**ELEC4010M Project 2 Report**

**Implementation robotic grasping**

DU Donghong



***Abstract*—This project not only implements the visualizing the possible robotic grasping but also ranks the quality of grasps in an interactive mode. In particular, the project is specified for the two and three point contacts grasping with friction.**

# I. INTRODUCTION

In this project, the aim is to implement a robotic grasping model in Matlab. It is composed of mainly four parts, the command lines for the user to input the information, the figure showing which edges or contacts are chosen, the 2D or 3D visualization of the contact points and the optional quality visualization.

# II. PREPARATION STEPS

## A. Polygon model preparation

The program firstly talks to the user, asking for positions for all vertices in a circle order. The program will display some special notation first and show the shape of polygon.

## B. GUI implementation

A graphical user interface (GUI) is easier for user to learn about and play around with our project.

By just choosing the edges and whether to show the quality graph or not, the key parts of the project can be shown directed through GUI.

A feature which is the processing bar is added to this project. The program has a huge amount of data to store, process and visualize, which could take some time. To improve the user experience, processing bar showing how far the program has gone is a good choice. At least the user can see something moving until the program is done.

III. MATH

The mathematical part of this project is mainly about force-closure and linear programming.

## A. Force-closure

In the case of two contacts points, the project2.m is complemented through linear programming. Fix one contact point as the origin first, we build the wrench matrix with contact point position and the friction coefficient. Then,

f = [1 1 1 1]; b = [-1 -1 -1 -1];

A is diagonal matrix whose non-zero element are all -1;

We have A \* x <=b and wrench \* x = zero vector, that is equivalent to positive solutions to wrench \* x = zero vector. In terms of linear algebra cases, it means the wrench can span the whole space.

In project2\_2pointFC.m, the program has another way to deal with 2 contacts cases. It uses theorem of planar antipodal grasp. If the line connecting the contact point lies inside both friction cones, this grasp is force-closure.

In the case of two contacts points, the project2.m is complemented through linear programming. Fix one contact point as the origin first, we build the wrench matrix with contact point position and the friction coefficient. Then,

f = [1 1 1 1 1 1]; b = [-1 -1 -1 -1 -1 -1];

A is diagonal matrix whose non-zero element are all -1;

We have A \* x <=b and wrench \* x = zero vector, the mathematics behind is similar to the first case.

## B. Quality measurement

* The quality measurement here is to find the possible force-closure case first, and note down the edges for frictional cones. For 2 contacts cases, we will have 4 forces if frictional. For 3 contacts cases, we will have 6 forces if frictional.
* Next is to move those forces back to (0, 0), and normalize them to unit 1. Define the quality of force-closure as the radius of largest circle centered at the origin of the force space that fit inside CF. The shape of CF is a convex polyhedron, which is the combination of all the normalized forces whose weight is between 0 and 1. To be noticed, all the actuator limit is set to (1+u^2)^(-0.5), u is frictional coefficient, to make the every edge forces for friction cones to be one.
* This project creates an alternative method called angle method to measure the quality for 2 contact points. In the project2\_2pointFC.m, the force-closure is checked by planar antipodal theorem. We measure the angle of the connecting line and the normal line in both local contact point and target contact point. Let the two angle be a1 and a2 in radian unit. Define the quality = exp(-a1)+exp(-a2). It means that if the connection line is more close to normal line, the quality of the grasp is higher.

# IV. CORE PROCEDURES

The core procedures of this project focus on the collection, processing and visualization of the data. To be emphasized, the information can be huge if we have many vertices and each is divided into multiple segments.

## Data input

What user inputs can be out of control if there is no constrains. We should set up some conditions according to what we are asking to regulate the input so that the program can cooperate with users to some extents. For example, the number of total vertices should be a positive integer larger than three. Otherwise, it could not be a polygon. Another constraint are that the range of the frictional coefficient should be in (0, 1] according the requirement of the project guideline, ‘friction contacts’. The program also ensures the input of segment should be positive integer, the x, y coordinates of vertices should be numbers.

Another feature here is that the program will record the unit of those number to remind the user for future convenience.

To combine 2 points cases and 3 points case, one prompt is to ask choosing which. 2 will go to 2 point cases and 3 will go to 3 point cases.

## B. data processing

The main part of data processing is to store the position of the vertices, judging force-closure grasp and calculation of the quality radius. The data storage is done publically to save the storage and improve the speed. If we copy those position every, it will cost a lot of time.

To judge whether the grasp is force-closure, the mathematic idea is in the Math section. Coding is similar, the same A, f, wrench. Use ‘linprog’ in Matlab, if the exitflag of result is 1, then it has solutions and it is force-closure. In the project\_2\_2pointFC.m, as previous part said, the program compares the angel of the lines involved to use the planar antipodal theorem. During the testing, they show the same result.

The calculation of quality radius to let the endpoint to be the vertices of an imagined polygon. The maximum circle inside the shape will be the minimum distance to each edge. Let e be the edge of two forces, the z-component of cross product of two forces is twice the area of this triangle. The distance from origin times the length of e is the same result. Later, just keep the minimum distance which will be the quality radius.

## C. Data visualization

In 2 contact point force-closure cases, choose 2 edges each times. The x, y axis is the distance between the solution contact point and the start point in the solution edge for each solution edges. The start and end point of the edge is shown in the legend.

For the rank of quality grasp of 2 contact points, the best grasp will output in the command line in the beginning. Not all the rank is shown because in practical, the polygon could have complex shape and it will be messy if all ranks are displayed. Besides, users always want the best one.

To deal with the practical cases deeper, the project2\_2\_2pointFC.m is to ask the user to choose an arbitrary point, then the program will give all possible solution. If the quality button is on, it also shows the quality with method talked in Math part. The larger the z value is, the higher the quality. It explicitly shows the rank of the quality for the given local contact point and the solution points calculated. To be noticed, the x is always the edge for contact points, but the y axis is for the solution points, which can be different edges while the record of distances are also their corresponding result.

If the quality button is turned on, a 3D graph will be shown. The x, y is same meaning, while the z stands for the value of the quality data. Actually, for 2 contact points, as only 2 edges can be chosen one time, the solution on the same edges have same friction coefficients, the quality radius will be the same.

In 3 contact point force-closure case, choose 3 edges each times. The x, y, z axis is the distance between the solution contact point and the start point in the solution edge for each solution edge. The start and end point of the edge is shown in the legend.

For the rank of quality grasp of 3 contact points, the best grasp will output in the command line in the beginning. If the quality button is turned on, a 3D graph will be shown. The x, y, z is same meaning, while value of the quality radius is shown as a ball at that position. Actually, for 3 contact points, as only 3 edges can be chosen one time, the solution on the same edges have same friction coefficients, the quality radius will be the same.

# V. CONCLUSION

This project is divided into three major parts. First part is to deal with the input data, their format and range. Second part is to linear programming of force-closure and calculate the quality radius. Third part is to rank the grasp and visualize the quality. It is the first time to deal with big data in programming. I use the hint from professor that use the distance as the axis. It really makes the visualization neat and clear.

# APPENDIX

The project has two .m file.

project2.m is for the 8 requirements in the project guideline and project2\_2pointFC is an extra file. I use the extra one to show the planar antipodal theorem and another way to visualize the force-closure. They are independent.

# REFERENCES

1. <https://www.mathworks.com/help/optim/ug/linprog.html>

(the build-in linear programming function in Matlab)

1. <https://www.mathworks.com/help/matlab/ref/inpolygon.html>

(the build-in function to judge whether a point is in a polygon)

1. [https://cn.mathworks.com/help/matlab/ref/uicontrol.html?searchHighli ght=uicontrol&s\_tid=doc\_srchtitle](https://cn.mathworks.com/help/matlab/ref/uicontrol.html?searchHighlight=uicontrol&s_tid=doc_srchtitle)

(library for making the GUI)

Codes:

project2.m

function sol = project2

clear all;

close all;

clc;

format long;

%% MATLAB version: R2016a

%{

Author: DU DONGHONG

20328343

dduaa@connect.ust.hk

2018 SPRING ELEC4010M PROJECT2

Baisc requirement

%}

%% basic requests for vertieces

[vertices,segment, v\_num, my\_unit ,u] = basic;

%{

%% visualize the polygon

for i =1:v\_num-1

figure(1);

line([vertices(1,i), vertices(1,i+1)], [vertices(2,i), vertices(2,i+1)]);

hold on;

end

line([vertices(1,v\_num), vertices(1,1)], [vertices(2,v\_num), vertices(2,1)]);

hold on;

axis equal;

%}

%% build contact points

contact = zeros(v\_num, 2, segment-1);

%% for the shape

shapex=[]; shapey = [];

for i = 1: v\_num

xs = vertices(1,i); xe = vertices(1,1+mod(i,v\_num));

ys = vertices(2,i); ye = vertices(2,1+mod(i,v\_num));

shapex=[shapex xs]; shapey=[shapey ys];

x\_interval = (xe-xs) / segment;

y\_interval = (ye-ys) / segment;

for j =1:segment -1

contact( i , 1 , j) = xs + x\_interval \* j;

contact( i , 2 , j) = ys + y\_interval \* j;

shapex=[shapex contact( i , 1 , j)];

shapey=[shapey contact( i , 2 , j)];

end

shapex=[shapex xe];

shapey=[shapey ye];

end

%% check force-closure

% 2-point or 3-point

choice = str2double( input('Input 2 or 3 for 2-point FC check or 3-point FC check','s'));

while isnan(choice) || fix(choice)~= choice || (choice~= 2 && choice~= 3)

choice = str2double( input('Invalid. Input 2 or 3 for 2-point FC check or 3-point FC check','s'));

end

disp('please just drag the legend away if it blocks something');

disp('We do not consider the vertice point for its unstability');

disp('Only consider the contact points by dividing edges with the number of segment');

if choice ==2

choice2;

else

choice3;

end

%% for 2 points FC

% find the 2 contacts first then use the Planar antipodal theorem

function choice2

%% basic of GUI

FigH = figure('position',[360 500 400 400],'name','2-point force closure');

axes( 'units','pixels', 'position',[100 50 200 200]);

LineH = plot(shapex,shapey,'black');

title(['unit is ' my\_unit]);

axis equal;

legend('shape');

if(~exist('qulityButton','var'))

quality2Button = uicontrol('Style','radiobutton','String','show quality measure ','pos',[0 0 200 25],'parent',FigH,'Callback', @callbackfnQ);

end

set(quality2Button,'Value',0);

TextH\_e = uicontrol('style','text','position',[170 310 100 15]);

TextH\_e.String =sprintf('edge# 2 in blue');

TextH\_c = uicontrol('style','text','position',[170 350 100 15]);

TextH\_c.String =sprintf('edge#1 in red');

Slider2E1 = uicontrol('style','slider','position',[100 330 200 20],'min', 1, 'max',v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfn2E1);

Slider2E2 = uicontrol('style','slider','position',[100 290 200 20], 'min', 1, 'max', v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfn2E2);

[bestRadius, location] = bestFC(v\_num);

fprintf('The best grasp can have a quality measure is %s in radius method\n',num2str(bestRadius));

if bestRadius == 0

fprintf('No valid grasp\n');

else

fprintf('At edge #%s',num2str(location(1)));

fprintf(', segment #%s \n',num2str(location(2)));

fprintf('At edge #%s',num2str(location(3)));

fprintf(', segment #%s \n',num2str(location(4)));

end

Edge1=1;

Edge2=1;

movegui(FigH, 'center')

replot2(Edge1,Edge2);

%inner function for choice2

%real-time response

function [bestQuality, location] = bestFC(v\_num)

bestQuality = 0;

location = zeros(4,1);

for edge1 = 1:v\_num

for edge2 = 1:v\_num

if ~(edge1 ==edge2)

for count1 = 1: segment -1

for count2 = 1: segment -1

[c1f1,c1f2,c1] = findVector(edge1,count1);

[c2f1,c2f2,c2] = findVector(edge2,count2);

% use the first contact point as the origin

wrench = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2);

% check force closure as Linear Feasibility Problem

res = checkFC(wrench);

if(res ==1)

temp = quality2Measure(wrench);

if bestQuality < temp

bestQuality = temp;

location(1) = edge1;

location(2) = count1;

location(3) = edge2;

location(4) = count2;

end

end

end

end

end

end

end

function w = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2)

w1 = [0;c1f1(1);c1f1(2)];

w2 = [0;c1f2(1);c1f2(2)];

d2 = c2-c1;

m2 = cross( [d2(1) d2(2) 0], [c2f1(1) c2f1(2) 0] );

m22 = cross( [d2(1) d2(2) 0] , [c2f2(1) c2f2(2) 0] );

w3 = [ m2(3);c2f1(1);c2f1(2)];

w4 = [m22(3);c2f2(1);c2f2(2)];

w = [w1 w2 w3 w4];

end

function res = checkFC(w)

if rank(w)==3

fu=[1 1 1 1];

b=[-1 -1 -1 -1];

A= [[-1 0 0 0];[0 -1 0 0];[0 0 -1 0];[0 0 0 -1]];

options = optimoptions('linprog','Display','none');

[x,fval,exitflag,output]= linprog(fu,A,b,w,zeros(3,1),[],[],[],options);

if exitflag==1

res = 1;

else

res = 0;

end

else

res = 0;

end

end

function quality = quality2Measure(wrench)

% use the Qual(CF) = sigma(a\_i \* F\_i), a\_i belongs to [0

% ,F\_i\_max]. FInd the largest radius of the ball centered

% at the origin of the wrench space

w = wrench;

for iter = 1:4

% normalize every forces first;

len = sqrt( (w(2,iter) )^2 + (w(3,iter) )^2);

w(2,iter) = w(2,iter) / len;

w(3,iter) = w(3,iter) / len;

w(1,iter) = atan2( w(3,iter) , w(2,iter));

end

v1 = zeros(2,1); v2 = 1; v3 = 1; v4 = zeros(2,1);

used = 0;

for iter = 1:4

if w(1,iter) == max (w(1,:))

v1 = w(2:3,iter);

elseif w(1,iter) == min (w(1,:))

v4 = w(2:3,iter);

else

if used == 0

v2 = iter;

used =1;

else

v3 = iter;

end

end

end

if w(1,v2) == max( w(1,v2) , w(1,v3))

second\_max = v2;

third\_max = v3;

else

second\_max = v3;

third\_max = v2;

end

v2 = w(2:3,second\_max);

v3 = w(2:3,third\_max);

v= [v1 v1+v2 v2 v2+v3 v3 v3+v4 v4 v4+v1];

radius = zeros(1,8);

for iter = 1:8

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,8));v(2, 1+mod(iter,8));0]);

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,8)))^2+(v(2,iter)-v(2, 1+mod(iter,8)))^2 );

radius(iter) = abs(area(3))/len;

end

quality = min(radius);

end

end

function callbackfnQ(source, eventdata)

if(get(quality2Button,'Value')==1)

replot2(Edge1,Edge2);

else

close(figure(3));

end

end

function callbackfn2E1(source, eventdata)

Edge1 =source.Value;

Edge1 = round(Edge1);

set(source,'Value', Edge1);

replot2(Edge1,Edge2);

end

function callbackfn2E2(source, eventdata)

Edge2 =source.Value;

Edge2 = round(Edge2);

set(source,'Value', Edge2);

replot2(Edge1,Edge2);

end

function replot2(edge1,edge2)

cla(LineH);

LineH = plot(shapex,shapey,'black');

hold on;

% for edge 1

ex1s = vertices(1,edge1); ex1e = vertices(1,1+mod(edge1,v\_num));

ey1s = vertices(2,edge1); ey1e = vertices(2,1+mod(edge1,v\_num));

LineH = plot(ex1s,ey1s,'or'); LineH = plot(ex1e,ey1e,'xr');

e1x=[]; e1y=[];

xe1\_interval = (ex1e-ex1s) / segment; ye1\_interval = (ey1e-ey1s) / segment;

for ii=0:segment

e1x=[e1x ex1s+xe1\_interval\*ii]; e1y = [e1y ey1s+ye1\_interval\*ii];

end

LineH = plot(e1x,e1y,'r');

% for edge 2

ex2s = vertices(1,edge2); ex2e = vertices(1,1+mod(edge2,v\_num));

ey2s = vertices(2,edge2); ey2e = vertices(2,1+mod(edge2,v\_num));

LineH = plot(ex2s,ey2s,'ob'); LineH = plot(ex2e,ey2e,'xb');

e2x=[]; e2y=[];

xe2\_interval = (ex2e-ex2s) / segment; ye2\_interval = (ey2e-ey2s) / segment;

for ii=0:segment

e2x=[e2x ex2s+xe2\_interval\*ii]; e2y= [e2y ey2s+ye2\_interval\*ii];

end

LineH = plot(e2x,e2y,'b');

% remark building

title(['unit is ' my\_unit]);

axis equal;

xlabel('X') % x-axis label

ylabel('Y') % y-axis label

legend('shape','start of edge#1','end of edge#1','edge1','start of edge#2','end of edge#2','edge2');

hold off;

relation2D(edge1, edge2);

end

function relation2D(edge1, edge2)

max\_q = 0;

if(get(quality2Button,'Value')==1)

fig3= figure(3);

cla(fig3);

set(fig3,'name','FC quality measure with 2 contacts');

end

f= figure(2);

%subplot(211);

%cla(f);

%subplot(212);

cla(f);

set(f,'name','FC in two contacts cases');

%subplot(211);

%subplot(212);

if ~(edge1 ==edge2)

hwait = waitbar(0,'please wait>>>>>>');

step = (segment-1)/100;

for count1 = 1: segment -1

% process the waitbar

if segment-1-count1<= 5/(segment-1)

waitbar(count1/(segment-1),hwait,'It is about to finish');

pause(0.05);

else

PerStr=fix(count1/step);

str=['running',num2str(PerStr),'%'];

waitbar(count1/(segment-1),hwait,str);

pause(0.05);

end

for count2 = 1: segment -1

[c1f1,c1f2,c1] = findVector(edge1,count1);

[c2f1,c2f2,c2] = findVector(edge2,count2);

% use the first contact point as the origin

wrench = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2);

% check force closure as Linear Feasibility Problem

res = checkFC(wrench);

if(res ==1)

distance1 = sqrt( sum( (c1-vertices(:,edge1)).\*(c1-vertices(:,edge1)) ));

distance2 = sqrt( sum( (c2-vertices(:,edge2)).\*(c2-vertices(:,edge2)) ));

f=figure(2);

hold on;

f=plot(distance1,distance2,'xr');

view(2);

if(get(quality2Button,'Value')==1)

fig3= figure(3);

hold on;

quality = quality2Measure(wrench);

if quality > max\_q

max\_q = quality;

end

stem3(distance1,distance2,quality);

view(3);

end

end

end

end

close(hwait);

end

f = figure(2);

xlabel('edge chosen first') % x-axis label

ylabel('edge chosen second') % y-axis label

axis equal;

if(get(quality2Button,'Value')==1)

fig3 = figure(3);

xlabel('edge chosen first') % x-axis label

ylabel('edge chosen second') % y-axis label

zlabel('quality radius')% z-axis label

fprintf('The maximum radius is %d .\n',max\_q);

end

function w = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2)

w1 = [0;c1f1(1);c1f1(2)];

w2 = [0;c1f2(1);c1f2(2)];

d2 = c2-c1;

m2 = cross( [d2(1) d2(2) 0], [c2f1(1) c2f1(2) 0] );

m22 = cross( [d2(1) d2(2) 0] , [c2f2(1) c2f2(2) 0] );

w3 = [ m2(3);c2f1(1);c2f1(2)];

w4 = [m22(3);c2f2(1);c2f2(2)];

w = [w1 w2 w3 w4];

end

function res = checkFC(w)

if rank(w)==3

fu=[1 1 1 1];

b=[-1 -1 -1 -1];

A= [[-1 0 0 0];[0 -1 0 0];[0 0 -1 0];[0 0 0 -1]];

options = optimoptions('linprog','Display','none');

[x,fval,exitflag,output]= linprog(fu,A,b,w,zeros(3,1),[],[],[],options);

if exitflag==1

res = 1;

else

res = 0;

end

else

res = 0;

end

end

function quality = quality2Measure(wrench)

% use the Qual(CF) = sigma(a\_i \* F\_i), a\_i belongs to [0

% ,F\_i\_max]. FInd the largest radius of the ball centered

% at the origin of the wrench space

w = wrench;

for iter = 1:4

% normalize every forces first;

len = sqrt( (w(2,iter) )^2 + (w(3,iter) )^2);

w(2,iter) = w(2,iter) / len;

w(3,iter) = w(3,iter) / len;

w(1,iter) = atan2( w(3,iter) , w(2,iter));

end

v1 = zeros(2,1); v2 = 1; v3 = 1; v4 = zeros(2,1);

used = 0;

for iter = 1:4

if w(1,iter) == max (w(1,:))

v1 = w(2:3,iter);

elseif w(1,iter) == min (w(1,:))

v4 = w(2:3,iter);

else

if used == 0

v2 = iter;

used =1;

else

v3 = iter;

end

end

end

if w(1,v2) == max( w(1,v2) , w(1,v3))

second\_max = v2;

third\_max = v3;

else

second\_max = v3;

third\_max = v2;

end

v2 = w(2:3,second\_max);

v3 = w(2:3,third\_max);

v= [v1 v1+v2 v2 v2+v3 v3 v3+v4 v4 v4+v1];

radius = zeros(1,8);

for iter = 1:8

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,8));v(2, 1+mod(iter,8));0]);

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,8)))^2+(v(2,iter)-v(2, 1+mod(iter,8)))^2 );

radius(iter) = abs(area(3))/len;

end

quality = min(radius);

end

end

end

%% for 3 points FC

function choice3

% three contact points

FigH = figure('position',[360 500 600 600],'name','3-point force closure');

axes( 'units','pixels', 'position',[100 50 400 400]);

LineH = plot(shapex,shapey,'black');

title(['unit is ' my\_unit]);

axis equal;

legend('shape');

if(~exist('qulityButton','var'))

quality3Button = uicontrol('Style','radiobutton','String','show quality measure ','pos',[0 0 200 25],'parent',FigH,'Callback', @callbackfnQ);

end

set(quality3Button,'Value',0);

TextH\_e1 = uicontrol('style','text','position',[250 510 100 15]);

TextH\_e1.String =sprintf('edge#3 in green');

TextH\_e2 = uicontrol('style','text','position',[250 550 100 15]);

TextH\_e2.String =sprintf('edge#2 in blue');

TextH\_e3 = uicontrol('style','text','position',[250 590 100 15]);

TextH\_e3.String =sprintf('edge#1 in red');

Slider2E1 = uicontrol('style','slider','position',[150 570 200 20],'min', 1, 'max',v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfn2E1);

Slider2E2 = uicontrol('style','slider','position',[150 530 200 20], 'min', 1, 'max', v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfn2E2);

Slider2E3 = uicontrol('style','slider','position',[150 490 200 20], 'min', 1, 'max', v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfn2E3);

[bestRadius, location] = bestFC(v\_num);

fprintf('The best grasp can have a quality measure is %s in radius method\n',num2str(bestRadius));

if bestRadius == 0

fprintf('No valid grasp\n');

else

fprintf('At edge #%s',num2str(location(1)));

fprintf(', segment #%s \n',num2str(location(2)));

fprintf('At edge #%s',num2str(location(3)));

fprintf(', segment #%s \n',num2str(location(4)));

fprintf('At edge #%s',num2str(location(5)));

fprintf(', segment #%s \n',num2str(location(6)));

end

E1=1; E2=1; E3=1;

movegui(FigH, 'center')

replot3(E1,E2,E3);

%inner function for choice2

%real-time response

function callbackfnQ(source, eventdata)

if(get(quality3Button,'Value')==1)

replot3(E1,E2,E3);

else

close(figure(3));

end

end

function [bestQuality, location] = bestFC(v\_num)

bestQuality = 0;

location = zeros(6,1);

for edge1 = 1:v\_num

for edge2 = 1:v\_num

for edge3 = 1:v\_num

if (~(edge1 ==edge2)&&(edge1==edge3))

for count1 = 1: segment -1

for count2 = 1: segment -1

for count3 = 1:segment-1

[c1f1,c1f2,c1] = findVector(edge1,count1);

[c2f1,c2f2,c2] = findVector(edge2,count2);

[c3f1,c3f2,c3] = findVector(edge3,count3);

% use the first contact point as the origin

wrench = findWrenchI(c1f1,c1f2,c1,c2f1,c2f2,c2,c3f1,c3f2,c3);

% check force closure as Linear Feasibility Problem

res = checkFCI(wrench);

if(res ==1)

temp = quality3MeasureI(wrench);

if bestQuality < temp

bestQuality = temp;

location(1) = edge1;

location(2) = count1;

location(3) = edge2;

location(4) = count2;

location(5) = edge3;

location(6) = count3;

end

end

end

end

end

end

end

end

end

function quality = quality3MeasureI(forces)

for iter = 1:6

% normalize every forces first;

len = sqrt( (forces(2,iter) )^2 + (forces(3,iter) )^2);

forces(2,iter) = forces(2,iter) / len;

forces(3,iter) = forces(3,iter) / len;

forces(1,iter) = atan2( forces(3,iter) , forces(2,iter));

end

%sort the force by the atan2 function

for iter = 1:6

for iter1=iter+1:6

if forces(1,iter1) < forces(1,iter)

temp = forces(:,iter);

forces(:,iter) = forces(:,iter1);

forces(:,iter1) = temp;

end

end

end

radius = inf;

mark = ones(1,6);

for iter = 1:6

if mark(iter)==0

continue;

end

for iter1 = iter+1:6

if abs(forces(1,iter) -forces(1,iter1))<1e-5

mark(iter1) = 0;

forces(2,iter) = forces(2,iter) + forces(2,iter1);

forces(3,iter) = forces(3,iter) + forces(3,iter1);

end

end

end

vv = [];

for iter = 1:6

if mark(iter) ==1

vv=[vv forces(2:3,iter)];

end

end

[m,n] = size(vv);

v= [];

for iter = 1:n

if iter~=1

v =[v vv(1:2,iter-1)+vv(1:2,iter)];

end

v =[v vv(1:2,iter)];

end

v =[v vv(1:2,1)+vv(1:2,n)];

[m,n] = size(v);

for iter = 1:n

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,n));v(2, 1+mod(iter,n));0]);

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,n)))^2+(v(2,iter)-v(2, 1+mod(iter,n)))^2 );

temp = abs(area(3))/len;

if temp < radius &&radius~=0

radius = temp;

end

end

quality = radius;

%{

for debug

if quality ==0

for iter = 1:12

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,12)))^2+(v(2,iter)-v(2, 1+mod(iter,12)))^2 );

if len < 1e-5 || (abs(atan(v(1,iter),v(2,iter)) - atan(v(1, 1+mod(iter,12)),v(2, 1+mod(iter,12))))< 1e-5)

continue;

end

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,12));v(2, 1+mod(iter,12));0])

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,12)))^2+(v(2,iter)-v(2, 1+mod(iter,12)))^2 )

temp = abs(area(3))/len;

if temp < radius &&radius~=0

radius = temp;

end

end

end

%}

end

function w = findWrenchI(c1f1,c1f2,c1,c2f1,c2f2,c2,c3f1,c3f2,c3)

w1 = [0;c1f1(1);c1f1(2)];

w2 = [0;c1f2(1);c1f2(2)];

d2 = c2-c1;

m2 = cross( [d2(1) d2(2) 0], [c2f1(1) c2f1(2) 0] );

m22 = cross( [d2(1) d2(2) 0] , [c2f2(1) c2f2(2) 0] );

w3 = [ m2(3);c2f1(1);c2f1(2)];

w4 = [m22(3);c2f2(1);c2f2(2)];

d2 = c3 - c1;

m3 = cross( [d2(1) d2(2) 0] , [c3f1(1) c3f1(2) 0] );

m33 = cross( [d2(1) d2(2) 0], [c3f2(1) c3f2(2) 0] );

w5 = [m3(3);c3f1(1);c3f1(2)];

w6 = [m33(3);c3f2(1);c3f2(2)];

w = [w1 w2 w3 w4 w5 w6];

end

function res = checkFCI(w)

if rank(w)==3

f=[1 1 1 1 1 1];

b=[-1 -1 -1 -1 -1 -1];

A= [[-1 0 0 0 0 0];[0 -1 0 0 0 0];[0 0 -1 0 0 0];[0 0 0 -1 0 0];[0 0 0 0 -1 0];[0 0 0 0 0 -1]];

options = optimoptions('linprog','Display','none');

[x,fval,exitflag,output]= linprog(f,A,b,w,zeros(3,1),[],[],[],options);

if exitflag==1

res = 1;

else

res = 0;

end

else

res = 0;

end

end

end

function callbackfn2E1(source, eventdata)

E1 =source.Value;

E1 = round(E1);

set(source,'Value', E1);

replot3(E1,E2,E3);

end

function callbackfn2E2(source, eventdata)

E2 =source.Value;

E2 = round(E2);

set(source,'Value', E2);

replot3(E1,E2,E3);

end

function callbackfn2E3(source, eventdata)

E3 =source.Value;

E3 = round(E3);

set(source,'Value', E3);

replot3(E1,E2,E3);

end

function replot3(edge1,edge2,edge3)

cla(LineH);

LineH = plot(shapex,shapey,'black');

hold on;

% for edge 1

ex1s = vertices(1,edge1); ex1e = vertices(1,1+mod(edge1,v\_num));

ey1s = vertices(2,edge1); ey1e = vertices(2,1+mod(edge1,v\_num));

LineH = plot(ex1s,ey1s,'or'); LineH = plot(ex1e,ey1e,'xr');

e1x=[]; e1y=[];

xe1\_interval = (ex1e-ex1s) / segment; ye1\_interval = (ey1e-ey1s) / segment;

for ii=0:segment

e1x=[e1x ex1s+xe1\_interval\*ii]; e1y = [e1y ey1s+ye1\_interval\*ii];

end

LineH = plot(e1x,e1y,'r');

% for edge 2

ex2s = vertices(1,edge2); ex2e = vertices(1,1+mod(edge2,v\_num));

ey2s = vertices(2,edge2); ey2e = vertices(2,1+mod(edge2,v\_num));

LineH = plot(ex2s,ey2s,'ob'); LineH = plot(ex2e,ey2e,'xb');

e2x=[]; e2y=[];

xe2\_interval = (ex2e-ex2s) / segment; ye2\_interval = (ey2e-ey2s) / segment;

for ii=0:segment

e2x=[e2x ex2s+xe2\_interval\*ii]; e2y= [e2y ey2s+ye2\_interval\*ii];

end

LineH = plot(e2x,e2y,'b');

% for edge3

ex3s = vertices(1,edge3); ex3e = vertices(1,1+mod(edge3,v\_num));

ey3s = vertices(2,edge3); ey3e = vertices(2,1+mod(edge3,v\_num));

LineH = plot(ex3s,ey3s,'og'); LineH = plot(ex3e,ey3e,'xg');

e3x=[]; e3y=[];

xe3\_interval = (ex3e-ex3s) / segment; ye3\_interval = (ey3e-ey3s) / segment;

for ii=0:segment

e3x=[e3x ex3s+xe3\_interval\*ii]; e3y= [e3y ey3s+ye3\_interval\*ii];

end

LineH = plot(e3x,e3y,'g');

% remark building

title(['unit is ' my\_unit]);

axis equal;

xlabel('X') % x-axis label

ylabel('Y') % y-axis label

legend('shape','start of edge#1','end of edge#1','edge1','start of edge#2','end of edge#2','edge2','start of edge#3','end of edge#3','edge3');

hold off;

relation3D(edge1, edge2,edge3);

end

function relation3D(edge1, edge2,edge3)

% for vision

max\_q = 0;

if(get(quality3Button,'Value')==1)

fig3= figure(3);

cla(fig3);

set(fig3,'name','FC quality measure with 3 contacts');

end

f= figure(2);

cla(f);

hold on;

set(f,'name','FC 3 contacts');

hwait = waitbar(0,'please wait>>>>>>');

step = (segment-1)/100;

if ~(edge1 ==edge2 && edge2==edge3)

for count1 = 1: segment -1

% process the waitbar

if segment-1-count1<= 5/(segment-1)

waitbar(count1/(segment-1),hwait,'It is about to finish');

pause(0.05);

else

PerStr=fix(count1/step);

str=['running',num2str(PerStr),'%'];

waitbar(count1/(segment-1),hwait,str);

pause(0.05);

end

for count2 = 1: segment -1

for count3 = 1: segment -1

[c1f1,c1f2,c1] = findVector(edge1,count1);

[c2f1,c2f2,c2] = findVector(edge2,count2);

[c3f1,c3f2,c3] = findVector(edge3,count3);

% use the first contact point as the origin

wrench = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2,c3f1,c3f2,c3);

% check force closure as Linear Feasibility Problem

res = checkFC(wrench);

if(res ==1)

distance1 = sqrt( sum( (c1-vertices(:,edge1)).\*(c1-vertices(:,edge1)) ));

distance2 = sqrt( sum( (c2-vertices(:,edge2)).\*(c2-vertices(:,edge2)) ));

distance3 = sqrt( sum( (c3-vertices(:,edge3)).\*(c3-vertices(:,edge3)) ));

f = figure(2);

hold on;

plot3(distance1,distance2,distance3,'xr');

view(3);

if(get(quality3Button,'Value')==1)

fig3= figure(3);

hold on;

quality = quality3Measure(wrench);

if quality ==0

return;

end

if quality > max\_q

max\_q = quality;

end

%x=0;y=0;z=0;

[x,y,z] = sphere;

fig3 = surf(x\*quality+distance1, y\*quality+distance2, z\*quality+distance3);

axis equal;

view(3);

end

end

end

end

end

end

close(hwait);

f=figure(2);

view(3);

xlabel('edge chosen first') % x-axis label

ylabel('edge chosen second') % y-axis label

zlabel('edge chosen third');

%axis equal;

if(get(quality3Button,'Value')==1)

fig3 = figure(3);

view(3);

xlabel('edge chosen first') % x-axis label

ylabel('edge chosen second') % y-axis label

zlabel('edge chosen third')% z-axis label

fprintf('The maximum radius is %d .\n',max\_q);

end

end

function quality = quality3Measure(forces)

for iter = 1:6

% normalize every forces first;

len = sqrt( (forces(2,iter) )^2 + (forces(3,iter) )^2);

forces(2,iter) = forces(2,iter) / len;

forces(3,iter) = forces(3,iter) / len;

forces(1,iter) = atan2( forces(3,iter) , forces(2,iter));

end

%sort the force by the atan2 function

for iter = 1:6

for iter1=iter+1:6

if forces(1,iter1) < forces(1,iter)

temp = forces(:,iter);

forces(:,iter) = forces(:,iter1);

forces(:,iter1) = temp;

end

end

end

radius = inf;

mark = ones(1,6);

for iter = 1:6

if mark(iter)==0

continue;

end

for iter1 = iter+1:6

if abs(forces(1,iter) -forces(1,iter1))<1e-5

mark(iter1) = 0;

forces(2,iter) = forces(2,iter) + forces(2,iter1);

forces(3,iter) = forces(3,iter) + forces(3,iter1);

end

end

end

vv = [];

for iter = 1:6

if mark(iter) ==1

vv=[vv forces(2:3,iter)];

end

end

[m,n] = size(vv);

v= [];

for iter = 1:n

if iter~=1

v =[v vv(1:2,iter-1)+vv(1:2,iter)];

end

v =[v vv(1:2,iter)];

end

v =[v vv(1:2,1)+vv(1:2,n)];

[m,n] = size(v);

for iter = 1:n

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,n));v(2, 1+mod(iter,n));0]);

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,n)))^2+(v(2,iter)-v(2, 1+mod(iter,n)))^2 );

temp = abs(area(3))/len;

if temp < radius &&radius~=0

radius = temp;

end

end

quality = radius;

%{

for debug

if quality ==0

for iter = 1:12

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,12)))^2+(v(2,iter)-v(2, 1+mod(iter,12)))^2 );

if len < 1e-5 || (abs(atan(v(1,iter),v(2,iter)) - atan(v(1, 1+mod(iter,12)),v(2, 1+mod(iter,12))))< 1e-5)

continue;

end

area = cross([v(1,iter);v(2,iter);0],[v(1, 1+mod(iter,12));v(2, 1+mod(iter,12));0])

len = sqrt( (v(1,iter)-v(1, 1+mod(iter,12)))^2+(v(2,iter)-v(2, 1+mod(iter,12)))^2 )

temp = abs(area(3))/len;

if temp < radius &&radius~=0

radius = temp;

end

end

end

%}

end

function w = findWrench(c1f1,c1f2,c1,c2f1,c2f2,c2,c3f1,c3f2,c3)

w1 = [0;c1f1(1);c1f1(2)];

w2 = [0;c1f2(1);c1f2(2)];

d2 = c2-c1;

m2 = cross( [d2(1) d2(2) 0], [c2f1(1) c2f1(2) 0] );

m22 = cross( [d2(1) d2(2) 0] , [c2f2(1) c2f2(2) 0] );

w3 = [ m2(3);c2f1(1);c2f1(2)];

w4 = [m22(3);c2f2(1);c2f2(2)];

d2 = c3 - c1;

m3 = cross( [d2(1) d2(2) 0] , [c3f1(1) c3f1(2) 0] );

m33 = cross( [d2(1) d2(2) 0], [c3f2(1) c3f2(2) 0] );

w5 = [m3(3);c3f1(1);c3f1(2)];

w6 = [m33(3);c3f2(1);c3f2(2)];

w = [w1 w2 w3 w4 w5 w6];

end

function res = checkFC(w)

if rank(w)==3

f=[1 1 1 1 1 1];

b=[-1 -1 -1 -1 -1 -1];

A= [[-1 0 0 0 0 0];[0 -1 0 0 0 0];[0 0 -1 0 0 0];[0 0 0 -1 0 0];[0 0 0 0 -1 0];[0 0 0 0 0 -1]];

options = optimoptions('linprog','Display','none');

[x,fval,exitflag,output]= linprog(f,A,b,w,zeros(3,1),[],[],[],options);

if exitflag==1

res = 1;

else

res = 0;

end

else

res = 0;

end

end

end

%% angle function for Planar antipodal theorem

%{

function Angle = angle( xs , ys , xe , ye, x\_edge , y\_edge )

Angle1 = atan2((ye-ys),(xe-xs));

Angle2 = atan2((y\_edge-ys),(x\_edge-xs));

Angle = abs(Angle1-Angle2);

if Angle >= pi\*1.5 + 1e-5

Angle = Angle - pi/2-pi;

elseif Angle >=pi + 1e-5

Angle = pi\*1.5-Angle;

elseif Angle >=pi/2 + 1e-5

Angle = Angle - pi/2;

else

Angle = pi/2 - Angle;

end

end

%}

function [vertices,segment, v\_num, my\_unit ,u] = basic

%% basic requests for vertieces

v\_num = str2double( input('How many vertices? Please enter a valid integer','s'));

while isnan(v\_num) || fix(v\_num)~= v\_num || v\_num<=2

v\_num = str2double( input('How many vertices? Please enter an integer','s'));

end

vertices = zeros (2,v\_num);

for i=1:v\_num

disp(['For #',num2str(i),'vertex']);

vertices(1,i) = str2double( input(' x is','s') );

while isnan( vertices(1,i) )

vertices(1,i) = str2double( input('Not a number, try again. x is','s') );

end

vertices(2,i) = str2double( input('y is','s') );

while isnan(vertices(2,i) )

vertices(2,i) = str2double( input('Not a number, try again. y is','s'));

end

end

segment = str2double( input('How many segment per edge? Please enter a valid integer','s'));

while isnan(segment) || fix(segment)~= segment || segment<=1

segment = str2double( input('How many segments per edge? Please enter an positive integer greater than 1','s'));

end

%%get the friction coefficient

u = str2double( input(' friction coefficient is','s') );

while isnan( u) || u<=0 || u>1

if u == 0

disp('This project only discusses frictional cases.');

end

u = str2double( input('Not a valid number, try again. friction coefficient is','s') );

end

my\_unit = input('what is the unit\n','s');

end

function [f1,f2,c] = findVector(edge,cp)

exs = vertices(1,edge); exe = vertices(1,1+mod(edge,v\_num));

eys = vertices(2,edge); eye = vertices(2,1+mod(edge,v\_num));

xe\_interval = (exe-exs) / segment; ye\_interval = (eye-eys) / segment;

cx = exs+xe\_interval\*cp; cy = eys+ye\_interval\*cp;

xq = zeros(1,2) ; yq = zeros(1,2);

if exs == exe

xq(1) = cx + 0.05; xq(2) = cx-0.05;

yq(1) = cy; yq(2) = cy;

elseif eys == eye

xq(1) = cx; xq(2) = cx;

yq(1) = cy+0.05; yq(2) = cy-0.05;

else

xq(1) = cx + 0.05; xq(2) = cx-0.05;

yq(1) =- 0.05\* (exe-exs)/(eye-eys) + cy;

yq(2) = 0.05\* (exe-exs)/(eye-eys) + cy;

end

[in, on] = inpolygon(xq,yq,vertices(1,:),vertices(2,:));

inside\_y = yq((in - on)>0);

inside\_x = xq((in - on)>0);

length =sqrt((inside\_x(1)-cx)^2 + (inside\_y(1)-cy)^2);

if exs == exe

f1 = 20\*[inside\_x-cx; length\*u];

f2 = 20\*[inside\_x-cx;- length \* u];

elseif eys == eye

f1 = 20\*[length\*u;inside\_y-cy];

f2 = 20\*[-length\*u;inside\_y-cy];

else

slope = (eye - eys)/(exe-exs);

slope\_a = atan(slope);

if slope\_a < 0

slope\_a = slope\_a +pi;

end

f1 = 20\*[inside\_x+cos(slope\_a)\*u\*length-cx;inside\_y+sin(slope\_a) \*u\* length-cy];

f2 = 20\*[inside\_x-cos(slope\_a)\*length-cx;inside\_y-sin(slope\_a) \*u\* length-cy];

end

c = [cx;cy];

end

end

project2\_2pointFC.m

function project2\_2pointFC

clear all;

close all;

clc;

%% MATLAB version: R2016a

%{

Author: DU DONGHONG

20328343

dduaa@connect.ust.hk

2018 SPRING ELEC4010M PROJECT2

extra interactive choosing point

%}

[vertices,segment, v\_num, my\_unit ,u] = basic;

disp('Please just drag the legend away if it blocks something');

disp('We do not consider the vertice point for its unstability');

disp('Red line for shape');

disp('Blue o for the contact you choose');

disp('Blue x for corresponding contact that could form a 2 contact points FC.')

disp('If there is no x, it means no solution for this contact');

contact = zeros(v\_num, 2, segment-1);

x=[]; y = [];

for i = 1: v\_num

xs = vertices(1,i); xe = vertices(1,1+mod(i+1,v\_num));

ys = vertices(2,i); ye = vertices(2,1+mod(i+1,v\_num));

x=[x xs]; y=[y ys];

x\_interval = (xe-xs) / segment;

y\_interval = (ye-ys) / segment;

for j =1:segment -1

contact( i , 1 , j) = xs + x\_interval \* j;

contact( i , 2 , j) = ys + y\_interval \* j;

x=[x contact( i , 1 , j)];

y=[y contact( i , 2 , j)];

end

x=[x xe];

y=[y ye];

end

%% basic of GUI

FigH = figure('position',[360 500 400 400],'name','find the possible solution for a contact for FC');

axes( 'units','pixels', 'position',[100 50 200 200]);

LineH = plot(x,y,'r');

title(['unit is ' my\_unit]);

axis equal;

xlabel('X') % x-axis label

ylabel('Y') % y-axis label

legend('shape');

TextH\_e = uicontrol('style','text','position',[170 310 60 15]);

TextH\_e.String =sprintf('edge# 1');

TextH\_c = uicontrol('style','text','position',[170 350 60 15]);

TextH\_c.String =sprintf('contact# 1');

SliderC = uicontrol('style','slider','position',[100 330 200 20],'min', 1, 'max', segment-1,'value',1,'SliderStep',[1/(segment-1) 1], 'Callback', @callbackfnC);

SliderE = uicontrol('style','slider','position',[100 290 200 20], 'min', 1, 'max', v\_num,'value',1,'SliderStep',[1/(v\_num-1) 1], 'Callback', @callbackfnE);

if(~exist('quality1Button','var'))

quality1Button = uicontrol('Style','radiobutton','String','quality measure extra','pos',[0 0 200 25],'parent',FigH,'Callback', @callbackfnQ1);

end

set(quality1Button,'Value',0);

Contact = 1; Edge = 1;

movegui(FigH, 'center')

replot;

%% real-time response

function callbackfnQ1(source, eventdata)

if(get(quality1Button,'Value')==1)

replot;

else

close(figure(3));

end

end

function callbackfnE(source, eventdata)

Edge =source.Value;

Edge = round(Edge);

set(source,'Value', Edge);

TextH\_e.String =sprintf('edge# %d',Edge);

replot;

end

function callbackfnC(source, eventdata)

Contact =source.Value;

Contact = round(Contact);

set(source,'Value', Contact);

TextH\_c.String =sprintf('contact# %d',Contact);

replot;

end

function replot

if(get(quality1Button,'Value')==1)

fig3= figure(3);

cla(fig3);

set(fig3,'name','FC quality measure with 2 contacts, angle method');

end

LineH = figure(1);

hold off;

LineH = plot(x,y,'r');

hold on;

LineH = plot( contact(Edge ,1 ,Contact),contact(Edge ,2 ,Contact),'ob');

title(['unit is ' my\_unit]);

axis equal;

xlabel('X') % x-axis label

ylabel('Y') % y-axis label

legend('shape','chosen contact');

%% find corresponding FC contact point

max\_q = 0;

for m = 1:v\_num

if m~= Edge

for p = 1: segment -1

x\_point\_one = contact( m, 1, p ); y\_point\_one = contact( m, 2, p );

x\_point\_two = contact(Edge ,1 ,Contact); y\_point\_two = contact(Edge ,2 ,Contact);

angle\_i = angle (x\_point\_one, y\_point\_one,x\_point\_two,y\_point\_two, vertices(1,m),vertices(2,m) );

angle\_j = angle (x\_point\_two, y\_point\_two,x\_point\_one,y\_point\_one, vertices(1,Edge),vertices(2,Edge) );

if angle\_i + 1e-5 < atan(u) && angle\_j + 1e-5 < atan(u)

LineH = figure(1);

hold on;

LineH = plot( contact(m ,1 ,p),contact(m ,2 ,p),'xb');

legend('shape','chosen contact','solution');

if(get(quality1Button,'Value')==1)

%% angle percentage

distance1 = sqrt( (x\_point\_two-vertices(1,Edge))^2+(y\_point\_two-vertices(1,Edge))^2 );

distance2 = sqrt( (x\_point\_one-vertices(1,m))^2+(y\_point\_one-vertices(1,m))^2 );

fig3= figure(3);

hold on;

z=exp(-angle\_j)+exp(-angle\_i);

if z > max\_q

max\_q = z;

end

stem3(distance1,distance2,z);

view(3);

end

end

end

end

end

if(get(quality1Button,'Value')==1)

fig3 = figure(3);

view(3);

xlabel('edge chosen first') % x-axis label

ylabel('edge chosen second') % y-axis label

fprintf('The maximum quality is %d .\n',max\_q);

end

end

function Angle = angle( xs , ys , xe , ye, x\_edge , y\_edge )

Angle1 = atan2((ye-ys),(xe-xs));

Angle2 = atan2((y\_edge-ys),(x\_edge-xs));

Angle = abs(Angle1-Angle2);

if Angle >= pi\*1.5 + 1e-5

Angle = Angle - pi/2-pi;

elseif Angle >=pi + 1e-5

Angle = pi\*1.5-Angle;

elseif Angle >=pi/2 + 1e-5

Angle = Angle - pi/2;

else

Angle = pi/2 - Angle;

end

end

function [vertices,segment, v\_num, my\_unit ,u] = basic

%% basic requests for vertieces

v\_num = str2double( input('How many vertices? Please enter a valid integer','s'));

while isnan(v\_num) || fix(v\_num)~= v\_num || v\_num<=2

v\_num = str2double( input('How many vertices? Please enter an integer','s'));

end

vertices = zeros (2,v\_num);

for i=1:v\_num

disp(['For #',num2str(i),'vertex']);

vertices(1,i) = str2double( input(' x is','s') );

while isnan( vertices(1,i) )

vertices(1,i) = str2double( input('Not a number, try again. x is','s') );

end

vertices(2,i) = str2double( input('y is','s') );

while isnan(vertices(2,i) )

vertices(2,i) = str2double( input('Not a number, try again. y is','s'));

end

end

segment = str2double( input('How many segment per edge? Please enter a valid integer','s'));

while isnan(segment) || fix(segment)~= segment || segment<=1

segment = str2double( input('How many segments per edge? Please enter an positive integer greater than 1','s'));

end

%%get the friction coefficient

u = str2double( input(' friction coefficient is','s') );

while isnan( u) || u<0 || u>1

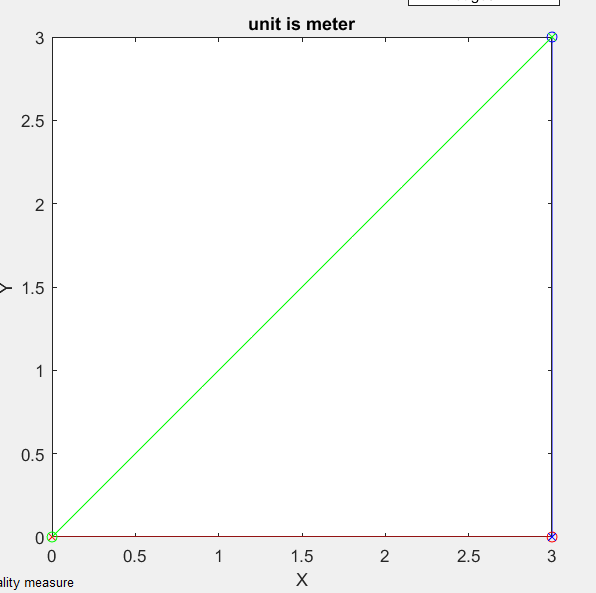
u = str2double( input('Not a number, try again. friction coefficient is','s') );

end

my\_unit = input('what is the unit\n','s');

end

end

Extra material:

Test case (just for convenience):

The following two inputs is to build triangle in the right

After the input, the UI can be click arbitrary.

Input1: (for project2.m two-contact mode)

0

0

3

3

3

0

10

1

Meter

2

Input2: (for project2.m three-contact mode)

0

0

3

3

3

0

10

1

Meter

3

Input3:(for project2\_2pointFC.m)

0

0

3

3

3

0

10

1

Meter