

**EMARO+ M1  
European Master on Advanced Robotics**

**Master CORO M1  
Master in Control and Robotics**

**Project Report**  
13 June 2018

**Design and Optimization of 3-RRR Planar Parallel Manipulator**

Amit KUMAR

Muhammad MUJTHID

Contents

[Introduction: 3](#_Toc516697644)

[Manipulator Architecture: 3](#_Toc516697645)

[Kinematic Modelling 4](#_Toc516697646)

[Condition Number of the Kinematic Jacobian Matrix 5](#_Toc516697647)

[Optimization Problem 6](#_Toc516697648)

[The Matlab Files: 7](#_Toc516697649)

[CAD Design: 11](#_Toc516697650)

[a) Individual Parts 11](#_Toc516697651)

[b) Product 13](#_Toc516697652)

[Observation 14](#_Toc516697653)

[References 15](#_Toc516697654)

# Introduction:

In this project, design optimization of planar parallel manipulator.is done. The design problem is solves by changing it into an optimization problem.

The goal was to minimize the size of the manipulator while keeping the design constraints in check.

The constraint for the problem is to have a workspace of a cylindrical regular workspace of diameter equal to 100 mm, its height corresponding to the range of rotation of the moving-platform. The range of rotation of the moving platform should be higher than 60◦ throughout the regular workspace. The inverse condition numbers of its normalized forward and inverse Jacobian matrices should be larger than 0.1 throughout the regular workspace. The manipulator is compact, i.e., the overall size of the manipulator is optimized and it is now as small as possible we designed. For a force equal to 10 N applied on the moving-platform along the normal to the plane of motion in its home configuration, the point-displacement of the moving-platform

should be lower than 0.1 mm.

For this project, 3RRR model selected for analysis, where prismatic joint is actuated.

## Manipulator Architecture:

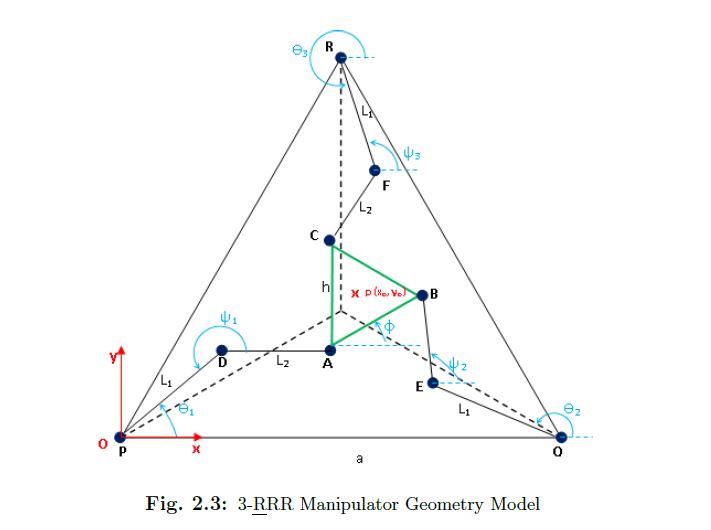


Figure 1:Manipulator Geometric Model

The 3-RRR manipulator each of the kinematic chains is the RRR-type and consists of three revolute joints. The 1st revolute joints of each legs are acted as actuators and attached to the base at point P, Q,and R. This manipulator is intended to position and to orient the equilateral triangle shaped

platform ABC in the plane of motion. The geometric center of the moving platform ABC is denoted by P, which is the operation point of the manipulator. The rotation angles of the three actuate revolute joints, i.e., , , and , are the input variables while the Cartesian coordinates of point P, i.e., xp and yp, and the orientation of the moving platform, i.e., , are the output variables. The base-platform is also an equilateral triangle with vertices P, Q, and R. Point O is the origin of reference frame. Below are the parameters describing the 3-RRR manipulator geometry:

Here are the parameters describing the manipulator geometry:

1. L1 length of \_rst intermediate links, i.e., L1 = PD = QE = RF;
2. L2: length of second intermediate links, i.e., L2 = DA = EB = FC;
3. a: a side length of the triangle-shaped base platform PQR, , a = PQ = QR= RP.
4. h: a side length of the triangle-shaped end-effector platform ABC, , h = AB= BC= CA;
5. rout: the cross section outer radius of both 1rst and second intermediate links.
6. rin: the cross section inner radius of both 2rst and second intermediate links.
7. rTool: the cross section radius of end-effector platform link.

# Kinematic Modelling

Where , i=1,2,3

=0+ (2)

Where, , multiply equation with

(3)

Transforming equation 3 into matrix form

A= , B=

# Condition Number of the Kinematic Jacobian Matrix

Geometric Constraints

The first constraint is related to the mechanism assembly,namely,

*L*b + *r* ≥ *R/*2 (i)

In order to avoid intersections between prismatic joints, the lower and upper bounds of the prismatic lengths are defined as follows:

0 *< < (ii)*

Condition Number

The condition number *κ*F(**M**) of a *m*×*n* matrix **M**, with *m* ≤ *n*, based on the Frobenius norm is d fined as follows,

*κ*F(M) = (iii)

Here, the condition number is computed based on the Frobenius norm as the latter produces a condition number that is analytic in terms of the posture parameters whereas the 2-norm does not. Besides, it is much costlier to compute singular values than to compute matrix inverses.

The terms of the direct Jacobian matrix of the 3-PRR PPM are not homogeneous as they do not have same units. Accordingly, its condition number is meaningless. Indeed, its singular values cannot be arranged in order as they are of different nature. However, from (i) and (ii), the Jacobian can be normalized by means of a normalizing length. Later on, the concept of characteristic length was introduced in(iii) in order to avoid the random choice of the normalizing length. For instance, the previous concept was used in finding the condition number of 3RRR parallel manipulator to analyze the kineto-static performance of with multiple inverse kinematic solutions, and therefore to select their best working mode.

Accordingly, for the design optimization of 3-RRR, the minimum of the inverse condition number of the kinematic Jacobian matrix, *κ*−1(**J**), is supposed to be higher than a prescribed value, say 0.1, throughout the manipulator workspace, for any rotation of its end-effector, i.e.,

min(*κ-1* (J)) ≥ 0.1

# Optimization Problem

The design of 3RRR is changed into an optimization problem where the objective is to make the manipulator compact. The manipulator should cover the workspace of a cylindrical regular workspace of diameter equal to 100 mm ,while having the inverse condition number of Kinematic Jacobian matrix greater than 0.1 for all points in workspace. The mathematical formulation of the problem is as follows:

Objective Function:*minimize* (*)*

Where (*)* (link lengths) are the variables of the optimization problem.

Constraints:

The function has *g* (*X*) inequality constraints. It has two inequality constraints i.e., the *invcond*(*J* ) and the summation of link lengths should be greater than or equal to the span of work space. Invcond(j) is the inverse conditioning number of Kinematic Jacobian matrix at different workspace points *X*.

The optimization problem is solved using Matlab’s **fmincon** function. The workspace points considered

For evaluation are the boundary points as shown below,

# The Matlab Files:

*“constraints.m”*: In this file we defined the inequality constraints. Here we do not have any equality constraint. Our two inequality constraints are the invcond(j) and the span of sum of L1 and L2.

For first constraint we begin by calculating condition number.

This part of code contains the function name, variables and the constant parameters like E and coordinates of A1, A2 and A3 etc.

function [g,h] = constraints(L)

%UNTITLED5 Summary of this function goes here

% Detailed explanation goes here

E = [0 -1;

1 0];

base = 174;

pb = (sqrt(3)/2)\*base;

r = 50;

platform = 15;

L1 = L(1);

L2 = L(2);

pp = (sqrt(3)/2)\*platform;

A = [-(base/2), -(pb/3);

base/2, -(pb/3);

0, (2\*pb)/3];

This part of code gives the definition of the boundary workspace points for which we will calculate condition number.

i = 1;

for alpha = 0: pi/4: (7\*pi)/4

Boundary(i,:) = [r\*cos(alpha), r\*sin(alpha)];

i = i+1;

end

Here now that we know the boundary coordinates for the centroid of the moving platform. We now need to define the coordinates of C corresponding to this boundary position over an angle width of pi/3. Hence we calculated 2 sets of coordinates of the three points. These sets define the spread over an angle of pi/3.

phi = pi/6;

for j = 1: 8

c11(j, :) = [Boundary(j,1)+pp\*cos(phi), Boundary(j,2)+pp\*sin(phi)];

c12(j, :) = [Boundary(j,1), Boundary(j,2)-pp];

c13(j, :) = [Boundary(j,1)-pp\*cos(phi), Boundary(j,2)+pp\*sin(phi)];

c21(j, :) = [Boundary(j,1)+pp\*cos(phi), Boundary(j,2)-pp\*sin(phi)];

c22(j, :) = [Boundary(j,1)-pp\*cos(phi), Boundary(j,2)-pp\*sin(phi)];

c23(j, :) = [Boundary(j,1), Boundary(j,2)+pp];

end

Now to find condition number we need the coordinates of all three points of each limb, namely A, B and C. Here is the code snippet to calculate B.

m = 1;

for k = 1: 8

A1C11 = [sqrt(((A(1,1)-c11(k,1))^2)+((A(1,2)-c11(k,2))^2))];

A1C21 = [sqrt(((A(1,1)-c21(k,1))^2)+((A(1,2)-c21(k,2))^2))];

A2C12 = [sqrt(((A(2,1)-c12(k,1))^2)+((A(2,2)-c12(k,2))^2))];

A2C22 = [sqrt(((A(2,1)-c22(k,1))^2)+((A(2,2)-c22(k,2))^2))];

A3C13 = [sqrt(((A(3,1)-c13(k,1))^2)+((A(3,2)-c13(k,2))^2))];

A3C23 = [sqrt(((A(3,1)-c23(k,1))^2)+((A(3,2)-c23(k,2))^2))];

B1y = ((A1C11)^2 + (L1)^2 - (L2)^2)/(2\*A1C11);

B1x = sqrt((L1)^2 - (B1y)^2);

B1(m, :) = [B1x, B1y];

B1(m+1, :) = [B1x, B1y];

B2y = ((A2C12)^2 + (L1)^2 - (L2)^2)/(2\*A2C12);

B2x = sqrt((L1)^2 - (B2y)^2);

B2(m, :) = [B2x, B2y];

B2(m+1, :) = [B2x, B2y];

B3y = ((A3C13)^2 + (L1)^2 - (L2)^2)/(2\*A3C13);

B3x = sqrt((L1)^2 - (B3y)^2);

B3(m, :) = [B3x, B3y];

B3(m+1, :) = [B3x, B3y];

m = m+2;

end

Following is the snippet to calculate Ui, the unit vector along AB:

for j = 1 : 16

A1B1\_vector = [A(1,1)-B1(j,1), A(1,2)-B1(j,2)];

A1B1 = sqrt((A(1,1)-B1(j,1))^2+(A(1,2)-B1(j,2))^2);

u1(j, :) = [A1B1\_vector/A1B1];

A2B2\_vector = [A(2,1)-B2(j,1), A(2,2)-B2(j,2)];

A2B2 = sqrt((A(2,1)-B2(j,1))^2+(A(2,2)-B2(j,2))^2);

u2(j, :) = [A2B2\_vector/A2B2];

A3B3\_vector = [A(3,1)-B3(j,1), A(3,2)-B3(j,2)];

A3B3 = sqrt((A(3,1)-B3(j,1))^2+(A(3,2)-B3(j,2))^2);

u3(j, :) = [A3B3\_vector/A3B3];

end

Following is the snippet to calculate Vi, the unit vector along BC:

p = 1;

for j = 1 : 8

B1C1(p) = [sqrt(((B1(p,1)-c11(j,1))^2)+((B1(p,2)-c11(j,2))^2))];

B1C1(p+1) = [sqrt(((B1(p+1,1)-c21(j,1))^2)+((B1(p+1,2)-c21(j,2))^2))];

B1C1\_vector(p, :) = [B1(p,1)-c11(j,1), B1(p,2)-c11(j,2)];

B1C1\_vector(p+1, :) = [B1(p+1,1)-c21(j,1), B1(p+1,2)-c21(j,2)];

B2C2(p) = [sqrt(((B2(p,1)-c12(j,1))^2)+((B2(p,2)-c12(j,2))^2))];

B2C2(p+1) = [sqrt(((B2(p+1,1)-c22(j,1))^2)+((B2(p+1,2)-c22(j,2))^2))];

B2C2\_vector(p, :) = [B2(p,1)-c11(j,1), B2(p,2)-c11(j,2)];

B2C2\_vector(p+1, :) = [B2(p+1,1)-c21(j,1), B2(p+1,2)-c21(j,2)];

B3C3(p) = [sqrt(((B3(p,1)-c13(j,1))^2)+((B3(p,2)-c13(j,2))^2))];

B3C3(p+1) = [sqrt(((B3(p+1,1)-c23(j,1))^2)+((B3(p+1,2)-c23(j,2))^2))];

B3C3\_vector(p, :) = [B3(p,1)-c11(j,1), B3(p,2)-c11(j,2)];

B3C3\_vector(p+1, :) = [B3(p+1,1)-c21(j,1), B3(p+1,2)-c21(j,2)];

p = p+2;

end

for j = 1 : 16

v1(j, :) = B1C1\_vector(j, :)/B1C1(j);

v2(j, :) = B2C2\_vector(j, :)/B2C2(j);

v3(j, :) = B3C3\_vector(j, :)/B3C3(j);

end

Following is the snippet to calculate Ki, the unit vector along CP:

q = 1;

for j = 1 : 8

C1P\_vector(q, :) = [c11(j,1)-Boundary(j,1), c11(j,2)-Boundary(j,2)];

C1P\_vector(q+1, :) = [c21(j,1)-Boundary(j,1), c21(j,2)-Boundary(j,2)];

C2P\_vector(q, :) = [c12(j,1)-Boundary(j,1), c12(j,2)-Boundary(j,2)];

C2P\_vector(q+1, :) = [c22(j,1)-Boundary(j,1), c22(j,2)-Boundary(j,2)];

C3P\_vector(q, :) = [c13(j,1)-Boundary(j,1), c13(j,2)-Boundary(j,2)];

C3P\_vector(q+1, :) = [c23(j,1)-Boundary(j,1), c23(j,2)-Boundary(j,2)];

C1P(q) = [sqrt(((c11(j,1)-Boundary(j,1))^2)+((c11(j,2)-Boundary(j,2))^2))];

C1P(q+1) = [sqrt(((c21(j,1)-Boundary(j,1))^2)+((c21(j,2)-Boundary(j,2))^2))];

C2P(q) = [sqrt(((c12(j,1)-Boundary(j,1))^2)+((c12(j,2)-Boundary(j,2))^2))];

C2P(q+1) = [sqrt(((c22(j,1)-Boundary(j,1))^2)+((c22(j,2)-Boundary(j,2))^2))];

C3P(q) = [sqrt(((c13(j,1)-Boundary(j,1))^2)+((c13(j,2)-Boundary(j,2))^2))];

C3P(q+1) = [sqrt(((c23(j,1)-Boundary(j,1))^2)+((c23(j,2)-Boundary(j,2))^2))];

q = q+2;

end

for j = 1: 16

k1(j, :) = C1P\_vector(j, :)/C1P(j);

k2(j, :) = C2P\_vector(j, :)/C2P(j);

k3(j, :) = C3P\_vector(j, :)/C3P(j);

end

Now we know condition number = inv(A)\*B. So we calculate condition number (represented by J), A (represented by J1) and B (represented by J2), Then from J we calculate inverse condition number, the required constraint.

for j = 1 : 16

J1 = [-pp\*(v1(j, :))\*E\*(k1(j, :))', v1(j, :);

-pp\*(v2(j, :))\*E\*(k2(j, :))', v2(j, :);

-pp\*(v3(j, :))\*E\*(k3(j, :))', v3(j, :)];

J2 = [L1\*v1(j, :)\*E\*(u1(j, :))', 0, 0;

0, L1\*v2(j, :)\*E\*(u2(j, :))', 0;

0, 0, L1\*v3(j, :)\*E\*(u3(j, :))'];

J = J1\J2;

c\_num = cond(J);

inv\_condition(j) = inv(c\_num);

end

Finally setting up the constraints:

g1 = 0.1-inv\_condition;

g2 = 150-L1+L2;

g = [g1, g2];

h = [];

end

*“objfun.m”*:

It is the objective function file:

function f = objfun( L )

%UNTITLED3 Summary of this function goes here

% Detailed explanation goes here

L1 = L(1);

L2 = L(2);

f = L1+L2;

end

*“opt\_3RRR”*:

This file calls the objective function and the constraint file. It uses fmincoin and optimset to get the required optimised results for our problem.

close all

clear all

clc

% Options including stopping criteria

options = optimset('Display','iter',...

'Tolfun',1e-6,...

'Tolx',1e-6,...

'MaxFunEval',200,...

'MaxIter',100);

% Starting point

L0 = [30,30]; %Given by the expert, physical problem

% minimum and maximum feasible values of L

lb = [40,60];

ub = [200,200];

% objfun minimization

L = fmincon('objfun',L0,[],[],[],[],lb,ub,'constraints',options);

L % Final Solution

# CAD Design:

## Individual Parts

|  |
| --- |
|  |

Figure  :End effector

|  |
| --- |
|  |

Figure 3: Links attached to end Effector

|  |
| --- |
|  |

Figure : PLatform

|  |
| --- |
|  |

Figure : Base Links

## Product

|  |
| --- |
|  |

Figure 6: RRR Robot

# Observation

In this project, the design problem of the 3RRR is converted into an optimization problem. This

Optimization problem is solved using Matlab’s **fmincon** function. The simulation should be done

formany different initial points so the solution should not stuck in the local minima. After running

Simulation for many iterations following link lengths were found

Using these link lengths, a parametric CAD model is made in CATIA. The model is simulating manually for different points.

# References

Raza Ur-Rehman, Stéphane Caro, Damien Chablat, Philippe Wenger. Multiobjective Design Optimization of 3-PRR Planar Parallel Manipulators. *20th CIRP Design conference*, Apr 2010, Nantes, France. pp.1-10, 2010.