Final Project

Team_9 Zhanqian Wu, Jie Mei, Enlin Gu, Jiani Wang
University of Pennsylvania

Penn Engineering
University of Pennsylvania

Contents

Plan

Methodology

Results & Discussion

Penn Engineering

Coordinate transformation

① **Camera frame to End-effector frame transformation**

$$P_e = T_c^e \cdot P_c$$ *get_H_ee_camera(self)*

② **End-effector frame to world frame transformation**

$$P_w = T_e^w \cdot P_e$$

$$T_e^w = T_e^b \cdot T_b^w = T_1^b \cdot T_2^1 \cdot T_3^2 \cdots T_e^6 \cdot T_b^w$$

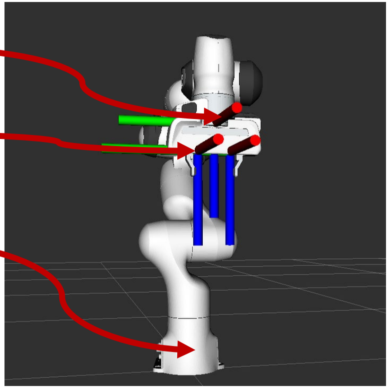
Where a homogeneous transformation matrix from frame i to i-1 is:

$$T_i^{i-1} = \begin{bmatrix} c_{\theta i} & -s_{\theta i}c_{\alpha i} & s_{\theta i}s_{\alpha i} & a_i c_{\theta i} \\ s_{\theta i} & c_{\theta i}c_{\alpha i} & -c_{\theta i}s_{\alpha i} & a_i s_{\theta i} \\ 0 & s_{\alpha i} & c_{\alpha i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Frame

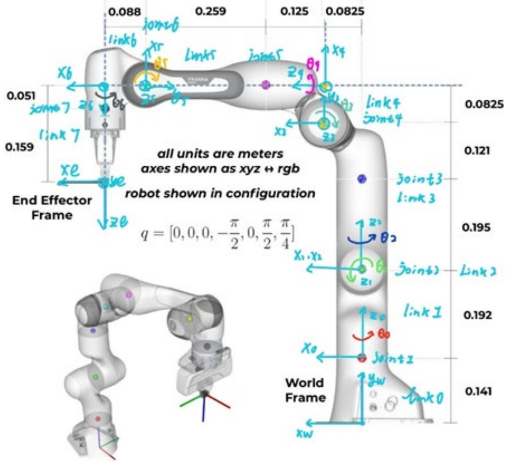
End effector Frame

World Frame



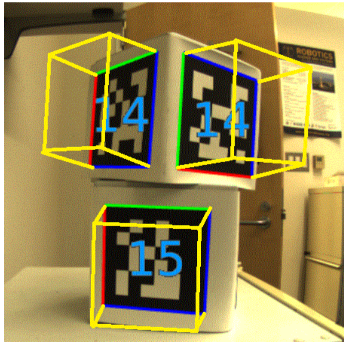
D-H

Link	a_i	α_i	d_i	θ_i
0	0	0	0.141	0
1	0	$-\frac{\pi}{2}$	0.192	q_0
2	0	$\frac{\pi}{2}$	0	q_1
3	0.0825	$\frac{\pi}{2}$	0.316	q_2
4	0.0825	$\frac{\pi}{2}$	0	$\frac{\pi}{2} + (q_3 + \frac{\pi}{2})$
5	0	$-\frac{\pi}{2}$	0.384	q_4
6	0.088	$\frac{\pi}{2}$	0	$(q_5 - \frac{\pi}{2}) - \frac{\pi}{2}$
7	0	0	0.21	$q_6 - \frac{\pi}{4}$

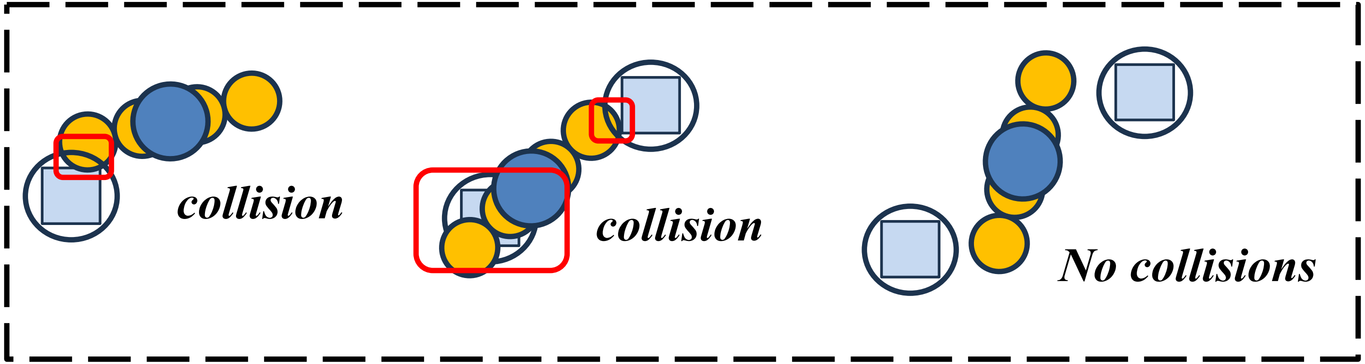


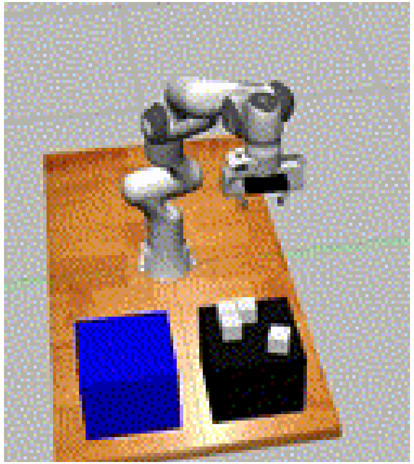
③ **AprilTag Detection**

AprilTag is a type of fiducial marker system used in computer vision and robotics for pose estimation and object tracking.



2D Collision Detection





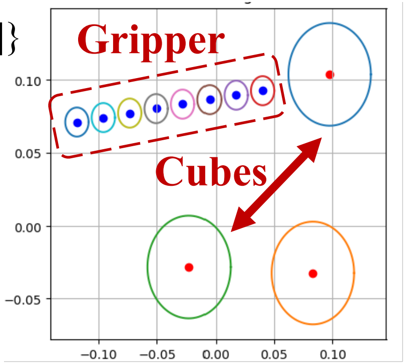
Real-time Navigation Map Structure:

{**Cube 1~n**: [*x,y,Radius, Orentation*], **Gripper**: [*x,y, Orentation*]}

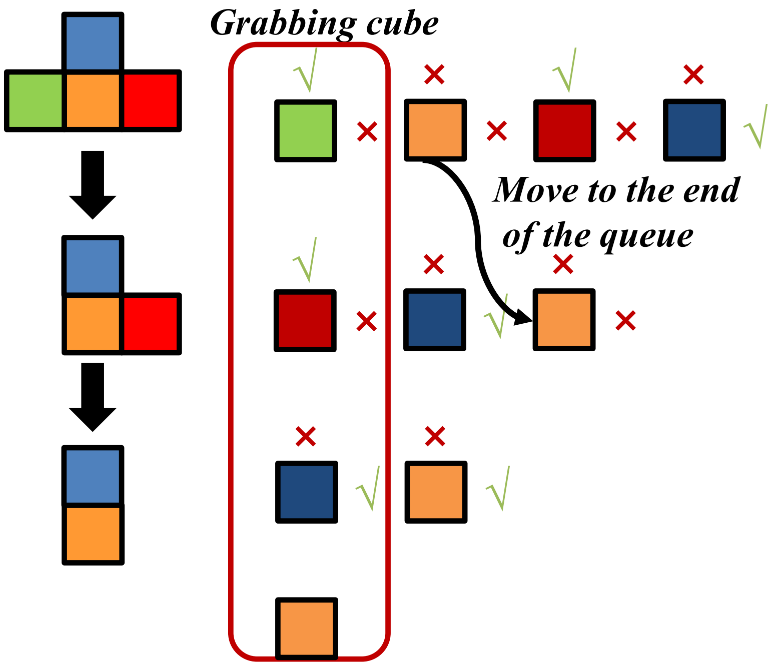
Collision Detection → Intersecting circles Detection

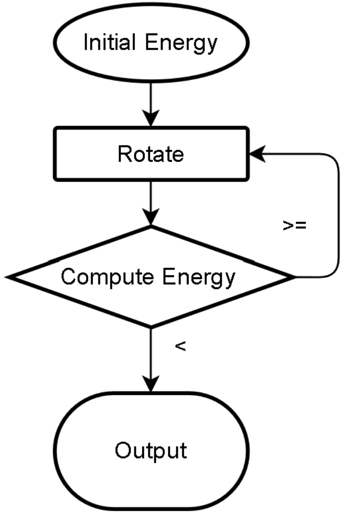
```
for center1 in obstacle_map:
    for center2 in gripper:
        distance = sqrt((center1[0] - center2[0]) ** 2 + (center1[1] -
center2[1]) ** 2) radius_sum = radius1 + radius2
        if distance <= radius_sum:
            count +=1
```

Real-time Navigation Map



Sequencing Algorithm & Gripper orientation

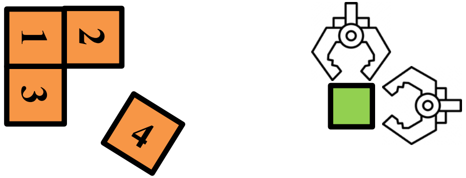




$$\text{Cost} = \mu \frac{1}{2}\left\| q_{current} - q_{target} \right\|$$

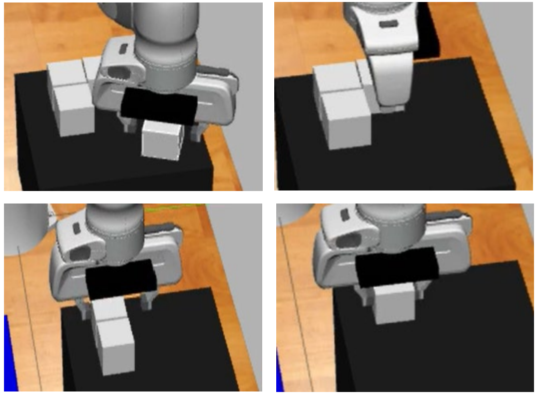
Where:

- *u=1 if collision free*
- *u=-1 if collision*



Without sequencing:
{1-2-3-4}→Fail

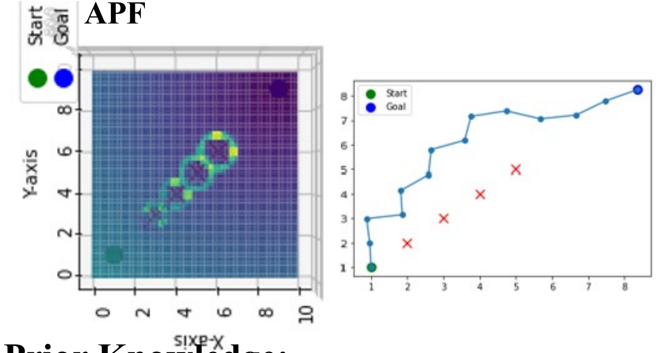
With sequencing:
{4-2-3-1}→Success



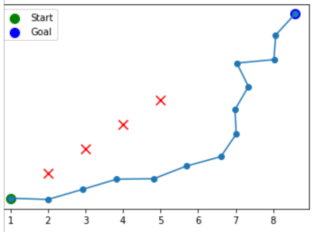
With the sorting algorithm, we were able to achieve a more efficient task.

Path Planning

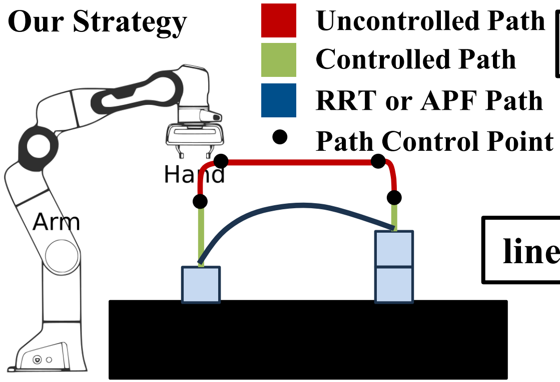
APF



RRT



Our Strategy



Path Planning

Control Point

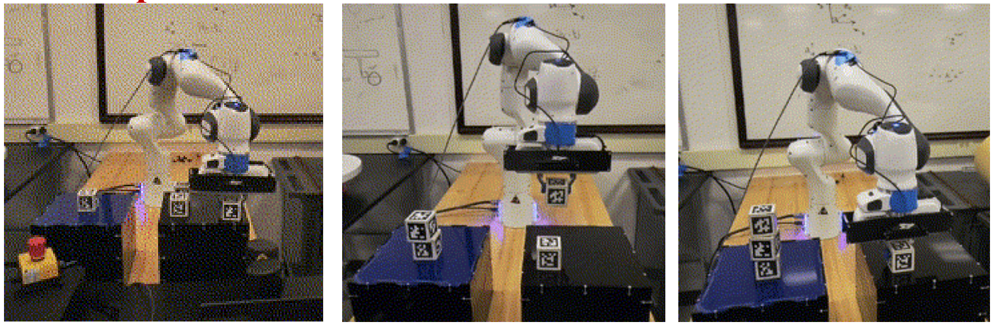
linear interpolation

Prior Knowledge:

① The longer the path, the longer the planning time

② Most collisions occur during the grab and drop process

We only need path planning for the initial and final stage to save computation time



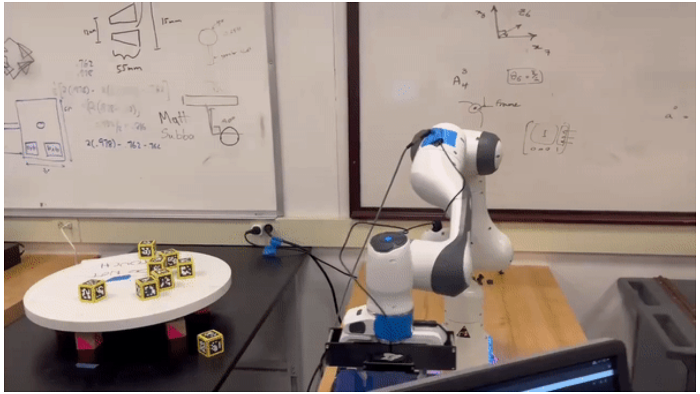
10 Random Tests:

	Average Planning Time	*Average Execution Time*	*Total Time*
APF	5.17↑	9.32↓	14.49
RRT	6.48↑	10.61	17.09↑
Our's	0.89↓	10.87↑	11.76↓

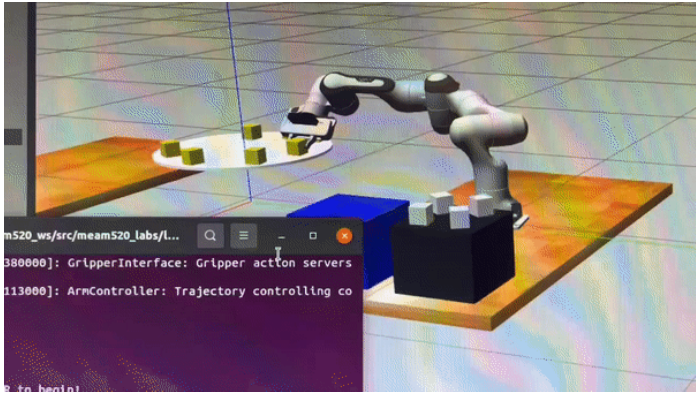
With this path planning strategy, we were able to achieve a more efficient task.

① Test Environment: CPU:I9-13900H, RAM 32G, Ubuntu 18.04

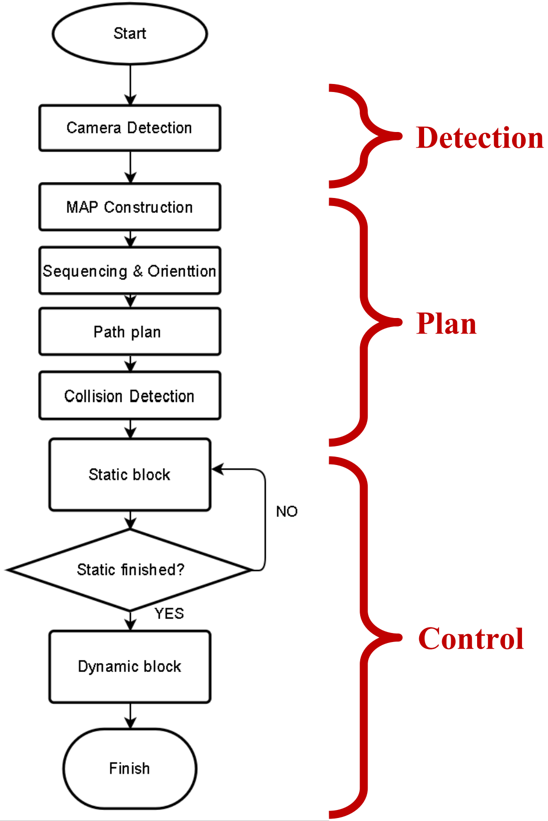
Dynamic Tasks Video Demo & Architecture



Initial Grabbing Pose



Reviewed Grabbing Pose

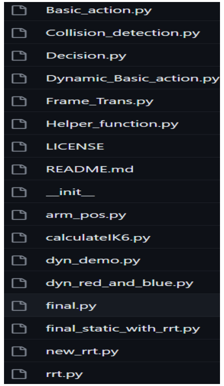


Overall Planning and Control Architecture

Test & Evaluation Plan

Code

- Divide task into Red, Blue, Static and Dynamic block catching
- Write grab / place and other basic actions into functions
- Check each function separately



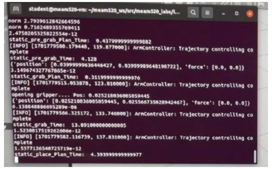
Code test → **Function test**

Error & Warning?

Working properly?

Simulation Testing Plan

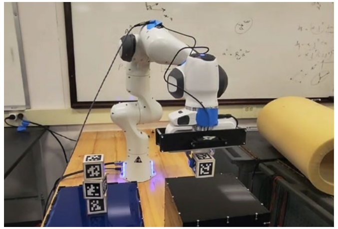
- Print how much time is used for each task
- Print matrix & joint angle to check state
- Report issues when occur



Subsystem test → **Static Block test** / **Dynamic Block test** → **Integrated Test**

Real Lab testing Plan

- Check if our points are correct & adjust
- Check if our code work for different block pose



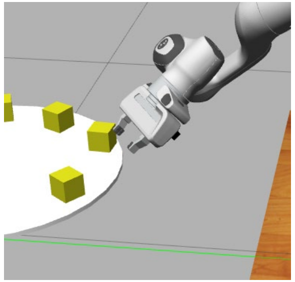
Metrics:

- Average time (s), success rate (%)

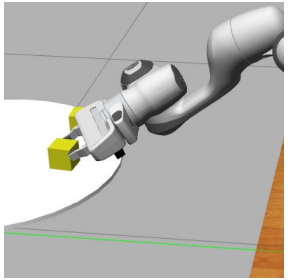
Dynamic Block Preliminary Outline (Test Plan)

- Ideal Approach:
  - Detect the blocks.
  - Solve the blocks' poses in robot's world frame.
  - Make a prediction on the moving blocks placed on the turntable.
  - Move the arm to the predicted grabbing position.
  - Pick and place the blocks.
- Actual Implementation
  - Set up and calculate pre-grab and grab position.
  - Solve for the corresponding poses using IK.
  - Move to the pre-grab then to the grab position.
  - Once the grab position is reached, continuously open/close the grip until a block is grabbed successfully.
  - Move to a safe pre-place position that won't influence the stack with collision detection.

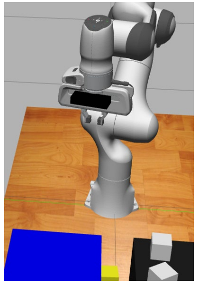
```
while static task is done and ROS is active:
    Close the gripper
    if block is grabbed successfully:
        Move arm to pre-place position
        Perform dynamic placing of the block
        Exit the inner loop
    else:
        retry
    Open the gripper
    Print "loop␣grip␣opened"
Move to pre-grab position
Increment the stack layers before the next pick
```



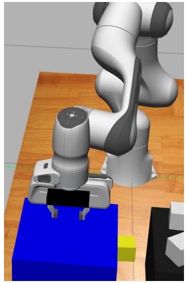
Pre-grab pose



Grab pose



Pre-place pose



Place pose

Dynamic Block Preliminary Tests

- Performance Metrics:
  - Position of the four pre-define grab and place positions. Mismatches needs to be considered in the physical testing.
  - Time Needed to complete each cycle of picking and placing the dynamic block.
  - Rate of success in grabbing the real setting (limited environment in the simulator).
  - Red VS Blue discrepancy.

- Next Steps:
  - Find efficient seeds for solving IK.
  - Maximize the successful rate by choosing and testing different positions.
  - Integrate with the static tasks.
  - ...

	Poses Mismatch (m)	Time before grab (s)	Grabbing Time (s)	Placing Time (s)	Success Rate
Avg.	+- 0.18	4.82	TBD	9.14	TBD

Thank you for your time!

Team 9

Penn Engineering
UNIVERSITY of PENNSYLVANIA