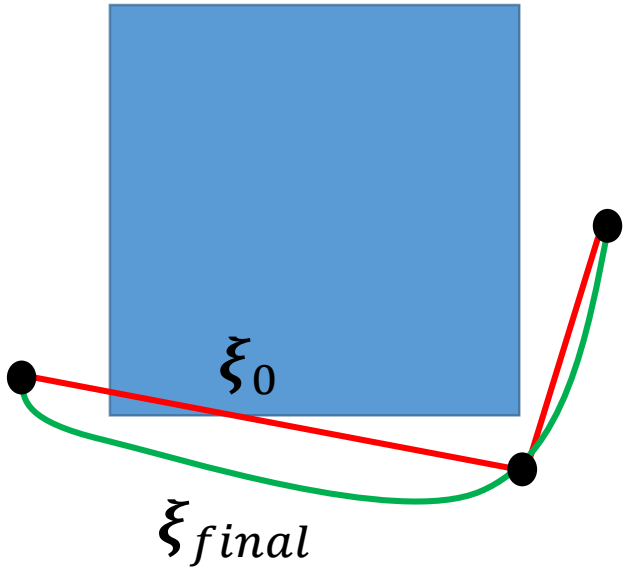
Trajectory Prediction

Goal:

using a motion dataset to give an initial trajectory $\boldsymbol{\xi_0}$ for a new planning problem $\boldsymbol{x}$



Classical linear initialization



Predicted trajectory

N. Jetchev and M. Toussaint, “Trajectory prediction: learning to map situations to robot trajectories”, in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 449-456.

Motion dataset

Planning problem:

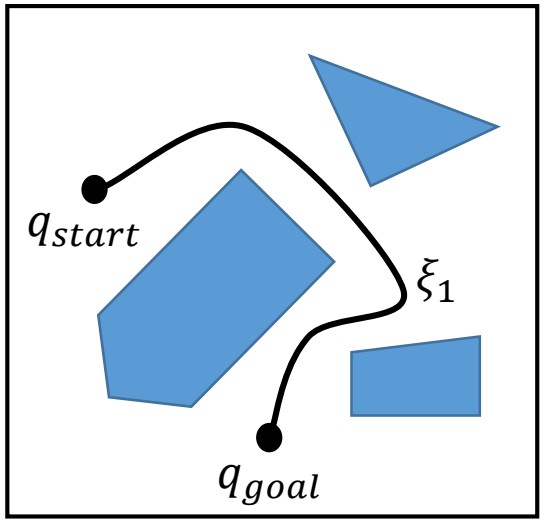
$$\boldsymbol{x} = [\boldsymbol{q}_{start}, \boldsymbol{q}_{goal}, \boldsymbol{obstacle}_1, \dots, \boldsymbol{obstacle}_K] \in X$$

Trajectory:

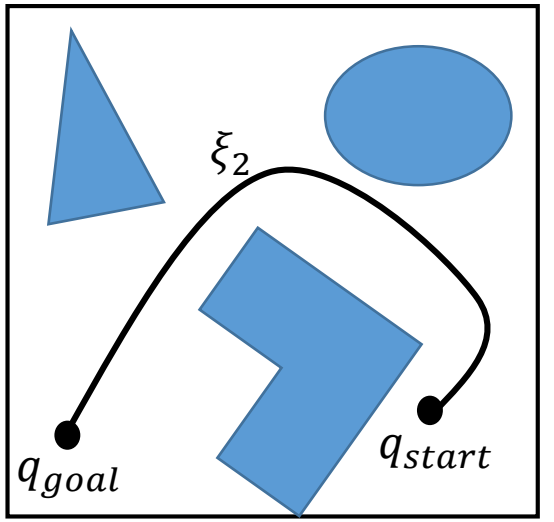
$$\boldsymbol{\xi} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_N] \in \Xi$$

$$\mathcal{D}$$

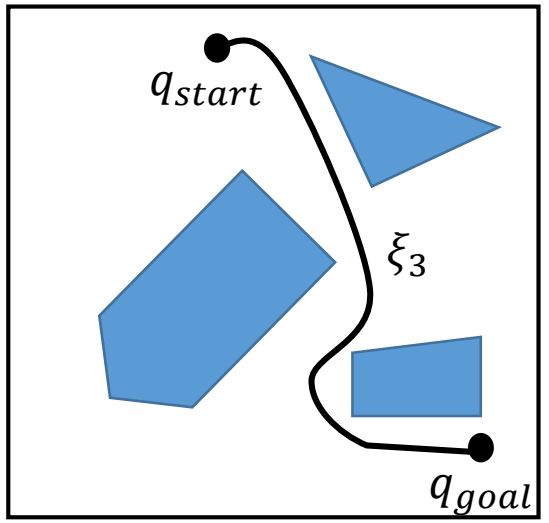
$$\boldsymbol{d}_1 = (\boldsymbol{x}_1, \boldsymbol{\xi}_1)$$
$$\boldsymbol{d}_2 = (\boldsymbol{x}_2, \boldsymbol{\xi}_2)$$
$$\vdots$$
$$\boldsymbol{d}_N = (\boldsymbol{x}_N, \boldsymbol{\xi}_N)$$



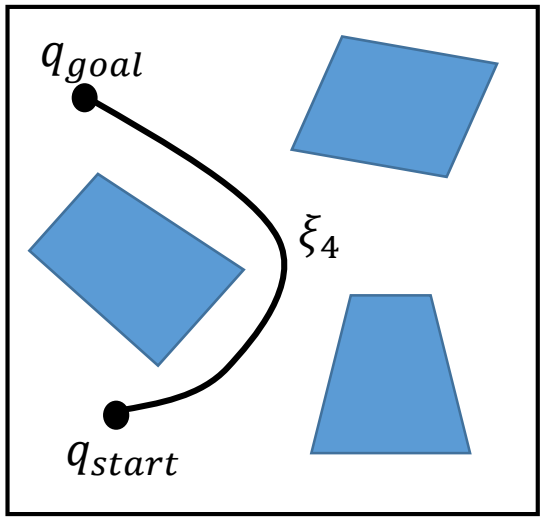
x_1



x_2

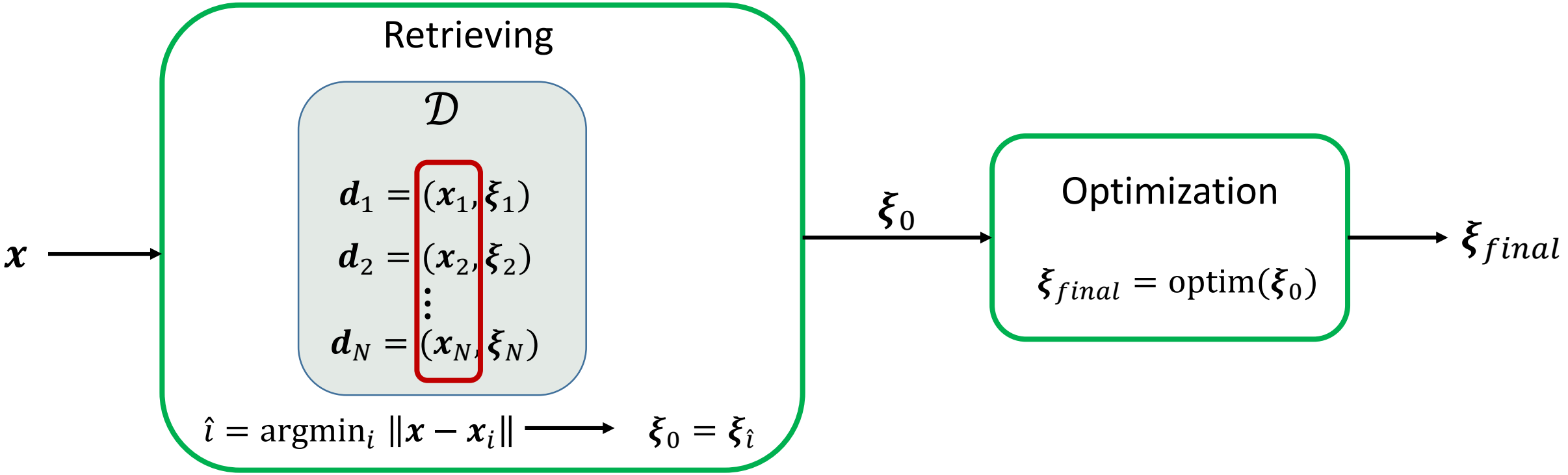


x_3



x_4

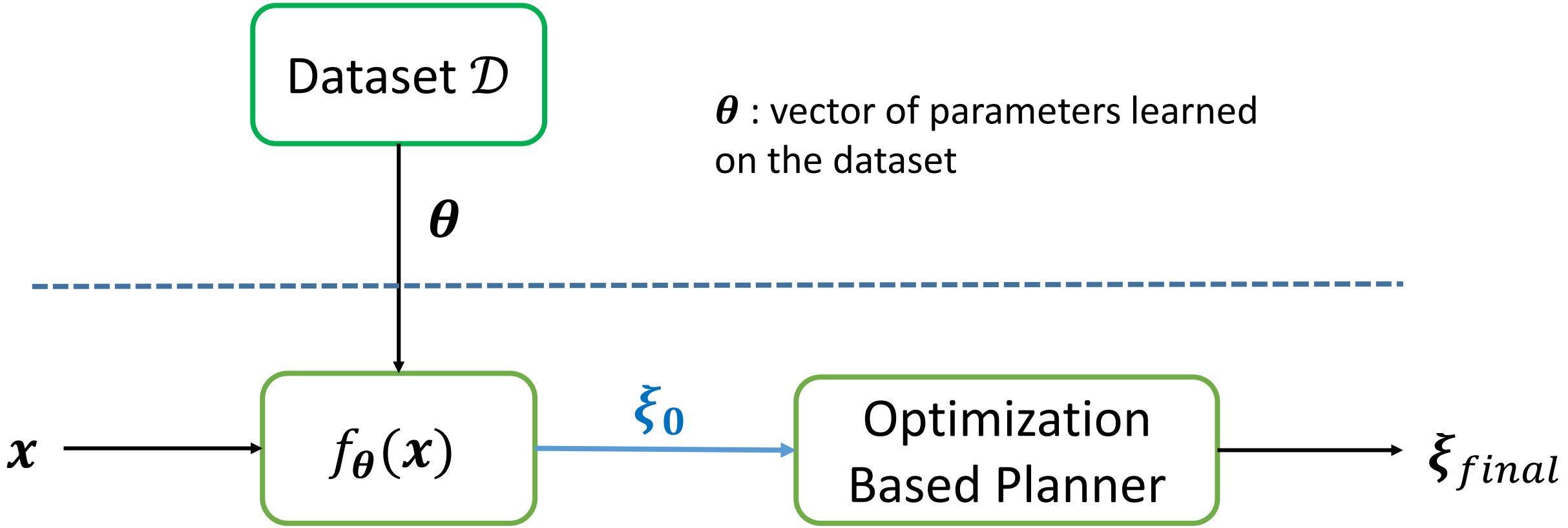
Traditional trajectory prediction



Problem: not a scalable method

Idea

Solution : learning on the dataset and **generating** new trajectories.



The methods **do not require the dataset** during runtime.

Mathematical representation of the problem

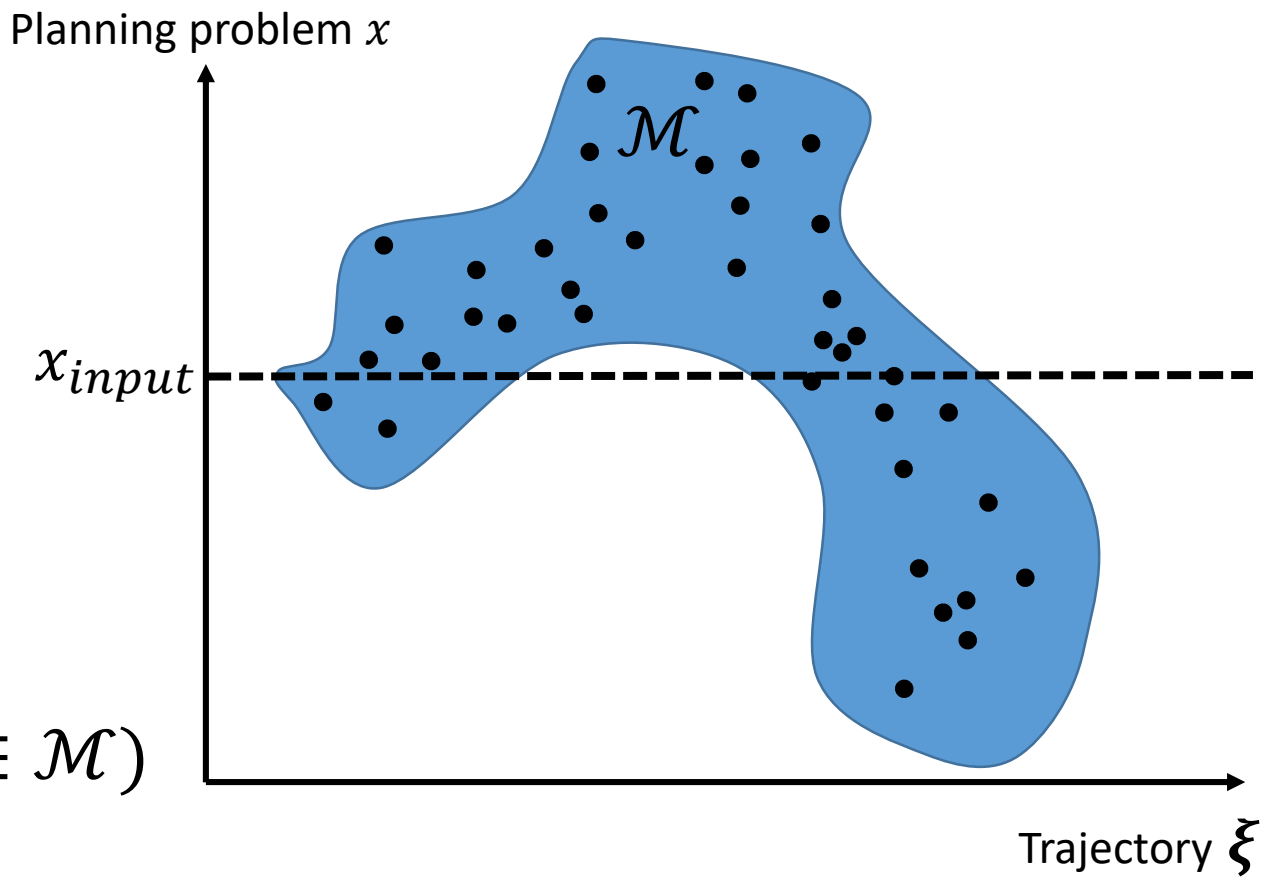
The dataset $\mathcal{D}$ lives inside a manifold $\mathcal{M}$ embedded in $X \times \Xi$.

Idea: approximating $\mathcal{M}$.

When $\boldsymbol{x}_{input}$ is given we **generate** $\boldsymbol{\xi}$ such as $\boldsymbol{d} = (\boldsymbol{x}_{input}, \boldsymbol{\xi}) \in \mathcal{M}$.

Hypothesis:

$\mathrm{p}(\boldsymbol{d}|\mathcal{D})$ is an approximation of $\mathrm{p}(\boldsymbol{d} \in \mathcal{M})$

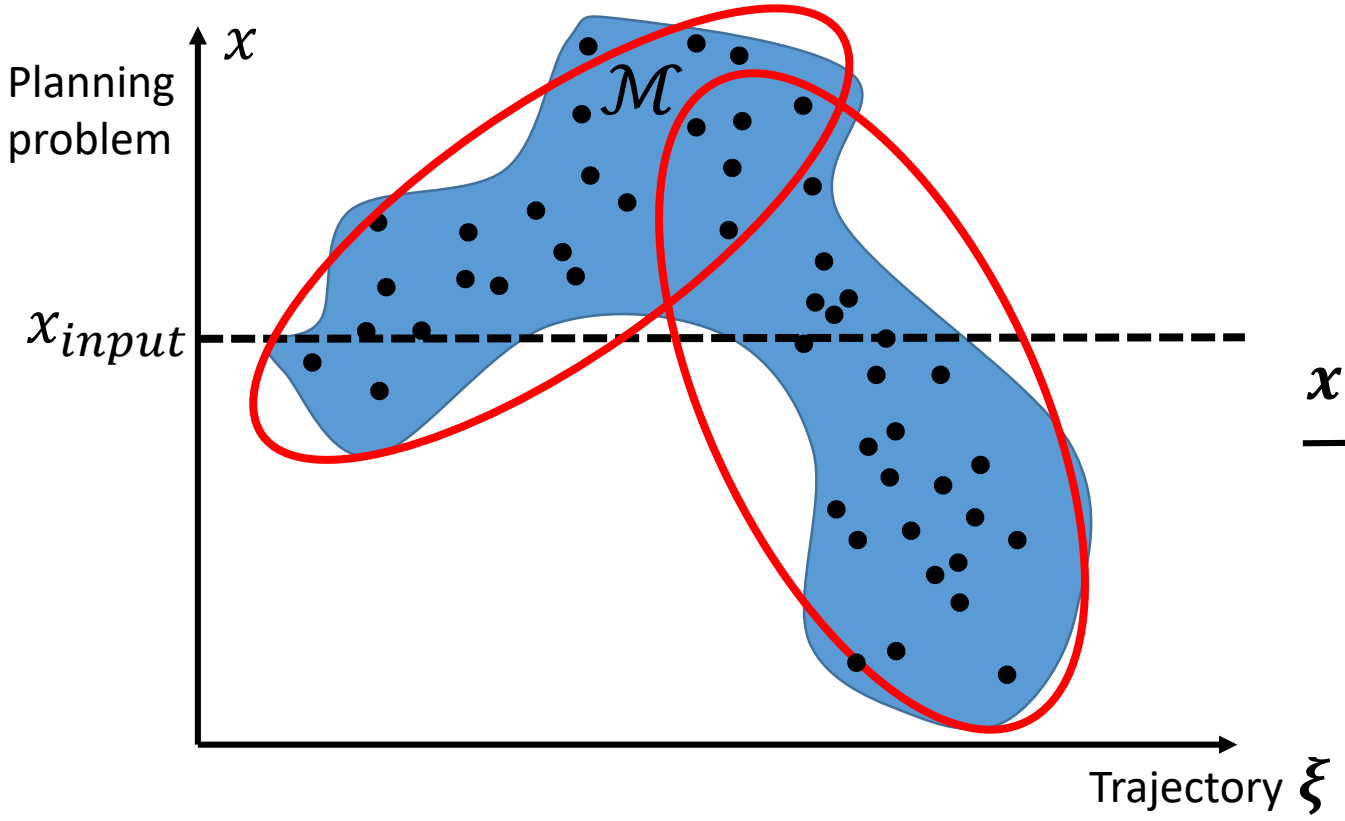


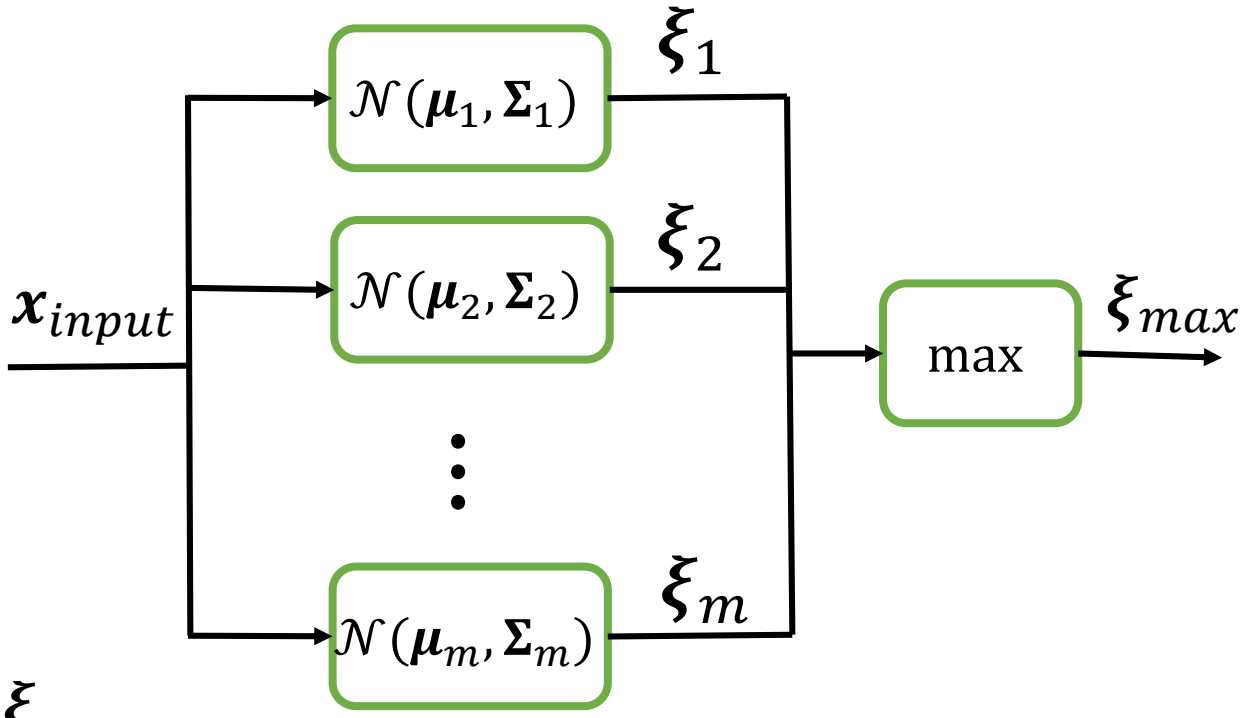
We use the **approximation of the manifold** $\mathcal{M}$ as a tool to find **suitable initial trajectories** $\boldsymbol{\xi}$ for a given planning problem $\boldsymbol{x}_{input}$.

Gaussian Mixture Model

$$\mathrm{p}(\boldsymbol{d}|\mathcal{D}) = \sum_{i=1}^{m} \pi_i \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

For each gaussian we find the optimal $\boldsymbol{d} = (\boldsymbol{x}, \boldsymbol{\xi})$ such as $\boldsymbol{x} = \boldsymbol{x}_{input}$





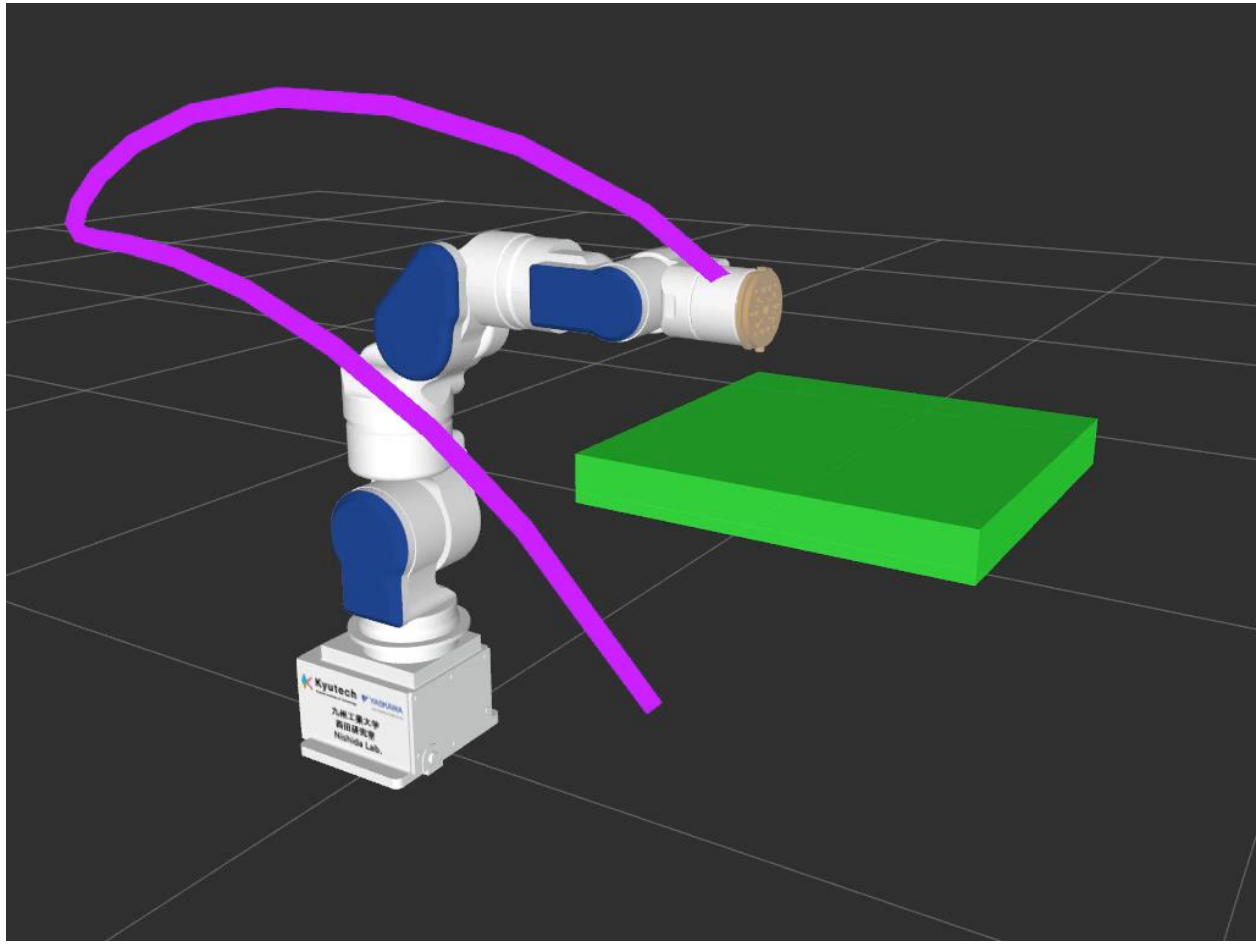
Experiment

Robot: 7 degree-of-freedom industrial arm

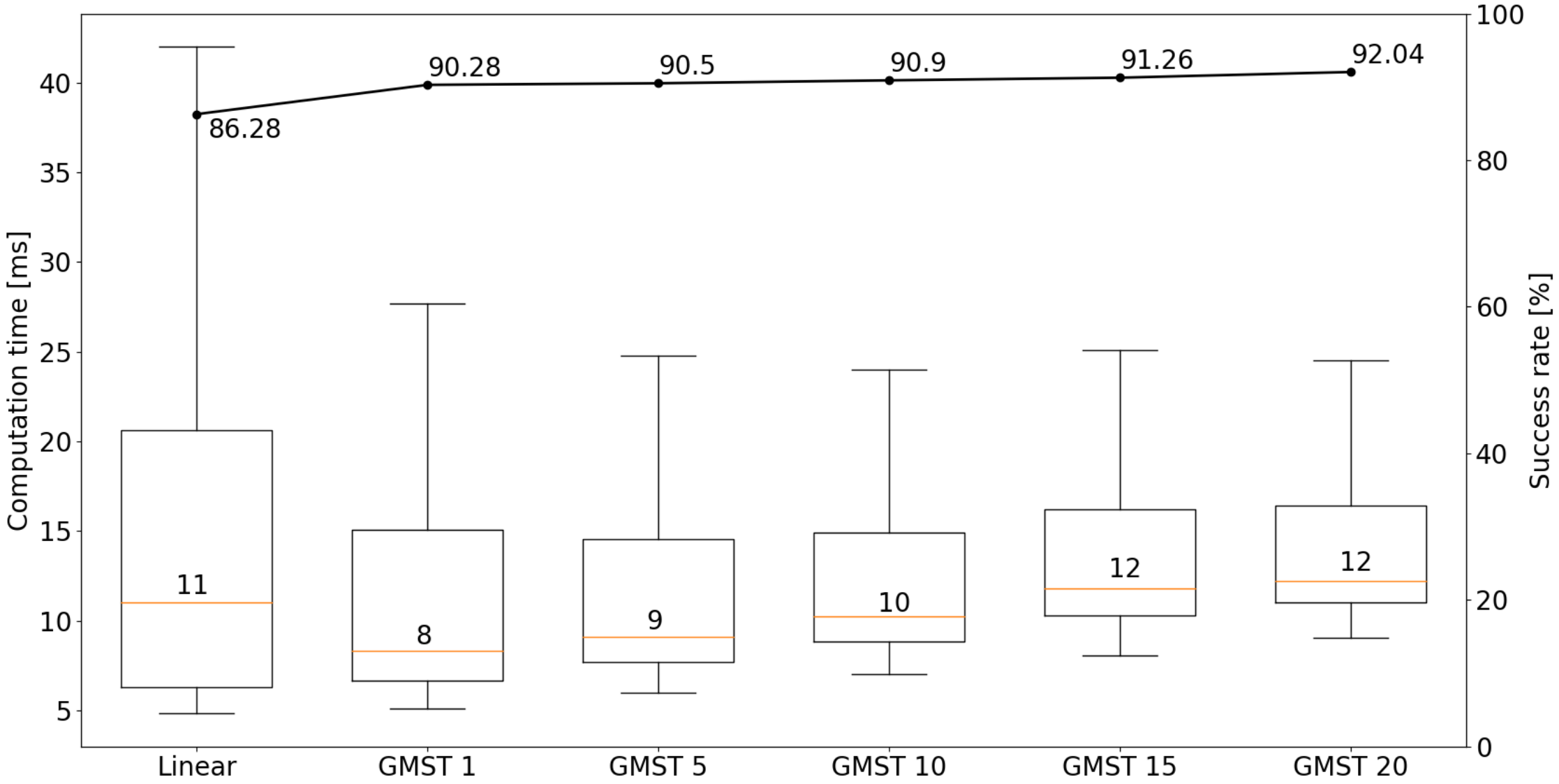
Optimizer: STOMP motion planner

Dataset length: 100,000

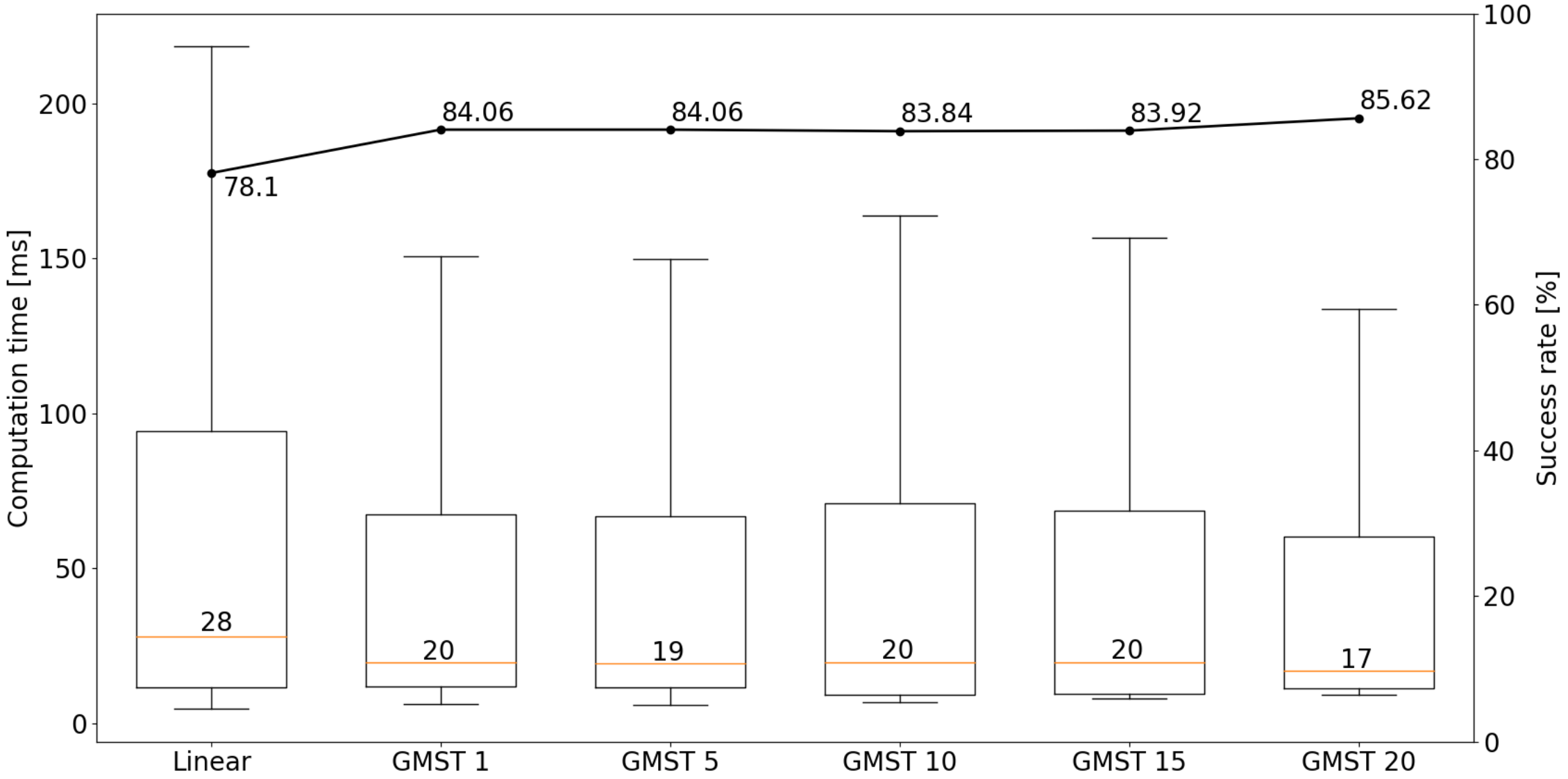
Planning problem: moving from above the obstacle to under the obstacle



Experimental results



Experimental results



Neural network generative models

Problem:

The GMST method shows good results but is relatively **slow**.

It also **scales poorly** with the number of dimensions.

Solution:

Machine learning generative models :

- Variational Autoencoder (VAE)
- Generative Adversarial Networks (GAN)

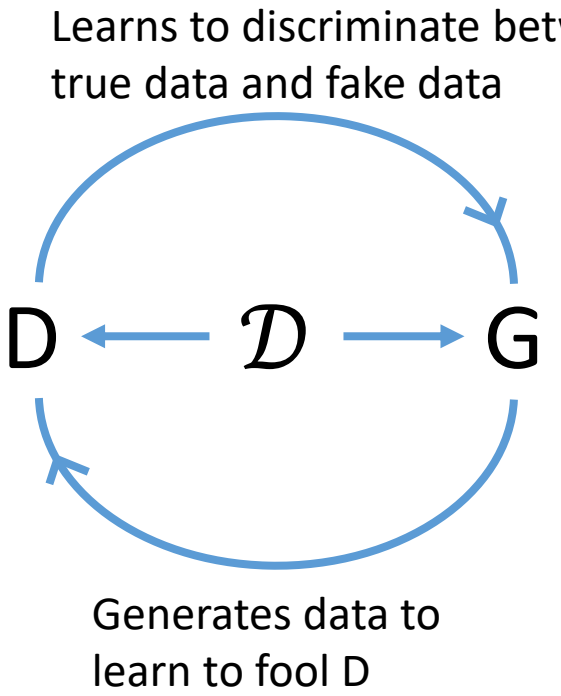
It is not easy to know when a generative model has converged during training.

Generative Adversarial Networks

A GAN is made of two neural networks that are in an adversarial setting.

- A generator G
- A discriminator D

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log D(1-D(G(z)))]$$

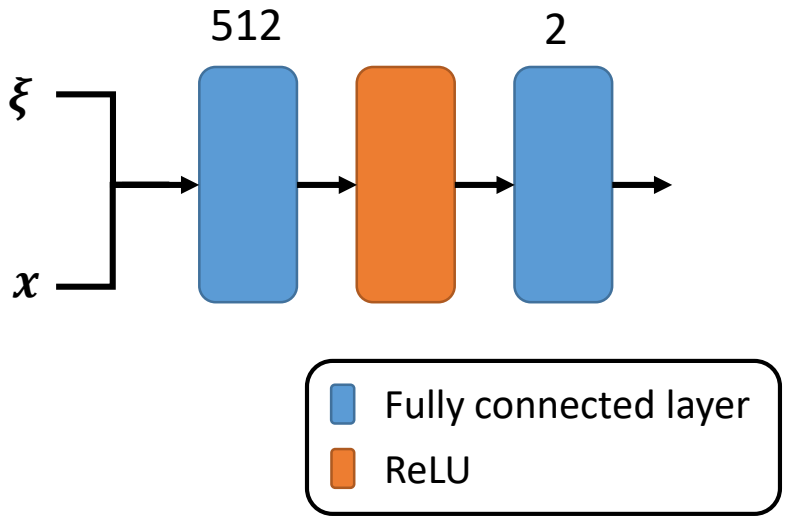


Conditional Generative Adversarial Networks

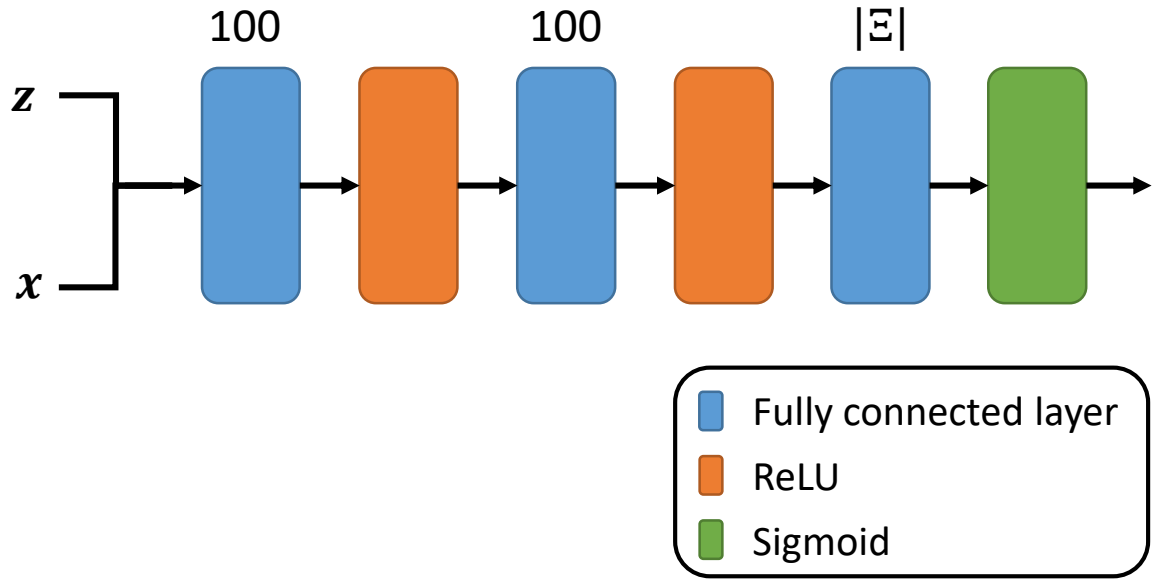
GAN generates data like the ones in $\mathcal{D}$.

Problem : we do not want to generate new problems but want to generate solutions ***conditionned*** on a given problem.

Conditional GAN (CGAN) :

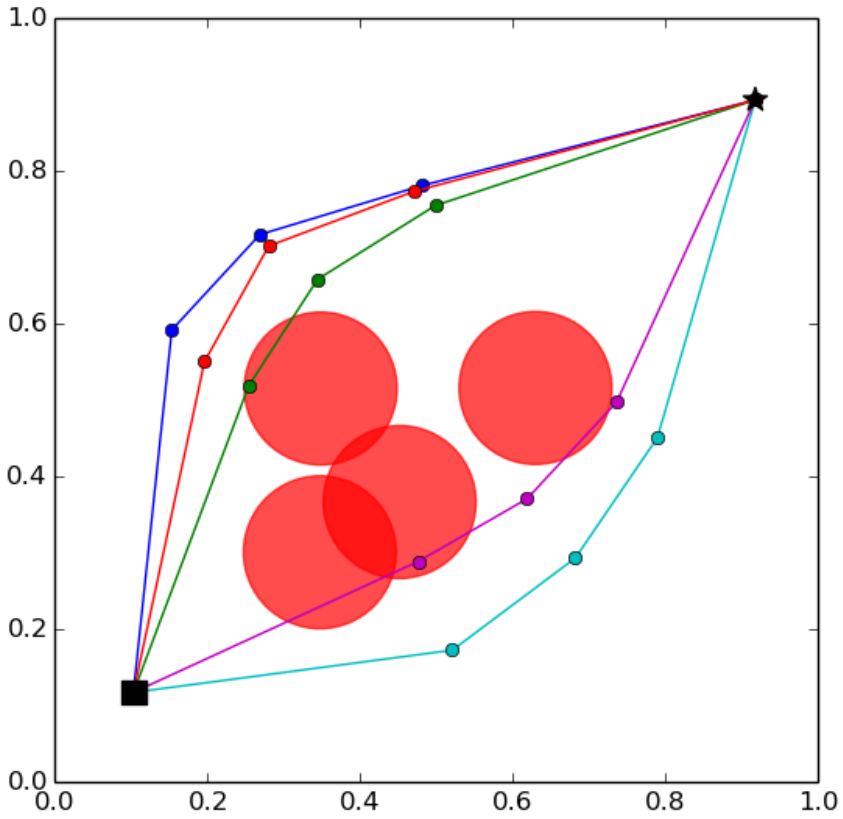


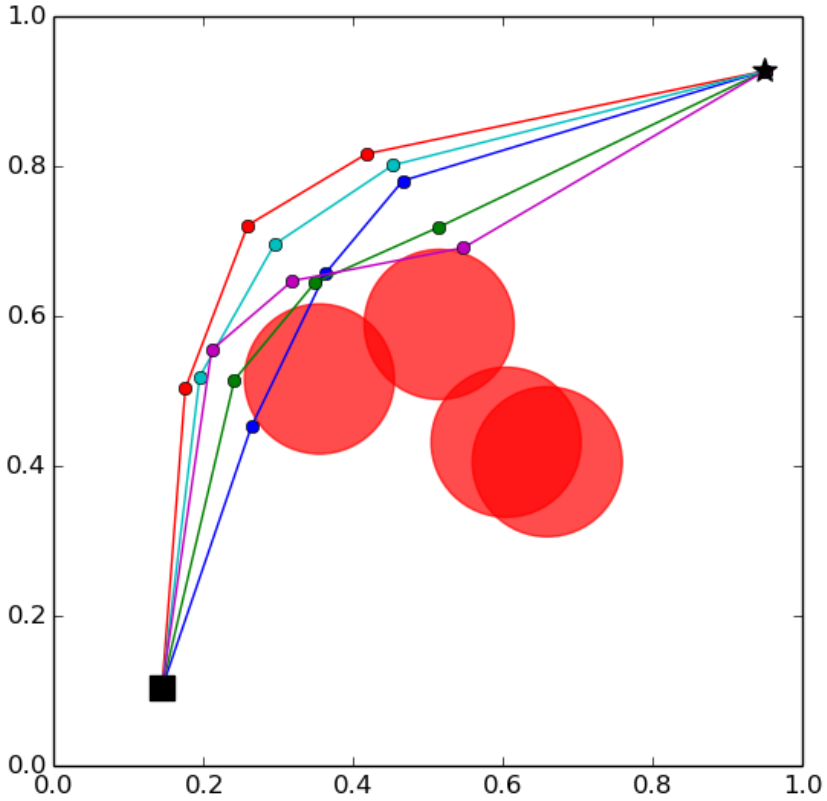
Discriminator



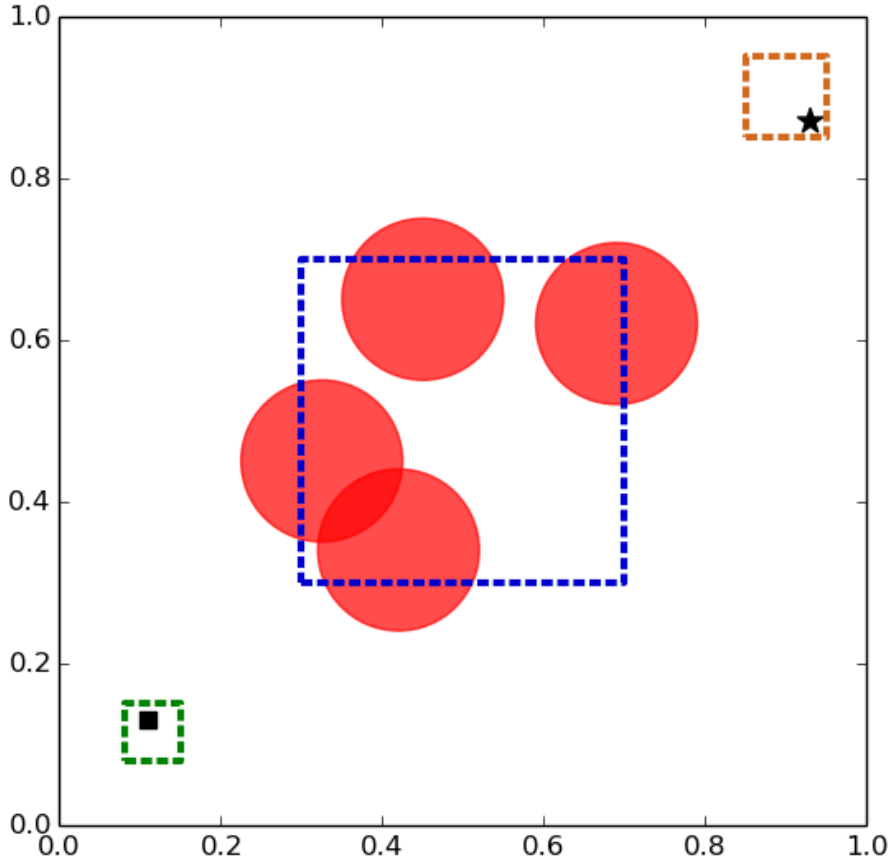
Generator

Trajectory prediction by a CGAN





Experiment results



Collision-free trajectories [%]	Linear	Nearest Neighbor	GMST	CGAN
2D 4 obstacles	96.8	**34.4**	90.2	61.9
20D 400 obstacles	74.5	98.9		**63.2**

Conclusion

Trajectory prediction :

is **efficient**, helps to **reduce the computation time** and gives **better initial trajectory** to be optimized.

GMST method :

- ○ **easy to use**
- × **model too simple**
- × **takes time to compute**

CGAN method :

- ○ **fast**
- ○ **high quality trajectory generation**
- × **difficult to use**

Both methods **do not require** the dataset at runtime.

Future work :

automatic data gathering
continuous learning