PROGRAMA PRINCIPAL

```
clc
clear
close all



%_______________________________________________________________________________
%---------------------------- MAP CREATION ----------------------------------
map_opt = input('Do you wish to crete a new map? [y/n] : ','s') ;
if strcmp(map_opt,'y')
    dim_x = input('X dimension of the map : ');
    dim_y = input('Y dimension of the map : ');
    map_name = input('name of the map (no blank spaces): ','s') ;
    [map_parameters] = create_map(dim_x,dim_y,map_name);
  elseif strcmp(map_opt,'n')
    filename = input('enter the name of the map to be used : ','s');
    load(filename)
end
%_______________________________________________________________________________
%-----------------------------------------------------------------------------



%_______________________________________________________________________________
%------------------------ SIMULATION PARAMETERS ---------------------------
sim_parameters.ant_pop          = 15;
sim_parameters.phmG_red         = 0.3;
sim_parameters.phmG_gain        = 50 ;
sim_parameters.phmG_exp         = 1 ;
sim_parameters.answer_apriori = 6;
sim_parameters.phm_limit        = 150; %
sim_parameters.iterations       = 65;
sim_parameters.compass_EN       = 'all'; % all , none
sim_parameters.compass_gain     = 10 ;
sim_parameters.keepdir_EN       = 'none'; % all , none
sim_parameters.keepdir_gain     = 1 ;
sim_parameters.show_sim         = 1 ;
%_______________________________________________________________________________
%-----------------------------------------------------------------------------


% delta TAU variation
Snf =
(sim_parameters.answer_apriori+1):1:(sim_parameters.answer_apriori+9);
delta_tau = sim_parameters.phmG_gain./((Snf-
sim_parameters.answer_apriori).^sim_parameters.phmG_exp) ;
figure;
    plot(Snf,delta_tau,'--o')
    grid on
    xlabel('steps from nest to food')
    ylabel('delta tau')


tic
[ACO_output] = ACO(sim_parameters,map_parameters);
te_ACO = toc ;
display(strcat('time elapsed :',32,num2str(te_ACO),32,'seconds,',32,'ACO
algorithm'));
```

```
[result_rc,result_xy,steps] =
find_result(ACO_output,sim_parameters,map_parameters); % [rows,cols]


V = 1 ;      dt = 0.001;      ti = 0 ;

[tf,trajectory] = fill_points(V,dt,result_xy);


xp1d = trajectory(:,1)' ;       xp2d = trajectory(:,2)' ;

zd   = [ xp1d ;
         xp2d ];
%__________________________________________________________________________
%--------------------------------------------------------------------------


t = ti:dt:tf ;



%__________________________________________________________________________
%-------------------------- SYSTEM PARAMETERS -----------------------------
m11 = 200 ;          m22 = 250 ;      Izz = 700 ;
d11 = 70 ;           d22 = 100 ;      d66 = 50 ;
d   = 1 ;
%__________________________________________________________________________
%--------------------------------------------------------------------------



%__________________________________________________________________________
%-------------------------- INITIAL CONDITIONS ----------------------------
x1    = zeros(length(t),1);             x2    = zeros(length(t),1);
xp1   = zeros(length(t),1);             xp2   = zeros(length(t),1);
u     = zeros(length(t),1);             v     = zeros(length(t),1);
psi   = zeros(length(t),1);             r     = zeros(length(t),1);
F     = zeros(length(t),1);             T     = zeros(length(t),1);

food_XY = [ map_parameters.food(2) size(map_parameters.map,1)+1-
map_parameters.food(1) ] ;
nest_XY = [ map_parameters.nest(2) size(map_parameters.map,1)+1-
map_parameters.nest(1) ] ;

xp1(1)   = nest_XY(1);                  xp2(1)   = nest_XY(2);

u(1)     = 1 ;                          v(1)     = 0 ;
psi(1)   = 0 ;                          r(1)     = 0 ;

xp1p = u(1)*cos(psi(1)) - v(1)*sin(psi(1)) ; % initial change in "xp1"
xp2p = u(1)*sin(psi(1)) + v(1)*cos(psi(1)) ; % initial change in "xp2"
%__________________________________________________________________________
%--------------------------------------------------------------------------
```

```
%___________________________________________________________________________
%---------------------- CONTROLLER PARAMETERS -----------------------------
A = [ 0 0 1 0 ;
      0 0 0 1 ;
      0 0 0 0 ;
      0 0 0 0 ];

B = [ 0 0 ;
      0 0 ;
      1 0 ;
      0 1 ];

q1 = 0.1;
q2 = 0.1;
q3 = 10;
q4 = 10;
Q = diag([ q1 q2 q3 q4 ]);      P = are(A,B*B',Q);

K = B'*P;       Kz = K(1:2,1:2);        Kzp = K(1:2,3:4);
%___________________________________________________________________________
%---------------------------------------------------------------------------



%___________________________________________________________________________
%----------------------------- TIME LOOP ----------------------------------
tic
k = 1;
for tt = ti:dt:(tf-dt)

    z = [  xp1(k)   ;
           xp2(k)   ];

    zp = [  xp1p   ;
            xp2p   ];


    %_____________________________CONTROLLER_____________________________
    f = [  (m22*v(k)*r(k)-d11*u(k))*cos(psi(k))/m11 - u(k)*r(k)*sin(psi(k))
+ (m11*u(k)*r(k)+d22*v(k))*sin(psi(k))/m22 - d*sin(psi(k))/Izz*((m11-
m22)*u(k)*v(k)-d66*r(k)) - (v(k)+r(k)*d)*r(k)*cos(psi(k)) ;
           (m22*v(k)*r(k)-d11*u(k))*sin(psi(k))/m11 + u(k)*r(k)*sin(psi(k))
- (m11*u(k)*r(k)+d22*v(k))*cos(psi(k))/m22 + d*cos(psi(k))/Izz*((m11-
m22)*u(k)*v(k)-d66*r(k)) - (v(k)+r(k)*d)*r(k)*sin(psi(k)) ];


    b = [  cos(psi(k))/m11      -d*sin(psi(k))/Izz   ;
           sin(psi(k))/m11       d*cos(psi(k))/Izz   ];

    uc = (b^-1)*(-Kz*(z-zd(:,k))-Kzp*zp - f);

    F(k) = uc(1) ;       T(k) = uc(2) ;
    %_______________________________________________________________________



    %______________FOSSEN MODEL, 3DOF, FAHIMI NOTATION_____________________
    up = (F(k) + m22*v(k)*r(k) - d11*u(k))/m11; % change in "u"
```

```
    u(k+1) = u(k) + dt*up ;

    vp = (-m11*u(k)*r(k) - d22*v(k))/m22 ; % change in "v"
    v(k+1) = v(k) + dt*vp ;

    rp = (T(k) - (m22-m11)*u(k)*v(k) - d66*r(k))/Izz ; % change in "r"
    r(k+1) = r(k) + dt*rp ;
    psi(k+1) = psi(k) + dt*r(k); % change in "psi"
    %_______________________________________________________________________



    %________________________CONTROL POINT_______________________________
    xp1p = u(k)*cos(psi(k)) - (v(k)+r(k)*d)*sin(psi(k)) ;
    xp1(k+1) = xp1(k) + dt*xp1p ;

    xp2p = u(k)*sin(psi(k)) + (v(k)+r(k)*d)*cos(psi(k)) ;
    xp2(k+1) = xp2(k) + dt*xp2p ;
    %_______________________________________________________________________



    %________________________CENTER OF GRAVITY_____________________________
    x1p = u(k)*cos(psi(k)) - v(k)*sin(psi(k)) ; % change in "x1"
    x1(k+1) = x1(k) + dt*x1p ;

    x2p = u(k)*sin(psi(k)) + v(k)*cos(psi(k)) ; % change in "x2"
    x2(k+1) = x2(k) + dt*x2p ;
    %_______________________________________________________________________



    k = k + 1 ;


    %_________________________STOP CONDITION______________________________
    if
(abs(xp1(k))>1.2*size(map_parameters.map,2))||(abs(xp2(k))>1.2*size(map_par
ameters.map,1))
        break
    end
    %_______________________________________________________________________



end
cost_func = sum(sqrt((xp1-xp1d').^2 + (xp2-xp2d').^2)) ;
te = toc;
display(strcat('time elapsed :',32,num2str(te),32,'seconds, controller'))
%___________________________________________________________________________
%---------------------------------------------------------------------------



%___________________________________________________________________________
%----------------------------- RESULTS ------------------------------------
tic
```

```
lineW = 3 ;

figure;
subplot(2,3,1);
    h = plot(t(1,1:k-1),xp1(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('x_{p} [m]','fontsize',12)
    set(h,'linewidth',lineW);
subplot(2,3,4);
    h = plot(t(1,1:k-1),xp2(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('y_{p} [m]','fontsize',12)
    set(h,'linewidth',lineW);
subplot(2,3,2);
    h = plot(t(1,1:k-1),u(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('u [m/s]','fontsize',12)
    set(h,'linewidth',lineW);
subplot(2,3,5);
    h = plot(t(1,1:k-1),v(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('v [m/s]','fontsize',12)
    set(h,'linewidth',lineW);
subplot(2,3,3);
    h = plot(t(1,1:k-1),psi(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('psi [rad]','fontsize',12)
    set(h,'linewidth',lineW);

subplot(2,3,6);
    h = plot(t(1,1:k-1),r(1:k-1,1));
    grid on
    xlabel('tiempo [seg]')
    ylabel('r [rad/s]','fontsize',12)
    set(h,'linewidth',lineW);

figure;
subplot(2,1,1);
    h = plot(t(1,1:k-1),F(1:k-1,1));
    grid on
    ylabel('FUERZA [N]','fontsize',12)
    xlabel('tiempo [seg]')
    set(h,'linewidth',lineW);
subplot(2,1,2);
    h = plot(t(1,1:k-1),T(1:k-1,1));
    grid on
    ylabel('TORQUE [Nm]','fontsize',12)
    xlabel('tiempo [seg]')
    set(h,'linewidth',lineW);


CloneFig(2,6); hold on;

h = plot(xp1,xp2,'c');
```

```
    xlabel('coordenada X [m]','fontsize',12)
    ylabel('coordenada Y [m]','fontsize',12)
    grid on
    set(h,'linewidth',lineW+1);
    axis([0.5 size(map_parameters.map,2)+0.5 0.5
size(map_parameters.map,1)+0.5])
 draw_boat_03(xp1,xp2,psi,8,0); hold on;

te = toc;
display(strcat('time elapsed :',32,num2str(te),32,'seconds, ilustrations'))
%_____________________________________________________________________________
%-----------------------------------------------------------------------------
```

SUB-RUTINA: CREAR MAPA

```
function [map_parameters] = create_map(dim_x,dim_y,map_name)

    pts_x = 1:dim_x ;           pts_y = 1:dim_y ;

    map = ones(length(pts_y),length(pts_x));

    % borders of the map are zeros, those are the frontier
    map(1,1:size(map,2)) = zeros(1,size(map,2)) ;           % upper border
    map(size(map,1),1:size(map,2)) = zeros(1,size(map,2)) ; % bottom botder
    map(1:size(map,1),1) = zeros(size(map,1),1);            % left border
    map(1:size(map,1),size(map,2)) = zeros(size(map,1),1);  % rigth border

    figure; hold on;

    axis([ 0.5 dim_x+0.5 0.5 dim_y+0.5 ]);
    h1 = gca ;
    h1.XTick = 0.5:1:dim_x+0.5 ;      h1.YTick = 0.5:1:dim_y+0.5 ;
grid on

    again = 1 ;

    while again == 1

        title('click on TWO points in the map to create an obstacle line')
        [x,y] = ginput(2);

        axis([ 0.5 dim_x+0.5 0.5 dim_y+0.5 ]);
        h1 = gca ;
        h1.XTick = 0.5:1:dim_x+0.5 ;      h1.YTick = 0.5:1:dim_y+0.5 ;
grid on

        xy = round([x,y]);      x = xy(:,1) ;         y = xy(:,2);

        if (x(1) ~= x(2)) && (y(1) ~= y(2))
            xv = min(x):0.01:max(x);
            yv = ( (y(2)-y(1))/(x(2)-x(1)) )*xv + ( (x(1)*y(2)-
x(2)*y(1))/(x(1)-x(2)) ) ;
          elseif (x(1) == x(2)) && (y(1) ~= y(2))
            yv =  min(y):1:max(y);
            xv = x(1) * ones(1,length(yv));
          elseif (x(1) ~= x(2)) && (y(1) == y(2))
            xv =  min(x):1:max(x);
            yv = y(1) * ones(1,length(xv));
          elseif (x(1) == x(2)) && (y(1) == y(2))
            yv = y(1);
            xv = x(1);
        end

        xv = round(xv) ;         yv = round(yv) ;

        cols = xv ;              rows = -yv+dim_y+1 ;

        wall_pts = length(rows);

        for i = 1:wall_pts
```

```
            fill([xv(i)-0.5 xv(i)-0.5 xv(i)+0.5 xv(i)+0.5],[yv(i)-0.5
yv(i)+0.5 yv(i)+0.5 yv(i)-0.5],'k')
            map(rows(i),cols(i)) = 0 ;
        end

        title('Do you wish to add more obstacles? [Y/N] : ')
        again = input('Do you wish to add more obstacles? [Y/N] : ','s');
        if (again == 'Y') || (again == 'y')
            again = 1 ;
          elseif (again == 'N') || (again == 'n')
            again = 0 ;
          else
            again = -1 ;
        end

    end

    title('chose the NEST point')
    [x,y] = ginput(1);
        axis([ 0.5 dim_x+0.5 0.5 dim_y+0.5 ]);
        h1 = gca ;
        h1.XTick = 0.5:1:dim_x+0.5 ;      h1.YTick = 0.5:1:dim_y+0.5 ;
grid on

    xy = round([x,y]);      x = xy(:,1) ;        y = xy(:,2);
    fill([x-0.5 x-0.5 x+0.5 x+0.5],[y-0.5 y+0.5 y+0.5 y-0.5],'b')
    nest = [ -xy(:,2)+dim_y+1 , xy(:,1) ] ;

    title('chose the FOOD point')
    [x,y] = ginput(1);
        axis([ 0.5 dim_x+0.5 0.5 dim_y+0.5 ]);
        h1 = gca ;
        h1.XTick = 0.5:1:dim_x+0.5 ;      h1.YTick = 0.5:1:dim_y+0.5 ;
grid on

    xy = round([x,y]);      x = xy(:,1) ;        y = xy(:,2);
    fill([x-0.5 x-0.5 x+0.5 x+0.5],[y-0.5 y+0.5 y+0.5 y-0.5],'r')
    food = [ -xy(:,2)+dim_y+1 , xy(:,1) ] ;

    if again ==0
        display('thanks')
       elseif (again == -1)
        display('ERROR, invalid input')
    end

    title(map_name)

    food_XY = [ food(2) size(map,1)+1-food(1) ] ;
    nest_XY = [ nest(2) size(map,1)+1-nest(1) ] ;

    map_parameters.map      = map ;
    map_parameters.nest     = nest ;
    map_parameters.food     = food ;
    map_parameters.nest_XY = nest_XY ;
    map_parameters.food_XY = food_XY ;
    map_parameters.name     = map_name ;

    filename = strcat(map_name,'.mat');
    save(filename,'map_parameters')
```

```
end

% 2015 12 10 - 13:32pm
% this program lets us make ou own maps using the mouse
% 17:12pm
% now we have change the obstacle creation, now we only need two points to
% draw a straight line , this is efficient for fine-grained maps
%
% 2016 01 08 - 11:39 am
% it could be more interactive if the user can read what to do in the
% title, lets change that
```

SUB-RUTINA: ACO

```
function [ACO_output] = ACO(sim_parameters,map_parameters)

tic

%______________________________________________________________________________
%--------------------------- GLOBAL VARIABLES ----------------------------
global map;          global map_r;        global map_c
global food_XY;      global nest_XY;      global food;        global nest;
global phmR_map;     global phmG_map;     global phmB_map;
global phm_limit;
global r_step
%______________________________________________________________________________
%------------------------------------------------------------------------------



map          = map_parameters.map ;
nest         = map_parameters.nest ;
food         = map_parameters.food ;

%______________________________________________________________________________
%------------------------ SIMULATION PARAMETERS --------------------------
show_sim         = sim_parameters.show_sim ;
ant_pop          = sim_parameters.ant_pop ;
phmG_red         = sim_parameters.phmG_red  ;
iterations       = sim_parameters.iterations ;
phmG_gain        = sim_parameters.phmG_gain ;
phmG_exp         = sim_parameters.phmG_exp ;
phm_limit        = sim_parameters.phm_limit ;
compass_gain     = sim_parameters.compass_gain ;
keepdir_gain     = sim_parameters.keepdir_gain ;
compass_EN       = sim_parameters.compass_EN ;
keepdir_EN       = sim_parameters.keepdir_EN ;
answer_apriori   = sim_parameters.answer_apriori ;
%______________________________________________________________________________
%------------------------------------------------------------------------------



%______________________________________________________________________________
%-------------------------------- MAP ------------------------------------

% ___MAP___ this matrix represents the available road for the ants
[ map_r , map_c ] = size(map) ;

% ___NEST AND FOOD POINTS___ start point and goal point in XY coordinates
food_XY = [ food(2) map_r+1-food(1) ] ;
nest_XY = [ nest(2) map_r+1-nest(1) ] ;

% ___PHEROMONE MAPS___these are the pheromone maps, the values in each
% matrix represent the concentration of pheromone in every pixel of the map
phmR_map     = zeros(size(map)) ; % red  pheromone : treassure mark
phmG_map     = zeros(size(map)) ; % gree pheromone : food to nest trail
phmB_map     = zeros(size(map)) ; % blue pheromone : nest to food trail

% __MARK THE FOOD with red pheromone
phmR_map(food(1),food(2)) = phm_limit ;
phmB_map(nest(1),nest(2)) = phm_limit ;
%______________________________________________________________________________
```

```
%-----------------------------------------------------------------------------




%__________________________________________________________________________
%------------------------------ COLONY -------------------------------------
% ___POSITION MATRIX___
positions = zeros(ant_pop,2);
for i=1:1:ant_pop
    positions(i,:) = nest ; % all the ants start in the nest
end

% ___INITIAL MODE___ initial mode of all the ants : static
mode = ones(ant_pop,1);

% ___STEP COUNTS___ number of steps taken...
steps_n2f = zeros(ant_pop,1); % ...from nest to food
steps_f2n = zeros(ant_pop,1); % ...from food to nest

% ___PHEROMONE GAINS___
phmG_gains = phmG_gain * ones(ant_pop,1);

% ___PHEROMONE EXPONENTIALS___
phmG_exps = phmG_exp * ones(ant_pop,1);

% ___COMPASS GAINS___
compass_gains = compass_gain * ones(ant_pop,1);

% ___KEEP_DIR GAINS___
keepdir_gains = keepdir_gain * ones(ant_pop,1);

% ___COMPASS enable___
if strcmp(compass_EN,'all')
    compass_en = ones(ant_pop,1);
    elseif (strcmp(compass_EN,'none'))
    compass_en = zeros(ant_pop,1);
end

% ___KEEPDIR enable___
if strcmp(keepdir_EN,'all')
    dir_keep = ones(ant_pop,1);
    elseif (strcmp(keepdir_EN,'none'))
    dir_keep = zeros(ant_pop,1);
end

% ___MEMORY CAPACITY AND MATRIX___ past directions are stored as positive
% integer numbers. Memory capacity: number of past values to be remembered
mem_cap      = 100 ;
past_dirs    = zeros(ant_pop,mem_cap) ;

% ___COLONY MATRIX___
%            1   2    3       4          5        6          7            8
9          10       11           12
colony = [ positions mode steps_n2f steps_f2n phmG_gains phmG_exps
compass_en compass_gains dir_keep keepdir_gains past_dirs ] ;
%__________________________________________________________________________
%-----------------------------------------------------------------------------
```

```
%________________________________________________________________________________
%------------------------------ DRAWINGS ----------------------------------

% ___HALF PIXEL SIZE___ half of the length of every square that composes
% the map, each square is a pixel of the map
r_step = 0.5 ;

% ___ILUSTRATION OF THE MAP___
if (show_sim == 1)||(show_sim == 2)
    figure;hold on;
    axis([ 0.5 map_c+0.5 0.5 map_r+0.5 ])
    draw_map()
    % paint one ant in the nest, cuz all the others are there too
    place_ant(colony(1,1:2),colony(1,3));
end


%________________________________________________________________________________
%--------------------------------------------------------------------------


pause(1)


%________________________________________________________________________________
%------------------------------ TIME LOOP ---------------------------------
time_step = 0.05 ; % in seconds
for iter = 1:1:iterations

    % __MOVE ANTS__
    for i=1:1:ant_pop
        [colony(i,:)] = update_ant(colony(i,:) , iter , i, answer_apriori);
    end

    % ___PHEROMONE EVAPORATION___ reduce the pheromone concentration
    phmG_map = phmG_map*(1-phmG_red) ;

    % pheromone concentration must never go beyond the stablished limits
    phmG_map = encajar_mat(phmG_map,phm_limit,0); % is this zero ???

    if (show_sim == 2)
        % paint road with new values of pheromone
        update_road();

        % paint ants in their new positions
        for i=1:1:ant_pop
            place_ant(colony(i,1:2),colony(i,3));
        end
        txt = strcat('iteration :',32,num2str(iter));
        xlabel(txt)
        pause(time_step)
    end

end
%________________________________________________________________________________
```

```
%-----------------------------------------------------------------------------


if (show_sim == 1)
    % paint road with new values of pheromone
    update_road();

    % paint ants in their new positions
    for i=1:1:ant_pop
        place_ant(colony(i,1:2),colony(i,3));
    end
end

te = toc;

ACO_output.elapsed_time = te ;
ACO_output.phm_map = phmG_map ;
ACO_output.colony = colony ;

end
```

SUB-RUTINA: DIBUJAR MAPA

```
function [] = draw_map()

    global map;          global map_r;        global map_c
    global phmR_map;     global phmG_map;     global phmB_map
    global phm_limit
    global r_step

    for y=1:1:map_r
        for x=1:1:map_c
            if map(map_r+1-y,x)==1
                square_x = [ x-r_step x-r_step x+r_step x+r_step ] ;
                square_y = [ y-r_step y+r_step y+r_step y-r_step ] ;
                color = [ phmR_map(map_r+1-y,x) phmG_map(map_r+1-y,x)
phmB_map(map_r+1-y,x) ] ./ phm_limit;
                fill(  square_x,square_y, color )
            elseif map(map_r+1-y,x)==0
                paint_wall(x,y)
            else
                % in case of something that is not path or wall... we'll
put a
                % grey square on it... it also helps us to know the
orientation
                % of the map
                square_x = [ x-r_step x-r_step x+r_step x+r_step ] ;
                square_y = [ y-r_step y+r_step y+r_step y-r_step ] ;
                fill(  square_x,square_y,[ 0.5 0.5 0.5 ] )
            end
           % pause(0.1)
        end
        %pause(0.2)
    end

end
```

SUB-RUTINA: DIBUJAR HORMIGA

```
function [] = place_ant(pos,state)

    global map_r

    r_step = 0.5 ;
    x = pos(2); y = map_r+1-pos(1) ;

    % always start the drawing from the left lower corner... ALWAYS
    % that will help with the "contour problem"
    ant_draw_x = [ -0.7  -0.7  -0.5  -0.6  -0.3  -0.2  0.2  0.3  0.6   0.5
0.7   0.7  ] ;
    ant_draw_y = [ -0.7   0.1   0.1   0.7   0.7   0.1  0.1  0.7  0.7   0.1
0.1  -0.7  ] ;
    xx = x*ones(1,length(ant_draw_x)) ;
    ant_draw_x = xx + r_step.*ant_draw_x ;
    yy = y*ones(1,length(ant_draw_y)) ;
    ant_draw_y = yy + r_step.*ant_draw_y ;

    if (state==1)
        fill(ant_draw_x,ant_draw_y,[ 0 0.2 0.8 ])
    elseif (state==2)
        fill(ant_draw_x,ant_draw_y,[ 0 0.8 0.2 ])
    end

end
```

SUB-RUTINA: ACTUALIZAR HORMIGA

```
function [ant_f] = update_ant(ant_i , iter , ant_index, answer_apriori)

    global map;          global map_r;
    global phmR_map;     global phmG_map;
    global phm_limit;    global food_XY;

    pos          = ant_i(1,1:2) ;
    mode_i       = ant_i(1,3) ;
    n2f_i        = ant_i(1,4) ;
    f2n_i        = ant_i(1,5) ;
    phmG_gain    = ant_i(1,6) ;
    phmG_exp     = ant_i(1,7) ;
    compass_EN   = ant_i(1,8) ;
    compass_gain= ant_i(1,9) ;
    keepdir_EN   = ant_i(1,10) ;
    keepdir_gain= ant_i(1,11) ;
    past_dir_i   = ant_i(1,12:length(ant_i)) ;

    %______________________________________________________________________
    %---------------------------- SEARCH FOOD -----------------------------

    switch mode_i

        case 0 % MODE 0 : the ant waits its turn to get out
            if ( iter == ant_index )
                mode_f   = 1 ;
            else
                mode_f   = mode_i ;
            end
            pos_f        = pos ;
            past_dir_f   = past_dir_i ;
            n2f_f        = n2f_i ;
            f2n_f        = f2n_i ;

        case 1 % MODE 1 : PROBABILISTIC FORWARD MARCH
            f2n_f = 0 ;

            % SNIFF AROUND : three types of pheromone : R , G and B
            pos_sur_R = sniff(phmR_map,pos) ;
            pos_sur_G = sniff(phmG_map,pos) ;
            pos_sur_G = encajar_mat(pos_sur_G,inf,1) ;

            % SNIFF AROUND : the obstacles in the map
            pos_sur_M = sniff(map,pos);

            % DECISION BASE : random decision component
            zufallig = rand(3,3); % german word that means RANDOM .. i
think xD

            % DECISION : in the begining, all steps are equally elegible
            decision = ones(3,3) ;

            % DECISION : random component
            decision = decision.*zufallig ;
```

```
                % DECISION : map obstacles and make sure the ant MOVES
(center=0)
                decision = decision.*pos_sur_M ;

                % DECISION : green pheromone trail
                decision = decision .* pos_sur_G ;

                % DECISION : compass direction
                if (compass_EN == 1)
                    pos_XY = [ pos(2) map_r+1-pos(1) ];
                    compass_dir = food_XY - pos_XY ;
                    compass_dir = atan2( compass_dir(2) , compass_dir(1) ) ;
                    compass_dir = compass_dir*180/pi ;
                    compass_dir = approx_ang(compass_dir);
                    compass = update_compass(compass_dir, compass_gain);
                    decision = decision.*compass ;
                end

                % DECISION : keep the direction taken
                if (keepdir_EN == 1)
                    keep        = keep_dir(past_dir_i(1,1),keepdir_gain) ;
                    decision    = decision .* keep ;
                end

                % DECISION (last): UPDATE STATE, red pheromone, treassure mark
                [tre_dir_r , tre_dir_c] = find( pos_sur_R >= 1 ) ;
                tre_dir = [ tre_dir_r , tre_dir_c ] ;
                if isempty(tre_dir)
                    tre_mark = ones(3,3);
                    mode_f   = 1 ;
                   else
                    tre_mark = zeros(3,3);
                    tre_mark( tre_dir_r , tre_dir_c ) = 1 ;
                    mode_f   = 2 ;
                end
                decision = decision.*tre_mark ;

                % determine the direction of the step with maximum probability
                [step_dir_r , step_dir_c] = find( decision ==
max(max(decision)) ) ;
                % define the new step direction
                step_dir          = [step_dir_r , step_dir_c] ;
                % store in memory the new taken direction
                past_dir_f        = circshift(past_dir_i,[0,1]) ;
                past_dir_f(1,1) = step_dir_r*10 + step_dir_c ;

                % UPDATE POSITION
                pos_f = [ pos(1)+step_dir(1)-2   pos(2)+step_dir(2)-2 ] ;
                n2f_f = n2f_i + 1 ;

                if (mode_f   == 2)
                      [ past_dir_f , n2f_f ] = erase_loops(past_dir_f) ;
                end

            case 2 % MODE 2 : DETERMINISTIC BACKWARD MARCH

                n2f_f = n2f_i ;
```

```
            phmG_cha = phmG_gain / ((n2f_f - answer_apriori)^phmG_exp) ; %
DELTA TAU

            past_dir_f = past_dir_i ;

            % transform the past direction into a back-step
            step_dir        = past_dir_i(f2n_i+1) ;
            step_dir_str    = num2str(step_dir) ;
            step_dir_r      = str2double(step_dir_str(1)) ;
            step_dir_c      = str2double(step_dir_str(2)) ;
            step_dir        = [ step_dir_r step_dir_c ] ;

            % UPDATE POSITION
            pos_f = [  pos(1)+2-step_dir(1)      pos(2)+2-step_dir(2) ];
            f2n_f = f2n_i + 1 ; % this is my index for the past directions

            if n2f_f == f2n_f ;
                mode_f   = 1 ;
                past_dir_f = zeros(1,length(past_dir_i)) ;
                f2n_f = 0 ;
                n2f_f = 0 ;
               else
                mode_f   = mode_i ;
            end

            % UPDATE MAP : GREEN PHEROMONE
            phmG_map(pos(1),pos(2))  = phmG_map(pos(1),pos(2)) + phmG_cha ;
            phmG_map(pos(1),pos(2))  =
encajar(phmG_map(pos(1),pos(2)),phm_limit,0) ;

        otherwise
            display('ERROR, invalid mode')
    end

    ant_f = [ pos_f , mode_f  , n2f_f , f2n_f , phmG_gain , phmG_exp ,
compass_EN , compass_gain , keepdir_EN , keepdir_gain , past_dir_f ] ;

end
```

SUB-RUTINA: ACTUALIZAR MAPA

```
function [] = update_road()

    global map
    global map_r
    global map_c
    global r_step
    global phmR_map
    global phmG_map
    global phmB_map
    global phm_limit

    for y=1:1:map_r
        for x=1:1:map_c
            if map(map_r+1-y,x)==1
                square_x = [ x-r_step x-r_step x+r_step x+r_step ] ;
                square_y = [ y-r_step y+r_step y+r_step y-r_step ] ;
                color = [ phmR_map(map_r+1-y,x) phmG_map(map_r+1-y,x)
phmB_map(map_r+1-y,x) ] ./ phm_limit;
                fill(  square_x,square_y, color )
            end
        end
    end

end
```

SUB-RUTINA: ENCONTRAR RESULTADO

```
function [result,result_xy,min_val] =
find_result(ACO_output,sim_parameters,map_parameters)

    colony = ACO_output.colony ;

    show_sim = sim_parameters.show_sim ;

    map         = map_parameters.map ;
    nest        = map_parameters.nest ;
    food        = map_parameters.food ;

    colony_col = size(colony,2);
    ant_pop = size(colony,1);
    mem_cap = colony_col - 11;
    ant = zeros(ant_pop,mem_cap);

    samples = 1 ;
    for i = 1:ant_pop
        if colony(i,3)==2
            ant(samples,:) = colony(i,12:colony_col);
            samples = samples + 1 ;
        end
    end

    if samples > size(colony,1) % wut?? ... how ??? ... ok, patch solution
        samples = size(colony,1); % look for the explanation later =/
    end

    solution = zeros(mem_cap,2,samples);
    solution_length = zeros(1,samples);
    pos = zeros(mem_cap,2);
    pos(mem_cap,:) = food ;

    k = 1 ;
    for i = 1:samples
        for ii = 1:mem_cap
            direction = ant(i,ii) ;

            if direction == 0
                break
            end

            % invert the direction
            switch direction
                case 11
                    direction = [3,3];
                case 12
                    direction = [3,2];
                case 13
                    direction = [3,1];
                case 21
                    direction = [2,3];
                case 22
                    direction = [2,2];
                case 23
```

```
                    direction = [2,1];
                case 31
                    direction = [1,3];
                case 32
                    direction = [1,2];
                case 33
                    direction = [1,1];
            end

            pos(mem_cap-ii,:) = [ pos(mem_cap-ii+1,1)+direction(1)-2
pos(mem_cap-ii+1,2)+direction(2)-2 ] ;

            if (pos(mem_cap-ii,:)) == nest
                solution(:,:,k) = pos ;
                solution_length(k) = ii;
                k = k + 1 ;
                break
            end
        end
    end
    k = k - 1 ;

    solution = solution(:,:,1:k);
    solution_length = solution_length(1:k);
    [min_val,min_ind] = min(solution_length);
    result = solution((100-min_val):mem_cap,:,min_ind);

    if (show_sim == 1) || (show_sim == 2)
        % convert result from [rows,cols] to [x,y]
        result_xy = zeros(size(result));
        rows = result(:,1) ;
        cols = result(:,2) ;
        for i = 1:size(result,1)
            result_xy(i,2) = size(map,1)+1-rows(i) ;
            result_xy(i,1) = cols(i) ;
        end
        hold on
        plot(result_xy(:,1),result_xy(:,2),'r-
o','linewidth',3,'markersize',7)
    end

end
```

SUB-RUTINA: LLENAR PUNTOS

```
function [tf,trajectory] = fill_points(V,dt,points)

    X = points(:,1);
    Y = points(:,2);

    plot(X,Y,'r-o','linewidth',2,'markersize',5)
        grid on
        hold on

    ni = length(X); % number of input points

    d = zeros(ni-1,1); % distances between each input point
    for i = 1:(ni-1)
        d(i) = sqrt( (X(i)-X(i+1))^2 + (Y(i)-Y(i+1))^2 );
    end

    tf = sum(d) / V + 40;

    no = length(0:dt:tf); % number of output points

    dD = sum(d) / no ;

    Xv = zeros(no,1);
    Yv = zeros(no,1);

    j = 1 ;
    for k = 1:ni-1

        dX = dD*cos(atan2((Y(k+1)-Y(k)),(X(k+1)-X(k)))) ;
        dY = dD*sin(atan2((Y(k+1)-Y(k)),(X(k+1)-X(k)))) ;

        dX = round(1e10*dX)/1e10 ;
        dY = round(1e10*dY)/1e10 ;
        % 3 problems
        %      dX = 0---> SOLVED
        %      dY = 0---> SOLVED
        %      dX = 0  &  dY = 0 ---> NOT SOLVED ... YET =)

        if (dX==0)&&(dY~=0)
            Ys = ( Y(k) : dY : (Y(k+1)-dY) )' ; % Y segment
            Xs = X(k)*ones(length(Ys),1) ;
            dj = length(Xs) ;
        elseif (dX~=0)&&(dY==0)
            Xs = ( X(k) : dX : (X(k+1)-dX) )' ; % X segment
            Ys = Y(k)*ones(length(Xs),1) ;
            dj = length(Xs) ;
        elseif (dX~=0)&&(dY~=0)
            Xs = ( X(k) : dX : (X(k+1)-dX) )' ; % X segment
            Ys = ( Y(k) : dY : (Y(k+1)-dY) )' ; % Y segment
            dj = length(Xs) ;
        end

        Xv(j:j+dj-1) = Xs ;
        Yv(j:j+dj-1) = Ys ;

%          plot(Xs,Ys,'ro','markersize',5)
```

```
            j = j + dj ;

        end

        trajectory = [ Xv , Yv ] ;
        % the following is a patch solution to avoid that annoying (0,0) point
        % at the end of the trajectory ... no time to fix that for now
        aux = 10;
        trajectory = trajectory(1:(size(trajectory,1)-aux),:);
        tf = tf - aux*dt ;
    end
```