Intelligent Robot Arm Manipulation
Final Project Report

Cong Fu, Deepesh Padala
March 29, 2025

Contents

1 Introduction

This report documents our final project on intelligent robot arm manipulation. The project consists of five main tasks: perception, control, grasping, localization & tracking, and planning. Each task builds upon the previous one to create a complete robot manipulation system capable of perceiving objects, planning trajectories, avoiding obstacles, and successfully grasping objects.

2 Task 1: Perception

Point Cloud

File: `src/point_cloud/point_cloud.py`

The idea of this part is to capture the point cloud of the object and use it to create mesh and bounding box for grasping. First, we move the robot arm to a pre-defined position up high to detect the object by extracting its segmentation mask.

After point cloud of the object in world coordinate system is reconstructed from this image, we use the centroid of all the cloud points on xy plane as the x and y coordinates of the reference point for further arm movement, and the maximal z coordinate of cloud points + height offset as the z coordinate of the reference point.

Then we define 4 poses for the robot arm to move to, where point clouds are extracted from image with 4 different directions. Here we use a simple trajectory interpolation in joint space for the movement, because there's no obstacle that needs to be considered.

ICP algorithm is used to align the point clouds and get a complete point cloud of the object, albeit with holes for some objects. Point clouds are visualized when running main function. Therefore interactive visualization needs to be closed to continue.

Bounding Box

File: `src/bounding_box/bounding_box.py`

Since the objects are placed on the flat table, we use the point cloud of the object to create an oriented bounding box around z-axis. The z axis of the bounding box is parallel to z axis of the world coordinate system, while x and y axis of the bounding box are acquired by PCA of the point cloud on the xy plane. Visualization is also provided to see the bounding box in pybullet.

3 Task 2: Control

File: `src/ik_solver/ik_solver.py`

Numerical inverse kinematics is used to get the joint angles of the robot to reach the target position. We use damped least squares method here. Due to the simulation environment, solving process will force the arm to move along with iterations in pybullet. Not that it will cause problems, but we explored the possibility of using shadow client to work around.

There are limitations, solving for long trajectory will result in large position and orientation error, which is why we use hard-coded trajectory in the first stage of planning. Details will be explained later.

4 Task 3: Grasping

Files:

- `src/grasping/grasp_execution.py`
- `src/grasping/grasp_generation.py`
- `src/grasping/mesh.py`

First we create the mesh of the object and the gripper. A simplified gripper mesh is hand crafted in open3d by referring to Franka Emika Panda robot Product Manual. Link: `https://download.franka.de/documents/Product%20Manual%20Franka%20Hand_R50010_1.1_EN.pdf`

Then we sample the grasp positions uniformly within the bounding box. For the sampling of grasp orientation, the gripper is forced to aim downwards and calculate length and width of the bounding box. The gripper width (direction of opening and closing of gripper fingers) during sample is always parallel to the shorter side of the two, therefore grasping will be easier and more consistent.

Next, we check the collision between the gripper and the object. Note that YcbPowerDrill is a bit special, because it's not a convex object, and much more complicated compared to other objects. Mesh quality of this particular object has a lot to be desired. And it caused the collision detection to fail (every grasp pose is detected as collision). So we use point cloud to detect collision for this object. All the other objects are detected by mesh collision detection.

After collision detection, we check the grasp containment. 2 ray planes are created along the directions of finger thickness. Each ray plane has 50 rays from root to the tip of the fingers. The criteria consist of several parts:

- At least 1 ray should intersect with the object for each ray plane as safe guard.
- Count intersection ratio of all rays as quality metric.
- Each intersected ray theoretically should have 2 intersections with the object. But for mesh collision detection, the rays will only yield the first (closet) intersection point. Therefore, we do ray casting bidirectionally to get the distance of intersection point to corresponding ray starting point. Shorter distance gives higher score. This criterion encourages the gripper to be closer to the thickest part of the object, which is really beneficial for grasping convex objects in the direction of gripper thickness.
- In the direction of gripper width, distance between center of the object and the center of the gripper on xy plane is calculated. Closer distance gives higher score. This criterion avoids grasping poses where one finger is too close to the object, which could cause collision due to precision error. After all, the grasping poses are realized in open3d, and the simplified grasp mesh is a bit different from the original end effector in pybullet.

Then we weight each criterion and sum them up to get the final score. The grasping pose with the highest score is selected as the final grasping pose. To avoid unwanted behaviour, we name the final grasping pose as pose 2, and define another position as pose 1, which is acquired by pure translation along its own -z axis (no rotation). In this case we move the arm from the state at the end of point cloud collection to pose 1 with simple interpolation in joint space, and then interpolate in cartesian space and solve IK for movement between pose 1 and pose 2, therefore straight trajectory can be guaranteed during actual grasping.

Finally we lift the object slowly to a predefined position and check if the object is actually grasped by calculating the actual distance between two fingers. Gripper finger's behaviour is achieved by position control with a specified torque applied. If the object somehow slips or falls down to the table, we iterate the perception part with point cloud and grasp sampling all over again as a fail safe. 3 chances will be given.

5 Task 4: Localization & Tracking

File: `src/obstacle_tracker/obstacle_tracker.py`

The high camera is used to detect the object by extracting the segmentation masks of the two red ball obstacles. Then we find the centroid of the corresponding mask, reconstruct the 3D position of the object in the world coordinate system. Radius is compensated along the direction from camera center to the centroid of the mask for each sphere to get their centers.

The states of the two spheres are returned. Additional boolean flag is used to check if the two spheres are away from the tray, which is used during final planning stage.

6 Task 5: Planning

Files:

- `src/path_planning/planning_executor.py`
- `src/path_planning/potential_field.py`
- `src/path_planning/rrt_star_cartesian.py`
- `src/path_planning/rrt_star.py`
- `src/path_planning/simple_planning.py`

Due to the aforementioned limitation of IK solver, solving for the end of long trajectory has some unexpected behaviours. The end of the trajectory would deviate from desired position. So we first move the gripper to a hard-coded pose. And then plan the final movement with RRT* as global planner and artificial potential field as local planner. We've also achieved a planner with pure hard-coded position as a comparison.

RRT* is achieved in joint space, so that the initial planning will also avoid collision between the obstacles and each joint of the robot arm. The global trajectory will provide an attractive force to the end effector during local planning. In this case potential field method considers total gradient from 3 types of sources: repulsive force from 2 obstacles, attractive force from end position, and attractive force from the trajectory. The repulsive force application radius has been increased so that larger objects won't collide with red ball obstacles as well.

Once the end effector reaches goal position up high, it'll wait until the obstacle balls are away from the tray before opening the gripper. Right now the threshold is 'x < 0.03' for larger obstacle and 'y < 0.03' for smaller obstacle, meaning the balls need to "move pass" the robot base.

7 Notes and Limitations

Due to the need for static render from high camera, the final trajectory will be slow. Some objects will be stuck with the gripper even if it's set to open at the final stage. Since it's an issue from simulation engine, there's nothing we could do unfortunately.

Grasp sampling is set to 2000, so it might be slow to test on other machines.

8 Contributions

Cong Fu: point clouds collection, bounding box, ik solver, obstacle tracker, grasp containment metrics, global planning, project integration

Deepesh Padala: local planning, point cloud fusion, grasp mesh creation, grasp collision detection