Assignment 1
Inverse Kinematics

16-711: Kinematics, Dynamics, and Control
Sumit Dan, Archit Dhanani, Daniel Piedrahita

16-745: Dynamic Optimization
Joe Norby
We created a model of a snake robot that orients the distal link (gripper) to a desired position and orientation while avoiding obstacles and satisfying joint limits and constraints. The model uses function optimization to compute the joint angles required to achieve these objectives. The model we created is detailed in part 1. Parts 2 and 3 explore modifications to the optimization method, via analytic derivatives and alternative algorithms, respectively. Part 4 explores the presence of multiple local minima.

**Part 1. Snake Robot Model**

Forward Kinematics:
In order to optimize for a set of joint angles that specifies a given robot configuration, the forward kinematics of an arbitrarily long snake robot was derived. The forward kinematics were derived assuming that the snake was constructed of N links, where N was between 1 and 10 links. In addition, it was assumed that each link was connected by a series of rotary actuators which rotated in the euler angles of roll, pitch, and yaw, in that order. After these assumptions were made, the forward kinematics were derived by utilizing a series of homogeneous transformation matrices. For each link, a homogeneous transformation matrix for the roll rotation, pitch rotation, yaw rotation, and link displacement is defined. Then, the overall homogeneous transformation matrix between the global coordinate frame and the end-effector coordinate frame can be computed by multiplying all of these transformation matrices together. Because this derivation is recursive, it is straightforward to write an algorithm that can calculate the forward kinematics for an N link snake.

Optimization:
In order to compute a set of joint angles that places our end-effector at a given final pose without a time-consuming and intensive derivation of the analytic inverse kinematics, we are posing the problem as an optimization problem. The optimization is trying to minimize a cost function while constrained by a set of nonlinear constraints by varying the joint angles of the robot.

Criterion Function:
Our criterion cost function is broken into two parts:

1) Final pose cost-
The final pose cost is computed by taking the 2-norm of the vector difference between the current pose (computed with forward kinematics) and the desired pose specified by the function. This cost is multiplied by a weight of 5 because it has high priority.

2) Joints near limit cost-
The joint limit cost is computed by first taking the mean of the joint limits for each type of joint (roll, pitch, yaw). Then, the current joint angle of each joint is subtracted by its joint limit mean, and then the 2-norm of this difference matrix is found. This cost is multiplied by a weight of 0.25 because it does not have very high priority.
The final cost of the solution of the snake robot is then the sum of the joint angle cost and the final position cost.

Constraint Function:
Our nonlinear constraint function is composed of one inequality violation. This inequality violation ensures that the snake robot never enters any of the obstacles. This was computed by first discretizing each link into a set of \( n \) points, where \( n = 10 \) in our algorithm. The distance of each of these points to every obstacle sphere is then computed, and the closest point to each obstacle is saved. If the closest point to the obstacle is within the obstacle then a check variable is set to +1, and if it is not within the obstacle it is set to 0. The check variables for each obstacles are then summed and saved as an inequality constraint. That way, if that sum is positive (some points are within obstacles), it will not satisfy the inequality constraint.

Bounds:
In addition to having a soft cost in the criterion function, the upper and lower bounds are also rigidly constrained by fmincon. The lower and upper bounds are saved for each roll, pitch, and yaw joints and then passed to the fmincon matlab function in the normal way. The fmincon function treats these bounds as constraints to the optimization routine.

Discussion:
We posed the optimization problem as a combination of cost criterion and constraints. In this way we can closely control what parameters have priority in any real world scenario. In our current configuration, we are implying that having the robot avoid joint limits and stay away from obstacles are of highest priority (they are defined as constraints). We believe that this is valid because when working with a real robot, keeping the robot from breaking (avoiding joint limits), and keeping the environment from breaking (avoiding obstacles) are of utmost importance. Once these constraints are satisfied, the optimization tries to minimize the cost criterions by moving the end-effector to the specified pose, and also (but with less importance) tries to keep joints as far from their limits as possible.

We initially tried keeping the final pose as a constraint to the minimization as well, but this did not work as we expected. It basically made the optimization take much longer, and it had a higher chance of erroring out because of the inability to find a solution. In hindsight, this makes sense seeing as it is not always possible to satisfy both of these constraints at the same time. This is why we ended up moving the final pose from a constraint to a cost criterion.

**Part 2. Analytic Derivatives**

To improve the performance of the optimization, gradients were introduced to the optimization algorithm. Since the cost function includes the forward dynamics of the snake robot, penalties on large joint angles, and an arbitrary number of links, finding analytical derivatives by hand would be exceedingly difficult. Instead, a finite difference approximation was implemented to
construct the gradient. Each joint angle was perturbed by a small amount, and the subsequent
changes in cost functions were divided by this small amount to yield the Jacobian, from which
the gradient was extracted. With even an approximation of the gradient, the algorithm has a
means to wisely choose the next point to evaluate (down the gradient), and finds the optimum
much more rapidly, as seen in Table 1. This change does not seem to have much of an effect
on the accuracy of the final solution. This makes sense as this method yields only an
approximation of the gradient, and therefore its accuracy is limited.

Part 3. Alternative Algorithms

To optimize the performance of the code, we tried different optimization algorithms and found
the results presented below. Using interior-point algorithm without providing gradient, the
computational time is very high. In comparison, providing gradient to this algorithms makes it the
computationally fast. Though the run time changed drastically, optimal function values for both
methods was relatively the same. Both sqp and active-set algorithms have similar performances
when running without gradient, but the sqp algorithm get significantly faster with gradient
information while active-set remains the same. The best performance (when looking at run
times) for the code was observed in the interior-point algorithm with gradient. CMA-ES, on the
other hand, is kind of stochastic method of optimization which does not need any gradient for
the optimization. Although CMA-ES is the most time consuming algorithm, it finds the most
optimal solution as compared to the other algorithms.

The algorithm ‘trust-region-reflective’ did not work since we had defined a non linear constraint.
This algorithm works with either bound constraints or linear constraints.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime without gradient (s)</th>
<th>Runtime with gradient (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>interior-point</td>
<td>100.8597</td>
<td>9.3533</td>
</tr>
<tr>
<td>sqp</td>
<td>69.231</td>
<td>16.2249</td>
</tr>
<tr>
<td>active-set</td>
<td>55.4535</td>
<td>42.8697</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>69.7241</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Figure 1: Solution with Interior Point Optimization Algorithm

Figure 2: Solution with SQP Optimization Algorithm
Part 4. Local Minima

For a given position and orientation, multiple local minima were determined by incorporating a cost to the previous poses of the snake robot and retaining a history of these final poses. After
the first solution (a specific set of joint angles) is determined through optimization, a cost is then associated to this specific solution and passed into the optimizer. In this way, the optimizer will attempt to find a new nontrivial solution in the sense that it will find as good of a solution as possible while being as different as possible from previous solutions.

However, qualifying nontrivial solutions as better or worse relative to each other is a matter that cannot be resolved without further information for real world applications. Depending on the specific scenario of the snake robot in real world applications, certain solutions will outshine others for reasons that can only be known if the conditions of the specific scenario are known. With regards to this assignment, the solution with the minimal criterion function evaluation is considered to be the best solution found.

We also considered finding new solutions by randomly varying the starting configuration of the robot, as this tends to lead to unique solutions, but thought that our approach was more systematic in ensuring that new solutions are quantitatively different that previous ones.

Figures of 5 Non-Trivial Solutions Below:
Figure 6: Nontrivial Solution 2

Figure 7: Nontrivial Solution 3
Figure 8: Nontrivial Solution 4

Figure 9: Nontrivial Solution 5. This is the best solution when looking at criterion function value.