Ariana Keeling

16-711 HW1

1/30/2017

Part 1:

[roll, pitch, yaw] = part1(target, link_lengths, min_roll, max_roll, min_pitch, max_pitch, min_yaw, max_yaw, obstacles)

The main program (part1()) optimizes some criteria within some constraints using the minimum joint limits as an initial guess. The constraints (const()) consist of the joint limits and the obstacles; the obstacle constraints are defined as a minimum distance of the sphere radius from both endpoints and the midpoint of each link to the center of the sphere. The criteria to be optimized (toOpt()) consist of the distance to the target, with a much smaller emphasis put on choosing angles that are far from the joint limits. The forward kinematics (kin()) are described as the distance in the x-direction in the rotated frame from the endpoint of one link to the endpoint of the next (as found in link_lengths); this is repeated for all links. A simple program converts an input quaternion to XYZ Euler angles (toEuler()). Once solutions have been found, a drawing program (roboDraw()) depicts the links as lines and the spherical obstacles.

The solution worked well for several different test cases.

Part 2:

[roll, pitch, yaw] = part2(target, link_lengths, min_roll, max_roll, min_pitch, max_pitch, min_yaw, max_yaw, obstacles)

This program (part2()) performs similarly to the first except in that it uses derivate functions for forward kinematics. Altered programs are denoted with a “WDiff” addition to the name.

The derivative function provides an answer faster, but requires more iterations than the original function. The solution is very different and arguably worse in that it includes a joint with a sharp angle which moves the joints into an undesirable pose as opposed to the original function where joint angles stay much further from the joint limits.

Part 3:

The interior-point algorithm found a solution in 38 iterations for a test case. The active-set option also completed quickly, requiring 57 iterations to return a trivially different solution. The sqp option completed the fastest, requiring 18 iterations to find the same solution as the active-set algorithm. I

was unable to convince Matlab to generate a gradient function for the optimization, so I was unable to evaluate the trust-region-reflective option. The CMA-ES implementation takes far more iterations (on the order of hundreds) to return a solution, although the algorithm solves as quickly as the others.

Part 4:

It might be possible to find different solutions to a given scenario by choosing different starting points (perhaps the minimum joint limits, maximum joint limits, and midpoint between the two) and evaluating the solutions against some criteria to determine whether they are critical. For example, if all of the joint angles are within $\pi/10$ of each other, the solutions may be trivial.