Peiyuan (Alexander) Liao

Csef 2018

research notebook

***Manipulator evaluation***: Motion of an anthropomorphic arm through a straight line in Cartesian space by interpolation & trajectory planning

Date: 9/3/17

Alexander Liao

Brainstorming

**I/O:**

Warning: floor tiles too small, making them 30.000000 x bigger - change the size or disable them

> In RTBPlot.create\_tiled\_floor (line 619)

In RTBPlot.create\_floor (line 575)

In SerialLink/plot (line 250)

In RobotTest (line 38)

linreg =

Linear regression model:

y ~ 1 + x1

Estimated Coefficients:

Estimate SE tStat pValue

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(Intercept) 10.039 0.0066362 1512.7 0

x1 -0.0019077 0.00053394 -3.5729 0.00045138

Number of observations: 185, Error degrees of freedom: 183

Root Mean Squared Error: 0.0508

R-squared: 0.0652, Adjusted R-Squared 0.0601

F-statistic vs. constant model: 12.8, p-value = 0.000451

**Code:**

clear Arm

t=[0 0 0 0 0 0 0];

l=[17.25 15.85 5.75];

xdata=[];

ydata=[];

L(1)= Link([t(1) 0 0 pi/2 0],'modified');

L(2)= Link([t(2)+pi/2 0 0 pi/2 0],'modified');

L(3)= Link([t(3)+pi/2 l(1) 0 pi/2 0],'modified');

L(4)= Link([t(4)+pi 0 0 pi/2 0],'modified');

L(5)= Link([t(5)+pi l(2) 0 pi/2 0],'modified');

L(6)= Link([t(6)+pi/2 0 0 pi/2 0],'modified');

L(7)= Link([t(7)+pi 0 l(3) pi/2 0],'modified');

Arm=SerialLink([L(1) L(2) L(3) L(4) L(5) L(6) L(7)]);

t=20;

%ti=0:t^-1:1;

er=ctraj(transl(0,10,10),transl(20,10,10),t);

solv=[];

solve1=Arm.jtraj(er(:,:,1),er(:,:,2),5);

finale=solve1;

for i=2:t-1

solv=Arm.jtraj(er(:,:,i),er(:,:,i+1),10);

finale =[finale;solv];

end

clear i

for i=1:size(finale)

for k=1:7

t(k)=finale(i,k);

end

intm=Arm.fkine(t);

intm2=intm.t;

xdata=horzcat(xdata,intm2(1));

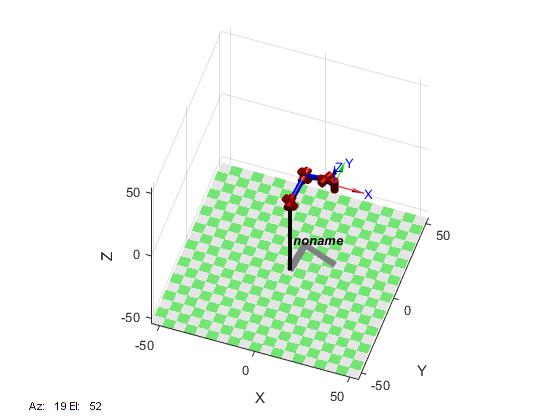
ydata=horzcat(ydata,intm2(2));

Arm.plot(t);

end

linreg=fitlm(xdata,ydata)

**Visualization:**



Improvement to allow

end

counter3 =counter3+1;

voxel1(:,:,counter3)=pixel1;

voxel2(:,:,counter3)=pixel2;

end

clear counter1 counter2

[sizeRow,sizeColumn,size3D]=size(voxel1);

%JointAngles=zeros(2,7);

for counter1=1:size3D

for counter2=1:sizeRow

LinPlan=trinterp(transl(voxel1(counter2,:,counter1)),transl(voxel2(counter2,:,counter1)),vxlPerRow);

TrajPlan=M.ikine(LinPlan);

JointAngles=[JointAngles;TrajPlan];

%SurfPlan=cat(4,SurfPlan,LinPlan);

end

for counter3=1:size(JointAngles)

hmTrans=M.fkine(JointAngles(counter3,:));

vector=hmTrans.t;

endX=horzcat(endX,vector(1));

endY=horzcat(endY,vector(2));

M.plot(JointAngles(counter3,:))

end

plot(endX,endY)

hold on

%clear counter2

%TrajPlan=Manipulator.ikinem(SurfPlan(:,:,1,counter1),SurfPlan(:,:,2,counter1),subVxlDensity)

%JointAngles=TrajPlan

%for counter2=2:vxlPerRow-1

% TrajPlan=Manipulator.jtraj(SurfPlan(:,:,counter2,counter1),SurfPlan(:,:,counter2+1,counter1),subVxlDensity);

% JointAngles=[JointAngles;TrajPlan];

%end

end

clear counter2

clear counter1 counter2 counter3

Date: 9/28/17

Alexander Liao

Brainstorming

planning in Cartesian

3D space:

**Code:**

%Initialization

clear all

theta=[0 0 0 0 0 0 0];

l=[17.25 15.85 5.75];

endX=[];

endY=[];

TrajPlan=[];

LinPlan=[];

SurfPlan=[];

JointSpaceTraj=[];

JointAngles=[];

hmTrans=[];

vector=[];

%subVxlDensity=19;

vxlPerRow=9;

counter3=0;

L(1)= Link([theta(1) 0 0 pi/2 0],'modified');

L(2)= Link([theta(2)+pi/2 0 0 pi/2 0],'modified');

L(3)= Link([theta(3)+pi/2 l(1) 0 pi/2 0],'modified');

L(4)= Link([theta(4)+pi 0 0 pi/2 0],'modified');

L(5)= Link([theta(5)+pi l(2) 0 pi/2 0],'modified');

L(6)= Link([theta(6)+pi/2 0 0 pi/2 0],'modified');

L(7)= Link([theta(7)+pi 0 l(3) pi/2 0],'modified');

M=SerialLink([L(1) L(2) L(3) L(4) L(5) L(6) L(7)]);

pixel=[];

%voxel=zeros(2\*(voxelPerRow),3,voxelPerRow);

for counter1=-20:20/(vxlPerRow-1):0

pixel1=[];

pixel2=[];

for counter2=10:20/(vxlPerRow-1):30

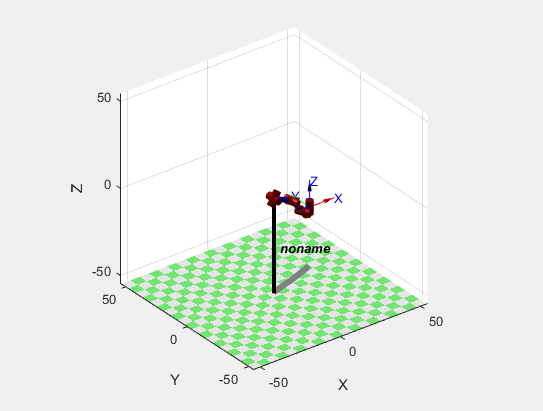
pixel1=vertcat(pixel1,[10,counter2,counter1]);

pixel2=vertcat(pixel2,[30,counter2,counter1]);

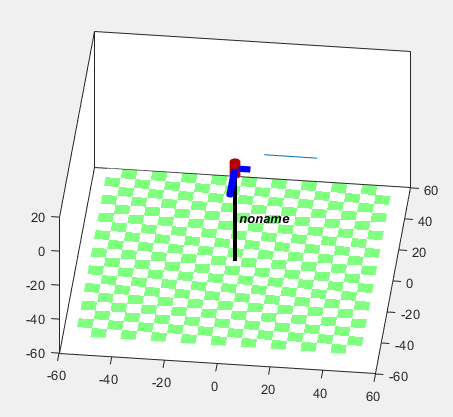
%pixel=vertcat(pixel,[counter2,10,counter1]);

%pixel=vertcat(pixel,[counter2,30,counter1]);

**Visualization 1:**



**Visualization 2 (failed when plotting the line of motion):**



**I/O:**

Warning: floor tiles too small, making them 30.000000 x bigger - change the size or disable them

> In RTBPlot.create\_tiled\_floor (line 619)

In RTBPlot.create\_floor (line 575)

In SerialLink/plot (line 250)

In Test (line 63)

Error using matlab.graphics.primitive.Group/set

Invalid or deleted object.

Error in SerialLink/animate (line 148)

set(handle, 'UserData', h);

Error in SerialLink/plot (line 297)

robot.animate(qq);

Error in Test (line 63)

M.plot(JointAngles(counter3,:))

**Workspace:**

**Pose planning**

LinPlan(:,:,1) =

1.0000 0 0 10.0000

0 1.0000 0 30.0000

0 0 1.0000 -17.5000

0 0 0 1.0000

LinPlan(:,:,2) =

1.0000 0 0 12.5000

0 1.0000 0 30.0000

0 0 1.0000 -17.5000

0 0 0 1.0000

LinPlan(:,:,3) =

1.0000 0 0 15.0000

0 1.0000 0 30.0000

0 0 1.0000 -17.5000

0 0 0 1.0000

LinPlan(:,:,4) =

1.0000 0 0 17.5000

0 1.0000 0 30.0000

0 0 1.0000 -17.5000

0 0 0 1.0000

LinPlan(:,:,5) =

1.0000 0 0 20.0000

0 1.0000 0 30.0000

0 0 1.0000 -17.5000

0 0 0 1.0000

**Workspace:**

JointAngles =

-0.4940 0.8524 1.2426 1.4271 0.0239 -0.5089 -0.7566

-0.3986 0.9342 1.3036 1.4916 -0.0537 -0.3972 -0.7325

-0.3292 1.0172 1.3613 1.5766 -0.1128 -0.2746 -0.6959

-0.2849 1.0957 1.4151 1.6820 -0.1534 -0.1431 -0.6494

-0.2650 1.1649 1.4633 1.8096 -0.1765 -0.0042 -0.5957

-0.2692 1.2217 1.5037 1.9638 -0.1852 0.1427 -0.5377

-0.3002 1.2653 1.5344 2.1539 -0.1833 0.3015 -0.4782

-0.3682 1.2968 1.5537 2.4043 -0.1755 0.4849 -0.4180

-0.5296 1.3179 1.5574 2.8316 -0.1687 0.7531 -0.3497

-0.3465 0.7728 1.1908 1.2621 -0.2330 -0.5622 -1.0570

-0.2384 0.8893 1.2739 1.3388 -0.3396 -0.4406 -1.0316

-0.1628 1.0120 1.3609 1.4319 -0.4200 -0.3027 -0.9847

-0.1182 1.1292 1.4499 1.5399 -0.4699 -0.1545 -0.9169

-0.1017 1.2289 1.5351 1.6637 -0.4895 -0.0033 -0.8332

-0.1097 1.3042 1.6107 1.8076 -0.4849 0.1475 -0.7413

-0.1403 1.3539 1.6743 1.9785 -0.4646 0.2996 -0.6472

-0.1962 1.3792 1.7250 2.1897 -0.4351 0.4606 -0.5526

-0.2920 1.3795 1.7624 2.4758 -0.4011 0.6492 -0.4537

-0.2864 0.6998 1.3830 1.3181 0.0787 -0.4369 -0.8139

-0.2048 0.7956 1.4300 1.3864 0.0110 -0.3302 -0.7748

-0.1509 0.8864 1.4667 1.4744 -0.0389 -0.2133 -0.7289

-0.1207 0.9689 1.4957 1.5820 -0.0733 -0.0884 -0.6791

-0.1121 1.0406 1.5180 1.7106 -0.0946 0.0433 -0.6271

-0.1247 1.1009 1.5334 1.8634 -0.1057 0.1824 -0.5749

-0.1605 1.1505 1.5410 2.0475 -0.1099 0.3321 -0.5238

-0.2266 1.1912 1.5400 2.2798 -0.1109 0.5011 -0.4746

-0.3504 1.2265 1.5276 2.6167 -0.1144 0.7191 -0.4252

-0.2511 0.5913 1.4258 1.4100 0.2142 -0.3067 -0.7806

-0.1851 0.6831 1.4607 1.4783 0.1618 -0.2117 -0.7339

-0.1457 0.7671 1.4785 1.5667 0.1225 -0.1058 -0.6859

-0.1286 0.8425 1.4841 1.6759 0.0934 0.0093 -0.6392

-0.1319 0.9095 1.4799 1.8077 0.0718 0.1333 -0.5953

-0.1563 0.9697 1.4673 1.9669 0.0549 0.2677 -0.5557

-0.2055 1.0260 1.4469 2.1632 0.0395 0.4173 -0.5210

-0.2915 1.0827 1.4186 2.4228 0.0214 0.5955 -0.4916

-0.4865 1.1564 1.3724 2.8915 -0.0133 0.8784 -0.4653

First attempt in formulating an evaluation method: absolute error between planned trajectory and straight line between nodes of cuboid partitions

Date: 9/29/17

Alexander Liao

Brainstorming

**Code:**

%Initialization

clear all

%SerialLink Parameters

theta=[0 0 0 0 0 0 0];

l=[17.25 15.85 5.75];

%Target Space

x1=10;

x2=20;

y1=10;

y2=20;

z1=-8;

z2=2;

%Specifications for Voxels

interpVal=3;

numVxl=5;

zLayerHeight=5;

%Final Result

dataGraySclImage=[];

%Creating the SerialLink object

L(1)= Link([theta(1) 0 0 pi/2 0],'modified');

L(2)= Link([theta(2)+pi/2 0 0 pi/2 0],'modified');

L(3)= Link([theta(3)+pi/2 l(1) 0 pi/2 0],'modified');

L(4)= Link([theta(4)+pi 0 0 pi/2 0],'modified');

L(5)= Link([theta(5)+pi l(2) 0 pi/2 0],'modified');

L(6)= Link([theta(6)+pi/2 0 0 pi/2 0],'modified');

L(7)= Link([theta(7)+pi 0 l(3) pi/2 0],'modified');

M=SerialLink([L(1) L(2) L(3) L(4) L(5) L(6) L(7)]);

%Generating grayscale image

for z=z1:zLayerHeight:z2;

if z==z2

disp('Test')

else

%Invoking the surface function for at the height "z"

[surfPlan,SizeZ] = surfaceRobotTest(L,M,interpVal,numVxl,x1,x2,y1,y2,z);

[SizeX,SizeY]=size(surfPlan);

endX=zeros(1,SizeX);

endY=zeros(1,SizeY);

%Calculating the trajectories from joint angle to Cartesian Coordinates

for i=1:size(surfPlan);

hmTrans=M.fkine(surfPlan(i,:));

vector=hmTrans.t;

endX(i)=vector(1);

endY(i)=vector(2);

%Optional Visualization: M.plot(surfPlan(i,:))

end

%Prelocating space for intermidiate matrices to save time

[~,y]=size(endY);

preFitx=zeros(1,(SizeZ-1)\*interpVal);

preFity=zeros(1,(SizeZ-1)\*interpVal);

standardD=zeros(1,y/((SizeZ-1)\*interpVal));

a=1;

b=(SizeZ-1)\*interpVal;

%Finding the standard deviation of actual trajectories to ideal

%ones

for i=1:(y/((SizeZ-1)\*interpVal))

for k=a:b

preFitx(k)=endX(1,k);

preFity(k)=endY(1,k);

end

[~,y3]=size(preFitx);

[~,y4]=size(endX);

normalx=preFitx(1):(preFitx(k)-preFitx(1))/(y3-1):preFitx(k);

normaly=preFity(1):(preFity(k)-preFity(1))/(y3-1):preFity(k);

for k1=1:1:((y4)/2)

preFity(k)=normaly(k)

end

for k2=(((y4)/2)+1):1:y4

preFitx(k)=normalx(k)

end

standardD=std([preFity;preFitx],0,1);

a=a+(SizeZ-1)\*interpVal;

b=b+(SizeZ-1)\*interpVal;

end

rowGrayScale=mat2gray(standardD);

dataGraySclImage=vertcat(dataGraySclImage,rowGrayScale);

end

end

%Trajectory Planning for a surface

function [surfPlan,SizeZ]= surfaceRobotTest(L,M,interpVal,numVxl,x1,x2,y1,y2,z)

surfPlan=[];

%Linear test along the y-direction

for i=x1:x2

terminal1=transl(i,y1,z);

terminal2=transl(i,y2,z);

[linPlan,SizeZ]=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

SizeZ=SizeZ;

%Linear test along the x-direction

for i=y1:y2

terminal1=transl(x1,i,z);

terminal2=transl(x2,i,z);

linPlan=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

end

%Trajectory Planning for a line

function [linPlan,SizeZ] = linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2)

linVoxel=trinterp(terminal1,terminal2,numVxl-1);

[~,~,SizeZ]=size(linVoxel);

linPlan=zeros((SizeZ-1)\*interpVal,numel(L));

for i=1:SizeZ-1

ptPlan=M.jtraj(linVoxel(:,:,i),linVoxel(:,:,i+1),interpVal);

index1=1+(i-1)\*interpVal;

index2=i\*interpVal;

linPlan(index1:index2,:)=ptPlan(:,:);

end

end

for i=1:size(surfPlan);

hmTrans=M.fkine(surfPlan(i,:));

vector=hmTrans.t;

endX(i)=vector(1);

endY(i)=vector(2);

%Optional Visualization: M.plot(surfPlan(i,:))

end

%Prelocating space for intermidiate matrices to save time

[~,y]=size(endY);

preFitx=zeros(1,(SizeZ-1)\*interpVal);

preFity=zeros(1,(SizeZ-1)\*interpVal);

standardD=zeros(1,y/((SizeZ-1)\*interpVal));

a=1;

b=(SizeZ-1)\*interpVal;

%Finding the standard deviation of actual trajectories to ideal

%ones

for i=1:(y/((SizeZ-1)\*interpVal))

for k=a:b

preFitx(k)=endX(1,k);

preFity(k)=endY(1,k);

end

[~,y3]=size(preFitx);

[~,y4]=size(endX);

normalx=preFitx(1):(preFitx(k)-preFitx(1))/(y3-1):preFitx(k);

normaly=preFity(1):(preFity(k)-preFity(1))/(y3-1):preFity(k);

for k1=1:1:((y4)/2)

preFity(k)=normaly(k)

end

for k2=(((y4)/2)+1):1:y4

preFitx(k)=normalx(k)

end

standardD=std([preFity;preFitx],0,1);

a=a+(SizeZ-1)\*interpVal;

b=b+(SizeZ-1)\*interpVal;

end

rowGrayScale=mat2gray(standardD);

dataGraySclImage=vertcat(dataGraySclImage,rowGrayScale);

end

end

%Trajectory Planning for a surface

function [surfPlan,SizeZ]= surfaceRobotTest(L,M,interpVal,numVxl,x1,x2,y1,y2,z)

surfPlan=[];

%Linear test along the y-direction

for i=x1:x2

terminal1=transl(i,y1,z);

terminal2=transl(i,y2,z);

[linPlan,SizeZ]=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

SizeZ=SizeZ;

%Linear test along the x-direction

for i=y1:y2

terminal1=transl(x1,i,z);

terminal2=transl(x2,i,z);

linPlan=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

end

%Trajectory Planning for a line

function [linPlan,SizeZ] = linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2)

linVoxel=trinterp(terminal1,terminal2,numVxl-1);

[~,~,SizeZ]=size(linVoxel);

linPlan=zeros((SizeZ-1)\*interpVal,numel(L));

for i=1:SizeZ-1

ptPlan=M.jtraj(linVoxel(:,:,i),linVoxel(:,:,i+1),interpVal);

index1=1+(i-1)\*interpVal;

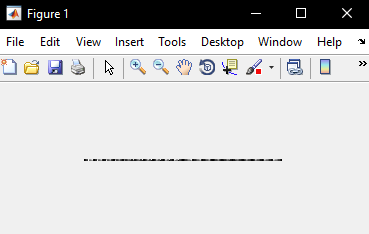
index2=i\*interpVal;

linPlan(index1:index2,:)=ptPlan(:,:);

end

end

**Visualization 1 (in grayscale image: darker means less error**



[linPlan,SizeZ]=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

SizeZ=SizeZ;

%Linear test along the x-direction

for i=y1:y2

terminal1=transl(x1,i,z);

terminal2=transl(x2,i,z);

linPlan=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

end

%Trajectory Planning for a line

function [linPlan,SizeZ] = linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2)

linVoxel=trinterp(terminal1,terminal2,numVxl-1);

[~,~,SizeZ]=size(linVoxel);

linPlan=zeros((SizeZ-1)\*interpVal,numel(L));

for i=1:SizeZ-1

ptPlan=M.jtraj(linVoxel(:,:,i),linVoxel(:,:,i+1),interpVal);

index1=1+(i-1)\*interpVal;

index2=i\*interpVal;

linPlan(index1:index2,:)=ptPlan(:,:);

end

end

**Workspace:**

**Trajectory planning of motion in 2D plane:**

Joint angles=

0.0162 0.1810 0.2488 0.9734 -1.4406 -1.3771 -2.3510

0.0151 0.1193 0.2329 1.1004 -1.4715 -1.3783 -2.5425

0.0140 0.0576 0.2170 1.2274 -1.5024 -1.3795 -2.7341

0.0140 0.0576 0.2170 1.2274 -1.5024 -1.3795 -2.7341

-0.8730 -0.3857 -0.1243 1.0747 -0.4216 -0.5829 -1.6583

-1.7600 -0.8289 -0.4655 0.9220 0.6592 0.2136 -0.5824

-1.7600 -0.8289 -0.4655 0.9220 0.6592 0.2136 -0.5824

-1.5690 -0.7554 -0.3825 1.0414 0.7982 0.3721 -0.5414

-1.3780 -0.6819 -0.2995 1.1608 0.9373 0.5306 -0.5004

0.4527 0.2919 0.7479 1.0388 -1.5221 -1.2385 -2.3574

0.2586 0.2036 0.5083 1.1650 -1.5215 -1.3053 -2.5500

0.0644 0.1153 0.2686 1.2911 -1.5208 -1.3722 -2.7427

0.0644 0.1153 0.2686 1.2911 -1.5208 -1.3722 -2.7427

-0.7971 -0.3293 -0.0995 1.1401 -0.3322 -0.5225 -1.5766

-1.6586 -0.7738 -0.4676 0.9891 0.8565 0.3272 -0.4106

-1.6586 -0.7738 -0.4676 0.9891 0.8565 0.3272 -0.4106

-1.4532 -0.6990 -0.3862 1.1008 0.9612 0.4916 -0.4053

-1.2477 -0.6242 -0.3049 1.2124 1.0660 0.6560 -0.3999

0.4542 0.3534 0.7597 1.1161 -1.5662 -1.2084 -2.3914

0.3225 0.2663 0.5868 1.2378 -1.5614 -1.2756 -2.5739

0.1908 0.1793 0.4138 1.3595 -1.5565 -1.3427 -2.7565

0.1908 0.1793 0.4138 1.3595 -1.5565 -1.3427 -2.7565

-0.6575 -0.2573 -0.0272 1.2092 -0.2566 -0.4280 -1.5091

-1.5059 -0.6939 -0.4683 1.0589 1.0432 0.4868 -0.2617

-1.5059 -0.6939 -0.4683 1.0589 1.0432 0.4868 -0.2617

-1.3186 -0.6289 -0.3911 1.1592 1.1025 0.6300 -0.2949

-1.1313 -0.5639 -0.3138 1.2595 1.1617 0.7732 -0.3281

0.4861 0.4276 0.8172 1.1990 -1.6275 -1.1592 -2.4351

0.4907 0.3612 0.8139 1.3181 -1.6490 -1.1918 -2.6115

0.4954 0.2948 0.8105 1.4372 -1.6706 -1.2243 -2.7878

0.4954 0.2948 0.8105 1.4372 -1.6706 -1.2243 -2.7878

-0.4012 -0.1489 0.1795 1.2805 -0.2318 -0.2698 -1.4689

-1.2979 -0.5926 -0.4515 1.1237 1.2070 0.6847 -0.1500

-1.2979 -0.5926 -0.4515 1.1237 1.2070 0.6847 -0.1500

-1.1444 -0.5443 -0.3783 1.2149 1.2304 0.7917 -0.2097

-0.9910 -0.4961 -0.3051 1.3062 1.2539 0.8987 -0.2693

0.5513 0.5310 0.9398 1.2927 -1.7302 -1.0725 -2.5025

0.5398 0.4582 0.9203 1.4084 -1.7442 -1.1184 -2.6691

0.5282 0.3854 0.9008 1.5240 -1.7582 -1.1643 -2.8358

0.5282 0.3854 0.9008 1.5240 -1.7582 -1.1643 -2.8358

-0.2288 -0.0442 0.2680 1.3530 -0.1972 -0.1204 -1.4537

-0.9857 -0.4737 -0.3648 1.1821 1.3638 0.9234 -0.0716

-0.9857 -0.4737 -0.3648 1.1821 1.3638 0.9234 -0.0716

-0.8620 -0.4447 -0.2939 1.2682 1.3685 0.9939 -0.1442

-0.7383 -0.4156 -0.2231 1.3544 1.3733 1.0644 -0.2168

0.6224 0.6798 1.1242 1.4124 -1.9206 -0.9421 -2.6373

0.5706 0.5720 1.0332 1.5152 -1.8818 -1.0337 -2.7659

0.5188 0.4641 0.9423 1.6180 -1.8429 -1.1254 -2.8945

0.5188 0.4641 0.9423 1.6180 -1.8429 -1.1254 -2.8945

-0.1203 0.0389 0.3288 1.4238 -0.1974 -0.0229 -1.4647

-0.7594 -0.3864 -0.2847 1.2295 1.4482 1.0797 -0.0349

-0.7594 -0.3864 -0.2847 1.2295 1.4482 1.0797 -0.0349

-0.6771 -0.3690 -0.2311 1.3135 1.4406 1.1198 -0.1118

-0.5949 -0.3516 -0.1775 1.3975 1.4329 1.1600 -0.1888

0.6406 0.8293 1.3004 1.5613 -2.1950 -0.8380 -2.8644

0.5574 0.6755 1.1140 1.6380 -2.0490 -0.9783 -2.9071

0.4742 0.5218 0.9276 1.7148 -1.9029 -1.1186 -2.9498

0.4742 0.5218 0.9276 1.7148 -1.9029 -1.1186 -2.9498

-0.0287 0.1055 0.3773 1.4951 -0.1945 0.0430 -1.4808

-0.5317 -0.3108 -0.1730 1.2754 1.5139 1.2046 -0.0118

-0.5317 -0.3108 -0.1730 1.2754 1.5139 1.2046 -0.0118

-0.4886 -0.3006 -0.1421 1.3583 1.4988 1.2230 -0.0900

-0.4455 -0.2903 -0.1112 1.4413 1.4838 1.2414 -0.1681

0.6000 0.8783 1.3207 1.6735 -2.3105 -0.8471 -2.9753

0.5155 0.7286 1.1196 1.7459 -2.1384 -0.9822 -2.9932

0.4311 0.5789 0.9184 1.8182 -1.9664 -1.1173 -3.0111

0.4311 0.5789 0.9184 1.8182 -1.9664 -1.1173 -3.0111

0.0546 0.1661 0.4360 1.5699 -0.2031 0.0883 -1.5041

-0.3218 -0.2467 -0.0464 1.3216 1.5603 1.2939 0.0029

-0.3218 -0.2467 -0.0464 1.3216 1.5603 1.2939 0.0029

-0.3103 -0.2395 -0.0379 1.4040 1.5418 1.2993 -0.0752

-0.2989 -0.2323 -0.0295 1.4865 1.5233 1.3046 -0.1533

0.4651 0.7652 0.9997 1.7126 -2.0465 -1.0153 -2.8197

0.4429 0.7148 0.9998 1.8263 -2.0745 -1.0522 -2.9657

0.4207 0.6643 0.9998 1.9401 -2.1025 -1.0891 -3.1116

0.4207 0.6643 0.9998 1.9401 -2.1025 -1.0891 -3.1116

0.1489 0.2373 0.5475 1.6547 -0.2555 0.1328 -1.5494

-0.1230 -0.1898 0.0951 1.3693 1.5915 1.3547 0.0129

-0.1230 -0.1898 0.0951 1.3693 1.5915 1.3547 0.0129

-0.1390 -0.1834 0.0812 1.4514 1.5714 1.3524 -0.0650

-0.1550 -0.1769 0.0672 1.5336 1.5513 1.3502 -0.1428

0.7200 1.6227 1.9723 1.0912 -0.8762 0.4372 -1.0640

0.5638 1.1937 1.5537 1.5851 -1.5915 -0.3119 0.9839

0.4077 0.7648 1.1350 2.0789 -2.3069 -1.0610 3.0317

0.4077 0.7648 1.1350 2.0789 -2.3069 -1.0610 3.0317

0.2363 0.3143 0.6918 1.7490 -0.3496 0.1646 1.5258

0.0650 -0.1362 0.2486 1.4192 1.6078 1.3901 0.0198

0.0650 -0.1362 0.2486 1.4192 1.6078 1.3901 0.0198

0.0239 -0.1295 0.2122 1.5011 1.5870 1.3841 -0.0579

-0.0172 -0.1227 0.1758 1.5830 1.5663 1.3780 -0.1356

0.7838 1.4140 1.9059 1.0881 -0.4173 0.3233 -0.7443

0.5748 1.1314 1.5749 1.6544 -1.4593 -0.3727 1.0788

0.3657 0.8488 1.2440 2.2207 -2.5013 -1.0686 2.9019

0.3657 0.8488 1.2440 2.2207 -2.5013 -1.0686 2.9019

0.3045 0.3835 0.8290 1.8461 -0.4468 0.1668 1.4633

0.2433 -0.0818 0.4140 1.4716 1.6078 1.4022 0.0246

0.2433 -0.0818 0.4140 1.4716 1.6078 1.4022 0.0246

0.1747 -0.0746 0.3512 1.5534 1.5870 1.3952 -0.0533

0.1061 -0.0673 0.2883 1.6352 1.5662 1.3883 -0.1311

0.0162 0.1810 0.2488 0.9734 -1.4406 -1.3771 -2.3510

0.2603 0.3193 0.5494 1.1010 -1.5479 -1.2563 -2.4023

0.5044 0.4577 0.8500 1.2286 -1.6552 -1.1355 -2.4536

0.5044 0.4577 0.8500 1.2286 -1.6552 -1.1355 -2.4536

0.5620 0.6688 1.0984 1.4370 -1.9854 -0.9818 -2.7135

0.6195 0.8799 1.3467 1.6454 -2.3155 -0.8281 -2.9735

0.6195 0.8799 1.3467 1.6454 -2.3155 -0.8281 -2.9735

0.7017 1.1470 1.6263 1.3668 -1.3664 -0.2524 -1.8589

0.7838 1.4140 1.9059 1.0881 -0.4173 0.3233 -0.7443

0.4210 0.1652 0.6845 1.0414 -1.4923 -1.3052 -2.4598

0.5390 0.3350 0.8839 1.1740 -1.6298 -1.1700 -2.5271

0.6570 0.5048 1.0833 1.3066 -1.7673 -1.0347 -2.5944

0.6570 0.5048 1.0833 1.3066 -1.7673 -1.0347 -2.5944

0.5433 0.5529 0.9658 1.4577 -1.8121 -1.0718 -2.6718

0.4296 0.6010 0.8483 1.6088 -1.8569 -1.1090 -2.7492

0.4296 0.6010 0.8483 1.6088 -1.8569 -1.1090 -2.7492

0.4339 0.7791 1.1019 1.8379 -2.2231 -1.0385 0.1228

0.4382 0.9573 1.3555 2.0669 -2.5894 -0.9681 2.9949

0.0813 0.0984 0.2986 1.1207 -1.4821 -1.3715 -2.5854

0.3703 0.2765 0.6896 1.2484 -1.6294 -1.2197 -2.6413

0.6593 0.4546 1.0806 1.3760 -1.7766 -1.0679 -2.6972

0.6593 0.4546 1.0806 1.3760 -1.7766 -1.0679 -2.6972

0.6128 0.5813 1.1338 1.5502 -1.9568 -1.0167 -2.8433

0.5663 0.7081 1.1869 1.7243 -2.1370 -0.9655 -2.9895

0.5663 0.7081 1.1869 1.7243 -2.1370 -0.9655 -2.9895

0.4169 0.7208 0.9743 1.8888 -2.0775 -1.0723 -3.0197

0.2674 0.7335 0.7617 2.0532 -2.0179 -1.1791 -3.0500

0.0548 0.0668 0.2621 1.2001 -1.4982 -1.3763 -2.6974

0.2985 0.2070 0.5762 1.3203 -1.6035 -1.2785 -2.7346

0.5422 0.3472 0.8904 1.4405 -1.7088 -1.1806 -2.7717

0.5422 0.3472 0.8904 1.4405 -1.7088 -1.1806 -2.7717

0.5184 0.4757 0.9559 1.6047 -1.8592 -1.1238 -2.8828

0.4947 0.6042 1.0215 1.7689 -2.0096 -1.0669 -2.9940

0.4947 0.6042 1.0215 1.7689 -2.0096 -1.0669 -2.9940

0.4389 0.7394 1.1599 1.9844 -2.2812 -1.0562 -0.0490

0.3831 0.8746 1.2984 2.1998 -2.5529 -1.0456 2.8959

0.1024 0.0437 0.3048 1.2828 -1.5169 -1.3754 -2.8053

0.3458 0.1922 0.6336 1.4032 -1.6353 -1.2721 -2.8442

0.5892 0.3406 0.9625 1.5235 -1.7536 -1.1688 -2.8831

0.5892 0.3406 0.9625 1.5235 -1.7536 -1.1688 -2.8831

0.5091 0.4411 0.9345 1.6798 -1.8527 -1.1516 -2.9670

0.4290 0.5417 0.9065 1.8360 -1.9519 -1.1344 -3.0508

0.4290 0.5417 0.9065 1.8360 -1.9519 -1.1344 -3.0508

0.3331 0.6285 0.8521 2.0335 -2.0264 -1.1700 -0.0088

0.2373 0.7154 0.7976 2.2310 -2.1010 -1.2057 3.0332

0.0015 0.0259 0.1959 1.3697 -1.5315 -1.3803 -2.9109

0.0615 0.1245 0.2678 1.4833 -1.5678 -1.3654 -2.9319

0.1215 0.2231 0.3396 1.5970 -1.6042 -1.3506 -2.9529

0.1215 0.2231 0.3396 1.5970 -1.6042 -1.3506 -2.9529

0.2514 0.3659 0.5913 1.7579 -1.7722 -1.2622 -3.0436

0.3813 0.5087 0.8431 1.9187 -1.9402 -1.1739 -3.1342

0.3813 0.5087 0.8431 1.9187 -1.9402 -1.1739 -3.1342

0.3012 0.6152 0.8468 2.1304 -2.0619 -1.1952 -0.1097

0.2211 0.7216 0.8506 2.3421 -2.1835 -1.2165 2.9148

-1.6988 -0.8134 -0.4272 0.9107 0.8152 0.2303 -0.4587

-1.2928 -0.6547 -0.3546 1.0193 1.1133 0.5846 -0.2517

-0.8868 -0.4960 -0.2821 1.1279 1.4114 0.9389 -0.0447

-0.8868 -0.4960 -0.2821 1.1279 1.4114 0.9389 -0.0447

-0.5746 -0.3805 -0.1389 1.2020 1.4957 1.1217 -0.0070

-0.2624 -0.2651 0.0043 1.2760 1.5800 1.3044 0.0308

-0.2624 -0.2651 0.0043 1.2760 1.5800 1.3044 0.0308

0.1889 -0.1851 0.3863 1.3588 1.6152 1.3770 0.0423

0.6401 -0.1052 0.7683 1.4415 1.6504 1.4495 0.0538

-1.5965 -0.7730 -0.3816 0.9769 0.8778 0.3209 -0.4475

-1.3656 -0.6541 -0.3901 1.0702 1.0804 0.5634 -0.2868

-1.1348 -0.5352 -0.3986 1.1634 1.2831 0.8060 -0.1262

-1.1348 -0.5352 -0.3986 1.1634 1.2831 0.8060 -0.1262

-0.7670 -0.4012 -0.2475 1.2424 1.4116 1.0355 -0.0717

-0.3992 -0.2671 -0.0965 1.3214 1.5401 1.2651 -0.0173

-0.3992 -0.2671 -0.0965 1.3214 1.5401 1.2651 -0.0173

-0.0813 -0.1745 0.1554 1.4040 1.5724 1.3334 -0.0042

0.2366 -0.0820 0.4073 1.4867 1.6047 1.4016 0.0090

-1.3704 -0.7086 -0.2911 1.0618 1.0398 0.4879 -0.3811

-1.1702 -0.6032 -0.3034 1.1401 1.1817 0.6963 -0.2689

-0.9700 -0.4977 -0.3156 1.2184 1.3235 0.9047 -0.1568

-0.9700 -0.4977 -0.3156 1.2184 1.3235 0.9047 -0.1568

-0.6769 -0.3811 -0.2000 1.2937 1.4266 1.0876 -0.1107

-0.3837 -0.2644 -0.0844 1.3690 1.5296 1.2706 -0.0646

-0.3837 -0.2644 -0.0844 1.3690 1.5296 1.2706 -0.0646

-0.1161 -0.1714 0.1223 1.4513 1.5595 1.3320 -0.0511

0.1515 -0.0784 0.3290 1.5336 1.5895 1.3935 -0.0376

-1.3528 -0.6921 -0.2865 1.1129 1.0042 0.5209 -0.4327

-1.1579 -0.5898 -0.2981 1.1907 1.1516 0.7192 -0.3196

-0.9630 -0.4875 -0.3098 1.2685 1.2989 0.9176 -0.2065

-0.9630 -0.4875 -0.3098 1.2685 1.2989 0.9176 -0.2065

-0.6728 -0.3735 -0.1966 1.3437 1.4077 1.0952 -0.1590

-0.3826 -0.2594 -0.0835 1.4189 1.5164 1.2729 -0.1116

-0.3826 -0.2594 -0.0835 1.4189 1.5164 1.2729 -0.1116

-0.1371 -0.1666 0.1031 1.5010 1.5465 1.3310 -0.0979

0.1084 -0.0738 0.2896 1.5830 1.5767 1.3892 -0.0843

-1.3780 -0.6819 -0.2995 1.1608 0.9373 0.5306 -0.5004

-1.1588 -0.5772 -0.2984 1.2411 1.1102 0.7357 -0.3766

-0.9397 -0.4726 -0.2973 1.3215 1.2832 0.9407 -0.2528

-0.9397 -0.4726 -0.2973 1.3215 1.2832 0.9407 -0.2528

-0.6424 -0.3619 -0.1770 1.3964 1.3975 1.1134 -0.2052

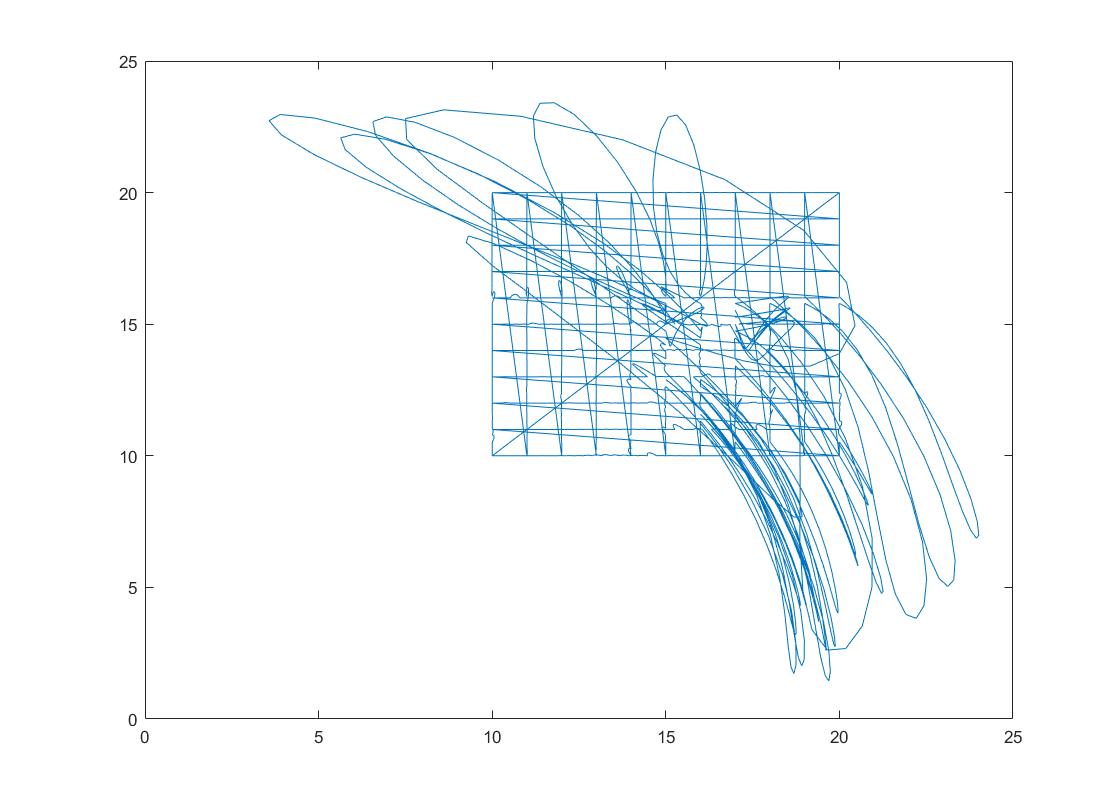
-0.3452 -0.2512 -0.0568 1.4712 1.5119 1.2861 -0.1577

-0.3452 -0.2512 -0.0568 1.4712 1.5119 1.2861 -0.1577

-0.1195 -0.1593 0.1158 1.5532 1.5390 1.3372 -0.1444

0.1061 -0.0673 0.2883 1.6352 1.5662 1.3883 -0.1311

**Visualization 2: Trajectory of the manipulator**



Tentative data generation: Gray scale images corresponding to the accuracy map of a manipulator in motion between nodes of cuboid partitions in Cartesian 3D space

Date: 9/30/17

Alexander Liao

Brainstorming

**Function 1:**

%Trajectory Planning for a line

function [linPlan,SizeZ] = linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2)

linVoxel=trinterp(terminal1,terminal2,numVxl);

[~,~,SizeZ]=size(linVoxel);

linPlan=zeros((SizeZ-1)\*interpVal,numel(L));

for i=1:SizeZ-1

intm=ctraj(linVoxel(:,:,i),linVoxel(:,:,i+1),interpVal);

ptPlan=M.ikine6s(intm);

index1=1+(i-1)\*interpVal;

index2=i\*interpVal;

linPlan(index1:index2,:)=ptPlan(:,:);

end

end

**Function 2:**

%Trajectory Planning for a surface

function [surfPlan,SizeZ]= surfaceRobotTest(L,M,interpVal,numVxl,x1,x2,y1,y2,z)

surfPlan=[];

for i=x1:abs((x2-x1)/(numVxl)):x2

terminal1=transl(i,y1,z);

terminal2=transl(i,y2,z);

[linPlan,SizeZ]=linearRobotTest(L,M,interpVal,numVxl,terminal1,terminal2);

surfPlan=vertcat(surfPlan,linPlan);

end

end

**Main file:**

%Initialization

clear all

numData=1;

%Target Space

spc\_x1=0.2;

xmin=spc\_x1;

spc\_x2=0.6;

xmax=spc\_x2;

spc\_y1=-0.2;

ymin=spc\_y1;

spc\_y2=0.2;

ymax=spc\_y2;

spc\_z1=-0.4;

zmin=spc\_z1;

spc\_z2=0;

zmax=spc\_z2;

%Specifications for Voxels

vxl\_interpVal=10;

vxl\_num=5;

vxl\_zLayerHeight=0.025;

%Final Result

performanceVector=[];

performanceMatrix=[];

inputMatrix=[];

x1=[xmin xmin xmax xmax];

y1=[ymin ymax ymax ymin];

z1=[zmin zmin zmin zmin];

x2=[xmin xmin xmax xmax];

y2=[ymin ymax ymax ymin];

z2=[zmax zmax zmax zmax];

x3=[xmin xmin xmin xmin];

y3=[ymin ymax ymax ymin];

z3=[zmax zmax zmin zmin];

x4=[xmax xmax xmax xmax];

y4=[ymin ymax ymax ymin];

z4=[zmax zmax zmin zmin];

x5=[xmin xmax xmax xmin];

y5=[ymin ymin ymin ymin];

z5=[zmax zmax zmin zmin];

x6=[xmin xmax xmax xmin];

y6=[ymax ymax ymax ymax];

z6=[zmax zmax zmin zmin];

for num=1:numData

rawdata=[];

performanceVector=[];

%{

%SerialLink Parameters

link\_theta=[0 0 0 0 0 0];

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[0 rand() 0 0 0 0];

link\_Alpha=[pi/2 0 -pi/2 pi/2 -pi/2 0];

linkType=[0 0 0 0 0 0];

%Creating the SerialLink object

linkL(1)= Link([link\_theta(1) link\_D(1) link\_A(1) link\_Alpha(1) linkType(1)]);

linkL(2)= Link([link\_theta(2) link\_D(2) link\_A(2) link\_Alpha(2) linkType(2)]);

linkL(3)= Link([link\_theta(3) link\_D(3) link\_A(3) link\_Alpha(3) linkType(3)]);

linkL(4)= Link([link\_theta(4) link\_D(4) link\_A(4) link\_Alpha(4) linkType(4)]);

linkL(5)= Link([link\_theta(5) link\_D(5) link\_A(5) link\_Alpha(5) linkType(5)]);

linkL(6)= Link([link\_theta(6) link\_D(6) link\_A(6) link\_Alpha(6) linkType(6)]);

link\_M=SerialLink([linkL(1) linkL(2) linkL(3) linkL(4) linkL(5) linkL(6)]);

link\_M.plot(link\_theta);

%Generating grayscale image

%}

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[ 0 rand() 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

mdl\_puma560;

link\_M=SerialLink(p560);

%{

link\_M.plot([0 0 0 0 0 0]);

hold on

patch(x1,y1,z1,'blue','facealpha',.25)

hold on

patch(x2,y2,z2,'blue','facealpha',.25)

hold on

patch(x3,y3,z3,'red','facealpha',.25)

hold on

patch(x4,y4,z4,'red','facealpha',.25)

hold on

patch(x5,y5,z5,'yellow','facealpha',.25)

hold on

patch(x6,y6,z6,'yellow','facealpha',.25)

hold on

%}

for z=spc\_z1:vxl\_zLayerHeight:spc\_z2

if z==spc\_z2

disp('\*');

else

%Invoking the surface function for at the height "z"

logic=0;

while logic==0

try

[surfPlan,SizeZ] = surfaceRobotTest(linkL,link\_M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

logic=1;

catch

warning('jtraj function error. Generating a new random arm.')

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[ 0 rand() 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

link\_M=SerialLink([l1,l2,l3,l4,l5,l6]);

logic=0;

end

end

[SizeX,SizeY]=size(surfPlan);

columnImage=surfPlan;

performanceVector=vertcat(performanceVector,columnImage);

end

end

inputVector=vertcat(link\_D',link\_A');

inputMatrix=horzcat(inputMatrix,inputVector);

performanceMatrix=horzcat(performanceMatrix,performanceVector);

end

for num=1:numData

rawdata=[];

performanceVector=[];

%{

%SerialLink Parameters

link\_theta=[0 0 0 0 0 0];

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[0 rand() 0 0 0 0];

link\_Alpha=[pi/2 0 -pi/2 pi/2 -pi/2 0];

linkType=[0 0 0 0 0 0];

%Creating the SerialLink object

linkL(1)= Link([link\_theta(1) link\_D(1) link\_A(1) link\_Alpha(1) linkType(1)]);

linkL(2)= Link([link\_theta(2) link\_D(2) link\_A(2) link\_Alpha(2) linkType(2)]);

linkL(3)= Link([link\_theta(3) link\_D(3) link\_A(3) link\_Alpha(3) linkType(3)]);

linkL(4)= Link([link\_theta(4) link\_D(4) link\_A(4) link\_Alpha(4) linkType(4)]);

linkL(5)= Link([link\_theta(5) link\_D(5) link\_A(5) link\_Alpha(5) linkType(5)]);

linkL(6)= Link([link\_theta(6) link\_D(6) link\_A(6) link\_Alpha(6) linkType(6)]);

link\_M=SerialLink([linkL(1) linkL(2) linkL(3) linkL(4) linkL(5) linkL(6)]);

link\_M.plot(link\_theta);

%Generating grayscale image

%}

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[ 0 rand() 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

mdl\_puma560;

link\_M=SerialLink(p560);

%{

link\_M.plot([0 0 0 0 0 0]);

hold on

patch(x1,y1,z1,'blue','facealpha',.25)

hold on

patch(x2,y2,z2,'blue','facealpha',.25)

hold on

patch(x3,y3,z3,'red','facealpha',.25)

hold on

patch(x4,y4,z4,'red','facealpha',.25)

hold on

patch(x5,y5,z5,'yellow','facealpha',.25)

hold on

patch(x6,y6,z6,'yellow','facealpha',.25)

hold on

%}

for z=spc\_z1:vxl\_zLayerHeight:spc\_z2

if z==spc\_z2

disp('\*');

else

%Invoking the surface function for at the height "z"

logic=0;

while logic==0

try

[surfPlan,SizeZ] = surfaceRobotTest(linkL,link\_M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

logic=1;

catch

warning('jtraj function error. Generating a new random arm.')

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[ 0 rand() 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

link\_M=SerialLink([l1,l2,l3,l4,l5,l6]);

logic=0;

end

end

[SizeX,SizeY]=size(surfPlan);

columnImage=surfPlan;

performanceVector=vertcat(performanceVector,columnImage);

end

end

inputVector=vertcat(link\_D',link\_A');

inputMatrix=horzcat(inputMatrix,inputVector);

performanceMatrix=horzcat(performanceMatrix,performanceVector);

end

patch(x2,y2,z2,'blue','facealpha',.25)

hold on

patch(x3,y3,z3,'red','facealpha',.25)

hold on

patch(x4,y4,z4,'red','facealpha',.25)

hold on

patch(x5,y5,z5,'yellow','facealpha',.25)

hold on

patch(x6,y6,z6,'yellow','facealpha',.25)

hold on

%}

for z=spc\_z1:vxl\_zLayerHeight:spc\_z2

if z==spc\_z2

disp('\*');

else

%Invoking the surface function for at the height "z"

logic=0;

while logic==0

try

[surfPlan,SizeZ] = surfaceRobotTest(linkL,link\_M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

logic=1;

catch

warning('jtraj function error. Generating a new random arm.')

link\_D=[ 0 0 rand() rand() 0 0];

link\_A=[ 0 rand() 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

link\_M=SerialLink([l1,l2,l3,l4,l5,l6]);

logic=0;

end

end

[SizeX,SizeY]=size(surfPlan);

columnImage=surfPlan;

performanceVector=vertcat(performanceVector,columnImage);

end

end

inputVector=vertcat(link\_D',link\_A');

inputMatrix=horzcat(inputMatrix,inputVector);

performanceMatrix=horzcat(performanceMatrix,performanceVector);

end

columnImage=surfPlan;

performanceVector=vertcat(performanceVector,columnImage);

end

end

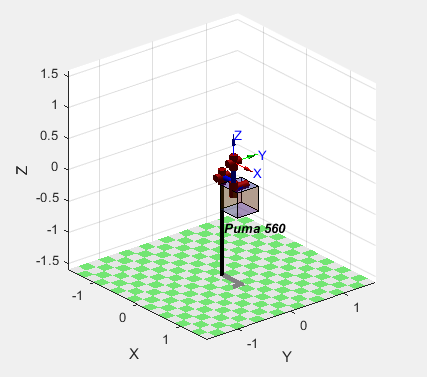
inputVector=vertcat(link\_D',link\_A');

inputMatrix=horzcat(inputMatrix,inputVector);

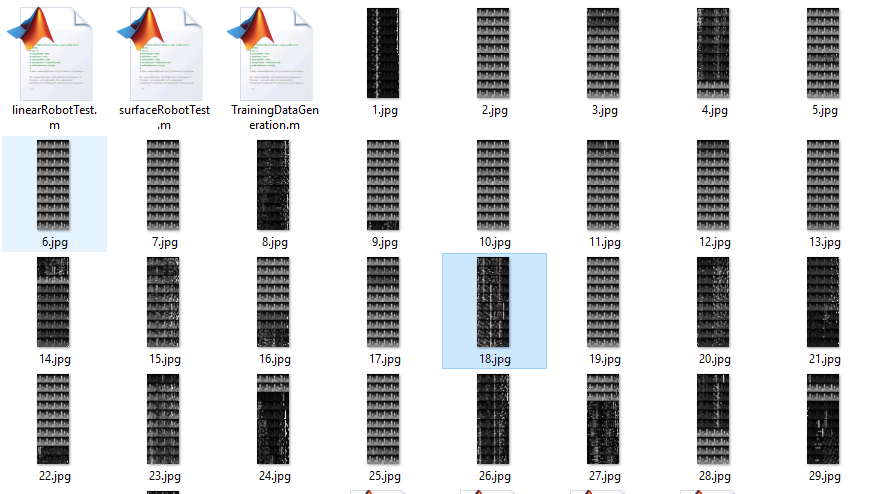
performanceMatrix=horzcat(performanceMatrix,performanceVector);

end

**Visualization 1: the transparent box denotes the boundary of the partitions**



**Sample data generation:**



***Goal: train a neural network that can predict the kinematic structure of a 7-DOF manipulator from the accuracy map***

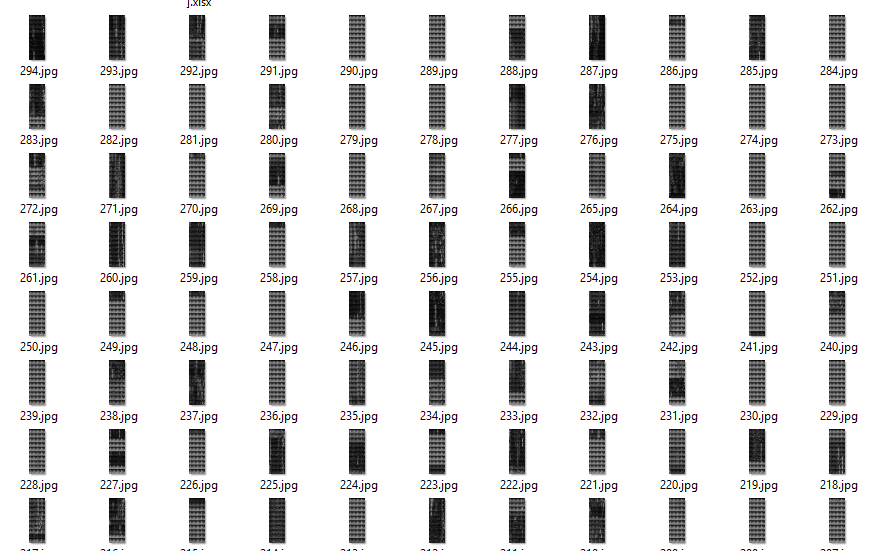
Date: 10/2/17

Alexander Liao

Idea 1

Intention: An input of artificially generated map of high accuracy will help the network to calculate the most suitable manipulator for the given task.

**Training batch of 1000 samples:**



**Rounded length of the first link (first 50 samples):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D3 | D27 | D13 | D16 | D23 |
| D15 | D1 | D5 | D17 | D2 |
| D19 | D10 | D28 | D0 | D14 |
| D10 | D3 | D28 | D13 | D9 |
| D7 | D13 | D30 | D28 | D6 |
| D15 | D29 | D16 | D28 | D11 |
| D9 | D11 | D24 | D6 | D23 |
| D7 | D27 | D8 | D10 | D18 |
| D3 | D12 | D27 | D1 | D2 |
| D7 | D16 | D2 | D22 | D13 |

**DH-parameters of a sample:**

2.533075365

11.99347947

7.796112086

24.00205441

12.94241482

27.31942783

5.455410849

7.914087496

4.366169412

4.082056761

26.07876623

17.39113762

16.49580606

4.348643947

1.570796327

1.570796327

1.570796327

1.570796327

1.570796327

1.570796327

1.570796327

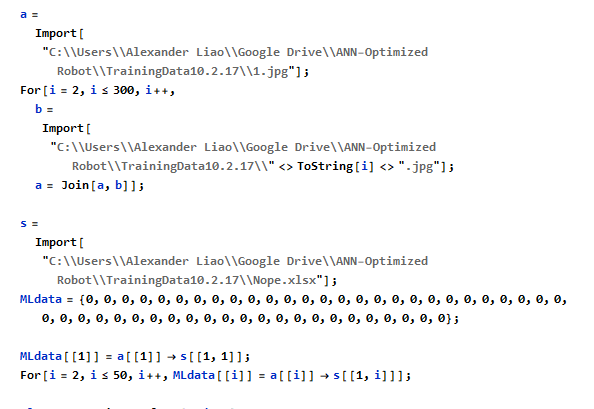
Implementation of a Convolutional Neural Network in Mathematica to predict the rounded length of the first link

Date: 10/4/17

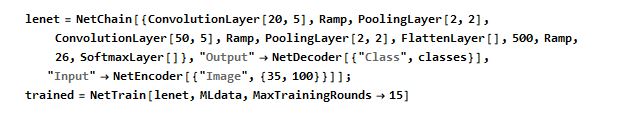
Alexander Liao

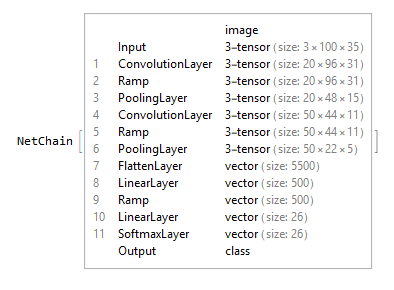
Idea 1

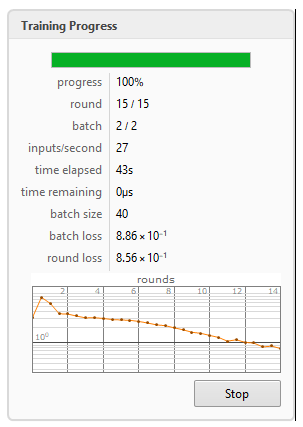
**Pre-processing of 50 samples:**



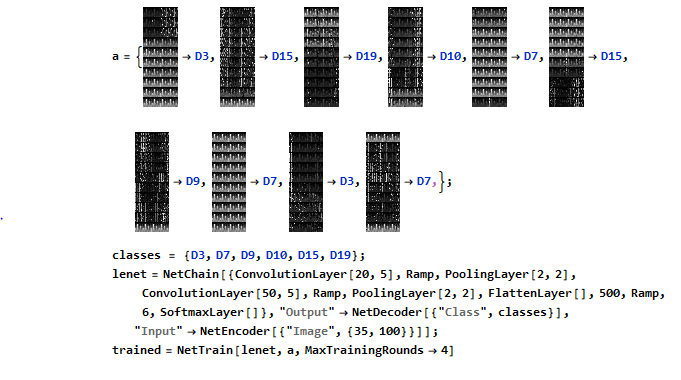
**ConvNet initialization:**



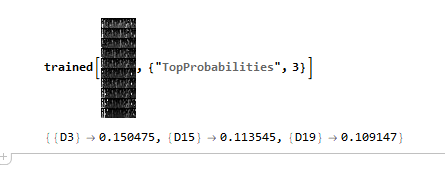


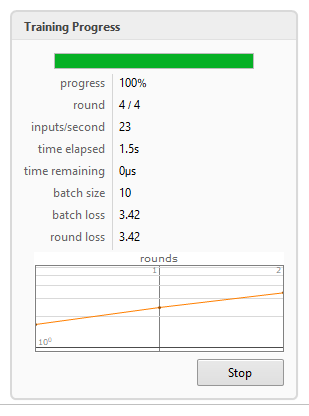


**2nd attempt:**

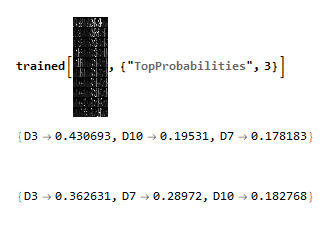


**Prediction failed:**





**Prediction failed even within the training batch (sample size too small):**



Minor modifications on the training data generation code

Date: 10/5/17

Alexander Liao

Idea 1

%Initialization

clear all

numData=1:50;

%Target Space

spc\_x1=10;

spc\_x2=20;

spc\_y1=10;

spc\_y2=20;

spc\_z1=-8;

spc\_z2=2;

%Specifications for Voxels

vxl\_interpVal=4;

vxl\_num=5;

vxl\_zLayerHeight=1;

%Final Result

Final\_dataImage=[];

Final=[];

parfor num= numData

rawdata=[];

Final\_dataImage=[];

%SerialLink Parameters

link\_theta=[0 0 0 0 0 0 0];

link\_D=30\*rand(1,7);

link\_A=30\*rand(1,7);

link\_Alpha=[pi/2,pi/2,pi/2,pi/2,pi/2,pi/2,pi/2];

linkType=[0 0 0 0 0 0 0];

linkNum=[1 2 3 4 5 6 7];

%Creating the SerialLink object

link1= Link([link\_theta(1) link\_D(1) link\_A(1) link\_Alpha(1) linkType(1)],'modified');

link2= Link([link\_theta(2) link\_D(2) link\_A(2) link\_Alpha(2) linkType(2)],'modified');

link3= Link([link\_theta(3) link\_D(3) link\_A(3) link\_Alpha(3) linkType(3)],'modified');

link4= Link([link\_theta(4) link\_D(4) link\_A(4) link\_Alpha(4) linkType(4)],'modified');

link5= Link([link\_theta(5) link\_D(5) link\_A(5) link\_Alpha(5) linkType(5)],'modified');

link6= Link([link\_theta(6) link\_D(6) link\_A(6) link\_Alpha(6) linkType(6)],'modified');

link7= Link([link\_theta(7) link\_D(7) link\_A(7) link\_Alpha(7) linkType(7)],'modified');

link\_M=SerialLink([link1 link2 link3 link4 link5 link6 link7]);

%Generating grayscale image

for z=spc\_z1:vxl\_zLayerHeight:spc\_z2

if z==spc\_z2

disp('\*');

else

%Invoking the surface function for at the height "z"

logic=0;

while logic==0

try

[surfPlan,SizeZ] = surfaceRobotTest(linkNum,link\_M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

logic=1;

catch

warning('jtraj function error. Generating a new random arm.')

%SerialLink Parameters

link\_theta=[0 0 0 0 0 0 0];

link\_D=30\*rand(1,7);

link\_A=30\*rand(1,7);

link\_Alpha=[pi/2,pi/2,pi/2,pi/2,pi/2,pi/2,pi/2];

linkType=[0 0 0 0 0 0 0];

%Creating the SerialLink object

link1= Link([link\_theta(1) link\_D(1) link\_A(1) link\_Alpha(1) linkType(1)],'modified');

link2= Link([link\_theta(2) link\_D(2) link\_A(2) link\_Alpha(2) linkType(2)],'modified');

link3= Link([link\_theta(3) link\_D(3) link\_A(3) link\_Alpha(3) linkType(3)],'modified');

link4= Link([link\_theta(4) link\_D(4) link\_A(4) link\_Alpha(4) linkType(4)],'modified');

link5= Link([link\_theta(5) link\_D(5) link\_A(5) link\_Alpha(5) linkType(5)],'modified');

link6= Link([link\_theta(6) link\_D(6) link\_A(6) link\_Alpha(6) linkType(6)],'modified');

link7= Link([link\_theta(7) link\_D(7) link\_A(7) link\_Alpha(7) linkType(7)],'modified');

link\_M=SerialLink([link1 link2 link3 link4 link5 link6 link7]);

logic=0;

end

end

[SizeX,SizeY]=size(surfPlan);

intm\_endX=zeros(1,SizeX);

intm\_endY=zeros(1,SizeY);

%Calculating the trajectories from joint angle to Cartesian Coordinates

for i=1:size(surfPlan)

hmTrans=link\_M.fkine(surfPlan(i,:));

intm\_vector=hmTrans.t;

intm\_endX(i)=intm\_vector(1);

intm\_endY(i)=intm\_vector(2);

%Optional Visualization: M.plot(surfPlan(i,:))

end

[~,surfaceDataPtsSize]=size(intm\_endY);

intm\_preFitx=[];

intm\_preFity=[];

counter=1;

intm\_residVector=[];

rawImageData=[];

numDataPt=1;

intm\_normal=[];

%Normalizing the trajectories of robot to lines

while numDataPt<=surfaceDataPtsSize

if numDataPt<= (surfaceDataPtsSize/2)

intm\_comparedData=intm\_endY(numDataPt);

intm\_normal=[];

intm\_preFity=[];

for k=1:(vxl\_interpVal-2)\*vxl\_num

if numDataPt<=surfaceDataPtsSize

b=numDataPt+vxl\_interpVal;

intm\_normal=vertcat(intm\_normal,intm\_comparedData);

intm\_preFity=vertcat(intm\_preFity,intm\_endY(numDataPt));

numDataPt=numDataPt+1;

end

end

try

intm\_residVector=abs(intm\_normal-intm\_preFity);

rawImageData=horzcat(rawImageData,intm\_residVector);

end

end

if numDataPt> (surfaceDataPtsSize/2)

intm\_comparedData=intm\_endX(numDataPt);

intm\_normal=[];

intm\_preFitx=[];

for k=1:(vxl\_interpVal-2)\*vxl\_num

if numDataPt<=surfaceDataPtsSize

b=numDataPt+vxl\_interpVal;

intm\_normal=vertcat(intm\_normal,intm\_comparedData);

intm\_preFitx=vertcat(intm\_preFitx,intm\_endX(numDataPt));

numDataPt=numDataPt+1;

end

end

try

intm\_residVector=abs(intm\_normal-intm\_preFitx);

rawImageData=horzcat(rawImageData,intm\_residVector);

end

end

end

rowImage=rawImageData;

Final\_dataImage=vertcat(Final\_dataImage,rowImage);

end

end

Final=vertcat(Final,mean(Final\_dataImage));

%imageInfo=strcat(int2str(num),'.jpg');

%imwrite(Final\_dataImage,imageInfo);

rawdata=vertcat(link\_D',link\_A');

[sizeRaw,~]=size(rawdata);

filename='Data.xlsx';

startData=sizeRaw\*(num-1)+1;

endData=sizeRaw\*num;

dataOption=strcat('A',int2str(startData),':A',int2str(endData));

xlswrite(filename,rawdata,dataOption);

end

Formulation of a research topic, proposal to the pre-engineering department

A Method to Evaluate the Manipulability of a 6-DOF Manipulator and its Application in Artificial Neural Network

Date: 10/15/17

Alexander Liao

Idea 1

Alexander Liao

I. Introduction

* The determination of the length of a manipulator is often based on the torque of the motor
* Yet the manipulability of the arm is less taken care of when engineers determine the length of each link
* Over-designing is a common practice
* The purpose of this method is to shift the attention from the torque to the effect of lengths of links themselves on the dexterity of the robot.

\*refer to code in 10/5/17\*

II. Method

* Central idea:
  + divide the predetermined space in to voxels with equal volume
  + find a data point to represent the voxel
  + obtain the joint configuration of the robot in order to reach the point
  + calculate the manipulability
  + store all results in a vector
* Prevalent methods of a manipulator’s manipulability at a given joint configuration include Yoshikawa’s manipulability measure
* Levenberg–Marquardt algorithm is implemented to solve the inverse kinematics equation
* Peter Corke’s Robotics Toolbox in MATLAB provides great help on the calculation
  + Generate 6-DOF manipulator with random d/a parameters (from 1 to 10) with modified D-H convention
  + Generate matrix for voxels
  + Generate matrix for trajectory planning (joint configurations)
  + Calculate the manipulability index for each configuration
  + Store results

IV. Applications

* Since both the input and the output are vectors, they can be fed into an artificial neural network to look for an optimized result
* However, this unconventional approach might require a modified network structure

V. Possible Defects

* There are errors in the solutions of inverse kinematics function
* *I’m not sure if the manipulability of arm at a configuration obtained by approximation is still significant*

Warning: ikine: rejected-step limit 100 exceeded (pose 1), final err 0.300062

> In SerialLink/ikine (line 244)

In SerialLink>@(varargin)r.ikine(varargin{:}) (line 846)

In SerialLink/jtraj (line 851)

In linearRobotTest (line 10)

In surfaceRobotTest (line 9)

In TrainingDataGeneration (line 59)

Finished annotation of *Robotics, Vision and Control* by Peter Corke

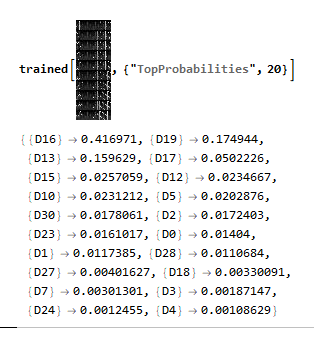
Date: 10/20/17

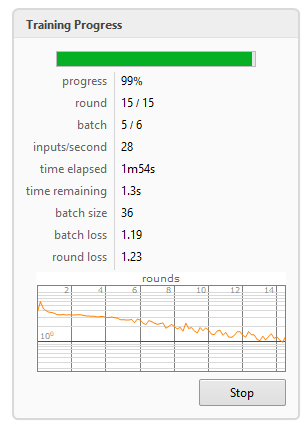
Alexander Liao

Idea 1

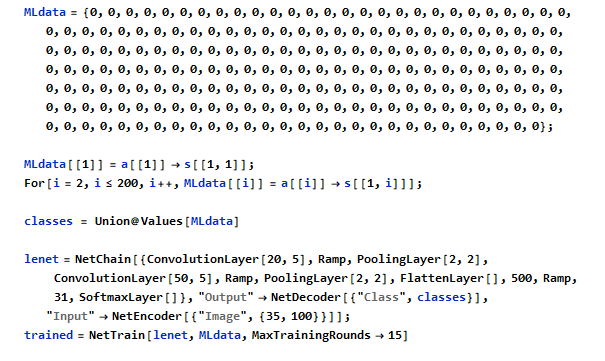
Improved version of ConvNet in Mathematica

**Successful prediction of member within the training batch**





**Initialization with 200 samples**

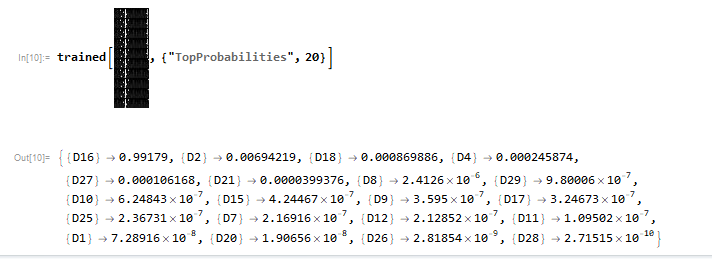


Date: 10/21/17

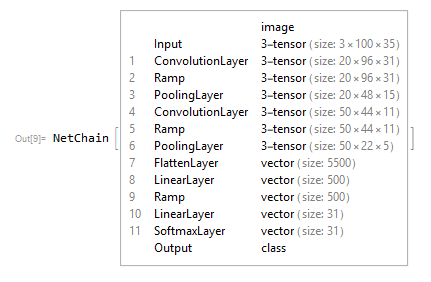
Alexander Liao

Idea 1

**Failed prediction of unseen sample:**



**ConvNet architecture**



Proposal 2nd draft:

Date: 10/22/17

Alexander Liao

Idea 1

A Method to Evaluate a Rigid-Link Manipulator and its Application in Artificial Neural Network

Alexander Liao

I. Introduction

* The determination of the length of a manipulator is often based on the torque of the motor
* Yet the manipulability of the arm is less taken care of when engineers determine the length of each link
* Over-designing is a common practice
* The purpose of this method is to shift the attention from the torque to the effect of lengths of links themselves on the performance of the robot.

II. Method

* Based on modified Denavit - Hartenberg convention, a specific vector can be created to define a specific rigid arm-type robot (assuming that all joints are revolute):

, where is the degree of freedom of the robot.

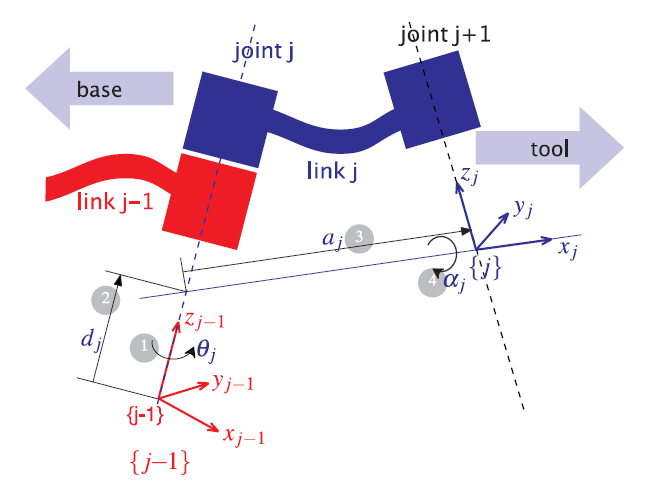
* The denotation of symbols in the vector can be found in Figure 1-1 (Corke, 2013, p. 138).

Figure -1

* The performance of robot in a given space is defined by its performance in every one of the voxels:
  + divide the predetermined space into voxels of equal volume
  + find a data point to represent the voxel
* Components of the test (For a 6 degrees-of-freedom robot):
  + Distance between the actual point of end-effector and the ideal point of the voxel based on the numerical approach of the inverse kinematics of the robot.
  + Manipulability of the robot in terms of Yoshikawa’s manipulability measure (a scalar) at different joint configurations
  + Joint speed at performing linear motions between voxels (x-direction and y-direction)
* Components of the test (For an under-actuated robot):
  + Whether the robot can reach the voxel
  + Manipulability of the robot for the voxels that are approachable.
* Evaluation primarily based on functions from Peter Corke’s Robotics Toolbox in MATLAB
  + Generate manipulator with a random (in an acceptable range) with modified D-H convention
  + Generate voxels that are stored in a matrix
  + Evaluate the performance of the robot that is then stored in a vector.
  + Output the two vectors in the form of a pair

III. Applications

* Although it has been proven mathematically that a 6-DOF manipulator is possible to reach every single position in the task-space, other factors like link collision and singularities could still create awkward situations (Corke, 2013, p. 150). So obviously some are better at performing tasks in a given space than others.
* For under-actuated rigid-link manipulators, it is useful to find out if a exists for a given space
* Since both the input and the output are vectors, they can be fed into an artificial neural network to look for a pattern, which means that an optimized result could be obtained.
* However, this unconventional approach might require a modified network structure.

References

Corke, P. I. (2013). *Robotics, vision and control: Fundamental algorithms in MATLAB*.

Exploring the possibility of using quaternions to represent poses

Date: 10/24/17

Alexander Liao

Idea 1

classdef VectorQuaternion

properties

t

q

end

methods

function r = VectorQuaternion(inT,inQ)

if size(inT)== [3 1]

r.t = inT;

else

error('t has to be a 3\*1 vector');

end

try

Quaternion(inQ);

catch

error('q has to be a unit Quaternion');

end

if inQ.norm==1

r.q= inQ;

else

error('q has to be a unit Quaternion');

end

end

function r = poseCompose(inVQ1,inVQ2)

if and(isQuaternion(inVQ1.q),isQuaternion(inVQ2.q))

r = VectorQuaternion([0;0;0],UnitQuaternion);

r.q=(inVQ1.q\*inVQ2.q);

q1 = inVQ1.q;

r.t=(inVQ1.t+(q1.R)\*inVQ2.t);

else

error('Only quaternion inputs are allowed');

end

end

function r = poseNegate(inVQ)

if isQuaternion(inVQ.q)

r = VectorQuaternion([0;0;0],UnitQuaternion);

q0= inVQ.q;

q\_inv= q0.inv;

t0= inVQ.t;

r.t=-(q\_inv.R)\*t0;

r.q=q\_inv;

else

error('Only quaternion input is allowed');

end

end

function r = VQtrans(inPt,inVQ)

if and(isQuaternion(inVQ.q),size(inPt)== [3 1])

q0= inVQ.q;

t0= inVQ.t;

r=-(q0.R)\*inPt+t0;

else

error('inputs must have the following form: desired point, quaternion');

end

end

end

end

function logic = isQuaternion(inQ)

try

determine = Quaternion(inQ)

logic = true();

catch

logic = false();

end

end

end

function r = poseNegate(inVQ)

if isQuaternion(inVQ.q)

r = VectorQuaternion([0;0;0],UnitQuaternion);

q0= inVQ.q;

q\_inv= q0.inv;

t0= inVQ.t;

r.t=-(q\_inv.R)\*t0;

r.q=q\_inv;

else

error('Only quaternion input is allowed');

end

end

function r = VQtrans(inPt,inVQ)

if and(isQuaternion(inVQ.q),size(inPt)== [3 1])

q0= inVQ.q;

t0= inVQ.t;

r=-(q0.R)\*inPt+t0;

else

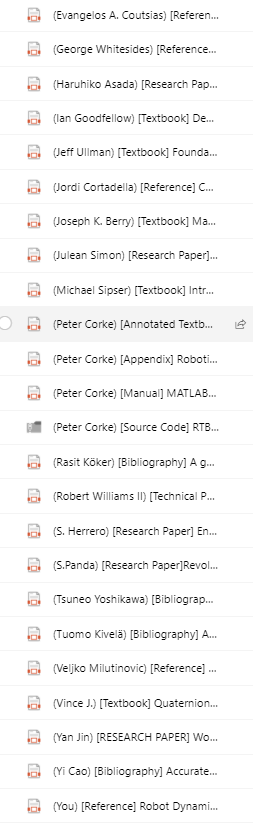
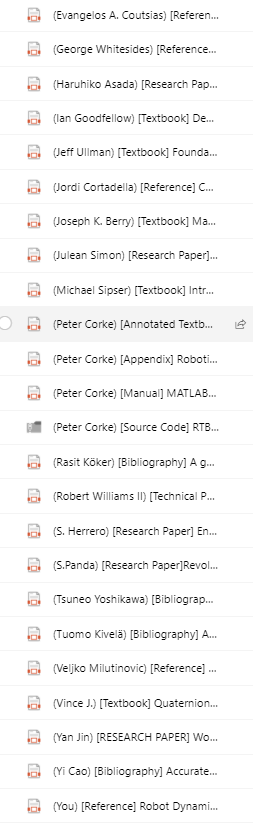
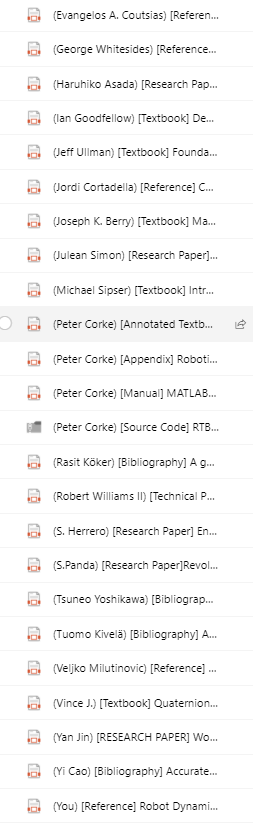
error('inputs must have the following form: desired point, quaternion');

end

end

end

end

Quaternion idea failed; starting literature research on web

Date: 10/25/17

Alexander Liao

Idea 1

***Goal: Train a neural network that can predict the manipulability map of manipulator given its kinematic structure***

Date: 10/29/17

Alexander Liao

Idea 2

First attempt of creating a script to generate random 6-DOF manipulators

**Code:**

D=10\*round(rand(1,6),3);

A=10\*round(rand(1,6),3);

alpha=zeros(1,6);

r1=rand(1,6)\*10;

%r2=rand(1,5)\*10;

r3=rand(1,6)\*10;

logic = 0;

for i = 1:5

if and(r1(i)>3, logic==0)

D(i)=0;

A(i)=0;

D(i+1)=0;

A(i+1)=0;

alpha(i)=0;

logic=1;

end

%if 6>r3(i)>3

% alpha(i)=pi/2;

%elseif r3(i)>6

% alpha(i)=-pi/2;

%end

end

lt=[0 0 0 0 0 0];

%lD=50\*round(rand(1,5),3);

%lA=50\*round(rand(1,5),3);

%lD(round(rand(1)\*5))=0;

%la=pi+pi/2\*-(rand(1,5));

%Creating the SerialLink object

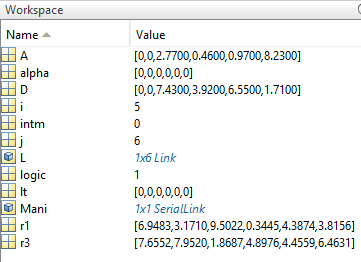
for j = 1:6

L(j)=Link([0 D(j) A(j) alpha(j)]);

end

intm = 0;

Mani=SerialLink(L);



Further improvements

Date: 10/30/17

Alexander Liao

Idea 2

**Scalar field of sum of manipulability measures in the trajectory** (transl(0.5,0,0),transl(0.5,0.5,0),10) **of a 6-DOF manipulator with random link lengths:**

times=5000;

answer=zeros(1,times);

error=answer;

matrix = zeros(18,times);

L=struct();

parfor me = 1:times

D=[ 0 0 0.2+0.5\*rand() 0.2+0.5\*rand() 0 0];

A=[ 0 0.2+0.5\*rand() 0 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

%{

D=[ 0 0 0 rand() rand() 0 ];

A=[ 0 0 rand() 0 0 0 ];

alpha=[ pi/2 pi/2 0 -pi/2 pi/2 -pi/2 ];

for j = 1:6

L(j)=Link([0 D(j) A(j) alpha(j)]);

end

%}

l1=Link([0 D(1) A(1) alpha(1)]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)]);

l4=Link([0 D(4) A(4) alpha(4)]);

l5=Link([0 D(5) A(5) alpha(5)]);

l6=Link([0 D(6) A(6) alpha(6)]);

Mn=SerialLink([l1,l2,l3,l4,l5,l6]);

ger=[0 0 0 0 0 0];

intm=0;

%step=0:pi/4:pi/2

traj= ctraj(transl(0.5,0,0),transl(0.5,0.5,0),10);

traj1=traj(:,:,1);

qMn=zeros(10,6);

logic=1;

logic2=1;

ger=Mn.ikine(traj1,'rlimit',500,'ilimit',500);

if sum(ger)==0

answer(me)=0;

logic=0;

end

if logic==1

qMn(1,:)=ger;

for i = 2:10

ger=Mn.ikine(traj(:,:,i),'rlimit',200,'q0',qMn(i-1,:));

if sum(ger)==0

logic2=0;

else

qMn(i,:)=ger;

end

end

if logic2==1

in=Mn.maniplty(qMn);

intm=sum(in);

%intm = M.maniplty([0 pi/4 pi/4 0 pi/4 pi/4]);

answer(me)=intm;

%error(me)=sum(err1);

end

end

matrix(:,me)=[D,A,alpha];

end

gee=zeros(1,6);

for t = 1:times

if answer(t)==max(answer)

gee=matrix(:,t);

end

end

nD=gee(1:6,1);

nA=gee(7:12,1);

nAlpha=gee(13:18,1);

for nj = 1:6

nL(nj)=Link([0 nD(nj) nA(nj) nAlpha(nj)]);

end

nM=SerialLink(nL);

q0=[0 0 0 0 0 0];

%nM.plot(q0);

%nM.teach();

scatter3(matrix(3,:),matrix(4,:),matrix(8,:),7.5,answer,'filled')

D=[0 0 0 0.15 0.4318 0];

A=[0 0 0.4318 0.0203 0 0];

alpha=[pi/2 0 -pi/2 pi/2 -pi/2 0];

for j = 1:6

L(j)=Link([0 D(j) A(j) alpha(j)]);

end

intm = 0;

M=SerialLink(L);

intm=0;

%step=0:pi/4:pi/2

traj= ctraj(transl(0.5,0,0),transl(0.5,0.5,0),10);

traj1=traj(:,:,1);

qMn=zeros(10,6);

logic=1;

logic2=1;

ger=Mn.ikine(traj1,'rlimit',500,'ilimit',500);

if sum(ger)==0

answer(me)=0;

logic=0;

end

if logic==1

qMn(1,:)=ger;

for i = 2:10

ger=Mn.ikine(traj(:,:,i),'rlimit',200,'q0',qMn(i-1,:));

if sum(ger)==0

logic2=0;

else

qMn(i,:)=ger;

end

end

if logic2==1

in=Mn.maniplty(qMn);

intm=sum(in);

%intm = M.maniplty([0 pi/4 pi/4 0 pi/4 pi/4]);

answer(me)=intm;

%error(me)=sum(err1);

end

end

matrix(:,me)=[D,A,alpha];

end

gee=zeros(1,6);

for t = 1:times

if answer(t)==max(answer)

gee=matrix(:,t);

end

end

nD=gee(1:6,1);

nA=gee(7:12,1);

nAlpha=gee(13:18,1);

for nj = 1:6

nL(nj)=Link([0 nD(nj) nA(nj) nAlpha(nj)]);

end

nM=SerialLink(nL);

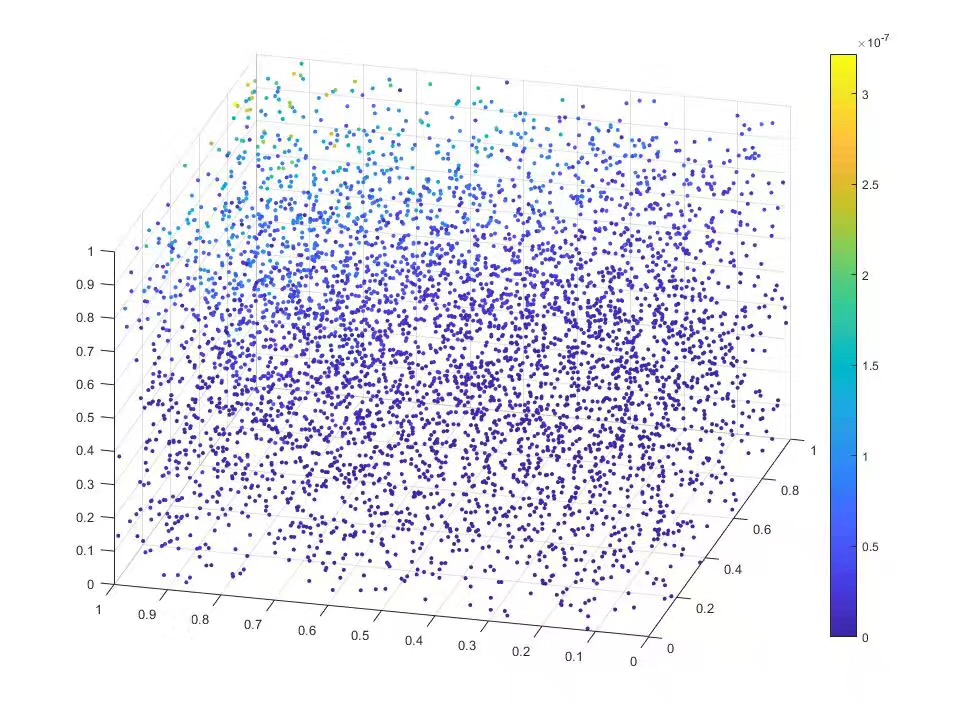
q0=[0 0 0 0 0 0];

%nM.plot(q0);

%nM.teach();

scatter3(matrix(3,:),matrix(4,:),matrix(8,:),7.5,answer,'filled')

**Visualization**



Manipulability measures of a given manipulator in vector form

Date: 10/31/17

Alexander Liao

Idea 2

**I/O:**

in =

1.0e-06 \*

0

0.1001

0

0.0954

0.0126

0

0.0245

0.0925

0.0873

0.1195

>>

**Code:**

D=[ 0 0 0.4793 1.3315 0 0];

A=[ 0 1.1765 0 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 D(1) A(1) alpha(1)]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)]);

l4=Link([0 D(4) A(4) alpha(4)]);

l5=Link([0 D(5) A(5) alpha(5)]);

l6=Link([0 D(6) A(6) alpha(6)]);

Mn=SerialLink([l1,l2,l3,l4,l5,l6]);

intm=0;

traj= ctraj(transl(0.5,0,0),transl(0.5,1,0),10);

qMn=Mn.ikine6s(traj);

in=Mn.maniplty(qMn)

intm=sum(in);

%intm = M.maniplty([0 pi/4 pi/4 0 pi/4 pi/4]);

**Visualization algorithm of the scope of the manipulability map:**

D=[ 0 0 rand() rand() 0 0];

A=[ 0 rand() 0 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 D(1) A(1) alpha(1)]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)]);

l4=Link([0 D(4) A(4) alpha(4)]);

l5=Link([0 D(5) A(5) alpha(5)]);

l6=Link([0 D(6) A(6) alpha(6)]);

Mn=SerialLink([l1,l2,l3,l4,l5,l6]);

xmin=0.2;

xmax=0.6;

ymin=-0.2;

ymax=0.2;

zmin=-0.4;

zmax=0;

q1=Mn.ikine6s(transl(xmin,ymin,zmin));

q2=Mn.ikine6s(transl(xmin,ymax,zmin));

q3=Mn.ikine6s(transl(xmax,xmin,zmin));

q4=Mn.ikine6s(transl(xmax,ymax,zmin));

q5=Mn.ikine6s(transl(xmin,ymin,zmax));

q6=Mn.ikine6s(transl(xmin,ymax,zmax));

q7=Mn.ikine6s(transl(xmax,ymin,zmax));

q8=Mn.ikine6s(transl(xmax,ymax,zmax));

mat=[q1;q2;q3;q4;q5;q6;q7;q8];

difmat=zeros(2,6);

for i=1:6

difmat(1,i)=min(mat(:,i));

difmat(2,i)=max(mat(:,i));

end

x1=[xmin xmin xmax xmax];

y1=[ymin ymax ymax ymin];

z1=[zmin zmin zmin zmin];

x2=[xmin xmin xmax xmax];

y2=[ymin ymax ymax ymin];

z2=[zmax zmax zmax zmax];

x3=[xmin xmin xmin xmin];

y3=[ymin ymax ymax ymin];

z3=[zmax zmax zmin zmin];

x4=[xmax xmax xmax xmax];

y4=[ymin ymax ymax ymin];

z4=[zmax zmax zmin zmin];

x5=[xmin xmax xmax xmin];

y5=[ymin ymin ymin ymin];

z5=[zmax zmax zmin zmin];

x6=[xmin xmax xmax xmin];

y6=[ymax ymax ymax ymax];

z6=[zmax zmax zmin zmin];

Mn.plot(qn);

hold on

patch(x1,y1,z1,'blue','facealpha',.25)

hold on

patch(x2,y2,z2,'blue','facealpha',.25)

hold on

patch(x3,y3,z3,'red','facealpha',.25)

hold on

patch(x4,y4,z4,'red','facealpha',.25)

hold on

patch(x5,y5,z5,'yellow','facealpha',.25)

hold on

patch(x6,y6,z6,'yellow','facealpha',.25)

hold on

diff=(difmat(2,:)-difmat(1,:))/20

x2=[xmin xmin xmax xmax];

y2=[ymin ymax ymax ymin];

z2=[zmax zmax zmax zmax];

x3=[xmin xmin xmin xmin];

y3=[ymin ymax ymax ymin];

z3=[zmax zmax zmin zmin];

x4=[xmax xmax xmax xmax];

y4=[ymin ymax ymax ymin];

z4=[zmax zmax zmin zmin];

x5=[xmin xmax xmax xmin];

y5=[ymin ymin ymin ymin];

z5=[zmax zmax zmin zmin];

x6=[xmin xmax xmax xmin];

y6=[ymax ymax ymax ymax];

z6=[zmax zmax zmin zmin];

Mn.plot(qn);

hold on

patch(x1,y1,z1,'blue','facealpha',.25)

hold on

patch(x2,y2,z2,'blue','facealpha',.25)

hold on

patch(x3,y3,z3,'red','facealpha',.25)

hold on

patch(x4,y4,z4,'red','facealpha',.25)

hold on

patch(x5,y5,z5,'yellow','facealpha',.25)

hold on

patch(x6,y6,z6,'yellow','facealpha',.25)

hold on

diff=(difmat(2,:)-difmat(1,:))/20

Improvements

Date: 11/1/17

Alexander Liao

Idea 2

**Scalar map algorithm version 2:**

times=100;

answer=zeros(1,times);

error=answer;

matrix = zeros(18,times);

parfor me =1:times

D=[ 0 0 rand() rand() 0 0];

A=[ 0 rand() 0 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 D(1) A(1) alpha(1)],'m',0,'l',[0.066,0,0;0,0.086,0;0,0,0.0125],[-0.0203,-0.0141,0.07]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)],'m',A(2)\*40);

l4=Link([0 D(4) A(4) alpha(4)],'m',D(3)\*40);

l5=Link([0 D(5) A(5) alpha(5)],'m',D(4)\*20);

l6=Link([0 D(6) A(6) alpha(6)]);

Mn=SerialLink([l1,l2,l3,l4,l5,l6]);

qtraj=ctraj(transl(0.5,0,0),transl(0.5,0.5,0),25);

traj=Mn.ikine6s(qtraj);

vec=Mn.maniplty(traj,'asada');

intm=sum(vec);

answer(me)=intm;

matrix(:,me)=[D,A,alpha];

end

gee=zeros(1,6);

for t = 1:times

if answer(t)==max(answer)

gee=matrix(:,t);

end

end

nD=gee(1:6,1);

nA=gee(7:12,1);

nAlpha=gee(13:18,1);

for nj = 1:6

nL(nj)=Link([0 nD(nj) nA(nj) nAlpha(nj)]);

end

nM=SerialLink(nL);

q0=[0 0 0 0 0 0];

%nM.plot(q0);

%nM.teach();

scatter3(matrix(3,:),matrix(4,:),matrix(8,:),7.5,answer,'filled')

for nj = 1:6

nL(nj)=Link([0 nD(nj) nA(nj) nAlpha(nj)]);

end

nM=SerialLink(nL);

q0=[0 0 0 0 0 0];

%nM.plot(q0);

%nM.teach();

scatter3(matrix(3,:),matrix(4,:),matrix(8,:),7.5,answer,'filled')

**Experimenting with kinematics structure of 6-DOF manipulators**

D=[ 0 0 14.7313 14.7713 0 0];

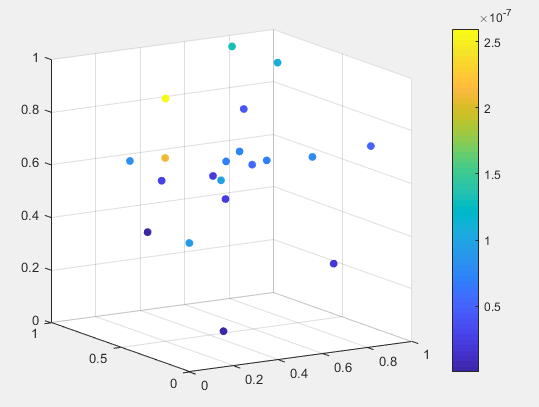
A=[ 14.7029 0 0 0 0 0];

alpha=[ pi/2 pi/2 -pi/2 pi/2 -pi/2 0];

D=[ 0 14.7313 0 14.7713 0 0];

A=[ 14.7029 0 0 0 0 0];

alpha=[ pi/2 -pi/2 pi/2 pi/2 -pi/2 0];



Proposal 3rd draft

Date: 11/2/17

Alexander Liao

Idea 2

A Portable Algorithm for Kinematic Evaluation of Robotic Manipulators Generated by Neural Network and Its Application in Optimization

Alexander Liao

Introduction

* Goal: design a test that could reflect properties of a manipulator’s kinematics model and mimic it by a neural network.
* Neural Networks are often used in research for robot control system (since 2001)
* Pure kinematics model of a serial-link manipulator is less taken care of compared to the dynamics model
* Yet the kinematic model of robot itself is significant as well-idealize other factors:
* *The performance of the manipulator is a function of the parameters that define the kinematics model*
* Yet traditional optimization includes arduous work in multivariable calculus (an optimization attempt of a specific redundant 8-DOF manipulator was done by NASA)
* Thus, it is reasonable to first find a way to describe the model, then use ANNs to mimic it
* Advantages of neural network: the process of LSTM/CNN/RNN/MLP is faster and more portable than analytical Jacobian/tensor arithmetic

Method

* Prerequisite:
  + Rigid-body
  + Serial-link
  + All-revolute
  + Standard D-H parameters
  + Functions based on MATLAB Robotics Toolbox (RTB) by professor Peter Corke
  + Neural network code based on Tensorflow
* 1.Designing the evaluation algorithm
  + 6-DOF with Spherical Wrist
    - Only three configurations are possible:

|  |  |  |  |
| --- | --- | --- | --- |
| j | d | a | alpha |
| 1 | 0 | 0 |  |
| 2 | 0 |  | 0 |
| 3 |  | 0 |  |
| 4 |  | 0 |  |
| 5 | 0 | 0 |  |
| 6 | 0 | 0 | 0 | |

|  |  |  |  |
| --- | --- | --- | --- |
| j | d | a | alpha |
| 1 | 0 |  |  |
| 2 | 0 | 0 |  |
| 3 |  | 0 |  |
| 4 |  | 0 |  |
| 5 | 0 | 0 |  |
| 6 | 0 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| j | d | a | alpha |
| 1 | 0 |  |  |
| 2 |  | 0 |  |
| 3 | 0 | 0 |  |
| 4 |  | 0 |  |
| 5 | 0 | 0 |  |
| 6 | 0 | 0 | 0 |

* + - Input：
      * d-vector
      * a-vector
      * alpha-vector
    - Target space for evaluation:
    - Evaluation
      * Reachability
      * Manipulability
      * Joint-Space Usage
      * Linearity
* 2.Using Neural Networks to mimic the test
* 3.Optimization

**The orientation of the manipulator end-effector when performing Cartesian linear movement between nodes in the cuboid partitions:**

x1=[xmin xmin xmax xmax];

y1=[ymin ymax ymax ymin];

z1=[zmin zmin zmin zmin];

x2=[xmin xmin xmax xmax];

y2=[ymin ymax ymax ymin];

z2=[zmax zmax zmax zmax];

x3=[xmin xmin xmin xmin];

y3=[ymin ymax ymax ymin];

z3=[zmax zmax zmin zmin];

x4=[xmax xmax xmax xmax];

y4=[ymin ymax ymax ymin];

z4=[zmax zmax zmin zmin];

x5=[xmin xmax xmax xmin];

y5=[ymin ymin ymin ymin];

z5=[zmax zmax zmin zmin];

x6=[xmin xmax xmax xmin];

y6=[ymax ymax ymax ymax];

z6=[zmax zmax zmin zmin];

**Initialization:**

%Initialization

clear all

numData=100;

%Target Space

spc\_x1=0.2;

xmin=spc\_x1;

spc\_x2=0.6;

xmax=spc\_x2;

spc\_y1=-0.2;

ymin=spc\_y1;

spc\_y2=0.2;

ymax=spc\_y2;

spc\_z1=-0.4;

zmin=spc\_z1;

spc\_z2=0;

zmax=spc\_z2;

%Specifications for Voxels

vxl\_interpVal=10;

vxl\_num=10;

vxl\_zLayerHeight=0.025;

%Final Result

performanceVector=[];

performanceMatrix=[];

inputMatrix=[];

for num=1:numData

rawdata=[];

performanceVector=[];

link\_D=[ 0 0 0.2290 0.9133 0 0];

link\_A=[ 0 0.1524 0 0 0 0];

link\_alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 link\_D(1) link\_A(1) link\_alpha(1)]);

l2=Link([0 link\_D(2) link\_A(2) link\_alpha(2)]);

l3=Link([0 link\_D(3) link\_A(3) link\_alpha(3)]);

l4=Link([0 link\_D(4) link\_A(4) link\_alpha(4)]);

l5=Link([0 link\_D(5) link\_A(5) link\_alpha(5)]);

l6=Link([0 link\_D(6) link\_A(6) link\_alpha(6)]);

linkL=[l1,l2,l3,l4,l5,l6];

mdl\_puma560;

link\_M=SerialLink(p560);

for z=spc\_z1:vxl\_zLayerHeight:spc\_z2

if z==spc\_z2

disp('\*');

else

%Invoking the surface function for at the height "z"

logic=0;

while logic==0

[surfPlan,SizeZ] = surfaceRobotTest(linkL,link\_M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

logic=1;

end

[SizeX,SizeY]=size(surfPlan);

columnImage=surfPlan;

performanceVector=vertcat(performanceVector,columnImage);

end

end

inputVector=vertcat(link\_D',link\_A');

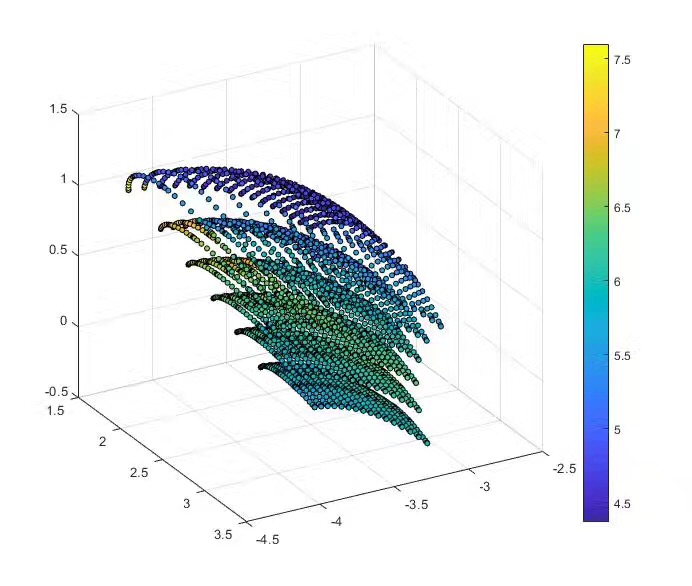
inputMatrix=horzcat(inputMatrix,inputVector);

performanceMatrix=horzcat(performanceMatrix,performanceVector);

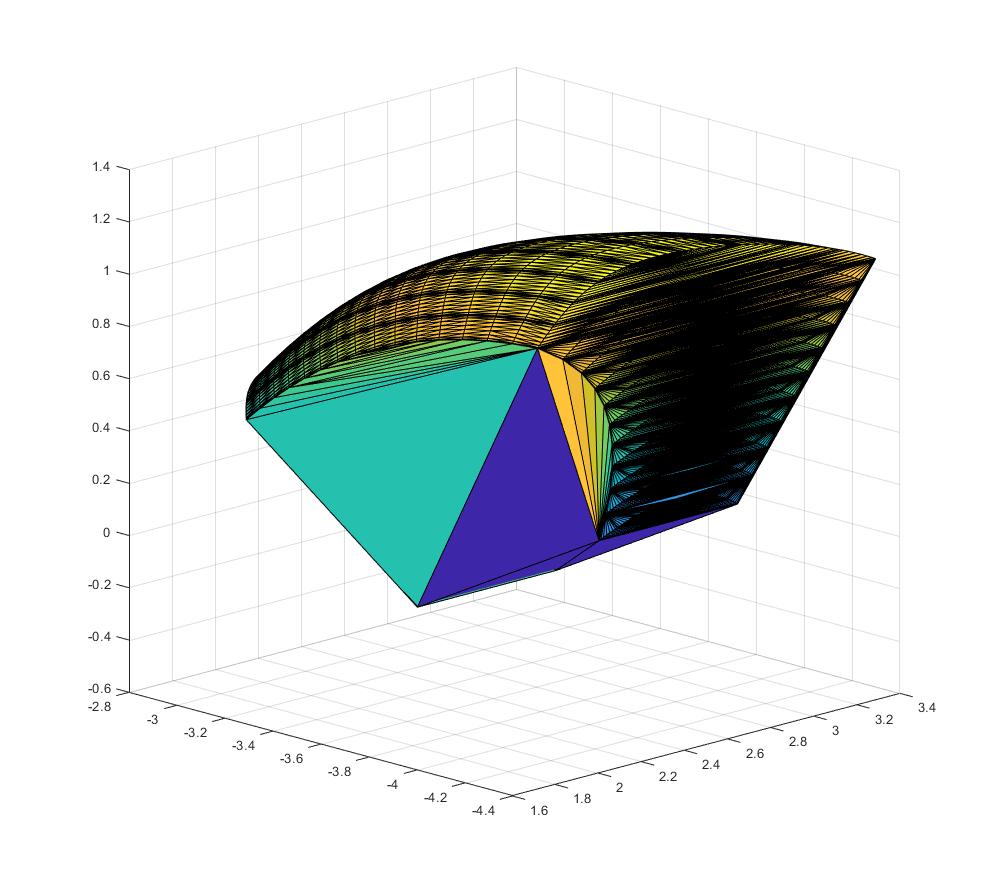
end

scatter3(performanceMatrix(:,1),performanceMatrix(:,2),performanceMatrix(:,3),15,(abs(performanceMatrix(:,4))+abs(performanceMatrix(:,5))+abs(performanceMatrix(:,6))),'filled','MarkerEdgeColor','k')

**Visualization for PUMA 560:**



**Convex hull:**



Further investigation of using convex hull volume to evaluate a manipulator

Date: 11/3/17

Alexander Liao

Idea 2

Attempt to use forward kinematics with uniform sampling of joint angles to map between joint space and workspace, failed due to the unachievable requirement on computational power

***Goal: Using neural network to predict the workspace of manipulator in the form of a binary map***

D=[ 0 0 0.4793 1.4315 0 0];

A=[ 0 1.0765 0.05 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 D(1) A(1) alpha(1)]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)]);

l4=Link([0 D(4) A(4) alpha(4)]);

l5=Link([0 D(5) A(5) alpha(5)]);

l6=Link([0 D(6) A(6) alpha(6)]);

Mn=SerialLink([l1,l2,l3,l4,l5,l6]);

final=[];

parfor i=1:62

i2=-pi+i\*0.1

for j=-pi/2:0.1:pi/2

for k = -pi/2:0.1:pi/2

intm=Mn.fkine([i2 j k 0 0 0]);

final=horzcat(final,intm.t);

end

end

end

%try

%[~,vol]=convhull(performanceVector(:,1),performanceVector(:,2),performanceVector(:,3));

%hullV=horzcat(hullV,vol);

%catch

% hullV=horzcat(hullV,0);

%end

Date: 11/4/17

Alexander Liao

Idea 3 (final)

Attempt to reimplement Castelli’s method in MATLAB

%Initialization

clear all

numData=500;

%Target Space

xmin=0.2;

xmax=0.6;

ymin=-0.2;

ymax=0.2;

zmin=-0.4;

zmax=0;

%Specifications for Voxels

itpVal=4;

%Final Result

perfmat=[];

performanceMatrix=[];

inputMatrix=[];

coordinateData=[];

parfor num=1:numData

%for num=1:numData

rawdata=[];

perfmat=[];

surfPlan=[];

intm=[];

%D=[ 0 0 round(rand(),4) round(rand(),4) 0 0];

%A=[ 0 round(rand(),4) round(rand(),3)/10 0 0 0];

D=[0 0 rand() rand() 0 0];

A=[0 rand() rand()/10 0 0 0];

alpha=[ pi/2 0 -pi/2 pi/2 -pi/2 0];

l1=Link([0 D(1) A(1) alpha(1)]);

l2=Link([0 D(2) A(2) alpha(2)]);

l3=Link([0 D(3) A(3) alpha(3)]);

l4=Link([0 D(4) A(4) alpha(4)]);

l5=Link([0 D(5) A(5) alpha(5)]);

l6=Link([0 D(6) A(6) alpha(6)]);

linkL=[l1 l2 l3 l4 l5 l6];

M=SerialLink([l1,l2,l3,l4,l5,l6]);

for z=zmin:(zmax-zmin)/itpVal:zmax

%if z==spc\_z2

% disp('\*');

%else

%Invoking the surface function for at the height "z"

%logic=0;

%while logic==0

% [surfPlan,SizeZ] = surfaceRobotTest(linkL,M,vxl\_interpVal,vxl\_num,spc\_x1,spc\_x2,spc\_y1,spc\_y2,z);

%logic=1;

surfPlan=[];

for i=xmin:(xmax-xmin)/itpVal:xmax

for j=ymin:(ymax-ymin)/itpVal:ymax

try

intm=M.ikine6s(transl(i,j,z));

catch

bool=0;

end

if all(intm.\*1==intm)

bool=1;

if bool==[]

bool=0;

end

else

bool=0;

end

if bool==[]

bool=0;

end

%coordinateData= horzcat(coordinateData,[i;j;z]);

surfPlan=horzcat(surfPlan,bool);

%size(surfPlan)

end

end

%end

[SizeX,~]=size(surfPlan);

%{

for i = 1:SizeX

if ~all(surfPlan(i,:).\*1==surfPlan(i,:))

reachL=0;

else

reachL=1;

end

end

%}

%columnImage=surfPlan;

perfmat=horzcat(perfmat,surfPlan);

%end

end

try

performanceMatrix=vertcat(performanceMatrix,perfmat);

inputVector=vertcat(D(3:4)',A(2:3)',xmin,xmax,ymin,ymax,zmin,zmax);

inputMatrix=vertcat(inputMatrix,inputVector');

catch

end

end

d=[];

a=neural5(inputMatrix');

b=round(a');

for i=1:numData

c=sum(b(i,:)==performanceMatrix(i,:))/125;

d=[d,c];

end

%{

for i =1:numData

a=neru4(inputMatrix(i,:));

b=round(a);

c=sum(b==performanceMatrix(i,:))/125;

d=[d,c];

end

%}