OPTIMAL WORK SOLUTION OF INVERSE KINEMATICS OF 3-LINK OPEN LOOP MECHANISM

Kevin Jose (12352), Hiren Mittal (12302), Hardik Soni (12280), Abhishek Khandelwal (12023)

ME752A: Optimization Methods in Engineering Design

Department of Mechanical Engineering, Indian Institute of Technology Kanpur
E-mail: kevinj@iitk.ac.in, hireni@iitk.ac.in, hardikv@iitk.ac.in, abhikhan@iitk.ac.in

ABSTRACT

In this paper, we have optimized the work input (minimize) to a 3-link open loop mechanism when it moves from some initial position to a desired position. This is done using a combination of Nedler-Mead Simplex search algorithm with a Penalty function. The 3-link open loop mechanism can be the modeling of a robotic arm which is given some input to perform a task involving moving from one location to another. So, the result we obtain can be applied for optimizing the work done by the robotic arm in moving from one point in space to other. But we are restricting the motion of mechanism to only one plane in this paper.

INTRODUCTION

A robotic arm is a robot manipulator, usually programmable, with similar functions to a human arm. The links of such a manipulator are connected by joints allowing either rotational motion or translational displacement. The links of the manipulator can be considered to form a kinematic chain. The business end of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand. The end effector can be designed to perform any desired task such as welding, gripping, spinning etc., depending on the application.

Serial manipulators are the most common industrial robots. They are designed as a series of links connected by motor-actuated joints that extend from a base to an end-effector. In its most general form, a serial robot consists of a number of rigid links connected with joints. Simplicity considerations in manufacturing and control have led to robots with only revolute or prismatic joints and orthogonal, parallel and/or intersecting joint axes (instead of arbitrarily placed joint axes).

We are designing the algorithm for actuation of a 3-link serial manipulator (which uses servo motors for precise movements). We are assuming the weight of each link including the motor to be 1 unit and the length of the entire link are taken to be 1. The torque on each motor is calculated which in turn is used to calculate the work. In this problem we are optimizing for total work input in all 3 motors using Nedler Mead Simplex search method.

OBJECTIVE FUNCTION

Since we seek the optimal work solution of the inverse kinematics of the 3-link open loop mechanism, the objective function is an expression of work for actuation from an initial position (0, π, 0) to any final position (Ω₁, Ω₂, Ω₃). A simplifying assumption is made i.e. that the motors are actuating sequentially (one after the other).

We have derived the expression for work from the T-w characteristics of a generic motor. The detailed expression of the objective function and its derivation is attached in the Appendix.

To reduce computational difficulty during the derivation, a function (of t vs. θ, existing in an intermediate step) is approximated a function in the derivation process by using a polynomial curve fit. In another instance, the derivation yields an expression which contains Imaginary Error Function (erfi) which is not a standard function defined in MATLAB, we handled that by approximating it over the range on which we required its value. The exact procedure is elucidated in the Appendix.

CONSTRAINTS

The main constraint at play is the location of the end effector, which can be expressed as:

\[ x = \cos(\Omega_1) + \cos(\Omega_2) + \cos(\Omega_3) \]
\[ y = \sin(\Omega_1) + \sin(\Omega_2) + \sin(\Omega_3) \]

where (x, y) is the desired location of the end effector as specified by the user.

These constitute equality constraints.

We also constrained the angles to executing a motion through (-π, +π) only. This is handled by formulating them as inequality constraints. (Ideally, this constraint must have been implicit in the minimization of work done i.e. avoiding redundant rotation beyond 2π but we included it nonetheless in the interest of faster convergence)

OPTIMIZATION TECHNIQUE

We have utilized the Nedler & Mead Simplex Search (NMSS) for optimization wherein the constraints are handled using an external penalty function. We decided to employ NMSS so that we can use our algorithm to handle not just the sample work function we derived but for any function which the user is free to define (which is expected to change with change in length of links and the specifications of the motors). We have purposefully avoided using a gradient based method so as to keep computational load to the base minimum and still maintain the ability to handle functions whose gradients may be badly defined.

By observing many test points α=2.3-2.5 and β=0.8-0.9 seems to be a good set of expansion and contraction factors.

The initial simplex is
The penalty constant $c$ varies from $10^{-6}$ to $10^{6}$.

The MATLAB code and the penalty function are attached in the Appendix.

RESULTS

Given below are the results of the performance of our algorithm over different end effector location.

(i) - Shows the link locations calculated by the algorithm in successive iterations superposed in a single image

(ii) - Shows the final optimal link location computed by the algorithm

(iii) - Shows the work done on achieving the configuration (link location) computed in respective iteration (X: Iteration No., Y axis: Work)

(iv) - Shows the sum of squares difference between end effector in a particular iteration and desired end effector location (X: Iteration No., Y axis: Location error)

Case I: (0.95, 1.2)

Case II: (-1, 2)
DISCUSSION

Dependence on Initial Simplex
To study the dependence of performance of our algorithm on different Initial Simplices, we ran our code for case I using 3 different Initial Simplices:

1) \[[0, 0, 0] [0.01, 0, 0] [0, 0.01, 0] [0, 0, 0.01]\]

2) \[[-0.2, 0, 0] [-0.21, 0, 0] [-0.2, 0.01, 0] [-0.2, 0, 0.01]\]

3) \[[0.5, 0, 0] [0.51, 0, 0] [0.5, 0.01, 0] [0.5, 0, 0.01]\]
The results:

**Link Positions**

1) ![Graph 1](image1)
2) ![Graph 2](image2)
3) ![Graph 3](image3)

It can be seen that even with different initial Simplices, our algorithm converges to points which may yield different link positions but the work done in all of them converges fairly well to the same value which suggests that our algorithm is robust even when multiple minima exist.

**Cost (Work)**

1) ![Graph 4](image4)
2) ![Graph 5](image5)
3) ![Graph 6](image6)

Path taken by the algorithm

By looking at (i),(iii),(iv), one can see that algorithm ‘jumps’ from one optimal point to the next in the initial iterations (focus on optimality) and the Location Error is high, but in later iterations the changes are gradual and more geared towards correct placement of end effector (geared towards feasibility). This is much more apparent in the videos hosted on the link given in ADDITIONAL RESOURCES. This behavior is as expected from penalty based methods.

**Extension to 3D**

While we have written the code for 2D motion of the mechanism, the code can be modified with little effort to handle 3D problems or even more linkages.

**ADDITIONAL RESOURCES**

For additional files related to this paper and its results, kindly visit the link given below:
[http://home.iitk.ac.in/~kevin/ME752A/](http://home.iitk.ac.in/~kevin/ME752A/)

Contents include all the `.m` files, pictures and videos of the algorithm being executed for multiple cases.

**REFERENCES**

1) Applied Mathematical Methods, Bhaskar Dasgupta
2) Optimization for Engineering Design, Kalyanmoy Deb
3) WolframAlpha

**AKNOWLEDGEMENT**

We would like to sincerely thank Prof. Bhasker Dasgupta and his TAs (Abhishek Ranjan, Ajinkya Jain and Yashesh Dhebar) for their unreleenting help and guidance without which this project would not have been possible.
function p=NMSSearch()
%__________________________________________________________________________
% NMSS to optimize 'f' with penalty function 'er' (which takes care of the
% constraints).
% Also uses 'plot_manupulator' to generate plots.
% 'f', 'er' and 'plot_manupulator' are functions defined seperately.
%__________________________________________________________________________
clc;
clear;
tic;

x=-1;
y=1.2;

e1=[.01 ;0 ;0 ];
e2=[0 ;.01 ;0 ];
e3=[0 ;0 ;.01];
e4=[0 ;0 ;0 ];

a=2.5;
b=0.9;

A=[e1,e2,e3,e4];
eps=0.000001;
c=eps;

cost1  = coder.nullcopy(zeros(1,100));
locerr = coder.nullcopy(zeros(1,100));

kev=1;
store=(sum(A')/4);

while(c<1000000)
  while(1)
    val=[f(A(:,1),c,x,y),f(A(:,2),c,x,y),f(A(:,3),c,x,y),f(A(:,4),c,x,y)];
    [wval,i]=max(val);
    bval=min(val);
    swval = max(val(val~=max(val)));
    cen=(A(:,1)+A(:,2)+A(:,3)+A(:,4)-A(:,i))/3;
    refpt=2*cen-A(:,i);
    refptval=f(refpt,c,x,y);
    flag=0;

    if refptval<bval
      xn=(1+a)*cen-a*A(:,i);
      flag=1;
    elseif refptval>swval
      xn=(1-b)*cen+b*A(:,i);
      flag=1;
    else if (swval<refptval && refptval<wval)
      xn=(1+b)*cen-b*A(:,i);
      flag=1;
    end
  end
end
if flag==0
    xn=(1+b)*cen-b*A(:,i);
end
val(i)=f(xn,c,x,y);
A(:,i)=xn;
Q=(0.25*sum((val-f(cen,c,x,y)).^2))^.5;
if Q<=eps;
    break;
end

KJ=(sum(A')/4)';

cost1(kev)=f(KJ,c,x,y)-c*er(KJ,x,y);
locerr(kev)=er(KJ,x,y);
store=[store;(sum(A')/4)];
kev=kev+1;

plot_manupulator(1,1,1,KJ,x,y);
pause(0.15);
c=c*2;
end
store;
figure
plot_manupulator(1,1,1,KJ,x,y);
figure
plot(locerr(1:kev-1));
figure
plot(cost1(1:kev-1));
toc;
function v = f(a,c,x,y) %Define function to be minimized
v = ((-9.0317*(erfim(0.1116-0.1456*a(1))-0.126458)) + (-50.623*(erfim(0.1726-0.0255*a(2))-0.197)) + (-15.062*(erfim(0.1937-0.08544*a(3))-0.2213)) )^2 + c*er(a,x,y);
end

function l = er(a,x,y) %Penalty Function
Xf= cos(a(1))+cos(a(2))+cos(a(3));
Yf= sin(a(1))+sin(a(2))+sin(a(3));
l=(Xf-x)^2+(Yf-y)^2 + max(0,a(1)-3.14)^2 + max(0,-a(1)-3.14)^2 + max(0,a(2)-3.14)^2 + max(0,-a(2)-3.14)^2 + max(0,a(3)-3.14)^2 + max(0,-a(3)-3.14)^2;
end
function plot_manipulator(L1,L2,L3,theta,x,y)

    l = linspace(0,L1,1000);
    m = linspace(0,L2,1000);
    n = linspace(0,L3,1000);
    % np: number of patches
    x1 = cos(theta(1));
    x2 = cos(theta(2));
    x3 = cos(theta(3));
    y1 = sin(theta(1));
    y2 = sin(theta(2));
    y3 = sin(theta(3));
    X1 = x1*l;
    X2 = x2*m;
    X3 = x3*n;
    Y1 = y1*l;
    Y2 = y2*m;
    Y3 = y3*n;
    for i = 1:1000
        X2(i) = X2(i)+X1(1000);
        Y2(i) = Y2(i)+Y1(1000);
    end
    for i = 1:1000
        X3(i) = X3(i)+X2(1000);
        Y3(i) = Y3(i)+Y2(1000);
    end
    X = [X1 X2 X3];
    Y = [Y1 Y2 Y3];
    hold on
    axis([-3.5 3.5 -1 3.5]);
    plot(X,Y)
    plot(x,y, 'r*');
    plot(0,0,'g*');
    hold off
Approximation of Imaginary Error Function

To approximate the value of Imaginary Error function over the range required. We identified 10 equally spaced points in the range and stored in ‘a’ and calculated the value of ERFI at each of these points using Wolfram Alpha and stored it in ‘b’. We then proceeded to fit a 3rd order polynomial to it.

> a
a =
-0.8400
-0.6800
-0.5200
-0.3600
-0.2000
-0.0400
0.1200
0.2800
0.4400
0.6000

>> b
b =
-1.2271
-0.9040
-0.6442
-0.4245
-0.2287
-0.0452
0.1361
0.3244
0.5305
0.7679

>> polyfit(a,b,3)

ans =
0.4849 -0.0062 1.1102 0.0009

\[
\text{ERFI}(x) \approx 0.4849x^3 - 0.0062x^2 + 1.1102x + 0.0009
\]

Figure 1: Polynomial curve fit for Imaginary Error Function
Consider a motor with following \(w \times T\) Torque characteristic:

\[ w \text{ (rpm)} \]

\[ \begin{array}{c}
\text{100} \\
\text{50} \\
\text{0}
\end{array} \]

\[ \begin{array}{c}
\text{T (in.-lbs)} \\
\text{13} \\
\text{14}
\end{array} \]

\[ \Rightarrow w = 100 \left(1 - \frac{T}{12.5}\right) \quad [T \text{ in lb-in}] \]

\[ \Rightarrow T = 12.5 \left(1 - \frac{w}{100}\right) \]

\[ \therefore T = 1.5 \left(1 - \frac{w}{100}\right) \quad [T \text{ in N-m}] \]

Now, consider the initial state when the first link is at \(\theta_1 = 0^\circ\), and second link is at \(\theta_2 = 180^\circ\) and third link is at \(\theta_3 = 0^\circ\). We will assume that at a time only 1 motor operates with the first motor operating first, then second motor and finally third motor so as to make the mechanism reach the final state.

All the three motors have the same speed \(w\) and torque characteristic as mentioned above (assumed).

Now, we will find the expressions for work done by each motor.

\[ T = I_\alpha = \frac{(3mL)^2 \alpha}{3} \]

Also, \[ T = 1.5 \left(1 - \frac{w}{100}\right) = \frac{3mL^2 \alpha}{3} \]

\[ \Rightarrow \text{Now, } \alpha = \frac{dw}{dt} \]

\[ \Rightarrow \text{On rearranging, } \left(\frac{3mL^2}{4.5}\right) \frac{100}{(100-w)} \frac{dw}{dt} = dt \rightarrow (\star) \]
\((*)\) - Governing differential equation.

Integrating \((*)\) gives,

\[-\frac{300 \text{ ml}^2}{4.5} \ln \left(\frac{100-w}{100}\right) = t\]

\(\Rightarrow w = 100 \left[1 - \exp\left(-\frac{4.5}{300 \text{ ml}^2} t\right)\right]\) \(\quad \text{(1)}\)

again, on integrating \((1)\) wrt time, we get:

\(\Theta = 100 \left\{ \frac{t}{4.5} + \frac{300 \text{ ml}^2}{4.5} \exp\left(-\frac{4.5 t}{300 \text{ ml}^2}\right) - \frac{300 \text{ ml}^2}{4.5} \right\}\) \(\quad \text{(2)}\)

By curve fitting in MATLAB, using command "polyfit" we find \(T(\Theta)\) in quadratic form.

\(\Rightarrow T = -1.4137\Theta^2 + 2.1664\Theta + 0.1124\)

Putting it in \((1)\) and then substituting \(w\) in \(T\)'s \(w\) characteristics eqn,

\(T = 1.5 \exp\left(-1.4137\Theta^2 + 2.1664\Theta + 0.1124\right)\)

Now, \(\frac{dw}{d\Theta} = T d\Theta\)

\(\Rightarrow w = \int T d\Theta\)

\(= 1.5 \int \exp\left(-1.4137\Theta^2 + 2.1664\Theta + 0.1124\right) d\Theta\)

\(\Rightarrow w_1 = -9.0317 \left[\exp\left(0.1116 - 0.1456\right) - 0.126458\right]\) \(\quad \text{(iii)}\)

Similarly, repeating all the above steps for second motor and third, we get:

\(w_2 = -50.623 \left[\exp\left(0.1726 - 0.02558\right) - 0.197\right]\) \(\quad \text{(iv)}\)

[here \(I_2 = \frac{(2m)^2}{3}\)]

and

\(w_3 = -15.062 \left[\exp\left(0.1937 - 0.08544\right) - 0.2213\right]\) \(\quad \text{(vii)}\)

[here \(I_3 = \frac{m}{3}\)]
Here erfi(x) is imaginary error function and,
\[ \text{erfi}(x) = i \text{erf}(ix) \]

erfi is not a defined standard function in MATLAB 2012.

So, we first found out the bound of domain of this function in our application.

It came out to be \((-0.84, 0.6)\)

Now, we took ten points in this range, to find the value of \(\text{erfi}(x)\) at these points and plotted them.

Again using polyfit we found a third order polynomial to curve fit \(\text{erfi}(x)\) in this domain.

The results of which are attached and it can be seen that the third order polynomial fits very well with the erfi.

So now, we replaced \(\text{erfi}(x)\) with the found third order polynomial for the optimization problem.