function [x]=forward\_kin(q)

% [x] = forward\_kin(q) returns the position of end point given

% joint values q: [theta1, theta2, d3, theta4, theta5, theta6]

%

global H l2 l3 l4 l5;

theta1=q(1); theta2=q(2); theta3=q(3);

d4=q(4); theta5=q(5); theta6=q(6);

a = [0 l2 0 0 0 0];

alpha = [sym(pi)/2 -sym(pi)/2 0 sym(pi)/2 -sym(pi)/2 0];

d = [H 0 l3 d4 0 l4+l5];

theta = [theta1 theta2 theta3 0 theta5 theta6];

dhparams = [a' alpha' d' theta']

for i=1:6

A(:,:,i)=[

cos(theta(i)), -sin(theta(i))\*cos(alpha(i)), sin(theta(i))\*sin(alpha(i)), a(i)\*cos(theta(i));

sin(theta(i)), cos(theta(i))\*cos(alpha(i)), -cos(theta(i))\*sin(alpha(i)), a(i)\*sin(theta(i));

0, sin(alpha(i)), cos(alpha(i)), d(i);

0 0 0 1

];

end

alpha(1)

sum = A(:,:,1);

for i=2:6

sum = sum\*A(:,:,i);

end

x = sum(:,4);

x(4) = [];

end

function [J, JL] = jacobian\_mat\_simplify(qi,params)

% jacobian\_mat computes jacobian matrix of the robot given configuration q

%q - 3x1 vector of current joint values

% params=[H,l2,l3]

l2 = params(2);

l3 = params(3);

c1 = cos(qi(1));

s1 = sin(qi(1));

c2 = cos(qi(2));

s2 = sin(qi(2));

d4=qi(3);

sig1=c1\*c2\*l2-c1\*s2\*(d4+l3);

sig2=s2\*l2+c2\*(d4+l3);

sig3=s1\*s2\*(d4+l3);

sig4=c2\*s1\*l2;

J = [sig3-sig4 , -(c1)\*sig2 , -c1\*s2 ;

sig1 , -(s1)\*sig2 , -s1\*s2 ;

0 , c1\*sig1+s1\*(sig4-sig3) , c2 ;

0 , s1 , 0 ;

0 , -c1 , 0;

1 , 0 , 0 ];

JL = J(1:3,:);

end

function [J\_dot, JL\_dot]=jacobian\_mat\_dot(qi,q\_doti,params)

% J\_dot – the jacobian's time derivative.

% q\_dot – the joint parameters' time derivative.

% joint values q: [theta1, theta2, d4]

% params:[H,l2,l3]

l2 = params(2);

l3 = params(3);

c1 = cos(qi(1));

s1 = sin(qi(1));

c2 = cos(qi(2));

s2 = sin(qi(2));

d4 = qi(3);

J\_dot = zeros(6,3);

dJdq = zeros(6,3,3);

dJdq(:,:,1)=[c1\*s2\*(d4+l3)-c1\*c2\*l2, s1\*(s2\*l2+c2\*(d4+l3)), s1\*s2;

-s1\*c2\*l2+s1\*s2\*(d4+l3), -c1\*(s2\*l2+c2\*(d4+l3)), -c1\*s2;

0 , -s1\*(c1\*c2\*l2-c1\*s2\*(d4+l3))+c1\*(-s1\*c2\*l2+s1\*s2\*(d4+l3))+c1\*(c2\*s1\*l2-s1\*s2\*(d4+l3))+s1\*(c1\*c2\*l2-c1\*s2\*(d4+l3)) ,0;

0 , c1 ,0;

0 , s1 ,0;

0 , 0 ,0

];

dJdq(:,:,2)=[ s1\*c2\*(d4+l3)+s2\*s1\*l2, -c1\*(c2\*l2-s2\*(d4+l3)), -c1\*c2;

-c1\*s2\*l2-c1\*c2\*(d4+l3), -s1\*(c2\*l2-s2\*(d4+l3)), -s1\*c2;

0 ,c1\*(-c1\*s2\*l2-c1\*c2\*(d4+l3))-s1\*(s1\*c2\*(d4+l3)+s2\*s1\*l2), -s2;

0 ,0 ,0;

0 ,0 ,0;

0 ,0 ,0

];

dJdq(:,:,3)=[s1\*s2, -c1\*c2, 0;

-c1\*s2,-s1\*c2, 0;

0,-c1\*c1\*s2-s1\*s1\*s2,0;

0,0,0;

0,0,0;

0,0,0

];

for i =1:6

for j=1:3

for k = 1:3

J\_dot(i,j) = J\_dot(i,j) + dJdq(i,j,k)\*q\_doti(k);

end

end

end

JL\_dot = J\_dot(1:3,:);

end

function [x] = x\_plan(prof,T,n,x0,xf)

%General Description:

%This function returns a matrix, with each column representing the

%position vector of the tool in world frame at a particular time t\_i

%Parameters:

%prof: which trajectory profile the user wishes to implement

%[NOTE: prof accepts 'constant', 'trapezoidal', and 'polynomial']

%T: time taken to move from x0 to xf

%n: number of position vectors recorded in T seconds

%x0: initial position of tool (3x1 vector)

%xf: final position of tool (3x1 vector)

x = zeros(3,n);

t = linspace(0, T, n);

switch prof

case 'constant'

for i=1:n

x(:,i) = x0 + ((xf-x0)./T).\*t(i);

end

case 'trapezoidal'

a = (xf-x0).\*(36/(5\*T^2));

for i=1:floor(n/6)

x(:,i) = ((1/2)\*(t(i)^2)).\*a + x0;

end

for i=floor(n/6)+1:floor(5\*n/6)

x(:,i) = ((T/6)\*t(i)-(T^2)/72).\*a + x0;

end

for i=floor(5\*n/6)+1:n

x(:,i) = (-(1/2)\*t(i)^2+T\*t(i)-(13/36)\*T^2).\*a + x0;

end

case 'polynomial'

c5 = (6/T^5).\*(xf-x0);

c4 = (-15/T^4).\*(xf-x0);

c3 = (10/T^3).\*(xf-x0);

for i=1:n

x(:,i) = c5.\*t(i)^5+c4.\*t(i)^4+c3.\*t(i)^3+x0;

end

end

end

function [v] = v\_plan(prof,T,n,x0,xf)

%General Description:

%This function returns a matrix, with each column representing a the

%velocity vector of the tool in world frame at a particular time t\_i

%Parameters:

%prof: which trajectory profile the user wishes to implement

%[NOTE: prof accepts 'constant', 'trapezoidal', and 'polynomial']

%T: time taken to move from x0 to xf

%n: number of velocity vectors recorded in T seconds

%x0: initial position of tool (3x1 vector)

%xf: final position of tool (3x1 vector)

v = zeros(3,n);

t = linspace(0, T, n);

switch prof

case 'constant'

for i=1:n

v(:,i) = (xf-x0)./T;

end

case 'trapezoidal'

a = (xf-x0).\*(36/(5\*T^2));

for i=1:floor(n/6)

v(:,i) = t(i).\*a;

end

for i=floor(n/6)+1:floor(5\*n/6)

v(:,i) = (T/6).\*a;

end

for i=floor(5\*n/6)+1:n

v(:,i) = (T-t(i)).\*a;

end

case 'polynomial'

c5 = (6/T^5).\*(xf-x0);

c4 = (-15/T^4).\*(xf-x0);

c3 = (10/T^3).\*(xf-x0);

for i=1:n

v(:,i) = 5.\*c5.\*t(i)^4+4.\*c4.\*t(i)^3+3.\*c3.\*t(i)^2;

end

end

function [a] = a\_plan(prof,T,n,x0,xf)

%General Description:

%This function returns a matrix, with each column representing a the

%acceleration vector of the tool in world frame at a particular time t\_i

%Parameters:

%prof: which trajectory profile the user wishes to implement

%[NOTE: prof accepts 'constant', 'trapezoidal', and 'polynomial']

%T: time taken to move from x0 to xf

%n: number of acceleration vectors recorded in T seconds

%x0: initial position of tool (3x1 vector)

%xf: final position of tool (3x1 vector)

a = zeros(3,n);

t = linspace(0, T, n);

switch prof

case 'constant'

for i=1:n

a(:,i) = 0;

end

case 'trapezoidal'

a1 = (xf-x0).\*(36/(5\*T^2));

for i=1:floor(n/6)

a(:,i) = a1;

end

for i=floor(n/6)+1:floor(5\*n/6)

a(:,i) = 0;

end

for i=floor(5\*n/6)+1:n

a(:,i) = -a1;

end

case 'polynomial'

c5 = (6/T^5).\*(xf-x0);

c4 = (-15/T^4).\*(xf-x0);

c3 = (10/T^3).\*(xf-x0);

for i=1:n

a(:,i) = 20.\*c5.\*t(i)^3+12.\*c4.\*t(i)^2+6.\*c3.\*t(i);

end

end

function [q] = q\_plan(x, elbows, params)

% General Description:

% This function returns a matrix with each column representing the

% joint position vector at a particular time t\_i

% Parameters:

% x: matrix of preplanned position vectors, forming a trajectory

% elbows: 1x2 matrix which decides

% params: [H,l2,l3]

[~,colnum]=size(x);

q = zeros(3,colnum);

for i=1:colnum

q(:,i) = inverse\_kin(x(:,i), elbows, params);

end

function [q\_dot] = q\_dot\_plan(q, v, T, method, params)

% General Description:

% This function returns a matrix with each column representing a

% vector of the derivatives of the joint positions with respect to time

% at a particular time t\_i

% Parameters:

% q - matrix of joint position vectors (found previously using q\_plan)

% v - matrix of velocity vectors of the tool vector over T seconds

% method - user specifies whether to calculate q\_dot numerically, or using

% analytical definition of x\_dot = JL\*q\_dot

% T-time taken to move from x0 to xf

% params=[H,l2,l3]

[~, colnum] = size(q);

q\_dot = zeros(3, colnum);

JL = zeros(3, 3, colnum);

switch method

case 'numerical'

for i = 1:3

q\_dot(i,:) = gradient(q(i,:), T/colnum);

end

case 'analytical'

for i=1:colnum

[~,JL(:,:,i)] = jacobian\_mat\_simplify(q(:,i), params);

q\_dot(:,i) = inv((JL(:,:,i)))\*v(:,i);

end

end

function [q\_dot2] = q\_dot2\_plan(q, q\_dot, a, T, method, params)

% General Description:

% This function returns a matrix with each column representing a

% vector of the derivatives of the joint positions with respect to time

% at a particular time t\_i

% Parameters:

% q - matrix of joint position vectors (found previously using q\_plan)

% q\_dot - matrix of joint velocity vectors (found previously using q\_dot\_plan)

% a - matrix of velocity vectors of the tool vector over T seconds

% method - user specifies whether to calculate q\_dot numerically, or using

% analytical definition of x\_dot2 -JL\_dot\*q = JL\*q\_dot2

% T-time taken to move from x0 to xf

% params=[H,l2,l3]

[~, colnum] = size(q);

q\_dot2 = zeros(3, colnum);

J=zeros(6,3,colnum);

JL = zeros(3, 3, colnum);

JL\_dot = zeros(3, 3, colnum-1);

switch method

case 'numerical'

for i = 1:3

q\_dot2(i,:) = gradient(q\_dot(i,:), T/colnum);

end

case 'analytical'

for i=1:colnum

[~,JL(:,:,i)] = jacobian\_mat\_simplify(q(:,i), params);

[~, JL\_dot(:,:,i)] = jacobian\_mat\_dot(q(:,i),q\_dot(:,i),params);

q\_dot2(:,i) = inv(JL(:,:,i))\*(a(:,i)-JL\_dot(:,:,i)\*q\_dot(:,i));

end

end