Advance Robot Control and Learning Project

Chair : MIRMI

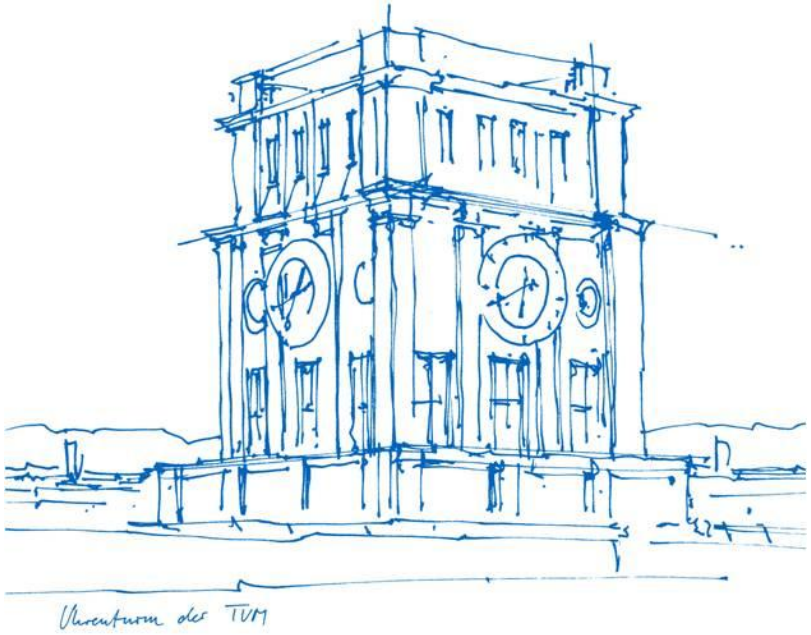
Project Members:

Batu Kaan Özen

Yanni Zhang

Farhan Khan

Peng Xie



Topics

1. Q1.1
2. Q1.2
3. Q2.1
4. Q2.2
5. Q2.3
6. Q3.1
7. Q3.2

Methods & Tools

Task 1.1: Check self-collision from specific joint configuration

Task 1.2: Identify and avoid self-collision from given joint and Cartesian trajectories

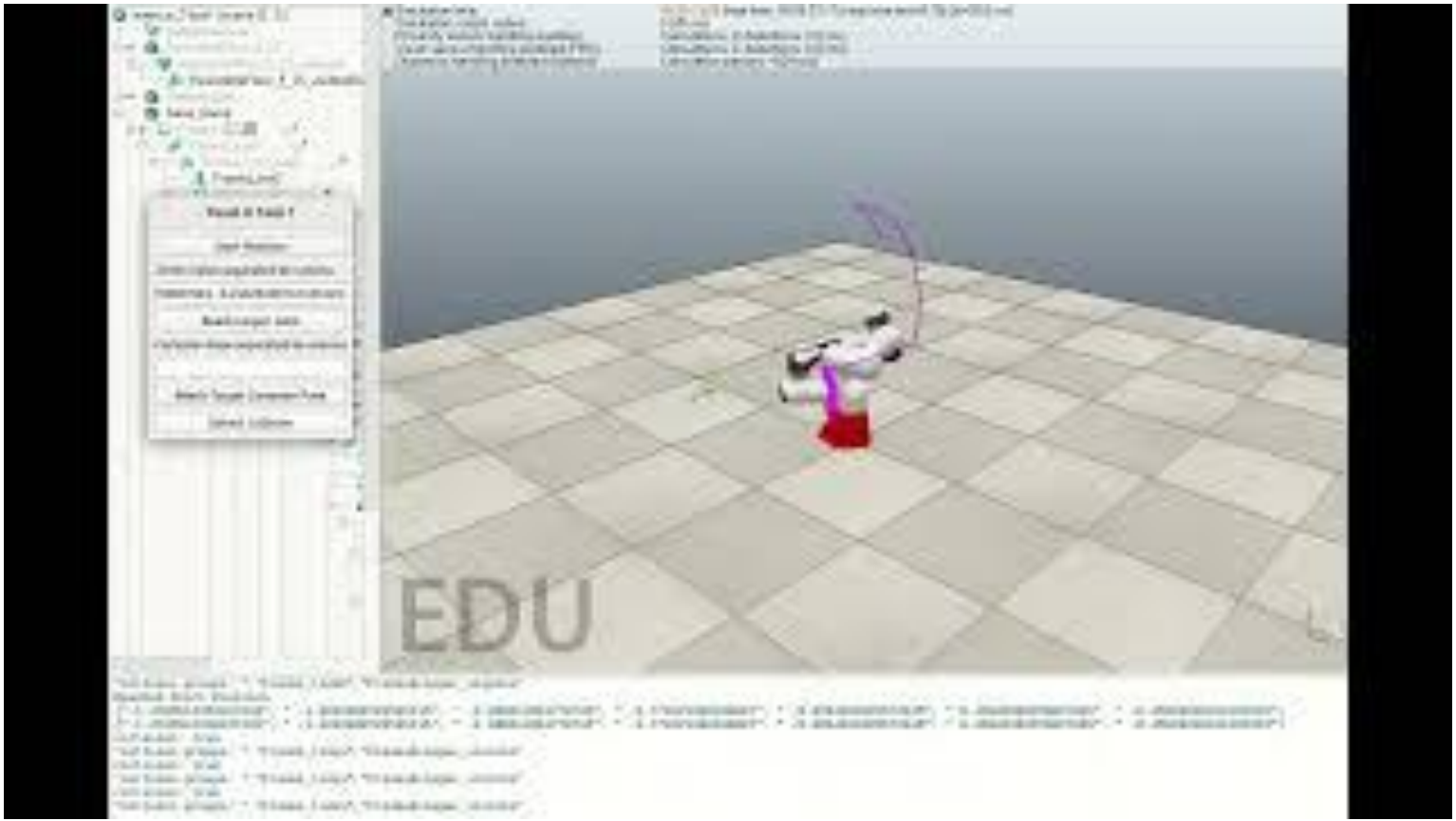
Environment: CoppeliaSim with Lua

1. simIK plugin for kinematics
2. simOMPL library for path planning without collision

Task 2.2, 3.1, 3.2

Pyrep

Task 1.1: Check self-collision from specific joint configuration



Farhan

Self Collision Identification

- ➔ Make collision collections set out of robot parts
- ➔ Remove adjoining elements as they always are in contact
- ➔ Make Collision Pairs out of the Collision Collection Set
- ➔ Check for collision between Collision Pairs using sim.checkCollision
- ➔ Colour the collided links

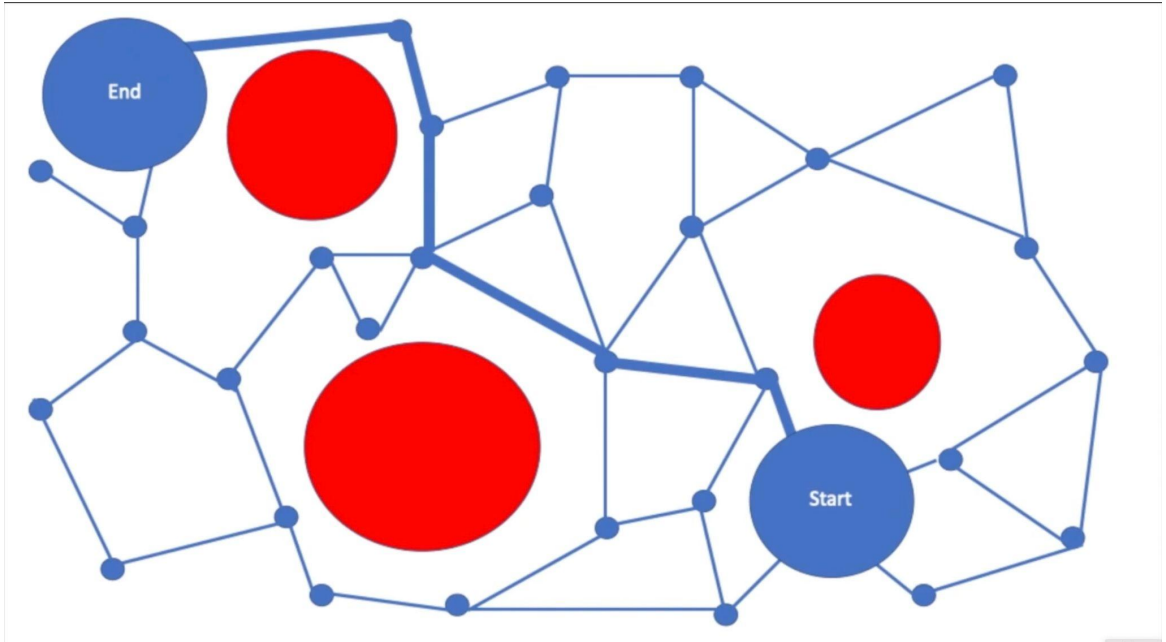
Task 1.2: Identify and avoid self-collision from given joint and Cartesian trajectories

Task 1.2 avoid self-collision from given joint trajectories

1. Check if the target pose is reachable using CoppeliaSim's inverse kinematics solver.
2. Check for self-collision using CoppeliaSim's collision detection function.
3. Modify the trajectory to avoid self-collision.
4. Verify the modified trajectory.
5. Repeat steps 2-4 until a valid trajectory is found.

Task 1.2 avoid self-collision from given joint trajectories

PRM (Possibilistic Map Road)



PRM:points are connected to nearest K Neighbors to create a graph structure.

Benefit: it can be queried repeatedly for paths without having a construct in new graph

Task 1.2 avoid self-collision from given joint trajectories

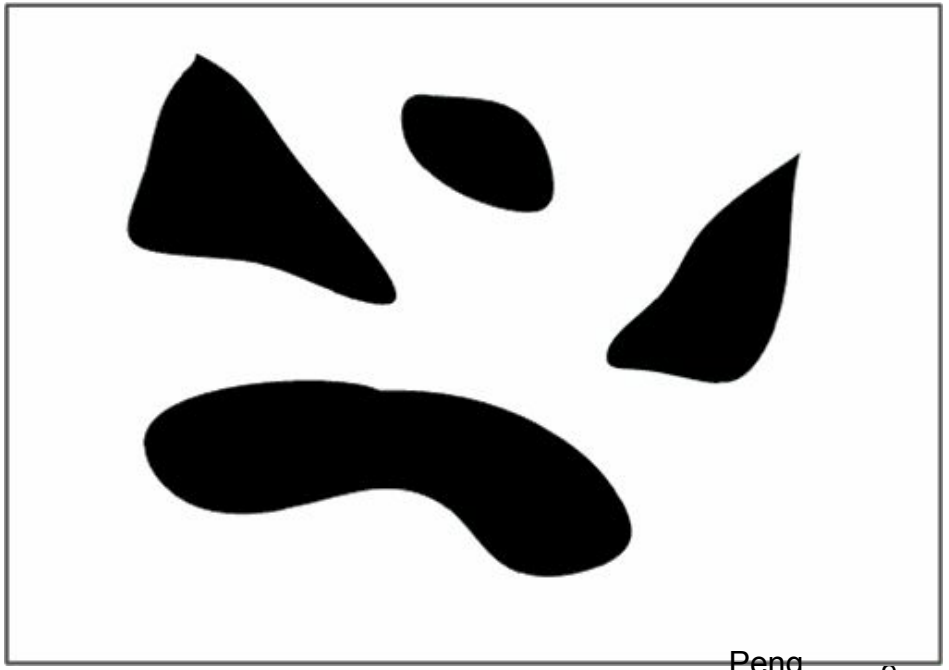
The relation between PRM and sparsity

PRM is particularly effective in high-dimensional spaces where there are many obstacles and the robot's motion is subject to constraints.

PRM is effective, because it is possible to represent a complex configuration space with fewer samples, while still preserving the essential structure of the space which is called sparsity in compressive sampling.

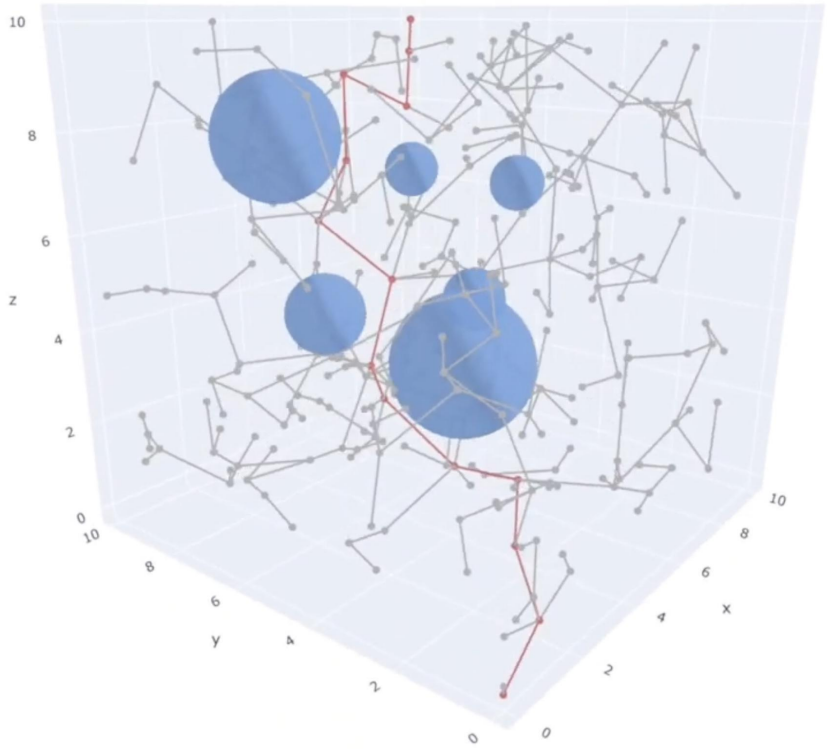
And PRM's measurement matrix always has good RIP(Restricted Isometry Property) so it is highly possible to acquire fewer samples than the Nyquist-Shannon sampling theorem would require.

It is proven that the random matrix has good RIP with high possibility.

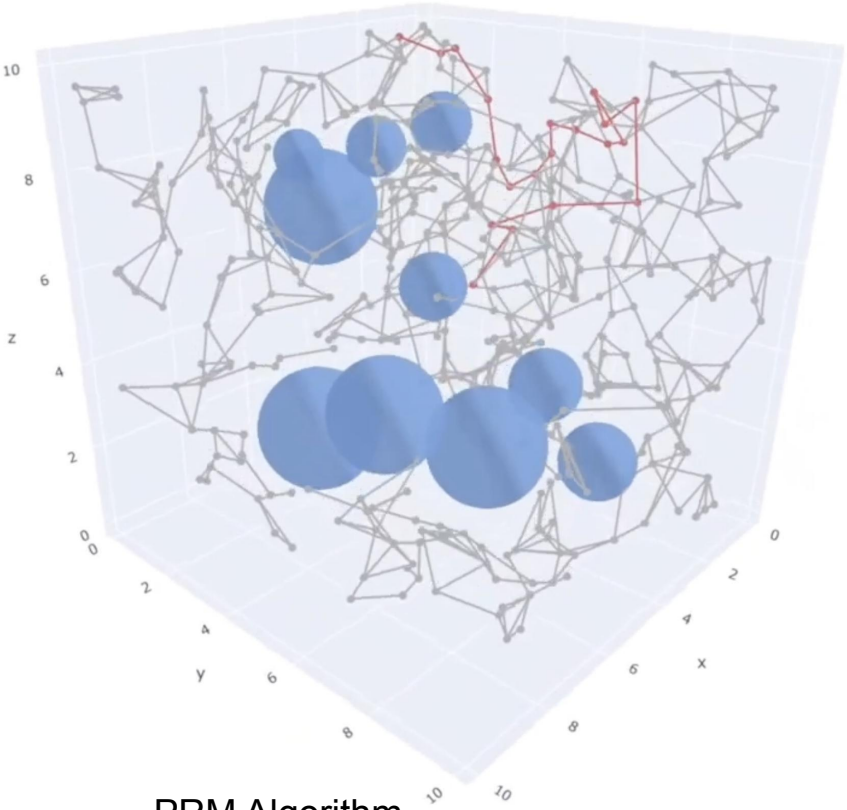


Task 1.2 avoid self-collision from given joint trajectories

RRT vs PRM



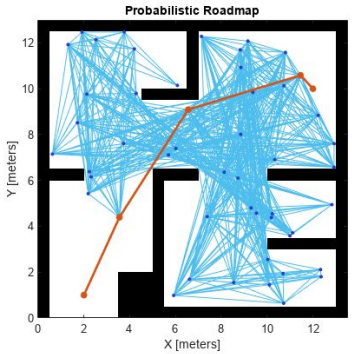
RRT Algorithm



PRM Algorithm

Task 2.1 End-effector trajectory planning

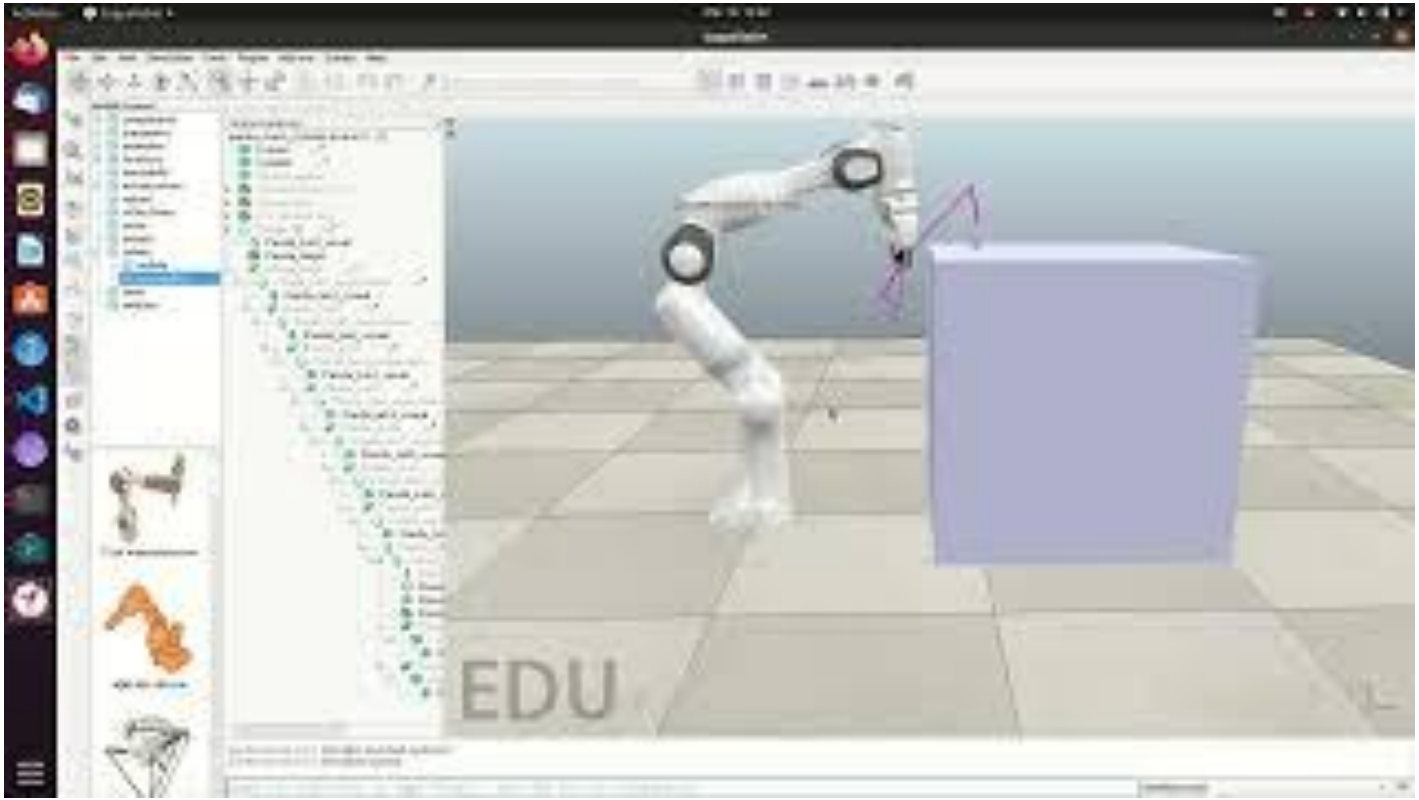
- Lazy PRM based path planning.



The graph is constructed based on probabilistic sampling of configurations, and is connected to form a roadmap.(Ik sampling)

Finding optimal path based of A* star.(just checking connected paths)

Task 2.1 a Simulation Environment without obstacle.



Task 2.1 b Simulation Environment with obstacle.

Task 2.2 Joint obstacle avoidance - exploiting null space

Nullspace: the region of state space where there is a redundancy of solutions.

$$\boldsymbol{\sigma}(t) = \boldsymbol{f}(\boldsymbol{q}(t))$$

where $\boldsymbol{\sigma}(t) \in \mathbb{R}^m$ is the task variable to be controlled, $\boldsymbol{q}(t) \in \mathbb{R}^n$ is the vector of the system configuration.

$$\dot{\boldsymbol{\sigma}}(t) = \frac{\partial \boldsymbol{f}(\boldsymbol{q}(t))}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}}(t) = \boldsymbol{J}(\boldsymbol{q}(t))\dot{\boldsymbol{q}}(t)$$

Least Squares Solution to Pursue Minimum-Norm Velocity: $\dot{q}_{\mathrm{des}} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{\sigma}}_{\mathrm{des}} = \boldsymbol{J}^{\mathrm{T}}\left(\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}}\right)^{-1}\dot{\boldsymbol{\sigma}}_{\mathrm{des}}$

Nullspace Projector: $P_{null} = (I_n - J^{\dagger}J)$, where n is the dimension of the task space, $\boldsymbol{J}^{\dagger}$ is the pseudo-inverse of Jacobian ***J***.

In Case of Redundancy: $\dot{q}_{des} = J^{\dagger}\,\dot{\sigma}_{des} + P_{null}\,\dot{q}_{null}$

There are still two variables remained to be designed: $\dot{\boldsymbol{\sigma}}_{\mathrm{des}}$ and $\dot{\boldsymbol{q}}_{\mathrm{null}}$.

1) Idea to design $\dot{\boldsymbol{\sigma}}_{\mathrm{des}}$: proportional gain of error.

$\dot{\sigma}_{des} = -k_p\,(current\ position\ -\ desired\ position)$

=> To keep the end-effector staying!

2) Idea to design $\dot{\boldsymbol{q}}_{\mathrm{null}}$: The closer the obstacle is, the more conservative the joints move.

=> End effector stays at the same position, while the joints move away from the obstacle.

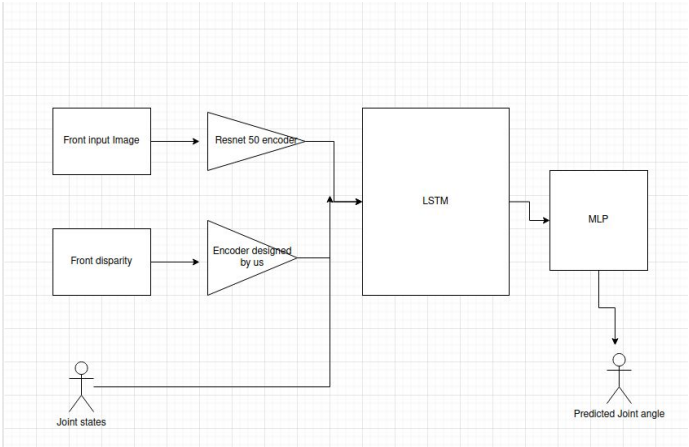
$$\dot{q}_{des} = J^{\dagger}\,\dot{\sigma}_{des} + P_{null}\,\dot{q}_{null}$$

Antonelli, Gianluca. "Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems." *IEEE Transactions on Robotics* 25.5 (2009): 985-994.

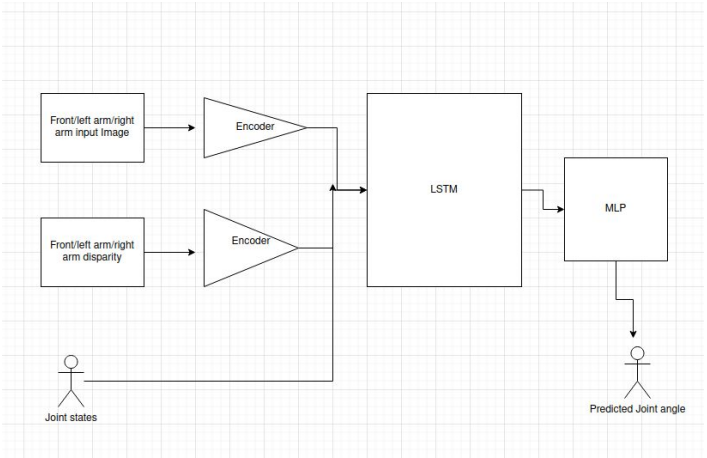
Task 2.2 Joint obstacle avoidance - exploiting null space

Task 2.3 Imitation learning

-> During the experiment, we selected **representation based predictive multimodal learning and our loss function was mean square error**. (Difference between predicted joint state and joint state one step later)

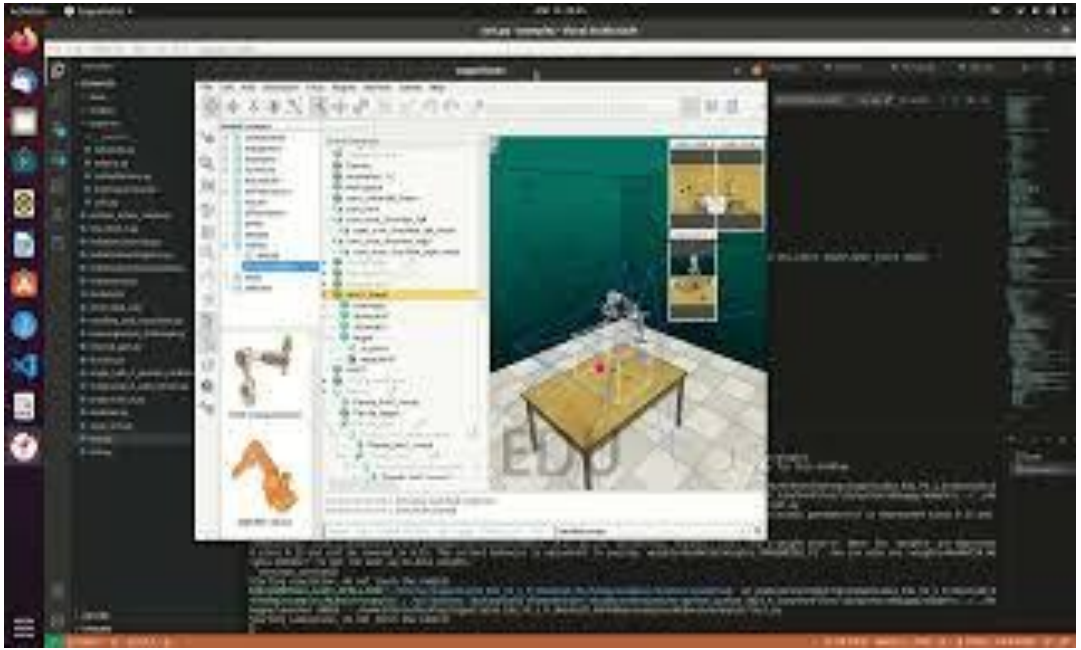


Case 1: Big latent representation.



Case 2:Small latent representation

Dataset collection for 10 different grasping and learning video.



2.3 Experimental result

Case1(small latent representation): 10000 epoch (40 action different reach task) : loss = %0.004

Case2(big latent representation): 10000 epoch (40 action different reach task) : loss = %0.043

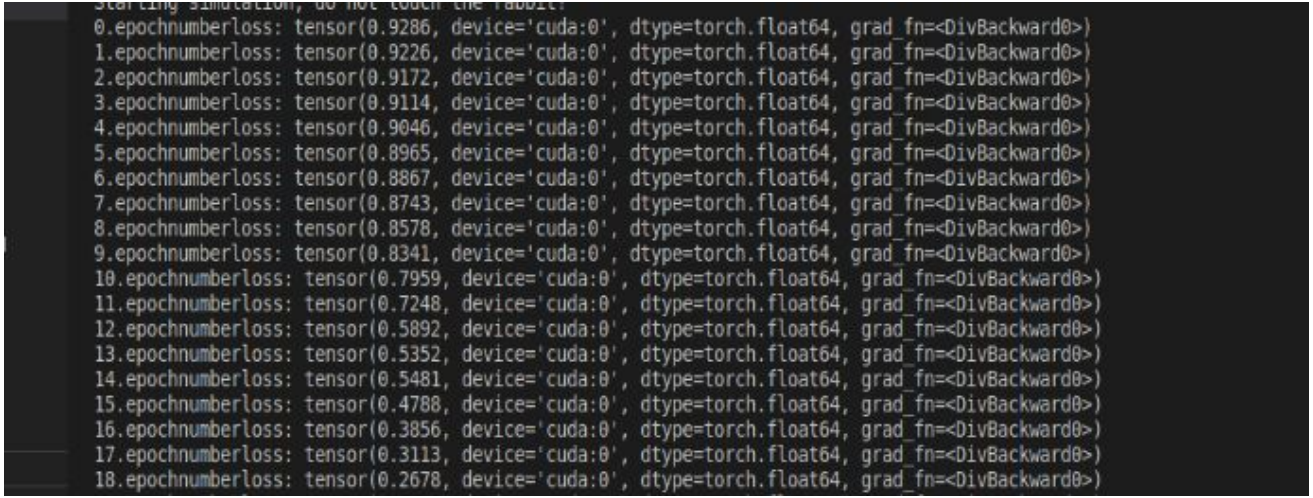
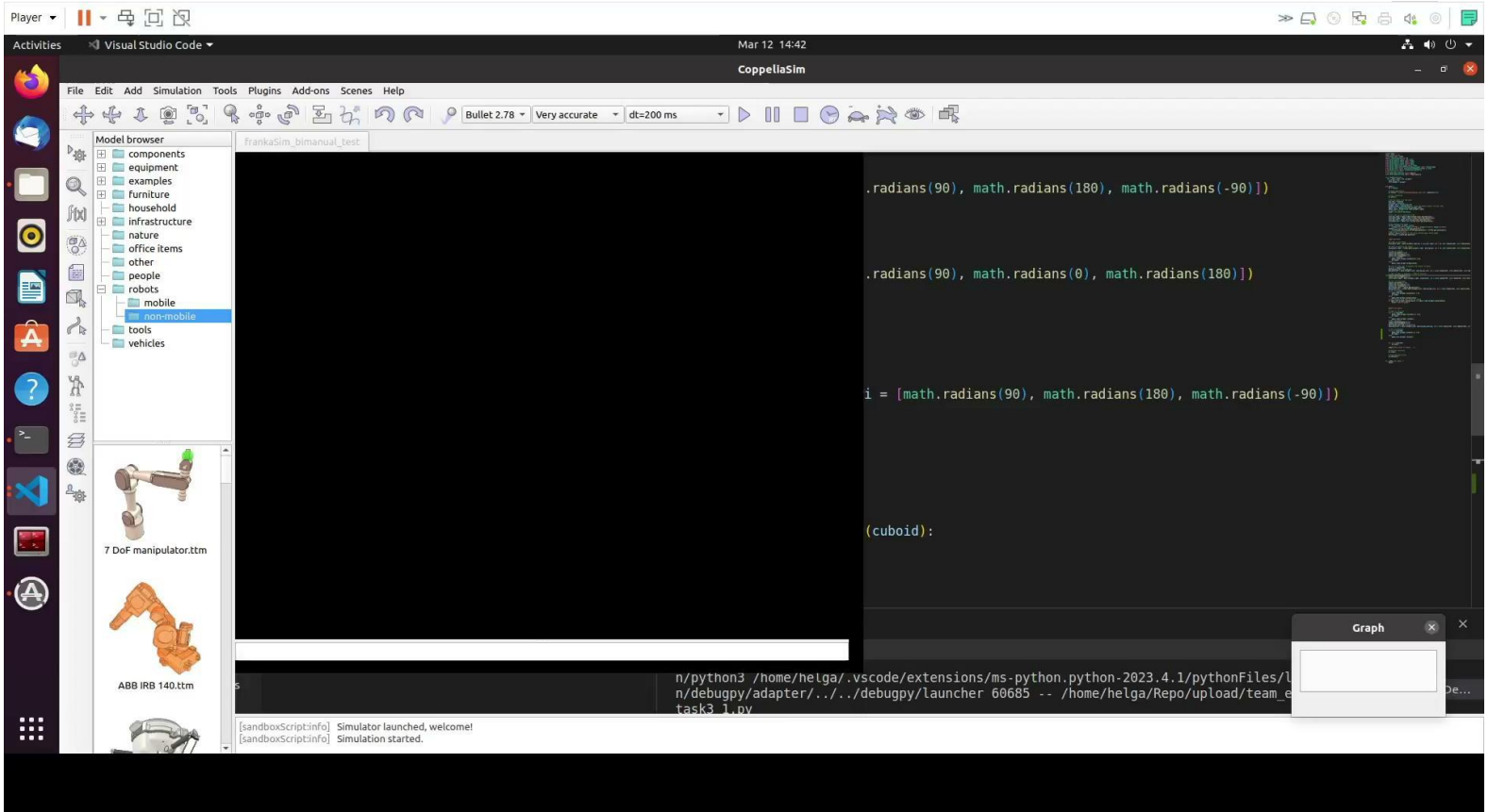


Figure: Constantly decreasing loss function.

Task 3.1 End-effector trajectory planning

```
from pyrep.const import ConfigurationPathAlgorithms
```



Expansive-Spaces

Tree planner

Task 3.2 Object manipulation - kinematic control architecture for cooperative and collaborative multi-robot systems

A two-layer architecture for kinematic control of cooperative and collaborative multi-robot systems.

Cooperative Task Space:
End-effector pose of joint i: $\boldsymbol{x}_i = \left[\boldsymbol{p}_i^T \;\; \boldsymbol{\phi}_i^T\right]^T$

Absolute frame & relative motion:

$$p_a = \frac{1}{N}\sum_{i=1}^{N} p_{a,i}\,,\ \phi_a = \frac{1}{N}\sum_{i=1}^{N}\phi_{a,i} \qquad x_{\boldsymbol{a}} = \begin{bmatrix} p_{\boldsymbol{a}} \\ \phi_{\boldsymbol{a}} \end{bmatrix}, \qquad x_{\boldsymbol{r},i} = \begin{bmatrix} p_{\boldsymbol{r},i} \\ \phi_{\boldsymbol{r},i} \end{bmatrix}$$
$$p_{r,i} = p_{i+1} - p_i\,,\ \phi_{r,i} = \phi_{i+1} - \phi_i$$

$$\boldsymbol{x}_a = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i = \boldsymbol{J}_a\boldsymbol{x}$$

$$J_a = \frac{1}{N}(\mathbf{1}_N^T \otimes \boldsymbol{I}_6) \Rightarrow J_a = 0.5[\boldsymbol{I}_6 \quad \boldsymbol{I}_6] \Rightarrow x_a = J_a x$$
$$J_r = [-\boldsymbol{I}_6 \quad \boldsymbol{I}_6] \Rightarrow x_r = J_r x$$

First Layer: to compute the reference end-effector trajectories for each arm.

$$\dot{x}_d = J_a^{\dagger}\left(\dot{x}_{a,d} + k_a^P * e_a\right) + J_r^{\dagger}\left(\dot{x}_{r,d} + k_r^P * e_r\right)$$

$$e_* = x_{*,d} - J_* x\,, where\ * = a, r$$

Here are two parameters $\mathbf{k}_r^{\mathbf{P}}$ and $\mathbf{k}_a^{\mathbf{P}}$ to be designed.

Second Layer: computation of the joint trajectories.

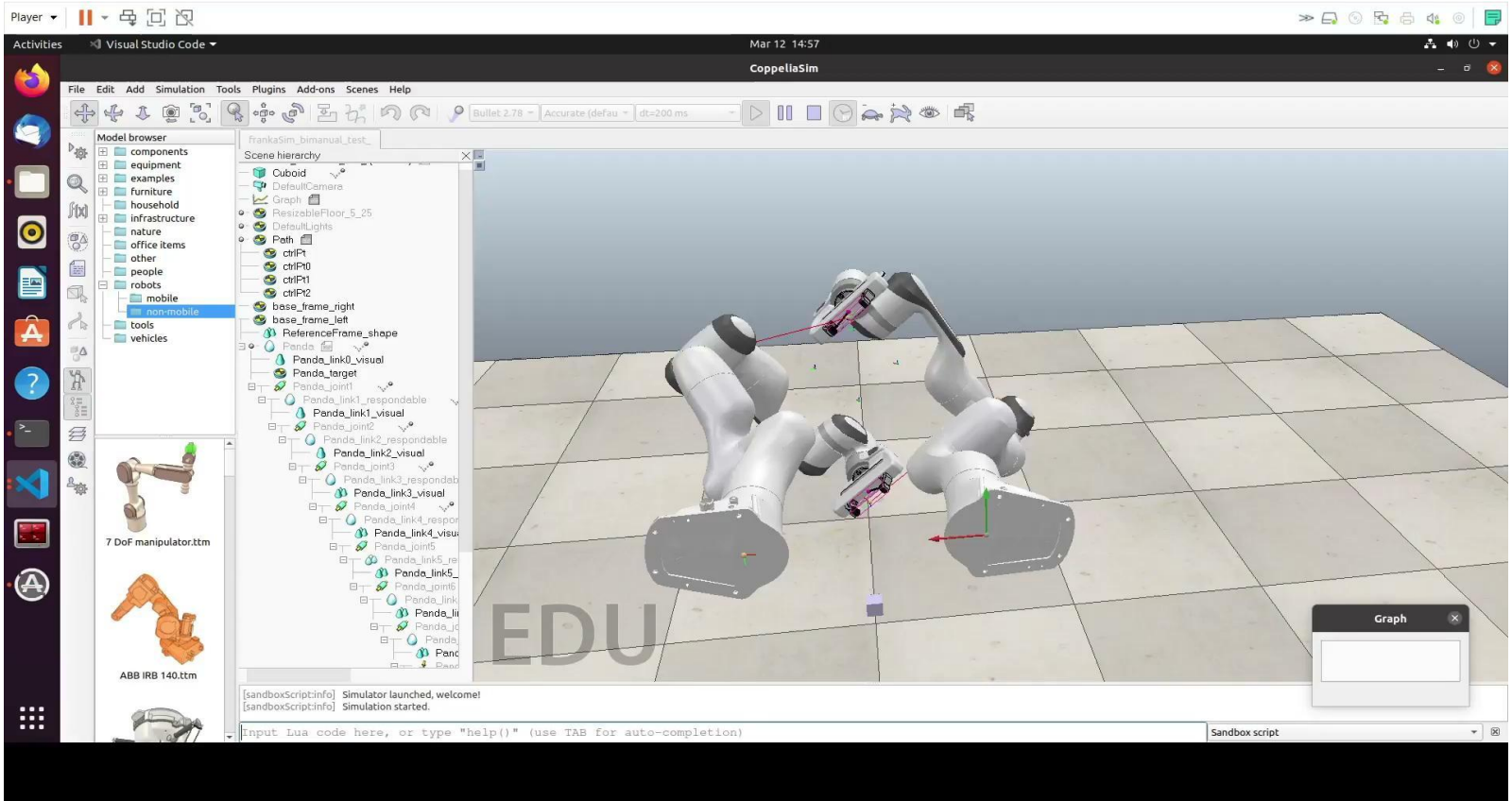
$$\dot{q}_{di} = J_i^{\dagger}(q_{di})(\dot{x}_{di} + K_i * e_i)$$

$$e_i = x_{di} - x_i$$

Consider the state of the different robotic arms as a whole.
Kinematic control: the control problem is solved into two stages: the desired task-space trajectory is transformed, via inverse kinematics, into the corresponding joint trajectories, which then constitute the reference inputs to some joint space control scheme.
Thus, allowing the problem of kinematic singularities and/or redundancy to be solved separately from the motion control problem.

Basile, Francesco, et al. "A decentralized kinematic control architecture for collaborative and cooperative multi-arm systems." Mechatronics 23.8 (2013): 1100-1112.

Task 3.2 Object manipulation - decentralized kinematic control architecture



Thank you for your attention