Anexos

ANEXO A: BUBBLE

```
function [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs,
comsys)

    %COLLISIONAVOIDANCE - Makes the robot avoid obstacles.
    %
    % [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs)
    %
    %
    % v_ref: Velocity reference
    % phi_ref: Orientation reference
    % status: State of the state-machine
    % robot: Instance of RobotModel
    % other_robs: An array of all the other robots
    % comsys: Instance of the inter-robot communication system

    % This is how you read properties of the robot.
    % Note that many properties of RobotModel, especially regarding
    % position, orientation, velocity and acceleration are read-only, i.e.
    % there will be an error if you write robot.x = 10
    % To see an overview of the properties, type "doc RobotModel".

    x_current = robot.x;
    y_current = robot.y;
    x_goal = robot.x_goal;
    y_goal = robot.y_goal;

    Status = robot.vars.ca_status;

    % vars is a struct where you can save your own variables on the robot
    %robot.vars.ca_status = 0;
    %robot.vars.myString = 'I am a robot';

    % If you need laserscan-data, get it like this:
    %[laserscan, beam_angles] = LaserScanner(robot, other_robs);
    % The function takes the actual (groundtruth) positions of the other
    % robots to calculate an array of measurements. Type "help
    % LaserScanner" for more information.
    % Note that the LaserScanner-function takes a significant amount of
    % time and will slow down your simulation.

    % If you want to work with the suspected positions reported to the
    % communication system (not necessarily the actual ones), you can do
    % something like this:
    %otherRobPositions = comsys.positions;
```

```
    % The following are defined by the example algorithm itself
    STATE_MACHINE = struct('TRAVELING2GOAL',0,'ARRIVING2GOAL',1,'ONGOAL',2);

    Darrival = 0.25;
    Dbreaking = 3 * Darrival;
    MaxVbreaking = 0.8;
    MinVbreaking = 0.0;


    d2goal = sqrt((y_goal - y_current)^2 + (x_goal - x_current)^2);
    phi_goal_ref = atan2((y_goal - y_current), (x_goal - x_current));


    if (d2goal > Dbreaking)
        Status = STATE_MACHINE.TRAVELING2GOAL;
    else
        if (d2goal > Darrival)
            Status = STATE_MACHINE.ARRIVING2GOAL;
        else
            Status = STATE_MACHINE.ONGOAL;
        end
    end
    % Computing the necessary reference to arrive to the goal
    phi_goal_ref = atan2((y_goal-y_current),(x_goal-x_current));
    [dist, beam_angles] = LaserScanner(robot, other_robs);



    switch Status

        case STATE_MACHINE.TRAVELING2GOAL

            [phi_ref,v_ref] = bubble(robot, dist, beam_angles, phi_goal_ref);


        case STATE_MACHINE.ONGOAL
            v_ref = 0;
            phi_ref = phi_goal_ref;

        case STATE_MACHINE.ARRIVING2GOAL
            a = (MaxVbreaking - MinVbreaking ) / ( Dbreaking - Darrival);
            b = (MaxVbreaking - a * Dbreaking );
            v_ref = a * d2goal + b;
            phi_ref = phi_goal_ref;

    end


end % function
```

```
function [phi_ref, v_ref] = bubble(robot, dist, beam_angles, phi_goal_ref)


     N = length(dist);                                  % Number of sensors
     laser_readings = dist;
     k =(0.2 : (3 - 0.2) / (N - 1) : 3);
     bubble_boundary = ones(1,N) .* k .*1.*1.8 ;     % Definition of the
bubble
     i = 1;
     Obstacles = 0;

    for i = 1 : N
        lectura = laser_readings(i);                % Read the sensors and
compare with the bubble
        frontera = bubble_boundary(i);
        if (lectura <= frontera)
            Obstacles = 1;
        end

        i = i + 1;
    end

    alphadist = 0;
    sumdist = 0;

    if Obstacles

        for i = 1:N
            alphadist = alphadist + beam_angles(i) * dist(i);
            sumdist = sumdist + dist(i);
        end

        phi_ref = alphadist / sumdist;          % Compute de rebound angle

        if abs(phi_ref-robot.phi)<0.01          % In case of conflict, turn.
            if phi_ref>0
               phi_ref = pi/2;
            else
               phi_ref = -pi/2;
            end
        end

         minD = min(dist);
         v_ref = robot.type.vMax * max(0.4, minD/30);   % Compute de distance
    else

         v_ref = robot.type.vMax;                   % If there are no
obstacles,
         phi_ref = phi_goal_ref;                    % go to the goal
    end


end %function
```

ANEXO B: VFF

```
function [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs,
comsys)
    %COLLISIONAVOIDANCE - Makes the robot avoid obstacles.
    %
    % [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs)
    %
    % This is an example algorithm without any actual collision avoidance.
    % The robot will simply move to the goal and stop there.
    %
    % v_ref: Velocity reference
    % phi_ref: Orientation reference
    % status: State of the state-machine
    % robot: Instance of RobotModel
    % other_robs: An array of all the other robots
    % comsys: Instance of the inter-robot communication system

    % This is how you read properties of the robot.
    % Note that many properties of RobotModel, especially regarding
    % position, orientation, velocity and acceleration are read-only, i.e.
    % there will be an error if you write robot.x = 10
    % To see an overview of the properties, type "doc RobotModel".
    x_current = robot.x;
    y_current = robot.y;
    x_goal = robot.x_goal;
    y_goal = robot.y_goal;
    vMax=robot.type.vMax;

    Status = robot.vars.ca_status;

    % vars is a struct where you can save your own variables on the robot
    %robot.vars.ca_status = 0;
    %robot.vars.myString = 'I am a robot';

    % If you need laserscan-data, get it like this:
    %[laserscan, beam_angles] = LaserScanner(robot, other_robs);
    % The function takes the actual (groundtruth) positions of the other
    % robots to calculate an array of measurements. Type "help
    % LaserScanner" for more information.
    % Note that the LaserScanner-function takes a significant amount of
    % time and will slow down your simulation.

    % If you want to work with the suspected positions reported to the
    % communication system (not necessarily the actual ones), you can do
    % something like this:
    %otherRobPositions = comsys.positions;

    % The following are defined by the example algorithm itself
    STATE_MACHINE = struct('TRAVELING',0,'ONGOAL',2);

    % Necessary constants for the system
```

```
    Darrival = 0.5;%
0.25;                                                                   %
CHANGE!!!
    Fct = +5;             % Target atracting force
    Fer = -1;             % Object repulsive force
    w = 0.25;             % Helps while reducing the oscilation

    % Active window parameters
    Nx_cells = 33;    %[cells]
    Ny_cells = 33;    %[cells]
    cell_size = 0.1;  %[m/cell]

    % State machine transitions controller
    % The state machine is composed by two states controled by the
    % distance to the goal
    %
    % |----TRAVELING----+---ONGOAL----+
    %               Darrival        Goal

    d2goal = sqrt((y_goal - y_current)^2 + (x_goal - x_current)^2);

    if (d2goal > Darrival)
        Status = STATE_MACHINE.TRAVELING;
    else
        Status = STATE_MACHINE.ONGOAL;
    end

    % Depending on the state the system uses the VFF algorithm at the
    % desired speed or the same one at speed zero.
    %
    %              |
    %        V_VFF +-------+
    %              |       +
    %            0 |       +-------+
    %              |-------+-------+
    %                  Darrival  Goal

    % Computing the atracting forces
    [Ft_x,Ft_y] =
TargetAttractingForce(Fct,robot.x,robot.y,robot.x_goal,robot.y_goal);

    % Computing the repulsive forces
    [laserscan, beam_angles] = LaserScanner(robot, other_robs);
    [Fr_x,Fr_y] =
ObstaclesRepulsiveForce(Fer,Nx_cells,Ny_cells,cell_size,laserscan,beam_angles
);

    % Computing cos_theta
    cos_theta = CosComputation(Fr_x,Fr_y,robot.v,robot.phi);

    % Adjusting repulsive forces
    [Fr_x,Fr_y] = AdjustedRepulsiveForce(w,Fr_x,Fr_y,cos_theta);

    % Adding all the forces to the system
    R = [Ft_x+Fr_x Ft_y+Fr_y];
```

```
    % Normalizing such vector
    R = R / norm(R);

    % Computing orientation
    delta = atan2(R(2),R(1));

    % Computing speed (Note: if Fr is zero, cos_theta will be zero)
    V = vMax * (1 - abs(cos_theta));

    % Debuging sytem
    if (robot.id == 1)
        DataRepresentation(R,'Phiref')
    end

    phi_ref = delta;

    switch Status
        case STATE_MACHINE.TRAVELING
            v_ref = V;
        case STATE_MACHINE.ONGOAL
            v_ref = 0;
    end

end % function
```

```
function DataRepresentation(var,title_str)
% Represents a data in a plot

MaxData = 150;

persistent Data;
persistent i;

if isempty(i)
    if (isvector(var))
        Data=zeros(MaxData,length(var));
        i = 1;
    else
        error('Data representation can not afford a matrix of information');
    end
end

if i >= MaxData
    i = 1;
end

for j = 1:length(var)
    Data(i,j) = var(j);
end
i = i + 1;


plot(Data)
title(title_str)
pause(0.001)


end
```

```
function [ Ft_x, Ft_y ] =
TargetAttractingForce(Fct,x_current,y_current,x_target,y_target)
% The function receives the current position of the robot and the target,
% and computes the target-attracting force.

d_t = sqrt((y_target-y_current)^2 + (x_target-x_current)^2);

Ft_x = Fct * (x_target - x_current) / d_t;
Ft_y = Fct * (y_target - y_current) / d_t;

end
```

```
function [ Fr_x, Fr_y ] =
ObstacleRepulsiveForce(Fer,Nx_cells,Ny_cells,cell_size,laserscan,beam_angles)
% The function receives the measurements of the laser and computes the
% repulsive forces of the system.

% Note: We have to satisfy the function (1) of the paper, thus we define
% x_0=0 and y_0=0 as well as x_t=d*sin(ang) y_t=d*cos(ang).

C = 1;  % Asumes that in the histogram there was allways 1 hit when a robot
is hit.

Fr_x = 0;
Fr_y = 0;

% Change of reference system. The laser is on the (0,0)
x_0 = 0;
y_0 = 0;

% Active window parameters
W_x = cell_size * Nx_cells / 2;
W_y = cell_size * Ny_cells / 2;

% Checking every measurement of the laser
for i = 1:length(laserscan)
    x_t = laserscan(i) * cos(beam_angles(i));
    y_t = laserscan(i) * sin(beam_angles(i));

    % Checking if the target is on the active window
    if (abs(x_t - x_0) <= W_x)    % Inside of the x part of the window
        if (abs(y_t - y_0) <= W_x)    % Inside of the y part of the window
            Fr_x = Fr_x + Fer * C * (x_t-x_0) / (laserscan(i)^3);
            Fr_y = Fr_y + Fer * C * (y_t - y_0) / (laserscan(i)^3);
        end
    end
end

end
```

```
function cos_theta = CosComputation(Fr_x,Fr_y,V_linear,phi)
% This function computes one special cosinus that is used in some parts of
% the algorithm as a specific variable.

Fr_norm = norm([Fr_x Fr_y]);

if (Fr_norm)
    % Exists a repulsion force
    v_x = V_linear * cos(phi);
    v_y = V_linear * sin(phi);
    if V_linear
        cos_theta = (v_x * Fr_x + v_y * Fr_y) / (Fr_norm * V_linear);
    %cos_theta=cos_theta*180/pi;
    else
        cos_theta = 0;
    end

else
    % Without repulsion force the value should be zero
    cos_theta = 0;
end

end
```

```
function [ Fr_x, Fr_y ] = AdjustedRepulsiveForce(w,Fr_x,Fr_y,cos_theta)
% Computes one special adjustement of the repulsive force of the system to
% reduce oscilations while affornting linear obstacles.

    Fr_x = w * Fr_x + (1 - w) * Fr_x * (-cos_theta);
    Fr_y = w * Fr_y + (1 - w) * Fr_y * (-cos_theta);

end
```

ANEXO C: ORCA

```
function [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs,
comsys)

    %COLLISIONAVOIDANCE - Makes the robot avoid obstacles.
    %
    % [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs)
    % v_ref: Velocity reference
    % phi_ref: Orientation reference
    % status: State of the state-machine
    % robot: Instance of RobotModel
    % other_robs: An array of all the other robots
    % comsys: Instance of the inter-robot communication system

    % Current position of the robot
    x_current = robot.x;
    y_current = robot.y;
    % position of the goal
    x_goal = robot.x_goal;
    y_goal = robot.y_goal;

    % Status of the robot
    Status = robot.vars.ca_status;
    STATE_MACHINE = struct('TRAVELING2GOAL',0,'ARRIVING2GOAL',1,'ONGOAL',2);


    Darrival = 0.25;
    Dbreaking = 3*Darrival;
    MaxVbreaking = 1;
    MinVbreaking = 0.0;
    % Note: With the values Darr=0.25 / Dbre=3*Darr / MaxVb=0 and MinVb=0
    % the system stops on the goal with an error no larger than 0.01m but
    % this is forcing an emergency stop.

    % State machine transitions controller
    % The state machine is composed by three states controled by the
    % distance to the goal
    %
    % |----TRAVELING2GOAL_FREE----+----ARRIVING2GOAL----+---ONGOAL----+
    %                          Dbreaking             Darrival        Goal


    % ra and rb defined in ORCA
    tau = 2;
    % As there is not a velocity sensor, get the position and the velicity
from other_robs
    numo = length(other_robs);
    vo = zeros(numo,1);
    xo = zeros(numo,1);
    yo = zeros(numo,1);
    phio = zeros(numo,1);
    for k = 1:numo
        xo(k) = other_robs(k).x;
        yo(k) = other_robs(k).y;
```

```
        po(k,:)=[xo(k), yo(k)];
        vo(k) = other_robs(k).v;
        phio(k) = other_robs(k).phi;
    end

    % Distance to the goal and heading needed
    d2goal = sqrt((y_goal - y_current)^2 + (x_goal - x_current)^2);
    phi_goal_ref = atan2((y_goal - y_current), (x_goal - x_current));

    % Define the Status of the robot depending of the distance to the goal
    if (d2goal > Dbreaking)
        Status = STATE_MACHINE.TRAVELING2GOAL;
    else
        if (d2goal > Darrival)
            Status = STATE_MACHINE.ARRIVING2GOAL;
        else
            Status = STATE_MACHINE.ONGOAL;
        end
    end

    switch Status

        case STATE_MACHINE.TRAVELING2GOAL
            [v_ref,phi_ref] = ORCA(robot, po, vo, phio, tau, phi_goal_ref);
        case STATE_MACHINE.ONGOAL
            v_ref = 0;
            phi_ref = phi_goal_ref;
        case STATE_MACHINE.ARRIVING2GOAL
            a = (MaxVbreaking - MinVbreaking ) / ( Dbreaking - Darrival);
            b = (MaxVbreaking - a * Dbreaking );
            v_ref = a * d2goal + b;
            phi_ref = phi_goal_ref;
    end


end % function
```

```
function [pt1,pt2,c2,r2,m1,m2] = VO(Pa,Pb,ra,rb,tau)
    % This function calculate the Velocity Obstacle geometrically.
    % Computing the tangency points and the slope.

    c1 = Pb - Pa;       % Center and radius of the circles
    r1 = ra + rb;
    c2 = c1 / tau;
    r2 = r1 / tau;

    d = norm(c1 - c2);
    teta = asin((r1 - r2)/d);
    teta = teta + pi/2;

    M1 = [cos(teta) -sin(teta); sin(teta) cos(teta)];
    teta = -teta;
    M2 = [cos(teta) -sin(teta); sin(teta) cos(teta)];

    aux = [c2 / norm(c2) * r2];

    pt1 = c2 + M1 * aux;      % The tangency point is calculate as a sum of
vectors
    pt2 = c2 + M2 * aux;

    gamma = atan2(c1(2)-c2(2),c1(1)-c2(1));
    alpha = gamma + abs(teta - pi / 2);
    m1 = tan(alpha);
    m2 = tan(gamma - abs(teta - pi / 2));
    end
```

```
function [Dent,Excp,m3,med] = prohibida(pt1,pt2,c2,r2,v,m1,m2)
    % This function calculate if the relative velocity is inside or outside
    % the VO, and calculate a parameter Excp that depends of the relative
    % position of v and the VO. It helps to calculate u and n.

    % Slope of the line that connect the two nips.This line divide the two
    % regions for Excp.
    m3 = (pt1(2) - pt2(2)) / (pt1(1) - pt2(1));

    % Slope of the vector relative velocity
    mv = v(2) / v(1);
    med = [0 0]';

    % If the VO is completelly horizontal
    if(m3 == Inf)

        % Distance from v to the first line
        d1 = calculadist(v,[v(1) m1 * (v(1) - pt1(1)) + pt1(2)]');

        % Distance fromv  to the second line
        d2 = calculadist(v,[v(1) m2 * (v(1) - pt2(1)) + pt2(2)]');

        % Distancefrom the two points with the line of v.
        d = calculadist([v(1) m2 * (v(1) - pt2(1)) + pt2(2)]',[v(1) m1 *
(v(1) - pt1(1)) + pt1(2)]');

         % Calculating if v is inside or outside the VO.
        if((  ((d1 <= (d / 2)) && (d2 < d)) || ((d2 <= (d / 2)) && (d1 <
d)) )&&( ((pt1(1) <= 0) &&(v(1) <= 0)) || ((pt1(1)>=0)&&(v(1)>=0)) ))
            if( (abs(v(1))>abs(pt1(1))) || ( (abs(v(1))<abs(pt1(1)))&&((v(1)-
c2(1))^2+(v(2)-c2(2))^2<=r2^2) ) )
                Dent = 1;
            else
                Dent = 0;
            end
        else
            Dent = 0;
        end

    % If the VO is completelly vertical
   elseif(m3 == 0)

        % Distance from v to the first line
        d1 = calculadist(v,[((v(2) - pt1(2)) / m1) + pt1(1) v(2)]');

        % Distance fromv  to the second line
        d2 = calculadist(v,[((v(2) - pt2(2)) / m2) + pt2(1) v(2)]');

        % Distancefrom the two points with the line of v.
        d = calculadist([((v(2) - pt2(2)) / m2) + pt2(1) v(2)]',[((v(2) -
pt1(2)) / m1) + pt1(1) v(2)]');
```

```
        if((  ((d1<=(d/2))&&(d2<d)) ||
((d2<=(d/2))&&(d1<d)) )&&( ((pt1(2)<=0)&&(v(2)<=0)) ||
((pt1(2)>=0)&&(v(2)>=0)) ))
            if( (abs(v(2))>abs(pt1(2))) || ( (abs(v(2))<abs(pt1(2)))&&((v(1)-
c2(1))^2+(v(2)-c2(2))^2<=r2^2) ) )
                Dent = 1;
            else
                Dent = 0;
            end
        else
            Dent = 0;
        end
    % any other case
    else
        d1 = calculadist(v,intersection(pt1,m1,v,m3));
        d2 = calculadist(v,intersection(pt2,m2,v,m3));
        d = calculadist(intersection(pt1,m1,v,m3),intersection(pt2,m2,v,m3));

        if(norm(mv) == Inf)
            med = [v(1) m3 * (v(1) - pt1(1)) + pt1(2)]';
        elseif(mv == 0)
            med = [((v(2) - pt1(2)) / m3) + pt1(1) v(2)]';
        else
            med = intersection(pt1,m3,v,mv);
        end

        if((  ((d1<=(d/2))&&(d2<d)) ||
((d2<=(d/2))&&(d1<d)) )&&( ((med(2)<=0)&&(v(2)<=0)) ||
((med(2)>=0)&&(v(2)>=0)) ))
            if( (calculadist(v,[0 0]')>calculadist(med,[0 0]')) ||
( (calculadist(v,[0 0]')<calculadist(med,[0 0]'))&&((v(1)-c2(1))^2+(v(2)-
c2(2))^2<=r2^2) ) )
                Dent = 1;
            else
                Dent = 0;
            end
        else
            Dent = 0;
        end
    end
    % To calculate in which region is v.
    Excp = 0;
    if Dent == 0
        c = (pt1(2)-m3*pt1(1));
        if  c > 0
            if (v(2) - m3 * v(2)) > c
                Excp = 0;
            else
                Excp = 1;
            end
        else
            if (v(2) - m3 * v(2)) > c
                Excp = 1;
            else
                Excp = 0;
            end
```

```
        end
    end
    % If Excp is 1, then the smaller distance to the lines is the distance
    % to the nip.

end
```

```
function [v_ref, phi_ref] = ORCA(robot,po , vo, phio, tau, phi_goal_ref)
    ra = 0.5;
    rb = 0.5;
    j = length(vo);       % Number of other robots
    cont =1;
    chocaria = 0;

    for k = 1 :j
        % Calculate if and with which robots it is possible to colide.
        disactual(k,1) =  calculadist([robot.x,robot.y], po(k,:)) ;
        disminima(k,1) = (norm(vo(k)) + robot.v) * tau;
        if disactual(k) <= disminima(k)
            chocaria(cont,1) = k;
            cont = cont +1;
        end
    end

    if chocaria > 0

        % Va and Pa is the current position of the robot.
        va =  robot.v * [cos(robot.phi), sin(robot.phi)]';
        vref = 1 * [cos(phi_goal_ref), sin(phi_goal_ref)]';
        Pa = [robot.x, robot.y]';

        for k = 1:length(chocaria)               % For each obstacle
            vb = vo(chocaria(k)) * [cos(phio(chocaria(k))),
sin(phio(chocaria(k)))]';   % The velocity of the obstacle
            Pb =
po(chocaria(k),:)';                                                  % And
the position

            v = va - vb;                         % relative velocity
            n = [0 0]';
            u = 0;

            [pt1,pt2,c2,r2,m1,m2] = VO(Pa,Pb,ra,rb,tau);                    %
Compute the VO
            [Dent,Excp,m3,med] = prohibida(pt1,pt2,c2,r2,v,m1,m2);          %
Check if v is inside or outside the VO
            [u,n] = calculaUN(pt1,pt2,c2,r2,v,m1,m2,m3,med,Dent,Excp);      %
Compute u and n

            % each robot define a restiction, now it solves a quadprog
            % problem to optimize the velocity (closer to Vpref with all
            % this restrictions.

            puntoA(k, :) = va' + (0.5 * u) * n';
            pendiente = -n(1) / n(2);
            A(k,:) = [-pendiente, 1];
            b(k,:) = puntoA(k,2) - pendiente * puntoA(k,1);
            % choose the half-plane (< or >) that is in the direcction of n
            if A(k,:) * (va + (3 * u)*n) > b(k)
                A(k,:) = -A(k,:);
                b(k) = -b(k);
```

```
            end
            % If v is outside the VO, the half-plane is the opposite (to
            % NOT enter in the VO.

           if ~Dent
                b(k) = -b(k);
                A(k,:) = -A(k,:);
           end
        end

            H = 2 * eye(2);
            A(end+1,:) = [1 0];
            A(end+1,:) = [0 1];
            b(end+1,:) = robot.type.vMax;
            b(end+1,:) = robot.type.vMax;
            A(end+1,:) = [-1 0];
            A(end+1,:) = [0 -1];
            b(end+1,:) = robot.type.vMax;
            b(end+1,:) = robot.type.vMax;
            % minimice |v - vpref|
            va = quadprog(H,-2 * vref,real(A),real(b));
            v_ref = norm(va);
            phi_ref = atan2 ( va(2) , va(1));
    else
         v_ref = robot.type.vMax; % If there are no obstacles, max vel to the
goal.
         phi_ref = phi_goal_ref;
    end


end
```

```
function [P]=intersection(p1,m1,p2,m2)
    % This function calcualte the intersection point of two lines
    % defined each one by a point and its slope.

    if(m1 == Inf)
        P = [p1(1) ((p1(1)-p2(1)) * m2) + p2(2)]';
    elseif(m1 == 0)
        P=[((p1(2) - p2(2)) / m2) + p2(1) p1(2)]';
    elseif(m2 == Inf)
        P = [p2(1) ((p2(1) - p1(1)) * m1) + p1(2)]';
    elseif(m2 == 0)
        P=[((p2(2)- p1(2)) / m1) + p1(1) p2(2)]';
    else
        X = (p2(2) - p2(1) * m2 + p1(1) * m1 - p1(2)) / (m1 - m2);
        Y = (X - p1(1)) * m1 + p1(2);
        P = [X Y]';
    end
end
```

```
function [u,n] = calculaUN(pt1,pt2,c2,r2,v,m1,m2,m3,med,Dent,Excp)
    % This function calculate the vector n and distance u, minimal distance
    % that brings v to the end of the VO region.
    u = 0;
    n = [0 0]';
    % The array and minimal distance to the first line
    n1 = [pt1(2) -pt1(1)'];
    m = n1(2) / n1(1);
    paux = intersection(pt1,m1,v,m);
    n1 = paux-v;
    n1 = n1 / sqrt(n1(1)^2 + n1(2)^2);
    df1 = double(calculadist(paux,v));

    % The array and minimal distance to the second line
    n2 = [pt2(2) -pt2(1)]';
    m = n2(2) / n2(1);
    paux = intersection(pt2,m2,v,m);
    n2 = paux - v;
    n2 = n2 / sqrt(n2(1)^2 + n2(2)^2);
    df2 = double(calculadist(paux,v));
    % The array and minimal distance to the circle
    df3 = 20000;
    n3 = c2 - v;

    if(Excp == 1)
        n1 = pt1 - v;                       % If the closet point from the
lines to v is not in VO,
        n1 = n1 / sqrt(n1(1)^2 + n1(2)^2);  % Then the closet point from the
lines of VO to V is the nip point.
        df1 = double(calculadist(pt1,v));
        n2 = pt2 - v;
        n2 = n2 / sqrt(n2(1)^2 + n2(2)^2);
        df2 = double(calculadist(pt2,v));
        df3 = calculadist(c2,v) - r2;
    end
```

```
    if(Dent ==0 )
        df3 = calculadist(c2,v) - r2;
    end

    if(Dent == 1)

    if(m3 == Inf)
        if((abs(v(1)) < abs(pt1(1))) && ((v(1)-c2(1))^2 + (v(2) -c2(2))^2 <=
r2^2))
            df3 = r2 - calculadist(c2,v);
            n3 = v - c2;
            n3 = n3 / sqrt(n3(1)^2 + n3(2)^2);
        end
    elseif( m3==0 )
        if((abs(v(2)) < abs(pt1(2))) && ((v(1) - c2(1))^2 + (v(2) - c2(2))^2
<= r2^2))
            df3 = r2 - calculadist(c2,v);
            n3 = v - c2;
            n3 = n3 / sqrt(n3(1)^2 + n3(2)^2);
        end
    else
        if((calculadist(v,[0 0]') < calculadist(med,[0 0]')) && ((v(1) -
c2(1))^2 + (v(2) - c2(2))^2 <= r2^2))
            df3 = r2 - calculadist(c2,v);
            n3 = v - c2;
            n3 = n3 / sqrt(n3(1)^2 + n3(2)^2);
        end
    end

    end

    if((df3 < df2) && (df3 < df1))  % Select the smallest.
                n = n3/norm(n3);
                u = df3;
    else
            if(df1 <= df2)
               n = n1/norm(n1);
               u = df1;
            else
               n = n2/norm(n2);
               u = df2;
            end
    end

end


function [D] = calculadist(P1,P2)
    % This function calculate the distance between two points P1 and P2
    D = sqrt((P1(1) - P2(1))^2 + (P1(2) - P2(2))^2);
end
```

ANEXO D: IMPORTADOR

```
% This program read in MATLAB the results of the simulation and calculate
% some datas to study the performance of the algorithm
clear all
clc
archivo = 'vff.txt';    % This has to be change with the name of the
simulation that you want to analyse
valorchoque = 0.5;               % A distance smaller than this one, would
mean a collision. It depends of the size of the robots
Datos = importdata(archivo);     % Import data from .txt

t0 = Datos.data(:,1);
id0 = Datos.data(:,2);           % Write them in arrays
x0 = Datos.data(:,3);
y0 = Datos.data(:,4);
% v0 = Datos.data(:,5);
% a0 = Datos.data(:,6);
% phi0 = Datos.data(:,7);
% omega0 = Datos.data(:,8);
% alpha0 = Datos.data(:,9);

totrob = max(id0);               % Number of robots
tamano = length(t0);             % Size of the arrays (totrob*sim)
sim = tamano/totrob;             % Number of simulation
i = 1;
k = 1;
DIST = zeros(1,totrob);          % predefinition of Distance

for j = 1 : tamano               % Change the vectors to a matrix, one column
for each robot and one row for each simulation step
    if i > totrob
        i = 1;
        k = k + 1;
    end
    X(k,i) = x0(j);
    Y(k,i) = y0(j);
    T(k,:) = t0(j);
    i = i + 1;
end

distancia = zeros(1,totrob);     % Calculate the distance traveled for each
robot.
for n = 1 : sim -1
    for m = 1 :totrob
       DIST(n,m) = calculadist([X(n,m), Y(n,m)], [X(n+1,m), Y(n+1,m)]);
       distancia(m) = distancia(m) + DIST(n,m);
    end
end

distancia_media = 0;             % Calculate the total mean distance
for kk = 1 : totrob
```

```
    distancia_media = distancia (kk) + distancia_media;
end
distancia_media = distancia_media / totrob;

choques = 0;
disrob = zeros(sim, 1);          % Calculate the total number of collisions,
taking into account that when they colide, the distance
                                 % Between robots is smaller than their size
for a = 1 :totrob
    for b = a + 1 : totrob
        disrob = sqrt( (X(:, a) - X(:, b)).^2 + (Y(:, a) - Y(:, b)).^2 );    %
Distance between robots
        overlap = (disrob < valorchoque);                                    %
One when they colide/Are coliding
        choques = choques + sum(diff(overlap)==1);                           %
This is to count collision just once.
    end
end


 TF3 = strcmp(archivo,'simulacion_3.txt');          % Define the initial
Stage
 TF4 = strcmp(archivo,'simulacion_4.txt');
 TF6 = strcmp(archivo,'simulacion_6.txt');
 TFp = strcmp(archivo,'simulacion_pared.txt');
 TFa = strcmp(archivo,'simulacion_alea.txt');
 TF100 = strcmp(archivo,'scalability_test.txt');
 distminmed = 0;

if TF3 %  archivo == 'simulacion_3.txt'

    Stage(1,:)=[1 8 4.5 0 10 1];
    Stage(2,:)=[9 1 3.9168 0 2 9];
    Stage(3,:)=[9 9 1 0 1 2];
end

if TF4
    Stage(1,:)=[2 8 0.2 0 8 1];
    Stage(2,:)=[8 1 2.3168 0 2 8];
    Stage(3,:)=[2 2 2.4168 0 8 7];
    Stage(4,:)=[8 7 4.6168 0 2 3];
end

if TF6
    Stage(1,:)=[2 8 4.5 0 10 1];
    Stage(2,:)=[9 1 2.3168 0 2 9];
    Stage(3,:)=[2 1 1 0 9 8];
    Stage(4,:)=[9 9 4.3168 0 2 0.5];
    Stage(5,:)=[2 5 0.4 0 9.3 5];
    Stage(6,:)=[9.2 4 1 0 0.5 6];
end

if TFp

    Stage(1,:) = [3 0.86 5.2482 0 5.5839 9.1606];
```

```
    Stage(2,:) = [2.6 4.7 0 0 2.6 4.7];
    Stage(3,:) = [3.6 4.7 0 0 3.6 4.7];
    Stage(4,:) = [5.3 4.7 0 0 5.3 4.7];
    Stage(5,:) = [6.2 4.7 0 0 6.2 4.7];
    Stage(6,:) = [7.1 4.7 0 0 7.1 4.7];

 end

 if TFa
    Stage(1,:) = [5.1980    9.7479    1.0470     0    0.3814    7.8687];
    Stage(2,:) = [ 6.4134    3.9920    0.0375     0    0.8204    3.2946];
    Stage(3,:) = [4.2806    8.7480    0.4294   0    0.2981    0.9449];
    Stage(4,:) = [ 4.2632    0.3333    0.2090     0     9.9095    3.0970];

 end


 if TF100
    L=10;     % Number of robots per line
    C=10;     % Number of robots per coln
    SBR=2;  % Space between robots

    Stage=zeros(L*C,6);
    %Stage(i=1:N,:)=[x_o(i) y_o(i) phi_o(i) v_o(i) x_g(i) y_g(i)]


    n=1;
    for i=1:L
        for j=1:C
            %disp(['Robot' num2str(n) ' L=' num2str(i) ' C=' num2str(j) '
pose ' num2str((i-1)*SBR-L*SBR/2) ' ' num2str(-j*SBR)])
            Stage(n,:)=[(i-1)*SBR-L*SBR/2 -j*SBR 0 0  (i-1)*SBR-L*SBR/2
+j*SBR];
            n=n+1;
        end
    end

 end
 % Calculate the minimal mean average as the average of the minimal
 % distance (straight line from the start point to the end one)
for k = 1 : totrob
    distmin(k,1) = calculadist(Stage(k,1:2),Stage(k,5:6));
    distminmed = distminmed + distmin(k,1);
end
% The nominal time is the time spent divide with the minimal time, the time
% that the robot with the larger minimal distance would take at the maximum
% velocity (1)

tiempo_normalizado = T(end)/max(distmin);
distminmed = distminmed/totrob;
distancia_normalizada = distancia_media/distminmed;
format short
Metricas = [choques;tiempo_normalizado;distancia_normalizada]
```

Simulación y Comparación de Algoritmos de Evitación de Obstáculos para múltiples robots

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Simulación y Comparación de Algoritmos de Evitación de Obstáculos para Múltiples Robots

Autor: Carlos Cobos Bandera

Tutor: Jesús Capitán Fernández

Dep. Ingeniería de sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Simulación y Comparación de Algoritmos de Evitación de Obstáculos para Múltiples Robots

Autor:
Carlos Cobos Bandera

Tutor:
Jesús Capitán Fernández
Profesor Ayudante Doctor

Dep. de Ingeniería sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Simulación y Comparación de Algoritmos de Evitación de Obstáculos para Múltiples Robots

Autor: Carlos Cobos Bandera

Tutor: Jesús Capitán Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia

A mis maestros buenos

Agradecimientos

Quiero mostrar mi agradecimiento a todas aquellas personas que me han apoyado y han confiado en mí en todo momento, haciendo una mención especial a mis padres, Jose María y Mari Carmen, que aun exigiendo siempre lo mejor de mí, siempre me han apoyado y han estado orgullosos de mi trabajo. A mi hermano Sergio, que sin su ayuda todo esto no habría sido posible. A mis compañeros de piso, que me aguantan incluso en época de exámenes. A mi abuela Paquita, con su apoyo incondicional propio de una buena abuela como es. A mi novia Rosa, que desde que empecé con ella lo que quedaba de carrera pasó volado, y sin una sola mancha en el expediente, sin su apoyo este trabajo no habría sido posible. Y sin nunca olvidar a mi tío Pepe, una persona que tras los malos resultados en los primeros parciales en primero de carrera, seguía confiando en mí cuando los demás empezaban a dudar, que padecía de cáncer y nunca llegó a conocer que finalmente saqué todas las asignaturas en primero, y cuatro años después toda la carrera con la entrega de este TFG. Allá donde estés, gracias.

Carlos Cobos Bandera

Sevilla, 2015

Resumen

El objetivo de este trabajo de Fin de Grado, es la comparación de diversos algoritmos de evitación de obstáculos para múltiples robot, capaces de garantizar una navegación sin colisiones en un espacio con múltiples UAVs. Para ello, se han codificado en MATLAB dichos algoritmos, y se han simulado, analizado y comparado los resultados obtenidos.

Para dicho fin se ha estructurado el trabajo de la siguiente manera:

En primer lugar se encuentra la introducción, donde se sitúan estos algoritmos en el contexto actual, para así entender por qué es interesante en este paradigma el estudio que se ha elegido como objeto de este Trabajo de Fin de Grado.

A continuación se introducen algunos de los algoritmos sencillos que se conocen hoy día, así como se describen los que para este trabajo se han codificado, simulado y comparado.

Posteriormente, se repasa brevemente el software que se ha utilizado, tanto el programa de MATLAB que realiza las simulaciones, el programa con el que se analizan los resultados y cómo lo hace, así como el controlador de versiones que se ha utilizado para la codificación de los algoritmos.

Por último, se presentan los resultados de las simulaciones y con ellos las comparaciones de los distintos algoritmos, terminando con las conclusiones obtenidas tras la realización de este Trabajo y las líneas de trabajo futuras que plantea este estudio.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

UAV	Unmanned Aircraft Vehicle. Vehículo aéreo no tripulado.
UAS	Unmanned Aircfaft Systems. Sistemas aéreo no tripulado.
RPAS	Remotely Piloted Aircraft Systems.
Bug	Algoritmo de esquivación de obstáculos [7].
VO	Velocidad Obstáculo.
RVO	Velocidad Obstáculo Recíproca.
$VO_B^A(v_B)$	Velocidad Obstáculo de A respecto al obstáculo B que viaja a v_B.
$A \oplus B$	Suma de Minkowski.
$\mid$	Tal que.
$\in$	Pertenece.
$\notin$	No pertenece.
$\forall$	Para todo.
$\cap$	Unión.
ORCA	Optimal Reciprocal Collision Avoidance. Velocidad Obstáculo Recíproca Optimizada.
VFF	Virtual Force Field. Campo de Fuerzas Virtuales
α_R	Ángulo de rebordeo.
H_j	Punto de encuentro con un obstáculo en el algoritmo BUG.
Q_m	Punto más cercano al destino hasta el momento.
$CA^{\tau}_{A\|B}(V_B)$	Velocidad que evita colisiones de un robot A respecto a un obstáculo B que se mueve a v_B durante un tiempo τ.
$\exists$	Existe.
τ	Tau, constant de tiempo en la que el algoritmo ORCA evita colisiones.
p_B	Posición del agente B.
p_A	Posición del agente A.
r_A	Radio del robot A.
r_B	Radio del robot B.
α_i	Ángulo del sensor láser i.
D_i	Distancia que mide el sensor láser i.
Circunnavegación	Rodeo del obstáculo.
v_A^{opt}	Velocidad óptima de A. La que le dirige hacia su objetivo.
v_B^{opt}	Velocidad óptima de B.
u	Distancia mínima entre la velocidad relativa y la frontera del obstáculo de velocidades.
v^{new}	Velocidad nueva.

1 INTRODUCCIÓN

"Una vez que hayas volado, caminarás por la tierra mirando al cielo, donde estuviste y donde tardarás en volver."

Leonardo da Vinci.

En primer lugar, se va a situar en qué contexto surge el estudio de este Trabajo de Fin de Grado y por qué resulta interesante.

Desde las primeras apariciones de los UAV, UAS, RPAS o drones, como son popularmente conocidos, han ido obteniendo un crecimiento y un desarrollo exponencial, hasta haberse puesto tan de moda, que hoy día se pueden encontrar a la venta en un Stand a la entrada de cualquier centro comercial.

Los motivos son evidentes. Por un lado, en las misiones aéreas que pudieran también ser realizadas por una aeronave convencional, el coste de usar una pequña aeronave no tripulada es mucho menor, por lo que da lugar a nuevas aplicaciones antes impensables.

Por otro lado, cuando se trata de misiones peligrosas, acaba con el riesgo para el piloto a bordo, es por ello que la principal aplicación hoy día sea en el ámbito militar.

Pero no es ésta la única, debido a su pequeño tamaño (en general) y a su maniobrabilidad, dan lugar a un abanico de posibilidades inmenso, en especial, cuando no tenemos una única aeronave, sino un enjambre, un sistema con gran cantidad de ellos.

Ilustración 1. Enjambre de UAVs

De todas las aplicaciones para las que los UAVs son ampliamente utilizados hoy en día, merecen la pena mencionar los siguientes:

1. Para llevar una carga de pago demasiado pesada de un punto A a un punto B, en lugar de utilizar una aeronave tripulada de gran tamaño, con un elevado coste, se puede realizar esta operación no con uno sino con un conjunto de UAVs sincronizados que son capaces de entre ellos llevar y colocar con precisión dicha carga de pago, ahorrando tiempo y coste [1].

Ilustración 3. Quadrotor

Ilustración 2. Quadrotor de DHL transportando un paquete

2. La empresa de ventas online Amazon ® , ya lleva años probando el envío de paquetes pequeños a domicilio con drones [2].

Ilustración 4. Amazon PrimeAir

3. Para protección de flora y fauna:

Se utilizan enjambres de drones para el seguimiento y la monitorización del estado y la posición de animales en peligro de extinción, de manera que se pueda garantizar la protección y supervivencia de las mismas, así como de los bosques, especialmente en verano son muy utilizados desde hace años para la vigilancia de los mismos, para localizar pirómanos o para la detección temprana de incendios, proporcionando una ayuda enorme a los equipos de bomberos.[3]

También están estudiando el uso de drones de mayor tamaño para realizar labores de prevención y extinción de incendios durante la noche [4].

Ilustración 5. UAV de Bomberos

4. En 2014, surgió de un concurso a manos de un grupo de universitarias españolas, la creación de un Dron para el transporte de órganos para trasplantes, así como toda una red de logística para sustituir el transporte de órganos de pequeña-mediana distancia por el transporte aéreo con drones [5].

De todos estos ejemplos, se observa en casi todos una característica común:

Situaciones en las que un enjambre o conjunto de drones se encuentran realizando una misión alrededor de la misma zona, en las que es necesario apoyar al sistema de control y guiado del dron para garantizar que no colisionen entre ellos ni con el entorno mientras realizan su misión.

Se trata por tanto de algoritmos, que deben ser computacionalmente sencillos, que puedan realizar las computadoras de UAVs de no muy alto coste, con los sensores que contengan a bordo, de manera autónoma y descentralizada, y garanticen una navegación sin colisiones.

El objetivo de este trabajo es analizar y comparar algoritmos capaces de evitar obstáculos y a un conjunto de UAVs entre si, es decir, evitar colisiones. Por tanto, habrá que tener en cuenta las restricciones que aportan los vehículos, que unas veces pueden parar en seco y girar (quadcopter) y otras veces no (ala fija). Para ello, se codifican los algoritmos en MATLAB y se simulan en un simulador aportado por el Departamento. A continuación se estudian los resultados, y se sacan las conclusiones oportunas.

Antes de continuar, es necesario comentar que los algoritmos estudiados no son aplicables únicamente a UAVs, sino también para otras múltiples aplicaciones, desde la misma función para robots en el suelo (De hecho, en los algoritmos simplificamos suponiendo el movimiento del UAV únicamente en un plano) hasta otras funciones tan dispares como son las simulaciones gráficas de ordenador, videojuegos, para la ingeniería del tráfico[6].

Aclarado esto, a lo largo de la memoria se utilizará el término de UAV, dron o robot indistintamente.

2 ALGORITMOS

"Cuando todo parezca estar en tu contra, recuerda que los aviones despegan con el aire en contra, no a favor."

Henry Ford

Se presentan a continuación una serie de algoritmos básicos de evitación de obstáculos que han sido estudiados. Se verá una breve introducción de todos ellos, tanto los que se han simulado, como los que no, mostrando a su vez el motivo por el que éstos no han sido probados en este Trabajo de Fin de Grado.

2.1. Algoritmos no simulados

En este apartado se muestran los algoritmos de evitación de obstáculos sencillos, que no hemos programado para este trabajo, tales como el algoritmo Bug y el Velocity Obstacle, se comienza por el más sencillo de éstos, el BUG.

2.1.1 Bug Algorithm

Los algoritmos de la familia Bug [7] son los más conocidos para resolver el problema de la navegación en un entorno de dos dimensiones desconocido. Llevan al robot de un punto A a un punto B ambos conocidos, sin ninguna información del entorno, o se paran en caso de que el objetivo sea inalcanzable, realizando una serie de simplificaciones, tales como tomar objetos como puntos y los sensores como ideales libres de ruido. Como se verá a continuación, no está pensado para su uso en un entorno con obstáculos en movimiento.

Desde el algoritmo Bug1, se han realizado múltiples modificaciones mejorando el mismo. Se procede a explicar el Bug1:

1) Se mueve directamente hacia el objetivo T, hasta que:
    a. El objetivo es alcanzado, se para el algoritmo.
    b. Se encuentra un obstáculo. Se define el punto H_j de encuentro y se va al paso 2.

2) Se circunnavega en el sentido de las agujas del reloj actualizando continuamente el punto Q_m, la posición más cercana al objetivo hasta el momento, hasta que:
    a. Se alcanza el objetivo, se para el algoritmo.
    b. Se alcanza H_j de nuevo (se ha dado la vuelta completa al obstáculo). Entonces se comprueba si la línea que une el punto Q_m con el objetivo T está o no obstruida por el obstáculo:
        i. Si es así, el objetivo es inalcanzable. Se para el algoritmo.

ii. En caso contrario, se vuelve hasta Q_m por el camino más corto, y se vuelve al paso 1.

Existen numerosas modificaciones o mejoras a este algoritmo, como es el Bug2.

Éste algoritmo es análogo al anterior, con la diferencia de que cuando se bordea el obstáculo, no es necesario dar al menos una vuelta completa, sino que cuando el robot intersecta con la línea recta que une los puntos H_j y el objetivo T, se dirige ya hacia éste.

Se muestra a continuación resultados de estos algoritmos obtenidos en otros estudios[7]:

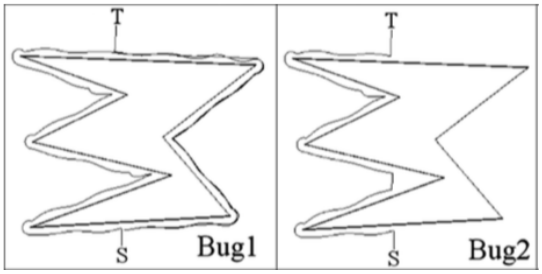

Ilustración 6. Representación del algoritmo Bug1 y Bug2

El motivo por el que no se ha implementado este algoritmo es evidente: En las bases de este algoritmo está implícito que se trate de obstáculos estáticos, ya que cuando se alcanza uno, se rebordea completamente, tarea dificilmente realizable cuando se trate de un obstáculo también móvil. Es por ello que de antemano se estiman malos resultados de este algoritmo ante las simulaciones a realizar, por lo que no se implementa.

2.1.2 Reciprocal Velocity Obstacle

Un algoritmo mucho más interesante es el Reciprocal Velocity Obstacle [8], en el que se resuelve el problema de navegación con varios agentes en tiempo real con objetos tanto estáticos como en movimiento.

Este algoritmo se basa en las velocidades de cada UAV, es necesario por ello conocer tanto la posición como la velocidad de los demás agentes u obstáculos, por lo que el dron necesitará llevar a bordo los sensores necesarios para esto. Estos sensores además, se suponen ideales. Como simplificación se asume también que los drones son circulares, que se mueven como hemos dicho en un plano y que los otros agentes también intentarán de la misma manera evitar una colisión.

Para entender el algoritmo hay que aclarar primero algunos conceptos:

2.1.2.1 Velocidad obstáculo

La Velocidad Obstáculo (Velocity Obstacle) se define de la siguiente manera [8]:

Sean A y B dos drones, sea $D(\mathrm{p},r)$ un disco de radio r y centrado en p:

$$D(\mathrm{p},r)=\{\mathrm{q} \mid |\mathrm{q}-\mathrm{p}| < r \},$$

se define $VO^{\tau}_{A|B}$, la velocidad obstáculo de A inducida por B en el tiempo τ, como:

$$VO^{\tau}_{A|B} = \{v|\exists\, t \in [0,\tau] :: tv \in D(p_B - p_A, r_A + r_B)\}$$

Que es el conjunto de velocidades v_A para A tal que produciría una colisión en algún momento del tiempo con el obstáculo B moviéndose a una velocidad v_B.

La Velocidad Obstáculo se define geométricamente en el plano de velocidades como se observa en la Ilustración 7.

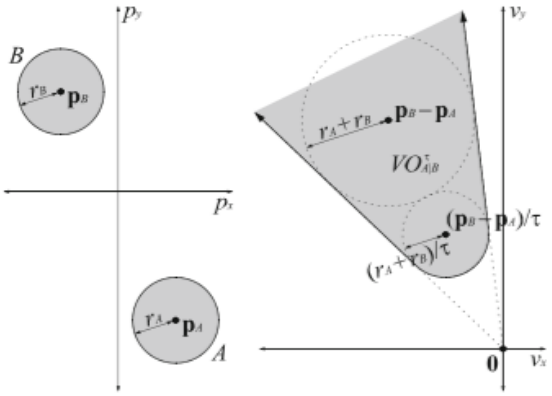

Ilustración 7. Velocidad obstáculo

Si la velocidad relativa entre A y B, v_A-v_B $_A \in VO^{\tau}_{A|B}$, entonces A y B chocarán en algún momento.

Consecuentemente, si se encuentra fuera de este espacio, podremos garantizar que A y B no se chocarán al menos durante un tiempo τ.

Se puede generalizar de la siguiente manera:

Supongamos que se conoce la velocidad de B, pero sabemos por su posición, y asumiendo tiene las mismas características que A, tales como velocidad máxima, velocidad óptima... se puede suponer un conjunto de velocidades V_B que podrá tomar B.

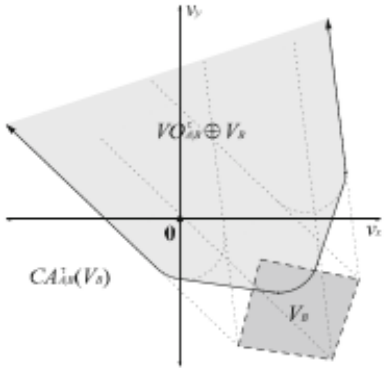

Ilustración 8. Collision avoidance

Siendo A $\oplus$ B la suma de Minkowsky de dos objetos A y B,

$A \oplus B = \{a+b \mid a \in A, b \in B\}$,

Para cualquier conjunto de velocidades V_B, si $\mathbf{v}_B \in V_B$ y $\mathbf{v}_A \in VO^{\tau}_{A|B} \oplus V_B$, entonces se garantiza que A y B no colisionarán a dicha velocidad por lo menos en un tiempo τ. Con esto podemos definir la velocidad que debe seguir el robot, $CA^{\tau}_{A|B}(V_B) = \{ v \mid v \notin VO^{t}_{A|B} \oplus V_B \}$.

Este algoritmo no se ha simulado ya que se ha optado por una modificación del mismo (ORCA) que consigue seleccionar de entre todas las velocidades fuera de la velocidad obstáculo del otro agente, la velocidad óptima.

2.2. Algoritmos simulados

En este apartado introducimos los algoritmos que hemos programado, simulado y analizado en este trabajo, que son el Bubble o burbuja; el Virtual Force Field, o campo de fuerzas virtuales y el ORCA, Optimized Reciprocal Colision Avoidance, comenzando por el más sencillo de éstos, el Bubble.

2.2.1. Bubble

Para este algoritmo [9] se considera que los drones constan de un anillo de sensores de distancia (infrasonidos, infrarrojos, láser…) que cubren un abanico de 180º delante del mismo.

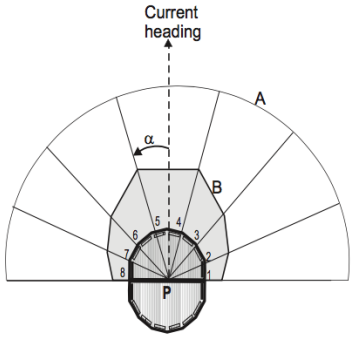

Ilustración 9. Rango de cobertura de los sensores

Con las medidas de éstos, se define una burbuja de sensibilidad, de la siguiente forma:

$bubble\_boundary[i]=K_i*V*delta\_t$ [9]

Siendo V la velocidad del dron, delta_t el intervalo de tiempo entre las sucesivas medidas del sensor, y K_i una constante escaladora. En nuestras simulaciones, toma valores distribuidos entre 0.02 y 3. Se observa que la burbuja está dinámicamente ajustada por la distancia que es capaz de recorrer el dron a la velocidad V en el intervalo de tiempo delta_t, asegurando que la burbuja nunca exceda el valor máximo de los sensores.

El algoritmo va continuamente midiendo y comprobando si hay obstáculos dentro de la burbuja, y el funcionamiento es que se describe a continuación.

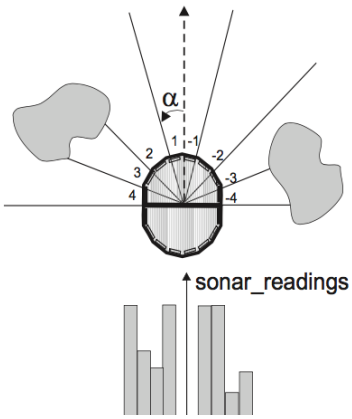

Ilustración 10. Lectura de obstáculos en Bubble

El dron se dirige directamente hacia su objetivo, hasta que encuentra uno o más obstáculos dentro de la burbuja sensible. En ese momento, calcula un "ángulo de rebordeo" en la dirección en la que haya menor densidad de obstáculos, y se dirige en dicha dirección hasta que deje atrás al obstáculo y el objetivo sea visible, dirigiéndose hacia el objetivo, hasta que se encuentre otro obstáculo.

La forma de calcular dicho "ángulo de rebordeo" es muy sencilla, considerando α_i el ángulo el sensor i, y D_i la medida de dicho sensor, y N el número total de sensores, el "ángulo de rebordeo" puede calcularse mediante la siguiente fórmula:

$$\alpha_R = \frac{\sum_{i=\frac{-N}{2}}^{\frac{N}{2}} \alpha_i \cdot D_i}{\sum_{\frac{-N}{2}}^{\frac{N}{2}} D_i}$$

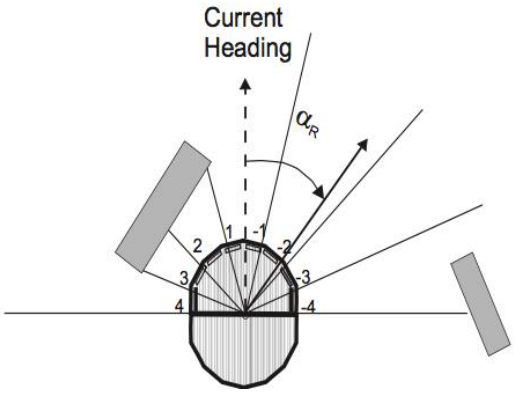

Ilustración 11. Ángulo de rebordeo

Hay que tener en cuenta un caso peculiar, en el que la distribución de obstáculos es simétrica respecto a la dirección actual del UAV, en el que la fórmula anterior nos da un resultado $\alpha_R = 0$, dirigiéndonos diréctamente hacia el obstáculo.

En las simulaciones, después de haber programado el algoritmo, se ha dado este caso, y se ha corregido fácilmente añadiendo las siguientes líneas:

```
if abs(phi_ref - robot.phi) < 0.01

        if phi_ref > 0
        phi_ref = pi / 2;
    else
        phi_ref = -pi / 2;
    end
end
```

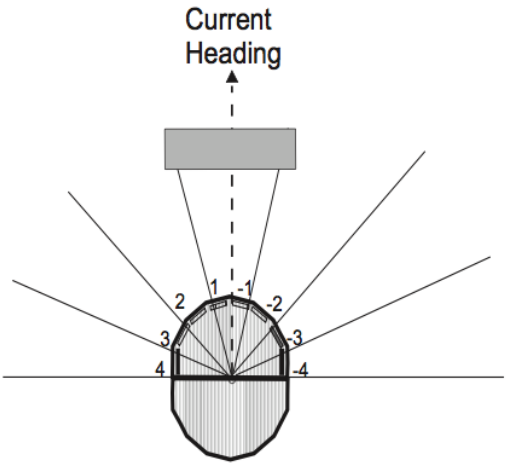

Ilustración 12. Error de cálculo del algoritmo Bubble

De manera que si se da este caso, con el primer bucle *if,* cuando se encuentra ante un obstáculo en estas condiciones, gira 90º. Esto es lo que nos indican en [9]. Hasta aquí funcionaría con obstáculos estáticos, pero si el otro obstáculo es móvil y sigue el mismo criterio, como en nuestro caso, girarían en la misma dirección, uno al lado del otro, alejándose ambos de sus objetivos. Para evitar esto, se crea el segundo bucle *if,* que según la dirección, gira en un sentido u en otro. Con esto, como se verá en las simulaciones, se soluciona el problema.

Este algoritmo, al igual que el Bug, está pensado para objetivos estáticos, sin embargo, a diferencia del Bug, al reaccionar antes de alcanzar al obstáculo, con una pequeña modificación puede mejorar su funcionamiento ante obstáculos móviles.

Además de la mencionada anteriormente, si se sigue este algoritmo tal cual, solo varía la dirección, pero no el módulo de la velocidad. Esto, lleva a múltiples colisiones. Al reducir la velocidad cuando se encuentra ante un obstáculo, se obtienen mejores resultados. El valor de dicha velocidad, se convierte en un parámetro que rige el comportamiento del algoritmo, cuanto mayor sea la velocidad, antes llega al objetivo, sobre todo en situaciones simples, pero ante otras más complicadas, puede llevar a colisiones o divergencias.

El código del algoritmo puede observarse en el Anexo A.

2.2.2. Virtual Force Field

Para el algoritmo de campos de fuerzas virtuales se partía de una implementación del mismo existente para el simulador.

Se basa en lo siguiente: Cada obstáculo, ejerce una fuerza virtual repulsiva sobre el UAV, y el objetivo ejerce una fuerza virtual atractiva sobre el mismo, por lo que el UAV se moverá en la dirección resultante de cada una de estas fuerzas virtuales, de manera que tiende a esquivar a los obstáculos y acercarse al objetivo. Las fuerzas repulsivas y atractivas van multiplicadas por unas constantes que sirven para ajustar el comportamiento del robot.

La constante que multiplica al vector de repulsión, significará lo conservador o temerario que será nuestro UAV. Si la constante es muy grande, el vector de fuerzas virtuales de repulsión afectará más, y el robot será más conservador, por el contrario si esta constante adquiere un valor pequeño, nuestro robot se acercará más a los obstáculos.

La constante que multiplica al vector de fuerza virtual de atracción, indicará cuánto le afectan los obstáculos a nuestro UAV. Si tiene un valor muy grande, apenas se inmutará con los obstáculos.

Por supuesto el valor de estas constantes se puede convertir en variable, dependiente por ejemplo, de la distancia al destino, para obtener así mejores resultados.

En la Ilustración 13 que se muestra a continuación, podemos ver una representación de los tres vectores que hemos comentado.

Solo se le ha realizado una modificación al código, para corregir un error que daba cuando se trataban de obstáculos estáticos al calcular el coseno de la dirección.

El código completo de este algoritmo puede encontrarse en el Anexo B de este documento.

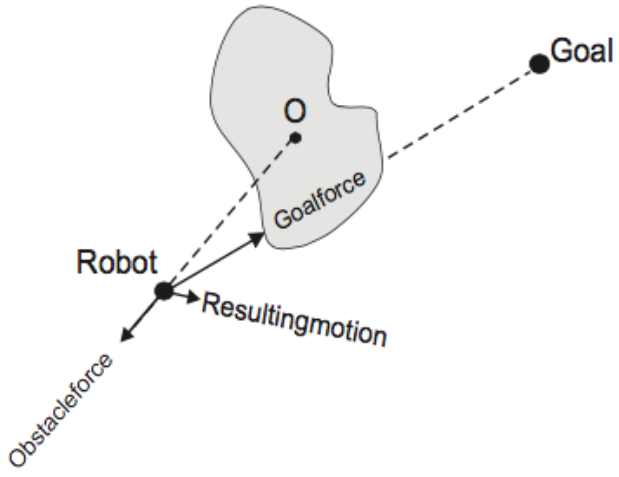

Ilustración 13. Fuerzas virtuales en VFF

2.2.3. Optimal Reciprocal Collision Avoidance

En el apartado **2.1.2** se ha visto la definición de la Velocidad Obstáculo, y la velocidad que evita las colisiones, $CA^{\tau}_{A|B}$. El algoritmo ORCA (Optimal Reciprocal Collision Avoidance) surge mejorando el anterior, para de todo el conjunto de velocidades que evita la colisión, elegir la velocidad óptima, es decir, la que menos nos desvía de nuestro objetivo, a la que llamamos $ORCA^{\tau}_{A|B}$.

Se tiene en cuenta que cada UAV tiene una velocidad máxima, posteriormente se verá para qué es necesaria, así como una velocidad óptima o preferida, que la hemos definido como la que se dirige hacia el objetivo.

Se puede definir geométricamente la velocidad $ORCA^{\tau}_{A|B}$ de la siguiente manera [8]:

Se asume que tanto A como B se mueven a su velocidad óptima o preferida $\mathbf{v_A}^{opt}$ y $\mathbf{v_B}^{opt}$, y supongamos que se encuentran camino de una colisión, es decir, $v_A - v_B \in VO^{\tau}_{A|B}$. Se define en este punto "u" como la distancia mínima desde el punto de la velocidad relativa de los UAV ($v_A - v_B$) hasta la frontera del cono de la Velocidad Obstáculo, y se define n como el vector unitario que apunta en ésta dirección. Matemáticamente:

$$u = (\text{argmin}_{v \in \partial VO^{\tau}_{A|B}} \| v - (\mathbf{v_A}^{opt} - \mathbf{v_B}^{opt}) \|) - (\mathbf{v_A}^{opt} - \mathbf{v_B}^{opt}).$$

De esta forma, u es el cambio más pequeño que hay que realizar a la velocidad relativa de A y B para evitar la colisión en tiempo τ. Es interesante comentar como la responsabilidad de evitar la colisión la comparte cada robot, de manera que el robot A cambia su velocidad en al menos, $\frac{1}{2}$ u, y asume que B cambiará en otro medio.

La velocidad será por tanto: $ORCA^{\tau}_{A|B} = \{v \mid (v - (v_A^{opt} + \frac{1}{2} u)) \cdot n >= 0\}$

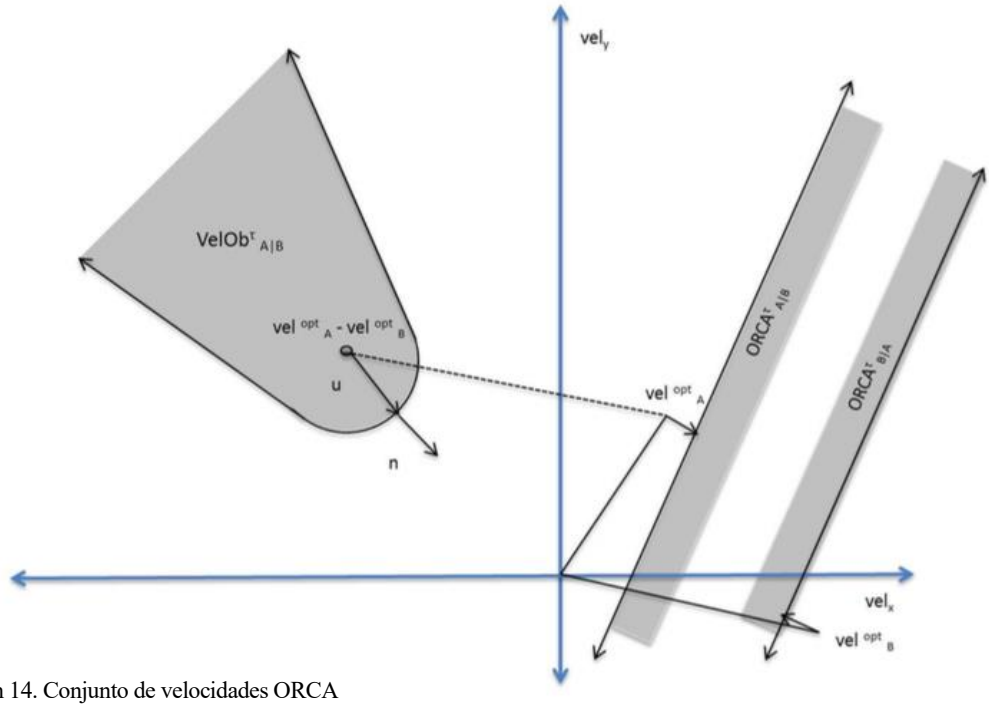

Ilustración 14. Conjunto de velocidades ORCA

Como podemos observar en la Ilustración 14, geométricamente esto genera un semiplano, en la dirección que apunta n, tal que si la velocidad queda dentro de éste, se garantiza que A y B no colisionarán, siendo la velocidad óptima la mínima que va a la frontera ($ORCA^{\tau}_{A|B} = (v_A^{opt} + \frac{1}{2}u)\cdot n$). En el caso de que la velocidad relativa no esté dentro de la velocidad obstáculo, se generará el semiplano contrario, y también se tiene en cuenta la restricción para escoger una velocidad que no nos acerque a la colisión.

Esto es así cuando tenemos 2 UAVs, para un caso general con N robots, nos encontramos ante un problema de "n-Body Collision Avoidance" [8].

El acercamiento a este problema es el siguiente:

Calculamos el conjunto de velocidades que están permitidas para A de manera que se evite la colisión con los N robots al menos en un tiempo τ, como la intersección de los semiplanos de las velocidades permitidas por cada robot. $ORCA_A^{\tau} = D(\mathbf{0}, v^{max}) \cap_{A-=B} \underset{\forall\ ORCA_{A|B}^{\tau}}{ORCA_{A|B}^{\tau}}$

Donde tenemos también en cuenta la restricción de la velocidad máxima de A (una circunferencia en el campo de velocidades).

Se trata por tanto de encontrar, la velocidad $\mathbf{v}^{new} = \underset{\mathbf{v}\in ORCA_A}{\operatorname{argmin}}\|\mathbf{v}-\mathbf{v}^{opt}\|$

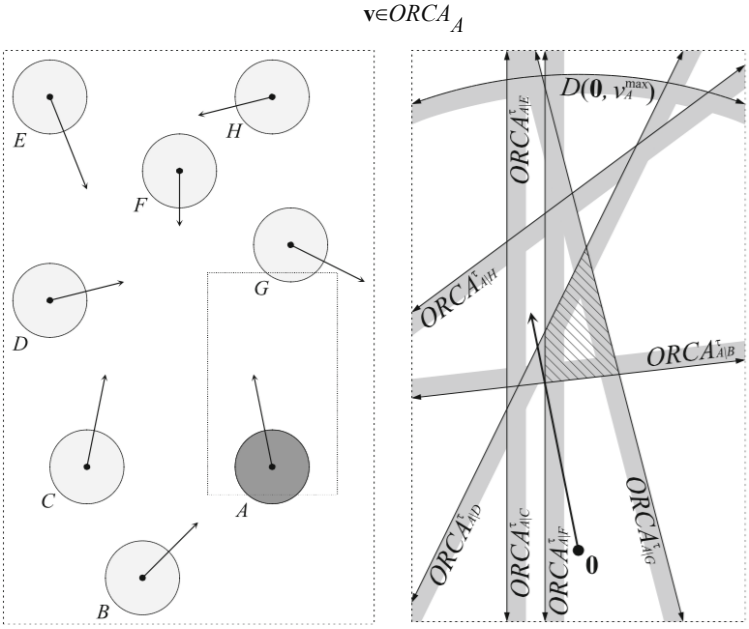

Ilustración 15. Conjunto de restricciones de múltiples ORCA

Esto se ha calculado resolviendo un problema de programación cuadrática, minimizando el módulo de la diferencia de velocidades entre la que calculamos y la óptima o preferida, sujeto a una serie de restricciones lineales (ORCA).

Nótese que la restricción de la velocidad máxima no es lineal, pero se ha linealizado.

El código del algoritmo puede encontrarse en el anexo C.

3 SOPORTE

Allá donde estés actúa siempre como si acabaras de llegar, y aún queda un amplio margen de mejora. Solo así alcanzarás la excelencia.

Anónimo

En este apartado se muestran cuáles son las herramientas que han sido utilizadas para la realización de este trabajo, que son principalmente MATLAB, con un programa simulador de algoritmos que fue prestado por el departamento, un programa realizado por el alumno que analiza los resultados de las simulaciones, así como el Mercurial, un controlador de versiones, con su interfaz TortoiseHg para Macintosh, para ayudar en la codificación de los algoritmos.

3.1 Simulador de MATLAB

Se ha utilizado en MATLAB un simulador que ha sido proporcionado por el departamento, el cual permite crear una serie de escenarios, con tantos UAVs como sean deseados, dándoles a éstos una posición inicial, una posición final o destino, y una orientación inicial.

A continuación, permite simular en tiempo real el movimiento de los drones siguiendo el algoritmo que se haya seleccionado.

El simulador permite escoger varios escenarios para realizar sucesivas simulaciones siempre con el mismo algoritmo, y tras de la visualización, guarda en una carpeta determinada un archivo ".txt" con los resultados de la simulación, donde se encuentran para cada instante de tiempo, valores como la posición, velocidad y aceleración de cada uno de los UAVs.

3.1.1 Estructura del programa

El programa está dividido en una serie de carpetas, como se observa en la siguiente Ilustración (16).

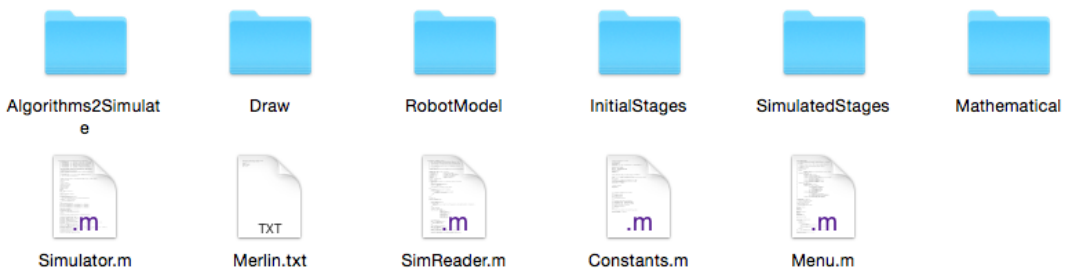

Ilustración 16. Organización de las carpetas del simulador

El programa principal es el *Menu.m*. Llamando a esta función desde Matlab es como el simulador comienza a funcionar.

Dentro de la carpeta SimulatedStages, hay una capeta por cada algoritmo, y dentro de esta, otra carpeta por cada escenario en el que se ha realizado una simulación con este algoritmo, con los archivos de los datos de dichas simulaciones.

Por otro lado, en la carpeta Algorithms2Simulate, se encuentra una carpeta por cada algoritmo. Ésta, debe contener al menos un archivo llamado *CollisionAvoidance.m,* y los demás programas que éste necesite.

3.1.2 Funcionamiento

Como ya se ha dicho, el programa comienza a funcionar al llamar a la función Menu. A partir de ahí, nos encontramos con el siguiente menú por el que se navega simplemente seleccionando las opciones con el teclado dentro de la *Command Window* de Matlab.

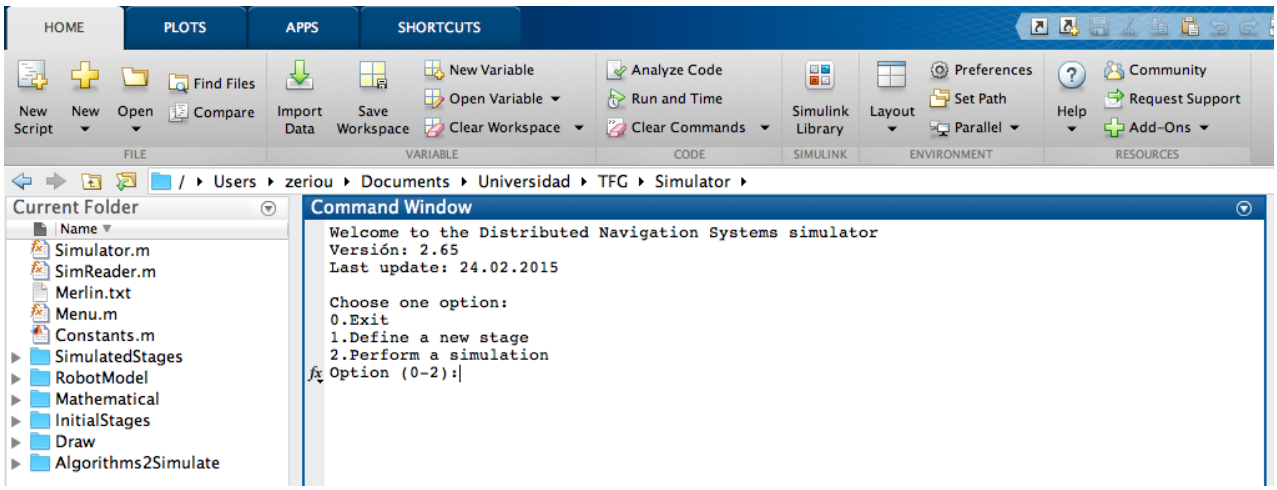

Ilustración 17. Menú del simulador

Se puede, por ejemplo, crear un nuevo escenario, con esta versión, solo a mano. Tras elegir el número de robots y el tamaño del escenario, se procede a situar la posición inicial y final de éstos con un simple click de ratón, como se observa en la Ilustración 19.

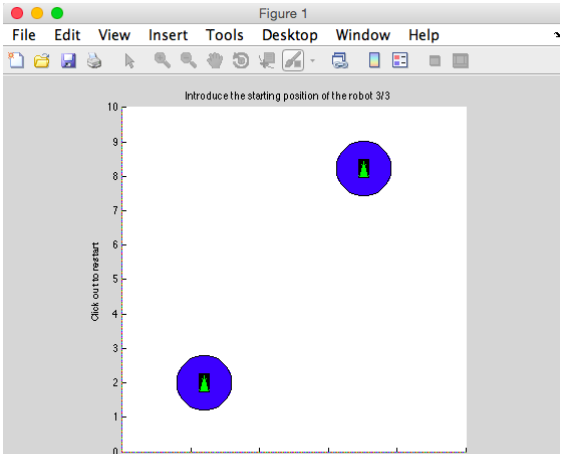

Ilustración 18. Creación de escenarios en el simulador

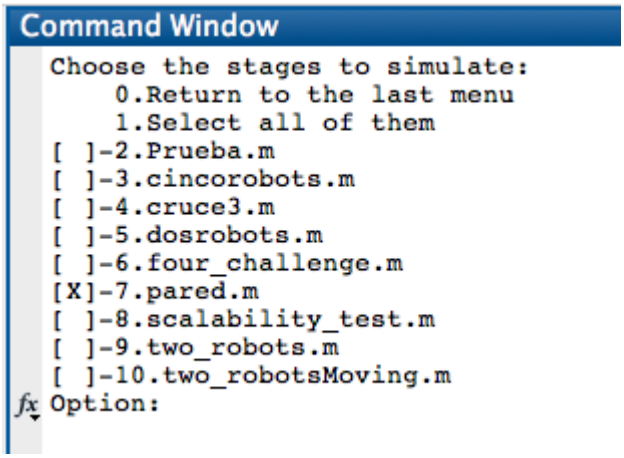

Ilustración 19. Selección de escenarios

Después de situarlos, se nombra al escenario, y podemos pasar a simular. Antes de simular, te pide que selecciones los escenarios que quieres simular (Ilustración 18), pudiendo seleccionar tantos como se quiera.

A continuación se elige el algoritmo a utilizar, y comienza la simulación, que tiene la interfaz que se observa en la Ilustración 20.

Donde podemos encontrar dentro del cuadrado del mapa la posición de los UAV con forma de quadrotor (los puntos negros), la posición final de cada uno (las cruces azules), la dirección actual del UAV (punto rojo) y la dirección de referencia (hacia el objetivo, o para evitar a un obstáculo), representada con el punto verde.

A la derecha encontramos un gráfico de barras con la velocidad de cada UAV, y un parámetro binario llamado active, que indica si el UAV ha llegado a su destino (valor 0) o aún no (valor 1).

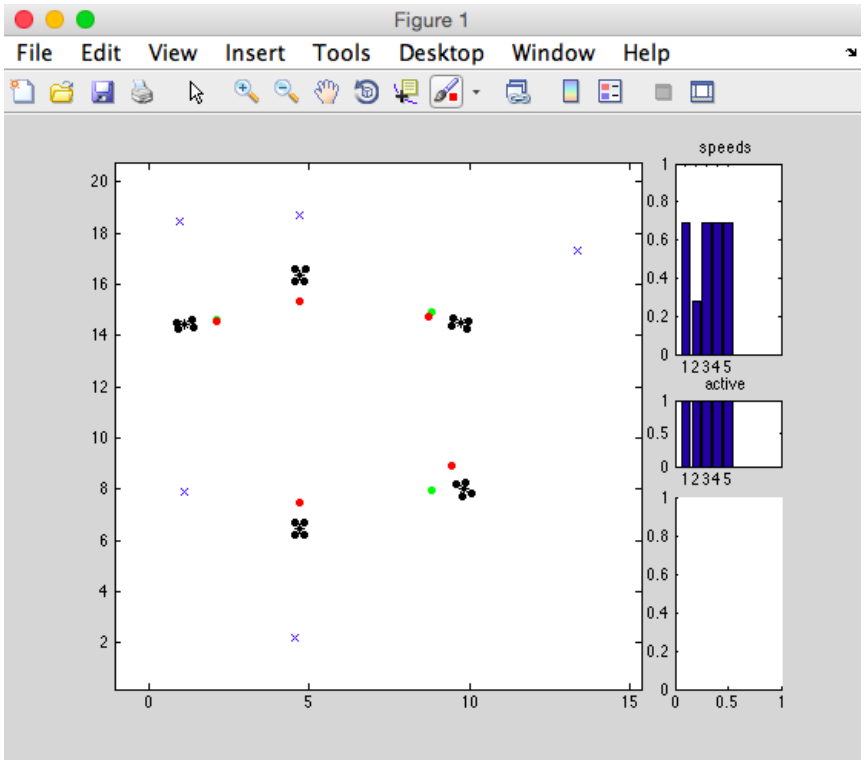

Ilustración 20. Interfaz de la simulación

Al terminarse la simulación, el programa te indica el tiempo que ha durado y los steps que ha realizado, y al salir del programa te muestra la llamada a la función Menu que tienes que realizar para repetir exactamente el mismo experimento.

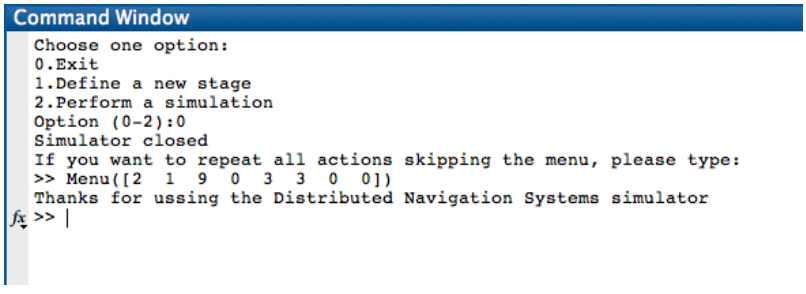

Ilustración 21. Final de una simulación

Se genera también al terminar, un archivo con todos los datos, como ya se ha comentado, que tiene la forma que se observa en la Ilustración 22.

cruce3.txt

```
Simulation name: cruce3
Number of robots: 3

Creation time:
YYYY MM DD hh mm ss
%d   %d %d %d %d %d
2015  9 13 18 24 45

     clock robot        x        y        v        a      phi    omega    alpha
       [s]   [1]      [m]      [m]    [m/s]  [m/s²]    [rad]  [rad/s] [rad/s²]
        %f    %d       %f       %f       %f       %f       %f       %f       %f
      0.00     1     4.69     3.52     0.00     0.00     1.57     0.00     0.00
      0.00     2     4.61     8.67     0.00     0.00     4.75     0.00     0.00
      0.00     3     2.12     5.89     0.00     0.00     5.18     0.00     0.00
      0.20     1     4.69     3.53     0.22     2.17     1.55    -0.39    -3.93
      0.20     2     4.61     8.66     0.22     2.17    -1.52     0.23     2.31
      0.20     3     2.12     5.88     0.22     2.17    -1.09     0.39     3.93
      0.30     1     4.69     3.56     0.36     1.45     1.50    -0.58    -1.82
      0.30     2     4.61     8.63     0.36     1.45    -1.51     0.01    -2.24
      0.30     3     2.14     5.86     0.36     1.45    -1.04     0.63     2.39
      0.40     1     4.70     3.60     0.32    -0.42     1.45    -0.43     1.50
      0.40     2     4.61     8.59     0.49     1.29    -1.50     0.04     0.35
      0.40     3     2.16     5.83     0.35    -0.13    -1.01    -0.01    -6.42
      0.50     1     4.70     3.64     0.48     1.65     1.44     0.13     5.61
      0.50     2     4.61     8.54     0.48    -0.14    -1.52    -0.37    -4.10
      0.50     3     2.18     5.79     0.52     1.70    -0.99     0.39     3.97
      0.60     1     4.71     3.68     0.48    -0.08     1.43    -0.31    -4.47
      0.60     2     4.61     8.49     0.49     0.08    -1.57    -0.62    -2.49
      0.60     3     2.21     5.75     0.50    -0.15    -0.97    -0.16    -5.46
      0.70     1     4.72     3.73     0.49     0.17     1.38    -0.58    -2.71
      0.70     2     4.61     8.44     0.49     0.06    -1.64    -0.77    -1.51
      0.70     3     2.24     5.70     0.65     1.41    -0.97     0.30     4.56
      0.80     1     4.73     3.78     0.51     0.14     1.32    -0.75    -1.64
      0.80     2     4.61     8.39     0.50     0.09    -1.72    -0.86    -0.92
      0.80     3     2.28     5.65     0.64    -0.07    -0.96    -0.21    -5.10
      0.90     1     4.74     3.83     0.52     0.15     1.24    -0.85    -1.00
      0.90     2     4.60     8.34     0.51     0.10    -1.81    -0.91    -0.56
      0.90     3     2.32     5.59     0.75     1.14    -0.96     0.26     4.77
      1.00     1     4.76     3.88     0.54     0.16     1.15    -0.91    -0.60
      1.00     2     4.58     8.29     0.52     0.12    -1.90    -0.95    -0.34
      1.00     3     2.36     5.53     0.82     0.73    -0.92     0.55     2.90
      1.10     1     4.79     3.93     0.55     0.16     1.06    -0.94    -0.37
      1.10     2     4.56     8.24     0.54     0.13    -2.00    -0.97    -0.20
      1.10     3     2.42     5.46     0.89     0.64    -0.86     0.73     1.76
      1.20     1     4.82     3.98     0.68     1.23     1.00    -0.18     7.65
      1.20     2     4.54     8.19     0.61     0.76    -2.05    -0.19     7.75
      1.20     3     2.48     5.39     0.94     0.54    -0.78     0.84     1.07
      1.30     1     4.86     4.04     0.77     0.89     1.01     0.29     4.64
      1.30     2     4.51     8.14     0.67     0.55    -2.05     0.28     4.70
      1.30     3     2.55     5.33     0.99     0.47    -0.69     0.90     0.65
      1.40     1     4.90     4.11     0.85     0.83     1.05     0.57     2.81
      1.40     2     4.48     8.07     0.72     0.49    -2.01     0.56     2.85
```

Ilustración 22. Datos de una simulación en un punto txt

En la que podemos apreciar como los datos se disponen en columnas, donde la primera de ellas contiene el tiempo, la segunda el ID del robot, y a continuación los diversos parámetros de cada robot en cada instante de tiempo.

Con esta disposición, en la columna por ejemplo, de la velocidad, se van alternando los valores de velocidad de cada robot. Es por esto, que una vez importados los datos a MATLAB, con la función "importdata()" hay que tratar los datos para ordenarlos de manera que se pueda operar cómodamente con ellos.

La función "importdata()", devuelve una estructura de un array con los datos, y dos celdas con textos y cabecera. El array de datos será el que tratemos.

Antes de pasar al siguiente punto, es necesario puntuar que con la cabecera que incluye el simulador en el ".txt", la función "importdata()" no funciona bien, por lo que se ha entrado en el código del simulador y se han comentado las líneas que realizaban ésto.

Una vez se obtienen las matrices, se realizan las operaciones necesarias para calcular una serie de métricas que se consideran interesantes a la hora de comparar los distintos algoritmos.

Las métricas mencionadas son las siguientes:

- El número de colisiones.
- El número de fracasos (o UAVs que no alcanzan su destino).
- Distancia media recorrida normalizada por los UAVs, definida de la siguiente manera: Se calcula la distancia media recorrida por cada UAV, y la distancia media mínima posible, definida como la media de las distancias en línea recta entre los puntos iniciales y sus respectivos destinos. La distancia media normalizada es el cociente entre la distancia media recorrida y la distancia media mínima, siempre mayor o igual que uno.
- El tiempo normalizado, definido como el cociente entre el tiempo que dura la simulación, entre el mínimo tiempo de duración posible, calculado como el cociente de la mayor distancia entre punto inicial-final entre la velocidad máxima del robot, también siempre mayor o igual a uno.

El código de este programa se puede encontrar en el Anexo D.

3.1.3 Algoritmos

Los algoritmos codificados en MATLAB podemos encontrarlos en los Anexos X.

La estructura del programa CollisionAvoidance debe ser siempre la que se observa a continuación, en un algoritmo ejemplo que trae el simulador, sin ningún tipo de evitación de obstáculos:

```
function [v_ref, phi_ref,Status] = CollisionAvoidance(robot, other_robs,
comsys)
    % This is an example algorithm without any actual collision avoidance.
    % The robot will simply move to the goal and stop there.

    % v_ref: Velocity reference
    % phi_ref: Orientation reference
    % status: State of the state-machine
    % robot: Instance of RobotModel
    % other_robs: An array of all the other robots
    % comsys: Instance of the inter-robot communication system

    x_current = robot.x;
    y_current = robot.y;
    x_goal = robot.x_goal;
    y_goal = robot.y_goal;

    Status = robot.vars.ca_status;

    % The following are defined by the example algorithm itself
    STATE_MACHINE =
struct('TRAVELING2GOAL_FREE',0,'ARRIVING2GOAL',1,'ONGOAL',2);
    Darrival = 0.25;
    Dbreaking = 3 * Darrival;
    MaxVbreaking = 0.7;
    MinVbreaking = 0.0;
    % Note: With the values Darr=0.25 / Dbre=3*Darr / MaxVb=0 and MinVb=0
    % the system stops on the goal with an error no larger than 0.01m but
    % this is forcing an emergency stop.

    % State machine transitions controller
    % The state machine is composed by three states controled by the
    % distance to the goal
    %
    % |----TRAVELING2GOAL_FREE----+----ARRIVING2GOAL----+---ONGOAL----+
    %                         Dbreaking             Darrival        Goal

    d2goal = sqrt((y_goal - y_current)^2 + (x_goal - x_current)^2);

    if (d2goal > Dbreaking)
        Status = STATE_MACHINE.TRAVELING2GOAL_FREE;
    else if (d2goal > Darrival)
            Status = STATE_MACHINE.ARRIVING2GOAL;
        else
            Status = STATE_MACHINE.ONGOAL;
        end
    end
```

```
    % On all states the robot commands the orientation to the goal but the
    % speed reference will depend on the distance to the goal.
    %
    %               |
    %          Vmax +-------
    % MaxVbreaking  |       +
    %               |            \
    % MinVbreaking  |                +
    %             0 |                +-------+
    %               |-------+--------+-------+
    %                   Dbreaking Darrival  Goal

    % Computing the necessary reference to arrive to the goal
    phi_goal_ref = atan2((y_goal-y_current),(x_goal-x_current));

    switch Status

        case STATE_MACHINE.TRAVELING2GOAL_FREE
            v_goal_ref = 1;

        case STATE_MACHINE.ARRIVING2GOAL
            a =( MaxVbreaking - MinVbreaking ) / ( Dbreaking - Darrival);
            b =( MaxVbreaking - a*Dbreaking );

            v_goal_ref = a * d2goal + b;

        case STATE_MACHINE.ONGOAL
            v_goal_ref = 0;
    end
    v_ref = v_goal_ref;
    phi_ref = phi_goal_ref;

end % function
```

En la que el robot actúa en función del valor de STATE_MACHINE, dependiendo este valor de la distancia al objetivo, y de si hay o no obstáculos en el camino.

Según este programa, podemos disponer de sensores láser que nos dan la distancia al primer obstáculo en cada una de las 180 direcciones que posee, utilizando la función LaserScanner. El único problema es que esta función es relativamente lenta, y al realizar simulaciones con varios robots, el tiempo de simulación es significativamente alto, perdiéndose demasiado tiempo en las simulaciones. En el apartado de futuras líneas de trabajo, se recomeinda una posible optimización de esta función.

También permite si se desea comunicación entre los robots, dando así información sobre la posición de cada robot por comunicaciones (no tiene por qué ser la posición real exacta).

Debido a que para el algoritmo ORCA necesitamos conocer tanto posición como velocidad, y este programa no tiene implementado de por sí un sensor de velocidad (como sí laser). Se ha utilizado la posición y la velocidad de cada otro robot de la información que contiene el vector de estructuras OtherRobs.

3.2 Control de versiones

Durante la codificación de cada uno de los algoritmos, se ha utilizado un control de versiones. Un control de versiones es la herramienta indispensable para cualquier programador, en este caso se ha utilizado el controlador Mercurial con su interfaz para Mac TortoiseHg.

Con éste, se crea un repositorio con los archivos de código con los que estás trabajando que se versionan. Su principal aplicación para este trabajo ha sido la de versionar el archivo cada vez que se realiza un cambio en el mismo, de manera que el programa lo reconoce, y puedes guardarlo en una nueva versión, observando que ha cambiado, que hay nuevo, y qué ya no está, y permitiendo comentar cada cambio para poder saber que se ha hecho de un vistazo rápido.

Esta utilidad permite, cuando se van realizando cambios y algo deja de funcionar, o tenemos algún determinado comportamiento, tener un control total de los cambios que se han realizado, para ayudar a localizar la causa.

Además, con solo un click puedes recuperar otra versión anterior, o volver a la última, podiendo tener todos los archivos actualizados. De esta manera ha permitido programar sin volverse loco y perder mucho tiempo en localizar fallos. Otra utilidad es el "Shelve" o cajón, donde te permite almacenar el estado de todos los archivos en el momento que se quiera, para volver a utilizarlos en ese estado más adelante.

Tiene una mucho mayor variedad de utilidades, como una que no se ha utilizado en este trabajo, pero que consiste en la creación de un clon del repositorio como servidor, de manera que varias personas puedan trabajar en un mismo proyecto en un clon a nivel local, y posteriormente subirlos al servidor, con un sistema de gestión de conflictos (si estos se dan).

La interfaz gráfica es la siguiente:

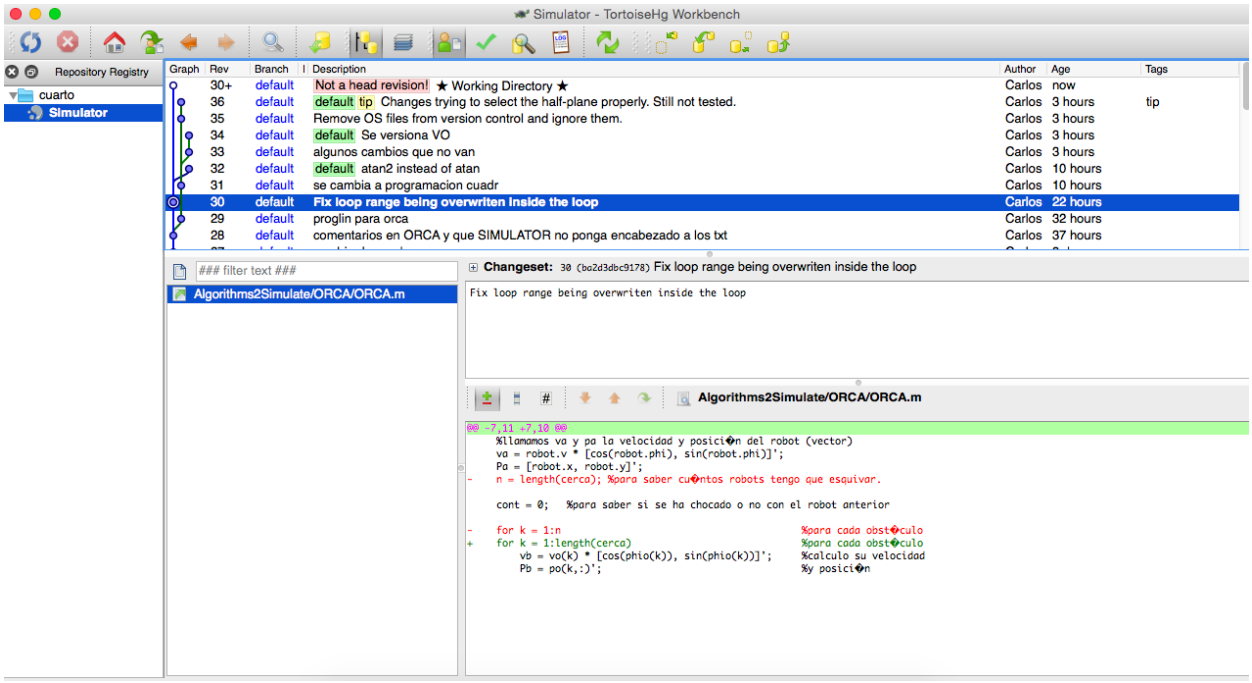

Ilustración 23. Interfaz del control de versiones TortoiseHg

Tras la experiencia de este trabajo, sin duda se recomienda la utilización de un control de versiones para cualquier proyecto de programación mínimamente complejo. Ha sido indispensable especialmente en la codificación del algoritmo ORCA.

4 SIMULACIONES

Es posible volar sin motores, pero no sin conocimiento y habilidad. Considero que es esto algo afortunado, para el hombre, por causa de su mayor intelecto, ya que es más razonable la esperanza de igualar a los pájaros en conocimiento, que igualar a la naturaleza en la perfección de su maquinaria

Wilbur Wright

En este capítulo se describen las simulaciones realizadas con cada algoritmo, así como los resultados obtenidos en las mismas.

Se van a realizar simulaciones de 5 escenarios distintos, como la aleatoriedad del simulador es muy pequeña, y se van a repetir el mismo escenario con cada algoritmo, con realizar 5 simulaciones de cada algoritmo en cada escenario se obtendrán resultados lo suficientemente significativos. En las tablas, se muestra la media de los resultados de estas 5 simulaciones.

Los escenarios a simular son abiertos (sin paredes en los extremos) y son los siguientes:

- Escenario 1:

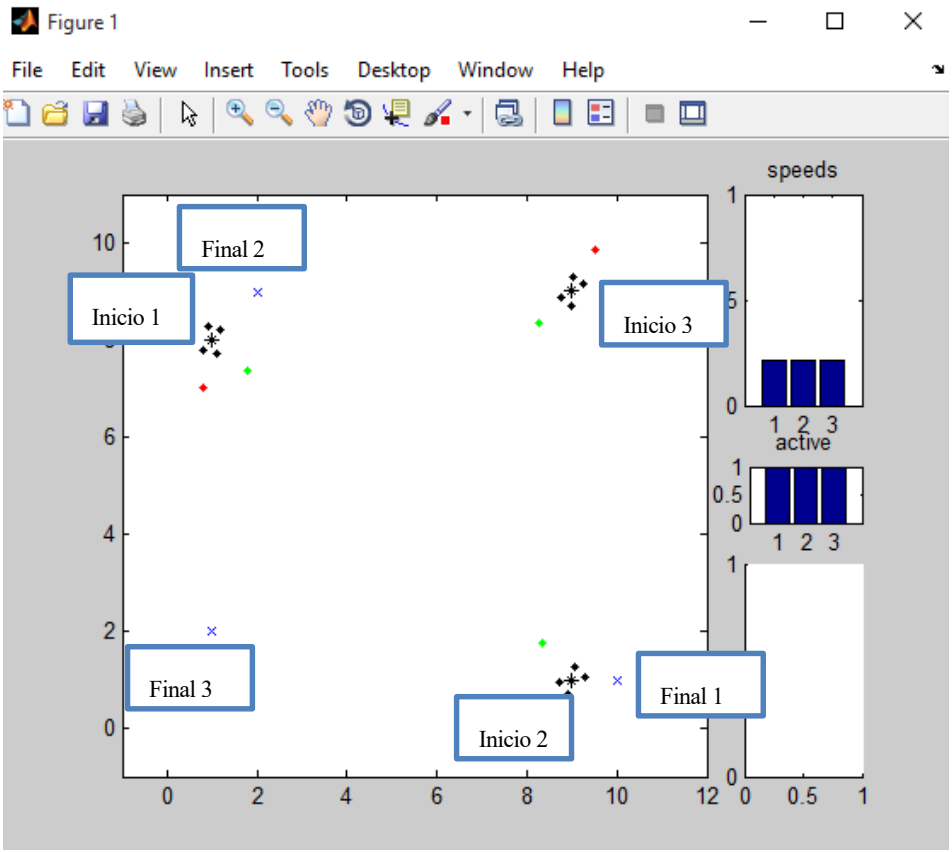

Ilustración 24. Primer escenario simulado

En este escenario, se observan las posiciones iniciales y finales para cada uno de los 3 robots que forman parte de él. Para llegar a su posición final, los tres tendrán que cruzarse en el centro.

- Escenario 2:

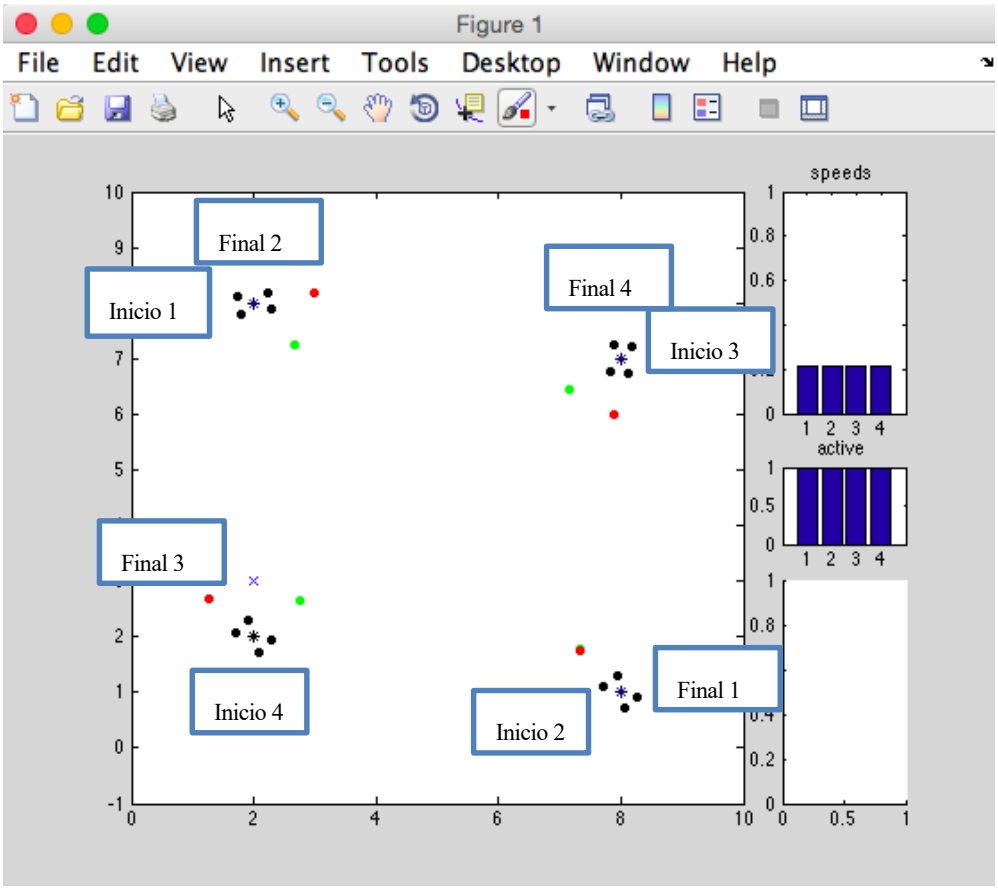

Ilustración 25. Segundo escenario simulado

En este caso, se tiene la misma situación, pero con cuatro robots. La posición inicial de cada uno es la final del que tiene enfrente, de manera que tendrán que cruzarse los cuatro por el centro del mapa.

- Escenario 3:

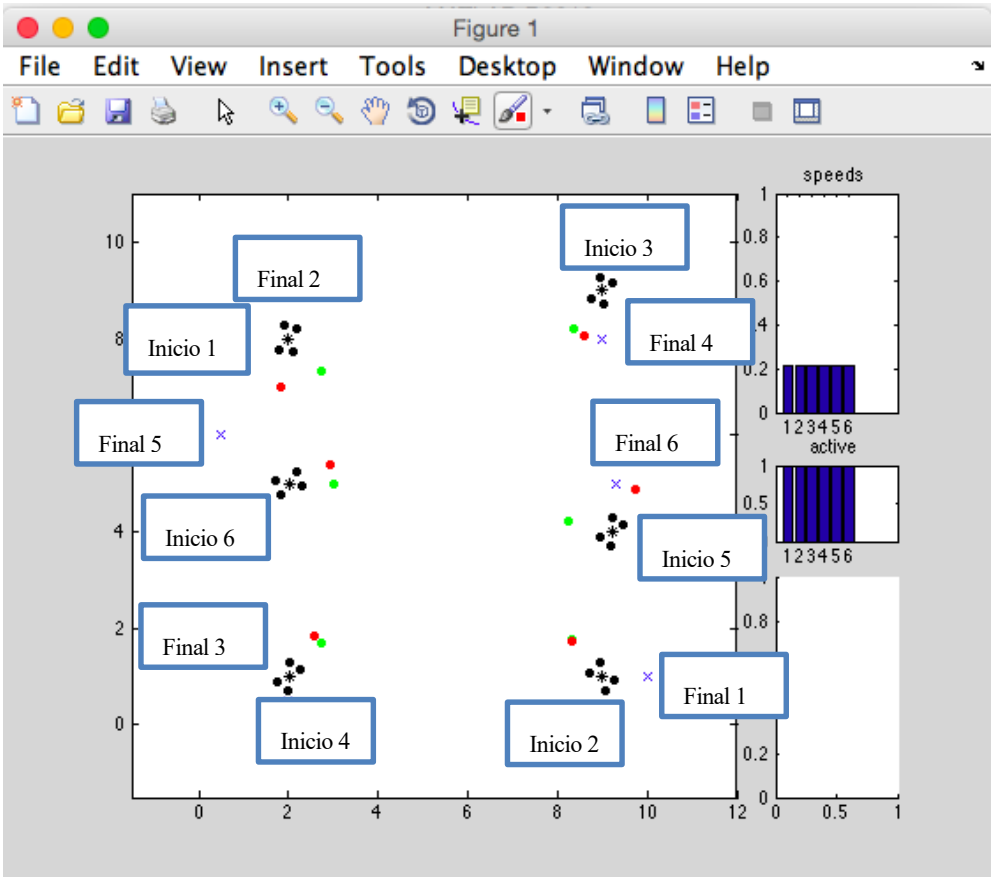

Ilustración 26. Tercer escenario simulado

En el escenario tres, se observa la misma situación con seis robots. A priori, es el escenario más complicado de todos.

- Escenario 4:

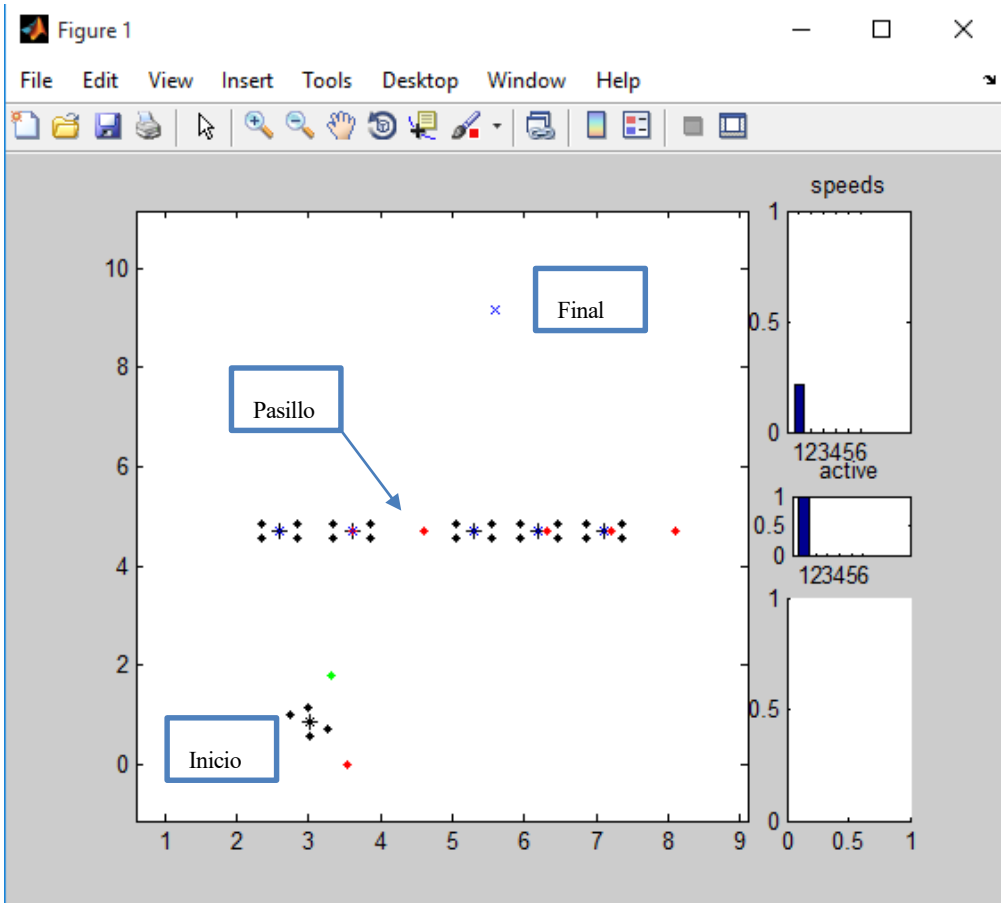

Ilustración 27. Cuarto escenario simulado

Con el Escenario 4, se pretende comparar el funcionamiento de los algoritmos ante obstáculos estáticos. Los obstáculos estáticos están representados con 5 robots cuya posición inicial y final es la misma. Además, entre la pared que forman los 5 robots estáticos, se deja un pequeño espacio o pasillo entre ellos, para comprobar el comportamiento de los algoritmos ante éste, si son o no capaces de detectarlo y cruzarlo.

- Escenario 5:

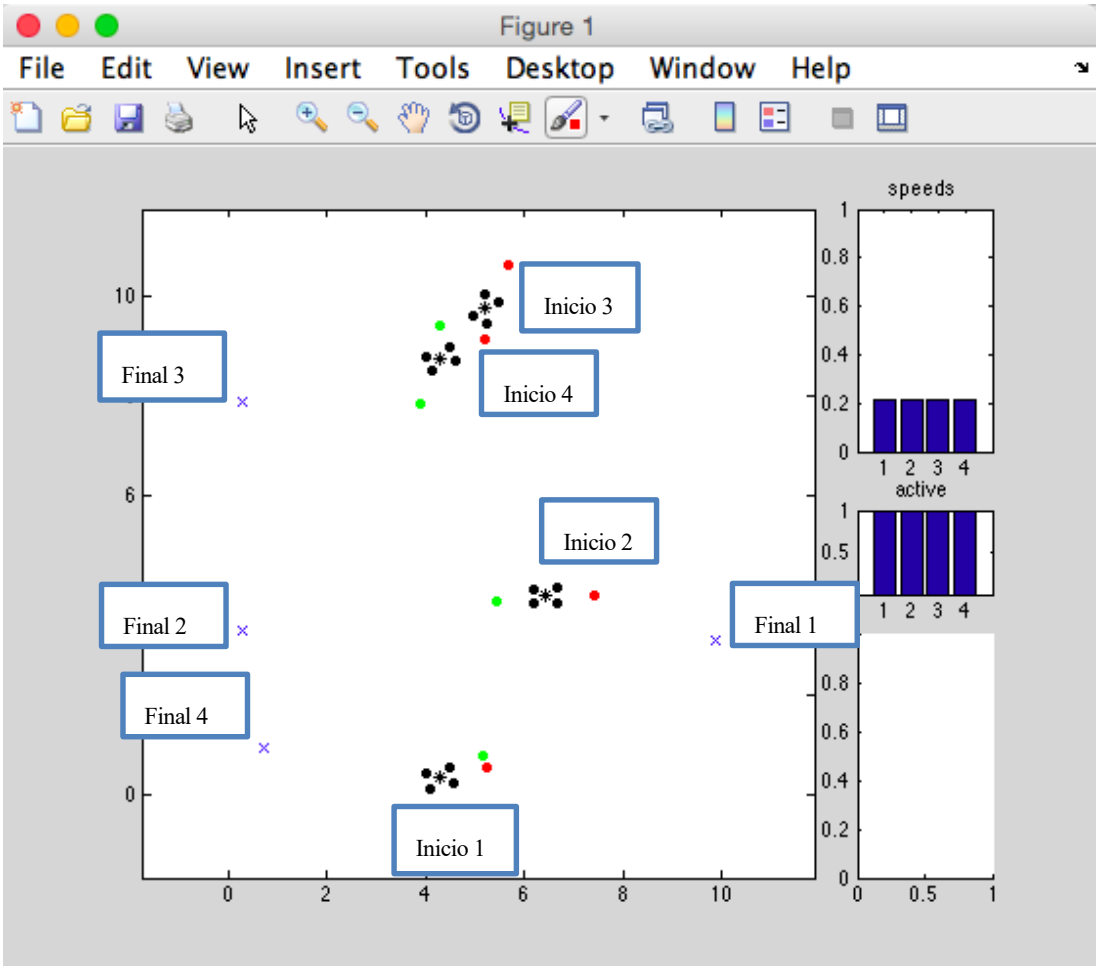

Ilustración 28 . Quinto escenario simulado

Tras simular los escenarios aparentemente más complicados, y un escenario con obstáculos estáticos, se genera un escenario aleatorio, en el que todos los robots se vean obligados a cruzarse, para estudiar el comportamiento de los algoritmos ante un escenario cualquiera.

Se recuerda, que el punto rojo indica la dirección en la que está apuntando el robot, por lo que el robot 1 y 2 se tendrán que cruzar.

En el siguiente apartado, se encuentran las métricas obtenidas en las simulaciones con cada algoritmo.

4.1 Bubble

Para esta simulación se ha tomado un valor de k de [0.2 : 3] y la velocidad mayor entre el 0.4 m/s y la menor distancia que mida el sensor dividida por 30.

Tabla 1. Métricas obtenidas con el algoritmo Bubble

Resultados	Colisiones	Tiempo normalizado	Distancia normalizada	Número de Fracasos
Escenario 1	0	1,3068	1,2227	0
Escenario 2	0	2.6466	1.5434	0
Escenario 3	0	8,4095	2,4776	0
Escenario 4	0	1,8520	1,4743	0
Escenario 5	0	5.9128	3.1555	2

Se puede observar como con las modificaciones que se le han realizado al algoritmo para que funcione con obstáculos móviles, el Bubble consigue hasta cierto punto adaptarse. En el primer escenario, el más sencillo, obtiene buenos resultados.

Conforme se complica el escenario, los resultados empeoran. Aun así, en los Escenarios 2 y 3 todos los robots consiguen llegar a su destino sin colisiones, pero con una distancia recorrida y tiempo empleado considerablemente mayores. El aumento de la distancia normalizada se debe a que este algoritmo, cuando tiene un obstáculo dentro de la burbuja, se olvida del objetivo y elige una dirección que únicamente evita la colisión. El mayor aumento del tiempo normalizado se debe a que la velocidad disminuye mucho cuando tiene obstáculos cerca para evitar la colisión, por tanto, cuanto más robots haya, más tiempo estarán cerca y más despacio van.

Ante objetos estáticos, para lo que está pensado, obtiene mejores resultados. Sin embargo, la ventana o pasillo que dejamos no la reconoció, dando toda la vuelta a los obstáculos. Era predecible de antemano que por la definición de este algoritmo no cruzaría este pasillo.

En el Escenario 5, los resultados obtenidos se explican porque los dos robots de arriba, llegan bien a su destino, sin embargo, los dos de abajo entran en una situación de conflicto, quedándose en bucle dando vueltas uno al lado del otro al objetivo final del robot que se encontraba al exterior. Se demuestra con esto que debido a la definición de este algoritmo, no se puede garantizar que siempre llegue a su destino, pues en situaciones aleatorias puede ocurrir esto.

4.2 VFF

En estas simulaciones, los valores de los coeficientes que multiplican a las fuerzas son de +5 para las fuerzas atractivas y -1 para las fuerzas repulsivas.

Tabla 2. Métricas obtenidas con el algoritmo VFF

Resultados	Colisiones	Tiempo normalizado	Distancia normalizada	Número de Fracasos
Escenario 1	0	1,8681	1,3014	0
Escenario 2	0	5.9656	4.5416	0
Escenario 3	0	1.7800	1.2449	0
Escenario 4	0	2,2086	1,7612	0
Escenario 5	0	1.4953	1.2322	0

A diferencia del Bubble, el VFF sí que tiene en todo momento en cuenta tanto el objetivo como los obstáculos. Es por ello que obtiene mejores resultados. Sin embargo, se observa como a pesar de llegar siempre a su destino sin colisiones, en algunos escenarios, sobre todo en el Escenario 2, no eligen el camino más corto.

En el escenario 4, ante objetos estáticos tampoco es capaz de cruzar por el hueco que dejan los robots. Ya se advertía en [9] que este algoritmo no funcionaba bien ante pasillos estrechos.

4.3 ORCA

Como este algoritmo supone robots esféricos, se han supuesto con un radio de 0.5m para evitar así las colisiones. El valor de τ es de 2.

Tabla 3. Métricas obtenidas con el algoritmo ORCA

Resultados	Colisiones	Tiempo normalizado	Distancia normalizada	Número de Fracasos
Escenario 1	0	1,3507	1,1131	0
Escenario 2	0	1,2690	1,0550	0
Escenario 3	0	1,2078	1,0664	0
Escenario 4	0	1,3803	1,1814	0
Escenario 5	0	1,5410	1,3757	0

El ORCA da en general muy buenos resultados en todos los escenarios, ya que elige la velocidad que no lleva a colisión más cercana a la velocidad que se dirige hacia el objetivo. Cabe destacar como en el Escenario 4, ante objetos estáticos, es capaz de detectar que entre la pared de robots hay un hueco por el que es capaz de pasar y lo cruza.

El resultado más alto en el Escenario 5 es debido a que los UAVs comienzan orientados en la dirección opuesta al objetivo, y debido a la dinámica del modelo de los robots, tarda más en dar la vuelta.

El tipo de robot es más clave en este algoritmo, ya que su actuación depende de la capacidad del robot para cambiar su velocidad. Ciertos robots como por ejemplo, los quadrotors, pueden parar en seco y cambiar su dirección, cosa que no podría hacer un UAV de ala fija.

Debido a los buenos resultados de este algoritmo, se prueba en un nuevo escenario, con 100 UAVs, para comprobar si con un gran número de robots sigue obteniendo buenos resultados. El escenario que se simula es el que se observa en la Ilustración 29, en la que el destino del primer UAV de cada columna, es la primera cruz azul, la segunda del segundo, y así sucesivamente. Cada robot de la columna tendrá que esquivar a los que ya han llegado y seguir hacia su destino.

Se obtiene como resultado los siguientes valores:

- Colisiones: 0
- Tiempo normalizado: 1.5575
- Distancia normalizada: 1.5353

Demostrando así el ORCA su potencial.

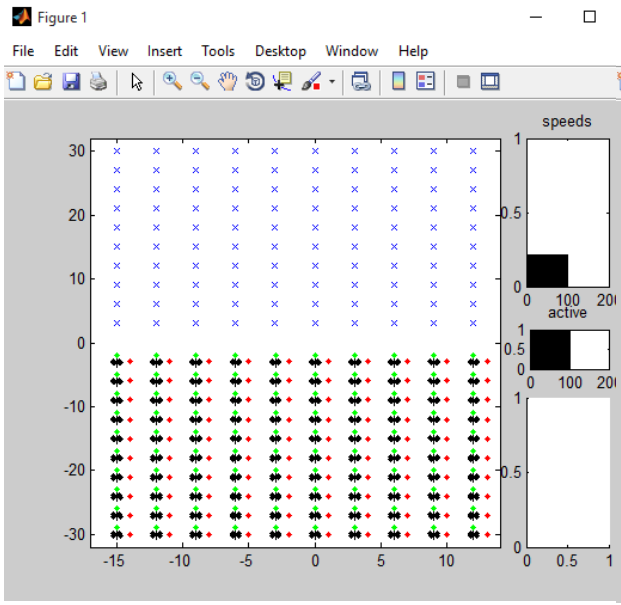

Ilustración 30. Escenario con 100 UAVs

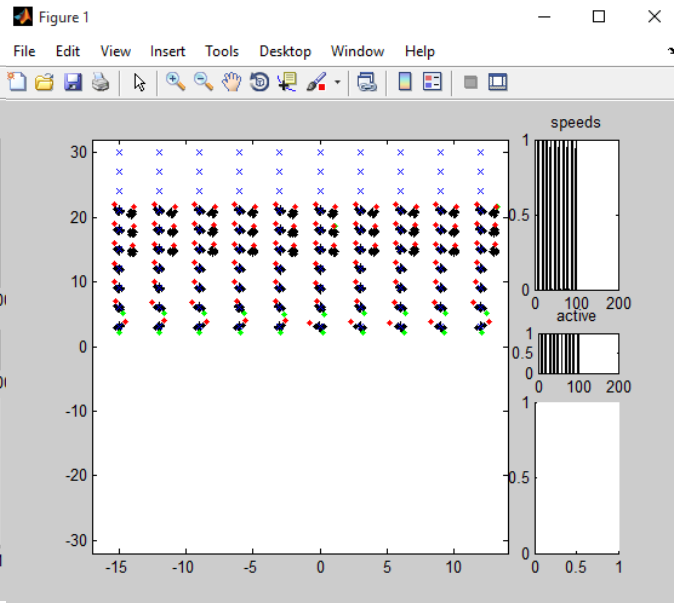

Ilustración 29. Movimiento de los 100 UAVs

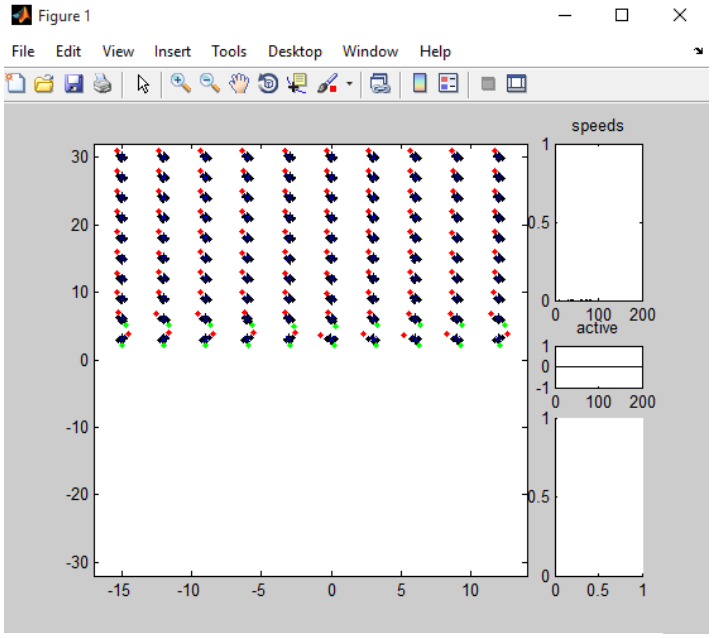

Ilustración 31. Posición final de los 100 UAVs

5 COMPARACIONES

Nadie puede imaginar que tan sustancial es el aire, hasta sentir su poder de sustentación debajo de uno. Inspira confianza

Otto Lilienthal

En este capítulo vamos a comparar los resultados de las simulaciones realizadas a los distintos algoritmos que nos permita sacar posteriormente conclusiones de las mismas.

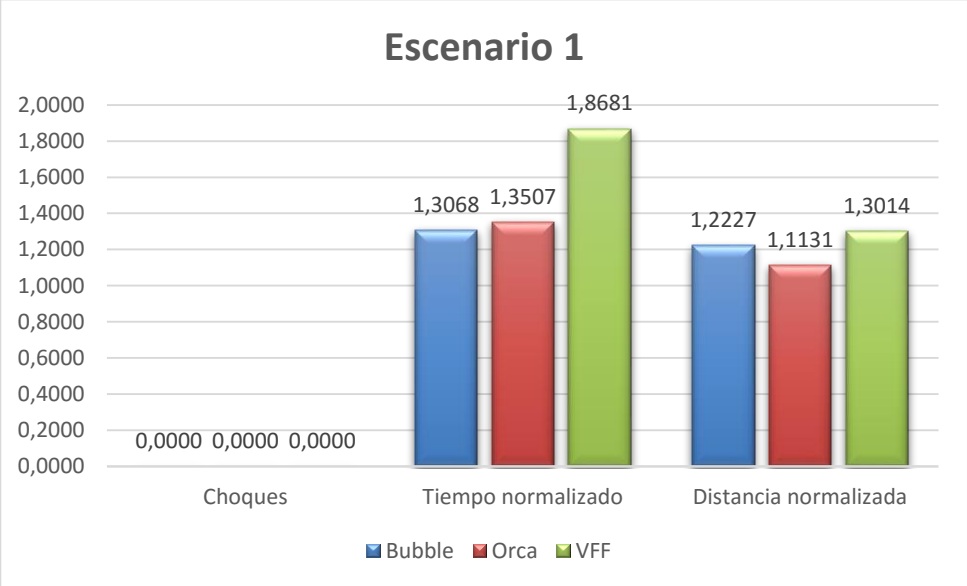

Ilustración 32. Comparación de las métricas obtenidas de los distintos algoritmos en el primer escenario

En el primer escenario, los tres algoritmos obtienen buenos resultados.

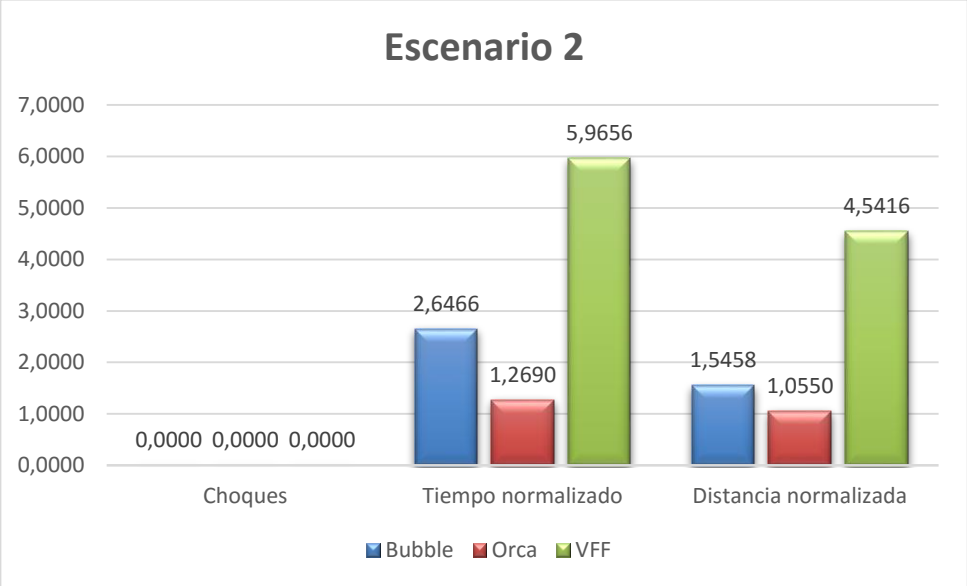

Ilustración 33. Comparación de las métricas obtenidas de los distintos algoritmos en el segundo escenario

En el Escenario 2, el cual enfrenta a 4 robots de esquina a esquina, se observa claramente la mejor actuación del ORCA.

Cabe destacar que en este caso el Bubble, obtiene mejores resultados que el VFF. Sin embargo esto se debe únicamente a la casuística de la distribución del escenario, sin ser concluyente debido a lo que se deduce de las demás simulaciones.

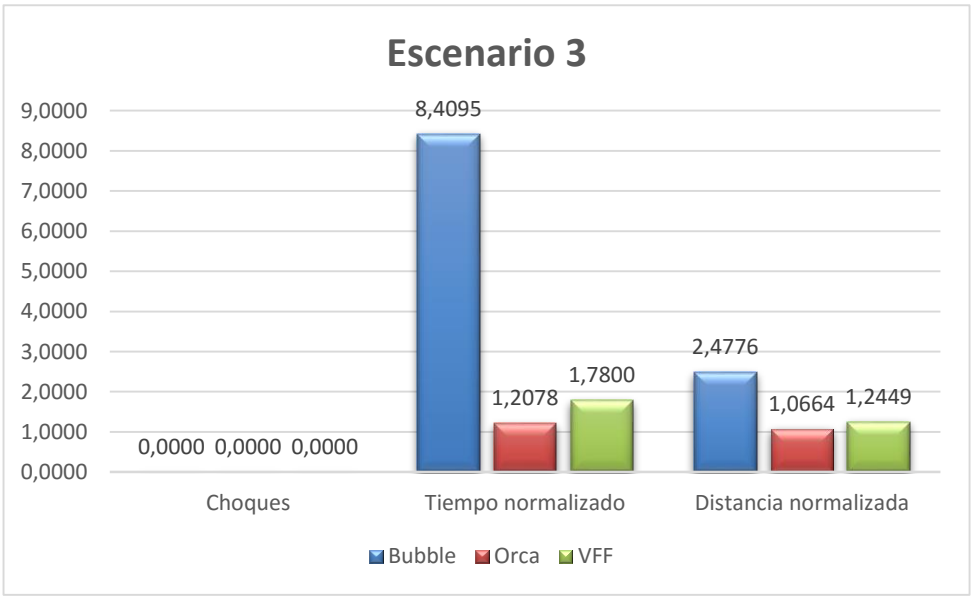

Ilustración 34. Comparación de las métricas obtenidas de los distintos algoritmos en el tercer escenario

En el Escenario 3, el más complejo, todos los algoritmos han logrado llevar a todos los UAVs a sus destinos sin colisiones. El Bubble, como es de esperar, tiene peores resultados. El hecho de que la diferencia entre la distancia normalizada y el tiempo normalizado sea tan grande es debido a que en este algoritmo hemos reducido la velocidad cuando hay obstáculos cerca, y al haber muchos robots cerca durante mucho tiempo, tardan mucho más en llegar al destino, sin embargo esto es necesario para que no se haya colisiones.

Tanto ORCA como VFF presentan muy buenos resultados en este escenario, siendo mejores de nuevo los del ORCA.

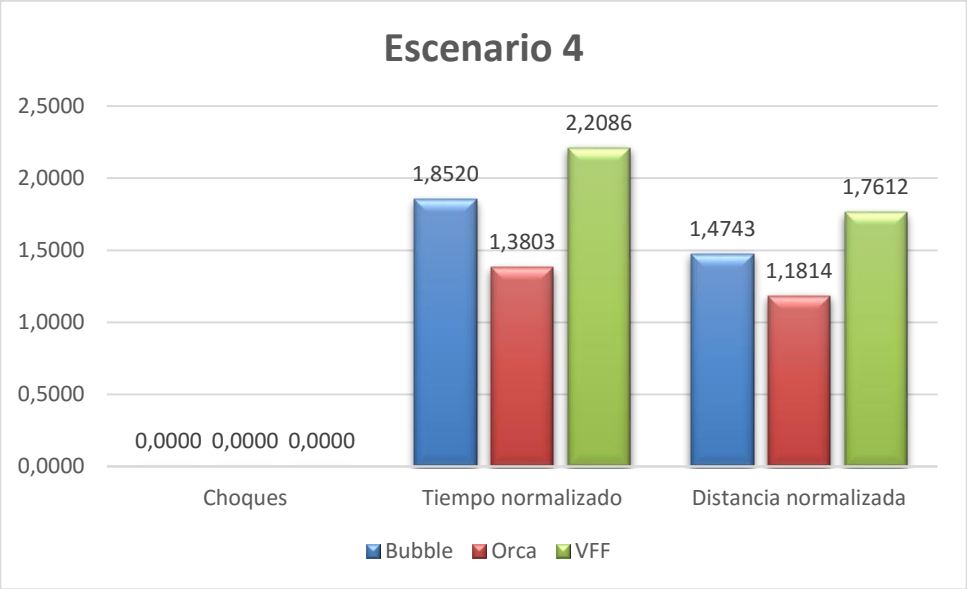

Ilustración 35. Comparación de las métricas obtenidas de los distintos algoritmos en el cuarto escenario

En el Escenario 4, se pretendía probar dos cosas, la primera, el comportamiento de los algoritmos ante obstáculos estáticos, y la segunda, comprobar si ante una apertura en una pared formada por varios UAVs estáticos, son capaces de detectarla y cruzarla o no.

Únicamente el ORCA ha conseguido tomar el camino más corto. Tanto Bubble como VFF han rodeado el muro. El Bubble ha llegado antes y recorriendo menos distancia. Sin embargo, bordeó la pared por el camino más largo. Esto es debido a que el VFF, a pesar de rodear la pared por el camino más corto, se pegó a esta antes de rodearla.

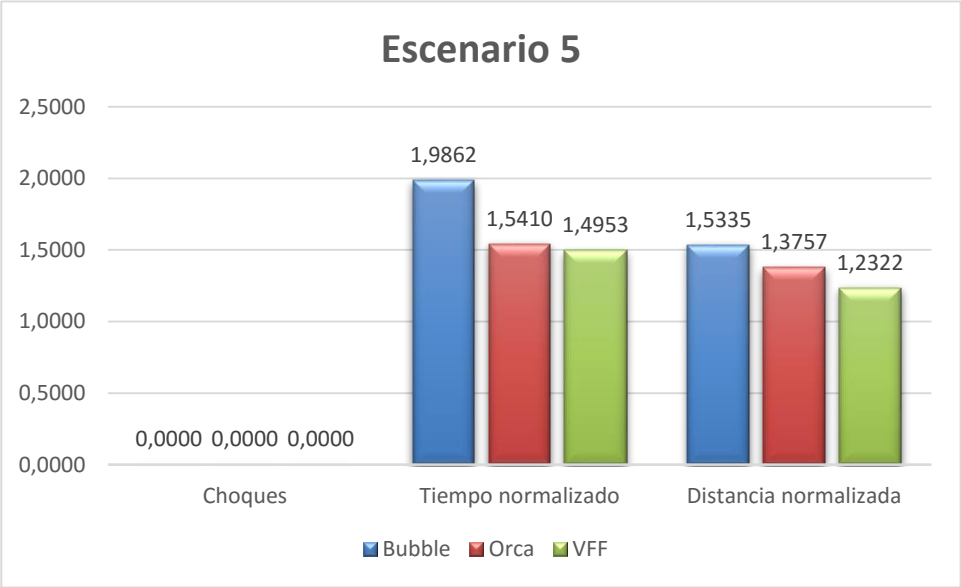

Ilustración 36. Comparación de las métricas obtenidas de los distintos algoritmos en el quinto escenario

En un escenario aleatorio como el Escenario 5, de nuevo se ve un resultado ajustado entre ORCA y VFF, mientras que el Bubble, no solo tiene peores resultados, si no que fracasa en la llegada al destino. Dos UAVs o robots se quedan dando vueltas en bucle a un objetivo sin llegar a él. Sin embargo, no se produce ninguna colisión.

6 CONCLUSIONES

Si Dios hubiera querido que voláramos hubiéramos nacido con alas

Milton Wright

A Raíz de lo que se ha estudiado, se pueden sacar ciertas conclusiones a cerca de estos algoritmos. Hay que destacar que los tres han demostrado un funcionamiento correcto, sin ninguna colisión en la simulación de todos los escenarios que se han definido en este trabajo.

El Bubble, con objetos estáticos o pocos robots, da buenos resultados. Es un algoritmo más simple, de manera que en función de la misión a realizar, si ésta no tiene muchos robots, éstos son simples con una pequeña computadora, y no se tienen grandes restricciones, este algoritmo puede llegar a ser el más apropiado.

Tanto el ORCA como el VFF han mostrado muy buenos resultados, pero el ORCA ha demostrado ser el que encuentra siempre el camino más corto y más rápido sin colisionar por el camino. Esto era lo que esperábamos antes de realizar este trabajo, ya que es un algoritmo conceptualmente mucho más desarrollado.

Como hemos dicho anteriormente con el Bubbe, la elección de uno u otro dependerá de la aplicación para la que se vaya a utilizar, de las exigencias que tenga ésta, los medios de los que disponga y el tipo de robot. Para misiones más críticas, sería recomendable implementar el ORCA, sin embargo si cumple las necesidades requeridas puede ser suficiente y más apropiado al ser computacionalmente más sencillo.

También cabe destacar, tras la realización de este trabajo, que los parámetros que definen el comportamiento de los algoritmos, es clave para el comportamiento del mismo, por lo que un análisis y optimización de éstos es clave.

7 LÍNEAS DE TRABAJO FUTURAS

El hombre mira el cielo pero no cree que en algún día estará ahí.

Leonardo da Vinci

Las líneas de trabajo futuras que se proponen a partir de la realización de este Trabajo de Fin de Grado, pueden englobarse en trabajo sobre el simulador, o sobre los algoritmos estudiados.

Por un lado, en lo que concierne al simulador, se propone la posibilidad de ampliar la forma en la que se generan los escenarios, desde la creación de escenarios más grandes con un mayor número de robots, hasta de crear escenarios aleatorios desde el Menu y guardarlos cuando sean interesantes.

Al igual que la posibilidad de escoger distintos modelos de robots para las simulaciones con un comportamiento dinámico distinto, o incluso poder también crearlos en el Menu.

Por otro lado, se propone mejorar el funcionamiento de la versión LaserScanner. Una manera que se ha pensado para esto, sin alterar mucho el funcionamiento de la función es la siguiente:

Esta función lee continuamente en un bucle el valor en cada ángulo, teniendo uno en cada dirección para una mayor precisión. Al hacer esto, una vez en cada ángulo con cada robot, toma una gran cantidad de tiempo.

Como a priori conocemos dónde están los obstáculos, y conocemos su forma (dentro de Type) y sus esquinas, se propone definir internamente unos sectores, donde se encuentran los obstáculos, y repasar únicamente los ángulos de este sector para encontrar la intersección más cercana de éste con el rayo láser.

De cara a la comunicación entre robots, que no ha sido utilizada en este trabajo, se propone que incluyese también información de la velocidad u otros estados del robot, de manera que ampliara el abanico de posibilidades. Así como la implementación en otra función, similar a la del láser, que simule a un sensor de velocidad.

En cuanto a los algoritmos, sería interesante implementar para el cálculo de la velocidad óptima en ORCA, el algoritmo Polygonal Sweep Algorithm que proponen en [10].

Se propone también un análisis a fondo sobre los parámetros que definen en el comportamiento de cada algoritmo, cosa que no era el objeto de este trabajo, pero que basándose en él podrían conseguirse resultados y comparaciones más precisas.

REFERENCIAS

[1] Aníbal Ollero, apuntes de clase asignatura UAV 2015.

[2] Tecnología El País, 2 DIC 2013.
[3] La Vanguardia, Jueves, 17 de septiembre 2015.

[4] ABC Tecnología, 26 de enero de 2015.

[5] El correo gallego, Viernes 07 de Noviembre de 2014.

[6] Jur van den Berg. Reciprocal Velocity Obstacle. 2008 IEEE International Conference on Robotics and Automation. Pasadena,CA,USA,May 19-23,2008.

[7] James Ng · Thomas Bräunl. Performance Comparison of Bug Navigation Algorithms. J Intell Robot Syst (2007) 50:73–84

[8] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha, Reciprocal n-Body Collision Avoidance Robotics Research, STAR 70, pp. 3–19.

[9] I. Susnea, A. Filipescu, G. Vasiliu, Member, IEEE, G. Coman, A. Radaschin. The Bubble Rebound Obstacle Avoidance Algorithm for Mobile Robots. 2010 8th IEEE International Conference on Control and Automation. Xiamen, China, June 9-11,2010.

[10] Jayasri K . Janardanan. Decentralized Collision Avoidance. Computer Science and Engineering: Theses, Dissertations, and Student Research 8-1-2013.