Anexo A: Algoritmos para la Simulación (Matlab)

A.1. Cálculo de Mínima Trayectoria

```
clc
clear
close all


L = 0.190;          % Longitud del vehículo
A = 0.143;          % Ancho del vehículo
deltamax = 12*pi/180;
R = L/tan(deltamax);
N = 100;

xi=0; yi=0; xf=4; yf=0; thetai=90; thetaf=45;

thetai = thetai*pi/180;
thetaf = thetaf*pi/180;
if (thetai < 0)
    thetai = thetai + 2*pi;
end
if (thetaf < 0)
    thetaf = thetaf + 2*pi;
end

figure(1)
hold on

% POSICION INICIAL
drawCar(xi,yi,thetai,L,A,'r')

c1xi = xi - R*sin(thetai);
c1yi = yi + R*cos(thetai);
c2xi = xi + R*sin(thetai);
c2yi = yi - R*cos(thetai);
plot(c1xi,c1yi,'+g',c2xi,c2yi,'+g')
Ci = [ c1xi c1yi c2xi c2yi ];

% Dibujar circulo
phi = 0:0.01:2*pi;
x = R*cos(phi) + c1xi;
y = R*sin(phi) + c1yi;
plot(x,y,'g','LineWidth',1)
x = R*cos(phi) + c2xi;
y = R*sin(phi) + c2yi;
plot(x,y,'g','LineWidth',1)

% POSICION FINAL
drawCar(xf,yf,thetaf,L,A,'b')

c1xf = xf - R*sin(thetaf);
c1yf = yf + R*cos(thetaf);
c2xf = xf + R*sin(thetaf);
c2yf = yf - R*cos(thetaf);
plot(c1xf,c1yf,'+g',c2xf,c2yf,'+g')
Cf = [ c1xf c1yf c2xf c2yf ];

% Dibujar circulo
phi = 0:0.01:2*pi;
x = R*cos(phi) + c1xf;
y = R*sin(phi) + c1yf;
plot(x,y,'g','LineWidth',1)
```

```
x = R*cos(phi) + c2xf;
y = R*sin(phi) + c2yf;
plot(x,y,'g','LineWidth',1)
grid
xlabel('x (m)')
ylabel('y (m)')
axis equal

figure(2)
title('Posición Inicial y Final del Robot Móvil')
xlabel('x (m)')
ylabel('y (m)')
hold on
drawCar(xi,yi,thetai,L,A,'r')
drawCar(xf,yf,thetaf,L,A,'b')

grid
axis equal

% Datos conocidos:
%     xi, xf, yi, yf, thetai, thetaf, Ci, Cf, R
%     Puntos de rectas tangentes internas y externas

[W11i,X11i,Y11i,Z11i] = TangenteInterior(Ci(1),Ci(2),R,Cf(1),Cf(2),R);
[W12i,X12i,Y12i,Z12i] = TangenteInterior(Ci(1),Ci(2),R,Cf(3),Cf(4),R);
[W21i,X21i,Y21i,Z21i] = TangenteInterior(Ci(3),Ci(4),R,Cf(1),Cf(2),R);
[W22i,X22i,Y22i,Z22i] = TangenteInterior(Ci(3),Ci(4),R,Cf(3),Cf(4),R);

[W11e,X11e,Y11e,Z11e] = TangenteExterior(Ci(1),Ci(2),R,Cf(1),Cf(2),R);
[W12e,X12e,Y12e,Z12e] = TangenteExterior(Ci(1),Ci(2),R,Cf(3),Cf(4),R);
[W21e,X21e,Y21e,Z21e] = TangenteExterior(Ci(3),Ci(4),R,Cf(1),Cf(2),R);
[W22e,X22e,Y22e,Z22e] = TangenteExterior(Ci(3),Ci(4),R,Cf(3),Cf(4),R);

text(Ci(1),Ci(2),'1i')
text(Ci(3),Ci(4),'2i')
text(Cf(1),Cf(2),'1f')
text(Cf(3),Cf(4),'2f')

% Dibujo los carros
x1 = xi + (A/2)*sin(thetai);
y1 = yi - (A/2)*cos(thetai);
x2 = xi - (A/2)*sin(thetai);
y2 = yi + (A/2)*cos(thetai);
xF = xi + (L/1)*cos(thetai);
yF = yi + (L/1)*sin(thetai);
x3 = xF - (A/2)*sin(thetai);
y3 = yF + (A/2)*cos(thetai);
x4 = xF + (A/2)*sin(thetai);
y4 = yF - (A/2)*cos(thetai);
xci = [ x1  x2  x3  x4  x1 ]';
yci = [ y1  y2  y3  y4  y1 ]';
plot(xci,yci,'r','LineWidth',1.5);

x1 = xf + (A/2)*sin(thetaf);
y1 = yf - (A/2)*cos(thetaf);
x2 = xf - (A/2)*sin(thetaf);
y2 = yf + (A/2)*cos(thetaf);
xF = xf + (L/1)*cos(thetaf);
yF = yf + (L/1)*sin(thetaf);
x3 = xF - (A/2)*sin(thetaf);
y3 = yF + (A/2)*cos(thetaf);
x4 = xF + (A/2)*sin(thetaf);
y4 = yF - (A/2)*cos(thetaf);
xcf = [ x1  x2  x3  x4  x1 ]';
ycf = [ y1  y2  y3  y4  y1 ]';
plot(xcf,ycf,'b','LineWidth',1.5);

hold on
```

```
plot(xi,yi,'or','MarkerFaceColor','r')
plot(xf,yf,'ob','MarkerFaceColor','b')

% Puntos Tangentes
pT = [ W11i X11i Y11i Z11i
       W12i X12i Y12i Z12i
       W21i X21i Y21i Z21i
       W22i X22i Y22i Z22i
       W11e X11e Y11e Z11e
       W12e X12e Y12e Z12e
       W21e X21e Y21e Z21e
       W22e X22e Y22e Z22e ];
[nf,~] = size(pT);
N = nf;

% Longitud de un arco: L = alfa*R (alfa en radianes)
%
% Longitud de arcos de posicion inicial: Lai
% Primera columna: longitudes de arco hacia un sentido
% Segunda columna: longitudes de arco hacia otro sentido
Lai = zeros(N,2);
alfaLai = zeros(N,2);
for i=1:N
    for j = 1:2
        C = sqrt((xi-pT(i,2*j-1))^2+(yi-pT(i,2*j))^2);
        alfa = acos(1 - C^2/(2*R^2));
        Lai(i,j) = alfa*R;
        alfaLai(i,j) = alfa;
    end
end

% Longitud de arcos de posicion final: Laf
% Primera columna: longitudes de arco hacia un sentido
% Segunda columna: longitudes de arco hacia otro sentido
Laf = zeros(N,2);
alfaLaf = zeros(N,2);
for i=1:N
    for j = 1:2
        C = sqrt((xf-pT(i,2*(j+2)-1))^2+(yf-pT(i,2*(j+2)))^2);
        alfa = acos(1 - C^2/(2*R^2));
        Laf(i,j) = alfa*R;
        alfaLaf(i,j) = alfa;
    end
end

% Longitudes de Rectas Tangentes: Internas y Externas
Lrt = zeros(N,2);
for i = 1:N
   for j = 1:2
       Lrt(i,j) = sqrt((pT(i,2*j-1)-pT(i,2*j-1+4))^2 + (pT(i,2*j)-
pT(i,2*j+4))^2);
   end
end

[nf,nc] = size(Lai);
path = zeros(nf,nc);
for i = 1:nf
    for j = 1:nc
        path(i,j) = Lai(i,j) + Lrt(i,j) + Laf(i,j);
        fprintf('path(%d,%d)=%f\n',i,j,path(i,j));
    end
end
fprintf('\n');

%% Calculo Minima Distancia
fig = 2;
for k = 1:16
    min = path(1,1);
```

```
minrow = 1;
mincol = 1;
for i = 1:nf
    for j = 1:nc
        if path(i,j) < min
            min = path(i,j);
            minrow = i;
            mincol = j;
        end
    end
end
P1 = pT(minrow,2*mincol-1:2*mincol);
P2 = pT(minrow,2*mincol+3:2*mincol+4);

% Grafica de la Trayetoria Minima
alfai = alfaLai(minrow,mincol);
alfaf = alfaLaf(minrow,mincol);

% Dibujo de la trayectoria
% angL1: angulo entre posicion inicial y el eje X
% angL2: angulo entre punto tangente y el eje X
if minrow == 1 || minrow == 5       %11
    signdeltai = 1;
    signdeltaf = 1;
    cxi = Ci(1);
    cyi = Ci(2);
    cxf = Cf(1);
    cyf = Cf(2);
elseif minrow == 2 || minrow == 6   % 12
    signdeltai = 1;
    signdeltaf = -1;
    cxi = Ci(1);
    cyi = Ci(2);
    cxf = Cf(3);
    cyf = Cf(4);
elseif minrow == 3 || minrow == 7   % 21
    signdeltai = -1;
    signdeltaf = 1;
    cxi = Ci(3);
    cyi = Ci(4);
    cxf = Cf(1);
    cyf = Cf(2);
elseif minrow == 4 || minrow == 8   % 22
    signdeltai = -1;
    signdeltaf = -1;
    cxi = Ci(3);
    cyi = Ci(4);
    cxf = Cf(3);
    cyf = Cf(4);
end

% Posicion inicial
C = sqrt((cxi+R-xi)^2+(cyi-yi)^2);
angL1i = abs(acos(1 - C^2/(2*R^2)));
if yi<cyi
    angL1i = 2*pi-angL1i;       % Si el angulo es por debajo
end
C = sqrt((cxi+R-P1(1))^2+(cyi-P1(2))^2);
angL2i = abs(acos(1 - C^2/(2*R^2)));
if P1(2)<cyi
    angL2i = 2*pi - angL2i;
end

ang1 = angL1i;
ang2 = angL2i;
a = abs(ang1-ang2);
if ang1>ang2
    ang1 = ang1-2*pi;
```

```
elseif ang1<ang2
    ang2 = ang2-2*pi;
end
b = abs(ang1-ang2);

dti = 0;
if a<b
    if angL1i<angL2i
        dti = 0.001;
    elseif angL1i>angL2i
        dti = -0.001;
    end
else
    if angL1i<angL2i
        angL2i = angL2i-2*pi;
        dti = -0.001;
    elseif angL1i>angL2i
        angL1i = angL1i-2*pi;
        dti = 0.001;
    end
end
if dti ~= 0
    phii = angL1i:dti:angL2i; phii = phii';
else
    phii = [angL1i angL2i]';
end

% Posicion final
C = sqrt((cxf+R-xf)^2+(cyf-yf)^2);
angL1f = abs(acos(1 - C^2/(2*R^2)));
if yf<cyf
    angL1f = 2*pi - angL1f;
end
C = sqrt((cxf+R-P2(1))^2+(cyf-P2(2))^2);
angL2f = abs(acos(1 - C^2/(2*R^2)));
if P2(2)<cyf
    angL2f = 2*pi - angL2f;
end

if angL1f<0
    angL1f = 2*pi + angL1f;
end
if angL2f<0
    angL2f = 2*pi + angL2f;
end

ang1 = angL2f;
ang2 = angL1f;
a = abs(ang1-ang2);
if ang1>ang2
    ang1 = ang1-2*pi;
elseif ang1<ang2
    ang2 = ang2-2*pi;
end
b = abs(ang1-ang2);

dtf = 0;
if a<b
    if angL2f<angL1f
        dtf = 0.001;
    elseif angL2f>angL1f
        dtf = -0.001;
    end
else
    if angL2f<angL1f
        angL1f = angL1f-2*pi;
        dtf = -0.001;
    elseif angL2f>angL1f
```

```
        angL2f = angL2f-2*pi;
        dtf = 0.001;
    end
end

if dtf ~= 0
    phif = angL2f:dtf:angL1f; phif = phif';
else
    phif = [angL2f angL1f]';
end

% VELOCIDADES
if minrow == 1 || minrow == 5       % 11
    signVel(1) = 1;
    if dti<0
        signVel(1) = -1;
    end
    signVel(3) = 1;
    if dtf<0
        signVel(3) = -1;
    end
elseif minrow == 2 || minrow == 6   % 12
    signVel(1) = 1;
    if dti<0
        signVel(1) = -1;
    end
    signVel(3) = -1;
    if dtf<0
        signVel(3) = 1;
    end
elseif minrow == 3 || minrow == 7   % 21
    signVel(1) = -1;
    if dti<0
        signVel(1) = 1;
    end
    signVel(3) = 1;
    if dtf<0
        signVel(3) = -1;
    end
elseif minrow == 4 || minrow == 8   % 22
    signVel(1) = -1;
    if dti<0
        signVel(1) = 1;
    end
    signVel(3) = -1;
    if dtf<0
        signVel(3) = 1;
    end
end
if thetai == pi
    signVel(1) = -1*signVel(1);
end

angLi = phii(end) - phii(1);
thetarti = thetai + angLi;

if(thetarti<0)
    thetarti = 2*pi + thetarti;
end

thetam = atan2(P2(2)-P1(2),P2(1)-P1(1));    % Pendiente recta central
if (round(1000*thetarti)/1000 == round(1000*thetam)/1000)
    signVel(2) = 1;
else
    signVel(2) = -1;
end

% Trayectoria
```

```
Laix = R*cos(phii)+cxi;
Laiy = R*sin(phii)+cyi;

st = 0.001;
Nrt = floor(max((P2(1)-P1(1))/st,(P2(2)-P1(2))/st));
Lrtx = linspace(P1(1),P2(1),Nrt); Lrtx = Lrtx';
Lrty = linspace(P1(2),P2(2),Nrt); Lrty = Lrty';

Lafx = R*cos(phif)+cxf;
Lafy = R*sin(phif)+cyf;

trayectoria = [ Laix Laiy
                Lrtx Lrty
                Lafx Lafy ];

figure(fig)
hold on
plot(trayectoria(:,1),trayectoria(:,2),'--m','LineWidth',0.7)
plot(xci,yci,'r','LineWidth',1.5)
plot(xi,yi,'or','MarkerFaceColor','r')
plot(xcf,ycf,'b','LineWidth',1.5)
plot(xf,yf,'ob','MarkerFaceColor','b')
plot(P1(1),P1(2),'*g')
plot(P2(1),P2(2),'*g')
axis equal
grid
xlabel('x (m)'), ylabel('y (m)')
title('Mínima Trayectoria Posible')
hold off
fig = fig + 1;

% Analizo si llega a la posicion final con el angulo correcto
theta = thetai;
if theta == 2*pi
    theta = 0;
end

if dti>0
    theta = theta+alfai;
elseif dti<0
    theta = theta-alfai;
end
if theta < 0
    theta = 2*pi+theta;
end
if theta > 2*pi
    theta = theta - 2*pi;
end
angrt = theta;

if dtf>0
    theta = theta+alfaf;
elseif dtf<0
    theta = theta-alfaf;
end
if theta < 0
    theta = 2*pi+theta;
end
if theta > 2*pi
    theta = theta - 2*pi;
end

angdes = round(1000*thetaf)/1000;
angreal = round(1000*theta)/1000;
if angdes == 2*pi
    angdes = 0;
elseif angreal == round(1000*2*pi)/1000
    angreal = 0;
```

```
    end

    fprintf('k = %d\n',k);
    fprintf('Punto P1 : %f,%f\n',P1(1),P1(2));
    fprintf('Punto P2 : %f,%f\n',P2(1),P2(2));
    fprintf('Angulos  : %f,%f\n',alfai*180/pi,alfaf*180/pi);
    fprintf('Min Path : %f\n',min);
    fprintf('Velocidad: %d::%d::%d\n',signVel(1),signVel(2),signVel(3));
    fprintf('angLi    : %f::%f\n',angL1i*180/pi,angL2i*180/pi);
    fprintf('angLf    : %f::%f\n',angL2f*180/pi,angL1f*180/pi);
    fprintf('Check    : %f::%f\n\n',theta*180/pi,thetaf*180/pi);

    if angreal == angdes
        break;
    end
    path(minrow,mincol) = 100;
end


% PORCENTAJES DE TIEMPO (para el caso de puntos de trayectoria en base a
polinomios)
n1 = alfai*R;
n2 = sqrt((P1(1)-P2(1))^2 + (P1(2)-P2(2))^2);
n3 = alfaf*R;
nt = n1+n2+n3;

porcT(1) = n1/nt;
porcT(2) = n2/nt;
porcT(3) = n3/nt;

figure(1)
plot(P1(1),P1(2),'og','MarkerFaceColor','g')
plot(P2(1),P2(2),'og','MarkerFaceColor','g')
plot(trayectoria(:,1),trayectoria(:,2),'-m','LineWidth',1.5)
axis equal
grid
xlabel('x (m)'),ylabel('y (m)')

figure(2)
title('Minima Trayectoria')
xlabel('x (m)')
ylabel('y (m)')
grid
shg

save minpath P1 P2 cxi cyi cxf cyf xi yi xf yf thetai thetaf angrt
signdeltai signdeltaf signVel porcT deltamax trayectoria L A
```

A.2. Cálculo de Rectas Tangentes

```
function [ W,X,Y,Z ] = TangenteInterior(cx1,cy1,r1,cx2,cy2,r2)
% Calculo de recta tangente interior
phi = 0:0.001:2*pi;
N = 100;

pos = 0;
if cx2 == cx1
    pos = 1;    % Vertical
```

```
elseif cy2 == cy1
    pos = 2;    % Horizontal
end

% 1) Recta cr - cR
if pos == 1
    a1 = (cx2-cx1)/(cy2-cy1);
    b1 = cx2 - a1*cy2;
    y1 = linspace(cy2,cy1,N);
    x1 = a1*y1 + b1;
else
    a1 = (cy2-cy1)/(cx2-cx1);
    b1 = cy2 - a1*cx2;
    x1 = linspace(cx2,cx1,N);
    y1 = a1*x1 + b1;
end

% 2) Recta perpendicular a la recta para cada circunferencia
%    P,Q -> r
%    S,T -> R

if pos == 1
    % P-Q
    p1x = cx1 - r1;
    p1y = cy1;
    p2x = cx1 + r1;
    p2y = cy1;
    P = [p1x,p1y];
    Q = [p2x,p2y];

    % S-T
    p1x = cx2 - r2;
    p1y = cy2;
    p2x = cx2 + r2;
    p2y = cy2;
    S = [p1x,p1y];
    T = [p2x,p2y];
elseif pos == 2
    % P-Q
    p1x = cx1;
    p1y = cy1 - r1;
    p2x = cx1;
    p2y = cy1 + r1;
    P = [p1x,p1y];
    Q = [p2x,p2y];

    % S-T
    p1x = cx2;
    p1y = cy2 - r2;
    p2x = cx2;
    p2y = cy2 + r2;
    S = [p1x,p1y];
    T = [p2x,p2y];
else
    % P-Q
    a2 = -1/a1;
    b2 = -a2*cx1+cy1;
    [p1x,p1y,p2x,p2y] = Cruce_RectaCirculo(a2,b2,cx1,cy1,r1);
    P = [p1x,p1y];
    Q = [p2x,p2y];

    % S-T
    a2 = -1/a1;
    b2 = -a2*cx2+cy2;
    [p1x,p1y,p2x,p2y] = Cruce_RectaCirculo(a2,b2,cx2,cy2,r2);
    S = [p1x,p1y];
    T = [p2x,p2y];
end
```

```
% 3) Unir P-T y Q-S
x2 = linspace(P(1),T(1),N);
y2 = linspace(P(2),T(2),N);

x3 = linspace(Q(1),S(1),N);
y3 = linspace(Q(2),S(2),N);

% 4)Armar circulo con diametro N-V y O-V
% Recta Central
a1 = (y1(2)-y1(1))/(x1(2)-x1(1));
b1 = y1(3)-a1*x1(3);
% Recta P-T
a2 = (y2(2)-y2(1))/(x2(2)-x2(1));
b2 = y2(3)-a2*x2(3);
% Recta Q-S
a3 = (y3(2)-y3(1))/(x3(2)-x3(1));
b3 = y3(3)-a3*x3(3);

% Calculo el punto V a partir de intersección de rectas
if pos == 1
    px = (b3-b2)/(a2-a3);
    py = a2*px + b2;
else
    px = (b2-b1)/(a1-a2);
    py = a1*px + b1;
end
V = [px,py];

% Calculo punto centro entre N-V
cv1x = (cx1 + V(1))/2;
cv1y = (cy1 + V(2))/2;
rv1 = sqrt((abs(cx1 - V(1))/2)^2 + (abs(cy1 - V(2))/2)^2);

% Calculo punto centro entre O-V
cv2x = (cx2 + V(1))/2;
cv2y = (cy2 + V(2))/2;
rv2 = sqrt((abs(cx2 - V(1))/2)^2 + (abs(cy2 - V(2))/2)^2);

% 5) Intersecciones de circulos
% Busco puntos
[px1,py1,px2,py2] = Cruce_CirculoCirculo(cx1,cy1,r1,cv1x,cv1y,rv1);
W = [px1,py1];
X = [px2,py2];

[px1,py1,px2,py2] = Cruce_CirculoCirculo(cx2,cy2,r2,cv2x,cv2y,rv2);
Z = [px1,py1];
Y = [px2,py2];

% Grafico Circulos con Tangentes Interiores
figure(1)
hold on
x = r1*cos(phi)+cx1;
y = r1*sin(phi)+cy1;
plot(x,y,'k')
x = r2*cos(phi)+cx2;
y = r2*sin(phi)+cy2;
plot(x,y,'k')

% W-Y
x = linspace(W(1),Y(1),N);
y = linspace(W(2),Y(2),N);
plot(x,y,'k')

% X-Z
x = linspace(X(1),Z(1),N);
y = linspace(X(2),Z(2),N);
plot(x,y,'k')
```

```
function [W,X,Y,Z] = TangenteExterior(cx1,cy1,r1,cx2,cy2,r2)
% Cálculo de recta tangente exterior
phi = 0:0.001:2*pi;
N = 100;

% 1) Recta N-O y punto medio H
px = (cx1+cx2)/2;
py = (cy1+cy2)/2;
H = [ px,py ];
rH = sqrt((px-cx1)^2+(py-cy1)^2);

if r1 == r2
    m = (cy2-cy1)/(cx2-cx1);
    if m == 0
        W = [cx1,cy1+r1];
        X = [cx1,cy1-r1];
        Y = [cx2,cy2+r2];
        Z = [cx2,cy2-r2];
    elseif m == inf
        W = [cx1+r1,cy1];
        X = [cx1-r1,cy1];
        Y = [cx2+r2,cy2];
        Z = [cx2-r2,cy2];
    else
        m = -1/m;
        b1 = cy1-m*cx1;
        [px1,py1,px2,py2] = Cruce_RectaCirculo(m,b1,cx1,cy1,r1);
        W = [px1,py1];
        X = [px2,py2];

        b2 = cy2-m*cx2;
        [px1,py1,px2,py2] = Cruce_RectaCirculo(m,b2,cx2,cy2,r2);
        Y = [px1,py1];
        Z = [px2,py2];
    end
else
    % 2) Circunferencia concentrica (mayor)
    %    radio = r2-r1 (r2>r1)
    rc = abs(r2-r1);
    if r2>r1
        cxc = cx2;
        cyc = cy2;
    else
        cxc = cx1;
        cyc = cy1;
    end

    % 3) Circunferencia centro H corta circunferencia concentrica
    %    en P y Q
    [px1,py1,px2,py2] = Cruce_CirculoCirculo(cxc,cyc,rc,H(1),H(2),rH);
    P = [px1,py1];
    Q = [px2,py2];

    % 4) Rectas del centro a P y Q y formar los puntos W y X
    %    que cruzan la circunferencia mayor
    if cx1 == cxc && cy1 == cyc
        R = r1;
    elseif cx2 == cxc && cy2 == cyc
        R = r2;
    end

    % Punto W
    m1 = (P(2)-cyc)/(P(1)-cxc);
    b1 = P(2)-m1*P(1);
```

```
    [px1,py1,px2,py2] = Cruce_RectaCirculo(m1,b1,cxc,cyc,R);
    d1 = sqrt((P(1)-px1)^2+(P(2)-py1)^2);
    d2 = sqrt((P(1)-px2)^2+(P(2)-py2)^2);
    if d1 < d2
        W = [px1,py1];
    else
        W = [px2,py2];
    end

    % Punto X
    m2 = (Q(2)-cyc)/(Q(1)-cxc);
    b2 = Q(2)-m2*Q(1);
    [px1,py1,px2,py2] = Cruce_RectaCirculo(m2,b2,cxc,cyc,R);
    d1 = sqrt((Q(1)-px1)^2+(Q(2)-py1)^2);
    d2 = sqrt((Q(1)-px2)^2+(Q(2)-py2)^2);
    if d1 < d2
        X = [px1,py1];
    else
        X = [px2,py2];
    end

    % 5) Se trazan radios paralelos a las rectas de W y X
    %    y cruzan Y y Z
    if R == r1
        cxnc = cx2;
        cync = cy2;
        Rn = r2;
    elseif R == r2
        cxnc = cx1;
        cync = cy1;
        Rn = r1;
    end

    % Punto Y
    m = m1;
    b = cync - m*cxnc;
    [px1,py1,px2,py2] = Cruce_RectaCirculo(m,b,cxnc,cync,Rn);
    d1 = sqrt((W(1)-px1)^2+(W(2)-py1)^2);
    d2 = sqrt((W(1)-px2)^2+(W(2)-py2)^2);
    if d1 < d2
        Y = [px1,py1];
    else
        Y = [px2,py2];
    end

    % Punto Z
    m = m2;
    b = cync - m*cxnc;
    [px1,py1,px2,py2] = Cruce_RectaCirculo(m,b,cxnc,cync,Rn);
    d1 = sqrt((X(1)-px1)^2+(X(2)-py1)^2);
    d2 = sqrt((X(1)-px2)^2+(X(2)-py2)^2);
    if d1 < d2
        Z = [px1,py1];
    else
        Z = [px2,py2];
    end
end

% Grafico Circulos con Tangentes Exteriores
figure(1)
hold on
x = r1*cos(phi)+cx1;
y = r1*sin(phi)+cy1;
plot(x,y,'k')
x = r2*cos(phi)+cx2;
y = r2*sin(phi)+cy2;
plot(x,y,'k')
```

```
% W-Y
x = linspace(W(1),Y(1),N);
y = linspace(W(2),Y(2),N);
plot(x,y,'k')

% X-Z
x = linspace(X(1),Z(1),N);
y = linspace(X(2),Z(2),N);
plot(x,y,'k')
```

A.3. Control del Robot con Trayectoria en Base a Polinomios

```
clc
close all
clear all

m = menu('Ganancias del Controlador',...
         'Generar nuevos datos','Cargar datos almacenados');
if m == 1
    ley_de_control
end

load Gain                  % Cargo ganancias del controlador
load minpath

dt = 0.005;
vmax = 2;

if thetai>pi; thetai = thetai - 2*pi; end
if thetaf>pi; thetaf = thetaf - 2*pi; end
if angrt>pi; angrt = angrt - 2*pi; end

tpf = 10;
tp1 = porcT(1)*tpf;
tp2 = tp1 + porcT(2)*tpf;
tp3 = tp2 + porcT(3)*tpf;
tp = [0 tp1 tp2 tpf ];
tp = [0 5 10 15];
N = length(tp);

% Valores iniciales y finales
xv = [ xi P1(1) P2(1) xf ];
yv = [ yi P1(2) P2(2) yf ];
vv = [ 0.001 signVel(1)*0.001 signVel(2)*0.001 signVel(3)*0.001 ];

vpv = zeros(1,N);
wv = zeros(1,N);
deltav = zeros(1,N);

L = 0.190;                 % Longitud del vehiculo
A = 0.143;                 % Ancho del vehiculo
R = L/tan(deltamax); % Radio formado por el carro para un delta maximo

FF1d = 0;
FF2d = 0;
FF1dp = 0;
FF2dp = 0;
x = xi;
y = yi;
```

```
theta = thetai;

thetavi = thetai;
thetavf = angrt;    % Simulación Final

xw(1,1) = xi;
yw(1,1) = yi;
xwd(1,1) = xi;
ywd(1,1) = yi;
k = 1;

disp('inicio..')

for n = 1:N-1
    if n == 3
        thetavf = thetaf;
    end

    % Sentido de direccion del movil
    if (vv(n+1)>0)
        vv(n) = abs(vv(n));
    elseif (vv(n+1)<0)
        vv(n) = -abs(vv(n));
    end

    if (vv(n)>0 && vv(n+1)>0)
        sentido = 1;    % positivo
    elseif (vv(n)<0 && vv(n+1)<0)
        sentido = 0;    % negativo
    end

    ti = tp(n);
    tf = tp(n+1);
    % Genero la trajectoria
    [F1d,F1dp,F1dpp,F2d,F2dp,F2dpp] = trajectory_path([xv(n) xv(n+1)],[yv(n)
yv(n+1)],[thetavi thetavf],[vv(n) vv(n+1)],[vpv(n) vpv(n+1)],[deltav(n)
deltav(n+1)],[ti dt tf],L);

    v = vv(n);
    w = wv(n);
    delta = deltav(n);
    F1 = x;
    F1p = v*cos(theta);
    F2 = y;
    F2p = v*sin(theta);

    m = 1;
    for tt = ti:dt:tf
        posx(k,1) = x;
        posy(k,1) = y;
        angtheta(k,1) = theta;
        angdelta(k,1) = delta;
        vel(k,1) = v;
        velang(k,1)= w;
        t(k,1) = tt;

        % Ley de control
        err1 = F1d(m,1) - F1;    err1p = F1dp(m,1) - F1p;
        err2 = F2d(m,1) - F2;    err2p = F2dp(m,1) - F2p;
        eps1 = F1dpp(m,1) + p(1)*err1p + p(2)*err2p + p(3)*err1 + p(4)*err2;
        eps2 = F2dpp(m,1) + q(1)*err1p + q(2)*err2p + q(3)*err1 + q(4)*err2;

        error1(k,1) = err1;
        error2(k,1) = err2;

        vp = eps1*cos(theta) + eps2*sin(theta);
        v = vp*dt + v;
        tandelta = L/(v*v)*(-eps1*sin(theta)+eps2*cos(theta));
```

```
        % Saturando la velocidad
        if (v>vmax)
            v = vmax;
        elseif (v<-vmax)
            v = -vmax;
        end

        % Calculo del angulo de timon
        delta =  atan2(tandelta,1);
        if delta>deltamax
            delta = deltamax;
        elseif delta <-deltamax
            delta = -deltamax;
        end

        % Modelo carro
        xp = v*cos(theta);
        yp = v*sin(theta);
        thetap = v/L*tan(delta);

        % Variables medibles
        x = xp*dt + x;
        y = yp*dt + y;
        theta = thetap*dt + theta;
        if theta > 2*pi || theta > pi
            theta = theta - 2*pi;
        end
        if theta < -pi
            theta = theta + 2*pi;
        end

        errorT(m,1) = sqrt((F1d(m,1)-x)^2+(F2d(m,1)-y)^2);

        F1p = xp;
        F2p = yp;
        F1 = x;
        F2 = y;

        k = k+1;
        m = m+1;
    end
    FF1d = [ FF1d F1d' ];
    FF2d = [ FF2d F2d' ];

    FF1dp = [ FF1dp F1dp'];
    FF2dp = [ FF2dp F2dp'];

    xw(n+1,1) = posx(end);
    yw(n+1,1) = posy(end);
    xwd(n+1,1) = F1d(end);
    ywd(n+1,1) = F2d(end);

    thetavi = thetavf;

    if (n==3);break;end
end

FF1d = FF1d(2:end);
FF2d = FF2d(2:end);
FF1dp = FF1dp(2:end);
FF2dp = FF2dp(2:end);

% Error
err = sqrt((posx(end)-xf)^2+(posy(end)-yf)^2+(angtheta(end)-thetaf)^2)

[xwd xw ywd yw]
[theta thetaf]*180/pi
```

```
angtheta = angtheta*180./pi;
angdelta = angdelta*180./pi;

%%
close all

figure(1)
plot(posx,posy,'-b'),grid
title('Plano X-Y')
ylabel('y (m)')
xlabel('x (m)')
hold on
plot(xi,yi,'og',xf,yf,'og')
text(xi,yi-0.1,'INICIO')
text(xf,yf+0.2,'FINAL')
hold off
axis equal

rx = xf*ones(size(posx));
ry = yf*ones(size(posy));
rtheta = thetaf*180/pi*ones(size(angtheta));

figure(2)
subplot(3,1,1),plot(t,posx,'-b',t,FF1d,'--r',t,rx,'g'),ylabel('x (m)'),grid
subplot(3,1,2),plot(t,posy,'-b',t,FF2d,'--r',t,ry,'g'),ylabel('y (m)'),grid
legend('Valor Real','Valor Deseado','Valor Final','Location','best')
subplot(3,1,3),plot(t,angtheta,'-b',t,rtheta,'g'),grid,
ylabel('\theta (deg)'),ylim([-180 180])
xlabel('Tiempo [seg]')
title(subplot(3,1,1),'Señales de las Variables de Estado')

figure(3)
subplot(2,1,1),plot(t,vel,'b'),grid,ylabel('Velocidad (m/s)')
subplot(2,1,2),plot(t,angdelta,'b'),grid
ylabel('\delta (deg)'),ylim([-45 45])
xlabel('Tiempo [seg]')
title(subplot(2,1,1),'Señales de Control')

figure(4)
plot(posx,posy,'-b'),grid
title('Plano X-Y')
ylabel('y (m)')
xlabel('x (m)')
axis equal
hold on

plot(xw,yw,'og',xwd,ywd,'*g')
legend('Trayectoria','Puntos Reales','Puntos Deseados','Location',
'southeast')

xr = linspace(P1(1),P2(1),100);
yr = linspace(P1(2),P2(2),100);
plot(xr,yr,':r')

phi = 0:0.01:2*pi;
x = R*cos(phi);
y = R*sin(phi);
plot(x+cxi,y+cyi,':r')

plot(x+cxf,y+cyf,':r')
plot(cxi,cyi,'*r',cxf,cyf,'*r');
hold off

figure(5)
subplot(2,1,1),plot(t,FF1d,'r'),grid,ylabel('x* (m)'),hold on
xwd = [xi P1(1) P2(1) xf];
plot(tp,xwd,'dg'),hold off
```

```
xlabel('Tiempo [seg]')

subplot(2,1,2),plot(t,FF2d,'r'),grid,ylabel('y* (m)'),hold on
ywd = [yi P1(2) P2(2) yf];
plot(tp,ywd,'dg'),hold off
xlabel('Tiempo [seg]')
title(subplot(2,1,1),'Posición Deseada de cada Salida Flat')

figure(6)
subplot(2,1,1),plot(t,FF1dp,'r'),grid,ylabel('xp* (m)')
subplot(2,1,2),plot(t,FF2dp,'r'),grid,ylabel('yp* (m)')
xlabel('Tiempo [seg]')
title(subplot(2,1,1),'Velocidad Deseada de cada salida Flat')

figure(7)
title('Trayectoria del Robot Móvil Tipo Ackerman');
xlabel('x (m)')
ylabel('y (m)')
hold on
axis([ -1 5 -1 5 ]);

writeObj=VideoWriter('tesis2.avi');
writeObj=VideoWriter('prueba.avi');
writeObj.FrameRate=5;
open(writeObj);

countmax = k-1;
for count = 1:3*1/(4*dt):countmax
   xz = posx(count,1);
   yz = posy(count,1);
   Pz = angtheta(count,1)*pi/180;
   x1 = xz + (A/2)*sin(Pz);
   y1 = yz - (A/2)*cos(Pz);
   x2 = xz - (A/2)*sin(Pz);
   y2 = yz + (A/2)*cos(Pz);
   xF = xz + L*cos(Pz);
   yF = yz + L*sin(Pz);

   x3 = xF - (A/2)*sin(Pz);
   y3 = yF + (A/2)*cos(Pz);
   x4 = xF + (A/2)*sin(Pz);
   y4 = yF - (A/2)*cos(Pz);
   xc = [ x1  x2  x3  x4  x1 ]';
   yc = [ y1  y2  y3  y4  y1 ]';

   figure(5)
   plot(xc,yc,'r','LineWidth',1)
   plot(xz,yz,'or','MarkerSize',6,'MarkerFaceColor','r')
   axis equal

   frame=getframe(gcf);
   writeVideo(writeObj, frame);

   pause(1/12);
   plot(xc,yc,'w','LineWidth',2)
   plot(xz,yz,'ow','MarkerSize',6,'MarkerFaceColor','w')
n
hold off
grid
close(writeObj);

plot(xc,yc,'r','LineWidth',2)
plot(xz,yz,'or','MarkerSize',6,'MarkerFaceColor','r')
axis equal
axis([-1 5 -1 5])

figure(8)
plot(t,error1,'r',t,error2,'b'),grid
```

```
xlabel('Tiempo [seg]')
legend('F1d - F1','F2d - F2')
title('Errores de cada Salida Flat con respecto a la Deseada')
```

A.4. Control del Robot con Trayectoria en Base a Funciones Switching

```
clc
close all
clear all

m = menu('Ganancias del Controlador',...
         'Generar nuevos datos','Cargar datos almacenados');
if m == 1
    ley_de_control
end

global cxi cyi cxf cyf
global R angrt
global xi xf yi yf
global P1 P2

load Gain
load minpath

if thetai>pi; thetai = thetai - 2*pi; end
if thetaf>pi; thetaf = thetaf - 2*pi; end
if angrt>pi; angrt = angrt - 2*pi; end

dt = 0.005;
vmax = 2;

x = xi;
y = yi;
theta = thetai;
delta = 0;
v = 0.001;

disp('inicio..')

k = 1;
tt = 0;
while(1)
    posx(k,1) = x;
    posy(k,1) = y;
    angtheta(k,1) = theta;
    angdelta(k,1) = delta;
    vel(k,1) = v;
    t(k,1) = tt;

    % Calculando DELTA
    df = deltaFunction(x,y,theta);
    if df == 0
        delta = 0;
    elseif df > 0
        delta = deltamax;
    elseif df < 0
        delta = -deltamax;
    end

    % Calculando VELOCIDAD
    [vf,~] = velFunction(x,y,theta,delta);
    if vf > 0
        v = vmax;
    elseif vf < 0
```

```
        v = -vmax;
    end

    % Modelo carro
    xp = v*cos(theta);
    yp = v*sin(theta);
    thetap = v/L*tan(delta);

    % Variables medibles
    x = xp*dt + x;
    y = yp*dt + y;
    theta = thetap*dt + theta;
    if theta > 2*pi || theta > pi
        theta = theta - 2*pi;
    end
    if theta < -pi
        theta = theta + 2*pi;
    end

    k = k + 1;
    tt = tt + dt;

    err = sqrt((x-xf)^2+(y-yf)^2+(theta-thetaf)^2);
    error(k,1) = err;
    if err < 0.01
        break
    end
end


%% Generar vector de trayectoria
F1d = posx';
F1dp = diff(F1d)./dt;   F1dp = [0 F1dp];
F1dpp = diff(F1dp)./dt; F1dpp = [0 F1dpp];

F2d = posy';
F2dp = diff(F2d)./dt;   F2dp = [0 F2dp];
F2dpp = diff(F2dp)./dt; F2dpp = [0 F2dpp ];

N = length(F1d);


% Condiciones iniciales para el control
x = xi;
y = yi;
theta = thetai;
v = 0.001;
w = 0;
delta = 0;

F1 = x;
F1p = v*cos(theta);
F2 = y;
F2p = v*sin(theta);

k = 1;
tt = 0;
% Lazo de control
for k = 1:N
    posx(k,1) = x;
    posy(k,1) = y;
    angtheta(k,1) = theta;
    angdelta(k,1) = delta;
    vel(k,1) = v;
    velang(k,1)= w;
    t(k,1) = tt;

    % Ley de control
```

```
    err1 = F1d(k) - F1;    err1p = F1dp(k) - F1p;
    err2 = F2d(k) - F2;    err2p = F2dp(k) - F2p;
    eps1 = F1dpp(k) + p(1)*err1p + p(2)*err2p + p(3)*err1 + p(4)*err2;
    eps2 = F2dpp(k) + q(1)*err1p + q(2)*err2p + q(3)*err1 + q(4)*err2;

    error1(k,1) = err1;
    error2(k,1) = err2;

    vp = eps1*cos(theta) + eps2*sin(theta);
    u = L/(v*v)*(-eps1*sin(theta)+eps2*cos(theta));

    % Cálculo de la velocidad
    v = vp*dt + v;
    if (v>vmax)
        v = vmax;
    elseif (v<-vmax)
        v = -vmax;
    end

    % Calculo del angulo del timon
    delta = atan2(u,1);
    if delta > deltamax
        delta = deltamax;
    elseif delta < -deltamax
        delta = -deltamax;
    end

    % Modelo carro
    xp = v*cos(theta);
    yp = v*sin(theta);
    thetap = v/L*tan(delta);

    % Variables medibles
    x = xp*dt + x;
    y = yp*dt + y;
    theta = thetap*dt + theta;
    if theta > pi
        theta = theta - 2*pi;
    end
    if theta < -pi
        theta = theta + 2*pi;
    end

    F1p = xp;
    F2p = yp;
    F1 = x;
    F2 = y;

    tt = tt + dt;
end
err = sqrt((posx(end)-FF1d(end))^2+(posy(end)-FF2d(end))^2+(angtheta(end)-
thetaf)^2)

error_x = x - xf
error_y = y - yf
error_theta = (theta - thetaf)*180/pi

angtheta = angtheta*180./pi;
angdelta = angdelta*180./pi;

%%
close all

figure(1)
plot(posx,posy,'-b'),grid
hold on
plot(trayectoria(:,1),trayectoria(:,2),'--k'),grid
title('Plano X-Y')
```

```
ylabel('y (m)')
xlabel('x (m)')
plot(xi,yi,'og',xf,yf,'og')
text(xi,yi-0.1,'INICIO')
text(xf,yf+0.2,'FINAL')
hold off
axis equal

rx = xf*ones(size(posx));
ry = yf*ones(size(posy));
rtheta = thetaf*180/pi*ones(size(angtheta));

figure(2)
subplot(3,1,1),plot(t,posx,'-b',t,F1d,'--r',t,rx,'g')
ylabel('x (m)'),grid,xlim([0 tt])
subplot(3,1,2),plot(t,posy,'-b',t,F2d,'--r',t,ry,'g')
ylabel('y (m)'),grid,xlim([0 tt])
legend('Valor Real','Valor Deseado','Valor Final','Location','best')

subplot(3,1,3),plot(t,angtheta,'-b',t,rtheta,'g'),grid
ylabel('\theta (deg)'),ylim([-180 180]),xlim([0 tt])
xlabel('Tiempo [seg]')
title(subplot(3,1,1),'Señales de las Variables de Estado')

figure(3)
subplot(2,1,1),plot(t,vel,'b'),grid,ylabel('Velocidad (m/s)')
ylim([-2.5 2.5]),,xlim([0 tt])
subplot(2,1,2),plot(t,angdelta,'b'),grid,ylabel('\delta (deg)')
ylim([-20 20]),,xlim([0 tt])
xlabel('Tiempo [seg]')
title(subplot(2,1,1),'Señales de Control')
close

figure(4)
title('Trayectoria Real del Robot Móvil Tipo Ackerman');
xlabel('x (m)')
ylabel('y (m)')
hold on

countmax = k-1;
for count = 1:50:countmax
   xz = posx(count,1);
   yz = posy(count,1);
   Pz = angtheta(count,1)*pi/180;

   x1 = xz + (A/2)*sin(Pz);
   y1 = yz - (A/2)*cos(Pz);
   x2 = xz - (A/2)*sin(Pz);
   y2 = yz + (A/2)*cos(Pz);
   xF = xz + L*cos(Pz);
   yF = yz + L*sin(Pz);

   x3 = xF - (A/2)*sin(Pz);
   y3 = yF + (A/2)*cos(Pz);
   x4 = xF + (A/2)*sin(Pz);
   y4 = yF - (A/2)*cos(Pz);
   xc = [ x1  x2  x3  x4  x1 ]';
   yc = [ y1  y2  y3  y4  y1 ]';

   plot(xc,yc,'r','LineWidth',1)
   plot(xz,yz,'or','MarkerSize',4,'MarkerFaceColor','r')
end

xz = posx(end,1);
yz = posy(end,1);
Pz = angtheta(end,1)*pi/180;

x1 = xz + (A/2)*sin(Pz);
```

```
y1 = yz - (A/2)*cos(Pz);
x2 = xz - (A/2)*sin(Pz);
y2 = yz + (A/2)*cos(Pz);
xF = xz + L*cos(Pz);
yF = yz + L*sin(Pz);

x3 = xF - (A/2)*sin(Pz);
y3 = yF + (A/2)*cos(Pz);
x4 = xF + (A/2)*sin(Pz);
y4 = yF - (A/2)*cos(Pz);
xc = [ x1  x2  x3  x4  x1 ]';
yc = [ y1  y2  y3  y4  y1 ]';

plot(xc,yc,'r','LineWidth',1)
plot(xz,yz,'or','MarkerSize',4,'MarkerFaceColor','r')

plot(trayectoria(:,1),trayectoria(:,2),'-k','LineWidth',1),grid

plot(xi,yi,'dg','MarkerFaceColor','g','MarkerSize',4)
plot(xf,yf,'dg','MarkerFaceColor','g','MarkerSize',4)
text(xi+0.1,yi-0.05,'INICIO')
text(xf+0.10,yf,'FIN')

hold off
grid on

axis equal
```

Anexo B: Algoritmos para la Implementación

B.1. Calibración de Sensores

```
// CALIBRACION DEL ACELEROMETRO Y GIROSCOPIO
void calibraMPU9265(void){
        // Variables utilizadas por el filtro pasa bajos
        long f_ax,f_ay,f_az;
        int p_ax,p_ay,p_az;
        long f_gx,f_gy,f_gz;
        int p_gx,p_gy,p_gz;
        int counter = 0;
        int temp[3];

        int ax,ay,az,gx,gy,gz;
        int axo,ayo,azo;
        int gxo,gyo,gzo;

        readAccelOffset(temp);
        axo = temp[0];
        ayo = temp[1];
        azo = temp[2];

        readGyroOffset(temp);
        gxo = temp[0];
        gyo = temp[1];
        gzo = temp[2];
        printf("OFFSETS actuales: %d\t%d\t%d\t\t%d\t%d\t%d\n\n",axo,ayo,azo,gxo,gyo,gzo);

        while(1){
                readAccel(temp);
                ax = temp[0];
                ay = temp[1];
                az = temp[2];
                readGyro(temp);
                gx = temp[0];
                gy = temp[1];
                gz = temp[2];
```

```
// Filtrar las lecturas
f_ax = f_ax - (f_ax>>5) + ax;
p_ax = f_ax>>5;
f_ay = f_ay - (f_ay>>5) + ay;
p_ay = f_ay>>5;
f_az = f_az - (f_az>>5) + az;
p_az = f_az>>5;

f_gx = f_gx - (f_gx>>3) + gx;
p_gx = f_gx>>3;
f_gy = f_gy - (f_gy>>3) + gy;
p_gy = f_gy>>3;
f_gz = f_gz - (f_gz>>3) + gz;
p_gz = f_gz>>3;

// Cada 100 lecturas corregir el offset
if (counter == 100){
        // Mostrar lecturas
        printf("promedio:\t");
        printf("%d\t%d\t%d\t\t%d\t%d\t%d\n",p_ax,p_ay,p_az,p_gx,p_gy,p_gz);

        // Calibrar el acelerometro a 1g en el eje z (ajuste del offset)
        if (p_ax>0) axo--;
        else {axo++;}
        if (p_ay>0) ayo--;
        else {ayo++;}
        if (p_az-16384>0) azo--;
        else {azo++;}

        writeByte(fd,AX_OFFSET_H,axo>>8 & 0xff);
        writeByte(fd,AX_OFFSET_L,axo & 0xff);
        writeByte(fd,AY_OFFSET_H,ayo>>8 & 0xff);
        writeByte(fd,AY_OFFSET_L,ayo & 0xff);
        writeByte(fd,AZ_OFFSET_H,azo>>8 & 0xff);
        writeByte(fd,AZ_OFFSET_L,azo & 0xff);

        // Calibrar el gyroscopio a 0°/s en todos los ejes (ajuste del offset)
        if (p_gx>0) gxo--;
        else {gxo++;}
        if (p_gy>0) gyo--;
        else {gyo++;}
```

```
                    if (p_gz>0) gzo--;
                    else {gzo++;}

                    writeByte(fd,GX_OFFSET_H,gxo>>8 & 0xff);
                    writeByte(fd,GX_OFFSET_L,gxo & 0xff);
                    writeByte(fd,GY_OFFSET_H,gyo>>8 & 0xff);
                    writeByte(fd,GY_OFFSET_L,gyo & 0xff);
                    writeByte(fd,GZ_OFFSET_H,gzo>>8 & 0xff);
                    writeByte(fd,GZ_OFFSET_L,gzo & 0xff);
                    counter = 0;
                }
                counter++;
        }
}

// CALIBRACION MAGNETOMETRO
void calibAK8963(float* dest1, float* dest2){
        cout<<"Calibracion Magnetometro\n";
        int mx,my,mz;
        uint16_t sample_count = 0;
        int32_t mag_bias[3] = {0,0,0}, mag_scale[3] = {0,0,0};
        int16_t mag_max[3] = {-32767,-32767,-32767}, mag_min[3] = {32767,32767,32767}, mag_temp[3] =
{0,0,0};

        delay(4000);
        sample_count = 1500;
        for(int i=0;i<sample_count;i++){
                st1 = 0;
                while(!(st1 & 0x01)){
                        st1 = readByte(mg,MAG_ST1);
                }
                readMag(temp);
                mx = temp[0];
                my = temp[1];
                mz = temp[2];

                st2 = readByte(mg,MAG_ST2);
                if (!(st2 & 0x08)){
                        mag_temp[0] = mx;
                        mag_temp[1] = my;
                        mag_temp[2] = mz;
```

```
                }
                for(int j=0;j<3;j++){
                        if(mag_temp[j]>mag_max[j]) mag_max[j] = mag_temp[j];
                        if(mag_temp[j]<mag_min[j]) mag_min[j] = mag_temp[j];
                }
                delay(12);
        }
        mag_bias[0] = (mag_max[0] + mag_min[0])/2;
        mag_bias[1] = (mag_max[1] + mag_min[1])/2;
        mag_bias[2] = (mag_max[2] + mag_min[2])/2;

        dest1[0] = (float)mag_bias[0]*mRes*magCalib[0];
        dest1[1] = (float)mag_bias[1]*mRes*magCalib[1];
        dest1[2] = (float)mag_bias[2]*mRes*magCalib[2];
        printf("AK8963 magBias (mG):\t%f\t%f\t%f\n",dest1[0],dest1[1],dest1[2]);

        mag_scale[0] = (mag_max[0] - mag_min[0])/2;
        mag_scale[1] = (mag_max[1] - mag_min[1])/2;
        mag_scale[2] = (mag_max[2] - mag_min[2])/2;
        float avg_rad = mag_scale[0] + mag_scale[1] + mag_scale[2];
        avg_rad /= 3.0;

        dest2[0] = avg_rad/((float)mag_scale[0]);
        dest2[1] = avg_rad/((float)mag_scale[1]);
        dest2[2] = avg_rad/((float)mag_scale[2]);
        printf("AK8963 magScale (mG):\t%f\t%f\t%f\n",dest2[0],dest2[1],dest2[2]);

        cout<<"Calibracion Finalizada!!"<<endl;
}
```

B.2. Cálculo de Mínima Trayectoria

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <cmath>
#include <cstdlib>
#include <limits>
```

```
#include <fstream>
#include <cairo.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace std;
using namespace cv;
Mat image;

// Funciones
float* loadData(void);
float* TangenteInterior(float,float,float,float,float);
float* TangenteExterior(float,float,float,float,float);
float* Cruce_RectaCirculo(float,float,float,float,float);
float* Cruce_CirculoCirculo(float,float,float,float,float,float);
void saveImage(void);
void dibujoCarro(float,float,float,float,float,float);

float c = 50;
float pi = M_PI;
float deltamax = 12*pi/180;

float L = 0.190;
float A = 0.143;
float R = L/tan(deltamax);

// Inicializo cairo
cairo_surface_t *surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32,10*c,10*c);
cairo_t *cr = cairo_create(surface);

int main(){
        float* datos;
        float xi,xf,yi,yf,thetai,thetaf;
        float Ci[4],Cf[4];
        float C,alfa,minpath;
        int minrow,mincol;
        float angL1i,angL2i,angL1f,angL2f,ang1,ang2,angrt,angdes,angreal,angLi;
        float a,b,dti,dtf,theta,thetarti,alfai,alfaf;
        float signVel[3];
        float P1[2],P2[2];
        float signdeltai,signdeltaf,vi,cxi,cyi,cxf,cyf,dir;
```

```
cairo_translate (cr,0.0,10*c);
cairo_scale (cr,1.0,-1.0);
cairo_set_source_rgb (cr,1,1,1);          // Fondo de la imagen blanco
cairo_paint (cr);
cairo_stroke (cr);

datos = loadData();
xi = *(datos+0);      yi = *(datos+1);      thetai = *(datos+2);
xf = *(datos+3);      yf = *(datos+4);      thetaf = *(datos+5);
Ci[0] = *(datos+6); Ci[1] = *(datos+7); Ci[2] = *(datos+8); Ci[3] = *(datos+9);
Cf[0] = *(datos+10); Cf[1] = *(datos+11); Cf[2] = *(datos+12); Cf[3] = *(datos+13);
dibujoCarro(xi,yi,thetai,xf,yf,thetaf);

// Cálculo de las rectas tangentes internas y externas
float *pT11i, *pT12i, *pT21i, *pT22i;
float *pT11e, *pT12e, *pT21e, *pT22e;
float pT[8][8];

pT11i = TangenteInterior(Ci[0],Ci[1],Cf[0],Cf[1],R);
pT[0][0]=*(pT11i+0); pT[0][1]=*(pT11i+1); pT[0][2]=*(pT11i+2); pT[0][3]=*(pT11i+3);
pT[0][4]=*(pT11i+4); pT[0][5]=*(pT11i+5); pT[0][6]=*(pT11i+6); pT[0][7]=*(pT11i+7);

pT12i = TangenteInterior(Ci[0],Ci[1],Cf[2],Cf[3],R);
pT[1][0]=*(pT12i+0); pT[1][1]=*(pT12i+1); pT[1][2]=*(pT12i+2); pT[1][3]=*(pT12i+3);
pT[1][4]=*(pT12i+4); pT[1][5]=*(pT12i+5); pT[1][6]=*(pT12i+6); pT[1][7]=*(pT12i+7);

pT21i = TangenteInterior(Ci[2],Ci[3],Cf[0],Cf[1],R);
pT[2][0]=*(pT21i+0); pT[2][1]=*(pT21i+1); pT[2][2]=*(pT21i+2); pT[2][3]=*(pT21i+3);
pT[2][4]=*(pT21i+4); pT[2][5]=*(pT21i+5); pT[2][6]=*(pT21i+6); pT[2][7]=*(pT21i+7);

pT22i = TangenteInterior(Ci[2],Ci[3],Cf[2],Cf[3],R);
pT[3][0]=*(pT22i+0); pT[3][1]=*(pT22i+1); pT[3][2]=*(pT22i+2); pT[3][3]=*(pT22i+3);
pT[3][4]=*(pT22i+4); pT[3][5]=*(pT22i+5); pT[3][6]=*(pT22i+6); pT[3][7]=*(pT22i+7);

pT11e = TangenteExterior(Ci[0],Ci[1],Cf[0],Cf[1],R);
pT[4][0]=*(pT11e+0); pT[4][1]=*(pT11e+1); pT[4][2]=*(pT11e+2); pT[4][3]=*(pT11e+3);
pT[4][4]=*(pT11e+4); pT[4][5]=*(pT11e+5); pT[4][6]=*(pT11e+6); pT[4][7]=*(pT11e+7);

pT12e = TangenteExterior(Ci[0],Ci[1],Cf[2],Cf[3],R);
pT[5][0]=*(pT12e+0); pT[5][1]=*(pT12e+1); pT[5][2]=*(pT12e+2); pT[5][3]=*(pT12e+3);
```

```
pT[5][4]=*(pT12e+4); pT[5][5]=*(pT12e+5); pT[5][6]=*(pT12e+6); pT[5][7]=*(pT12e+7);

pT21e = TangenteExterior(Ci[2],Ci[3],Cf[0],Cf[1],R);
pT[6][0]=*(pT21e+0); pT[6][1]=*(pT21e+1); pT[6][2]=*(pT21e+2); pT[6][3]=*(pT21e+3);
pT[6][4]=*(pT21e+4); pT[6][5]=*(pT21e+5); pT[6][6]=*(pT21e+6); pT[6][7]=*(pT21e+7);

pT22e = TangenteExterior(Ci[2],Ci[3],Cf[2],Cf[3],R);
pT[7][0]=*(pT22e+0); pT[7][1]=*(pT22e+1); pT[7][2]=*(pT22e+2); pT[7][3]=*(pT22e+3);
pT[7][4]=*(pT22e+4); pT[7][5]=*(pT22e+5); pT[7][6]=*(pT22e+6); pT[7][7]=*(pT22e+7);

int N = 8;
// Longitud de un arco: L=2*pi*R*alfa/360;

// Longitud de arcos de posicion inicial: Lai
// Primera columna: longitudes de arco hacia un sentido
// Segunda columna: longitudes de arco hacia el sentido opuesto
float Lai[N][2];
float alfaLai[N][2];

for (int i=0;i<N;i++)
        for (int j=0;j<2;j++){
                C = sqrt(pow(xi-pT[i][2*j],2)+pow(yi-pT[i][2*j+1],2));
                alfa = acos(1-C*C/(2*R*R))*180/pi;
                Lai[i][j] = 2*pi*R*alfa/360;
        alfaLai[i][j] = alfa;
        }

// Longitud de arcos de posicion final: Laf
// Primera columna: longitudes de arco hacia un sentido
// Segunda columna: longitudes de arco hacia el sentido opuesto
float Laf[N][2];
float alfaLaf[N][2];

for (int i=0;i<N;i++)
        for (int j=0;j<2;j++){
                C = sqrt(pow(xf-pT[i][2*(j+2)],2)+pow(yf-pT[i][2*(j+2)+1],2));
                alfa = acos(1-C*C/(2*R*R))*180/pi;
                Laf[i][j] = 2*pi*R*alfa/360;
                alfaLaf[i][j] = alfa;
        }
```

```
// Longitudes de Rectas Tangentes: Internas y Externas
float Lrt[N][2];
for (int i=0;i<N;i++)
        for (int j=0;j<2;j++){
                Lrt[i][j] = sqrt(pow(pT[i][2*j]-pT[i][2*j+4],2)+pow(pT[i][2*j+1]-pT[i][2*j+1+4],2));
        }

float path[N][2];
for (int i=0;i<N;i++)
        for (int j=0;j<2;j++){
                path[i][j] = Lai[i][j] + Lrt[i][j] + Laf[i][j];
        }

// Calculo de Minima Distancia
for (int k=0;k<2*N;k++){
        minpath = path[0][0];
        minrow = 0;
        mincol = 0;
        for (int i=0;i<N;i++)
                for (int j=0;j<2;j++)
                        if (path[i][j]<minpath){
                                minpath = path[i][j];
                                minrow = i;
                                mincol = j;
                        }

        P1[0] = pT[minrow][2*mincol]; P1[1] = pT[minrow][2*mincol+1];
        P2[0] = pT[minrow][2*mincol+4];         P2[1] = pT[minrow][2*mincol+5];

        alfai = alfaLai[minrow][mincol]*pi/180;
        alfaf = alfaLaf[minrow][mincol]*pi/180;

        // angL1: angulo entre posicion inicial y el eje X
        // angL2: angulo entre punto tangente y el eje X
        if (minrow == 0 || minrow == 4){      //11
                signdeltai = 1;
                signdeltaf = 1;
                cxi = Ci[0];
                cyi = Ci[1];
                cxf = Cf[0];
                cyf = Cf[1];
```

```
}
else if (minrow == 1 || minrow == 5){ //12
        signdeltai = 1;
        signdeltaf = -1;
        cxi = Ci[0];
        cyi = Ci[1];
        cxf = Cf[2];
        cyf = Cf[3];
}
else if (minrow == 2 || minrow == 6){ //21
        signdeltai = -1;
        signdeltaf = 1;
        cxi = Ci[2];
        cyi = Ci[3];
        cxf = Cf[0];
        cyf = Cf[1];
}
else if (minrow == 3 || minrow == 7){ //22
        signdeltai = -1;
        signdeltaf = -1;
        cxi = Ci[2];
        cyi = Ci[3];
        cxf = Cf[2];
        cyf = Cf[3];
}

// Posicion inicial
C = sqrt(pow(cxi+R-xi,2)+pow(cyi-yi,2));
angL1i = abs(acos(1 - C*C/(2*R*R)));
C = sqrt(pow(cxi+R-P1[0],2)+pow(cyi-P1[1],2));
angL2i = abs(acos(1 - C*C/(2*R*R)));
if (yi<cyi) angL1i = 2*pi - angL1i;
if (P1[1]<cyi)          angL2i = 2*pi - angL2i;
if (angL1i<0)           angL1i = 2*pi + angL1i;
if (angL2i<0)           angL2i = 2*pi + angL2i;

ang1 = angL1i;
ang2 = angL2i;
a = abs(ang1-ang2);
if (ang1>ang2)                  ang1 -= 2*pi;
else if (ang1<ang2)ang2 -= 2*pi;
```

```
b = abs(ang1-ang2);
if (a<b){
        if (angL1i<angL2i) dti = 1;
        else if (angL1i>angL2i)         dti = -1;
}
else{
        if (angL1i<angL2i){
                angL2i -= 2*pi;
                dti = -1;
        }
        else if (angL1i>angL2i){
                angL1i -= 2*pi;
                dti = 1;
        }
}

// Posicion final
C = sqrt(pow(cxf+R-xf,2)+pow(cyf-yf,2));
angL1f = abs(acos(1 - C*C/(2*R*R))); // angulo entre centro x+R y posicion x final
C = sqrt(pow(cxf+R-P2[0],2)+pow(cyf-P2[1],2));
angL2f = abs(acos(1 - C*C/(2*R*R)));
if (yf<cyf) angL1f = 2*pi - angL1f;
if (P2[1]<cyf)          angL2f = 2*pi - angL2f;
if (angL1f<0)           angL1f = 2*pi + angL1f;
if (angL2f<0)           angL2f = 2*pi + angL2f;

ang1 = angL2f;
ang2 = angL1f;
a = abs(ang1-ang2);
if (ang1>ang2)                  ang1 -= 2*pi;
else if (ang1<ang2)ang2 -= 2*pi;
b = abs(ang1-ang2);

if (a<b){
        if (angL2f<angL1f) dtf = 1;
        else if (angL2f>angL1f)         dtf = -1;
}
else{
        if (angL2f<angL1f){
                angL1f -= 2*pi;
                dtf = -1;
```

```
        }
        else if (angL2f>angL1f){
                angL2f -= 2*pi;
                dtf = 1;
        }
}
// Velocidades
if (minrow == 0 || minrow == 4){                                    //11
        signVel[0] = 1;
        if (dti<0)          signVel[0] = -1;
        signVel[2] = 1;
        if (dtf<0)          signVel[2] = -1;
}
else if (minrow == 1 || minrow == 5){           //12
        signVel[0] = 1;
        if (dti<0)          signVel[0] = -1;
        signVel[2] = -1;
        if (dtf<0)          signVel[2] = 1;
}
else if (minrow == 2 || minrow == 6){           //21
        signVel[0] = -1;
        if (dti<0)          signVel[0] = 1;
        signVel[2] = 1;
        if (dtf<0)          signVel[2] = -1;
}
else if (minrow == 3 || minrow == 7){           //22
        signVel[0] = -1;
        if (dti<0)          signVel[0] = 1;
        signVel[2] = -1;
        if (dtf<0)          signVel[2] = 1;
}
if (thetai == pi)       signVel[0] = -1*signVel[0];

angLi = angL2i - angL1i;
thetarti = thetai + angLi;
if (thetarti < 0)           thetarti += 2*pi;
else if (thetarti > 2*pi)       thetarti -= 2*pi;

if (P1[1]>P2[1]){
        if (0<thetarti && thetarti<pi)          signVel[1] = -1;
        else if (pi<thetarti && thetarti<2*pi)      signVel[1] = 1;
```

```
                    }
                    else if (P1[1]<P2[1]){
                              if (0<thetarti && thetarti<pi)              signVel[1] = 1;
                              else if (pi<thetarti && thetarti<2*pi)      signVel[1] = -1;
                    }
                    // Analizo si llega a la posicion final con el angulo correcto
                    theta = thetai;
                    if (theta == 2*pi)     theta = 0;
                    if (dti>0)             theta += alfai;
                    else if (dti<0)                theta -= alfai;
                    if (theta < 0)                 theta += 2*pi;
                    if (theta > 2*pi)      theta -= 2*pi;
//                  if (0<theta && theta<pi || theta == 0)              vi = 1;
//                  else if (pi<theta && theta<2*pi || round(1000*theta)/1000 == round(1000*pi)/1000)
          vi = -1;

                    angrt = theta;
//                  if (round(1000*P1[1])>round(1000*P2[1]))            dir = -1;
//                  else if (round(1000*P1[1])<round(1000*P2[1]))       dir = 1;
//                  else if (P1[0]>P2[0])                               dir = -1;
//                  else if (P1[1]<P2[1])                               dir = 1;
//                  vi = dir*vi;
//                  signVel[1] = vi;

                    if (dtf>0)             theta += alfaf;
                    else if (dtf<0)                theta -= alfaf;
                    if (theta < 0)                 theta += 2*pi;
                    if (theta > 2*pi)      theta -= 2*pi;

                    angdes = round(1000*thetaf)/1000;
                    angreal = round(1000*theta)/1000;
                    if (angdes == 2*pi)                                 angdes = 0;
                    else if (angreal == round(1000*2*pi)/1000)          angreal = 0;
                    if (angdes < 0)                angdes += 2*pi;
                    if (angreal < 0)       angreal += 2*pi;


% Mostrar en pantalla
cout<<"Punto P1 : "<<P1[0]<<","<<P1[1]<<endl;
cout<<"Punto P2 : "<<P2[0]<<","<<P2[1]<<endl;
cout<<"Angrt     : "<<angrt*180/pi<<endl;
cout<<"Angulos : "<<alfai*180/pi<<","<<alfaf*180/pi<<endl;
```

```
cout<<"angLi(1-2):  "<<angL1i*180/pi<<"::"<<angL2i*180/pi<<endl;
cout<<"angLf(2-1): "<<angL2f*180/pi<<"::"<<angL1f*180/pi<<endl;
cout<<"Min Path : "<<minpath<<endl;
cout<<"Velocidad : "<<signVel[0]<<"::"<<signVel[1]<<"::"<<signVel[2]<<endl;
cout<<"Check     : "<<theta*180/pi<<"::"<<thetaf*180/pi<<endl;
cout<<endl;

        if (abs(angreal-angdes)/angdes*100<0.2 || (angreal==0 && angdes==0) )          break;
        path[minrow][mincol] = 100;
        }
        cairo_set_source_rgb (cr,0,1,0);
        cairo_move_to (cr,P1[0]*c,P1[1]*c);
        cairo_line_to (cr,P2[0]*c,P2[1]*c);
        cairo_stroke (cr);
        if (dti>0) cairo_arc (cr,cxi*c,cyi*c,R*c,angL1i,angL2i);
        if (dti<0) cairo_arc_negative (cr,cxi*c,cyi*c,R*c,angL1i,angL2i);
        cairo_stroke (cr);
        if (dtf>0) cairo_arc (cr,cxf*c,cyf*c,R*c,angL2f,angL1f);
        if (dtf<0) cairo_arc_negative (cr,cxf*c,cyf*c,R*c,angL2f,angL1f);
        cairo_stroke (cr);
        saveImage();

        ofstream min_path ("minpath.txt");
        min_path<<P1[0]<<endl;
        min_path<<P1[1]<<endl;
        min_path<<P2[0]<<endl;
        min_path<<P2[1]<<endl;
        min_path<<xi<<endl;
        min_path<<yi<<endl;
        min_path<<thetai<<endl;
        min_path<<xf<<endl;
        min_path<<yf<<endl;
        min_path<<thetaf<<endl;
        min_path<<angrt<<endl;
        min_path<<signdeltai<<endl;
        min_path<<signdeltaf<<endl;
        min_path<<signVel[0]<<endl;
        min_path<<signVel[1]<<endl;
        min_path<<signVel[2]<<endl;
        min_path.close();
```

```
	return 0;
}

//*********************************************************************//
//*********************************************************************//

float* loadData(){
	static float datos[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};

	string num;
	ifstream data ("data.txt");
	for (int i=0;i<14;i++){
		getline (data,num);
		datos[i] = strtof(num.c_str(),0);
	}
	data.close();
	return datos;
}


// Funciones de Rectas Tangentes
float* TangenteInterior(float cx1,float cy1,float cx2,float cy2,float R){
	static float pTan[8] = {0,0,0,0,0,0,0,0}; // {Wx,Wy,Xx,Xy,Yx,Yy,Zx,Zy}

	float a1,b1,a2,b2,a3,b3;
	float x1i,y1i,x1f,y1f;
	float x2i,y2i,x2f,y2f;
	float x3i,y3i,x3f,y3f;
	float p1x,p1y,p2x,p2y;
	float cv1x,cv2x,cv1y,cv2y,rv1,rv2;
	float P[2], Q[2], S[2], T[2];
	float* pCruce;

	int pos=0;
	if(cx2 == cx1) pos = 1;		// Vertical
	if(cy2 == cy1) pos = 2;		// Horizontal

	// 1) Recta cr - cR
	if(pos == 1){
		a1 = (cx2-cx1)/(cy2-cy1);
		b1 = cx2 - a1*cy2;
```

```
            y1i = cy2;                          y1f = cy1;
            x1i = a1*y1i + b1; x1f = a1*y1f + b1;
}
else{
            a1 = (cy2-cy1)/(cx2-cx1);
            b1 = cy2 - a1*cx2;
            x1i = cx2;                          x1f = cx1;
            y1i = a1*x1i + b1; y1f = a1*x1f + b1;
}

// 2) Recta perpendicular a la recta para cada circunferencia
//    P,Q -> radio menor
//    S,T -> radio mayor

if (pos == 1){
            // P-Q
            p1x = cx1 - R;
            p1y =  cy1;
            p2x = cx1 + R;
            p2y = cy1;
            P[0] = p1x; P[1] = p1y;
            Q[0] = p2x; Q[1] = p2y;

            // S-T
            p1x = cx2 - R;
            p1y =  cy2;
            p2x = cx2 + R;
            p2y = cy2;
            S[0] = p1x; S[1] = p1y;
            T[0] = p2x; T[1] = p2y;
}
else if (pos == 2){
            // P-Q
            p1x =  cx1;
            p1y = cy1 - R;
            p2x =  cx1;
            p2y = cy1 + R;
            P[0] = p1x; P[1] = p1y;
            Q[0] = p2x; Q[1] = p2y;

            // S-T
```

```
            p1x =  cx2;
            p1y = cy2 - R;
            p2x =  cx2;
            p2y = cy2 + R;
            S[0] = p1x; S[1] = p1y;
            T[0] = p2x; T[1] = p2y;
}
else{
// P-Q
a2 = -1/a1;
b2 = -a2*cx1+cy1;
pCruce = Cruce_RectaCirculo(a2,b2,cx1,cy1,R);
P[0] = *(pCruce+0); P[1] = *(pCruce+1);
Q[0] = *(pCruce+2); Q[1] = *(pCruce+3);

// S-T
a2 = -1/a1;
b2 = -a2*cx2+cy2;
pCruce = Cruce_RectaCirculo(a2,b2,cx2,cy2,R);
S[0] = *(pCruce+0); S[1] = *(pCruce+1);
T[0] = *(pCruce+2); T[1] = *(pCruce+3);
}

// 3) Unir P-T y Q-S
x2i = P[0];          x2f = T[0];
y2i = P[1];          y2f = T[1];
x3i = Q[0];          x3f = S[0];
y3i = Q[1];          y3f = S[1];

// 4) Armar circulo con diametro N-V y O-V
// Recta Central
a1 = (y1f-y1i)/(x1f-x1i);
b1 = y1i - a1*x1i;
// Recta P-T
a2 = (y2f-y2i)/(x2f-x2i);
b2 = y2i - a2*x2i;
// Recta Q-S
a3 = (y3f-y3i)/(x3f-x3i);
b3 = y3i - a3*x3i;
```

```
		// Calculo el punto V a partir de la interseccion de rectas
		float V[2];
		if (pos == 1){
			p1x = (b3-b2)/(a2-a3);
			p1y = a2*p1x + b2;
		}
		else{
			p1x = (b2-b1)/(a1-a2);
			p1y = a1*p1x + b1;
		}
		V[0] = p1x;		V[1] = p1y;

		// Calculo punto centro entre N-V
		cv1x = (cx1 + V[0])/2;
		cv1y = (cy1 + V[1])/2;
		rv1 = sqrt(pow((abs(cx1-V[0])/2),2) + pow((abs(cy1-V[1])/2),2));

		// Calculo punto centro entre O-V
		cv2x = (cx2 + V[0])/2;
		cv2y = (cy2 + V[1])/2;
		rv2 = sqrt(pow((abs(cx2-V[0])/2),2) + pow((abs(cy2-V[1])/2),2));

		// 5) Intersecciones de circulo
		// Busco Puntos
		pCruce = Cruce_CirculoCirculo(cx1,cy1,R,cv1x,cv1y,rv1);
		pTan[0] = *(pCruce+0);		pTan[1] = *(pCruce+1);		// W(x,y)
		pTan[2] = *(pCruce+2);		pTan[3] = *(pCruce+3);		// X(x,y)

		pCruce = Cruce_CirculoCirculo(cx2,cy2,R,cv2x,cv2y,rv2);
		pTan[4] = *(pCruce+2);		pTan[5] = *(pCruce+3);		// Y(x,y)
		pTan[6] = *(pCruce+0);		pTan[7] = *(pCruce+1);		// Z(x,y)

		return pTan;
}

float* TangenteExterior(float cx1,float cy1,float cx2,float cy2,float R){
		static float pTan[8] = {0,0,0,0,0,0,0,0};

		float inf = numeric_limits<float>::infinity();
		float m,b;
		float* p;
```

```
        m = (cy2-cy1)/(cx2-cx1);
        if (m == 0){
                pTan[0]  =  cx1;  pTan[1]  =  cy1+R;
                pTan[2]  =  cx1;  pTan[3]  =  cy1-R;
                pTan[4]  =  cx2;  pTan[5]  =  cy2+R;
                pTan[6] = cx2; pTan[7] = cy2-R;
        }
        else if (m == inf){
                pTan[0]  =  cx1+R;  pTan[1]  =  cy1;
                pTan[2]  =  cx1-R;  pTan[3]  =  cy1;
                pTan[4]  =  cx2+R;  pTan[5]  =  cy2;
                pTan[6] = cx2-R; pTan[7] = cy2;
        }
        else{
                m = -1/m;
                b = cy1 - m*cx1;
                p = Cruce_RectaCirculo(m,b,cx1,cy1,R);
                pTan[0] = *(p+0);    pTan[1]=*(p+1);
                pTan[2] = *(p+2);    pTan[3]=*(p+3);

                b = cy2 - m*cx2;
                p = Cruce_RectaCirculo(m,b,cx2,cy2,R);
                pTan[4] = *(p+0);    pTan[5]=*(p+1);
                pTan[6] = *(p+2);    pTan[7]=*(p+3);
        }

        return pTan;
}

// Funciones de cruces
float* Cruce_RectaCirculo(float m,float b,float cx,float cy,float r){
        static float pCruce[4] = {0,0,0,0};          // {x1,y1,x2,y2}
        float A,B,C,AA,BB,CC;

        A = -2*cx;
        B = -2*cy;
        C = cx*cx + cy*cy - r*r;
        AA = 1 + m*m;
        BB = 2*m*b + B*m + A;
        CC = b*b + B*b + C;
```

```
		pCruce[0] = (-BB + sqrt(BB*BB - 4*AA*CC))/(2*AA);
		pCruce[2] = (-BB - sqrt(BB*BB - 4*AA*CC))/(2*AA);
		pCruce[1] = m*pCruce[0] + b;
		pCruce[3] = m*pCruce[2] + b;

		return pCruce;
}

float* Cruce_CirculoCirculo(float c1x,float c1y,float r1,float c2x,float c2y,float r2){
		static float pCruce[4] = {0,0,0,0};		// {x1,y1,x2,y2}

		float m,b,AA,BB,CC;
		float* p;

		if (c1y != c2y){
			m = -(c1x-c2x)/(c1y-c2y);
			b = -((c1x*c1x+c1y*c1y-r1*r1)-(c2x*c2x+c2y*c2y-r2*r2))/(-2*(c1y-c2y));
			p = Cruce_RectaCirculo(m,b,c1x,c1y,r1);
			pCruce[0] = *(p+0);
			pCruce[1] = *(p+1);
			pCruce[2] = *(p+2);
			pCruce[3] = *(p+3);
		}
		else{
			pCruce[0] = -((c1x*c1x-r1*r1)-(c2x*c2x-r2*r2))/(-2*(c1x-c2x));
			pCruce[2] = pCruce[0];

			AA = 1;
			BB = -2*c1y;
			CC = pCruce[0]*pCruce[0] - 2*c1x*pCruce[0] + c1x*c1x + c1y*c1y - r1*r1;
			pCruce[1] = (-BB + sqrt(BB*BB - 4*AA*CC))/(2*AA);
			pCruce[3] = (-BB - sqrt(BB*BB - 4*AA*CC))/(2*AA);
		}

		return pCruce;
}

void dibujoCarro(float xi,float yi,float thetai,float xf,float yf,float thetaf){
		float x1,x2,x3,x4,xF;
		float y1,y2,y3,y4,yF;
```

```
// POSICION INICIAL
// Dibujo carrito
x1 = xi + (A/2)*sin(thetai);
y1 = yi - (A/2)*cos(thetai);
x2 = xi - (A/2)*sin(thetai);
y2 = yi + (A/2)*cos(thetai);
xF = xi + (L/1)*cos(thetai);
yF = yi + (L/1)*sin(thetai);
x3 = xF - (A/2)*sin(thetai);
y3 = yF + (A/2)*cos(thetai);
x4 = xF + (A/2)*sin(thetai);
y4 = yF - (A/2)*cos(thetai);

cairo_set_source_rgb (cr,1,0,0);
cairo_move_to (cr,x1*c,y1*c);
cairo_line_to (cr,x2*c,y2*c);
cairo_line_to (cr,x3*c,y3*c);
cairo_line_to (cr,x4*c,y4*c);
cairo_close_path (cr);
cairo_stroke (cr);

cairo_arc (cr,xi*c,yi*c,0.05*c,0,2*pi);
cairo_fill (cr);
cairo_stroke (cr);

// POSICION FINAL
// Dibujo carrito
x1 = xf + (A/2)*sin(thetaf);
y1 = yf - (A/2)*cos(thetaf);
x2 = xf - (A/2)*sin(thetaf);
y2 = yf + (A/2)*cos(thetaf);
xF = xf + (L/1)*cos(thetaf);
yF = yf + (L/1)*sin(thetaf);
x3 = xF - (A/2)*sin(thetaf);
y3 = yF + (A/2)*cos(thetaf);
x4 = xF + (A/2)*sin(thetaf);
y4 = yF - (A/2)*cos(thetaf);

cairo_set_source_rgb (cr,0,0,1);
cairo_move_to (cr,x1*c,y1*c);
```

```
        cairo_line_to (cr,x2*c,y2*c);
        cairo_line_to (cr,x3*c,y3*c);
        cairo_line_to (cr,x4*c,y4*c);
        cairo_close_path (cr);
        cairo_stroke (cr);

        cairo_arc (cr,xf*c,yf*c,0.05*c,0,2*pi);
        cairo_fill (cr);
        cairo_stroke (cr);
}

void saveImage(){
        cairo_surface_write_to_png (surface,"carpath.png");
        image = imread("carpath.png");
        imshow("Trayectoria Minima",image);
        waitKey(0);
}
```

B.3. Control del Robot para Tiempo Mínimo

```
#include <iostream>
#include <signal.h>
#include <stdio.h>
#include <cmath>
#include <string>
#include <stdint.h>
#include <unistd.h>
#include <cstdlib>
#include <limits>
#include <chrono>
#include <fstream>
#include "cairo.h"
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "wiringPi.h"
#include "wiringPiI2C.h"
#include "softPwm.h"
```

```
#include "MadgwickAHRS.h"
#include "mpu9265.h"

// Pines encoder
#define ENA             17
#define N1              27
#define N2              22
#define N3              10
#define N4              9
#define ENB             11
#define EN_MOTOR        13
#define ENCODER         18

using namespace std;
using namespace cv;

float* loadDatos(void);
void plotImg(float);
void dibujoCarro(float,float,float,float,float,float);
void saveImage(void);
float deltaFunction(float,float,float);
float velFunction(float,float,float,float);
int sign(float);
void iniPorts(void);
void escapeManual(int);
void encoderISR(void);
void getDataCarro(float*);

typedef chrono::high_resolution_clock Clock;

float c = 50;
float pi = M_PI;
float deltamax = 12*pi/180;
float vmax = 0.5*1;

float L = 0.190;
float A = 0.143;
float R = L/tan(deltamax);

float p1 = 25.3467, p2 = 316.2278;
float r1 = 25.3467, r2 = 316.2278;
```

```
Mat image;

// Inicializo cairo
cairo_surface_t *surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32,10*c,10*c);
cairo_t *cr = cairo_create(surface);

float P1[2], P2[2];
float x,y,theta;
float xr,yr,thetar;
float angrt;
float xi,yi,thetai,xf,yf,thetaf;

float* datos;
float signVel[3];
float xp,yp,thetap,v,w,delta,sentido,vp,tandelta;
float ti,dt,tf;
float signdeltai,signdeltaf;
float* coefFd;
float err;

int temp[3];
float carro[3];
float axx_old,ayy_old,azz_old;
float gxx_old,gyy_old,gzz_old;
float x_old = 0,y_old = 0;
float dx_old = 0, dy_old = 0;
float dxp_old = 0, dyp_old = 0;

int ne = 0;
int Ne = 20;            // Pulsos/vuelta

ofstream dxEncoder ("dx_encoder.txt");
ofstream dyEncoder ("dy_encoder.txt");

int loop = 1;
int main(int argc, char* argv[]){
        float v_old = 0, delta_old = 100;;

        (void)signal(SIGINT,escapeManual);
        (void)signal(SIGQUIT,escapeManual);
```

```
iniPorts();
iniMPU9265();
if (argc == 2 && !strcmp(argv[1],"C"))          calibraMPU9265();
digitalWrite(ENA,HIGH);
softPwmCreate(ENB,0,100);

// Datos de minima trayectoria
datos = loadDatos();                    // Leo datos de minPath
P1[0] = *(datos+0); P1[1] = *(datos+1);
P2[0] = *(datos+2); P2[1] = *(datos+3);
xi = *(datos+4);    yi = *(datos+5);    thetai = *(datos+6);
xf = *(datos+7);    yf = *(datos+8);    thetaf = *(datos+9);
angrt = *(datos+10);
signdeltai = *(datos+11);
signdeltaf = *(datos+12);
signVel[0] = *(datos+13);
signVel[1] = *(datos+14);
signVel[2] = *(datos+15);
if (thetai > pi)        thetai -= 2*pi;
if (thetaf > pi)        thetaf -= 2*pi;
if (angrt > pi)                 angrt -= 2*pi;

ofstream Fxw        ("Fx_deseado.txt");
ofstream Fxpw       ("Fxp_deseado.txt");
ofstream Fxppw      ("Fxpp_deseado.txt");
ofstream Fyw        ("Fy_deseado.txt");
ofstream Fypw       ("Fyp_deseado.txt");
ofstream Fyppw      ("Fypp_deseado.txt");
ofstream veld       ("vel_deseada.txt");
ofstream deltad ("delta_deseado.txt");

float Fxp_old = 0, Fyp_old = 0;

x = xi; xp = 0;
y = yi; yp = 0;
theta = thetai;
delta = 0;
v = 0;

int N = 1;
dt = 0.003;
```

```
float tt = 0;
        // GENERACION DE PUNTOS DE TRAYECTORIA DESEADA
        while(1){
                Fxw<<x<<endl;
                Fyw<<y<<endl;
                Fxpw<<xp<<endl;
                Fypw<<yp<<endl;
                Fxppw<<(xp - Fxp_old)/dt<<endl;
                Fyppw<<(yp - Fyp_old)/dt<<endl;
                Fxp_old = xp; Fyp_old = yp;

                // VARIABLE DE CONTROL - DELTA
                delta = deltaFunction(x,y,theta);
                if (delta > 0){
                        delta = deltamax;
                }
                else if (delta == 0){
                        delta = 0;
                }
                else if (delta < 0){
                        delta = -deltamax;
                }

                // VARIABLE DE CONTROL - VELOCIDAD
                v = velFunction(x,y,theta,delta);
                if (v > 0){
                        v = vmax;
                }
                else if (v < 0){
                        v = -vmax;
                }
                // MODELO DEL UGV
                xp = v*cos(theta);
                yp = v*sin(theta);
                thetap = v/L*tan(delta);

                x = x + xp*dt;
                y = y + yp*dt;
                theta = theta + thetap*dt;

                // Limitando de 180 a -180
```

```
            if (theta > 2*pi || theta > pi)          theta -= 2*pi;
            if (theta < -pi)                         theta += 2*pi;

            err = sqrt(pow(x-xf,2) + pow(y-yf,2) + pow(theta-thetaf,2));
            if (err < 0.06)                  break;
            N++;
            tt += dt;
        }
        Fxw.close();
        Fxpw.close();
        Fxppw.close();
        Fyw.close();
        Fypw.close();
        Fyppw.close();

        veld.close();
        deltad.close();

printf("x final:    %2.5f\n",x);
printf("y final:    %2.5f\n",y);
printf("theta final: %2.5f\n",theta*180/pi);
printf("Tiempo Minimo: %2.5f\n",tt);

cout<<"Presione Enter para continuar..."<<endl;
getchar();

////////////////////////////////////////////////////////////////
        // CONTROL Y NAVEGACION
        string num;

        ifstream Fxr        ("Fx_deseado.txt");
        ifstream Fxpr       ("Fxp_deseado.txt");
        ifstream Fxppr      ("Fxpp_deseado.txt");
        ifstream Fyr        ("Fy_deseado.txt");
        ifstream Fypr       ("Fyp_deseado.txt");
        ifstream Fyppr      ("Fypp_deseado.txt");

        ofstream posx("posx.txt");
        ofstream posy("posy.txt");
        ofstream angtheta("angtheta.txt");
        ofstream angdelta("angdelta.txt");
```

```
ofstream velocidad("velocidad.txt");
ofstream encoderPulsos("encoder.txt");

float F1d, F1dp, F1dpp, F2d, F2dp, F2dpp;
float F1 = xi, F1p = 0, F2 = yi, F2p = 0;
float eps1, eps2;
float u;

float vel = 40;
auto t1 = Clock::now();

// INICIO DE LAZO DE CONTROL
//      for (int k=0;k<=N;k++){
while(1){
        auto t2 = Clock::now();
        dt = chrono::duration<double,milli>(t2-t1).count(); dt /= 1000.0;
        t1 = t2;
        printf("%d: dt = %f\t",loop,dt); loop++;

        posx<<x<<endl;
        posy<<y<<endl;
        angtheta<<theta*180/pi<<endl;
        angdelta<<delta*180/pi<<endl;
        velocidad<<v<<endl;

        getline(Fxr,num);    F1d = strtof(num.c_str(),0);
        getline(Fxpr,num); F1dp = strtof(num.c_str(),0);
        getline(Fxppr,num); F1dpp =strtof(num.c_str(),0);
        getline(Fyr,num);    F2d = strtof(num.c_str(),0);
        getline(Fypr,num); F2dp = strtof(num.c_str(),0);
        getline(Fyppr,num);F2dpp = strtof(num.c_str(),0);

        // LEY DE CONTROL
        eps1 = F1dpp + p1*(F1dp - F1p) + p2*(F1d - F1);
        eps2 = F2dpp + r1*(F2dp - F2p) + r2*(F2d - F2);

        vp = eps1*cos(theta) + eps2*sin(theta);
        u = L/(v*v)*(-eps1*sin(theta) + eps2*cos(theta));

        v = vp*dt + v;
        if (v > vmax)          v = vmax;
```

```
                if (v < -vmax)          v = -vmax;

                delta = atan2f(u,1);
                if (delta > deltamax)           delta = deltamax;
                if (delta < -deltamax)          delta = -deltamax;
                if (delta > 0){
                        digitalWrite(N2,LOW);
                        digitalWrite(N1,HIGH);
                }
                else if (delta == 0){
                        digitalWrite(N1,LOW);
                        digitalWrite(N2,LOW);
                }
                else if (delta < 0){
                        delta = -deltamax;
                        digitalWrite(N1,LOW);
                        digitalWrite(N2,HIGH);
                }
                // Calculando VELOCIDAD
                if (v > 0){
                        digitalWrite(N4,LOW);
                        digitalWrite(N3,HIGH);
                        softPwmWrite(ENB,vel);
                }
                else if (v < 0){
                        digitalWrite(N3,LOW);
                        digitalWrite(N4,HIGH);
                        softPwmWrite(ENB,vel);
                }
                // Carro Real
                getDataCarro(carro);
                x     = carro[0];
                y     = carro[1];
                theta = carro[2];

                // Limitando de 180 a -180
                if (theta > 2*pi || theta > pi)         theta -= 2*pi;
                if (theta < -pi)                        theta += 2*pi;

//              delay(1);
        }
```

```
		// FIN DE LAZO DE CONTROL

		cout<<"x_final : "<<xf<<" :: "<<x<<endl;
		cout<<"y_final : "<<yf<<" :: "<<y<<endl;
		cout<<"theta_final: "<<thetaf*180/pi<<" :: "<<theta*180/pi<<endl;

		Fxr.close();
		Fxpr.close();
		Fxppr.close();
		Fyr.close();
		Fypr.close();
		Fyppr.close();

		posx.close();
		posy.close();
		angtheta.close();
		angdelta.close();
		velocidad.close();
		encoderPulsos.close();
		dxEncoder.close();
		dyEncoder.close();

//		plotImg(1);
//		dibujoCarro(xi,yi,thetai,x,y,theta);
//		saveImage();

		digitalWrite(N1,LOW);
		digitalWrite(N2,LOW);
		digitalWrite(ENA,LOW);
		digitalWrite(N3,LOW);
		digitalWrite(N4,LOW);
		softPwmWrite(ENB,0);
		delay(100);

		return 0;
}

//****************************************************************//
//****************************************************************//
float* loadDatos(){
		static float datos[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
	string num;
	ifstream minpath ("minpath.txt");
	for (int i=0;i<16;i++){
		getline  (minpath,num);
		datos[i] = strtof(num.c_str(),0);
	}
	minpath.close();
	return datos;
}
void plotImg(float m){
	cairo_translate (cr,0.0,10*c);
	cairo_scale (cr,1.0,-1.0);

	cairo_set_source_rgb (cr,1,1,1);		// Fondo de la imagen blanco
	cairo_paint (cr);
	cairo_stroke (cr);
	cairo_set_source_rgb (cr,1,0,1);

	ifstream posx		("posx.txt");
	ifstream posy		("posy.txt");
	string xx,yy;
	float x,y;

	getline (posx,xx);
	getline (posy,yy);
	x = strtof(xx.c_str(),0);
	y = strtof(yy.c_str(),0);
	cairo_move_to		(cr,x*c,y*c);

	for (int n=2;n<=m;n++){
		getline (posx,xx);
		getline (posy,yy);
		x = strtof(xx.c_str(),0);
		y = strtof(yy.c_str(),0);
		cairo_line_to(cr,x*c,y*c);
	}
	cairo_stroke(cr);
	posx.close();
	posy.close();
}
```

```
void dibujoCarro(float xi,float yi,float thetai,float xf,float yf,float thetaf){
        float x1,x2,x3,x4,xF;
        float y1,y2,y3,y4,yF;

        // POSICION INICIAL
        // Dibujo carrito
        x1 = xi + (A/2)*sin(thetai);
        y1 = yi - (A/2)*cos(thetai);
        x2 = xi - (A/2)*sin(thetai);
        y2 = yi + (A/2)*cos(thetai);
        xF = xi + (L/1)*cos(thetai);
        yF = yi + (L/1)*sin(thetai);
        x3 = xF - (A/2)*sin(thetai);
        y3 = yF + (A/2)*cos(thetai);
        x4 = xF + (A/2)*sin(thetai);
        y4 = yF - (A/2)*cos(thetai);

        cairo_set_source_rgb (cr,1,0,0);
        cairo_move_to (cr,x1*c,y1*c);
        cairo_line_to (cr,x2*c,y2*c);
        cairo_line_to (cr,x3*c,y3*c);
        cairo_line_to (cr,x4*c,y4*c);
        cairo_close_path (cr);
        cairo_stroke (cr);

        cairo_arc (cr,xi*c,yi*c,0.04*c,0,2*pi);
        cairo_fill (cr);
        cairo_stroke (cr);

        // POSICION FINAL
        // Dibujo carrito
        x1 = xf + (A/2)*sin(thetaf);
        y1 = yf - (A/2)*cos(thetaf);
        x2 = xf - (A/2)*sin(thetaf);
        y2 = yf + (A/2)*cos(thetaf);
        xF = xf + (L/1)*cos(thetaf);
        yF = yf + (L/1)*sin(thetaf);
        x3 = xF - (A/2)*sin(thetaf);
        y3 = yF + (A/2)*cos(thetaf);
        x4 = xF + (A/2)*sin(thetaf);
        y4 = yF - (A/2)*cos(thetaf);
```

```
		cairo_set_source_rgb (cr,0,0,1);
		cairo_move_to (cr,x1*c,y1*c);
		cairo_line_to (cr,x2*c,y2*c);
		cairo_line_to (cr,x3*c,y3*c);
		cairo_line_to (cr,x4*c,y4*c);
		cairo_close_path (cr);
		cairo_stroke (cr);

		cairo_arc (cr,xf*c,yf*c,0.04*c,0,2*pi);
		cairo_fill (cr);
		cairo_stroke (cr);
}

void saveImage(){
		cairo_surface_write_to_png (surface,"XY.png");
		image = imread("XY.png"); imshow("Trayectoria
		XY",image);
		waitKey(0);
}

float deltaFunction(float x,float y,float theta){
		float D,Di,Df;
		float f1,f2,sf;
		float difang,difpos,difposx,difposy;

		if (theta == pi/2)	theta -= 1e-6;

		D = sqrt(pow(xi-xf,2) + pow(yi-yf,2));
		Di = sqrt(pow(x-xi,2) + pow(y-yi,2));
		Df = sqrt(pow(x-xf,2) + pow(y-yf,2));

		// ECUACION RECTA IMAGINARIA
		// Y - X*tan(theta) - (y-x*tan(theta)) = 0
		// (X,Y) -> Punto a analizar a partir de la recta formada por (x,y,theta)
		f1 = (P1[1] - P1[0]*tan(theta) - (y-x*tan(theta)))*sign(cos(theta))*round(Df/D);
		f2 = (yf - xf*tan(angrt) - (P2[1]-P2[0]*tan(angrt)))*sign(cos(theta))*round(Di/D);
		sf = f1 + f2;

		difang = round(1e1*abs(theta-angrt)*180/pi)/1e1;
//		difpos = round(5e1*sqrt(pow(x-P2[0],2) + pow(y-P2[1],2)))/5e1;
//		if (difang <= 0.1 && difpos != 0)		sf = 0;
```

```
difposx = round(5e1*(x - P2[0]))/5e1;
difposy = round(5e1*(y - P2[1]))/5e1;
difpos = difposx + difposy;

if (difang <= 0.1 && difpos != 0)          sf = 0;
        return sf;
}

float velFunction(float x, float y, float theta, float delta){
        float di1,df2,Dip,Dfp;
        float fact[3];
        float f1,f2,f3,vf;

        if (theta == 0)         theta = 1e-6;

        // Constantes
        di1 = sqrt(pow(xi-P1[0],2) + pow(yi-P1[1],2));
        df2 = sqrt(pow(xf-P2[0],2) + pow(yf-P2[1],2));

        // Variables
        Dip = sqrt(pow(xi-x,2) + pow(yi-y,2));
        Dfp = sqrt(pow(xf-x,2) + pow(yf-y,2));

        fact[0] = !(floor(Dip/di1));
        fact[1] = !(!(floor(Dip/di1)*floor(Dfp/df2)));
        fact[2] = !(floor(Dfp/df2));

        if (delta == 0){
                fact[0] = 0;
                fact[1] = 1;
                fact[2] = 0;
        }

        float mH = -1/tan(theta);
        f1 = (P1[1] - P1[0]*mH)*fact[0];
        f2 = (P2[1] - P2[0]*mH)*fact[1];
        f3 = (yf - xf*mH)*fact[2];

        vf = ((f1 + f2 + f3) - (y-x*mH))*sign(theta);
        return vf;
}
```

```
int sign(float num){
	int signo;

	if (num != 0)		signo = num/fabs(num);
	if (num == 0)		signo = 1;

	return signo;
}

void iniPorts(void){
	if (wiringPiSetupGpio() == -1){
		cout<<"Error: wiringPi setup failed\n";
		exit(0);
	}

	pinMode(ENCODER,INPUT);
	pinMode(ENA,OUTPUT);
	pinMode(ENB,OUTPUT);
	pinMode(N1,OUTPUT);
	pinMode(N2,OUTPUT);
	pinMode(N3,OUTPUT);
	pinMode(N4,OUTPUT);

	digitalWrite(ENA,LOW);
	digitalWrite(ENB,LOW);
	digitalWrite(N1,LOW);
	digitalWrite(N2,LOW);
	digitalWrite(N3,LOW);
	digitalWrite(N4,LOW);

	if (wiringPiISR(ENCODER,INT_EDGE_FALLING,&encoderISR) < 0){
		cout<<"Error: wiringPi ISR failed\n";
		exit(0);
	}
}

void getDataCarro(float* dest){
	float a = 0.9;
	float dxo,dyo,dxpo,dypo;

	float dxenc,dyenc;
```

```
float acelx,acely;
int ax,ay,az,gx,gy,gz;
float axx,ayy,azz,gxx,gyy,gzz;

readAccel(temp); ax = temp[0];        ay = temp[1];        az = temp[2];
readGyro(temp); gx = temp[0];         gy = temp[1];        gz = temp[2];

if (ax>-200 && ax<200)        ax = 0;
if (ay>-200 && ay<200)        ay = 0;
if (az>-300 && az<300)        az = 0;

if (gx>-300 && gx<300)        gx = 0;
if (gy>-300 && gy<300)        gy = 0;
if (gz>-300 && gz<300) gz = 0;

axx = ax*aRes;      ayy = ay*aRes;      azz = az*aRes;
gxx = gx*gRes;      gyy = gy*gRes;      gzz = gz*gRes;

// Filtrado Digital: y[n] = (1-a)*x[n] + a*y[n-1]
axx = (1-a)*axx + a*axx_old;
ayy = (1-a)*ayy + a*ayy_old;
azz = (1-a)*azz + a*azz_old;

gxx = (1-a)*gxx + a*gxx_old;
gyy = (1-a)*gyy + a*gyy_old;
gzz = (1-a)*gzz + a*gzz_old;

axx_old = axx;
ayy_old = ayy;
azz_old = azz;
gxx_old = gxx;
gyy_old = gyy;
gzz_old = gzz;

// FILTRO KALMAN / MADGWICK - ORIENTACION
readMag(temp);    mx = temp[0];       my = temp[1];       mz = temp[2];
mxx = mx*mRes;    myy = my*mRes;      mzz = mz*mRes;
Kalman_MadgwickAHRS (axx,ayy,azz,gxx,gyy,gzz,,mxx,myy,mzz,dt);
computeAngles();
if (yaw > 2*pi || yaw > pi) yaw -= 2*pi;
if (yaw < -pi)                      yaw += 2*pi;
```

```
        yaw += thetai;

        // FILTRO DE KALMAN - POSICION (axx = aT , ayy = aN)
        float A11 = 0.97500944, A12 = 0.00098747, A21 = -0.31226807, A22 = 0.99984320;
        float W11 = 0.00000049582, W21 = 0.00099994762;
        float L11 = 0.02499055, L21 = 0.31226807;

        dxenc = ne*dt*cos(yaw);
        dyenc = ne*dt*sin(yaw);
        ne = 0;

dxEncoder<<dxenc<<endl;
dyEncoder<<dyenc<<endl;

        acelx = axx*cos(yaw) + ayy*cos(yaw);
        acely = axx*sin(yaw) + ayy*sin(yaw);

        dxo = (A11*dx_old + A12*dxp_old) + 0*W11*acelx + L11*dxenc;
        dxpo = (A21*dx_old + A22*dxp_old) + 0*W21*acelx + L21*dxenc;
        dyo = (A11*dy_old + A12*dyp_old) + 0*W11*acely + L11*dyenc;
        dypo = (A21*dy_old + A22*dyp_old) + 0*W21*acely + L21*dyenc;

        dx_old = dxo; dxp_old = dxpo;
        dy_old = dyo; dyp_old = dypo;
        x_old = dxo + x_old;
        y_old = dyo + y_old;

        dest[0] = x_old;
        dest[1] = y_old;
        dest[2] = yaw;
}

void encoderISR(void){
        ne++;
}

void escapeManual(int sig){
        digitalWrite(ENB,LOW);
        delay(100);
        digitalWrite(N3,LOW);
        digitalWrite(N4,LOW);
```

```
            delay(100);

            digitalWrite(ENA,LOW);
            delay(100);
            digitalWrite(N1,LOW);
            digitalWrite(N2,LOW);
            delay(100);

            printf("\Escape Manual!!!\n");
            exit(0);
}
```