Chris Hazard—Assignment 1

Input:

```
target= [4 2 1 eul2quat([0 0 1.5708])];
link_length=[1;1;2;3;1;1;1;2;1];
min_roll=[0,0,0,0,0,0,0,0,0]';
max_roll=[pi/2,pi/2,pi/2,pi/2,pi,pi/2,pi/2,pi,pi/2]';
min_pitch=[0,0,0,0,0,0,0,0,0]';
max_pitch=[pi/2,pi/2,pi/2,pi,pi,pi,pi/2,pi/2,pi/2]';
min_yaw=[0,0,0,0,0,0,0,0,0]';
max_yaw=[pi/2,pi/2,pi,pi/2,pi,pi/2,pi/2,pi/2,pi/2]';
```

See the output figures in the folder (part1fig.mat and part2fig.mat)

Part1 output

Comparison of part 1 to part 2:

Part 1

Objective value = 0.0023

r =

1.1487

1.3498

1.2318

1.3332

2.0818

0.9569

0.6052

1.8411

0.2792

p =

0.1620

0.4133

0.5916

1.2253

1.3169

2.4608

1.0864

0.6781

0.5413

y =

0.5710

0.1588

1.2112

0.2073

1.1834

0.4936

1.0318

1.0470

1.2694

Elapsed time is 40.559323 seconds.

Part 2

Objective function value = 6.1771e-13

r =

0.4017

0.3147

0.5724

0.3772

2.7012

1.1880

0.9529

0.5856

1.3626

p =

1.2177

0.4720

1.0685

1.8231

0.5925

0.6465

0.7271

0.1846

0.2702

y =

1.0172

1.2797

2.8037

1.2790

1.4942

0.9042

0.6103

1.0701

0.1496

Elapsed time is 56.530487 seconds.

Part 2 was a little slower but gave a better result (comparing the two objective function values). Part 1 just used fmincon without the gradient, and part 2 supplied the numerical gradient of the objective function with respect to each of the roll/pitch/yaw parameters for each joint. (There's also a function called autodiff in the folder, ignore that: I was going to use automatic differentiation by premultiplying homogenous transformation matrices and postmultiplying homogenous matrices, then you would get a vector for output position equaling C1*rotmatrix(variable)*C2*startpos, where startpos is [0 0 0 1]' and c1 and c2 are the constant pre and post multiplied transformation matrices---then the gradient of x,y, and z with respect to the variable (yaw, pitch, or roll of the current joint), is the derivative of positions 1 2 and 3 in the resulting matrix with respect to the variable: then the derivate of yaw, pitch,roll can be calculated by symbolically multiplying C1*rotmatrix(variable)*C2 to get the homogenous transformation, take the upper left 3x3 matrix,which is the rotation matrix, and convert it to yaw, pitch, roll symbolically and calculate the derivate of each of those with respect to the variable------ but decided it was much more work than just the numerical gradient

Part 3:

interior-point, sqp, active-set and CMA-ES algorithms

SQP:

Objective function =5.0122e-14

r =

0.7855

1.3944

1.2567

1.2197

0.1742

0.1477

1.4277

3.1113

1.5550

p =

1.4161

0.4203

1.0049

0.3718

2.8331

2.3758

0.1203

0.1361

0.1449

y =

0.8921

1.4869

2.3908

1.3891

1.6891

1.4807

1.0207

0.8588

0.3756

Elapsed time is 14.137460 seconds.

Interior-point:

Objective functions =2.1752e-11

r =

0.7416

0.7811

0.7050

0.3572

1.0358

1.2426

1.1088

1.6878

1.3822

p =

0.4911

0.8148

0.3341

1.4914

0.2910

0.7788

1.4316

0.7715

0.9783

y =

0.1012

1.2406

2.9851

1.2232

0.9142

0.3442

0.3507

1.3206

0.2436

Elapsed time is 37.387610 seconds.

Active set:

Objective value =1.6173e-08

r =

0.4636

1.4836

1.0538

1.0714

1.1395

0.2512

0.4107

0.2305

0.6947

p =

0.0183

1.1408

1.5265

1.7774

2.9141

0.5352

1.4209

0.6490

0.1228

y =

0.7990

0.9800

0.4030

0.3871

0.7815

1.5253

1.2788

1.0134

0.0738

Elapsed time is 13.963642 seconds.

It looks like SQP is the most accurate method with runtime comparable to the active set method, which is the fastest. The output figures are saved as sqp.fig, activeset.fig, etc. (By the way the code that generates these outputs is in test.m, which can call part1, part2, and cma-es)

For the CMA, the time obviously depends on how you configure the parameters (number of generations, population size, etc.) but for the parameters I have set its running time is comparable to the above methods, and its performance is significantly worse:

Iteration 300: Best Cost = 0.17003

That's because CMA isn't suitable for problems where gradients can be calculated either analytically or numerically and constraints can be easily represented and individually taken into account by the algorithm—CMA is best for very ill-behaved non-differentiable nonlinear optimization. Also note that the learning curve for the CMA is very sporadic: most of the time it is on a plateau,then it suddenly reaches a new plateau and continues on until it finds another one, so figuring out when to stop is tricky. (see CMA.fig)

Part 4

The best way to do this in general is to randomly seed the optimization with a bunch of initial points and rank them by how good they are and then have the user select one of the top n configurations. If you know nothing about where to look for the best set of control parameters, randomly picking them over some large range is about the best you can do (in part1 and part2 these were initialized uniformly over the range of allowable roll/pitch/yaw values), but if you do have a better idea of what region the best solutions lie in, you can change the shape of the distribution you use to sample these parameters to seed the optimization (like a Gaussian)---this will give you more points closer to the region you suspect has the most desirable local minima.