Basic probability

1.

(a)

(b)

This is proved using the formula for the joint moment generating function of the linear transformation of a random vector.

The joint moment generating function of X X is

Therefore, the joint moment generating function of Y is

which is the moment generating function of a multivariate normal distribution with mean $A+Bmu $ and covariance matrix $BVB^{intercal }$

2.

(a)

(b)

Hands-on Exercises

1. (a)

function [ matrix ] = DCM\_ZYX( psi, tetta, phi )

% Question 1-(a)

% yaw - rotation around Z (psi)

% pitch - rotation around Y (tetta)

% roll - rotation around X (phi)

psi\_mx = [cos(psi), -sin(psi), 0;...

sin(psi), cos(psi), 0;...

0, 0, 1];

tetta\_mx=[cos(tetta), 0, sin(tetta);...

0, 1, 0;...

-sin(tetta), 0, cos(tetta)];

phi\_mx = [1, 0, 0;...

0, cos(phi), -sin(phi);...

0, sin(phi), cos(phi)];

matrix = psi\_mx\*tetta\_mx\*phi\_mx;

end

(b)

0.7289 0.0677 0.6813

0.3510 0.8174 -0.4567

-0.5878 0.5721 0.5721

(c)

function [ tetta, phi, psi]= DCM\_Euler\_B2G(rotMatrix)

if rotMatrix(1,3)==1 || rotMatrix(1,3)==-1

psi=0;

if rotMatrix(1,3)==-1

tetta=pi/2;

phi=psi+atan2(rotMatrix(2,1),rotMatrix(3,1));

else

tetta=-pi/2;

phi=-psi+atan2(-rotMatrix(2,1),-rotMatrix(3,1));

end

else

tetta(1)=-asin(rotMatrix(1,3));

tetta(2)=pi-tetta(1);

phi(1)=atan2(rotMatrix(2,3)/cos(tetta(1)),rotMatrix(3,3)/cos(tetta(1)));

phi(2)=atan2(rotMatrix(2,3)/cos(tetta(2)),rotMatrix(3,3)/cos(tetta(2)));

psi(1)=atan2(rotMatrix(1,2)/cos(tetta(1)),rotMatrix(1,1)/cos(tetta(1)));

psi(2)=atan2(rotMatrix(1,2)/cos(tetta(2)),rotMatrix(1,1)/cos(tetta(2)));

end

phi=phi\*180/pi;

tetta=tetta\*180/pi;

psi=psi\*180/pi;

end

(d)

tetta = -22.2422 202.2422

phi = 1.1161 -178.8839

psi = -28.4518 151.5482

2.

point = ([450;400;50;1]);

cam\_trans = [0.5363 -0.8440 0 451.2459;

0.8440 0.5363 0 -257.0322;

0 0 1 -400;

0 0 0 1];

point\_cam\_ref=cam\_trans\*point;

%explicit expression

point\_cam\_ref(1)=cam\_trans(1,1)\*point(1)+cam\_trans(1,2)\*point(2)+cam\_trans(1,3)\*point(3)+cam\_trans(1,4)\*point(4);

point\_cam\_ref(2)=cam\_trans(2,1)\*point(2)+cam\_trans(2,2)\*point(2)+cam\_trans(2,3)\*point(3)+cam\_trans(2,4)\*point(4);

point\_cam\_ref(3)=cam\_trans(3,1)\*point(3)+cam\_trans(3,2)\*point(2)+cam\_trans(3,3)\*point(3)+cam\_trans(3,4)\*point(4);

point\_cam\_ref(4)=cam\_trans(4,1)\*point(4)+cam\_trans(4,2)\*point(2)+cam\_trans(4,3)\*point(3)+cam\_trans(4,4)\*point(4);

The point in the camera coordinates is defined by

coordinates: [354.98 337.29 -350.00]

3. (a)

%first find dcm matrix for 1 degree transformation

transMatrix=zeros(4);

transMatrix(1:3,1:3) = DCM\_ZYX(pi/180,0,0);%real rotation

%build a transformation matrix knowing robot moves 1.01 meters each time

transMatrix(1,4)=1.01; %real movement

transMatrix(4,4)=1;

position = ([0; 0; 0; 1]);

for i=1:10

position(1:4,i+1)=transMatrix\*position(1:4,i);

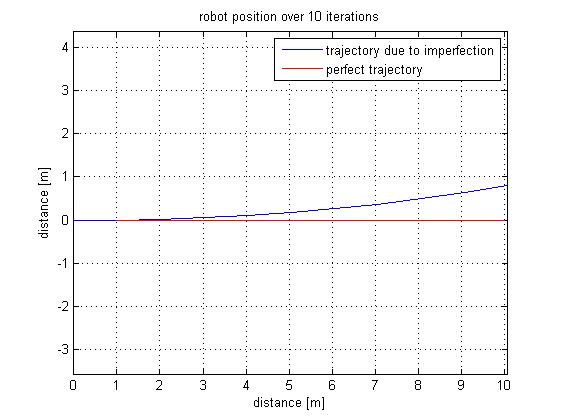
end

perfectTraj(1,:)=1:10;

perfectTraj(2,:)=0;

error=hypot((position(1,end)-perfectTraj(1,end)),(position(2,end)-perfectTraj(2,end)))

(b)



error = 0.7934 meter

4.

roll\_angle = d2r(10);

pitch\_angle = d2r(28);

yaw\_angle = d2r(5);

% Quaternions

quat\_roll = [cos(roll\_angle/2),(sin(roll\_angle/2)),0,0];

quat\_pitch = [cos(pitch\_angle/2),0,(sin(pitch\_angle/2)),0];

quat\_yaw = [cos(yaw\_angle/2),0,0,(sin(yaw\_angle/2))];

quat\_total=mult\_quat(mult\_quat(quat\_roll,quat\_pitch),quat\_yaw);

function [ t ] = mult\_quat( r,v )

% mulitplies 2 quaternions

% t = [r(1)\*v(1)-dot(r(2:4),v(2:4)),...

% cross(r(2:4),v(2:4)+r(1)\*v(2:4)+v(1)\*r(2:4))];

t0 = r(1)\*v(1)-dot(r(2:4),v(2:4));

t1 = v(1)\*r(2)+v(2)\*r(1)-v(3)\*r(4)+v(4)\*r(3);

t2 = v(1)\*r(3)+v(2)\*r(4)+v(3)\*r(1)-v(4)\*r(2);

t3 = v(1)\*r(4)-v(2)\*r(3)+v(3)\*r(2)+v(4)\*r(1);

t = [t0 t1 t2 t3];

end

The final quaterion for given rotation is the following:

[ 0.5840 -0.5977 0.4818 -0.2639 ]