自主避障算法报告

1. 算法概述

1.1 系统框架

Fig 1.1 系统框架图

1.2 算法流程

Fig 1.2 算法流程图

1.3 运行说明

1. 运行环境：Matlab 64bit /Simulink 。

2. 包含以下文件和函数：

3. 函数调用图

Fig 1.3 函数调用图

2. 机械臂与三维环境建模

2.1 机械臂建模

Fig 2.1 D-H 模型

```
%% ——机械臂几何模型——
%
function [Base,Arm1,Arm2,Arm3,Arm4,Arm5,Arm6,Arm7]=arm(L,q,C_Base,C_Arm,C_Hand)

% 关节角
% deg=pi/180;
% q=[30 30 0 50 0 0 0]*deg;

L1=L(1);
L2=L(2);
L3=L(3);
L4=L(4);
L5=L(5);
L6=L(6);
L7=L(7);

%——cylinder 生成圆柱体数据
n=20;[x,y,z]=cylinder(1,n);
X0=[zeros(1,n+1);x;zeros(1,n+1)];
Y0=[zeros(1,n+1);y;zeros(1,n+1)];
Z0=[z(1,:);z;z(2,:)];

%——机械臂外形尺寸

%——Cube_0 基座
Base=cube(C_Base.Coordinate(1),C_Base.Coordinate(2),C_Base.Coordinate(3),...
          C_Base.L,C_Base.W,C_Base.H);

%——cylinder_1
```

```
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=L1.d*Z0;

L0=Link('d', 0, 'a', 0, 'alpha', 0);
T=L0.A(0);

[x1,y1,z1]=transform(X,Y,Z,T);
Arm1=cylind(x1,y1,z1);

%——cylinder_2
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=80*Z0-40;

T=L1.A(q(1));

[x1,y1,z1]=transform(X,Y,Z,T);
Arm2=cylind2(x1,y1,z1);

%——cylinder_3
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=L3.d*Z0;

T=L1.A(q(1))*L2.A(q(2));

[x2,y2,z2]=transform(X,Y,Z,T);
Arm3=cylind(x2,y2,z2);

%——cylinder_4
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=80*Z0-40;

T=L1.A(q(1))*L2.A(q(2))*L3.A(q(3));

[x2,y2,z2]=transform(X,Y,Z,T);
Arm4=cylind2(x2,y2,z2);

%——cylinder_5
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=L5.d*Z0;
```

```
T=L1.A(q(1))*L2.A(q(2))*L3.A(q(3))*L4.A(q(4));

[x3,y3,z3]=transform(X,Y,Z,T);
Arm5=cylind(x3,y3,z3);

%——cylinder_7
X=C_Hand.R*X0;
Y=C_Hand.R*Y0;
Z=L7.d*Z0;

T=L1.A(q(1))*L2.A(q(2))*L3.A(q(3))*L4.A(q(4))*L5.A(q(5))*L6.A(q(6));

[x4,y4,z4]=transform(X,Y,Z,T);
Arm7=cylind(x4,y4,z4);

%——cylinder_6
X=C_Arm.R*X0;
Y=C_Arm.R*Y0;
Z=80*Z0-40;

T=L1.A(q(1))*L2.A(q(2))*L3.A(q(3))*L4.A(q(4))*L5.A(q(5));

[x3,y3,z3]=transform(X,Y,Z,T);
Arm6=cylind2(x3,y3,z3);

%——plane——
% ezmesh('0',120)
% delete(get(gca,'title'))

end

%%
%——坐标变换
function [xt,yt,zt] = transform(x,y,z,T)
p1=[x(1,:);y(1,:);z(1,:)]; p1=T*p1;
p2=[x(2,:);y(2,:);z(2,:)]; p2=T*p2;
p3=[x(3,:);y(3,:);z(3,:)]; p3=T*p3;
p4=[x(4,:);y(4,:);z(4,:)]; p4=T*p4;

xt=[p1(1,:);p2(1,:);p3(1,:);p4(1,:)];
yt=[p1(2,:);p2(2,:);p3(2,:);p4(2,:)];
zt=[p1(3,:);p2(3,:);p3(3,:);p4(3,:)];
end

%——圆柱体1
```

```
function cyObj = cylind(x,y,z)
[TRI,V]= surf2patch(x,y,z);
cy.Vertices = V;
cy.Faces = TRI;
cy.FaceVertexCData = ones(size(V,1),1)*[0.80,0.30,0.10];   % parula
cy.FaceColor = 'interp';
cyObj = patch(cy,'LineStyle','none');
end

%——圆柱体2
function cyObj = cylind2(x,y,z)
[TRI,V]= surf2patch(x,y,z);
cy.Vertices = V;
cy.Faces = TRI;
cy.FaceVertexCData = flag(size(V,1));
cy.FaceColor = 'interp';
cyObj = patch(cy,'LineStyle','none');
end

%——cube——
function cbObj = cube(x0,y0,z0,a,b,c)
V1=[-1;  1;  1; -1; -1;  1;  1; -1;];
V2=[-1; -1;  1;  1; -1; -1;  1;  1;];
V3=[-1; -1; -1; -1;  1;  1;  1;  1;];
F= [1 2 3 4; 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 5 6 7 8;];

[THETA,PHI,~]=cart2sph(V1,V2,V3);
R=ones(size(V1(:,1)));
[V1,V2,V3]=sph2cart(THETA,PHI,R);
V=[a*V1+x0 b*V2+y0 c*V3+z0];

cb.Vertices = V;
cb.Faces = F;
cb.FaceVertexCData = bone(size(V,1));
cb.FaceColor = 'interp';
cbObj = patch(cb,'LineStyle','none'); % alpha(0.2)
end
```

2.2 三维环境建模

```
%% ———3D环境模型———
% 包含  长方体/圆柱体/球体
%
```

```
function [cbObj,cbObj2,cyObj,spObj] = sence(CU,CU2,CU3,CY,SP)

%——plane——
% ezmesh('0'),delete(get(gca,'title'));
[x,y]=meshgrid(-2000:100:2000);
z = -100*ones(size(x,1));
mesh(x,z,-y,'EdgeColor','k','FaceAlpha',0.6);

%——cube——
cbObj=cb(CU.Coordinate(1),CU.Coordinate(2),CU.Coordinate(3), CU.L, CU.W, CU.H);
cbObj2=cb2(CU2.Coordinate(1),CU2.Coordinate(2),CU2.Coordinate(3),...
            CU2.L, CU2.W, CU2.H);
cb(CU3.Coordinate(1),CU3.Coordinate(2),CU3.Coordinate(3), CU3.L, CU3.W, CU3.H);

%——cylinder——
cyObj=cy(CY.Coordinate(1),CY.Coordinate(2),CY.Coordinate(3), CY.R, CY.H);

%——sphere——
spObj=sp(SP.Coordinate(1),SP.Coordinate(2),SP.Coordinate(3), SP.R);

end

%%
%——cube——
function cbObj = cb(x0,y0,z0,a,b,c)
V1=[-1;  1;  1; -1; -1;  1;  1; -1;];
V2=[-1; -1;  1;  1; -1; -1;  1;  1;];
V3=[-1; -1; -1; -1;  1;  1;  1;  1;];
F= [1 2 3 4; 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 5 6 7 8;];

[THETA,PHI,~]=cart2sph(V1,V2,V3);
R=ones(size(V1(:,1)));
[V1,V2,V3]=sph2cart(THETA,PHI,R);
% V=[a*V1+x0 b*V2+y0 c*V3+z0];
V=[(a*V1+x0) (c*V3+z0) -(b*V2+y0)];

cb.Vertices = V;
cb.Faces = F;
cb.FaceVertexCData = bone(size(V,1)); % bone
cb.FaceColor = 'interp';
cbObj = patch(cb,'LineStyle','none');
end

%——cube——
function cbObj = cb2(x0,y0,z0,a,b,c)
```

```
V1=[-1;  1;  1; -1; -1;  1;  1; -1;];
V2=[-1; -1;  1;  1; -1; -1;  1;  1;];
V3=[-1; -1; -1; -1;  1;  1;  1;  1;];
F= [1 2 3 4; 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 5 6 7 8;];

[THETA,PHI,~]=cart2sph(V1,V2,V3);
R=ones(size(V1(:,1)));
[V1,V2,V3]=sph2cart(THETA,PHI,R);
% V=[a*V1+x0 b*V2+y0 c*V3+z0];
V=[(a*V1+x0) (c*V3+z0) -(b*V2+y0)];

cb.Vertices = V;
cb.Faces = F;
cb.FaceVertexCData = gray(size(V,1)); % bone
cb.FaceColor = 'interp';
cbObj = patch(cb,'LineStyle','none');
end

%--cylinder--
function cyObj = cy(x0,y0,z0,r,h)
n=20;[x,y,z]=cylinder(1,n);
x=[zeros(1,n+1);x;zeros(1,n+1)];
y=[zeros(1,n+1);y;zeros(1,n+1)];
z=[z(1,:);z;z(2,:)];
% vx=r*x+x0;vy=r*y+y0;vz=h*z+z0;
vx=r*x+x0;vy=h*z+z0;vz=r*y+y0;vz=-vz;
[TRI,V]= surf2patch(vx,vy,vz);

cy.Vertices = V;
cy.Faces = TRI;
cy.FaceVertexCData = lines(size(V,1));   % parula
cy.FaceColor = 'interp';
cyObj = patch(cy,'LineStyle','none');
end

%--sphere--
function spObj = sp(x0,y0,z0,r)
f=@(x,y,z)(x-x0).^2+(y-y0).^2+(z-z0).^2-r^2;
[x,y,z]=meshgrid(r*linspace(-1.1,1.1,20));
x=x+x0;y=y+y0;z=z+z0;val=f(x,y,z);
[p,v]=isosurface(x,y,z,val,0);

sp.Vertices = v;
sp.Faces = p;
sp.FaceVertexCData = parula(size(v,1));  %
```

```
sp.FaceColor = 'interp';
spObj = patch(sp,'LineStyle','none'); alpha(0.8); axis equal
end
```

3. 碰撞检测方法

3.1 算法原理

- SAT
- GJK

3.2 算法实现

```
function flag = GJK(shape1,shape2,iterations)
% GJK Gilbert-Johnson-Keerthi Collision detection implementation.
% Returns whether two convex shapes are are penetrating or not
% (true/false). Only works for CONVEX shapes.
%

%Point 1 and 2 selection (line segment)
v = [0.8 0.5 1];
[a,b] = pickLine(v,shape2,shape1);

%Point 3 selection (triangle)
[a,b,c,flag] = pickTriangle(a,b,shape2,shape1,iterations);

%Point 4 selection (tetrahedron)
if flag == 1 %Only bother if we could find a viable triangle.
    [a,b,c,d,flag] = pickTetrahedron(a,b,c,shape2,shape1,iterations);
end

end

function [a,b] = pickLine(v,shape1,shape2)
%Construct the first line of the simplex
b = support(shape2,shape1,v);
a = support(shape2,shape1,-v);
end
```

```
function [a,b,c,flag] = pickTriangle(a,b,shape1,shape2,IterationAllowed)
flag = 0; %So far, we don't have a successful triangle.

%First try:
ab = b-a;
ao = -a;
v = cross(cross(ab,ao),ab); % v is perpendicular to ab pointing in the general direction o

c = b;
b = a;
a = support(shape2,shape1,v);

for i = 1:IterationAllowed %iterations to see if we can draw a good triangle.
    %Time to check if we got it:
    ab = b-a;
    ao = -a;
    ac = c-a;

    %Normal to face of triangle
    abc = cross(ab,ac);

    %Perpendicular to AB going away from triangle
    abp = cross(ab,abc);
    %Perpendicular to AC going away from triangle
    acp = cross(abc,ac);

    %First, make sure our triangle "contains" the origin in a 2d projection
    %sense.
    %Is origin above (outside) AB?
    if dot(abp,ao) > 0
        c = b; %Throw away the furthest point and grab a new one in the right direction
        b = a;
        v = abp; %cross(cross(ab,ao),ab);

        %Is origin above (outside) AC?
    elseif dot(acp, ao) > 0
        b = a;
        v = acp; %cross(cross(ac,ao),ac);

    else
        flag = 1;
        break; %We got a good one.
    end
    a = support(shape2,shape1,v);
end
```

```
end

function [a,b,c,d,flag] = pickTetrahedron(a,b,c,shape1,shape2,IterationAllowed)
%Now, if we're here, we have a successful 2D simplex, and we need to check
%if the origin is inside a successful 3D simplex.
%So, is the origin above or below the triangle?
flag = 0;

ab = b-a;
ac = c-a;

%Normal to face of triangle
abc = cross(ab,ac);
ao = -a;

if dot(abc, ao) > 0 %Above
    d = c;
    c = b;
    b = a;

    v = abc;
    a = support(shape2,shape1,v); %Tetrahedron new point

else %below
    d = b;
    b = a;
    v = -abc;
    a = support(shape2,shape1,v); %Tetrahedron new point
end

for i = 1:IterationAllowed %Allowing 10 tries to make a good tetrahedron.
    %Check the tetrahedron:
    ab = b-a;
    ao = -a;
    ac = c-a;
    ad = d-a;

    %We KNOW that the origin is not under the base of the tetrahedron based on
    %the way we picked a. So we need to check faces ABC, ABD, and ACD.

    %Normal to face of triangle
    abc = cross(ab,ac);

    if dot(abc, ao) > 0 %Above triangle ABC
        %No need to change anything, we'll just iterate again with this face as
```

```
        %default.
    else
        acd = cross(ac,ad);%Normal to face of triangle

        if dot(acd, ao) > 0 %Above triangle ACD
            %Make this the new base triangle.
            b = c;
            c = d;
            ab = ac;
            ac = ad;
            abc = acd;
        else
            adb = cross(ad,ab);%Normal to face of triangle

            if dot(adb, ao) > 0 %Above triangle ADB
                %Make this the new base triangle.
                c = b;
                b = d;
                ac = ab;
                ab = ad;
                abc = adb;
            else
                flag = 1;
                break; %It's inside the tetrahedron.
            end
        end
    end

    %try again:
    if dot(abc, ao) > 0 %Above
        d = c;
        c = b;
        b = a;
        v = abc;
        a = support(shape2,shape1,v); %Tetrahedron new point
    else %below
        d = b;
        b = a;
        v = -abc;
        a = support(shape2,shape1,v); %Tetrahedron new point
    end
end

end
```

```
function point = getFarthestInDir(shape, v)
%Find the furthest point in a given direction for a shape
XData = get(shape,'XData'); % Making it more compatible with previous MATLAB releases.
YData = get(shape,'YData');
ZData = get(shape,'ZData');
dotted = XData*v(1) + YData*v(2) + ZData*v(3);
[maxInCol,rowIdxSet] = max(dotted);
[maxInRow,colIdx] = max(maxInCol);
rowIdx = rowIdxSet(colIdx);
point = [XData(rowIdx,colIdx), YData(rowIdx,colIdx), ZData(rowIdx,colIdx)];
end

function point = support(shape1,shape2,v)
%Support function to get the Minkowski difference.
point1 = getFarthestInDir(shape1, v);
point2 = getFarthestInDir(shape2, -v);
point = point1 - point2;
end
```

4. RRTstar 路径生成

4.1 算法原理[1]

4.2 算法实现[2]

```
%% 3D RRT star
%
function trace = RRTStar_3D(DH,L,C_Base,C_Arm,C_Hand,cbObj,cbObj2,cyObj,...
                 start,goal,randompoint,EPS,lamdaEPS)

% EPS = 20;
% lamdaEPS = 4;

numphi=10;  % 臂型角离散数
iterationsAllowed=2;

q_start.coord = start';
q_start.cost = 0;
q_start.parent = 0;
```

[1] ABC
[2] ABD

```
q_start.theta = invkine(start,DH,numphi);

q_goal.coord = goal';
q_goal.cost = 0;
q_goal.theta = invkine(goal,DH,numphi);

nodes(1) = q_start;    % 节点

%%
for i = 1:1:size(randompoint,1)
    tic
    disp(['采样迭代次数' num2str(i)])
    q_rand = randompoint(i,:);
%     plot3(q_rand(1), q_rand(2), q_rand(3), '.')

    % Break if goal node is already reached
    for j = 1:1:length(nodes)
        if nodes(j).coord == q_goal.coord
           break
        end
    end

    % Pick the closest node from existing list
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist_3d(n.coord, q_rand);
        ndist = [ndist tmp];
    end
    [val, idx] = min(ndist);
    q_near = nodes(idx);

    q_new.coord = steer3d(q_rand, q_near.coord, val, EPS);

    [CollisionFlag,q_noCollision] = CollisionCheck(DH,L,C_Base,C_Arm,C_Hand,cbObj,cbObj2,cy
        q_new,numphi,iterationsAllowed);
    q_new.theta = q_noCollision;

    figure(1);
    if CollisionFlag
    line([q_near.coord(1), q_new.coord(1)], [q_near.coord(2), q_new.coord(2)],...
         [q_near.coord(3), q_new.coord(3)],'Color','k','LineWidth',1);
    drawnow   % 强烈建议不要，速度变慢很多
    hold on
    q_new.cost = dist_3d(q_new.coord, q_near.coord) + q_near.cost;
```

```
    % Within a radius r, find all existing nodes
    q_nearest = [];
    r = lamdaEPS*EPS;
    neighbor_count = 1;
    for j = 1:1:length(nodes)
        if (dist_3d(nodes(j).coord, q_new.coord)) <= r
           q_nearest(neighbor_count).coord = nodes(j).coord;
           q_nearest(neighbor_count).cost = nodes(j).cost;
           neighbor_count = neighbor_count+1;
        end
    end

    % Initialize cost to currently known value
    q_min = q_near;
    C_min = q_new.cost;

    % Iterate through all nearest neighbors to find alternate lower cost paths
    for k = 1:1:length(q_nearest)
        if q_nearest(k).cost + dist_3d(q_nearest(k).coord, q_new.coord) < C_min
           q_min = q_nearest(k);
           C_min = q_nearest(k).cost + dist_3d(q_nearest(k).coord, q_new.coord);
           line([q_min.coord(1), q_new.coord(1)], [q_min.coord(2), q_new.coord(2)],...
                [q_min.coord(3), q_new.coord(3)],'LineWidth',0.2);
           hold on
        end
    end

    % Update parent to least cost-from node
    for j = 1:1:length(nodes)
        if nodes(j).coord == q_min.coord
            q_new.parent = j;
        end
    end

    % Append to nodes
    nodes = [nodes q_new];

    end
    time=toc;
    disp(['采样迭代用时' num2str(time) 's'])
end

%%
D = [];
```

```
for j = 1:1:length(nodes)
    tmpdist = dist_3d(nodes(j).coord, q_goal.coord);
    D = [D tmpdist];
end

% Search backwards from goal to start to find the optimal least cost path
[~, idx] = min(D);
% q_final = nodes(idx);
q_goal.parent = idx;
q_end = q_goal;
trace=q_end;
nodes = [nodes q_goal];
while q_end.parent ~= 0
    startq = q_end.parent;
    line([q_end.coord(1), nodes(startq).coord(1)], [q_end.coord(2), nodes(startq).coord(2)
         [q_end.coord(3), nodes(startq).coord(3)], 'Color', 'g', 'LineWidth', 3);
    hold on
    q_end = nodes(startq);
    trace = [trace q_end];
end

    trace = fliplr(trace);

end

%%-------------------------------------------------------------------------
%
function d = dist_3d(q1,q2)
    d = sqrt((q1(1)-q2(1))^2 + (q1(2)-q2(2))^2 + (q1(3)-q2(3))^2);
end

%
function A = steer3d(qr, qn, val, eps)
   qnew = [0 0];
   if val >= eps
       qnew(1) = qn(1) + ((qr(1)-qn(1))*eps)/dist_3d(qr,qn);
       qnew(2) = qn(2) + ((qr(2)-qn(2))*eps)/dist_3d(qr,qn);
       qnew(3) = qn(3) + ((qr(3)-qn(3))*eps)/dist_3d(qr,qn);
   else
       qnew(1) = qr(1);
       qnew(2) = qr(2);
       qnew(3) = qr(3);
   end
   A = [qnew(1), qnew(2), qnew(3)];
end
```

5. 数值仿真实现

6. TODO lists

- [x] @mentions, **formatting**, and tags supported
- [x] list syntax required (any unordered or ordered list supported)
- [ ] this is a complete item
- [ ] this is an incomplete item

参考文献