Homework 1
By: Puneet Singhal

Frames are defined at the tip of each link with one extra (first) frame defined as fixed base of the robot. So for a manipulator with n joints, there are n+1 frames with n+1th frame as end effector. Pose of end effector is defined as **p: [x, y, z, q0, q1, q2, q3].**

Euler angles at each joint j_i are used to find homogenous transformation of frame (i) and frame (i+1) as follows:

$$H_{i+1}^{i} = \begin{matrix} R_{i+1}^{i} & R_{i+1}^{i} * \begin{bmatrix} l_i \\ 0 \\ 0 \end{bmatrix} \\ 0 & 1 \end{matrix}$$

Vector x is defined as a stack of [roll angles; pitch angles; yaw angles];

Given a target position of end effector, x_d , the inverse kinematics is done as a search for joint angles that minimizes the **cost**: $(p_d – p)^T Q(p_d – p) + w*(\text{deviation from joint limits})^{-1}$

MATLAB function **fmincon** was used to carry on the optimization.

Joint limits are additionally modelled as hard constrained by putting the bounds in fmincon arguments.

Obstacles were modelled as non-linear constraints (number of links X number of obstacles). The constraint is calculated by solving the equation of each link as line segment with each sphere. The constraints are modelled as hard inequalities and passed into fmincon argument.

Part 1. The results of optimization/ inverse kinematics are as:

	target	3.000	3.000	2.000	0.707	0.408	0.408	0.408
Without gradient	position 1	3.007	3.000	2.013	0.708	0.391	0.431	0.400
	error 1	0.0235%						
	run time 1	111.03						

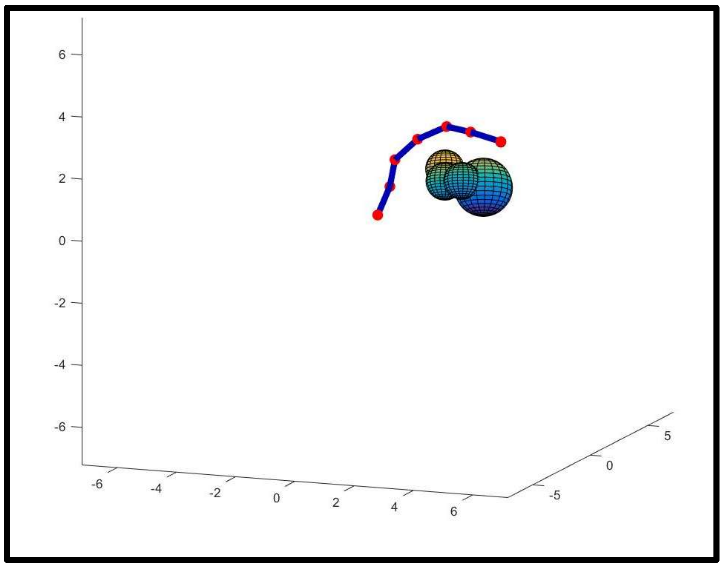


Figure 1: final pose of the robot without including gradient in the optimizer

Part 2. In this part, gradient of cost was calculated with respect to each joint angles and included into the optimizer, fmincon. Upon several unsuccessful attempts on using standard automatic differentiation toolboxes (failed due to constraint on input argument type), I finally wrote a small code to find gradient using "forward finite difference" approach: calculate the delta changes in end effector pose upon forward perturbation of one joint angles at a time.

The results of optimization/ inverse kinematics are as:

	target	3.000	3.000	2.000	0.707	0.408	0.408	0.408
With Gradient - "interior -point"	position 2	3.005	2.988	2.057	0.720	0.336	0.453	0.405
	error 2	0.2245%						
	run time 2	12.88						

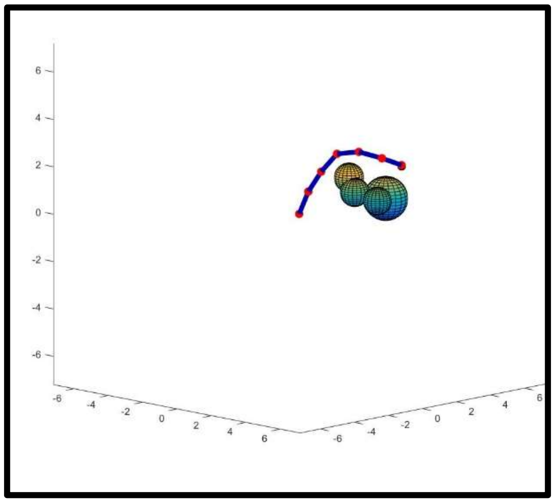
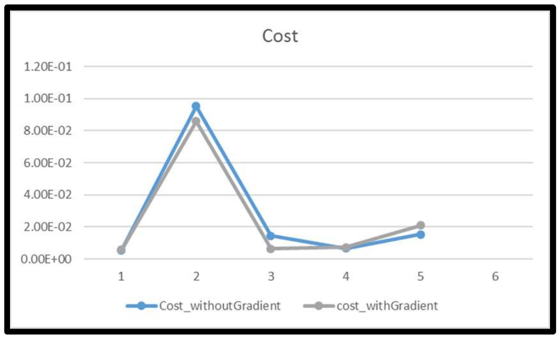


Figure 2: final pose of manipulator after including gradient in the optimizer and using "active-set" Algorithm in fmincon

Few trials were conducted to compare the results with and without including gradient.



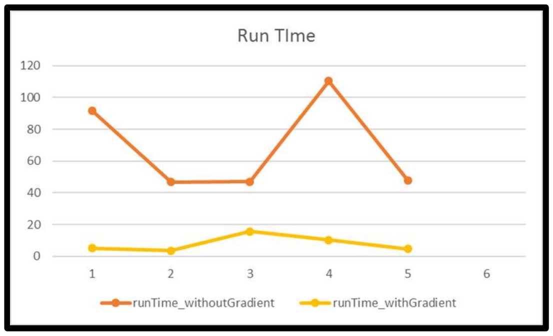


Figure 3: comparison of cost and run time between optimizer without and with gradient

Although the cost is similar in two cases, the run time has significantly come down.

Part 3. In this part, a new optimizer is explored, CMA-ES (*Covariance Matrix Adaptation Evolution Strategy)*. The optimizer is taken from internet and remains unchanged except decreasing the number of MaxIterations.
CAMES also takes lower and upper bounds as hard constraints but other inequalities were modelled inside the cost function. The cost function now returns NaN whenever the constraints related to obstacles avoidance get violated. This works well as the optimizer will just ignore those points.

The results of optimization/ inverse kinematics including the comparison table with different algorithms inside fmincon are shown below:

	target	3.000	3.000	2.000	0.707	0.408	0.408	0.408
CMAES	position 3	3.017	3.006	1.989	0.696	0.416	0.377	0.447
	error 3	0.0646%						
	run time 3	94.28						
interior-point	position 2	3.005	2.988	2.057	0.720	0.336	0.453	0.405
	error 2	0.2245%						
	run time 2	12.88						
SQP	position 3	3.002	3.001	2.001	0.708	0.406	0.413	0.405
	error 3	0.000864%						
	run time 3	3.10						
active-set	position 3	3.000	3.000	2.000	0.707	0.408	0.408	0.408
	error 3	0.0000000513%						
	run time 3	3.06						

Although cmaes gave better results in terms of reducing the relative error by 1 order, it did not do that well in terms of run time. Also, it was observed that increasing the bound on Max Iterations improves the error further but at a big expense of run time cost.

The best results were found wwhile using “Active-Set” algorithm inside fmincon.

The order of performance in terms of run time was observed as:

$$Active\ Set > SQP > Interior\ Point > CMAES > Interior\ Point\ (excluding\ Gradient)$$

The order of performance in terms of accuracy was observed as:

$$Active\ Set > SQP > CMAES \sim Interior\ Point\ (excluding\ Gradient) > \ Interior\ Point$$

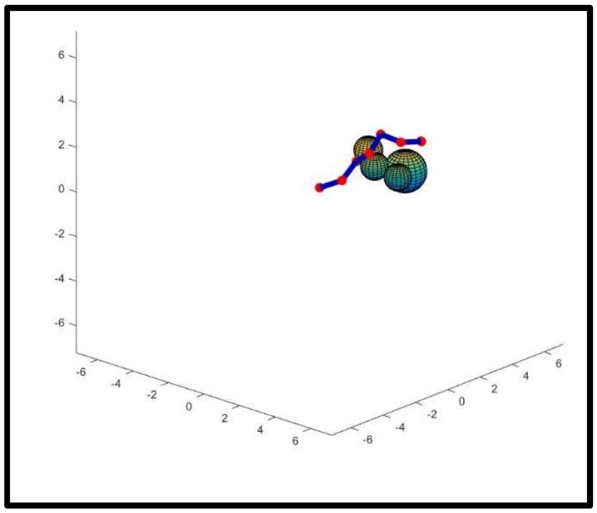


Figure 4: final robot pose using CMAES

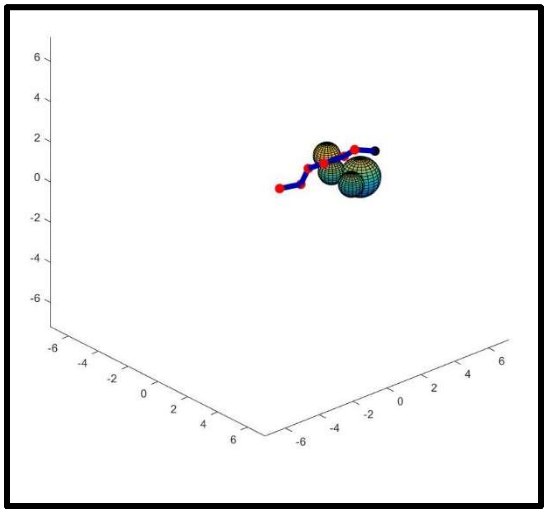


Figure 5: Final robot pose using "active-set" Algorithm in fmincon

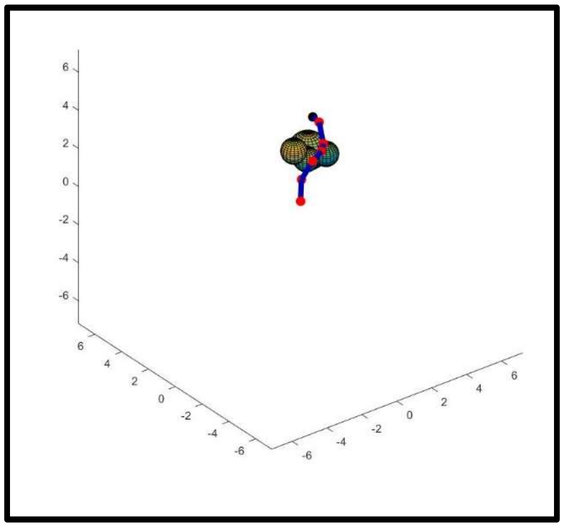


Figure 6: Final robot pose using "SQP" algorithm in fmincon

Part 4. In this part, the fmincon was used with “active-set” algorithm to catch all the local minimum that comes while the optimizer is running. The local minimas with decreasing cost points (left to right) are as:

	set 1	set 2	set 3	set 4	set 5	set 6
cost	0.057445	0.02809	0.017828	0.010118	0.005075	0.001787
r1 (deg)	11.49792	10.34167	11.24301	11.7638	11.53658	12.3193
r2 (deg)	21.75383	21.90032	23.69133	21.719	22.10394	22.55699
r3 (deg)	30.17478	31.81528	31.04233	32.04301	33.31883	35.28631
r4 (deg)	36.06649	36.59041	32.81675	34.1792	32.55745	30.40239
r5 (deg)	50.37528	51.92784	53.17439	51.81438	51.55295	51.10728
r6 (deg)	50.70236	47.3133	47.89691	45.83603	47.79836	48.27633
p1 (deg)	51.66504	55.22651	51.31115	51.85024	50.55845	49.15293
p2 (deg)	59.52149	58.11626	55.24543	54.86866	54.3456	53.10545
p3 (deg)	66.67028	63.41623	60.40981	64.30647	61.78014	60.00572
p4 (deg)	68.46077	66.10622	67.76618	69.64268	68.00642	67.48325
p5 (deg)	68.48818	65.38836	69.36519	68.79414	69.83116	71.68023
p6 (deg)	68.81147	67.88147	67.14503	66.83959	67.43816	67.40812
y1 (deg)	73.6793	71.91129	75.48768	71.77261	71.62646	70.93161
y2 (deg)	81.34203	82.79342	84.44238	82.25654	83.7536	84.63876
y3 (deg)	87.04177	85.16859	87.60818	91.15016	93.14556	96.45088
y4 (deg)	114.1434	113.422	113.1507	112.5001	112.3426	111.3539
y5 (deg)	120.4935	118.8234	119.915	121.3517	120.7795	121.2109
y6 (deg)	141.6468	138.1486	136.471	139.4983	137.7669	136.0691

For the similar accuracy, there is a variation of up to 10 degrees in joint angles. Depending on the requirement, we can make use of these local minimas instead of disposing them off during optimization.

Learning:

- FMINCON is a powerful and vast toolbox. It can be used for number of applications and depending on options chosen, user can optimize on accuracy and performance.
- CMAES: More study is required in this domain to understand the tool and concept better. In limited time, a standard tool was used without understanding the operation of this technique. A new method of returning NaN in cost function was identified to incorporate inequalities in CMAES. Finally, this technique is being used in all parts of the problem set.
- A lot of **automatic differentiation** toolboxes were explored for eg. AutoDiff in MATLAB library. Due to **specific input argument** type requirement, I was not able to use them for my application. Instead, finite difference approach was used to calculated gradient.