Final Project

5.3 Dynamic Manipulation

David Nichol (dan1)

Alden Higgins (jah17)

Sam Tormy (srt7)

Problem

The problem we are trying to solve is how to simulate a series of joints that are linked together. In other words, given a list of joints linked together, our program applies torques in response to gravity in order to move the joints such that each is in their own respective goal state. This problem is significant because in many robotic applications, the robot's will be formed not as a single box entity but as a chain of joints that are used to manipulate objects. Hence, this joint simulation solves a problem that is vital for robotic movement.

Our Approach

Our project solution is centered over the ODE that we created for the joint manipulator. In our program, we follow the math outlined in Leon Zlajpah's paper “Dynamic simulation of n-R planar manipulators.” By converting the mathematical model to C++, we are able to solve for the qdots of each joint.

In order to represent the dynamic space, we create a compund state space with one SO2 state space and 1 Real Vector State Space for each joint. This means that we will have n * (SO2 + RealVectorStateSpace) spaces, or 2n total spaces.

In order to reach the goal states, we apply RRT to our controls and keep iterating until we have found our final goal states for each joint.

Experiments Conducted

We tested our program by calculating some simple joints by hand, such as 1 and 2 joint simulations with no torque, and walked through the sections of math with the debugger to ensure that each section was calculating the correct value. For higher dimensions, we used our visualizer to look at what was happening with the joints to verify that the way that the joints moved was realistic and probable.

We experimented mostly on 1, 2, and 3 joints, and saw that the joints would eventually reach their designed spots. While most of this we could easily see in the simulator, we could also see it quantitatively, as the final states that were printed to console would reach the intended goal states.

In our experiments, we assumed that each joint had an equal weight and mass, however, this program is easily extensible such that if we wanted to add different lengths of joints, minimal

code will have to be changed.

Analysis

We saw the joints very quickly approach the goal states that whenever we ran our simulations. For end states that were hanging down, the joints would fall down to gravity. For joints that were on the other side of the start states (which we typically tested as every joint pointed straight to the right), the joints would swing around. Quantitatively speaking, for example, we saw a joint with a start state SO2: 0, Real Vector 0 and a goal state SO2: -3.13, Real Vector 0 move like so:

0 0

```
-0.0735617 -1.4707
-0.293454 -2.91769
-0.652703 -4.22775
-1.1263 -5.15463
-1.66115 -5.41387
-2.18319 -4.90733
-2.6231 -3.81899
-2.9375 -2.4423
-3.10889 -0.980906
-3.13342 0.490485
```

To give a 2 joint example, start states SO2: 0,0, Real Vector 0,0, and an end state SO2: -3.13 1.27 Real Vector 0,0 move like so:

0 0 0 0

```
-0.272271 -5.40538 -0.209142 -4.14032
-1.03446 -9.28227 -0.779587 -6.70775
-1.96667 -8.55981 -1.40442 -5.20265
-2.64771 -4.87026 -1.78006 -2.39396
-2.93303 -0.858263 -1.91166 -0.350505
```

So, as we see, the joints are moving towards the final state, until at last reaches the final state, as it should.