Super Assembling Arms

Yun Jiang, Nan Xiao, and Hanpin Yan
{yj229, nx27, hy95}@cornell.edu

***Abstract*— Although there are more and more things personal robots can do for us at home, they are unable to accomplish work of relatively large scale, such as assembling furniture. In order for a robot to do that, lots of work need to be done for planning, vision, localizing, controlling, etc. In this paper, we focus on the planning part.**

Our robot only needs to know where the individual pieces we want it to assemble are located initially as well as where they should be after assembled. Our algorithm can be divided into three distinct parts. First, the robot has to take the final positions of the objects and compute the correct order to move them. Next, given the order, we plan the path from initial and final locations for each object. Lastly, we take the paths and move the robot's arms accordingly. For the experiments, we use the Willow Garage PR2 (Personal Robot 2) and all simulations are done in the Robot Operating System (ROS).

I. INTRODUCTION

ACCORDING to a study by ABI Research, the personal robotics market will be worth $15 billion by 2015. There are many ways in which personal robots can help us on a daily basis, including floor cleaning, laundry folding, dish washing, food- and medicine-dispensing, surveillance, etc. As Bill Gates stated in 2007, there would be "a robot in every home" in the near future.

In addition to the tasks mentioned in the previous paragraph that a personal robot can help us, one unique task that has not really been touched upon is to make a robot assemble things for human beings, such as tables, desks, bed, Ikea bookshelves, etc. Imagine that you just moved to a new apartment and bought a lot of unassembled furniture from a store, say Ikea, and you are so tired after such a busy day. Would it be nice if there is a robot that can assemble the furniture while you can just relax? The ultimate goal is to design a personal robot that can assemble anything. Many things need to be done in order to reach the ultimate goal. For this particular project, our focus is mainly on planning, which involves planning for what sequence the robot should move and assemble the objects, planning for which path to take to move the individual object to its desired location, as well as planning for how to move the arms.

The whole procedure of assembling can be very complex. There are two major challenges: finding out the sequence of assembling and the trajectory of arms to move objects. For the first challenge, many factors have to be taken into account, such as gravity, and constraints occurred by other objects. Sometimes, even human beings need guidance to figure out sequence of assembling. The second challenge is caused by limitation of robot itself (some pose is impossible to reach) and also the collision with environment. Sometimes, these two issues have to be considered together, because the order of movements can limit the number of possible trajectory and in the other hand, the configuration of arm can make some order impossible to achieve. However in this project, we are only trying to solve relative simple and general cases to start with.

We build a framework for assembling, which consists of three parts, *Task Planning*, *Object Planning* and *Arm Planning*. The first part is to solve the first challenge, and the latter two are for the second one. In the rest of the report, we will first show some prior work, and then some problems we encountered in this project. In algorithm part, we will describe the whole frame work in detail, and then give the result of simulation. Finally, we will analysis the result and also discuss possible future work.

II. PREVIOUS WORK

Not much work related to assembling for personal robots has been done previously. However, there are several topics on planning that we take ideas from. We look at different planners such as RRT, KPIECE, SBL, CHOMP, etc. A third-party software package called MSL (Motion Strategy Library, which has existing implementation of different versions of RRT planners) is used in our implementation.

III. PROBLEM ENCOUNTERED

As mentioned in the previous section, not much work had been done previously on assembling. None of us had too much experience in robotics, especially planning. A lot of our time was actually spent on learning, reading papers, researching, and how to format the problem instead of how to solve the problem itself.

Probably the biggest issue for our project was getting us familiar with and actually able to use the Robot Operating

System (ROS). This was taking tens of hours. We first had troubles installing ROS on our computers, due to the fact that the installation guide on the ROS website was not too clear. We then had issues running the pr2_gazebo package in ROS. Only the computers that have some specific models of graphics cards were able to run the pr2_gazebo package with no problem. After we understood the issue and solved it by ordering and installing new graphics cards on the computers, a lot of time had already been passed by. In additional to these technical issues, ROS itself was very difficult to use. For example, ROS had very low precision for the coordinates in simulation. Sometimes the robot failed to move the objects to their desired location due to this poor precision. Also, the documentation for different ROS packages on the website was not very straightforward to understand either, so lots of our time was devoted on trials and errors.

IV. ALGORITHM

Since our focus is on planning, we assume the robot has full knowledge about the environment, and does not take in sensor data. For the planning part, first the robot needs to know how the furniture it is trying to assemble looks like at its final state (ex. a bookshelf) and find out the best sequence to move the individual pieces. This is called *Task Planning*. Then, given the sequence, it plans a plausible path for each object. We call the above step *Object Planning*. The last step is called *Arm Planning*, which is when the robot finds out the grasping point for the objects and moves them to their final state.

Figure 4.1 shows the flowchart of our system.

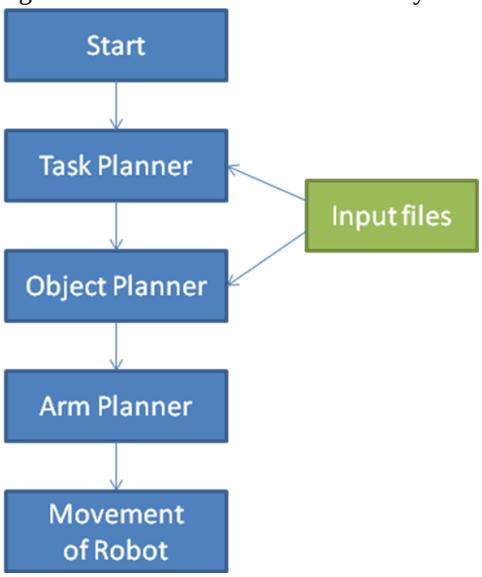

Figure 4.1 Flowchart of planning algorithm

A. Inputs

We require the object file itself and its initial and goal states. The object file contains the coordinates of the vertices of the triangular meshes that are used to represent an object's surface. The initial and goal states contain the location (x, y, and z) and the orientation (alpha, beta, and gamma) of the initial and goal positions respectively. We also require an initial obstacle file which is also in mesh format. The initial obstacle file tells the robot where the obstacles are prior to doing the assembling task.

Figure 4.2 shows how we organize input files.

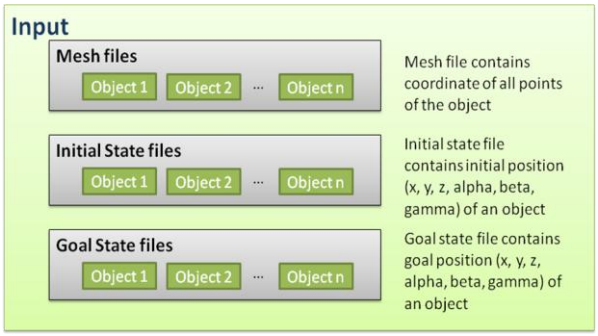

Figure 4.2 Input files in a folder for a task

B. Task Planning

The task planner computes the sequence of the objects that need to be moved for an assembling task.

For each object, task planner takes its mesh data, initial state and final state as input. If necessary, the planner finds the order to disassemble the objects from the initial state first and put them onto a predefined parking lot. We call this the preparing stage. Then in building stage, the planner finds the order to assemble the objects to final state.

The planner uses the lowest corner point of an object to represent that particular object during planning. In most of the cases, the planner simply outputs a sequence of objects based on their lowest corner coordinates, from bottom to top, which means the robot needs to put down the bottom objects prior to putting down the top objects. However, for some cases such as a bookshelf, we need to assemble the outer part first and then insert the boards in the middle into the shelf. Here, we introduce the idea of an "inner" object. An object is defined as "inner" if its lowest point is not directly above another object. The planner takes care of the "inner" objects at the end.

Figure 4.3 shows the flowchart of the task planner.

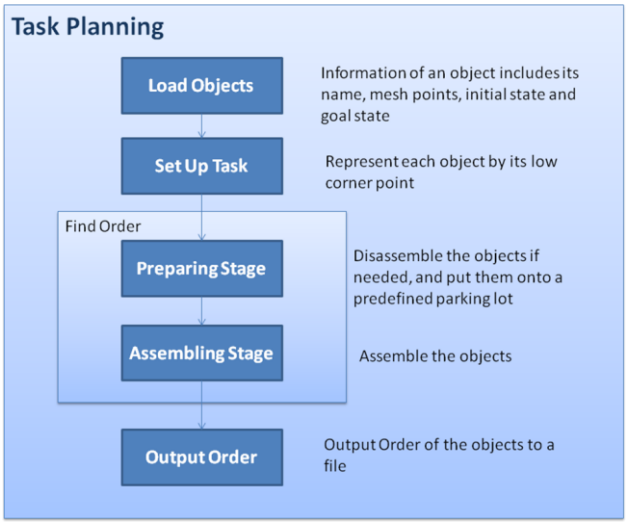

Figure 4.3 Flowchart of Task Planner

C. Object Planning

The object planner computes path from initial location to final location for each object. The path is constructed by a set of nodes, which represented by coordinate, on the path.

The planner takes the order computed by the task planner, as well as the initial obstacles existed in the environment as its input. For each object, MSL (Motion Strategy Library, which contains a set of different planners) is used to plan for its path. The object that is already moved to its final location is considered also as an obstacle and therefore it is added to the initial obstacle file. The planner smoothes the path by interpolation, and outputs a proper path to move the object from initial location to final location without hitting obstacles for each object.

Figure 4.4 shows the flowchart of this planner.

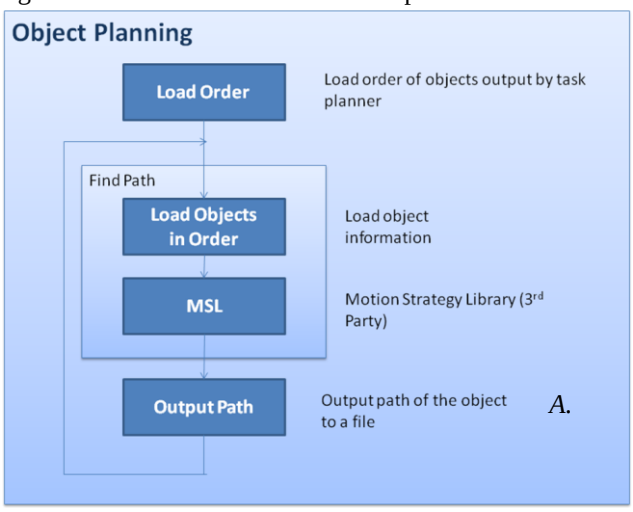

Figure 4.4 Flowchart of Object Planner

D. Arm Planning

Now, given the path for moving an object, we need to find out the ultimate trajectory for the arms. The given path is known to be collision-free, thus if the arms can move along the same path, then the object will not collide with others. So the goal for this part is to fit the trajectory to the given path. It is natural that during the moving, the relative position of hands to the object won't change. Therefore it is reasonable to assume the grasping points of the object are always fixed. Thus, given the path of object, the path of grasping points is given as well. As a result, we can simply find the angles of all the joints of the arm (then an arm configuration is uniquely defined) for each points in the path using IK (inverse kinematics) solver, and then use simple interpolation to link the points to obtain the trajectory.

More specifically, the arm planner takes in the path outputted by the object planner and first transforms it into path of grasping point (if two arms cooperate together, then two paths will be calculated respectively). Then it calculates the IK solution for each grasping point, and last, links them together to get the trajectory and send it the arm controller.

Figure 4.5 describes how this planner works.

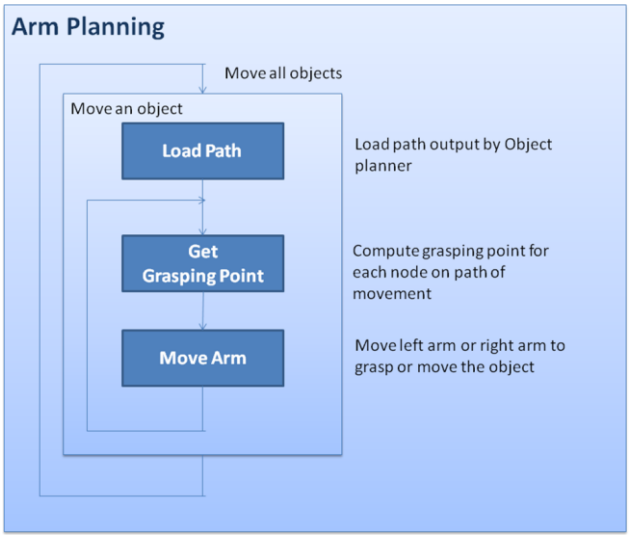

Figure 4.5 Flowchart of Arm Planner

V. Simulation and Results

Simulation Setup

Because we do not have access to a real PR2 robot, we evaluate our algorithm in simulations. We build several different scenes to test our algorithm, from the easiest (3 blocks) to the most difficult (assembling a table with four legs and two boards). For each simulation scenario, we first generate objects' 3D mesh data using tools such as

AutoCAD, and then design the scene including all the initial and goal positions and orientations of the objects, as well as their grasping points. The complexity of our simulations is limited by robot itself and ROS in several ways. First, since in this project we focus on arms rather than moving the whole body, the robot has limited reaching space which is less than the square of the length of its arms. Second, in terms of unit in ROS, the gripper can open as much as 0.09, which means the robot can only grab very thin board or cylinder-shape objects. This limitation makes moving large blocks almost impossible. Third, the usual tolerance of precision is 0.02~0.05, so it will fail on complicated assembling cases, like screwing or picking up small pieces. Also, because the front part of grippers is flat, it is hard for the robot to pick up things on the ground unless they either stand up or are aligned along the edge of tables. After input data is ready, we load the robot and the objects into gazebo (ROS simulating environment), and run the pipeline to get the order and paths for objects and trajectories for arms, and then control PR2 to finish the task.

B. Assembling a Table

In this task, the robot is given a task to assemble a table with four legs, one top board, and one base.

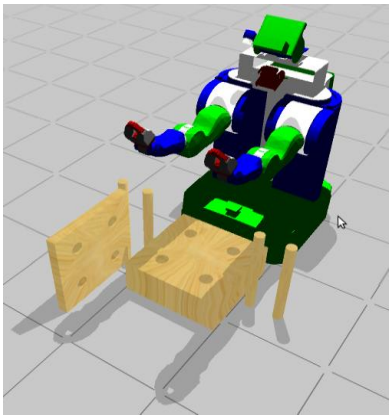

Figure 5.1

As we can see in Figure 5.1, in the beginning, the base board is placed on the ground in front of the robot so that the robot can reach the furthest corner of it. And there are 2 legs on each side of the robot. The top board is on the right side of the robot. We put it vertically instead of horizontally because it is easier to be picked up. The target table is shown in Figure 5.2, where the four legs are inserted into four deep holes in the pedestal, and also the top board is placed flat and stably above legs.

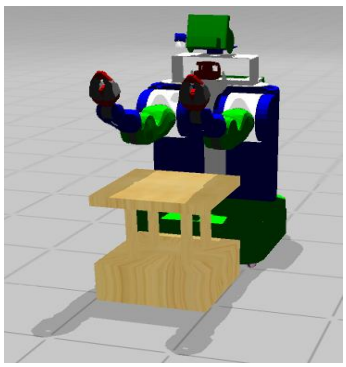

Figure 5.2

According to our task planner, the assembling order is two legs in two further holes first, then two nearer ones, and the top board for the last. This order is reasonable. However, whether the task will be successful also depends on the other two parts. Especially, paths of each object should be feasible for the robot, although there are an infinite number of collision-free paths to move them.

In object planning, we used RCRRTExtExt from the MSL software package as the planner. Generally, for the four sticks, the simplest path is first lift it from the ground, move it right above the hole, and then put it down into the hole. And we do not need to rotate them in the process. But for the top board, we need to rotate it and the time for rotation is crucial due to the fact that rotation needs to be done when there is enough space and the four legs will not be on the way. The resulting path is lift it a little bit and then rotate it and move it horizontally to the place slightly higher than the goal position and eventually put it down. We can see that all paths are simple and short. They don't contain any redundant rotation or detour. Although originally they have some zigzag parts, we take care of it by smoothing the paths. Another thing need to be noticed is for the last board, we use only one arm to move it. But one can tell that a better way to move it is using two arms which will make movement more stable. However, due to the limited moving space for the left arm, it cannot reach the top board and therefore using right arm alone is the only solution.

In the last part, arm planning, as mentioned earlier, we use IK to find exact position of each joint for each point in the trajectory. Although the trajectory is usually smooth because points are close to enough to each other, sometimes angles given by IK might jump from $-\pi$ to π, and somehow arm controller would make the arm rotate 360 degree to reach the goal. We have solved this simply by increasing or decreasing the second angle until it is closest to the first one.

The simulation is not successful for the first time. This is caused by some precision issues. Because IK solver can be very sensitive to the position, if the object is moved by even 0.1 unit, this can cause no IK solution being found. Thus,

we actually spend a lot of time making sure that the IK solutions can be found for all points. Even all trajectories are found, during the manipulation, sticks sometimes could not be inserted into the hole because it touched the edge. Thus we make the hole a little wider than the stick, but also deeper to hold it tight. Figure 5.3 shows one screen-shot of the procedure moving legs.

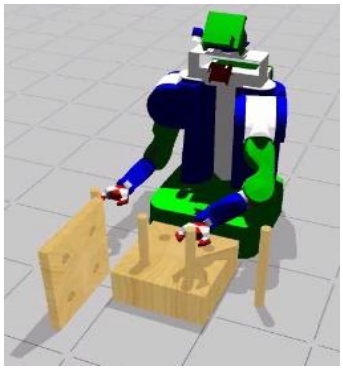

Figure 5.3

An even severer problem is that when we are moving the last top board using right arm, it is very unstable and the board is swinging around all the time. As you can see from the video we submitted, although the arm can move to the goal position for the last part, the board itself is not in the correct place because it slips from the gripper a little, as shown in Figure 5.4.

Figure 5.4

In summary, the order, paths outputted by planner are correct, but IK solver in ROS is more sensitive and unstable than we expected, which makes simulation unnecessarily harder. We think this issue also reflects our algorithm is not robust enough. If we consider more possible candidates rather than just one path for each object, and if we can adjust the path in real time, simulation will be more successful.

VI. LIMITATION AND FUTURE WORK

As you can see, our scale is relatively small right now. There are constraints such that objects cannot be too heavy for a robot to lift with one of its arms and they need to be placed within reachable distance from the robot. The small scale is partly due to the fact that we are mainly focusing on planning and have not spent a large amount of time dealing with other issues such as moving the robots around and lifting very heavy objects. In the future, we can move the robot around and ask it to pick up objects from several locations and finally assemble them together.

Our task planner can only deal with relatively small amount of cases. We can also enhance its algorithm to make it more general and robust so that it will be able to deal with more corner cases. After all, our ultimate goal is to make our robot capable of assembling anything.

Path planning is done by using the RRT planner in the MSL package. Currently we have issues that the path outputted by the planner is not smooth enough, even after the interpolation. In the future we can cooperate with Ian Baldwin from MSL to have a better idea about how to use the planner and how to improve it, if possible. A clear and smooth path is vital to the success of the project because it can also make the arm planning much easier.

As mentioned earlier, we want to move the robot around. This can help us in planning for the movement of the arms because there will be less constraints for us if the robot is able to move itself. For the heavier objects, we will want the robot to actually use both of its arms to lift and move them. The benefit of using both arms is that it will hold the objects tighter so that more accuracy can be achieved.

REFERENCES

[1] SVN Repository for our projects. Available: https://forge.cornell.edu/svn/repos/src/planning

[2] Robot Operating System (ROS). [Online]. Available: http://www.ros.org/

[3] Motion Strategy Library (MSL). [Online]. Available: http://msl.cs.uiuc.edu/msl/