

Learning to Optimize Motion Planning

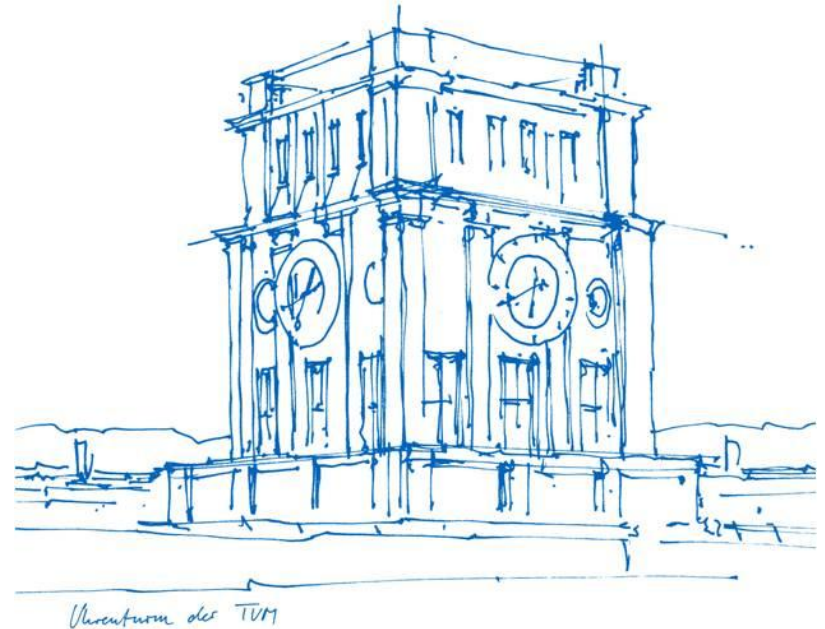
Team 2: Yufan Zhao, Mingyang Wang

Advisor: Johannes Tenhumberg

Technical University Munich

Advanced Deep Learning for Robotics, WS20/21

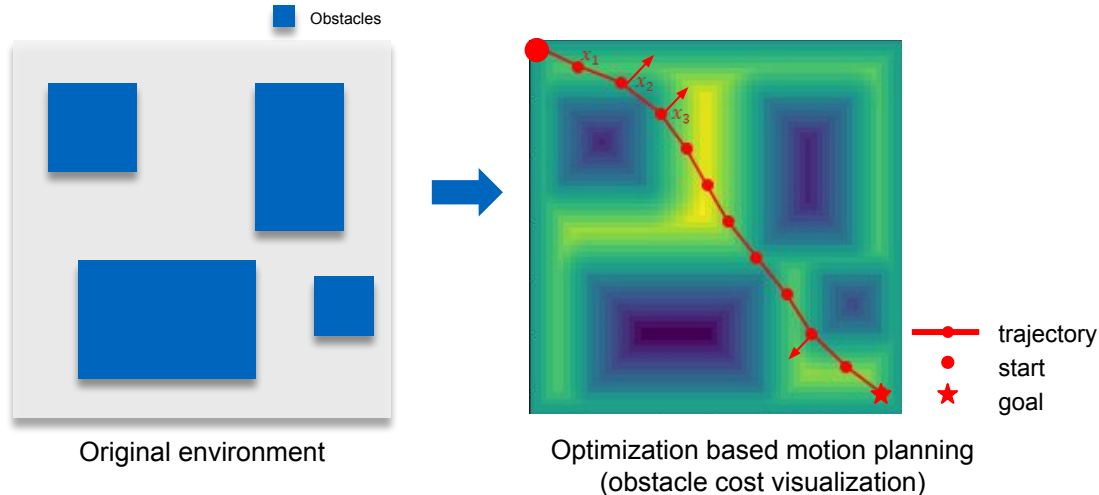
11. March 2021



Motivation - Optimization-based motion planning

• Optimization-based motion planning (OMP)

- find a feasible path = minimize the **objective function**
- collision free path \rightarrow lower cost; shorter path \rightarrow lower cost



Trajectory $\xi = \{x_1, x_2, \dots, x_n\}$



Objective Function
(obstacle cost + length cost)

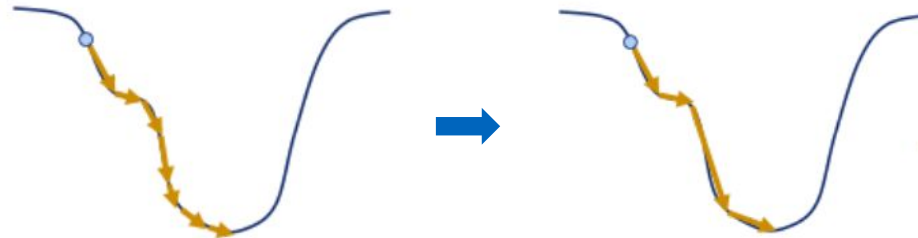
$$U(\xi) = F_{obs}(\xi) + \lambda F_{length}(\xi)$$

with $\xi = \{x_1, x_2, \dots, x_n\}$ - trajectory

Motivation - Learning to optimize

- **Learning to optimize**

- **Goal:** minimize the objective function $U(\xi)$
 - First thought: gradient-descent-based algorithm; Update step $\Delta\xi = -\gamma * \partial U / \partial \xi$
 - Idea: Can we choose the update vector through learning?
- Train a **Reinforcement Learning** agent to optimize a certain class of objective function specifically.



GD-wise optimization

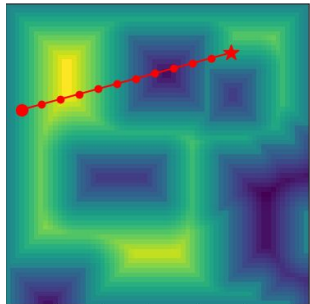
train the RL agent to
learn the update size

Update step = Agent's action
→ learn to choose it reasonably

Learning to Optimize - Whole pipeline

1 - Generate Motion Planning Environment

- **Environment:** 2D, 64*64 with randomly generated start and end point
- **Obstacle:** N=10 randomly generated rectangles



easy environment example

2 - Set up Reinforcement Learning Framework

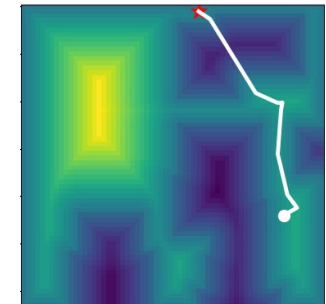
- **Objective function $U(\xi)$** based on the trajectory
- **State:** Trajectory
 $\xi = \{x_1, x_2 \dots x_N\}$
- **Action:** Step vector
 $\Delta\xi = \{\Delta x_1 \dots \Delta x_N\}$
- **State transition:** $\xi = \xi + \Delta\xi$
- **Observation:** Gradient $\partial U / \partial \xi$
- **Reward:** $-U(\xi) / -\mu^* U(\xi)$
reward factor μ to encourage feasible solution

3 - Pretraining and Reinforcement Learning

- **Policy Network** = RL Agent
Dim = 64, 128, 256
- **Pretraining:** GD-based
($\partial U / \partial \xi, -\gamma \partial U / \partial \xi$)
- **Reinforcement Learning:**
PPO algorithm
- **Training Set:** 100 RL envs
Easy, hard, mix training
Hard: 200-step-GD with straight line initialization fails

4 - Test and Evaluation

- **Easy & Hard test benchmark:**
 - 1000 easy test environments
 - 100 hard test environments
- **Test planners in 200 steps**



hard environment example

Result – comparison of RL and GD motion planner

- Try different hyperparameter settings:
 - latent dimension of policy network: 64/128/256;
 - reward factor $\mu=0.1$ / 1: strength to encourage feasible solution
 - Training epochs = 10k / 80k
- RL agent output performs GD agent

Table 1 Performance of RL motion planner using different hyperparameters (success rate in testing envs)

Test Benchmark	Easy Benchmark					Hard Benchmark				
Hyperparameters	Dim = 64 $\mu = 1$	Dim=64 $\mu = 0.1$		Dim = 128 $\mu = 0.1$	Dim = 256 $\mu = 0.1$	Dim = 64 $\mu = 1$	Dim = 64 $\mu = 0.1$		Dim = 128 $\mu = 0.1$	Dim = 256 $\mu = 0.1$
Training Epochs	10k	10k	80k	10k	10k	10k	10k	80k	10k	10k
Easy Training	81.4%	82.7%	84.3%	81.4%	81.5%	20%	23%	30%	22%	24%
Mix Training	80.9%	83.2%	84.8%	81.7%	82.6%	24%	27%	33%	25%	25%
Hard Training	80.1%	82.0%	83.3%	81.7%	81.8%	27%	25%	37%	25%	25%
Gradient Descent		79.7%					0%			

Result - Runtime Analysis

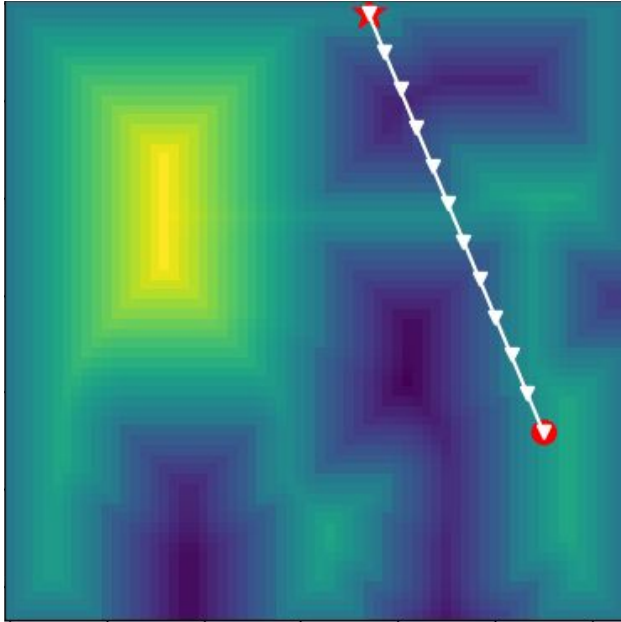
- Test on 100 environments where both GD and RL succeed and GD needs at least 10 steps
 - **RL agent** trained on easy environments (dim=64, $\mu=0.1$) with various training epochs (10k, 30k, 50k, 80k)
 - **GD agent** use 0.1 as learning rate

Table 2 Runtime performance of RL and GD motion planner

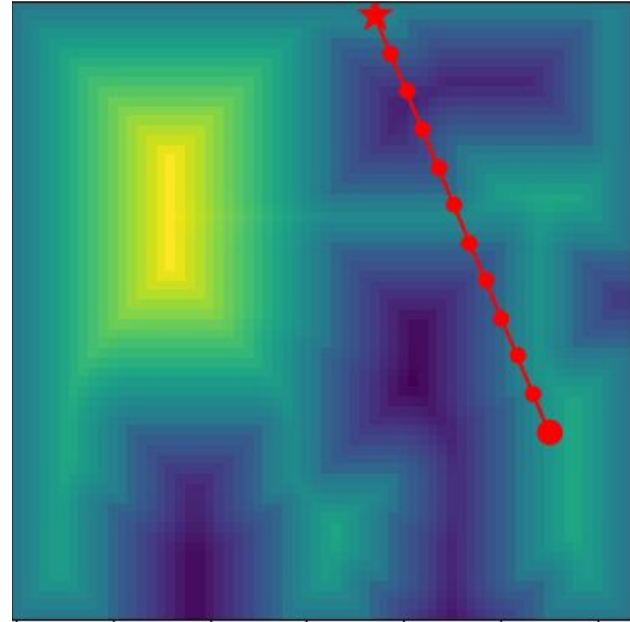
Algorithm & Training Epochs	time per iteration	iterations to find feasible solution	Average time
Gradient Descent	1.31ms	26.43	34.62ms
RL- 10k epochs	1.71ms	24.82	42.44ms
RL- 30k epochs		17.83	30.49ms
RL - 50k epochs		10.94	18.71ms
RL - 80k epochs		8.28	14.66ms

Result – visualization of RL and GD motion planner

- Some local optima that stuck gradient descent can be avoided by the learned optimization algorithm



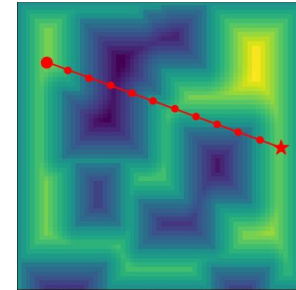
Env with gradient descent



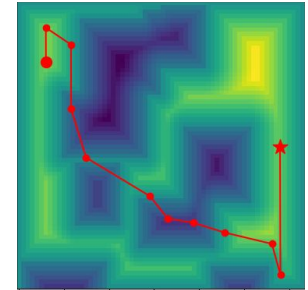
Env with reinforcement learning

Multiple random initialization

- Multi-start for optimization-based motion planning
previously: straight line from start to end point as initialization
now: multiple random initialization



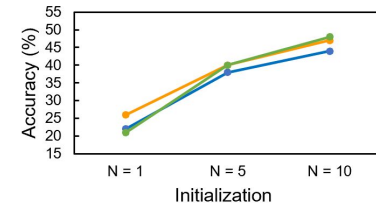
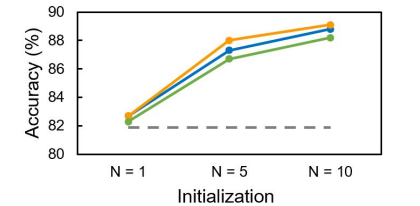
Straight line initialization



random start initialization

Table 2 Performance comparison w vs. w/o random start (success rate in testing envs)

Training env \ Benchmark	Easy Benchmark			Hard Benchmark		
	N=1	N=5	N=10	N=1	N=5	N=10
Random Start						
Easy Training	82.7%	87.3%	88.8%	22.0%	38.0%	44.0%
Mix Training	82.7%	88.0%	89.1%	26.0%	40.0%	47.0%
Hard Training	82.3%	86.7%	88.2%	21.0%	40.0%	48.0%
Gradient Descent	79.7%	84.4%	86.2%	0%	32.0%	41.0%



Larger observation space



- Previous observation space: current gradient
- Extend the observation space:
 - + multiple steps gradient ($n = 2, 5$); obstacle information(position and size) 
 - + obstacle cost  → better on hard environments

Table 3 Performance comparison with different observations

Benchmark	Easy Benchmark			Hard Benchmark		
Policy Network Latent Dimension	64	128	256	64	128	256
Initial Observation	87.3%	87.9%	88.8%	38.0%	45.0%	41.0%
Extended Observation: Gradient + Obstacle Cost	88.5%	87.6%	88.3%	55.0%	57.0%	53.0%
Gradient Descent	84.4%			32.0%		

performance evaluated with easy training, multi start $N=5$ and reward factor $\mu = 0.1$

Image based agent – settings

- Image as of workspace as observation
- Cnn policy
- Skip connection for gradient

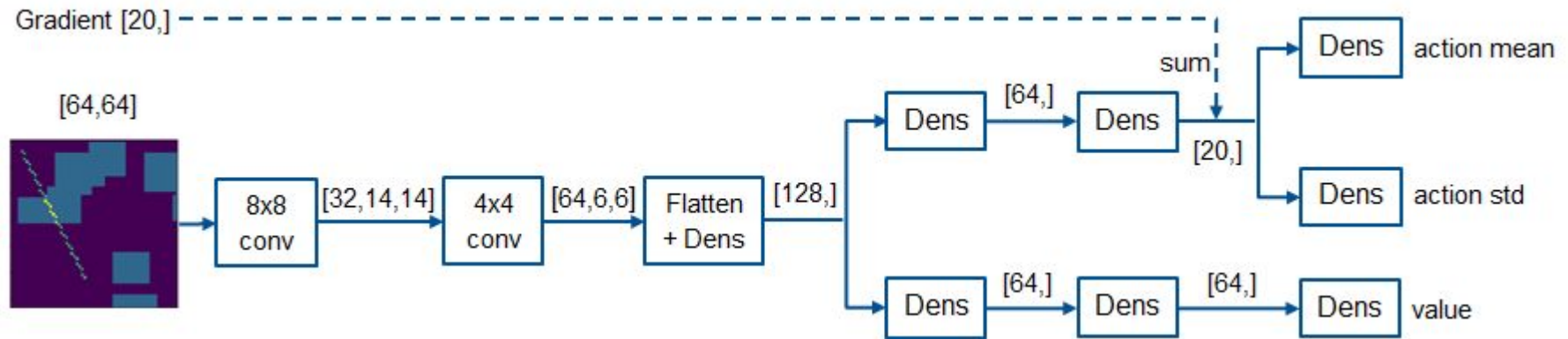
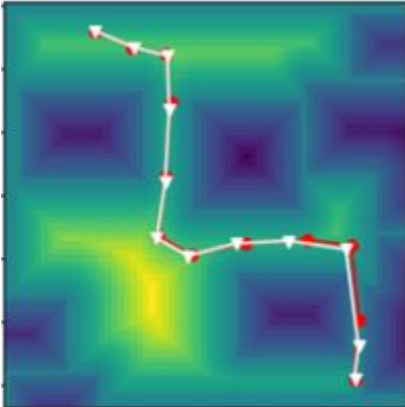
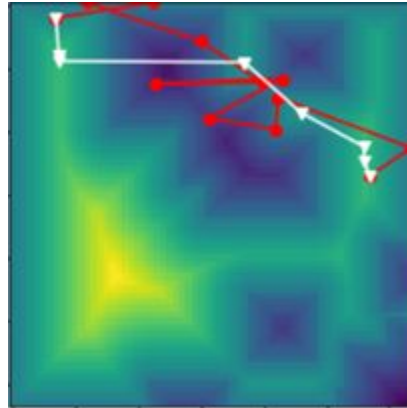


Image based agent – results

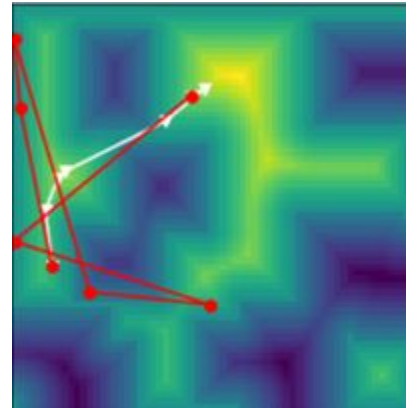
Pretrained gradient-based agent



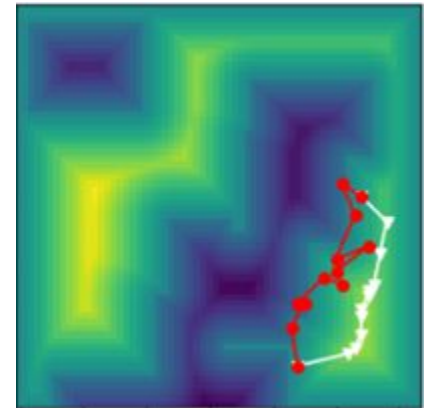
Pretrained image-based agent



RL image-based agent with gradient info



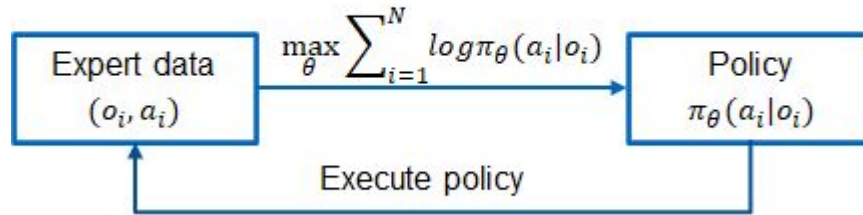
RL image-based agent without gradient info



Pretraining (supervision mechanism) is very important

Discussion

- Recursive pretraining to solve distribution mismatch problem(DAgger)



- Incorporate supervision mechanism into RL framework
 - Expert MP algorithm to fix failed rollouts
- Guided policy search
 - Optimize time-varying linear policy $\psi: \mathcal{N}(K_t s_t + k_t, G_t), t = 1 \dots T$
 - Clone the behavior of ψ to π with trajectories generated by ψ

Conclusions

- Motion planning is posed as an optimization problem, which can be solved by general handcrafted optimization algorithm like gradient descent or a learned algorithm
- With suitable hyperparameters, the learned optimization algorithm outperforms gradient descent by avoiding some local optima
- The learned algorithm also increase the planning speed by a factor of 2.4
- Multiple random initialization can further improve the successful rate of the RL planner
- Supervision mechanism is very important. Incorporating supervision mechanism with RL should be future focus

Thank you for your attention!

Back up - cost calculation

1. Discretize the path to point $\{x_0, x_1 \dots x_n, x_{n+1}\}$, x_0 is the start point and x_{n+1} is the end point
2. On the line segment between x_i and x_{i+1} , we sample additional points: $\{\xi_1^{i,i+1} \dots \xi_s^{i,i+1}\}$
3. Obstacle cost: $\sum_{i=1}^n c(x_i) + \sum_{i=0}^n \sum_{j=1}^s c(\xi_j^{i,i+1})$

$$c(x) = \begin{cases} -\mathcal{D}(x) + \frac{\varepsilon}{2} & \text{if } \mathcal{D}(x) < 0 \\ \frac{1}{2\varepsilon} (\mathcal{D}(x) - \varepsilon)^2 & \text{if } 0 < \mathcal{D}(x) \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{D}(x)$ is the distance between x and its nearest obstacle

4. Length cost: $\sum_{i=0}^n (x_{i+1} - x_i)^2$
5. Final objective function:

$$U(x_1 \dots x_n) = \sum_{i=1}^n c(x_i) + \sum_{j=1}^s c(\xi_j^{i,i+1}) + \lambda \sum_{i=0}^n (x_{i+1} - x_i)^2$$

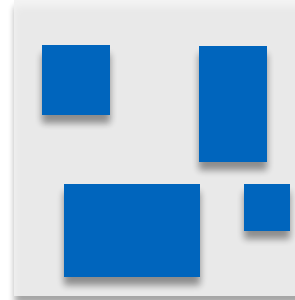
Back up - Objective function

- Set up motion planning environments
- Define objective function
= Obstacle cost + Smoothness cost

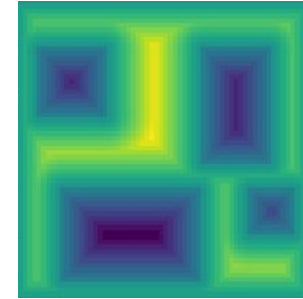
$$U(\xi) = F_{obs}(\xi) + \lambda F_{smooth}(\xi)$$

$$F_{obs}(\xi) = \frac{1}{N_S} \sum_{n=1}^N \sum_{n_s=1}^{N_S} c(x(\xi_{n,n_s}))$$

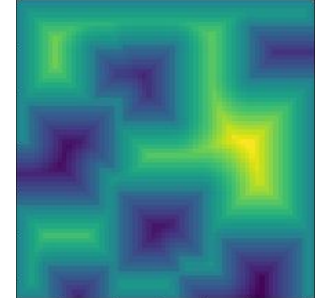
$$F_{smooth}(\xi) = \sum_{n=1}^{N-1} (x(\xi_{n+1}) - x(\xi_n))^2$$



Original environment



Obstacle cost Visualization
(easy case)



Obstacle cost Visualization
(hard case)

$$\text{with } c(x) = \begin{cases} -D(x) + \frac{1}{2}\varepsilon, & \text{if } D(x) < 0 \\ \frac{1}{2\varepsilon}(D(x) - \varepsilon)^2, & \text{if } 0 < D(x) \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

where we use signed distance field for obstacle presentation

$$D(x) = d(x) - \bar{d}(x)$$

