31391 - Software Frameworks for Autonomous Systems

Mini project 1 - Pick & Place

Table of content:

Applied algorithm

The plan for the robot consisted of several minor tasks:

1. Subscribe to the publisher "/gazebo/model_states" and using the callback function get the position of the bucket and the cubes
2. Examine whether the cubes are on the table and run the loop for each cube until none of them remain on the table:
    2.1. Move the robot to the “safe” position above the center of the table
    2.2. Position the effector above the chosen cube
    2.3. Open the gripper
    2.4. Move the effector towards the cube in cartesian coordinates
    2.5. Close the gripper slightly and move the effector to the lowest position
    2.6. Close the gripper
    2.7. Move the effector upwards in cartesian coordinates
    2.8. Move the effector above the bucket
    2.9. Open the gripper
3. If the cube is not reachable then move to the next cube

Code description

class position_finder():

This is the main class of our node. It contains the necessary attributes, which serve as “global” variables, which is especially useful for the callback functions.

def initializeStuff(self):

This is our main method.

- It initializes the RobotCommander, the PlanningSceneInterface and the MoveGroupCommander inner classes.
- It creates a subscriber to get the information about the items inside the gazebo environment.
- It creates two publishers to be able to control the joints and the trajectories of the robot.
- It calls the method findStuff() in a loop, so that every cube ends up inside the bucket.

def findStuff(self):

This is the method in which we define the pattern of movement to reach every cube.

- It starts at a neutral configuration around the middle of the table.
- Then it moves above the next inspected cube, which means going:
    - Above the cube
    - Slightly lower every time
    - Reaching the cube
    - Gripping the cube
    - Returning above the cube slowly and steadily

- Checks if the cube is labeled as reachable and is outside the bucket
- Then only if it is, it returns above the bucket
- And finally, the gripper opens and the cube drops to the bucket

def SlowlyReach(self, config, min_precision, max_precision, cartesian = False):
This is the method that repeatedly calls the motion planner, every time with a different tolerance level, depended on the precision needed by our current movement (i.e. the neutral position could be a little off, while the “reach the cube” position must be as precise as possible)

def moveArmCartesian(self,config):
This is the method that does the actual motion planning using cartesian coordinates. It uses the current position and the goal position of the arm as “waypoints” and checks whether or not the “goal position” is a valid one, meaning if the current cube is even reachable by our arm (i.e. it might be too far away). This happens by trying to force a more successful trajectory percentage wise, by adding more intermediary points in the linear interpolation.

def moveArm(self, config):
This function is similar to *moveArmCartesian()*, but works in joint coordinates.

def gripperState(self,tmp):
This is the method that handles the gripper of the robot (open and close). It utilizes the JointState publisher.

def callback(self, ms):
This is our callback function, which continuously gets pose messages from the pose subscriber and saves them as the positions of the cubes and the bucket from the gazebo environment.

def main(args):
This is our main function, which initializes the node, opens and closes the moveit_commander, creates an instance of our *position_finder()* class and calls its *initializeStuff()* method, which begins the chain reaction.

Encountered problems

Many cubes are being spawned in unreachable or specific positions. In particular:

- Cube too far away for the arm to reach. This was partially solved by labeling cubes as unreachable.
- One cube on top of another.
- Two cubes being too close to each other or even touching - causes the collision with the gripper.
- Cube too close or touching the bucket - collision occurs.
- MoveIt seems to not be able to do the inverse kinematics for medium and large distances. This was partially solved by dividing trajectory into smaller steps and managing the goal tolerance variable.

The solution that we thought of was creating an initial phase of “area scanning”. During this time:

1. Imaginary circles would be drawn for every item on the table
2. Every item inside the radius of those circles would be labeled as problematic
3. The arm would push every item to a valid initial position

Results

The arm, as it can be seen in the attached video, works pretty robustly and completes the task in most cases. However, when one of the described cases occurs during the “cube spawn” process, then the arm might get stuck throughout the task.