INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

IMPLEMENTACIÓN DE ROBÓTICA INTELIGENTE

MÓDULO 1 | MINI RETO

Implementación de controlador difuso y BUG2

P R E S E N T A :

MARÍA FERNANDA HERNÁNDEZ MONTES - A01704918
IZAC SAUL SALAZAR FLORES - A01197392
MIZAEL BELTRÁN ROMERO - A01114973
NOEMI CAROLINA GUERRA MONTIEL - A00826944

A CARGO DE:
GUILLERMO SOTELO
HERMAN CASTAÑEDA
LUIS ALBERTO UBANDO
RIGOBERTO LÓPEZ

13 MAYO 2022

Objetivo

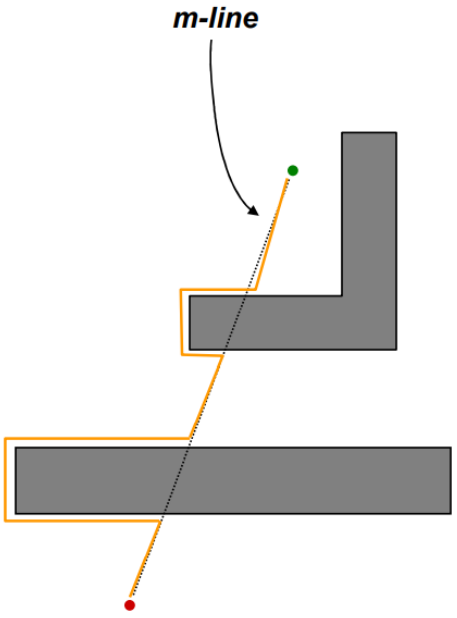
Realizar e implementar un algoritmo de planificación de movimientos para un robot móvil, siguiendo las bases del algoritmo del Bug 2 e incorporando un control difuso para la evasión de obstáculos.

Introducción

Para el desarrollo de este reto se utilizó un controlador de lógica difusa (fuzzy logic) lo cuál permite implementar funciones que creen un documento tipo FIS (Fuzzy Interface System), el cuál es un proceso para formular un mapeo de un input a un output y tomar decisiones en base a las necesidades del programa. Junto con el controlador se utilizó un algoritmo de tipo BUG2 cuya lógica permite mapear una pendiente del punto de inicio al punto destino y seguirla al mismo tiempo que evita obstáculos que se le presentarán en el camino. BUG2 no es parte del controlador en sí sino que es una extensión que se le añadió para permitir la evasión de obstáculos en la ruta.

Algoritmo Bug 2

Los algoritmos "bug" son altamente utilizados para encontrar planes de navegación y movimiento eficientes, gracias a sus comportamientos simples de seguir a una pared y continuar en línea recta hacia la meta. Asumen que se puede calcular la distancia del punto de inicio al punto final, que cuentan con un sensor para identificar los obstáculos y que se tiene un mundo razonable con un número finito de obstáculos de área menor a los límites.



El bug 2 consiste en obtener una recta "m" de la distancia entre el punto de inicio y el punto final. Una vez que es calculada la recta, el robot comienza su desplazamiento siguiendo la línea hasta encontrar un obstáculo. Al encontrar el obstáculo, lo circunnavega (hacia la derecha o izquierda) hasta volver a tocar la línea recta y una vez que la encuentra continúa su camino hacia la meta. El robot hace estos pasos las veces que sean necesarias dependiendo del número de obstáculos hasta alcanzar el destino.

Img 1. Choset H. (s.f.). *Ejemplo Bug 2*

Control difuso

Se realizó un controlador difuso mediante el Matlab Toolbox de tres entradas (centro, izquierda y derecha) y una salida.

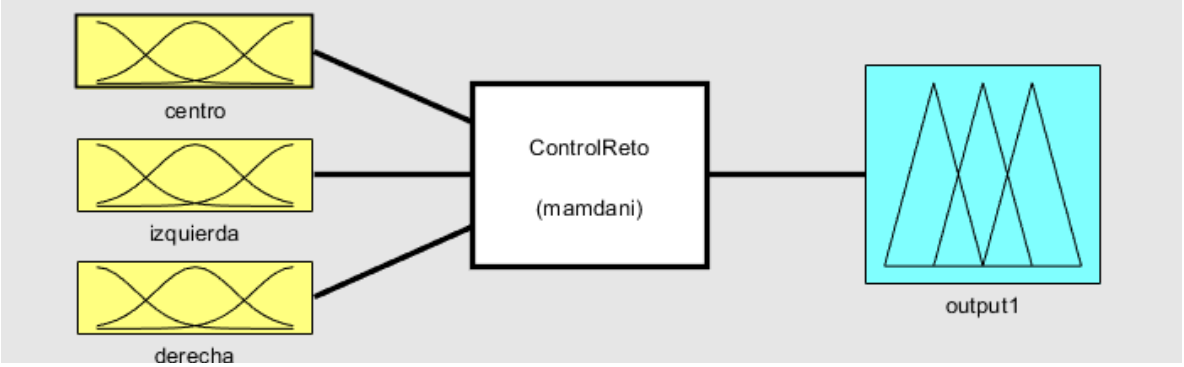


Figura 1. Control difuso

Las entradas cuentan con un rango de 0 a 4 y tres funciones de pertenencia: Muy Cerca (MC), Cerca (C) y Lejos (L). Adicionalmente, las salidas cuentan con un rango de -2 a 2 y con siete funciones de pertenencia: MCR, CR, LLR, LL, Z, CL, y MCL, donde las que tienen una "R" como terminación se refieren a que van hacia la derecha y las de terminación "L" que van a la izquierda. Dependiendo de si se encuentran Muy Cerca, Cerca o Lejos, se tiene diferente comportamiento de salida.

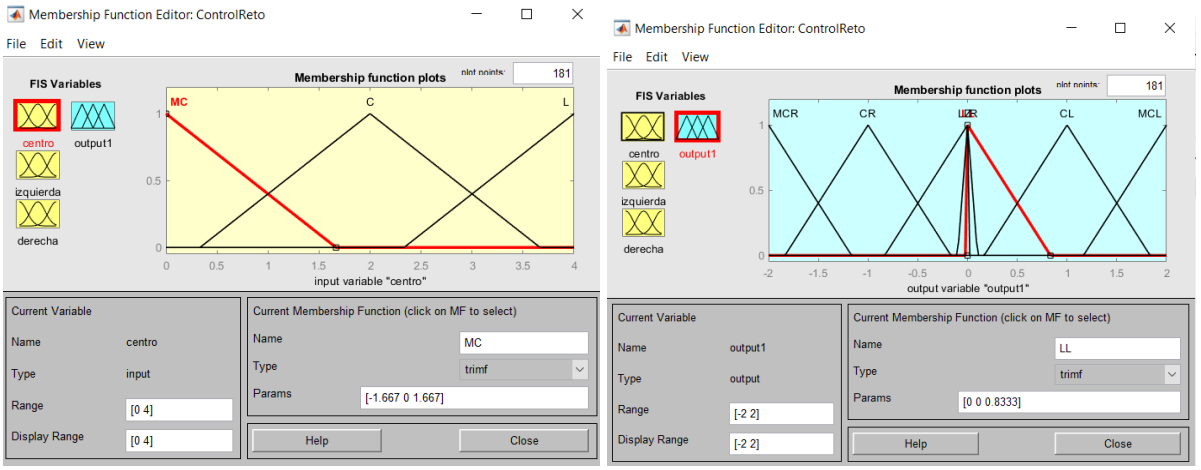


Figura 2. Funciones de membresía y rangos para las entradas y salidas

Posteriormente, se cuenta con las reglas creadas, las cuales forman parte crucial del comportamiento de control. Se tienen 24 reglas en total, enfocadas en lograr que el robot logre circunnavegar un obstáculo continuamente. Con esto en mente, la lógica detrás de las reglas es que si el centro está cerca o muy cerca, la salida será hacia la izquierda, mientras que si el centro se encuentra lejos, la salida será a la derecha. De esta manera, cuando el robot se acerca a un obstáculo, da la vuelta a la izquierda siguiendo su perímetro, y cuando se aleja

del obstáculo va hacia el otro lado para formar la circunnavegación. Estas reglas se expresan en la figura 3 y 4.

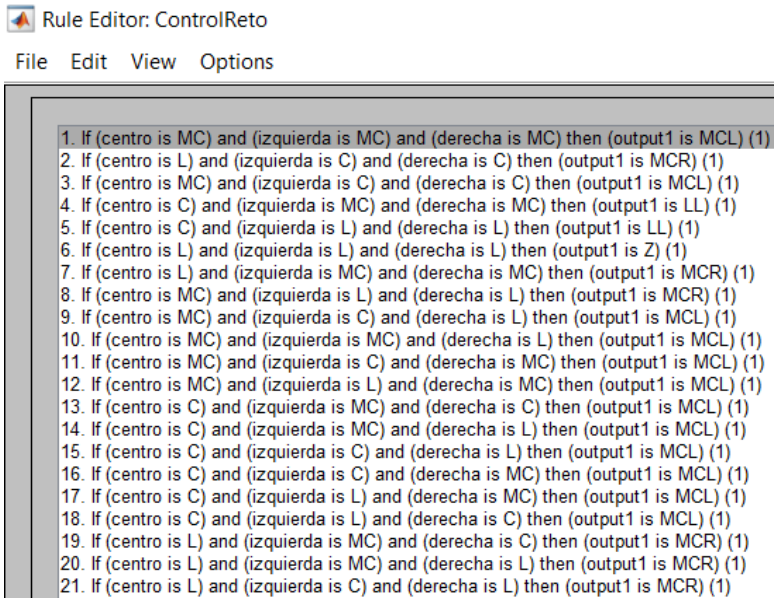


Figura 3. Editor de reglas de control difuso

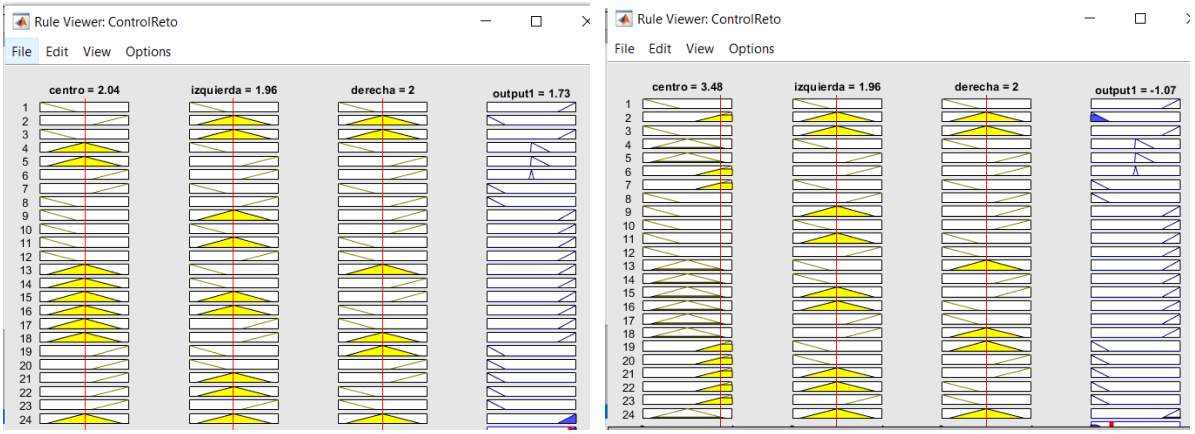


Figura 4. Reglas: Si centro es C o MC salida a la izquierda (1) Si centro es L salida a la derecha (2)

Finalmente, se tienen los siguientes resultados de gráficas de superficie para las entradas centro-izquierda, así como para las entradas centro-derecha.

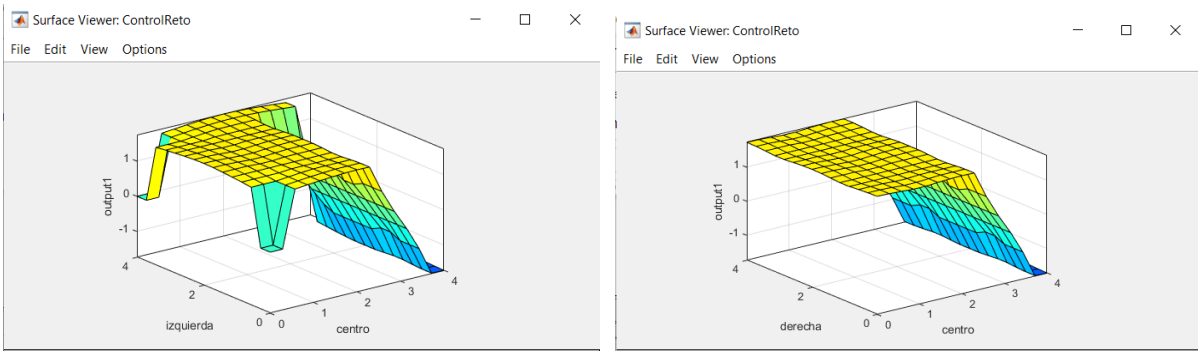


Figura 5. Superficie

Código en MATLAB

Se utilizó Matlab para el desarrollo de este proyecto debido a que se consideró que sería lo más ideal al permitirnos acceder a mucha más documentación, la lógica del controlador difuso era un poco más simple que usando Python y ROS gracias al toolbox de Fuzzy Logic, y en Matlab se puede hacer la simulación directa en el programa sin la necesidad de usar terceros como Gazebo o Rviz.

El código principal implementado, con su debida documentación, es el siguiente:

```
%% Mini-Reto: Modulo 3 Control Difuso
% Algoritmo Bug 2 con control difuso
% Equipo 3:
% Noemi Carolina Guerra Montiel
% Mizael Beltran Romero
% Izac Salazar
% Maria Fernanda Hernandez
% Viernes 13 de mayo del 2022

close all;
clear;

%% Inicializacion de variables

% Cargar matriz de obstaculos
load Obstaculo2.mat
entorno=Obstaculo2;

% Definir mapa y figura 1
refMap = binaryOccupancyMap(entorno,1);
refFigure = figure('Name','Sim robot con obstaculos');
show(refMap);
% Definir mapa y figura 2
[mapdimx,mapdimy] = size(entorno);
map = binaryOccupancyMap(mapdimy,mapdimx,10);
mapFigure = figure('Name','Contornos');
show(map);

% Definir cinematica y controlador PurePursuit
diffDrive = differentialDriveKinematics("VehicleInputs","VehicleSpeedHeadingRate");
controller = controllerPurePursuit('DesiredLinearVelocity',2,'MaxAngularVelocity',3);

% Cargar documento de control difuso
control=readfis('controlReto.fis');

% Lidar
sensor = rangeSensor;
sensor.Range = [0,10];
sensor.HorizontalAngleResolution;

%% Coordenadas

% Coordenadas de incio y meta
path = [4 6;  22 22];
```

```
% Definir x, y de P1 y P2
x1 = path(1,1);
x2 = path(end,1);
y1 = path(1,2);
y2 = path(end,2);
% Calculo de pendiente
m = (y2-y1)/(x2-x1);
% Constante para ecuacion de y
b = y1 - m*x1;
% Vector de puntos en x,y de la linea
xVector = linspace(path(1,1), path(2,1), 100);
yVector = linspace(path(1,2), path(2,2), 100);
pts = [xVector(:), yVector(:)];
x = pts(1);
% Calculo de angulo del segmento de linea
aRectaDeg = atand(m); % Angulo en grados
aRecta = deg2rad(aRectaDeg); % Angulo en radianes

% Llamar a la figura de referencia
figure(refFigure);
hold on
% Plot del segmento de linea del inicio a la meta
plot(path(:,1),path(:,2), 'o-');
hold off

% Definir características del control NO difuso
controller.Waypoints = path;
LookaheadDistance=2;
controller.LookaheadDistance=LookaheadDistance;

% Pose inicial con el robot viendo hacia denlante
initPose = [path(1,1) path(1,2), aRecta];
% Pose de la meta
goal = [path(end,1) path(end,2)]';
% Coordenada x inicial y final
poses(:,1) = initPose';

% Iniciacion del indice y posicion
currX3 = 0;
currY3 = 0;
ind=0;

%% Llamado de funciones
% Mientras el robot no llegue a la meta
while ((x2 - currX3) > 0.3) && ((y2 - currY3) > 0.3)
% Si no es el inicio, inicia la posicion en donde se quedo
if ind > 0
   initPose = [currX3 currY3, aRecta];
end
% Llama la funcion principal con la posicion inicial
% Le pasa el control, x, m, b y el ind como parametros adicionales
[currX2, currY2, ang2] =
exampleHelperDiffDriveCtrl(diffDrive,controller,initPose,goal,refMap,map,refFigure,mapFi
gure,sensor, control, x, m, b, ind, aRecta);
% Posicion actualizada y angulo de la recta
initPose2 = [currX2  currY2, aRecta];
```

```
ind=ind+1;
% Ya que circumnavego un obstaculo y encontro la linea, vuelve a llamar a
% la funcion con la posicion y rotacion actualizada.
[currX3, currY3, ang3] =
exampleHelperDiffDriveCtrl(diffDrive,controller,initPose2,goal,refMap,map,refFigure,mapF
igure,sensor, control, x, m, b, ind, aRecta);
end

%% Funcion para la navegacion del robot
function [currX, currY, ang] =
exampleHelperDiffDriveCtrl(diffDrive,ppControl,initPose,goal,map1,map2,fig1,fig2,lidar,
control, x ,m, b, ind, aRecta)

% Tiempo de simulacion
sampleTime = 0.05;              % Sample time [s]
t = 0:sampleTime:100;          % Time array
% Array para guardar valores de posicion y rotacion
poses = zeros(3,numel(t));    % Pose matrix
poses(:,1) = initPose';

% Tasa de iteracion de estado
r = rateControl(1/sampleTime);

% Obtener los ejes de las figuras
ax1 = fig1.CurrentAxes;
ax2 = fig2.CurrentAxes;
% Array para guardar los angulos
qA = [];
    % Ciclo de inicio hasta que termina el tiempo de simulacion
    for idx = 1:numel(t)
        % Obtener posicion actual
        position = poses(:,idx)';
        currPose = position(1:2);

        % Distancia entre la meta y la posicion actual
        dist = norm(goal'-currPose);

        % Terminar si el robot llega a la meta con tolerancia de 0.3
        if (dist < .3)
            disp("Llegue a la meta!!")
            break;
        end

        % Calcular los valores de y de la recta dependiendo del valor
        % actual de x
        y = m*currPose(1) + b;

        % Actualizar el mapa con las medidas del sensor
        figure(2)
        [ranges, angles] = lidar(position, map1);
        scan = lidarScan(ranges,angles);
        validScan = removeInvalidData(scan,'RangeLimits',[0,lidar.Range(2)]);
        insertRay(map2,position,validScan,lidar.Range(2));
        show(map2);

        % Condicion que avisa si el robot encontro la linea
```

```
        if ( (find(y-currPose(2)) < 0.1 | find((y-currPose(2)) > -0.1)) &
(find((x-currPose(1)) < 0.3) | find((x-currPose(1)) > 0.5)) & (currPose(2) >
poses(2,1)+0.1))
        disp("Llegue a pendiente")
        break;
        end

        % Ejecute el controlador Pure Pursuit y convierta la salida a velocidades de
rueda
        [vRef,wRef] = ppControl(poses(:,idx));

        vs=double(validScan.Ranges);
        an=validScan.Angles;

        vc=10;
        vl=10;
        vr=10;

        % Distancia del robot a los obstaculos
        safeDistance=3;

        for i=1:length(vs)
            if an(i)>-0.5236 && an(i)<0.5236
            if(~isnan(vs(i)))
            if(vs(i)<safeDistance&&vs(i)<vc)
                vc=vs(i);
            end
            end
            end
            if an(i)>-1.5708 && an(i)<-0.5236
            if(~isnan(vs(i)))
            if(vs(i)<safeDistance&&vs(i)<vl)
                vl=vs(i);
            end
            end
            end
            if an(i)>0.5236 && an(i)<1.5708
            if(~isnan(vs(i)))
            if(vs(i)<safeDistance&&vs(i)<vr)
                vr=vs(i);
            end
            end
            end
        end

        if((vc<safeDistance)&&(vc<dist)||...
                (vl<safeDistance)&&(vl<dist)|| ...
                (vr<safeDistance)&&(vr<dist))
            dist;
        if vc>4
        vc=4;
        end
        if vl>4
        vl=4;
        end
        if vr>4
```

```
        vr=4;
        end
        v=[vc; vl; vr];
        % Evaluacion del control difuso
        wRef=evalfis(control,v);
        %pause;
        end

        % Realizar paso de integración discreta hacia adelante
        vel = derivative(diffDrive, poses(:,idx), [vRef wRef]);
        poses(:,idx+1) = poses(:,idx) + vel*sampleTime;


        % Actualizar visualizacion
        plotTrvec = [poses(1:2, idx+1); 0];
        plotRot = axang2quat([0 0 1 poses(3, idx+1)]);
        % Guardar angulos en qA array
        qA(end+1) = poses(3, idx+1);


        % Eliminar la imagen del último robot para evitar mostrar varios robots
        if idx > 1 || ind > 0
           items = get(ax1, 'Children');
           delete(items(1));
        end

        % Plot robot al mapa conocido
        plotTransforms(plotTrvec', plotRot, 'MeshFilePath', 'groundvehicle.stl', 'View',
'2D', 'FrameSize', 1, 'Parent', ax1);
        % Plot robot al mapa nuevo
        plotTransforms(plotTrvec', plotRot, 'MeshFilePath', 'groundvehicle.stl', 'View',
'2D', 'FrameSize', 1, 'Parent', ax2);

        % Esperar para iterar a la velocidad adecuada
        waitfor(r);

    end
    % Salidas de la funcion (posicion y rotacion final)
    currX = currPose(1);
    currY = currPose(2);
    %ang = qA(1, end)
    ang = aRecta;
end
```

Resultados

Se probó el diseño del bug 2 con diferentes matrices de obstáculos y todos los casos llegaron a la meta. En la figura 6 vemos al bug seguir el trayecto de la línea sin ningún obstáculo, mientras que en la figura 7, atraviesa un obstáculo antes de llegar a su destino final. Cuando se encuentra con el obstáculo, lo rodea por el lado izquierdo y continúa su circunnavegación hasta volver a encontrar la línea.

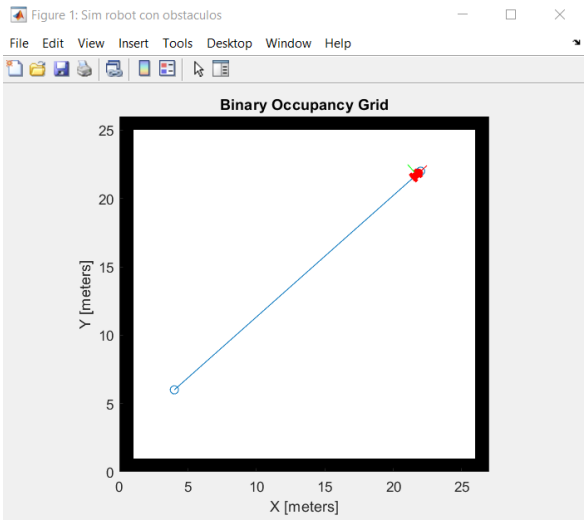


Figura 6. Obstáculo 0

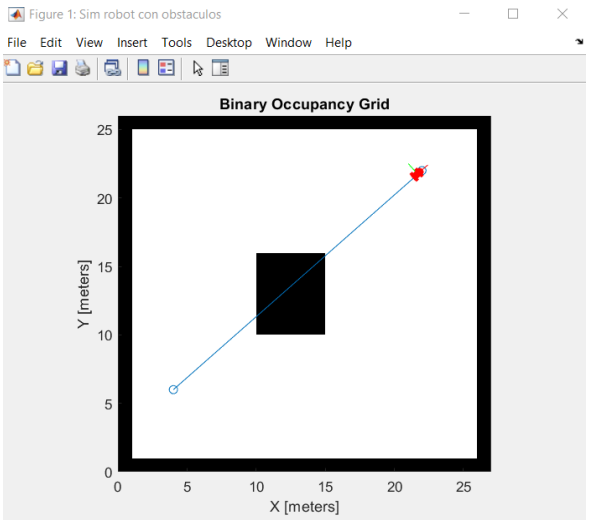


Figura 7. Paredes.mat

Posteriormente, en la figura 8, se tienen dos obstáculos, los cuales el robot logra rodear sin problema y llegar a la meta.

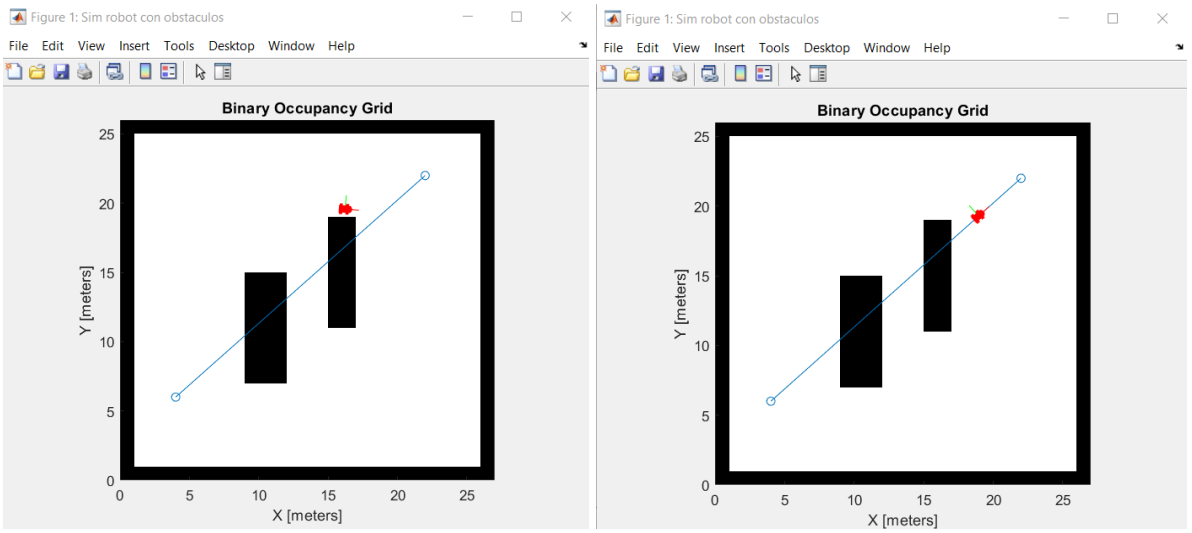


Figura 8. Obstáculo doble

Finalmente, en la figura 9, se tiene un obstáculo más amorfo. Para este último caso, si bien el robot alcanza la meta y realiza la circunnavegación correctamente, el robot no alcanza a girar lo suficientemente rápido, lo cual ocasiona que colisione con uno de los pequeños cuadrados que conforman el obstáculo. Este es el caso que más dio problemas por la naturaleza del obstáculo, por lo que comprobó que el modelo de control funciona mejor cuando el obstáculo tiene una forma relativamente regular y se encuentra alejado del inicio del robot para que este lo alcance a sensar con suficiente tiempo para calcular cómo esquivarlo.

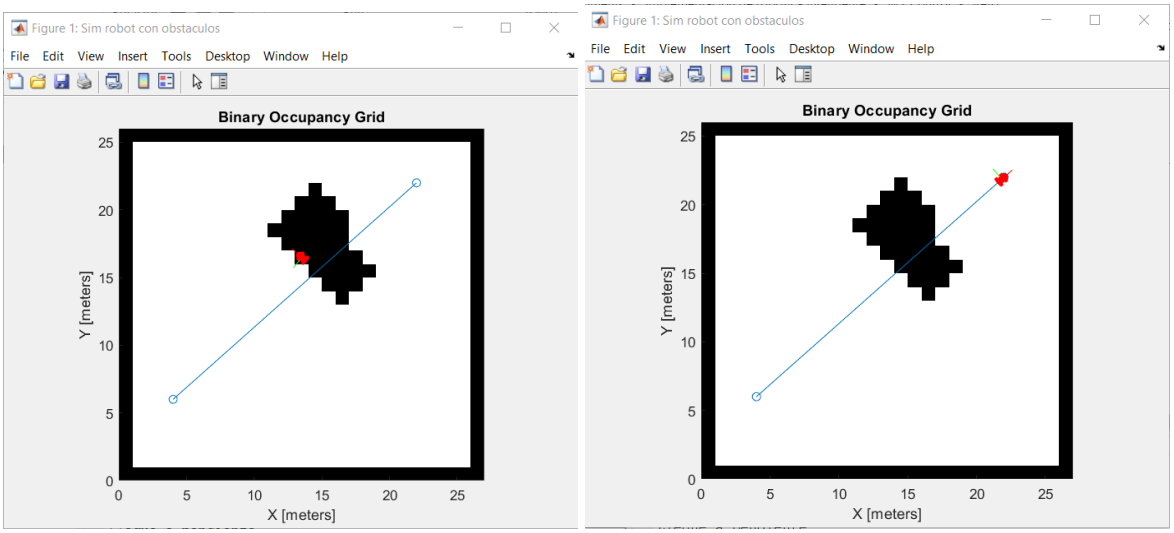


Figura 9. Obstáculo irregular problemático

Las demostraciones de cada uno de estos obstáculos se encuentran en el siguiente video: https://youtu.be/090gA_-aMM8

Conclusión

En conclusión, el Bug 2 es un algoritmo mucho más eficiente que el del Bug 0 previamente implementado y su movimiento puede ser diseñado a partir de una serie de reglas de un controlador difuso relativamente sencillo. Dependiendo de los obstáculos implementados, puede que el algoritmo se desempeñe de mejor o peor manera, por lo que es importante analizar la clase de mapa que se tiene antes de decidir el Bug a implementar. Se pueden hacer algunas mejoras al código, entre ellas hacer que el robot nunca pase el límite de los obstáculos (mediante el cambio o adición de reglas) y hacer que cuando encuentre la línea rote de forma uniforme hasta alcanzar la posición angular de la recta de forma menos brusca. Finalmente, este mini-reto nos ayudó a ver de una manera más visual las aplicaciones en la vida real del control difuso, así como las herramientas, funciones y comandos para su correcta implementación en Matlab.

Bibliografía

Choset H. (s.f.). *Robotic Motion Planning: Bug Algorithms*. Recuperado de: https://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf

Datta S. (2021). *Path-Navigator-Robot*. Recuperado de: https://github.com/souvik0306/Path-Navigator-Robot/blob/master/exampleHelperDiffDriveCtrl.m