AI & Robotics: Lab Course
Weekly Exercise 2

Joaquim Ortiz de Haro & Jung-Su Ha

Learning & Intelligent Systems Lab, TU Berlin

Marchstr. 23, 10587 Berlin, Germany

Summer 2022

1 In class: Path Optimization & velocity/acceleration objectives

The second example in **course3-Simulation/03-motion** demonstrates a full path optimization example. Note the differences: We now specify the **times** argument when adding an objective, which is an single time slice or interval specified as a floating number. We defined the path to have 1 phase with 40 steps-per-phase; the **times={1.}** means the objective only holds at phase 1 (end of the path).

Further note the `qItself` objective with **order=1**, which constrains the joint velocity to be zero at the end of the path.

a) Remove the `qItself` objective - why is the result optimal?

b) Add the `qItself` objective again. Additionally, constrain the relative acceleration (i.e., **order=2** `positionRel`) of the object w.r.t. the gripper to have (!)constant value $(0, 0, -0.1)$ during the interval $[0.7, 1.]$. What is this doing?

c) Modify the previous to impose a reasonable grasp approach to the object, that works for any orientation of the object.

2 Explore collision features, and enforce touch

So far we neglected collisions – and it is generally a fair approach to first try to design motions that inherently stay away from collisions even without using collision features, as the latter imply local optima.

The `distance` feature returns the negative distance between the given pair of frames (where the frames need to be convex shapes). You should impose an inequality (lower-equal zero) to force the solver to avoid penetrations. By changing the target you can also add a margin.

a) Add an additional obstacle, e.g. a sphere, to the scene, with which your moving gripper (from the previous exercise) collides. Then add a `distance` inequality objective between the new sphere and the **"R_gripper"** object. In addition, also add the same between **R_gripper** and **object**, which should modify the grasp approach. You can also try analogously for **R_finger1** and **R_finger2**.

b) Now, remove previous grasp or collision objectives, and only add a distance equal to zero objective on the **R_finger1** and **object** as the goal constraint. This should generate a touching motion. This is a typical ingredient in generating pushing interactions.

3 Interacting with "real" objects

The two examples in **course3-Simulation/03-motion** do not really interact with the "real" world (Simulation, in our case), but only compute some motion in your model configuration. Let us change that:

a) Add a new object named "myObj" of shape type **ssBox** (see note below).

b) Setup a "real" world loop, and shortcut perception by always querying **RealWorld.frame("object").getPosition()** and **RealWorld.frame("object").getQuaternion()** to get the pose of the object "object". Set the pose of "myObj" to the same pose.

c) Try to use a finger of either of the arms to continuously touch the falling object in the real world loop.

You can use the skeleton code **e02-realObject.ipynb** as a starting point.

Note: **ssBox** means sphere-swept box. This is a box with rounded corners. This should be your default primitive shape. The shape is determined by 4 numbers: x-size, y-size, z-size, radius of corners. The 2nd most important shape type is **ssCvx** (sphere-swept convex), which is determined by a set of 3D points, and sphere radius that is added to the points' convex hull. (E.g., a capsule can also be described as simple **ssCvx**: 2 points with a sweeping radius.) The sphere-swept shape primitives allow for well-defined Jacobians of collision features.

4 Tricky use of inequalities, scaling, and target

This is a bit tricky to figure out, but if you do, you really understood the use of the scaling, target and inequalities.

As in Exercise 1, consider IK for grasping a cylinder again. Let's care only about the gripper position, not it's orientation. The position needs to be in the interval $[-l/2, l/2]$ along the z-axis of the cylinder, if it has length l. We can model this with 4 constraints:

- The x-component of the `positionRel`(gripperCenter,object) needs to be equal 0
- The y-component of the `positionRel`(gripperCenter,object) needs to be equal 0
- The z-component of the `positionRel`(gripperCenter,object) needs to be lower-equal $l/2$
- The z-component of the `positionRel`(gripperCenter,object) needs to be greater-equal $-l/2$

Can you figure out how to realize this? Tip: Choosing a scale **np.array([0,0,1])** picks out the z-component of a 3D feature (as it means multiplication with $(0, 0, 1)^\top$.) But note, the target always needs to live in the original 3D feature space!