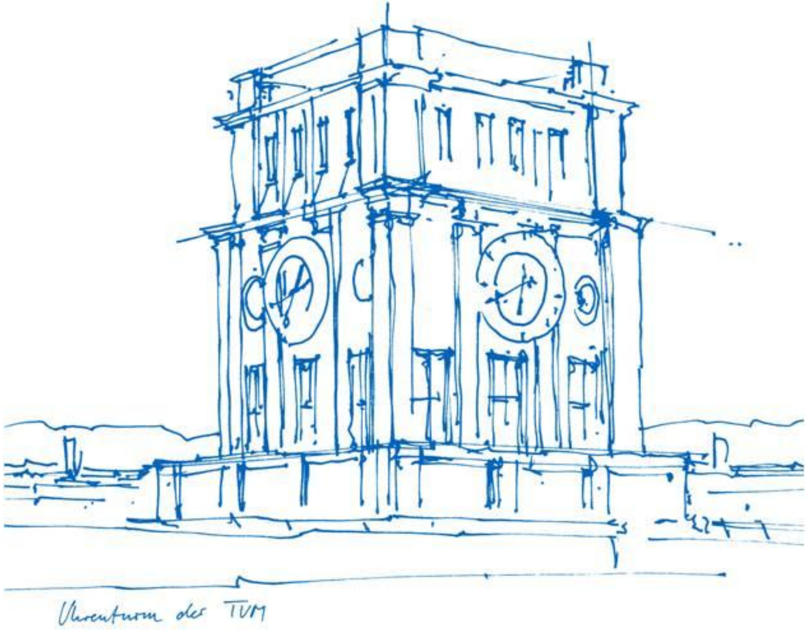
Learning to Optimize Motion Planning

Team 2: Yufan Zhao, Mingyang Wang

Advisor: Johannes Tenhumberg



Technical University Munich

Advanced Deep Learning for Robotics, WS20/21

14. January 2021

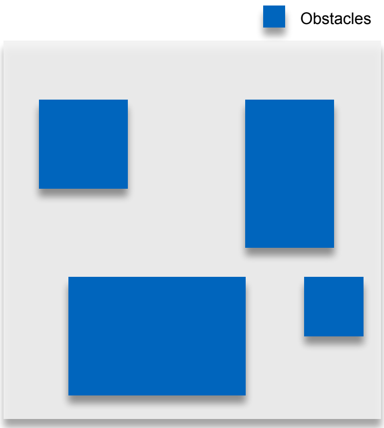
Outlook

- Problem statement
    - Optimization based motion planning
    - Reinforcement learning for optimization
- First results
- Next step

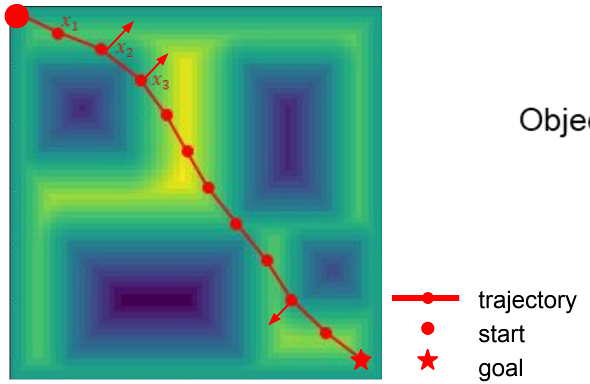
Problem Statement - OMP

- Optimization-based motion planning(OMP)

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



Original environment



Optimization based motion planning (obstacle cost visualization)

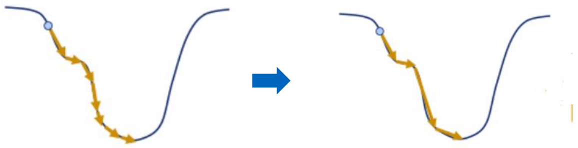
Objective function = Obstacle cost + Length cost

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

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

Problem Statement - Learning to optimize

- Learning to optimize
  - Goal: minimize the objective function $U(\xi) = F_{obs}(\xi) + \lambda F_{length}(\xi)$
    - First thought: gradient descent, momentum, Adam…
    - General idea: in each iteration, a step vector is computed using some update formula.
    - Can the step vector be chosen automatically?
  - Taking advantage of Reinforcement Learning
    - Learn an update policy which has better performance



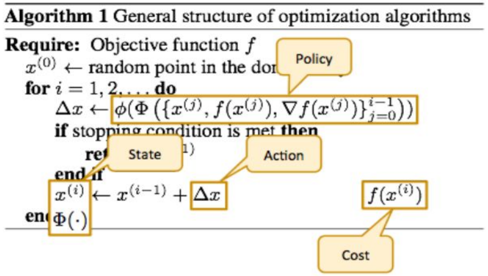
GD-like algorithm

learn the update size

Algorithm 1 General structure of optimization algorithms

Require: Objective function f
$x^{(0)} \leftarrow$ random point in the domain of f
for $i = 1, 2, \dots$ **do**
$\Delta x \leftarrow \boxed{\phi(\{x^{(j)}, f(x^{(j)}), \nabla f(x^{(j)})\}_{j=0}^{i-1})}$
if stopping condition is met **then**
return $x^{(i-1)}$
end if
$x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
end for

Gradient Descent	$\phi(\cdot) = -\gamma \nabla f(x^{(i-1)})$
Momentum	$\phi(\cdot) = -\gamma \left(\sum_{j=0}^{i-1} \alpha^{i-1-j} \nabla f(x^{(j)})\right)$
Learned Algorithm	$\phi(\cdot) =$ Neural Net

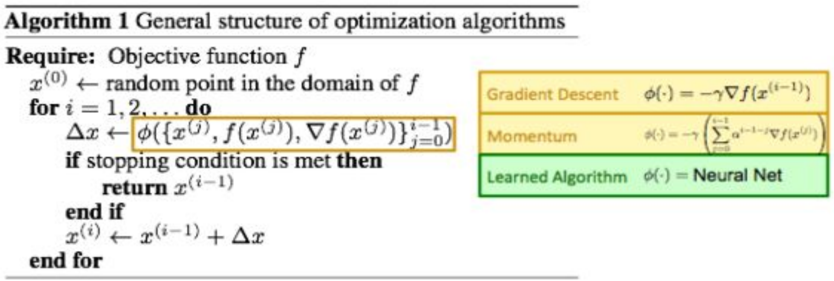


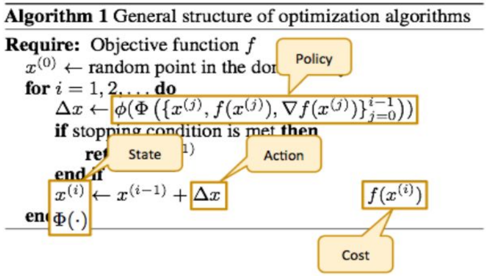
Problem Statement - Learning to optimize

- Use reinforcement learning to train the agent
  - State space → current trajectory
  - Action space → update step size (points' movement of the trajectory)
  - Observation space → gradient of current objective function(to be extended)
  - Reward → negative value of objective function.

    → If a collision-free trajectory is found, increase the reward

    --> encourage the agent to reach the goal





Whole pipeline

1. **Generate training MP environments:**
   - 2d point robot, 100 environments
   - Boundary: $[0,64]^2$
   - 10 rectangle obstacles
   - Random positions and random size $[10,20]$
   - Random start and end point

2. **Set up reinforcement learning framework:**
   - Objective function $U(\boldsymbol{x})$ by a MP environment
   - State: $\boldsymbol{x}$
   - Observation: gradient $\partial U / \partial x$
   - Action: update of optimization algorithm $\Delta \boldsymbol{x}$
   - State transition: $\boldsymbol{x} = \boldsymbol{x} + \Delta \boldsymbol{x}$
   - Reward: $-U(\boldsymbol{x})$

3. **Pretraining and Reinforcement learning:**
   - Supervised pretraining for policy network
   - Training data: $(\partial U / \partial x, -\gamma\, \partial U / \partial x)$
   - Reinforcement learning: PPO 10k iterations
   - on 100 RL environments
   - Policy network = learned optimization algorithm

4. **Test and evaluation:**
   - Easy test benchmark: 1000 MP test environments
   - Environment generation same as training
   - Hard test benchmark: 100 MP test environments
   - 200 steps GD with straight line initialization fail

First result – comparison of RL and GD motion planner

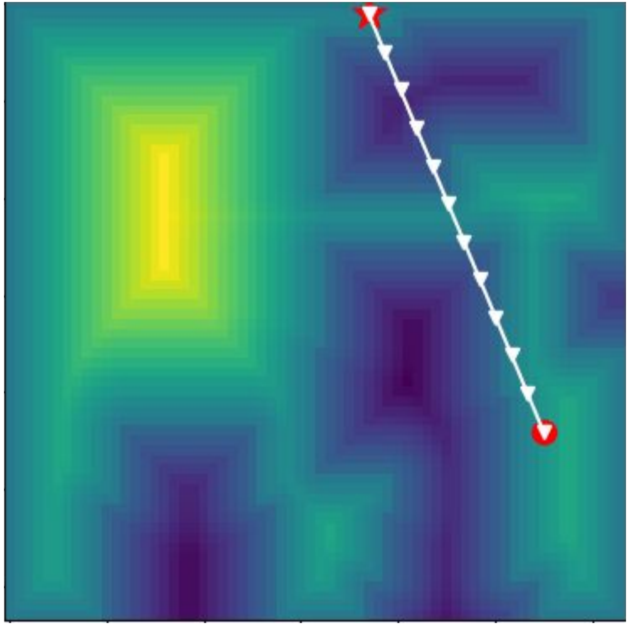
- With suitable hyperparameter, RL agent output performs GD agent

Policy Net	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,64,64,20]	[20,100,100,20]	[20,20,20,20]
Reward factor	1	1	1	0.1	0.1	0.1	Only true	Only true	Only true	0.1	0.1
Train env	easy	hard	mix	easy	hard	mix	easy	hard	mix	mix	mix
RL ✔ GD ✔	73.60%	73.80%	73.40%	72.90%	73.30%	73.40%	71.70%	71.90%	69.90%	73.50%	69.40%
RL ✘ GD ✘	19.70%	19.00%	18.40%	19.70%	19.10%	17.40%	17.90%	16.80%	18.90%	17.20%	19.80%
RL ✔ GD ✘	4.50%	5.20%	5.80%	4.50%	5.10%	6.80%	6.30%	3.60%	5.30%	7.00%	4.40%
RL ✘ GD ✔	2.20%	2.00%	2.40%	2.90%	2.50%	2.40%	4.10%	7.70%	5.90%	2.30%	6.40%
RL ✔	78.10%	79.00%	79.20%	77.40%	78.40%	80.20%	78.00%	75.50%	75.20%	80.50%	73.80%
GD ✔	75.80%	75.80%	75.80%	75.80%	75.80%	75.80%	75.80%	79.60%	75.80%	75.80%	75.80%
RL ✔ on hard	22.00%	23.00%	23.00%	22.00%	22.00%	**31.00%**	24.00%	13.00%	21.00%	**30.00%**	16.00%

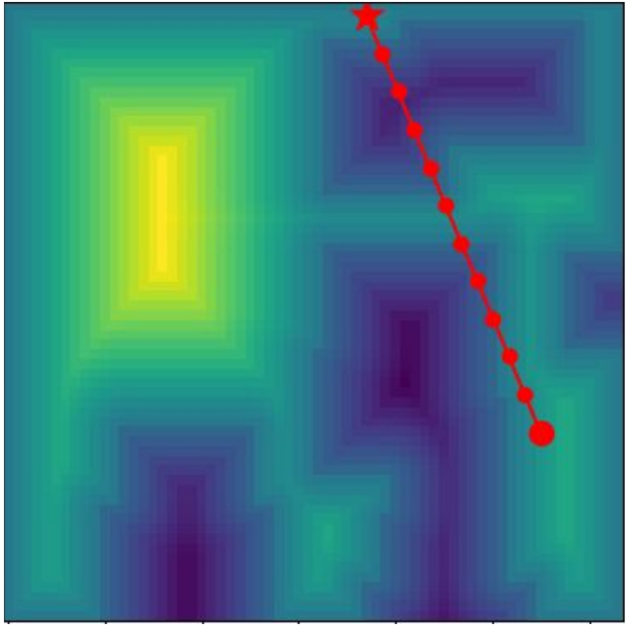
First result – visualization of RL and GD motion planner

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

Env_1 with gradient descent



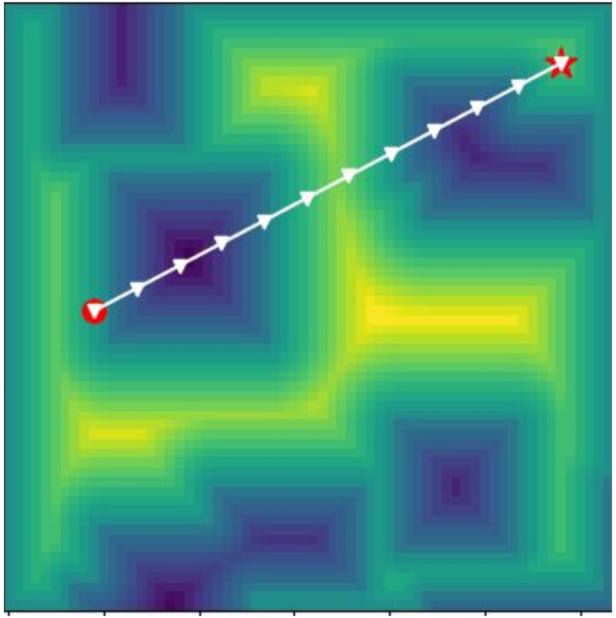
Env_1 with reinforcement learning



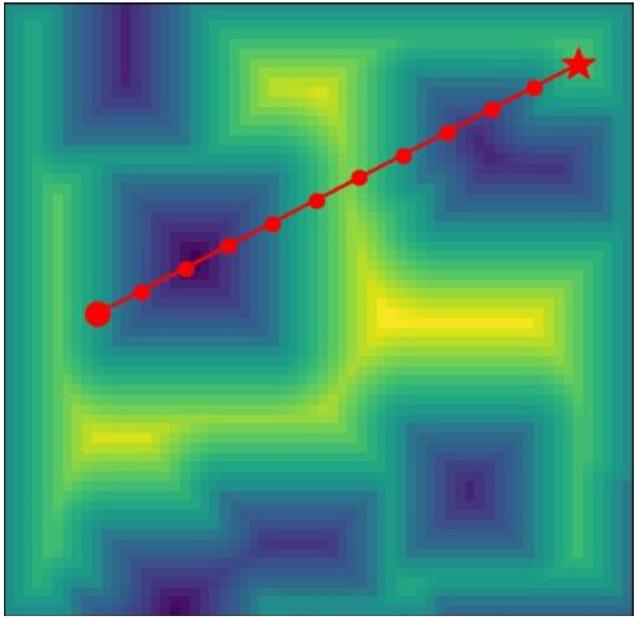
First result – visualization of RL and GD motion planner

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

Env_2 with gradient descent



Env_2 with reinforcement learning



First result – larger observation space

- Observation space includes multiple steps' gradient, position and objective value:
  - $[{}^{\partial U}/_{\partial x}|_{x^{(t)}}, \boldsymbol{x}^{(t)}, U(\boldsymbol{x}^{(t)}) \ldots {}^{\partial U}/_{\partial x}|_{x^{(t-k)}}, \boldsymbol{x}^{(t-k)}, U(\boldsymbol{x}^{(t-k)})]$
- Larger observation space did not yield better result as expected
- Only similar performance can be yielded
- Maybe we have not found the suitable hyperparameter to achieve better performance

Observation space	1grad+1pos +1val	1grad+1pos +1val	2grads	2grads	5grads	5grads
Policy Net	[41,128, 128,20]	[41,200, 200,20]	[40,128, 128,20]	[40,256, 256,20]	[100,320, 320,20]	[100,512, 512,20]
Reward factor	0.1	0.1	0.1	0.1	0.1	0.1
Train env	easy	easy	easy	easy	easy	easy
RL ✔ on easy	62.00%	42.00%	78.80%	54.80%	48.60%	48.10%
GD ✔ on easy	81.00%	72.00%	75.20%	75.80%	75.80%	75.80%
RL ✔ on hard	10.00%	2.00%	**20.00%**	2.00%	0.00%	0.00%

Current progress - random multi-start

- Multi-start for optimization-based motion planning

  previously: straight line from start to end point as initialization

  now: randomly generate points as initialization → more flexibility (1 straight line plus 9 random initialization now)

Table Performance comparison w vs. w/o random start (success rate in 1000 cases)

Training env	easy				hard				mix			
Benchmark	easy		hard		easy		hard		easy		hard	
Random start	x	✓	x	✓	x	✓	x	✓	x	✓	x	✓
RL	77.4%	**86.6%**	22.0%	**51.0%**	87.0%	**89.5%**	15.9%	**56.0%**	80.5%	**86.5%**	16.8%	**55.0%**
GD	75.8%	85.9%	-	46.0%	86.0%	88.0%	-	48.0%	80.0%	85.5%	-	49.0%

Feasible solution infeasible result

Next step

- Curriculum learning

  train the agent starting from easy environments, then increase the complexity of environments gradually

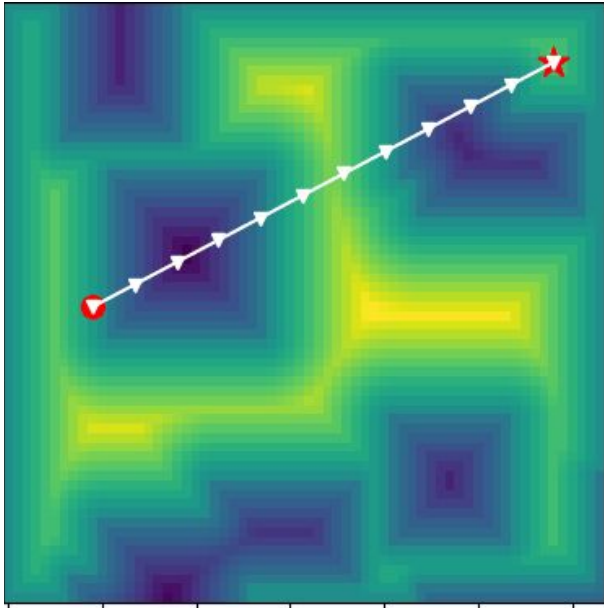
- Hyperparameter tuning

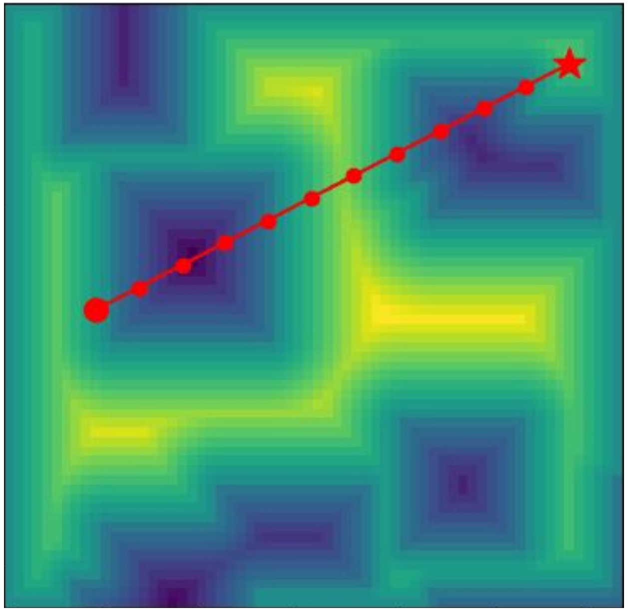
  find better hyperparameters combination to improve rl agent performance

- Image as observation space

  set the color image of MP environment and current path as observation

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: $\sum_{i=1}^{n} c(x_i) + \sum_{i=0}^{n} \sum_{j=1}^{s} c(\xi_j^{i,i+1})$

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

$\mathcal{D}(x)$ 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

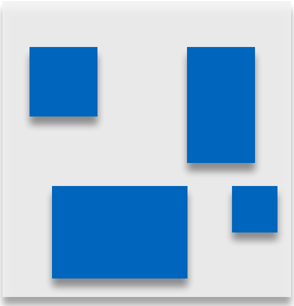
- Set up motion planning environments
- Define objective function

  = Obstacle cost + Smoothness cost

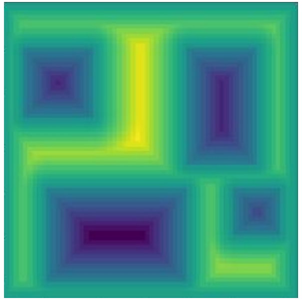
$$U(\xi) = F_{obs}(\xi) + \lambda F_{smooth}(\xi)$$

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

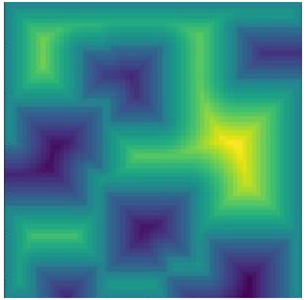
$$F_{smooth}(\xi) = \sum_{n=1}^{N-1}\left(x(\xi_{n+1}) - x(\xi_n)\right)^2$$



Original environment



Obstacle cost Visualization (easy case)



Obstacle cost Visualization (hard case)

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

where we use signed distance field for obstacle presentation

$$\mathrm{D}(x) = d\ (x) - \dot{d}\ (x)$$