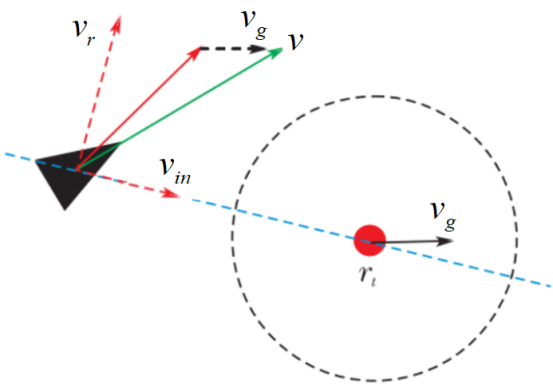
**Circle01算法描述**

# 功能描述

在无障碍情况下，实现单个机器人对静止目标和运动目标的环绕包围

# 算法描述

将机器人的速度分解为3个部分：



1. 向着目标的**直线运动速度**



1. 常数，机器人离轨道的最短距离，是机器人和目标的距离，是期望包围半径；机器人离轨道越近，机器人接近目标的速度越小，当机器人到达轨迹上时；
2. 是从**目标指向机器人**的矢量，时，机器人在圆内，与机器人入轨的方向相同，但是，因此要加“**—**”修正方向；反之，当，机器人在圆外，也需要加“**—**”修正方向。
3. 垂直于该运动方向以目标为中心的**环绕运动速度**



该速度为常量，即机器人在轨道上做匀速圆周运动。

1. 目标的运动速度

该速度的作用是使得机器人和目标是相对静止的。

编程实现该算法的要点：

1、将t0时间内各速度方向的位移，分解为X轴和Y轴方向上的位移，然后叠加合成机器人的实际位移，然后得出t0时间后机器人的位置信息。

# 实现步骤

## 定义“机器人”类文件Robot2D\_A.m

%-----------------------------------Robot2D\_A.m-----------------------------------

classdef Robot2D\_A < handle

properties

x %机器人横坐标

y %机器人纵坐标

Wd %期望角速度

R %期望半径

k %入轨速度控制因子

Ox %目标的横坐标

Oy %目标的纵坐标

end

properties(Dependent)

xo %以目标为原点时，机器人的相对横坐标

yo %以目标为原点时，机器人的相对纵坐标

vin %机器人和目标连线上，指向轨迹的速度

vr %环绕速度，垂直于Vin

r %机器人到目标的距离

th %机器人与目标连线与X轴的夹角

end

methods

function obj = Robot2D\_A(x0, y0, Wd0, R0, k0, Ox0, Oy0)

obj.x = x0;

obj.y = y0;

obj.Wd = Wd0;

obj.R = R0;

obj.k = k0;

obj.Ox = Ox0;

obj.Oy = Oy0;

end

%Dependent 属性的计算公式要放在get方法中

function xo = get.xo(obj)

xo = obj.x-obj.Ox;

end

function yo = get.yo(obj)

yo = obj.y-obj.Oy;

end

function r = get.r(obj)

r = sqrt(obj.xo^2 + obj.yo^2);

end

%计算入轨速度

function vin = get.vin(obj)

vin = -obj.k \* (obj.r - obj.R); %入轨速度和机器人与轨道之间的距离成正比。

end

%计算环绕速度

function vr = get.vr(obj)

vr = obj.R \* obj.Wd;

end

%计算机器人当前位置与X轴的夹角

if (obj.xo >= 0) %第一、四象限

th = atan(obj.yo / obj.xo);

else %第二、三象限

th = pi + atan(obj.yo / obj.xo);

end

end

end

## 机器人的包围环绕函数**FGtrace.m**

%跟随者对目标进行环绕包围的函数

function FGtrace(Obj)

xo = Obj.x - Obj.Gx; %以目标为原点时，机器人的相对横坐标

yo = Obj.y - Obj.Gy; %以目标为原点时，机器人的相对纵坐标

th = atan(yo / xo); %计算机器人当前位置与X轴的夹角

r = sqrt(xo^2 + yo^2); %得到机器人与目标之间的距离

vt = - Obj.k \* (r - Obj.R); %计算入轨速度

vr = Obj.R \* Obj.Wd; %计算环绕速度

St = vt \* Obj.to; %机器人在vt方向上移动的距离

Sr = vr \* Obj.to; %机器人在vr方向上移动的距离

%跟随者的环绕修正,当跟随者的相对位置处于不同象限时，to内的位移方向会发生改变

if (xo < 0) %第二、三象限

th = pi + th;

end

%计算机器人在时间间隔内，在X和Y方向上的移动距离

Obj.Gdx = St \* cos(th) + Sr \* sin(th);

Obj.Gdy = St \* sin(th) - Sr \* cos(th);

end

## 机器人的避障函数**ABtrace.m**

%只有障碍物作用下的跟随者避障轨迹

%该函数只有距离限制，没有角度限制

function ABtrace(Obj)

xo = Obj.x - Obj.Bx; %以障碍物为原点时，机器人的相对横坐标

yo = Obj.y - Obj.By; %以障碍物为原点时，机器人的相对纵坐标

th = atan(yo / xo); %计算机器人当前位置与X轴的夹角

r = sqrt(xo^2 + yo^2); %得到机器人与障碍物之间的距离

%跟随者的方向修正,当跟随者的相对位置处于不同象限时，to内的位移方向会发生改变

if (xo < 0) %第二、三象限

th = pi + th;

end

if r <= Obj.Rb

vb = Obj.kb / r^2; %障碍物产生的斥力

Obj.Bdx = vb \* Obj.to \* cos(th);

Obj.Bdy = vb \* Obj.to \* sin(th);

else

Obj.Bdx = 0;

Obj.Bdy = 0;

end

end

## 主函数**circle01.m**

### 初始化，生成跟踪者对象

%初始化，1#跟踪者负责环绕静止目标

Gx = 2; %机器人起点横坐标

Gy = 1; %机器人起点纵坐标

GWd = 0.04; %期望角速度

GR = 3; %期望半径

Gk = 0.001; %入轨速度控制因子

GOx = 3; %目标的横坐标

GOy = 3; %目标的纵坐标

f1 = Robot01(Gx, Gy, GWd, GR, Gk, GOx, GOy); % 1#跟踪者

%初始化，2#跟踪者负责环绕1#目标

x = 5; %跟踪者起点横坐标

y = 3; %跟踪者起点纵坐标

Wd = 3; %期望角速度

R = 1; %期望半径

k = 0.5; %入轨速度控制因子

Ox = Gx; %目标的横坐标

Oy = Gy; %目标的纵坐标

f2 = Robot01(x, y, Wd, R, k, Ox, Oy); % 2#跟踪者

### 初始化跟踪者对象的位置数组

t0 = 0.1;

t = t0: t0 :100; %机器人的移动时间

%初始化1#跟踪者的位置数组

X1 = zeros(1,1000);

Y1 = zeros(1,1000);

%初始化2#跟踪者的位置数组

X2 = zeros(1,1000);

Y2 = zeros(1,1000);

### 动画演示及绘图框架初始化

%设置1#跟踪者的运动轨迹,1#跟踪者负责环绕运动目标，即2#跟踪者

h1 = animatedline;

h1.Color = 'green';

h1.LineStyle = ':';

h1.LineWidth = 2;

%设置2#跟踪者的运动轨迹，2#跟踪者负责环绕静止目标

h2 = animatedline;

h2.Color = 'red';

h2.LineStyle = ':';

h2.LineWidth = 0.5;

%设置各个目标的特征标记

hold on

axis([-2 7 -2 7]);

axis equal;

%静止目标

C = plot(GOx,GOy,'+');

C.LineWidth = 2;

%1#跟踪者

p1 = plot(Gx,Gy,'o','MarkerFaceColor','green');

%2#跟踪者

p2 = plot(x,y,'o','MarkerFaceColor','red');

hold off

### 目标运动算法描述

for i = 1:1000

%----------------------- 1#跟踪者运动 -----------------------

Gt(i) = f1.vt \* t0; %目标在G.vt方向上移动的距离

Gr(i) = f1.vr \* t0; %目标在G.vr方向上移动的距离

%求解t0后目标的位置

dGx = Gt(i) \* cos(f1.th) + Gr(i) \* sin(f1.th);

dGy = Gt(i) \* sin(f1.th) - Gr(i) \* cos(f1.th);

X1(i) = f1.x + dGx;

Y1(i) = f1.y + dGy;

%更新目标的位置信息

f1.x = X1(i);

f1.y = Y1(i);

%----------------------- 2#跟踪者运动 -----------------------

%更新运动目标的实时位置

f2.Ox = f1.x;

f2.Oy = f1.y;

St(i) = f2.vt \* t0; %跟踪者在ri.vt方向上移动的距离

Sr(i) = f2.vr \* t0; %跟踪者在ri.vr方向上移动的距离

%求解t0后跟踪者的位置

dx = St(i) \* cos(f2.th) + Sr(i) \* sin(f2.th);

dy = St(i) \* sin(f2.th) - Sr(i) \* cos(f2.th);

X2(i) = f2.x + dGx + dx;

Y2(i) = f2.y + dGy + dy;

%更新跟踪者的位置信息

f2.x = X2(i);

f2.y = Y2(i);

%动画设置

addpoints(h1,X1(i),Y1(i));

addpoints(h2,X2(i),Y2(i));

p1.XData = X1(i);

p1.YData = Y1(i);

p2.XData = X2(i);

p2.YData = Y2(i);

drawnow

end

### qt

## 其他