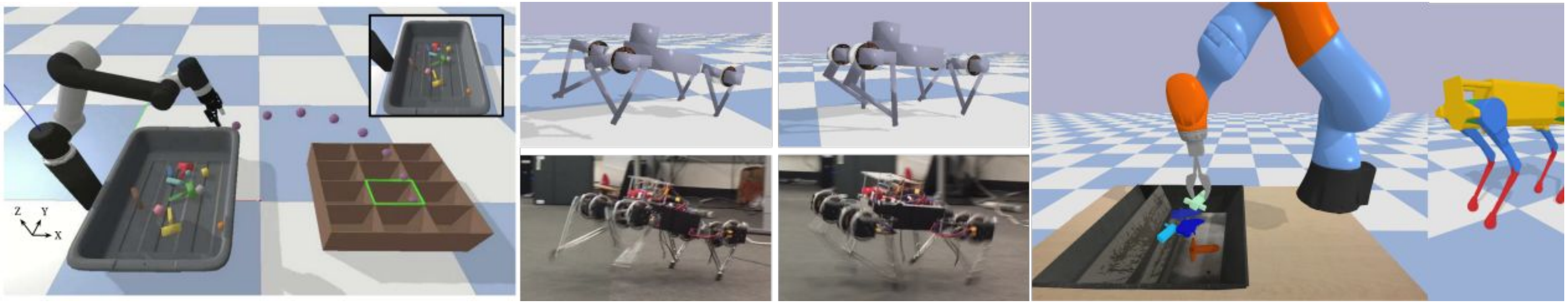
HW3 Review

Reminders

- Due Date:
  - It’s due on Thursday **April 7th** at 5:00 PM EST.
  - Late submission policy is on the courseworks website.
  - Extra credits for submitting 48 hours early.
- Submissions:
  - Submit on courseworks.
  - Submit all and only the requested files as listed on the handout.

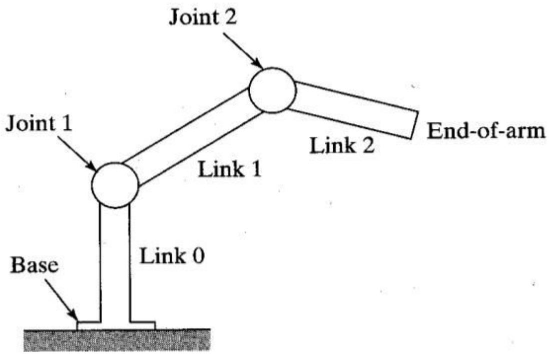
Preliminaries: Pybullet

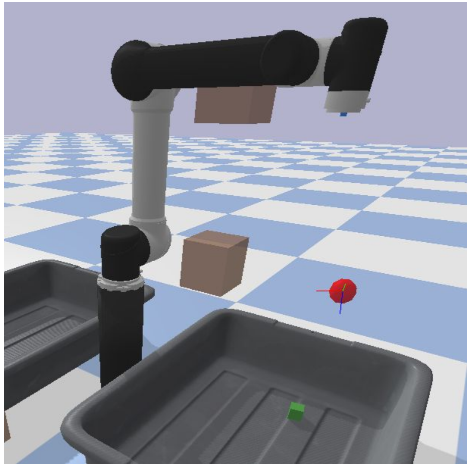
- Real-Time Physics Simulation
- QuickStart Guide



Preliminaries: UR5 Robot

- Links, Joints (revolute, prismatic, fixed)
- # of Joints = # of Degrees of Freedom (DoF)
- Robot movement: movement of the joints
- Specified by a URDF file
- UR5: 6 revolute joints (range ± 360°)





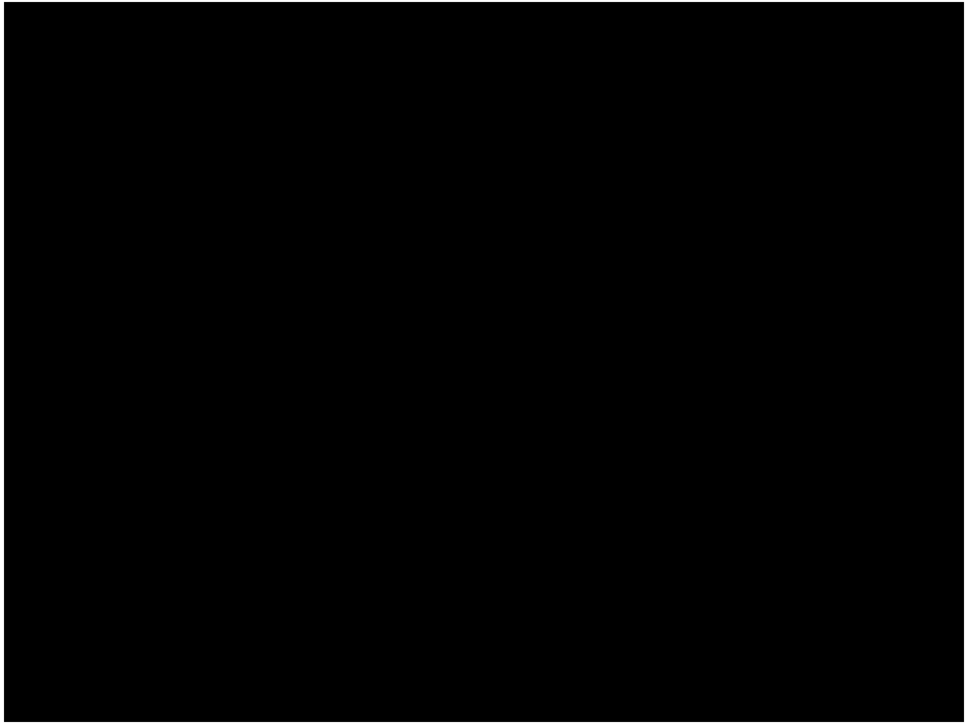
Problem 1 Basic robot movement

TODOS: sim.py

- Given a target position and orientation of the robot's end effector, compute the values for all UR5 joint angles to reach them.

Hints (function and attributes you may need):

- p.calculateInverseKinematics
    - Optimize over joint configurations to reach the target pose; **tune the optional parameters to pass the tests**
- PyBulletSim.robot_body_id
- PyBulletSim.robot_end_effector_link_index



Problem 2 Grasping

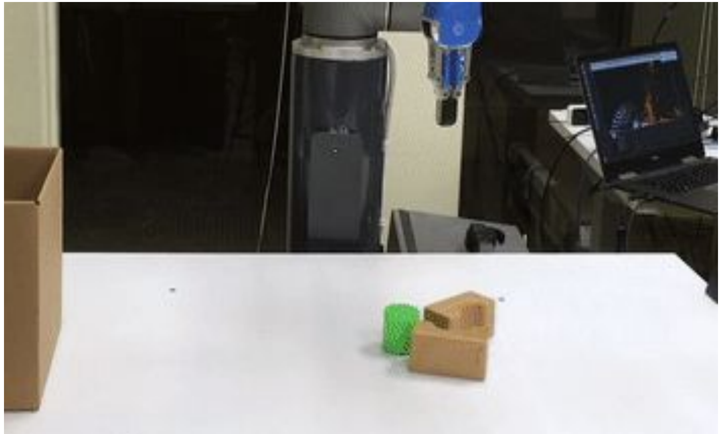
TODOS: sim.py, main.py

- Get position and orientation of the object (as the pick pose)
- Execute a pick and place action primitive

Review: Quaternion and Euler angles (lec1+2)

Hints (some functions you may need):

- p.getBasePositionAndOrientation
- p.getEulerFromQuaternion
- PyBulletSim.open_gripper
- PyBulletSim.close_gripper
- PyBulletSim.move_tool
- PyBulletSim.robot_go_home



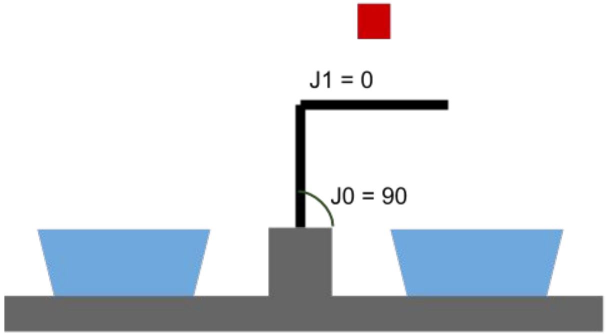
Top Down Grasping: Only need (x,y,z) and yaw (rotation about z axis)

Problem 3 RRT
(Rapidly-exploring random tree)

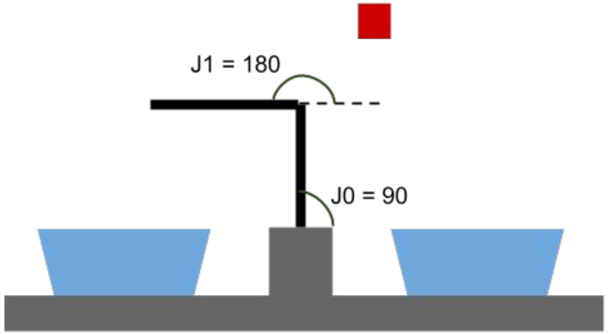
```
Algorithm rrt
    Input:
    - q_init: initial configuration
    - q_goal: goal configuration
    - MAX_ITERS: max number of iterations
    - delta_q: steer distance
    - steer_goal_p: probability of steering towards the goal
    Output:
    - path

    V <- {q_init}; E <- {}
    for i = 1 to MAX_ITERS
        q_rand <- SemiRandomSample(steer_goal_p) # with steer_goal_p
           ↪ probability, return q_goal. With (1 - steer_goal_p),
           ↪ return a uniform sample
        q_nearest <- Nearest(V, E, q_rand)
        q_new <- Steer(q_nearest, q_rand, delta_q)
        if ObstacleFree(q_nearest, q_new):
            V <- Union(V, {q_new})
            E <- Union(E, {(q_nearest, q_new)})
            if Distance(q_new, q_goal) < delta_q:
                V <- Union(V, {q_goal})
                E <- Union(E, {(q_new, q_goal)})
                path <- calculate the path from q_init to q_goal
                return path
    return None
```

Problem 3 - RRT 2D Example

delta_q = 10



Initial



Goal

Tree

Propose Random

Nearest Node

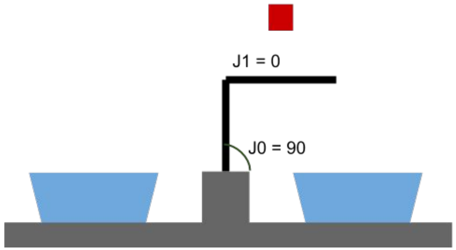
Steer

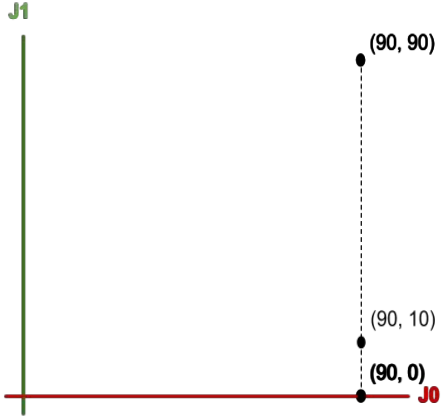
Check Collision

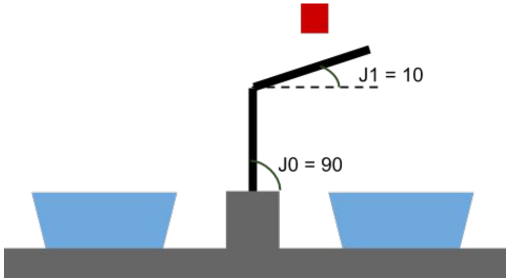
Add to tree

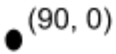
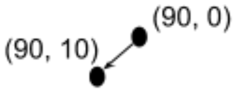
Problem 3 - RRT 2D Example

delta_q = 10



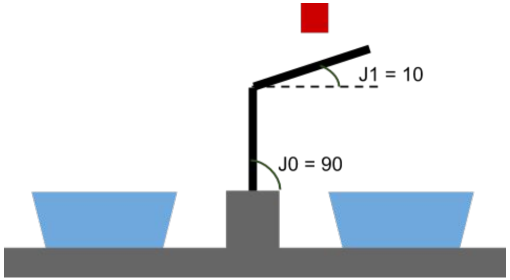


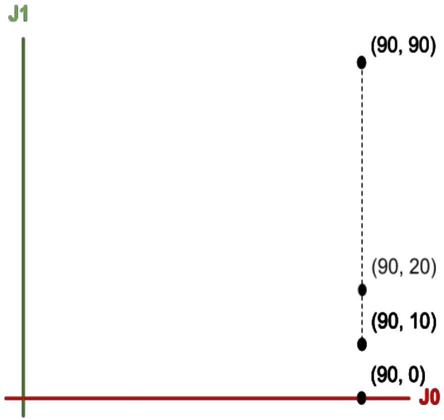


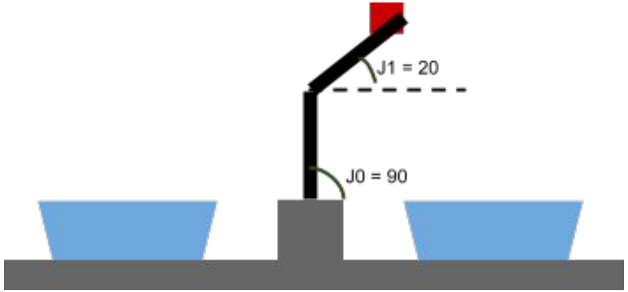
Tree	Propose Random	Nearest Node	Steer	Check Collision	Add to tree
	(90, 90)	(90, 0)	(90, 10)	None	

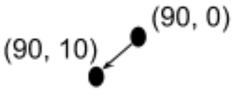
Problem 3 - RRT 2D Example

delta_q = 10



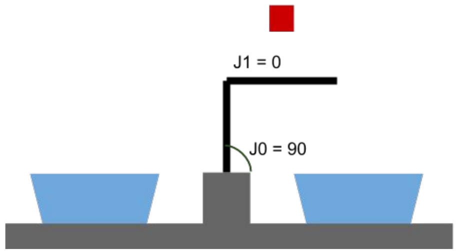


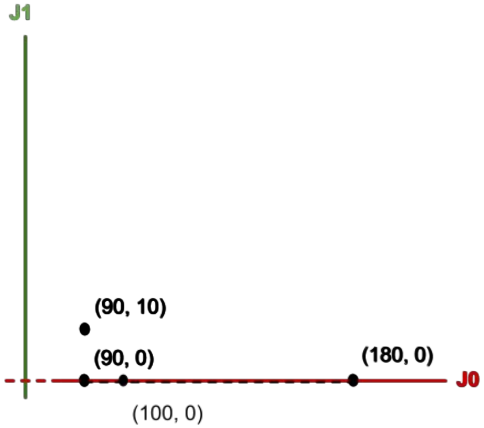


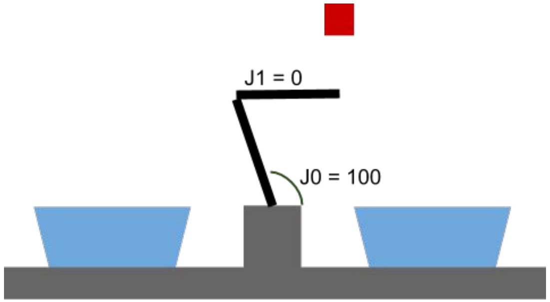
Tree	Propose Random	Nearest Node	Steer	Check Collision	Add to tree
	(90, 90)	(90, 10)	(90, 20)	Yes	

Problem 3 - RRT 2D Example

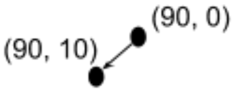
delta_q = 10





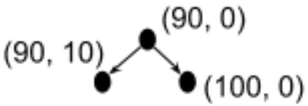


Tree



Propose Random	Nearest Node	Steer	Check Collision
(180, 0)	(90, 0)	(100, 0)	No

Add to tree



Problem 3 RRT
(Rapidly-exploring random tree)

- State space
  - 6DoFs of UR5, each DoF has range [-2π, 2π]
  - View them as 6D vectors
  - May be helpful to implement as numpy arrays
- Distance metric
  - Possible choices, $p, q \in [-2\pi, 2\pi]^6$
    - $$d_1 = ||p - q||_1 = \sum_{i=1}^{6} |p_i - q_i|$$
    - $$d_2 = ||p - q||_2 = \sqrt{\sum_{i=1}^{6} (p_i - q_i)^2}$$
    - $$d_3 = \sum_{i=1}^{6} \min(2\pi - |p_i - q_i|, |p_i - q_i|)$$
  - Note: be cautious with units (radians or degrees), and modify ***delta_q*** accordingly
- Steer $$q_{new} = q_{near} + (q_{rand} - q_{near}) * \frac{\delta_q}{d(q_{rand}, q_{near})}$$

Problem 3 RRT
(Rapidly-exploring random tree)

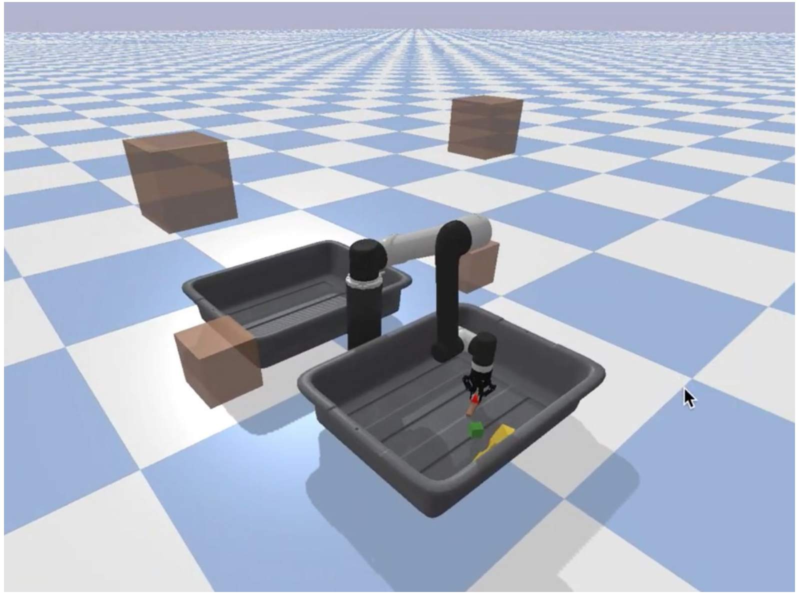
- Check obstacle free: ***check_collision()***
  - Technically, should check every point along the path
  - For this assignment, we will assume ***delta_q*** is sufficiently small and check the end configurations only
  - Note: to implement this, we moved the robot to the configuration and check if there is collision, so it’s normal for the robot to “shake around” during path finding
- Error: “***Timeout: robot is taking longer than 5s to reach the target joint state. Skipping...***”
  - This means your robot stucks when executing a command
  - Check if your robot runs into a obstacle
  - It’s possible that ***delta_q*** is too large so the robot runs into collision (as aforementioned, we didn’t check collision along the path)
- In ***execute_path()***
  - “wait” in the comment means stepping the simulation: ***env.step_simulation(1e2)***
  - time.sleep() is only going to sleep your computer in the real world, not the physical world…

Extra Credits

- Task: Transfer all objects to another bin

- Challenge: Ground truth object pose is not available

- Solution: Use RGBD images and object 3D model to obtain grasp pose

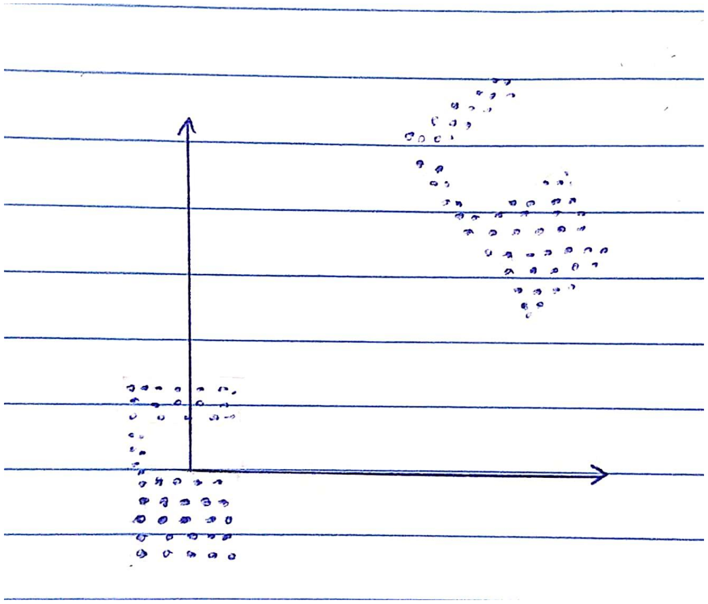


Extra Credits: Figuring out the object pose

Input: Current observation of a known object as a *point cloud*

Output: the object pose

Solution: ICP

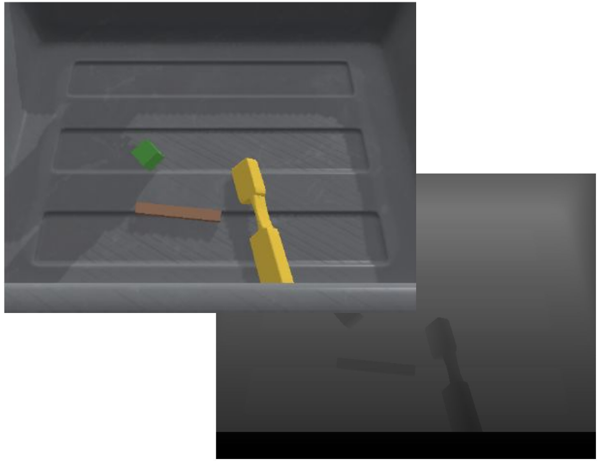


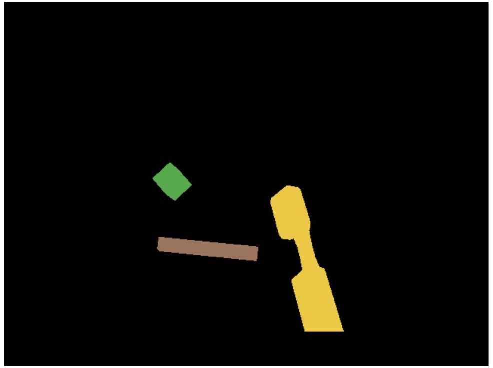
Extra Credits: Get object point cloud

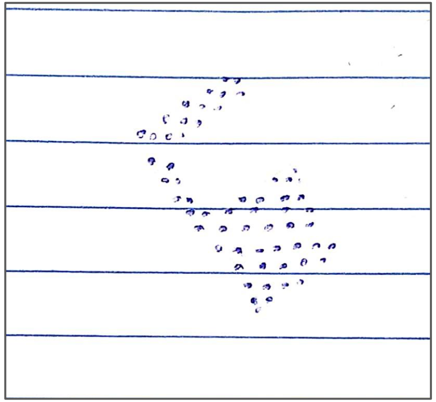
Input: Current observation of the scene as a *RGBD image*

Output: Current observation of the object as a *point cloud*

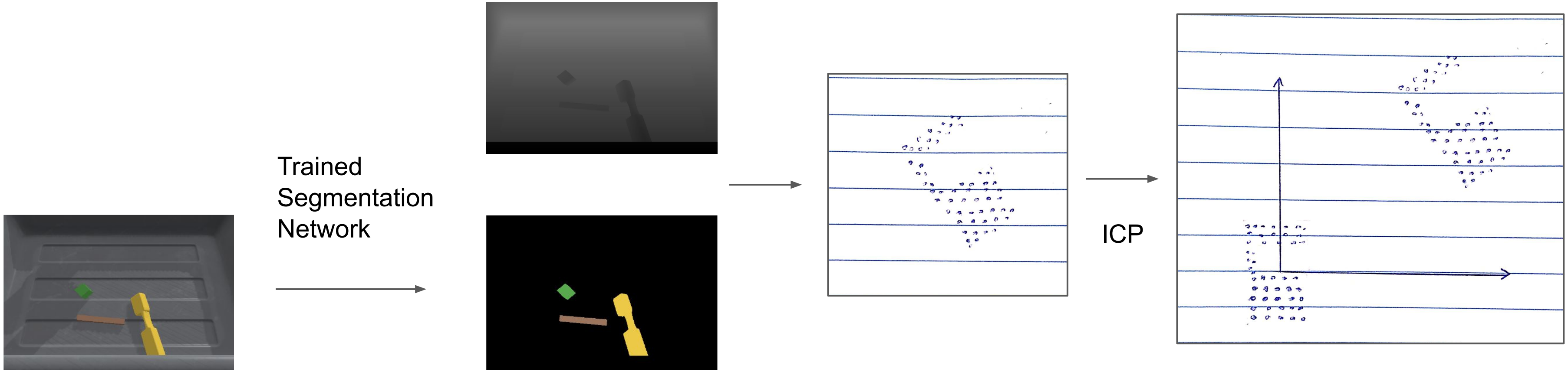
Solution: Semantic Segmentation







Extra Credits: [Overall pipeline]



- In this project, we will assume that the grasp pose is the object pose. In reality, it's itself a hard problem to figure out a good grasp pose.
- It's fine if the robot needs a few tries to grasp an object.
- It's possible that objects fall off during path execution, or the objects do not fall off even when the gripper opens. It's fine as long as your robot picks the object up, finds the correct path, and executes all the required actions.
- It's also possible that objects are initialized with hard poses so that robot cannot pick them up. You can simply try to re-run the script, or reset objects in the script.

Submission

- Check the pdf for detailed instructions
- Don’t wait for last moment to upload the video.

Questions?