ECE 590.04 Homework 4: Motion Planning

Due date: 3/6, 5:00pm

In this assignment you will use the Klamp't interface to plan collision-free paths for the Amazon Picking Challenge.

GUI Overview. If you run "python hw4.py", you will see essentially an implementation of HW2, with a few changes.

- The robot now has parallel-jaw grippers (we probably will be using more sophisticated grippers for the challenge)…
- The robot now moves to the configurations requested via smooth trajectories. There is also a difference between the *commanded configuration* which is shown as a transparent green, and the *actual configuration* which is drawn in grey as usual.
- Pressing 's' toggles simulation, pressing 'q' quits.
- If you uncomment the 2 lines starting with "simworld = load_apc_world()" in main(), collisions between the robot, shelf, and objects will be physically simulated.

IMPORTANT: A small bug was just recently fixed; it would not allow you to see the simulated objects. To get this fix, you will need to run "git pull" on Klamp't and rebuild the python API using "sudo make python-install".

API Documentation. Your code will mainly be added into hw4.py and hw4_planner.py.

The biggest change between HW2 and HW4 is that your PickingController is now running in its own thread, which communicates to the robot's low-level controller. The low-level controller (LowLevelController) provides an API that you should use, but not modify. The picking controller now must call LowLevelController.setMilestone()/appendMilestone()/commandGripper() to send commands to the robot, and call getSensedConfig()/getCommandedConfig()/isMoving() to query the status of the robot.

It is also important to understand that there are now two world models:

- The *simulation world*, which acts a stand-in for the real world. You should not directly access it. You can only make changes to this world and sense its state via actuators and sensors (i.e., using the LowLevelController interface).
- The *planning world*, which is free for your planner to modify at-will to solve IK, run collision checking, etc.

For planning, you will implement subroutines of the LimbPlanner class. Part of this work will use the Klamp't collision testing API in the klampt.robotcollide module, and the Klamp't motion planning API in the klampt.cspace module. Again, the Klamp't Python API is documented in greater detail in http://motion.pratt.duke.edu/klampt/pyklampt_docs.

Questions.

[For questions 1, 2, and 3 you will NOT be simulating robot-shelf-object interactions, i.e., uncomment the line "simworld = load_baxter_world()" in main(). For questions 4 and 5, you WILL be simulating robot-shelf-object interactions, i.e., uncomment the two lines beginning with "simworld = load_apc_world()" in main().]

1. Run the picking controller to pick up and put down a few objects and carefully observe the motion of the robot.

   Describe at three or more issues that you observe that might cause problems during execution. For each issue, describe a potential method for overcoming it. Write your answers at the bottom of hw4.py.

2. Modify the code for move_camera_to_bin(), move_to_grasp_object(), and place_in_order_bin() so they ALWAYS produce self-collision-free IK solutions. You may disregard collisions with the object or shelf for this part, and ignore collisions along the path. It is suggested that you place your implementation in the LimbPlanner.check_collision_free() method in hw4_planner.py, since it will give you a head start for question 3.

   For collision testing, you can loop through all objects in the planning world that you wish to test, access their geometries using x.geometry(), and call geom1.collides(geom2). For example, to test for a robot link against a terrain, you may call world.robot(0).getLink(6).geometry().collides(world.terrain(0).geometry()). For a less verbose solution, you are free to use the WorldCollider class in klampt.robotcollide.

   Describe your method in a few sentences (i.e., which collisions do you test, and when during the IK solver loop) and run some experiments to determine an approximation for how many IK solutions are in self-collision. Write your description and results of your experiments at the bottom of hw4.py.

3. Modify the code for move_camera_to_bin() so it produces entirely collision-free trajectories. Specifically, no object or shelf collisions should be observed during the entire duration of the trajectory. The following are recommended steps for doing so.

   First, implement LimbPlanner.check_collision_free() so that it checks collisions with the shelf and objects. Note that the objects are not instantiated in the planning world even though they are in the knowledge base. You may either 1) when you sense a new object, instantiate it in the world using properly-sized geometries, and then re-initialize the WorldCollider object, or 2) test collisions manually in check_collision_free by loading cubes from disk into a Geometry3D object, and calling a collision check. In either case, example code can be found in spawn_objects_from_ground_truth() in hw4.py.

Next, implement the LimbCSpace class in hw4_planner.py to fill out hooks for defining a search domain and performing feasibility testing by calling the methods of LimbPlanner. Note that you will be planning in the 7DOF space of the limbs, rather than the whole 15DOFs of robot. This class inherits from klampt.cspace.CSpace, which allows you to invoke a motion planner using the klampt.cspace.MotionPlan class.

Implement LimbPlanner.plan_limb() by invoking a MotionPlan with your desired parameters. Note: the LimbPlanner.plan() function can automatically assemble limb paths into whole-body paths.

Finally, to send the paths to the controller, you will need to use the self.controller.appendMilestone() functions.

4. Standard sampling-based motion planners do not work particularly well for planning grasping paths. Explain why, in a few sentences, in terms of the shape of the configuration space obstacles. Write your answer at the bottom of hw4.py.

   Come up with a method in move_to_grasp_object() that uses a pre-grasp motion to reliably grasp the object with a collision free path, except for the very end of the motion when the object is finally grasped.

   Also, the existing grasps do not work well for the smaller objects. Devise some different grasps that will allow you to successfully pick them up with a collision free motion.

   Finally, test your planner in simulation. Tune your method and report on the final reliability that you get for these test objects.

5. BONUS (20 points): extracting the object during place_in_order_bin() requires some modifications to prevent the robot from scraping the object in the bin. Implement a planner that makes sure that the object does not collide with the shelf. Describe your changes.

Submission. Place your written answers in the indicated spaces at the bottom of hw4.py**.** Submit your hw4.py and hw4_planner.py files, and any other files that you have changed or added, and submit them on the Sakai website.