Manipulation Planning for the JSK Kitchen Assistant Robot Using OpenRAVE

Rosen Diankov (University of Tokyo), Kenji Sato (University of Tokyo), Hiroaki Yaguchi (University of Tokyo), Kei Okada (University of Tokyo), Masayuki Inaba (University of Tokyo)

1. Abstract

Using the OpenRAVE motion planning system, we show the minimal number of steps to get the JSK Kitchen Assistant Robot to autonomously pick-up cups and dishes from the sink and place them in a dishwasher (Figure 2). Starting from just the basic CAD and kinematics models of the robot and target objects, it took a total of **9 hours** of development with OpenRAVE to achieve the final autonomous pick-and-place system. We originally developed this demo as a case study of how quick a planning system tailored for a specific task can be created with OpenRAVE. In this paper, we describe the entire 9-hour process we went through with OpenRAVE to achieve this demo.

2. What is OpenRAVE?

OpenRAVE is an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications (Appendix, [1]). It has many functions for analyzing the geometric structure of robotics scenarios and applying them to moving robots throughout the workspace. OpenRAVE does two things really well:

- For each robot, use IKFast to generate an inverse kinematics program specifically tailored for that robot's structure. This allows **all** singularity configurations and divide by zero conditions to be handled for. Most of the generated solvers run in 5 μs.
- Can easily combine multiple constraints like avoiding collisions, grasping objects, maintain sensor visibility, to connect a robot's initial and goal configurations together.

OpenRAVE works in three stages (Figure 1) where it pre-computes a database of all *static* information used to solve the problem. This information is usually the target object and robot CAD models. Although the database generation times can take anywhere from a couple of minutes to a couple of days, they reduce the number of parameters the user has to deal with when planning in the real environment. Figure 3 shows a more detailed peek inside pick-and-place planning system. All the robots and obstacles are stored in COLLADA format [2] using extra robot specifications [3]. Each robot has a list of manipulators that hold the chains within the robot hierarchy that inverse kinematics can be applied to. In the database generation step, stable grasp are generated and the analytic equations to inverse kinematics are solved. Finally the pick-and-place problem requires two samplers to be created:

Fig.1 OpenRAVE works in a 3-stage system.

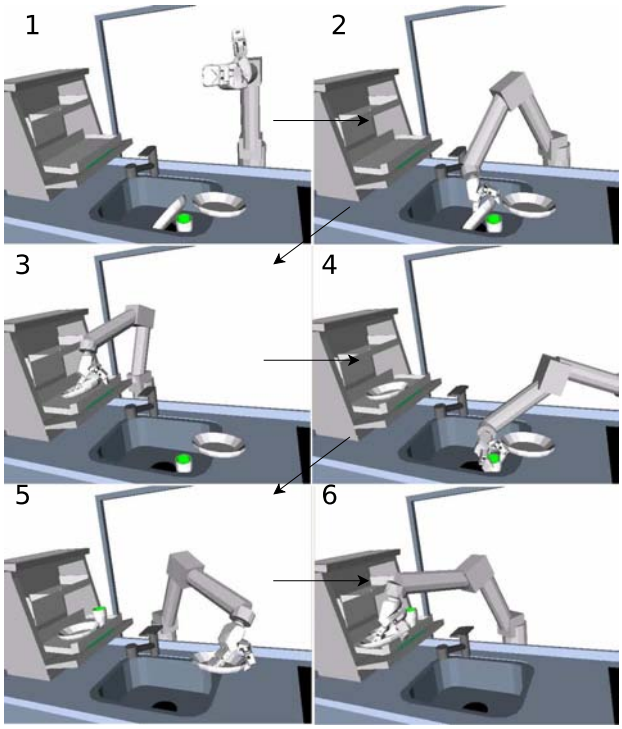

Fig.2 A 6DOF robot picks objects from the sink and places them in a dishwasher.

- a sampler to find a robot configuration where the hand is grasping the target object,
- and a sampler to find a valid destination of the target object while the robot is grasping it.

The manipulation planning phase then samples from both the current and goal samplers and attempts to connect them in the robot configuration space while avoiding collisions.

3. Robot Preparation

Once the robot geometry and joints are defined, we need to attach manipulator meta-data that:

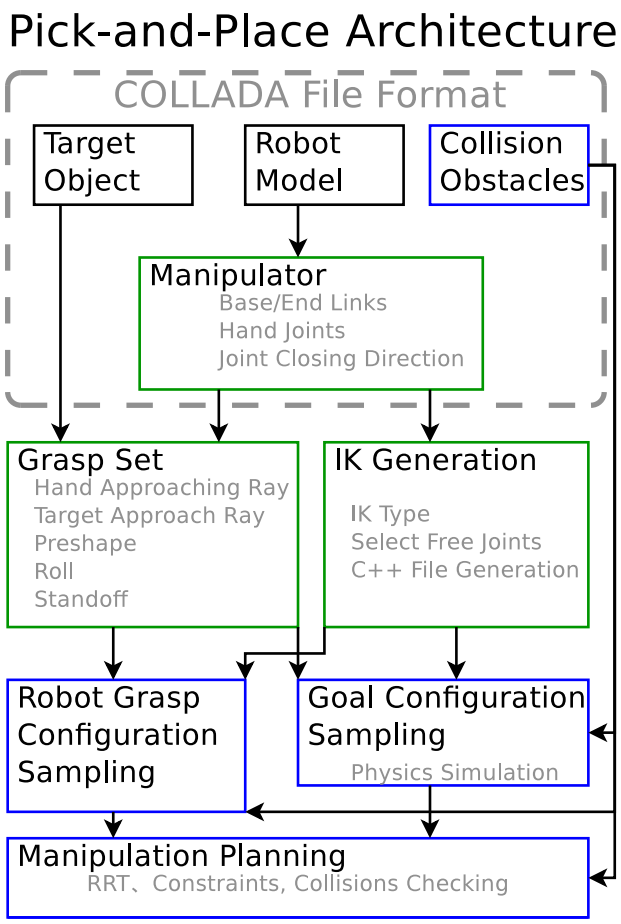

Fig.3 The OpenRAVE pick-and-place planning system is composed of **Given Information**, Computed Information, and Environment Information.

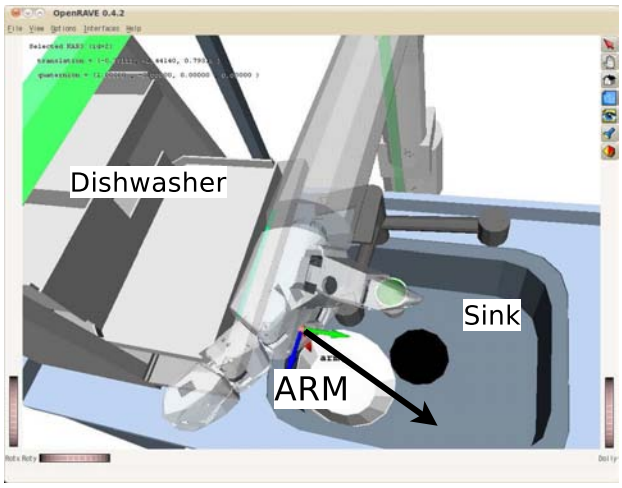

Fig.4 The kitchen scene with a 7DOF robot that can slide across the kitchen on a rail. The robot has an **arm** manipulator defined with an grasping approach direction normal to its palm.

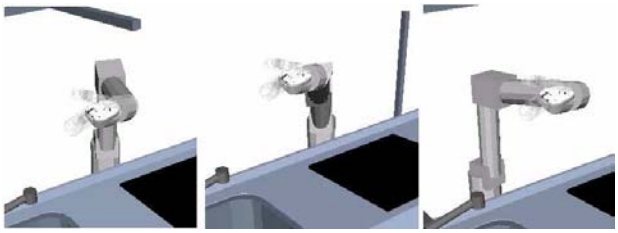

Fig.5 Given a 6D hand position, the IK solvers generated from IKFast allow the entire solution space to be explored.

- Defines the base and end effector links. Kinematic structure analysis allows the chain to be extracted from that,
- Defines the hand joints,
- Defines the closing direction for each of the joints,
- Defines the coordinate system of the manipulator.
- and defines the inverse kinematics types that are applicable/interesting to the robot.

Figure 4 shows the robot manipulator coordinate frame and the scene we'll be using to plan in. The Kitchen Assistant Robot (KAR) has 6 rotation joints and 1 slider joints that allows it to slide across the kitchen.

4. Database Generation

OpenRAVE IKFast is used to analyze the robot kinematics, solve the inverse kinematics equations, and write their solution to a C++ file. The C++ file is compiled into a dynamically loadable object and called during planning. In OpenRAVE, the **databases.inversekinematics** module is responsible for this management

```
openrave.py --database inversekinematics
 --robot=myrobot.dae  --manipname=arm  --iktests=100
 --iktype=transform6d --freejoint=base_joint
```

The IK type can be one of Transform6D, Rotation3D, Translation3D, Direction3D, Ray4D, Lookat3D, TranslationDirection5D, TranslationXY2D, and TranslationLocalGlobal6D. Because *Transform6D* only needs 6 joints to compute the IK, one joint has to be *freed* from the IK computation using –freejoint. Figure 5 shows 3 IK solutions given the same hand position.

OpenRAVE generates grasps by sampling rays on the target object's surface and manipulator's surface. The actual grasper module aligns the two rays, sets the hand to an intitial *preshape* and moves the gripper and the its joints until they hit the target. Once the gripper cannot close any further, all the contact points are extracted and the *force closure* criteria is computed; this guarantees that the gripper will be able to absorb any force the target object experiences. In OpenRAVE, the **databases.grasping** module is responsible for this management:

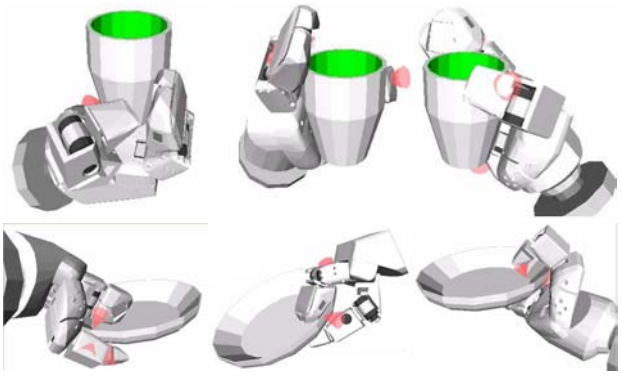

Fig.6 Possible grasps for a cup and dish. These grasps are stored in a set, which is then searched for when looking for a feasible grasp in the environment.

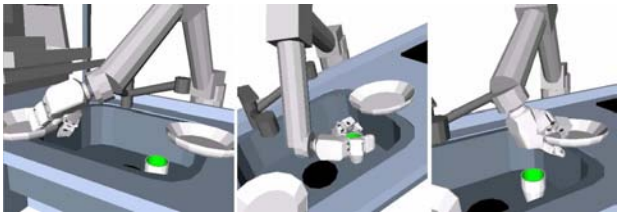

Fig.7 Robot configurations of stable grasps.

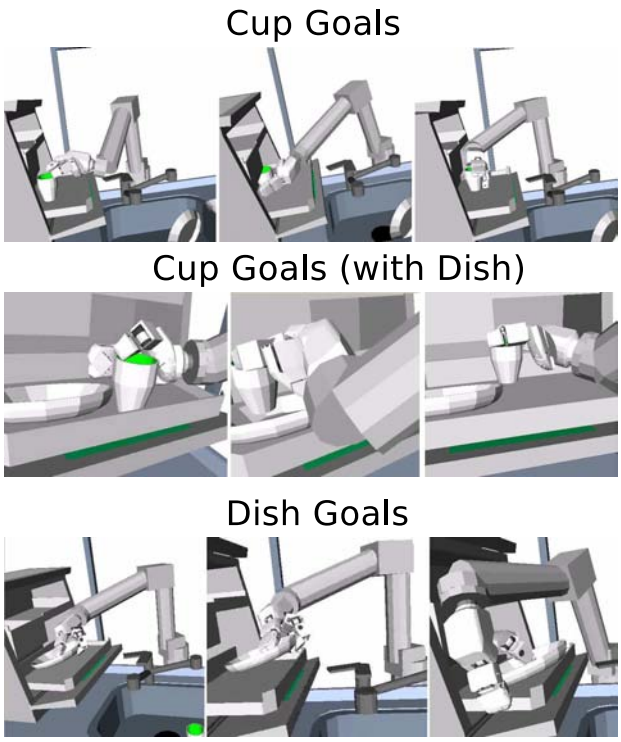

Fig.8 Robot configurations of feasible destinations of each object.

```
openrave.py --database grasping
 --robot=myrobot.dae --manipname=arm
 --target=myobject.dae --friction=0.1
 --preshape="-0.685 -1.48 0 0 0"
 --manipulatordirection="0.09 0.8  0.58"
 --graspingnoise=0.01
```

The command line interface takes the robot, its manipulator, the target object and a set of grasp parameters. –graspingnoise is the most important parameter since it prunes out fragile grasps that might be prone to slipping the target object; its value represents the amount of positional noise of the target pose the grasp should be robust to. The final grasps can be viewed with the –show command (Figure 6).

5. Scene Verification

Once all the databases are completed, we create a sample scene and test if the robot can grasp the objects from nominal positions (Figure 7). The **GraspingModel.computeValidGrasps** function selects grasps valid in the environment by checking collision detection, inverse kinematics, if the fingers would hit any other object when closing, if the robot can start approaching from at least a couple of centimeters away. Note that the same grasp can be achievable by many different robot configurations.

We then need to model the entire space of object goal placements such that they will be safely placed inside the dishwasher without falling out. We do this by sampling random 6D poses of each object on top of the dishwasher try and use a physics engine to drop the object down. Once dropped, we wait until everything settles before deciding if that was a good place to drop the object at. Once the object destination is chosen, we check if the robot can be moved to there using a grasp from the grasp set. Figure 8 shows the goal positions of three different scenes. Notice that sometimes the cup can be placed on top of the plate if the flat region of the plate is big enough.

6. Manipulation Planning

The final phase combines the grasp and goal samplers to sample configurations of the robot that can simultaneously pick up an object and place it in the dish washer. The BiRRT planner [4] is first started with one goal configuration sampled. Then as the planner runs, we call the goal configuration sampler in parallel and continue seeding the planner with more goals. Because each grasp is defined as a 6D pose, a Transform6D IK is the easiest to work with. The **examples.graspplanning** is a demo that comes with OpenRAVE and shows how to use this machinery. The main functions are in the **TaskManipulation** interface, and the following example shows how to use it:

```
openrave.py --example graspplanning
```

Figure 2 shows one run of the kitchen robot.

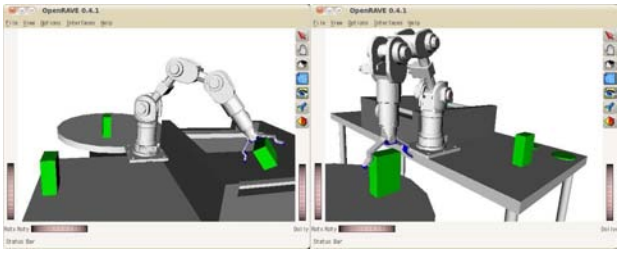

Fig.9 OpenRAVE supports manipulation planning with robots with only 5 joints.

Unfortunately, robots with only 5 joints cannot have Transform6D IK, so most researchers give up or attempt to parameterize the grasps so that the full 6D pose is not needed. However, OpenRAVE supports using the TranslationDirection5D IK type to do grasp planning with robots with 5 joints (Figure 9). In OpenRAVE, run the following command to see the demo in live action:

```
openrave.py --example graspplanning
 --scene=data/katanatable.env.xml
```

Note that we do not consider *regrasping* in this demo, but it would be possible to plan with regrasping by adding an intermediate 6D position of the target object.

7. Conclusion

In the paper we showed all the steps necessary to do pick-and-place tasks with OpenRAVE. The entire demo shown in OpenRAVE was done in 9 hours starting from just the basic robot and target object CAD models. The most time consuming part was generating the grasp set, since over 50,000 grasp candidates had to be tested. However, new OpenRAVE versions allow the grasp computation to be parallelized over a cluster of computers using ROS [5].

OpenRAVE is a huge system and there is a lot of unexplored functions that can help in modeling tasks and injecting planning and search-based components into them.

References

[1] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[2] R. Arnaud and M. C. Barnes, *COLLADA: Sailing the Gulf of 3D Digital Content Creation.* AK Peters, Ltd, 2006.

[3] Rosen Diankov and Ryohei Ueda, "Robot-specific COLLADA Extension Proposal." [Online]. Available: http://openrave.programmingvision.com/wiki/index.php/Format:XML

[4] J. Kuffner and S. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "(ros): an open-source robot operating system," in *ICRA Workshop on Open Source Software in Robotics*, 2009.