16-745 Assignment 1

Xianyi Cheng   Email: xianyic@andrew.cmu.edu   AndrewID: xianyic

To test the code, please run ***part1.m***, ***part2.m***, ***part3.m***, and ***part4_cmp.m***.

Part1

In this assignment, we solved the IK problem in the optimization way. In Part1, I first implemented the forward kinematics of the snake robot. Based on the FK, the distance from the robot joints to the obstacles can then be computed as constraints. At last, the criterion is defined and tuned. I use matlab function 'fmincon' to perform optimization.

Forward Kinematics

The forward kinematics is implemented according to [1]. The forward kinematics map is given by

$$\boxed{g_{st}(\theta) = e^{\widehat{\xi}_1\theta_1} e^{\widehat{\xi}_2\theta_2} \cdots e^{\widehat{\xi}_n\theta_n} g_{st}(0)}$$

where ξ_i is the twist of the ith rotational axis, and θ_i is the corresponding angle. For each link, the order is followed by roll, pitch and yaw.

Constraints

In this optimization problem, the constraints are:

1) Joint angle limit: the lower and upper bound of the variables.
2) Obstacle avoidance: the distances to the obstacles are larger than the obstacle radiuses.
3) The target position: the position should be equal to the target position. (Notice that normally, this constraint is actually satisfied by defining the distance from current position to the target as the cost to optimize. But since we are required to optimize the joint angles at the same time, it is better that this distance to be set as a constraint to ensure the robot reach the target.)

Cost function

The cost function is the function to optimized in this problem. We have two goals: reaching the target and keeping the joint angles as far as possible to the joint limits. Thus the cost function is defined as:

cost = C1*distance(target, current position) + C2*distance(joint angles, joint limits) where C1 and C2 are the coefficient of the importance of the terms. In my implementation, I let the C1 to be much larger than C2, since our priority is to reach the goal. We set C1 = 100, and C2 = 0.1.

Optimization

For part 1, I chose the optimization algorithm to be ‘sqp’ and kept the other optimization options as matlab defaults. For the initial point, I randomly sample it from (-pi, pi). I set the lower bound of the joint angles as -pi, and the upper bound as pi.

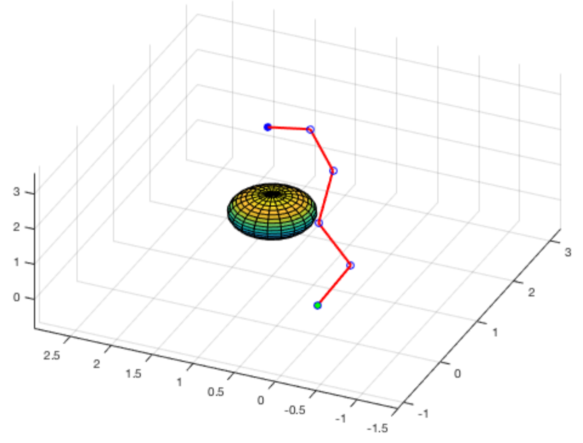
Optimization Performance

Robot: a snake robot with 5 links. The length of each link is 1.

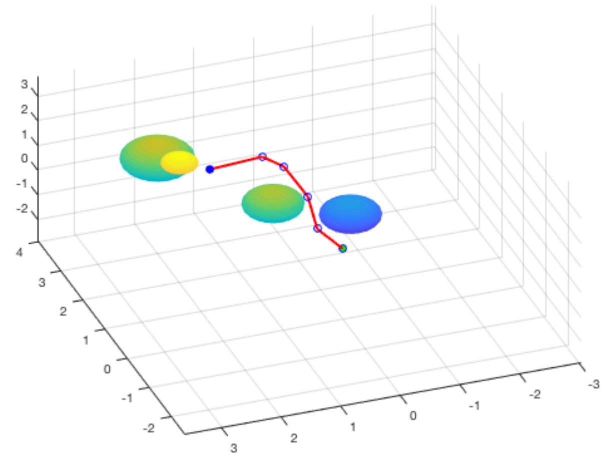
Obstacle: the balls as shown in the figure.

The figures below are some test cases and their optimized solutions.

Test case 1



Test case 2



Discussion

We can actually have multiple choices of what are the constraints and what in the cost function. Below are several options:

1) Without the constraint: target - current position = 0; Cost Function: C1*distance(target, current position) + C2*distance(joint angles, joint limits)
2) With the constraint: target - current position = 0; Cost Function: C1*distance(target, current position) + C2*distance(joint angles, joint limits)
3) Without the constraint: target - current position = 0; Cost Function: C2*distance(joint angles, joint limits)

For 1), we will take the risk of not even close to the target. Although letting C1 to be much larger than C2 might be a solution, it still cannot guarantee reaching the target. And by observation, the number of iterations required to reach the target is larger. For 2), the optimizer quickly reached the target, but it spent many iterations on optimizing the joint angles in very small steps, which might not be necessary in the practical use. For 3), the optimizer also reached the target quickly, but with bad initialization, it might not be able to find a solution that satisfy the position constraints.

Considering about the robustness and speed for optimization, we finally choose 2). However, other choices can also be used for this problem.

Part 2

In this part, I firstly compute the derivative of the tip position and orientation to the joint angles. Then I provide the derivative to the optimization routines.

Compute the derivative

The derivative/Jacobian is computed according to [1][2]. We can firstly get the Jacobian of the spatial twist:

$$J^s_{st}(\theta) = \left[\left(\frac{\partial g_{st}}{\partial \theta_1} g_{st}^{-1}\right)^\vee \cdots \left(\frac{\partial g_{st}}{\partial \theta_n} g_{st}^{-1}\right)^\vee \right].$$

$$\begin{aligned}
\left(\frac{\partial g_{st}}{\partial \theta_i}\right) g_{st}^{-1} &= e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} \frac{\partial}{\partial \theta_i}\left(e^{\hat{\xi}_i\theta_i}\right) e^{\hat{\xi}_{i+1}\theta_{i+1}} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0)\, g_{st}^{-1} \\
&= e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} (\hat{\xi}_i) e^{\hat{\xi}_i\theta_i} \cdots e^{\hat{\xi}_n\theta_n} g_{st}(0)\, g_{st}^{-1} \\
&= e^{\hat{\xi}_1\theta_1} \cdots e^{\hat{\xi}_{i-1}\theta_{i-1}} (\hat{\xi}_i) e^{-\hat{\xi}_{i-1}\theta_{i-1}} \cdots e^{-\hat{\xi}_1\theta_1}
\end{aligned}$$

Let the p = [x, y, z, q0, q1, q2, q3]' be the tip position and orientation, then from [2], we can get:

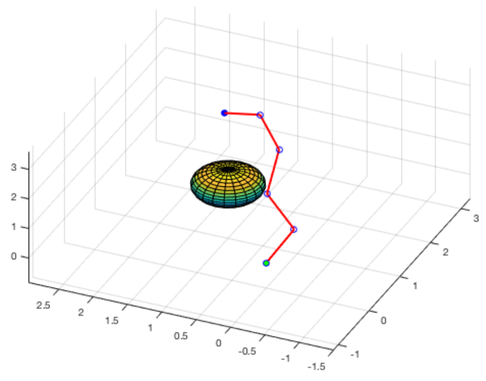
$$\frac{dp}{d\theta} = \frac{dp}{d\xi} \cdot J^s_{st}(\theta)$$

ξ is the spatial twist.

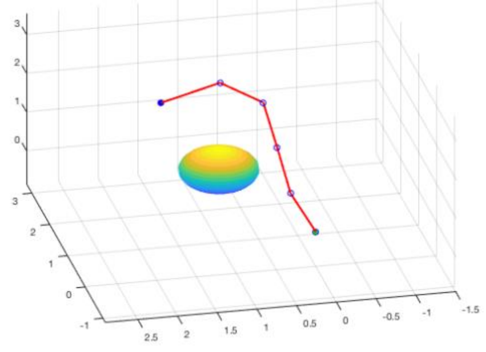
$$\frac{dp}{d\xi} = \frac{1}{2}\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & q_3 & -q_2 \\ -q_3 & q_0 & q_1 \\ q_2 & -q_1 & q_0 \end{bmatrix}$$

Discussion

Providing the derivative largely increases the performance of the optimization processes, converging much faster and solutions better. The number of iterations is almost a quarter of that without derivative. And the solution is much better. Below is the comparison:



without derivatives



with derivatives

The two figures are the solutions provided under the same target. The left one is the solution of optimization without provided derivatives, while the right one with derivatives. To solve the left one, we need about 170 iterations. But we only need about 35 iterations with derivatives.

Part 3

To measure the performance of each algorithm, I run all the optimization methods for several times with random sampling of the initial point. The performance is shown in the following table. (To make more convincing comparison, this experiments should be run for more times, but I can only run the comparison experiments for 3 times by the deadline of submission, sorry.)

Elapse time: the total time for optimization

feasible: if the algorithm finding a solution that satisfy all the constraints

average fval: the cost function value for the feasible solutions

# iter: the number iterations to run before the optimizer stopped

Trial 1	elapse time (s)	Feasible	fval	# iter
sqp	31.724056	yes	-414.718064	61

Interior-point	133.941372	yes	-483.685713	548
Active-set	/	No (far from)	/	/
CMA-ES	2257.435104	yes	-530.858607	1308

Trial 2	**elapse time**	**Feasible**	**fval**	**# iter**
sqp	159.278748	yes	-383.643979	79
Interior-point	239.994587	yes	-521.767953	1000
Active-set	/	No (far from)	/	/
CMA-ES	2037.748812	yes	-531.118766	1164

Trial 3	**elapse time**	**Feasible**	**fval**	**# iter**
sqp	140.176245	yes	-452.624998	97
Interior-point	189	No, but not far. Distance to the target: 0.102778	-503.332973	1000
Active-set	/	no	/	/
CMA-ES	4702.851626	yes	-533.613420	2268

Discussion about the performance of different algorithm

interior-point: this algorithm can compute each iteration very fast, but it actually spent more iterations in optimization.

sqp: computation speed for each iteration is much slower than the interior-point, but it can find feasible solutions very quickly.

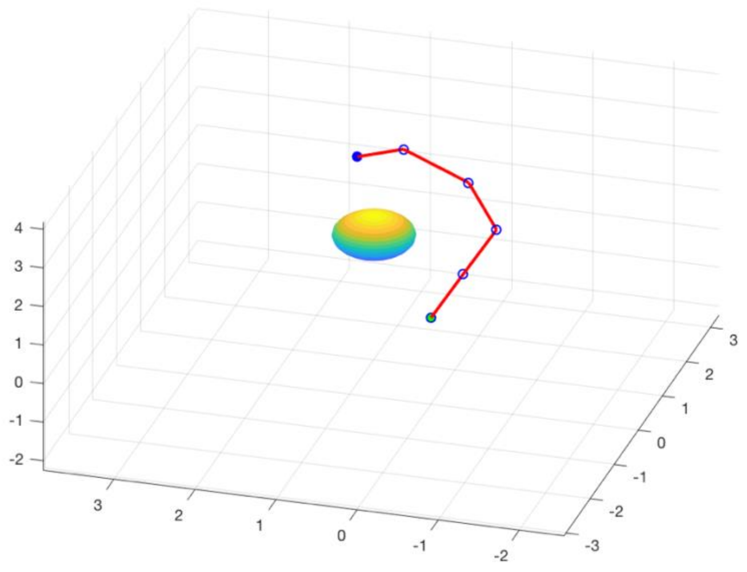
Active-set: this method takes very large steps in optimization. And cannot converge to the local minima that satify the constraints.

Cma-es: this method spent much longer time in optimization. But this method leads to very good solution, which might be much better than the others. The solution can both satisfy the constraints very nicely, and give better cost function value. **Mostly importantly, during our implementation, no matter what the initial point is, the cma-es method can always find the global optima.**

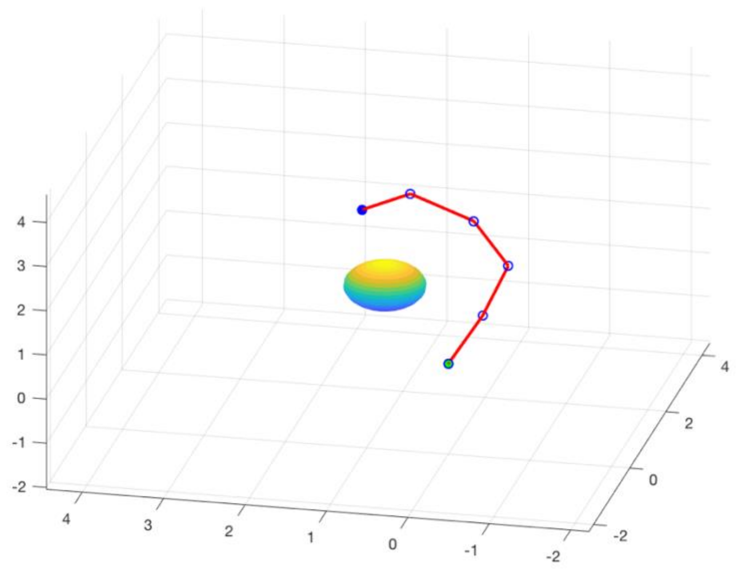
Part 4

The solutions provided by the optimizer are highly dependent on their initial points. So, the easiest way to provide multiple local optimal solutions is to start with different initial points. I run the optimization for 10 times with random sampling of the initial point. After obtaining these optimal solutions, I compared these solutions by the cost function values. The following figures are the visualization of several local minimas under the algorithm ‘spq’ with test case 1.

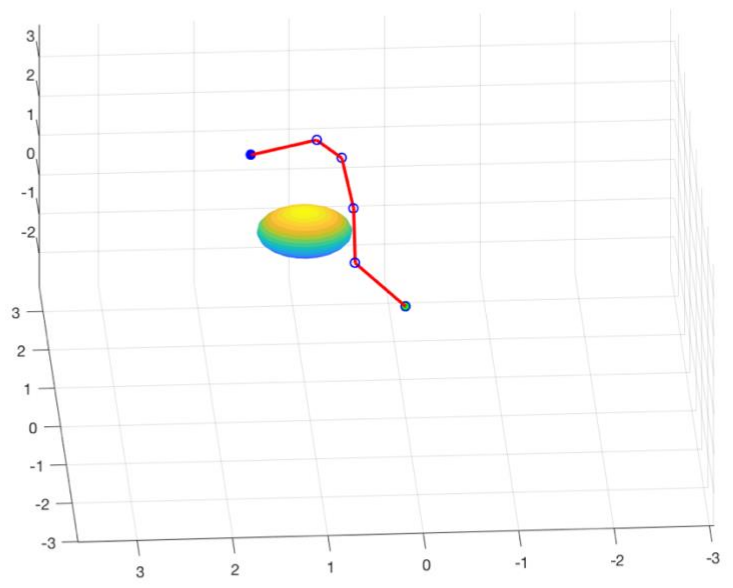
Best



2nd best



3rd best



Reference

[1] Murray, Richard M. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[2] Graf, Basile. "Quaternions and dynamics." *arXiv preprint arXiv:0811.2889* (2008).