Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Modelado y Simulación de una Flota de Drones

Autor: Enrique Carrasco Domínguez

Tutor: D. Eduardo Fernández Camacho

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Modelado y Simulación de una Flota de Drones

Autor:
Enrique Carrasco Domínguez

Tutor:
D. Eduardo Fernández Camacho
Catedrático de Universidad

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado: Modelado y Simulación de una Flota de Drones

Autor: Enrique Carrasco Domínguez
Tutor: D. Eduardo Fernández Camacho

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mis padres Germán y María Eugenia, a mi hermano Germán y a Raquel, a don Eduardo por otorgarme la oportunidad de realizar este proyecto, a mis compañeros y a todas las personas que me han acompañado durante estos años.

Enrique Carrasco Domínguez
Sevilla, 2019

Resumen

El objetivo de este Proyecto de Fin de Grado es el estudio y la simulación de las trayectorias que una flota de drones tendrá que seguir para completar una misión específica como es el reconocimiento de un terreno, cumpliendo diferentes requisitos. Contiene el desarrollo básico para un único dron, para dos drones y su interacción entre ellos y finaliza con el control de una flota compleja de ocho drones. La programación ha sido elaborada en MATLAB y se ha dado forma al proyecto con sus herramientas SimuLink y SimMechanics.

Abstract

The purpose of this final proyect is the simulation of the trajectories that a drone fleet must follow to complete a mission such as ground reconnaissance or ground recon following specific and diverse requirements. It contains the basic development for a single drone, two drones and the interaction between them and the control of a complex fleet of eight drones. The code has been programmed using MATLAB and its tools SimuLink and SimMechanics.

Índice

1 Objetivos e introducción

El presente proyecto tiene como objetivo la simulación de una flota de drones para el análisis o la exploración de una cierta área de terreno focalizándose en la elaboración de las trayectorias que cada uno de los drones tendrá que seguir para que así el proceso se lleve a acabo de forma optimizada. Dado que se supondrá que cada uno de los vehículos aéreos no tripulados dispondrá de su propio sistema de control a bajo nivel, en este proyecto se llevará a cabo un desarrollo a alto nivel sobre las trayectorias, optimización y evaluación de las mismas, y por supuesto de su seguridad.

2 Modelo del dron y entorno utilizado

En la realización de este proyecto se ha decidido hacer uso de vehículos aéreos no tripulados de tipo cuadrirrotor, comúnmente conocidos como quadcopters. Este modelo se caracteriza por su versatilidad y maniobrabilidad además de que actualmente en el mercado existen una infinidad de modelos en diferentes rangos de precio, autonomía y capacidad de carga, lo que abriría la posibilidad de implementar lo expuesto en este proyecto en todo rango de actividades con un nivel de presupuesto bajo o elevado y para aplicaciones tanto en el ámbito civil como militar.

Este proyecto de fin de grado se basa en el trabajo inicial de Mohamed Abdelkader Zahana, *Multi-Quadrotor Control using Simulink and SimMechanics* [3], que se encuentra ubicado en la plataforma *File Exchange* de MathWorks. El proyecto mencionado permite la simulación de varios drones en el mismo entorno además de permitir su visualización utilizando el recurso SimMechanics de MATLAB, figura 2.1. Gracias a esto, es posible tener una representación virtual de los drones a tiempo real y sus movimientos. Según se indica, si se desea añadir más vehículos habría que duplicar los bloques los drones.

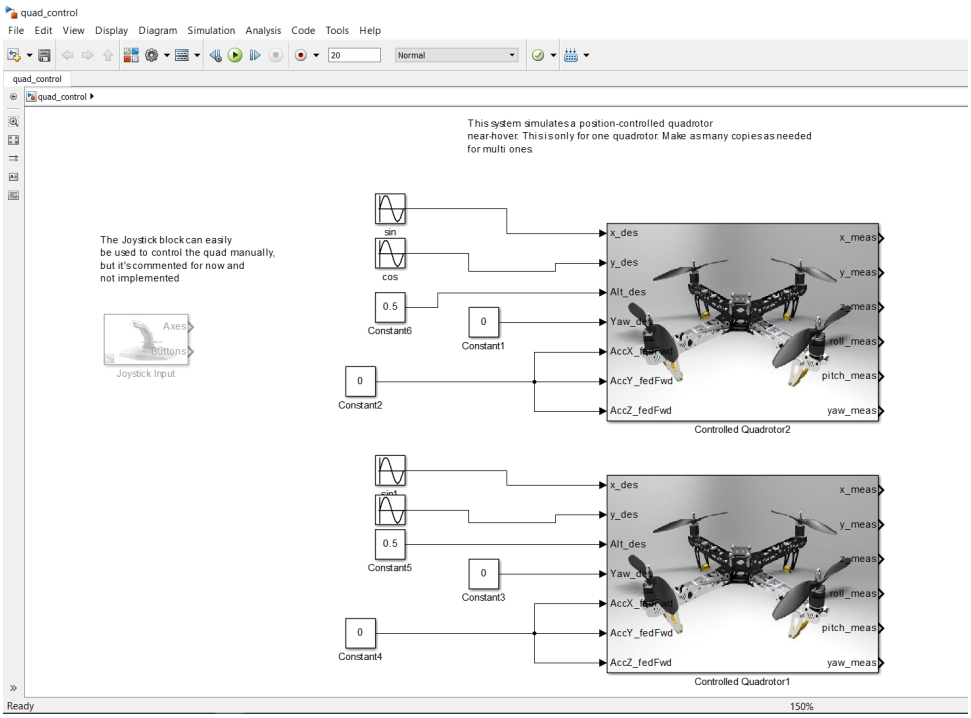

Figura 2.1 Modelo de SimuLink del proyecto incial.

Dentro de cada bloque, existen subsistemas que permiten el control del dron así como su representación en el entorno SimMechanics. Se compone de tres subsistemas, el primero es el controlador de posición, el segundo el controlador de altitud y actitud y el tercero es el que modela el dron, figura 2.2.

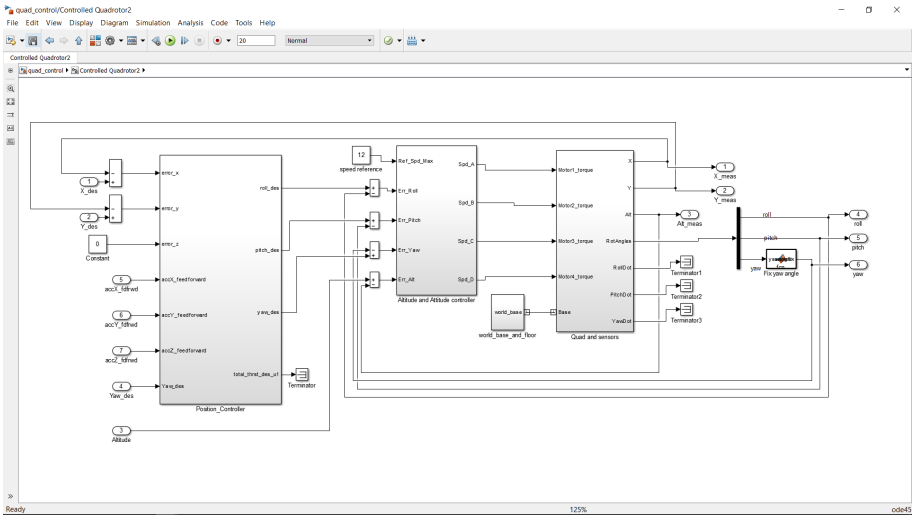

Figura 2.2 Bloque Quadrotor en SimuLink.

El bloque de control de posición, figura 2.3, tiene como entradas el error en la posición según los ejes x, y, z y el ángulo de guiñada. Como salidas ofrece los ángulos de alabeo, cabeceo, guiñada además del empuje total deseado. Los errores en la posición están controlados por un controlador tipo proporcional y derivativo (PD).

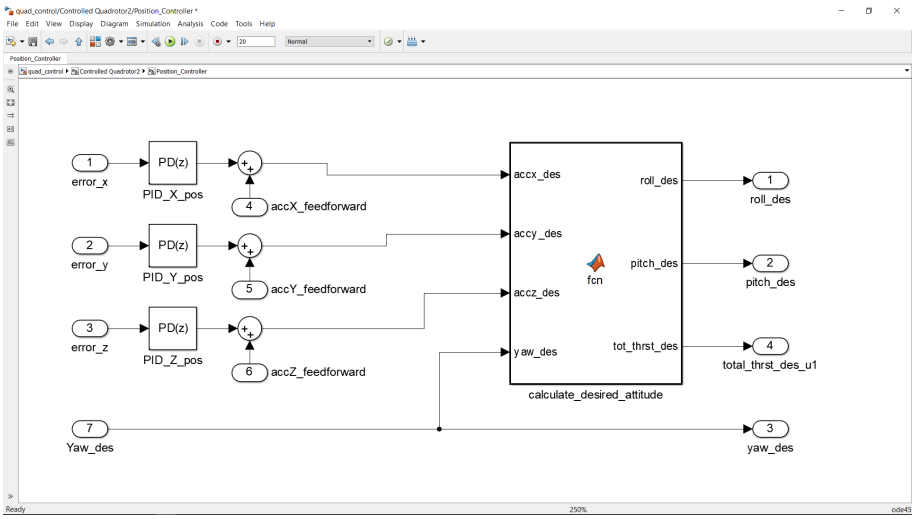

Figura 2.3 Bloque control de posición.

El bloque de control de la altitud y actitud, figura 2.4, tiene como entradas la velocidad máxima de referencia, el error en altitud, alabeo, cabeceo y guiñada. Para realizar la acción de control utilizan controladores de tipo proporcional y derivativo (PD) excepto para el error de altitud, que emplea un controlador tipo proporcional integral y derivativo (PID). Las salidas de este bloque son las velocidades de referencia de cada uno de los cuatro motores del dron.

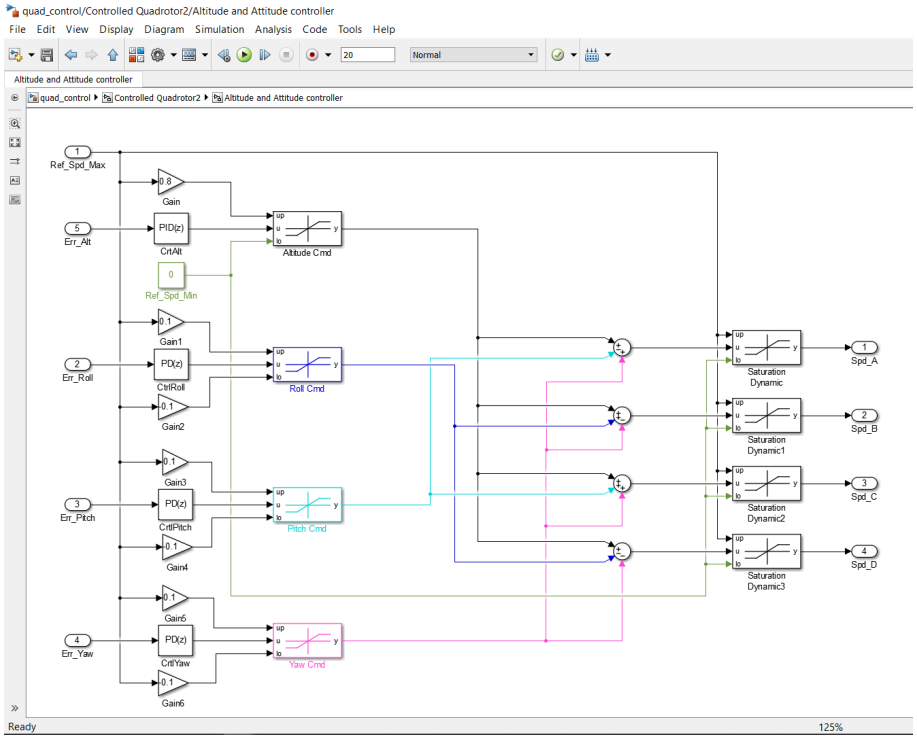

Figura 2.4 Bloque control de altitud y actitud.

El bloque del dron y sus sensores, figura 2.5, tiene como entrada la velocidad de referencia de cada uno de los motores del dron y además el modelo del entorno para su simulación en SimMechanics. El modelo da como resultado la posición del dron y sus ángulos. Se puede observar como dos de los motores giran en sentido opuesto.

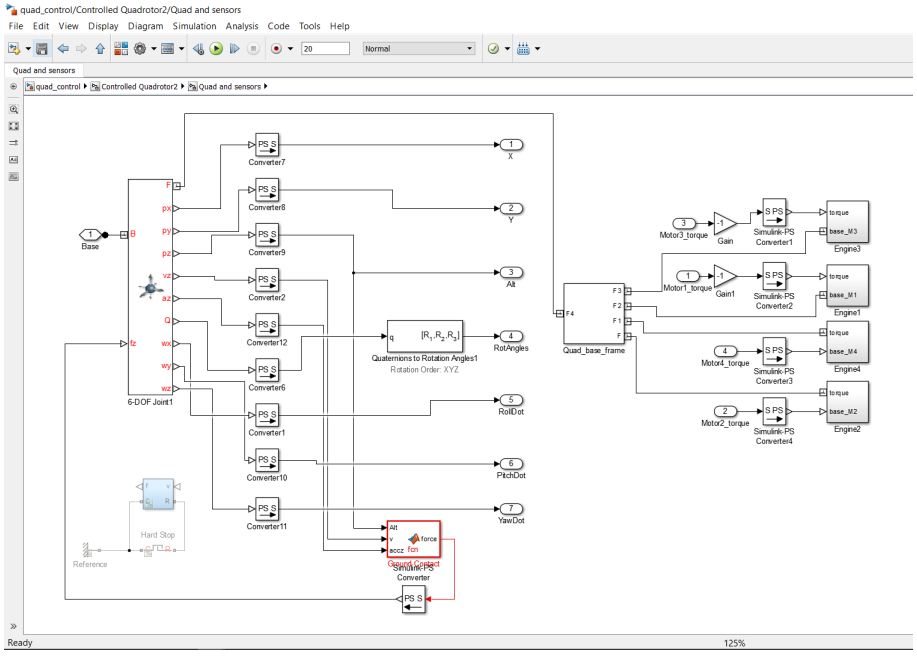

Figura 2.5 Bloque control del dron y sus sensores.

En el bloque de simulación del entorno, figura 2.6, se definen los parámetros que rigen el entorno

por el que circulará el dron y que permitirá su representación en la aplicación SimMechanics de MATLAB.

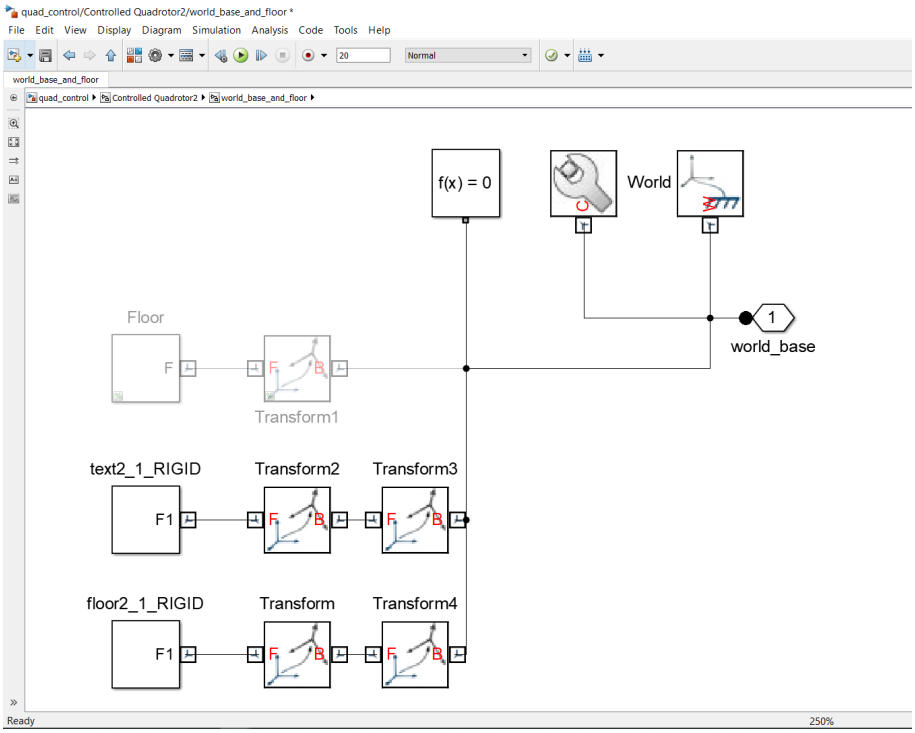

Figura 2.6 Bloque de simulación del entorno.

En la figura 2.7 se muestra la representación tridimensional del modelo del dron en el entorno de SimMechanics.

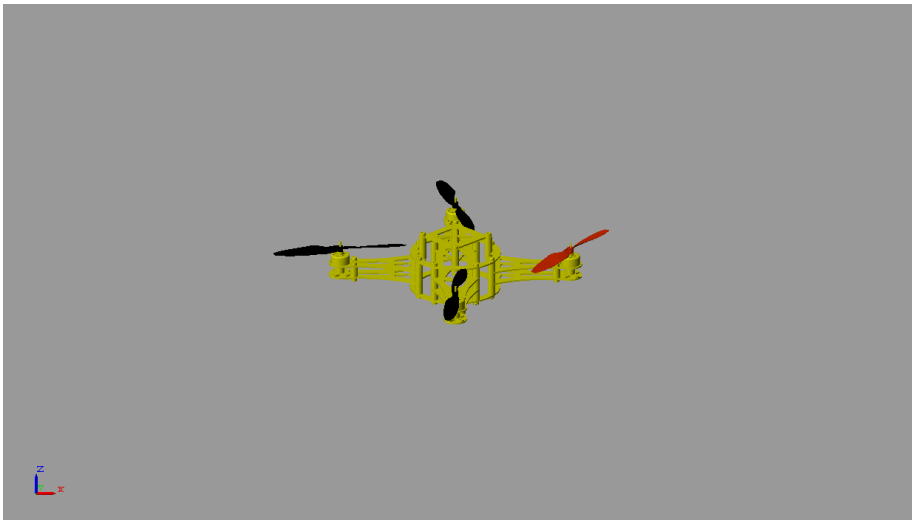

Figura 2.7 Entorno SimMechanics.

3 Algoritmos para un único dron

El desarrollo técnico del proyecto comenzará con los algoritmos y programas necesarios para la elaboración de las trayectorias referidas a un único vehículo aéreo no tripulado. Para representar de manera más ilustrativa los resultados de los algoritmos, además de hacer uso de la función *plot3.m* de MATLAB, se utiliza el entorno representado en el simulador con *SimMechanics*. Este entorno de simulación contiene un modelo de quadrotor cuya dinámica interna no se modificará ya que este proyecto se focaliza en la elaboración y optimización de las trayectorias que tendrán que seguir los diferentes vehículos aéreos no tripulados.

3.1 Control de trayectorias para un dron

Considerando la trayectoria que seguirá el dron, es necesario crear una función que conociendo la posición actual del vehículo y los puntos por los que éste debe circular, genere como resultado una serie de puntos adecuada. En el modelo de Simulink se define el bloque *Seguidor Waypoints*, código 3.1, que tiene como entradas la posición actual del vehículo aéreo no tripulado, los diferentes waypoints por los que éste debe circular y de manera adicional se le ha añadido un punto denominado destino final donde debería permanecer una vez haya recorrido todos los waypoints [5].

Es necesario establecer una variable global *A* de tipo *Simulink.signal* que vaya iterando según se recorran los diferentes puntos de la trayectoria. Conociendo la posición actual del vehículo y el punto que se quiere alcanzar, es posible definir una variable denominada *distancia* que calcula a qué distancia en línea recta se encuentra el punto objetivo.

La finalidad de esta es poder dar un volumen de tolerancia que será modificable por el usuario dando por válido el paso por un waypoint de manera más holgada y sensata. En esta primera simulación se ha decidido que este volumen de tolerancia sea una esfera de radio 0.2 *metros*. Una vez el vehículo se encuentre a una distancia inferior a la acordada, la variable global incrementaría seleccionando así el nuevo waypoint.

Código 3.1 Seguidor de waypoints para un único dron.

```
1  % ================================================================
2  % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3  % ==========          ENRIQUE CARRASCO DOMÍNGUEZ          ==========
4  % ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019    ==========
5  % ================================================================
6  % ==========        FUNCIÓN SEGUIDOR DE WAYPOINTS         ==========
7  % ================================================================
8
9  function [ruta,distancia,a] =seguidorwaypoints(posicion_actual,...
```

```
                                                    waypoints,destino_final)

x=posicion_actual(1); y=posicion_actual(2); z=posicion_actual(3);

waypoints=[waypoints;(destino_final)'];

[m,n]=size(waypoints); global A;

distancia=sqrt((waypoints(A,1)-x)^2+(waypoints(A,2)-y)^2+...
               (waypoints(A,3)-z)^2);

if distancia<0.2, A=A+1; end

if A>m, A=m; end

ruta=waypoints(A,:)'; a=length(waypoints)-A;

end
```

Se realizan varias simulaciones con una serie de waypoints predefinidos, código 3.2 para comprobar que funcionamiento del sistema es todo un éxito.

Posteriormente se modificarán las funciones para así dar la posibilidad de que varios vehículos barran la superficie de forma más rápida y se anotará el tiempo del recorrido total para las diversas configuraciones del terreno y precisión en los waypoints permitiendo así realizar una comparación en las futuras configuraciones. Los resultados se muestran en la figura 3.1.

Código 3.2 Primera simulación un único dron.

```
% ====================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
% ==========    TRABAJO DE FIN DE GRADO, SEVILLA 2019      ==========
% ====================================================================
% =========       MODELO UN DRON PRIMERA SIMULACIÓN         =========
% ====================================================================

posini1=[0 0 0];

waypoints = [0 0 0; 1 0 1; 2 0 1; 2 2 1; 3 2 1; 4 2 2];

destino_final = [waypoints(length(waypoints),:)];

sim('quad_control')

figure(1)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(waypoints(:,1),waypoints(:,2),waypoints(:,3),'o-','LineWidth',1.25);
grid on;

title('Primera Simulación un Único Dron','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
legend('Dron 1','Waypoints');
```

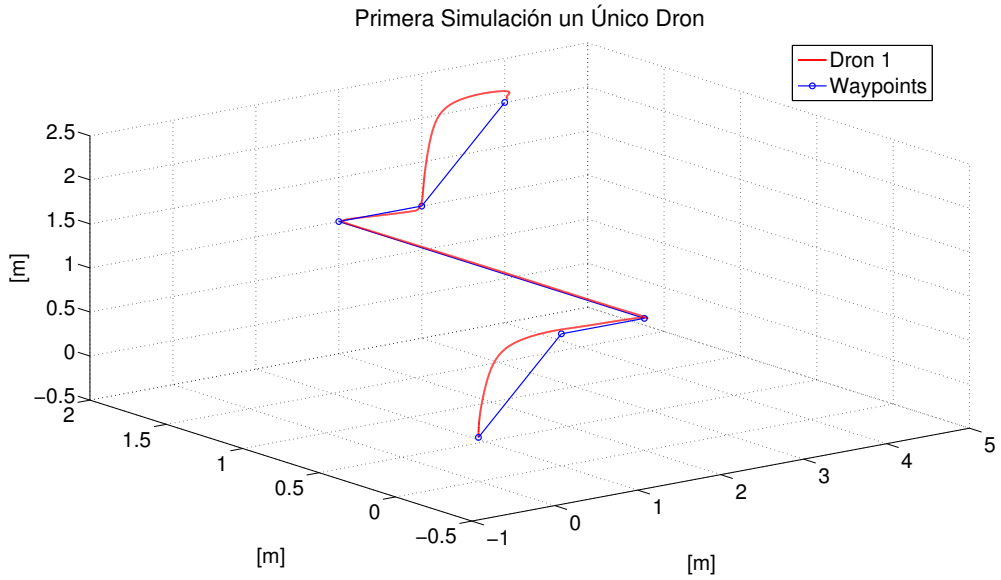

Figura 3.1 Primera simulación un único dron.

La mejor forma de cronometrar el tiempo empleado en realizar el recorrido es creando una función *Cronometro*, código 3.3, que tendrá como entrada el tiempo de simulación, el número de waypoints restantes para el final del recorrido, así como la distancia al punto objetivo para poder paralizar el tiempo de un modo más restrictivo. Con el bloque *Stop Simulation* una vez se cumpla las condiciones waypoints restantes igual a cero y distancia menor a la establecida, la simulación se detendrá, pudiéndose calcular así de forma precisa la duración del recorrido.

Código 3.3 Cronómetro para un único dron.

```
% ====================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ====================================================================
% ==========               FUNCIÓN CRONÓMETRO              ==========
% ====================================================================

function y  = cronometro(t,a,distancia)
if t>0 && a==0 && distancia<0.05,
     y=1;
else y=0;
end
```

3.2 Barrido del terreno en una superficie plana

La manera más fácil de comenzar el desarrollo es suponiendo un área plana de superficie cuadrada con una longitud dada. Para ello se usará la función *surf.m* de MATLAB indicando la longitud de los ejes x y z. Para el eje x se necesita un vector unidimensional cuya longitud coincida con la del terreno, así mismo para el eje y. Como la superficie será plana el eje z será una matriz de ceros en concordancia con los ejes x e y. El color elegido para el terreno será el verde. La superficie vendrá definida según el código 3.4

Código 3.4 Generación de superficie plana.

```
% ================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ================================================================
% ==========        GENERADOR DE SUPERFICIE PLANA         ==========
% ================================================================

superficie=[20,30]; dimx=superficie(1); dimy=superficie(2);

figure(1)
x=0:dimx; y=0:dimy; z=zeros(length(x),length(y));
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none')
camlight left;
lighting phong

title('Generación de Superficie Plana','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
```

En esta primera toma de contacto la superficie estará constituida por una serie de puntos cuya precisión es baja, se ha definido un punto por metro, formando una malla. Posteriormente se realizará un modelado del terreno con mayor precisión para efectuar las distintas simulaciones. Para cambiar la precisión con la que se crearía la superficie es tan sencillo como incluir en los vectores que definen tanto al eje x como al y, un paso determinado. Por defecto al insertar en MATLAB $x = 0:5$ éste crea un vector que contiene los números del cero al cinco con un paso de uno. Al añadir un paso menor, por ejemplo 0.5, la longitud del vector $x = 0:0.5:5$ se incrementaría y consecuentemente el número de puntos en la malla haciendo que la superficie sea más precisa. El resultado se muestra en la figura 3.2

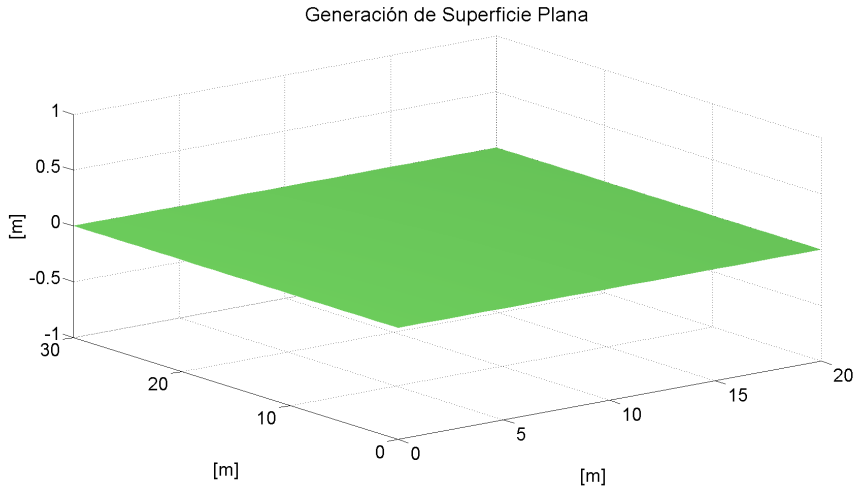

Figura 3.2 Generación de superifice plana.

Una vez definida la superficie se necesitará calcular un recorrido formado por varios waypoints por los que el dron tendrá que circular para así completar la tarea deseada, ya sea la toma de datos, exploración o supervisión de la superficie.

El primer método de recorrido de la superficie será el más lógico y sencillo implementable. Se empezará desde una esquina de la superficie y se barrerá en línea recta con un espaciado entre líneas de recorrido variable por el usuario. El espaciado será denominado como precisión e indicará

cuantos puntos, o metros, se desplazará el dron antes de comenzar el barrido de la próxima línea de terreno. Se creará una matriz de waypoints ordenada de tal forma que cumpla con los requisitos expuestos anteriormente, código 3.5.

Código 3.5 Generador de waypoints en superificie plana.

```
% ================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ================================================================
% ==========          GENERADOR DE WAYPOINTS PLANO         ==========
% ================================================================

function waypoints = generador_waypoints_plano(superficie,...
                     altura_barrido,precision)

dimx=superficie(1); dimy=superficie(2);

precisionx=precision(1); precisiony=precision(2);

puntosx=[0:precisionx:dimx]';
puntosxinv=[dimx:-precisionx:0]';
puntosy=[0:precisiony:dimy]';
alpha=ones(size(puntosx));
e=1; wp=[];

    for i=1:length(puntosy)
        wp=[wp; abs(((e+1)/2)*puntosx+((e-1)/2)*puntosxinv),...
            alpha*puntosy(i) ];
        e=-e;
    end

%Superficie Plana: (altura_barrido = constante):

wp=[wp, ones(length(wp),1)*altura_barrido];
waypoints=wp;

end
```

Una vez obtenida la matriz que contiene los waypoints se representará en el mismo gráfico, figura 3.3, añadiendo la superficie para comprobar así el funcionamiento del algoritmo.

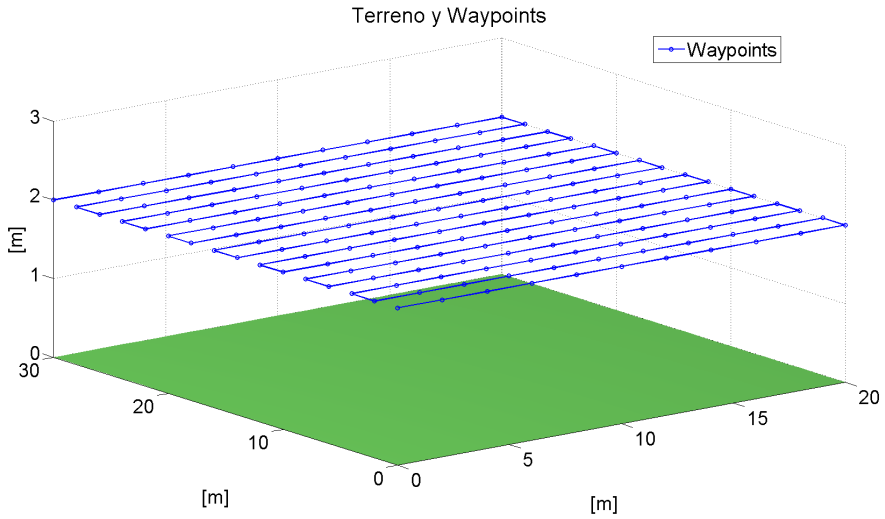

Figura 3.3 Terreno y waypoints generados en superficie plana.

Para completar el desarrollo, se simula el modelo del dron, código 3.6, junto con la superficie y waypoints generados, figura 3.4.

Código 3.6 Simulación un dron en superficie plana.

```
% ====================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
% ==========   TRABAJO DE FIN DE GRADO, SEVILLA 2019       ==========
% ====================================================================
% ==========       MODELO UN DRON SUPERFICIE PLANA         ==========
% ====================================================================

posini1=[0 0 2]; superficie=[20 20]; altura_barrido=2; precision=[2 2];

waypoints = generador_waypoints_plano(superficie,altura_barrido,precision);

destino_final = [waypoints(length(waypoints),:)];

sim('quad_control')

TIEMPO=max(t)

dimx=superficie(1);
dimy=superficie(2);
x=0:dimx; y=0:dimy; z=zeros(length(x),length(y));

figure(1)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(waypoints(:,1),waypoints(:,2),waypoints(:,3),'o-','LineWidth',1.25);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Simulación Dron en Superficie Plana','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Waypoints');
```

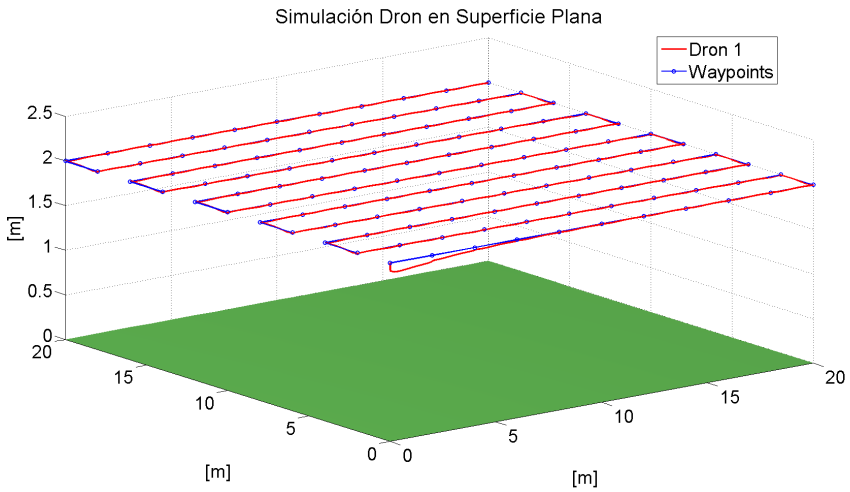

Figura 3.4 Simulación de un dron en superficie plana.

Se comprobará el funcionamiento en diversas situaciones con dimensiones del terreno y precisión en los waypoints variables, tabla 3.1.

Tabla 3.1 Tiempo de barrido en superficie plana con un único dron.

Superficie [m]	Precisión [m]	Tiempo [s]
20 x 20	2 x 2	166.03
50 x 50	2 x 2	932.28
50 x 50	1 x 2	1635.42
40 x 20	1 x 1	1059.60

3.3 Barrido del terreno en una superficie montañosa

El objetivo de este proyecto es que pueda ser en un futuro explotado en un entorno real. Es por ello por lo que se recreará una superficie con un nivel irregular que se asemeje con la mayor precisión posible a un terreno cualquiera. Para facilitar esta tarea se hará uso de la función *peaks* que ofrece una superficie tal y como se requiere, figura 3.5.

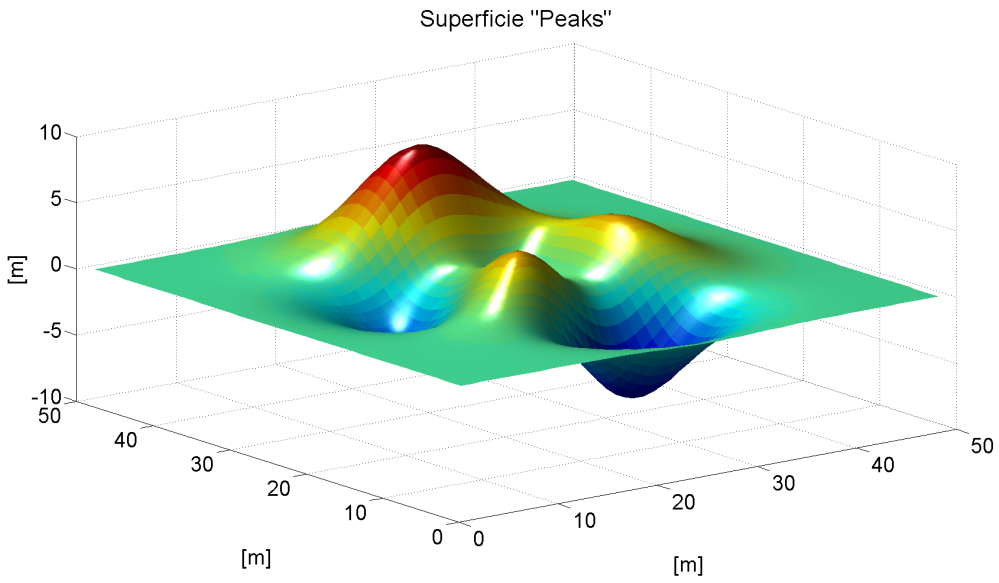

Figura 3.5 Superficie generada con la función *Peaks*.

Dado que las futuras aplicaciones derivadas del proyecto no podrían hacer uso de un terreno demasiado irregular es necesario suavizar la función para asemejar más la superficie a una que realmente pueda ser válida en la realidad. El problema se puede solucionar fácilmente multiplicando la función por un valor que se hará variable según la extensión del terreno a analizar.

Se ha decidido que la altura máxima de la superficie sea aproximadamente un 10% de la longitud del terreno, para ello se multiplica la función *peaks* por $0.01 * superficie(1) siendo superficie(1)$ la longitud del terreno respecto al eje x. Es necesario modificar la posición de la superficie dado que el modelo utilizado contiene un suelo impenetrable a una altura $z = 0$. Para ello se eleva el terreno

de modo que el punto más bajo quede a la altura $z = 0$.

Ahora hay que actualizar la función generadora de waypoints para que la trayectoria del dron sea concordante con el terreno, código 3.7. Esta función tendrá como entrada adicional la distribución según el eje z de la superficie. El procedimiento es análogo al de la superficie plana cambiando únicamente la tercera columna de la matriz por la altura de la superficie en ese punto y añadiéndole el valor de la altura de barrido deseada.

La función *peaks* tiene como punto de origen el $(1,1)$ así que se desplazarán todos los waypoints hacia la derecha sumándole una matriz de unos del mismo tamaño que la de los waypoints. Es preciso tener en cuenta que en el eje z también se estaría sumando uno, se corrige sin mayor dificultad el problema restándole una unidad al valor de ese mismo eje.

Código 3.7 Generador waypoints en superficie montañosa.

```
% ====================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ====================================================================
% ==========         GENERADOR DE WAYPOINTS MONTAÑA        ==========
% ====================================================================

function waypoints = generador_waypoints_montana(superficie,z,...
                     altura_barrido,precision)

dimx=superficie(1); dimy=superficie(2);

precisionx=precision(1); precisiony=precision(2);

puntosx=[0:precisionx:dimx]';
puntosxinv=[dimx:-precisionx:0]';
puntosy=[0:precisiony:dimy]';
alpha=ones(size(puntosx));
e=1; wp=[];

for i=1:length(puntosy)
    wp=[wp; abs(((e+1)/2)*puntosx+((e-1)/2)*puntosxinv),...
        alpha*puntosy(i) ];
    e=-e;
end

wp=[wp,zeros(length(wp),1)];

for i=1:length(wp)
    wp(i,3)=z(wp(i,2)+1,wp(i(1))+1)+altura_barrido-1;
end

wp=wp+ones(size(wp));
waypoints=wp;

end
```

Se realiza una simulación, código 3.8, para comprobar el funcionamiento del programa descubriendo así un comportamiento que, aunque podría considerarse válido, se corregirá mediante interpolación, figura 3.6. Según está definido el modelo empleado éste, una vez establecido el waypoint posterior, si está a una altura diferente ya sea superior o inferior, se nivela de manera rápida originando un recorrido por escalones.

Código 3.8 Simulación un dron en superficie montañosa.

```
% ====================================================================
```

```
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
% ==========   TRABAJO DE FIN DE GRADO, SEVILLA 2019       ==========
% ======================================================================
% ==========      MODELO UN DRON SUPERFICIE MONTAÑOSA      ==========
% ======================================================================

superficie=[20 20]; altura_barrido = 2; precision = [2 2];
z = 0.01*superficie(1)*peaks(superficie(1)+1);
[Q,~] = min(z); Q = -min(Q); z = z+Q*ones(size(z));

waypoints = generador_waypoints_montana(superficie,z,altura_barrido,...
                                        precision);

posini1 = waypoints(1,:);

destino_final = [waypoints(length(waypoints),:)];

sim('quad_control')

TIEMPO=max(t)

figure(1)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(waypoints(:,1),waypoints(:,2),waypoints(:,3),'o-','LineWidth',1.25);
hold on; surf(z,'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Simulación Dron en Superficie Montañosa','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Waypoints');
```

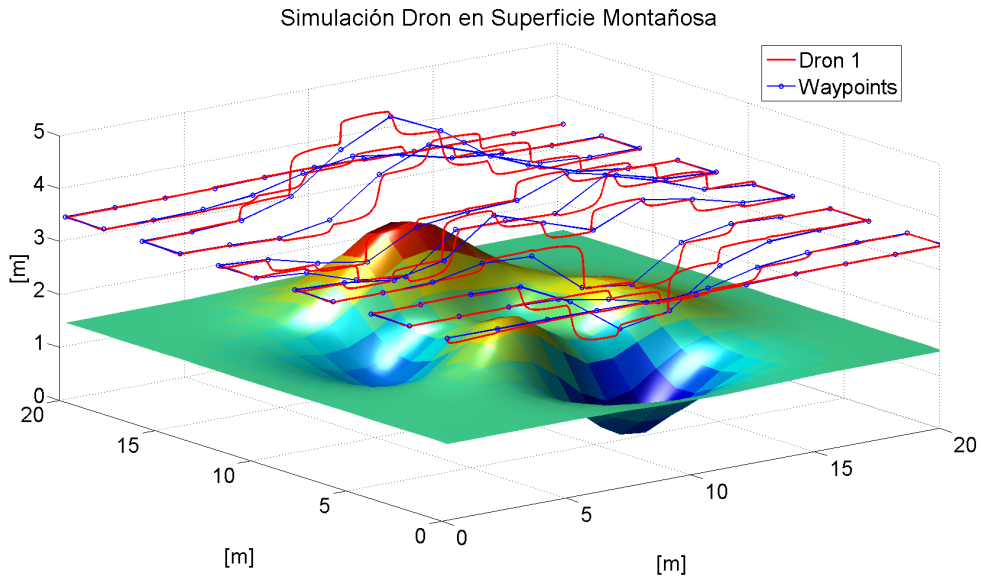

Figura 3.6 Simulación de un dron en superficie montañosa.

Como primera aproximación se utilizará una interpolación lineal para la altura del dron, por definición una interpolación lineal viene dada por la expresión (3.1):

$$z = \frac{z_1 - z_0}{x_1 - x_0}\ (x - x_0) + z_0 \tag{3.1}$$

Se modificará la función *Seguidor de Waypoints* para que modifique la altura del vehículo de forma progresiva según la interpolación lineal, añadiendo la función de interpolación y excluyendo los casos de los extremos cuando la variable global es igual uno y cuando coincide con la dimensión de la matriz waypoints, código 3.9. Se suprimen estas condiciones debido a que de lo contrario se estaría buscando una posición en la matriz de waypoints que no existe provocando un error en el programa. Además, para que este método funcione en todas las ocasiones, se descarta la condición en la cual el denominador se hace nulo hecho que, si se dan las condiciones, provocaría el fallo del programa al dar como resultado un valor infinito.

Código 3.9 Seguidor Waypoints con interpolación lineal.

```
% ==================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ==================================================================
% ==========        FUNCIÓN SEGUIDOR DE WAYPOINTS          ==========
% ==================================================================

function [ruta,distancia,a] =seguidorwaypoints(posicion_actual,...
                                               waypoints,destino_final)

x=posicion_actual(1); y=posicion_actual(2); z=posicion_actual(3);

waypoints=[waypoints;(destino_final)'];

[m,n]=size(waypoints); global A;

distancia=sqrt((waypoints(A,1)-x)^2+(waypoints(A,2)-y)^2+...
               (waypoints(A,3)-z)^2);

if distancia<0.2, A=A+1; end

if A>m, A=m; end

ruta=waypoints(A,:)'; a=length(waypoints)-A;

% Para suavizar las subidas y bajadas, interpolación lineal.

if A~=1 && A~=m && waypoints(A,1)-waypoints(A-1,1)~=0
    ruta(3) = waypoints(A-1,3)+((waypoints(A,3)-waypoints(A-1,3))/...
              (waypoints(A,1)-waypoints(A-1,1)))*(x-waypoints(A-1,1));
end

end
```

Posteriormente se comprueba el funcionamiento del programa y corroborando que efectivamente el vehículo recorre la trayectoria deseada corregida mediante la utilización de la interpolación lineal, figura 3.7.

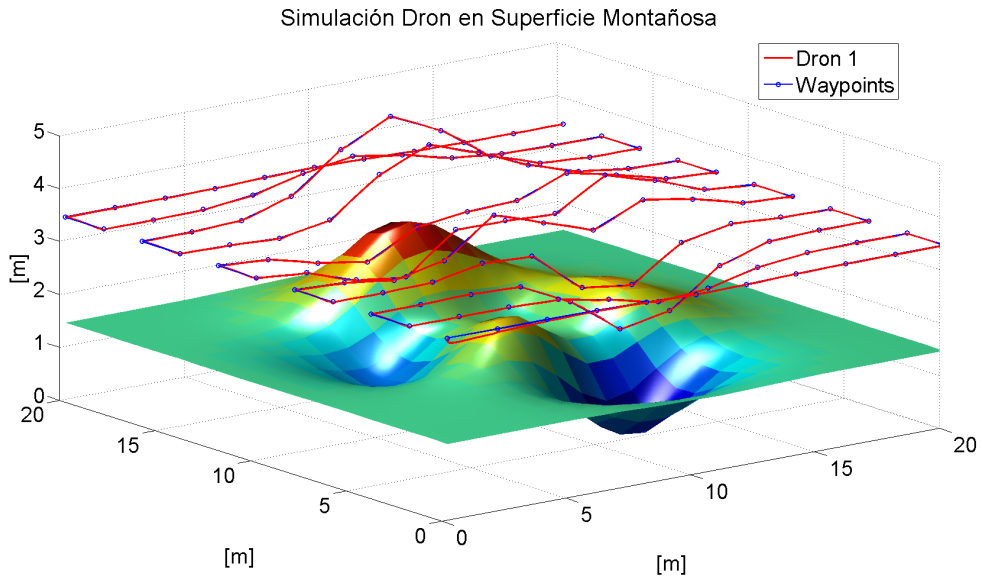

Figura 3.7 Modelo con interpolación lineal.

Dependiendo de la aplicación a realizar es posible que este tipo de trayectoria interpolada de manera lineal no sea suficiente para estar acorde con los objetivos de la misión. De modo similar a la interpolación lineal, es posible suavizar la trayectoria del dron haciendo uso de interpolación cuadrática. La interpolación cuadrática viene definida por la expresión (3.2):

$$z = b_0 + b_1\ (x - x_0) + b_2\ (x - x_0)\ (x - x_1) \tag{3.2}$$

$$b_0 = z(x_0)$$

$$\mathrm{b}_1 = \frac{z(x_1) - z(x_0)}{x_1 - x_0}$$

$$\mathrm{b}_2 = \frac{\dfrac{z(x_2) - z(x_1)}{x_2 - x_1} - \dfrac{z(x_1) - z(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

Se modifica el código de la función *Seguidor de Waypoints* de manera tal que cumpla con los requisitos que establece dicha interpolación, código 3.10. Este método requiere el conocimiento de tres puntos en lugar de dos por lo que las restricciones poseerán las restricciones correspondientes para garantizar el buen funcionamiento del programa en todas las situaciones.

Código 3.10 Seguidor Waypoints con interpolación cuadrática.

```
1  % ================================================================
2  % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3  % ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
4  % ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
5  % ================================================================
6  % ==========         FUNCIÓN SEGUIDOR DE WAYPOINTS         ==========
7  % ================================================================
8
```

```
function [ruta,distancia,a] =seguidorwaypoints(posicion_actual,...
                                                waypoints,destino_final)

x=posicion_actual(1); y=posicion_actual(2); z=posicion_actual(3);

waypoints=[waypoints;(destino_final)'];

[m,n]=size(waypoints); global A;

distancia=sqrt((waypoints(A,1)-x)^2+(waypoints(A,2)-y)^2+...
               (waypoints(A,3)-z)^2);

if distancia<0.2, A=A+1; end

if A>m, A=m; end

ruta=waypoints(A,:)'; a=length(waypoints)-A;

% Para suavizar las subidas y bajadas, interpolación cuadrática.

if A~=1 && A~=m && A~=m-1 && waypoints(A,1)-waypoints(A-1,1)~=0 && ...
waypoints(A+1,1)-waypoints(A,1)~=0 && waypoints(A+1,1)-waypoints(A-1,1)~=0

    b0=waypoints(A-1,3);
    b1=(waypoints(A,3)-waypoints(A-1,3))/(waypoints(A,1)-waypoints(A-1,1));
    b2=((waypoints(A+1,3)-waypoints(A,3))/(waypoints(A+1,1)-...
         waypoints(A,1))-b1)/(waypoints(A+1,1)-waypoints(A-1,1));
    ruta(3)=b0+b1*(x-waypoints(A-1,1))+b2*(x-waypoints(A-1,1))*...
            (x-waypoints(A,1));

end

end
```

El resultado con dicha interpolación se muestra en la figura 3.8:

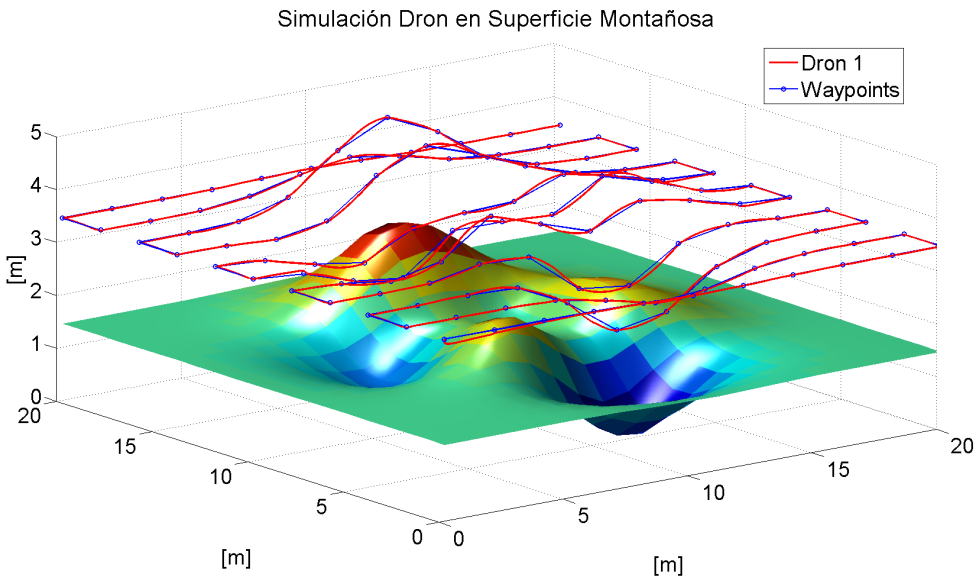

Figura 3.8 Modelo con interpolación cuadrática.

Análogamente al modelo con superficie plana, se realizan diversas simulaciones en condiciones diferentes y se anota el tiempo empleado por el dron en recorrer las trayectorias para así, posteriormente, compararlas con el uso de más de un vehículo para el barrido, tabla 3.2.

Tabla 3.2 Tiempo de barrido en superficie montañosa con un único dron.

Superficie [m]	Precisión [m]	Tiempo [s]
20 x 20	2 x 2	166.10
50 x 50	2 x 2	932.36
50 x 50	1 x 2	1640.21
30 x 30	1 x 1	1187.45

3.4 Problema del viajante para un dron

Una forma de barrido alternativa podría establecerse generando una serie de puntos de interés aleatorios y calculando posteriormente la ruta óptima por la que el dron tendría que circular. Éste problema es conocido como "Traveling Salesman Problem" en inglés o "Problema del Viajante" [1].

Existen múltiples algoritmos que resuelven el problema del viajante al tratarse de un problema complejo. Este proyecto se basará en el código de Joseph Kirk, *Traveling Salesman Problem - Nearest Neighbor* [2], ubicado en la plataforma *File Exchange* de MathWorks. Este algoritmo genera la ruta óptima de entre todas las posibles considerando la ruta óptima como la de menor distancia, código 3.11.

Las condiciones del algoritmo son:

- La ruta visita todos los puntos volviendo además al de partida.
- Cada punto es visitado una única vez.

A continuación, se muestra una descripción del funcionamiento del algoritmo:

1. Seleccionar un waypoint actual.
2. Encontrar el camino más corto que conecte el waypoint seleccionado con otro que no haya sido visitado.
3. Establecer el nuevo waypoint como waypoint actual.
4. Marcar el waypoint anterior como visitado.
5. Si todos los waypoints han sido visitados se termina el programa.
6. Si no, ir al paso número dos.

Código 3.11 Función para la resolución del Problema del Viajante.

```
%TSP_NN Traveling Salesman Problem (TSP) Nearest Neighbor (NN) Algorithm
%   The Nearest Neighbor algorithm produces different results depending on
%   which city is selected as the starting point. This function determines
%   the Nearest Neighbor routes for multiple starting points and returns
%   the best of those routes
%
% Summary:
%     1. A single salesman travels to each of the cities and completes the
%        route by returning to the city he started from
%     2. Each city is visited by the salesman exactly once
%
% Input:
%     USERCONFIG (structure) with zero or more of the following fields:
```

```
14 %     - XY (float) is an Nx2 matrix of city locations, where N is the number of ...
       cities
15 %     - DMAT (float) is an NxN matrix of point to point distances/costs
16 %     - POPSIZE (scalar integer) is the size of the population (should be <= N)
17 %     - SHOWPROG (scalar logical) shows the GA progress if true
18 %     - SHOWRESULT (scalar logical) shows the GA results if true
19 %     - SHOWWAITBAR (scalar logical) shows a waitbar if true
20 %
21 % Input Notes:
22 %     1. Rather than passing in a structure containing these fields, any/all of
23 %        these inputs can be passed in as parameter/value pairs in any order ...
       instead.
24 %     2. Field/parameter names are case insensitive but must match exactly ...
       otherwise.
25 %
26 % Output:
27 %     RESULTSTRUCT (structure) with the following fields:
28 %         (in addition to a record of the algorithm configuration)
29 %     - OPTROUTE (integer array) is the best route found by the algorithm
30 %     - MINDIST (scalar float) is the cost of the best route
31 %
32 % Usage:
33 %     tsp_nn
34 %       -or-
35 %     tsp_nn(userConfig)
36 %       -or-
37 %     resultStruct = tsp_nn;
38 %       -or-
39 %     resultStruct = tsp_nn(userConfig);
40 %       -or-
41 %     [...] = tsp_nn('Param1',Value1,'Param2',Value2, ...);
42 %
43 % Example:
44 %     % Let the function create an example problem to solve
45 %     tsp_nn;
46 %
47 % Example:
48 %     % Request the output structure from the solver
49 %     resultStruct = tsp_nn;
50 %
51 % Example:
52 %     % Pass a random set of user-defined XY points to the solver
53 %     userConfig = struct('xy',10*rand(50,2));
54 %     resultStruct = tsp_nn(userConfig);
55 %
56 % Example:
57 %     % Pass a more interesting set of XY points to the solver
58 %     n = 100;
59 %     phi = (sqrt(5)-1)/2;
60 %     theta = 2*pi*phi*(0:n-1);
61 %     rho = (1:n).^phi;
62 %     [x,y] = pol2cart(theta(:),rho(:));
63 %     xy = 10*([x y]-min([x;y]))/(max([x;y])-min([x;y]));
64 %     userConfig = struct('xy',xy);
65 %     resultStruct = tsp_nn(userConfig);
66 %
67 % Example:
68 %     % Pass a random set of 3D (XYZ) points to the solver
69 %     xyz = 10*rand(50,3);
70 %     userConfig = struct('xy',xyz);
71 %     resultStruct = tsp_nn(userConfig);
72 %
73 % Example:
74 %     % Turn off the plots but show a waitbar
75 %     userConfig = struct('showProg',false,'showResult',false,'showWaitbar',true)...
       ;
76 %     resultStruct = tsp_nn(userConfig);
77 %
```

```
78  % See also: tsp_ga, tspo_ga, tspof_ga, tspofs_ga, distmat
79  %
80  % Author: Joseph Kirk
81  % Email: jdkirk630@gmail.com
82  % Release: 2.0
83  % Release Date: 05/01/2014
84  function [waypoints, varargout] = tsp_nn(varargin)
85
86      % Initialize default configuration
87      defaultConfig.xy          = 10*rand(100,2);
88      defaultConfig.dmat        = [];
89      defaultConfig.popSize     = Inf;
90      defaultConfig.showProg    = true;
91      defaultConfig.showResult  = true;
92      defaultConfig.showWaitbar = false;
93
94      % Interpret user configuration inputs
95      if ~nargin
96          userConfig = struct();
97      elseif isstruct(varargin{1})
98          userConfig = varargin{1};
99      else
100         try
101             userConfig = struct(varargin{:});
102         catch
103             error('Expected inputs are either a structure or parameter/value ...
                    pairs');
104         end
105     end
106
107     % Override default configuration with user inputs
108     configStruct = get_config(defaultConfig,userConfig);
109
110     % Extract configuration
111     xy          = configStruct.xy;
112     dmat        = configStruct.dmat;
113     popSize     = configStruct.popSize;
114     showProg    = configStruct.showProg;
115     showResult  = configStruct.showResult;
116     showWaitbar = configStruct.showWaitbar;
117     if isempty(dmat)
118         nPoints = size(xy,1);
119         a = meshgrid(1:nPoints);
120         dmat = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),nPoints,nPoints);
121     end
122
123     % Verify Inputs
124     [N,dims] = size(xy);
125     [nr,nc] = size(dmat);
126     if N ~= nr || N ~= nc
127         error('Invalid XY or DMAT inputs!')
128     end
129     n = N;
130
131     % Sanity Checks
132     popSize     = max(1,min(n,round(real(popSize(1)))));
133     showProg    = logical(showProg(1));
134     showResult  = logical(showResult(1));
135     showWaitbar = logical(showWaitbar(1));
136
137     % Initialize the Population
138     pop = zeros(popSize,n);
139
140     % Run the NN
141     distHistory = zeros(1,popSize);
142     if showProg
143         figure('Name','TSP_NN | Current Solution','Numbertitle','off');
144         hAx = gca;
```

```
145     end
146     if showWaitbar
147         hWait = waitbar(0,'Searching for near-optimal solution ...');
148     end
149     for p = 1:popSize
150         d = 0;
151         thisRte = zeros(1,n);
152         visited = zeros(1,n);
153         I = p;
154         visited(I) = 1;
155         thisRte(1) = I;
156         for k = 2:n
157             dists = dmat(I,:);
158             dists(logical(visited)) = NaN;
159             dMin = min(dists(~visited));
160             J = find(dists == dMin,1);
161             visited(J) = 1;
162             thisRte(k) = J;
163             d = d + dmat(I,J);
164             I = J;
165         end
166         d = d + dmat(I,p);
167         pop(p,:) = thisRte;
168         distHistory(p) = d;
169 
170         if showProg
171             % Plot the Current Route
172             rte = thisRte([1:n 1]);
173             if dims > 2, plot3(hAx,xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
174             else plot(hAx,xy(rte,1),xy(rte,2),'r.-'); end
175             title(hAx,sprintf('Total Distance = %1.4f',distHistory(p)));
176             drawnow;
177         end
178 
179         % Update the waitbar
180         if showWaitbar && ~mod(p,ceil(popSize/325))
181             waitbar(p/popSize,hWait);
182         end
183 
184     end
185     if showWaitbar
186         close(hWait);
187     end
188 
189     % Find the Minimum Distance Route
190     [minDist,index] = min(distHistory);
191     optRoute = pop(index,:);
192 
193     if showResult
194         if showProg
195             % Plot the Best Route
196             rte = optRoute([1:n 1]);
197             if dims > 2, plot3(hAx,xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
198             else plot(hAx,xy(rte,1),xy(rte,2),'r.-'); end
199             title(hAx,sprintf('Total Distance = %1.4f',minDist));
200         end
201         % Plots the NN Results
202         figure('Name','TSP_NN | Results','Numbertitle','off');
203         subplot(2,2,1);
204         pclr = ~get(0,'DefaultAxesColor');
205         if dims > 2, plot3(xy(:,1),xy(:,2),xy(:,3),'.','Color',pclr);
206         else plot(xy(:,1),xy(:,2),'.','Color',pclr); end
207         title('City Locations');
208         subplot(2,2,2);
209         imagesc(dmat(optRoute,optRoute));
210         title('Distance Matrix');
211         subplot(2,2,3);
212         rte = optRoute([1:n 1]);
```

```
213         if dims > 2, plot3(xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
214         else plot(xy(rte,1),xy(rte,2),'r.-'); end
215         waypoints = [xy(rte,1),xy(rte,2),xy(rte,3)]; %%%%%%%%%%%%
216         title(sprintf('Total Distance = %1.4f',minDist));
217         subplot(2,2,4);
218         plot(sort(distHistory,'descend'),'b','LineWidth',2);
219         title('Distances');
220         set(gca,'XLim',[0 popSize+1],'YLim',[0 1.1*max([1 distHistory])]);
221     end
222
223     % Return Output
224     if nargout
225         resultStruct = struct( ...
226             'xy',          xy, ...
227             'dmat',        dmat, ...
228             'popSize',     popSize, ...
229             'showProg',    showProg, ...
230             'showResult',  showResult, ...
231             'showWaitbar', showWaitbar, ...
232             'optRoute',    optRoute, ...
233             'minDist',     minDist, ...
234             'pop',         pop);
235
236         varargout = {resultStruct};
237     end
238
239 end
240 % Subfunction to override the default configuration with user inputs
241 function config = get_config(defaultConfig,userConfig)
242
243     % Initialize the configuration structure as the default
244     config = defaultConfig;
245
246     % Extract the field names of the default configuration structure
247     defaultFields = fieldnames(defaultConfig);
248
249     % Extract the field names of the user configuration structure
250     userFields = fieldnames(userConfig);
251     nUserFields = length(userFields);
252
253     % Override any default configuration fields with user values
254     for i = 1:nUserFields
255         userField = userFields{i};
256         isField = strcmpi(defaultFields,userField);
257         if nnz(isField) == 1
258             thisField = defaultFields{isField};
259             config.(thisField) = userConfig.(userField);
260         end
261     end
262
263 end
```

En todo el desarrollo anterior del proyecto la distancia entre waypoints no superaba los 2.5 *metros* de media. Esto otorgaba robustez y seguridad en el barrido. Empleando este sistema no se garantiza el cumplimiento de estas distancias por lo que será necesaria la creación de una nueva función que, si la distancia entre waypoints supera un cierto valor, genere waypoints intermedios disminuyendo así esta distancia, esta función se denomina *Adiccion de waypoints*, código 3.12, que genera una matriz de waypoints corregida.

Código 3.12 Función adicción de waypoints.

```
1 % ================================================================
2 % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3 % ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
4 % ==========   TRABAJO DE FIN DE GRADO, SEVILLA 2019       ==========
```

```
% ================================================================
% ==========       FUNCIÓN ADICCIÓN DE WAYPOINTS          ==========
% ================================================================

function waypoints_out = adiccion_waypoints(waypoints_in)
e=0; i=1; [m,~]=size(waypoints_in); waypoints_out = waypoints_in;
while e==0

    if i==m, break, end

    x0=waypoints_out(i,1); y0=waypoints_out(i,2); z0=waypoints_out(i,3);
    x1=waypoints_out(i+1,1); y1=waypoints_out(i+1,2); z1=waypoints_out(i+1,3);
    d = sqrt((x0-x1)^2+(y0-y1)^2+(z0-z1)^2);

    if d>2.5
       waypoints_alpha=[waypoints_out(1:i,:); ...
       (x1+x0)/2 (y1+y0)/2 (z1+z0)/2; waypoints_out(i+1:m,:)];
       waypoints_out=waypoints_alpha;
    else i=i+1;
    end
    [m,~]=size(waypoints_out);
end
```

Posteriormente se escribe el código 3.13 que contempla todo lo explicado con anterioridad generando las figuras 3.9, 3.10, 3.11, 3.12 y 3.13, donde se muestran los resultados.

Código 3.13 Simulación problema del viajante para un dron.

```
% ================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
% ==========   TRABAJO DE FIN DE GRADO, SEVILLA 2019       ==========
% ================================================================
% ==========    MODELO UN DRON PROBLEMA DEL VIAJANTE       ==========
% ================================================================

superficie = [20 20]; dimx=superficie(1); dimy=superficie(2);
x=0:dimx; y=0:dimy; z=zeros(length(x),length(y));

mtxpuntos = [20*rand(50,2), 2+rand(50,1)];

userConfig = struct('xy',mtxpuntos);
[waypoints, resultStruct_1] = tsp_nn(userConfig);
waypoints_s = waypoints;
waypoints = adiccion_waypoints(waypoints); posini1 = waypoints(1,:);

destino_final = [waypoints(length(waypoints),:)];

sim('quad_control')

figure(1)
plot3(mtxpuntos(:,1),mtxpuntos(:,2),mtxpuntos(:,3),'*','LineWidth',2);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Generación de Puntos Aleatorios','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Puntos Aleatorios');

```

```
figure(2)
plot3(waypoints_s(:,1),waypoints_s(:,2),waypoints_s(:,3),'o-','LineWidth',2);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Trayectoria Problema del Viajante','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Waypoints');

figure(3)
plot3(waypoints(:,1),waypoints(:,2),waypoints(:,3),'o-','LineWidth',2);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Trayectoria Problema del Viajante Corregida','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Waypoints');

figure(4)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(waypoints(:,1),waypoints(:,2),waypoints(:,3),'o-','LineWidth',1.25);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Simulación Dron Problema del Viajante','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Waypoints');
```

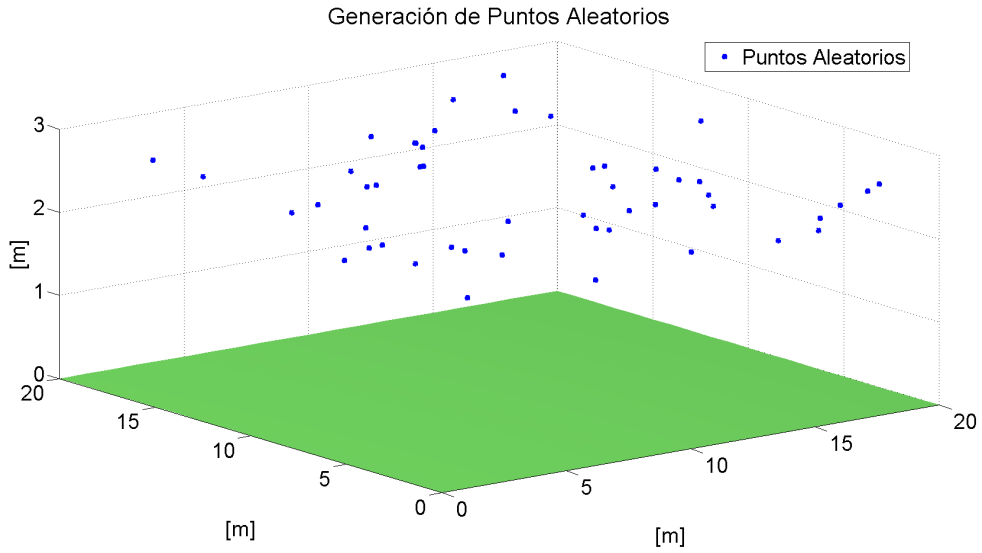

Figura 3.9 Generación de puntos aleatorios.

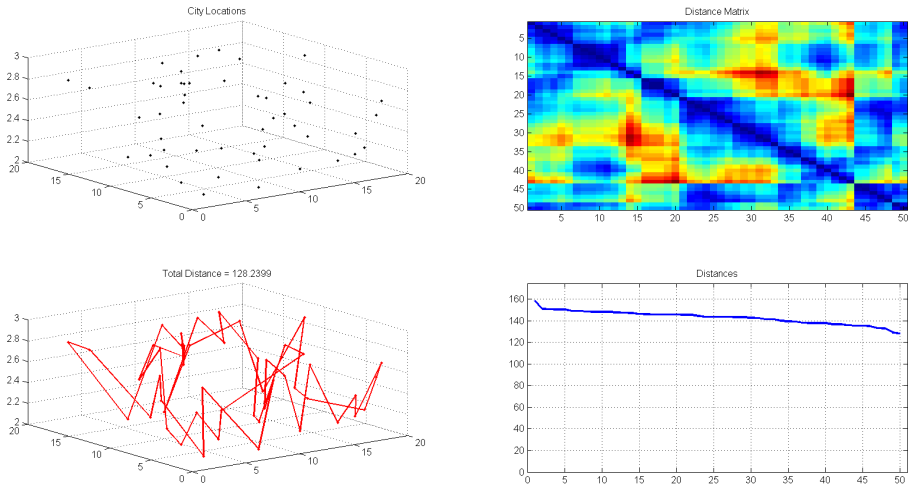

Figura 3.10 Solución de la función "tsp_nn.m".

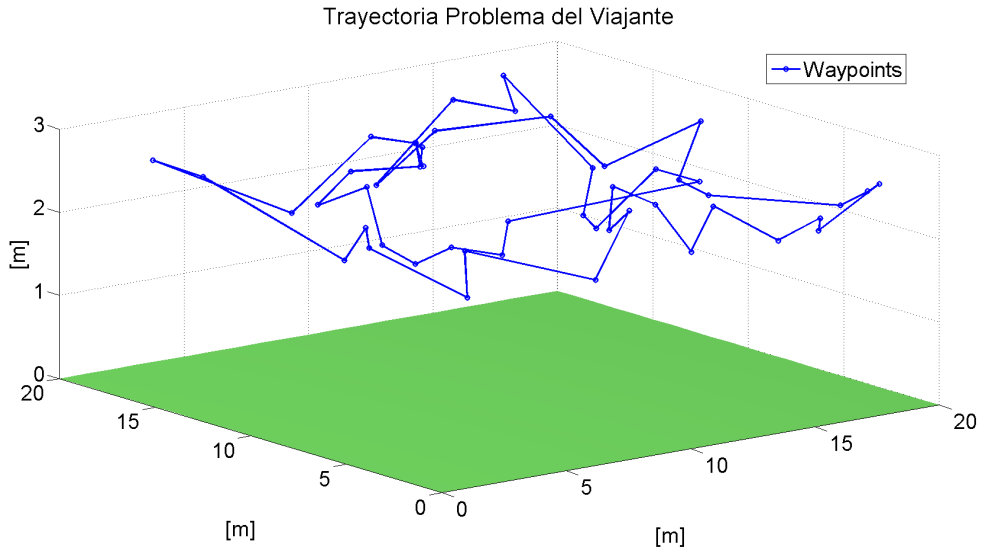

Figura 3.11 Waypoints de la solución al problema del viajante para un dron.

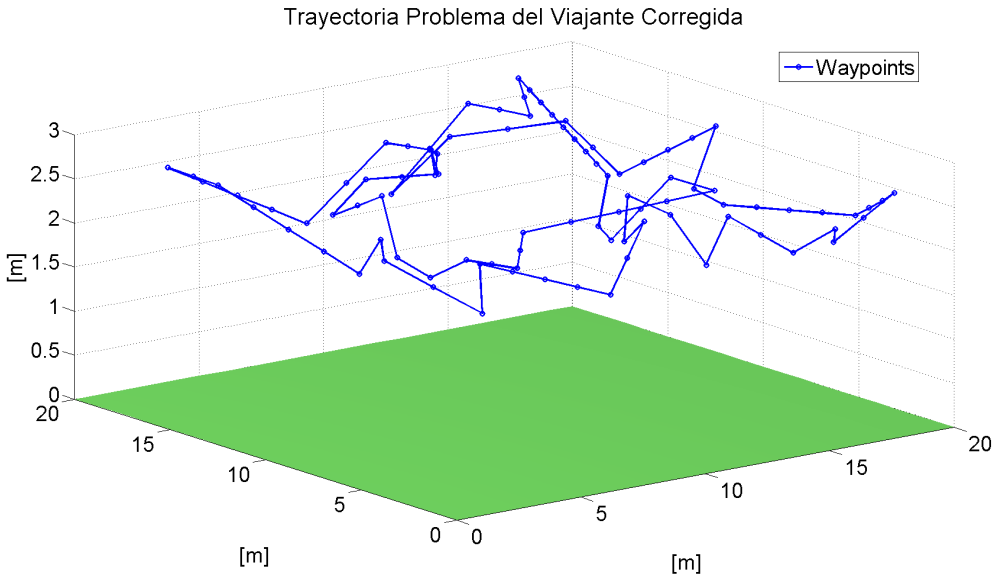

Figura 3.12 Waypoints corregidos de la solución al problema del viajante para un dron.

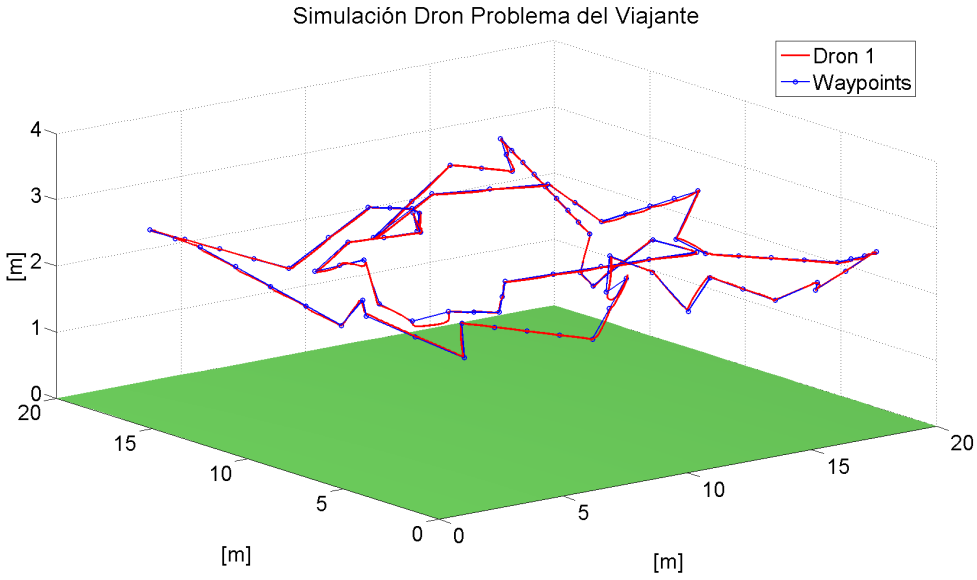

Figura 3.13 Simulación del problema del viajante para un dron.

Una vez concluido todo el desarrollo del modelo con un único dron, en la figura 3.14, se muestra el cómo está implementado con SimuLink.

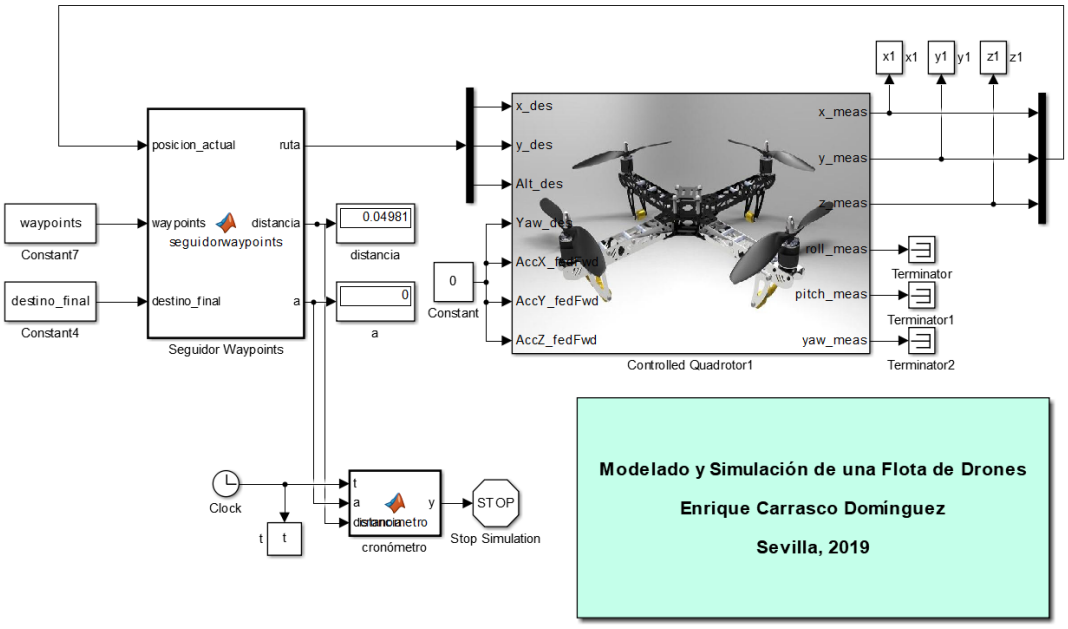

Figura 3.14 Modelo con interpolación lineal.

4 Algoritmos para dos drones

El objetivo principal del proyecto es el modelado y simulación de una flota de drones, esto implica la presencia de más de un dron. Una vez completado el desarrollo de todo el contenido necesario para el correcto funcionamiento de un vehículo en solitario, es hora de extrapolar el contenido para el control de una flota de dos vehículos aéreos. Se ha decidido primero desarrollar el trabajo con dos vehículos para que, una vez completado y comprobado el correcto funcionamiento de los programas, sea más sencilla la modificación de éstos para controlar una flota mayor.

Añadir un nuevo vehículo al problema es sencillo dado que la herramienta con la que se está trabajando permite llevar a cabo este proceso simplemente duplicando el bloque del primer dron. Internamente se cambian los datos de posición inicial en los ejes x, y, z añadiendo el vector *posini_2* para el segundo vehículo. Además, se guardará en el espacio de trabajo de MATLAB su posición que viene definida por los vectores x_2, y_2, z_2.

4.1 Control de trayectorias para dos drones

Para llevar a cabo el control de las trayectorias de ambos drones simultáneamente, se debe modificar el bloque *Seguidor Waypoints*, código 4.1. Éste tendrá como entradas las posiciones de ambos vehículos, así como las matrices de waypoints correspondientes y la posición final en la que cada uno de ellos permanecerá una vez finalizados los recorridos. Las salidas de la función serán las rutas de cada uno de los vehículos, las distancias entre la posición actual y el waypoint objetivo, además del número de waypoints restantes para finalizar el recorrido. Para suavizar la trayectoria se ha decidido utilizar el método de interpolación lineal.

Es necesario, de un modo similar a como se realizó para el control de un único dron, establecer una nueva variable global B que funcione de manera análoga a como lo hacía la variable global A. Se ha decidido añadir una nueva variable para que cada vehículo por separado pueda ir recorriendo su trayectoria sin que el estado del otro afecte de ningún modo.

Código 4.1 Seguidor de Waypoints para dos drones.

```
% ================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========           ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019     =========
% ================================================================
% ========= FUNCIÓN SEGUIDOR DE WAYPOINTS PARA DOS DRONES  =========
% ================================================================

function [ruta_1,ruta_2,distancia_1,distancia_2,a,b] = ...
          seguidorwaypoints_multiuav(posiciones_actuales,waypoints_1,...
          waypoints_2,destino_final_1,destino_final_2)
```

```
12
13 posicion_actual_1 = posiciones_actuales(1:3);
14 posicion_actual_2 = posiciones_actuales(4:6);
15
16 x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
17 x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);
18
19 waypoints_1=[waypoints_1;(destino_final_1)'];
20 waypoints_2=[waypoints_2;(destino_final_2)'];
21
22 [m1,~]=size(waypoints_1); [m2,~]=size(waypoints_2);
23
24 global A, global B
25
26 distancia_1=sqrt((waypoints_1(A,1)-x1)^2+(waypoints_1(A,2)-y1)^2+...
27 (waypoints_1(A,3)-z1)^2);
28 distancia_2=sqrt((waypoints_2(B,1)-x2)^2+(waypoints_2(B,2)-y2)^2+...
29 (waypoints_2(B,3)-z2)^2);
30
31 if distancia_1<0.2, A=A+1; end
32 if A>m1, A=m1; end
33
34 if distancia_2<0.2, B=B+1; end
35 if B>m2, B=m2; end
36
37 ruta_1=waypoints_1(A,:)'; ruta_2=waypoints_2(B,:)';
38 a=length(waypoints_1)-A; b=length(waypoints_2)-B;
39
40 % Para suavizar las subidas y bajadas, interpolación lineal.
41
42 if A~=1 && A~=m1 && waypoints_1(A,1)-waypoints_1(A-1,1)~=0
43     ruta_1(3) = waypoints_1(A-1,3)+((waypoints_1(A,3)-waypoints_1(A-1,3))/...
44               (waypoints_1(A,1)-waypoints_1(A-1,1)))*(x1-waypoints_1(A-1,1));
45 end
46
47 if B~=1 && B~=m2 && waypoints_2(B,1)-waypoints_2(B-1,1)~=0
48     ruta_2(3) = waypoints_2(B-1,3)+((waypoints_2(B,3)-waypoints_2(B-1,3))/...
49               (waypoints_2(B,1)-waypoints_2(B-1,1)))*(x2-waypoints_2(B-1,1));
50 end
```

Adicionalmente, se actualiza la función *Cronometro*, código 4.2, para que detenga la simulación cuando ambos vehículos hayan completado la trayectoria. Este proceso se realiza de forma sencilla añadiendo las variables del segundo vehículo.

Código 4.2 Cronómetro para dos drones.

```
1  % ==================================================================
2  % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3  % ==========          ENRIQUE CARRASCO DOMÍNGUEZ          ==========
4  % ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019    ==========
5  % ==================================================================
6  % ==========              FUNCIÓN CRONÓMETRO              ==========
7  % ==================================================================
8
9  function y  = cronometro(t,a,b,distancia_1,distancia_2)
10 if t>0 && a+b==0 && distancia_1<0.05 && distancia_2<0.05
11     y=1;
12 else y=0;
13 end
```

Una vez modificada la función se comprueba su correcto funcionamiento, figura 4.1, en un entorno en el que no sea posible la colisión entre ambos. Para ello, se establecen las posiciones iniciales de los drones, así como la serie de waypoints que definen diferentes trayectorias.

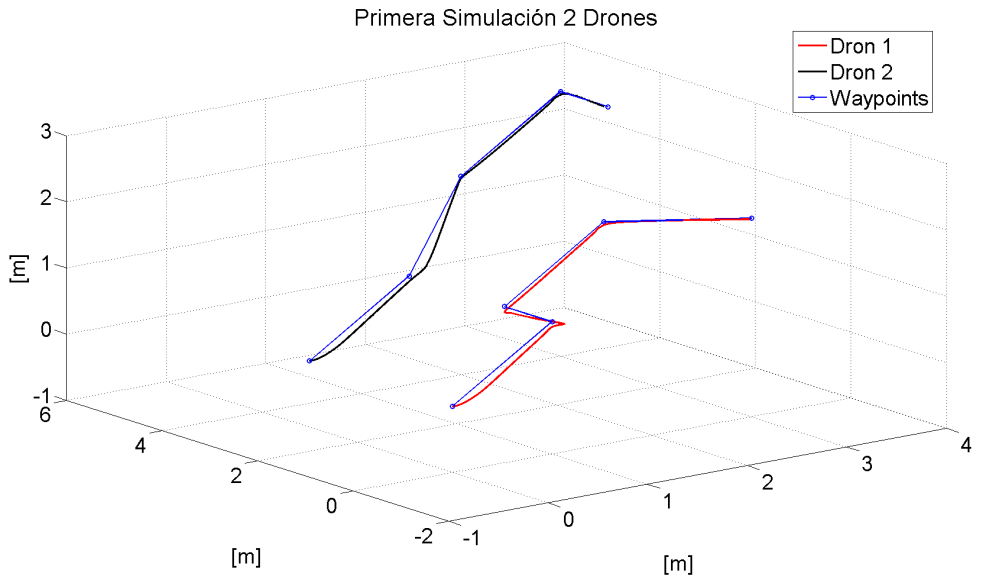

Figura 4.1 Primera simulación dos drones.

Se observa un funcionamiento correcto, cada uno de ellos circula por su trayectoria y el sistema es todo un éxito.

4.2 Evasión de colisión

Con la introducción de un nuevo dron sobre el terreno de aplicación, el primer problema que se encuentra es el de una posible colisión entre ellos. Desde el punto de vista de una futura aplicación real del proyecto, supondría pérdidas económicas derivadas tanto de la reparación o reemplazo de los vehículos y/o de sus cargas de pago como de los retrasos ocasionados.

Las trayectorias que se definirán tendrán en cuenta este factor por lo que el riesgo de colisión se reduce al mínimo. No obstante, dado que una colisión sería una amenaza importante para el proyecto, se desarrolla la función *Sistema Evasion de Colision*, código 4.3.

Esta función tiene como entrada las posiciones de los vehículos, además de las rutas establecidas para cada uno de ellos. El objetivo es detectar la cercanía entre los dos drones y realizar una corrección en la trayectoria, ya sea únicamente en uno de ellos o en ambos, para así incrementar la distancia de seguridad y evitar una colisión. Tras una serie de pruebas en diversas condiciones se decide que sea solo un vehículo el que modifique su trayectoria evitando así la colisión. El vehículo número uno será el encargado de modificar su trayectoria y lo hará elevando su altura en un metro si se detecta que la distancia entre los dos vehículos es inferior a 1,5 *metros* y se encuentra a una altura igual o superior a la del dron que tendría que esquivar. Si se diera el caso opuesto, es decir, el dron que reacciona se encuentra a una altura inferior, éste descendería medio metro para así evitar la colisión.

Código 4.3 Sistema de evasión de colisión.

```
% ======================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========            ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019      =========
% ======================================================================
% =========           SISTEMA EVASIÓN DE COLISIÓN           =========
% ======================================================================
```

```
8
9  function [ruta_corregida_1,ruta_corregida_2] = ...
10           evasion_colision(posiciones_actuales,ruta_1,ruta_2)
11
12 posicion_actual_1 = posiciones_actuales(1:3);
13 posicion_actual_2 = posiciones_actuales(4:6);
14
15 x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
16 x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);
17
18 distancia_entre_drones = abs(sqrt((x1-x2)^2+(y1-y2)^2+(z1-z2)^2));
19
20 ruta_corregida_1 = ruta_1; ruta_corregida_2 = ruta_2;
21
22 if distancia_entre_drones<1.5
23     if z2<=z1
24         ruta_corregida_1(3) = ruta_corregida_1(3)+1;
25     else ruta_corregida_1(3) = ruta_corregida_1(3)-0.5;
26     end
27 end
28 end
```

Se comprueba el funcionamiento del programa en diversas situaciones, código 4.4, comprobando que el comportamiento es el adecuado evitando siempre la colisión entre los vehículos, figuras 4.2, 4.3, 4.4 y 4.5.

Código 4.4 Prueba de evasión de colisión.

```
1  % ======================================================================
2  % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3  % ==========           ENRIQUE CARRASCO DOMÍNGUEZ           ==========
4  % ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
5  % ======================================================================
6  % ==========    MODELO DOS DRONES EVASIÓN DE COLISIÓN     ==========
7  % ======================================================================
8
9  punto_objetivo = [0 0 0];
10
11 % Ruta de colisión frontal
12
13 superficie = [6 2];
14 dimx = superficie(1); dimy = superficie(2);
15 x = 0:dimx; y = 0:dimy; z = zeros(length(x),length(y));
16
17 posini_1 = [0 1 2]; posini_2 = [6 1 2];
18
19 waypoints_1 = [0 1 2; 1 1 2; 2 1 2; 3 1 2; 4 1 2; 5 1 2; 6 1 2];
20 waypoints_2 = [6 1 2; 5 1 2; 4 1 2; 3 1 2; 2 1 2; 1 1 2; 0 1 2];
21
22 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
23 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
24
25 sim('quad_control_2drones')
26
27 figure(1)
28 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
29 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
30 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
31 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
32 'LineWidth',1.25); hold on;
33 plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
34 'LineWidth',1.25);
35
36 camlight left; grid on;
37 lighting phong
38
```

```
39 title('Simulación 2 Drones: Evasión de Colisión Frontal','FontSize', 24);
40 xlabel('[m]','FontSize', 22);
41 ylabel('[m]','FontSize', 22);
42 zlabel('[m]','FontSize', 22);
43 set(gca, 'FontSize', 22);
44 xlim([0 superficie(1)]);
45 ylim([0 superficie(2)]);
46 legend('Dron 1','Dron 2');
47
48
49 % Ruta de colisión a 45
50
51 superficie = [5 5];
52 dimx = superficie(1); dimy = superficie(2);
53 x = 0:dimx; y = 0:dimy; z = zeros(length(x),length(y));
54
55 posini_1 = [0 0 2]; posini_2 = [0 5 2];
56
57 waypoints_1 = [0 0 2; 1 1 2; 2 2 2; 3 3 2; 4 4 2; 5 5 2];
58 waypoints_2 = [0 5 2; 1 4 2; 2 3 2; 3 2 2; 4 1 2; 5 0 2];
59
60 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
61 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
62
63 sim('quad_control_2drones')
64
65 figure(2)
66 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
67 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
68 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
69 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
70 'LineWidth',1.25); hold on;
71 plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
72 'LineWidth',1.25);
73
74 camlight left; grid on;
75 lighting phong
76
77 title('Simulación 2 Drones: Evasión de Colisión a 45  ','FontSize', 24);
78 xlabel('[m]','FontSize', 22);
79 ylabel('[m]','FontSize', 22);
80 zlabel('[m]','FontSize', 22);
81 set(gca, 'FontSize', 22);
82 xlim([0 superficie(1)]);
83 ylim([0 superficie(2)]);
84 legend('Dron 1','Dron 2');
85
86 % Ruta de colisión en paralelo
87
88 superficie = [4 5];
89 dimx = superficie(1); dimy = superficie(2);
90 x = 0:dimx; y = 0:dimy; z = zeros(length(x),length(y));
91
92 posini_1 = [0 0 2]; posini_2 = [4 0 2];
93
94 waypoints_1 = [0 0 2; 1 1 2; 2 2 2; 2 3 2; 1 4 2; 0 5 2];
95 waypoints_2 = [4 0 2; 3 1 2; 2 2 2; 2 3 2; 3 4 2; 4 5 2];
96
97 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
98 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
99
100 sim('quad_control_2drones')
101
102 figure(3)
103 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
104 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
105 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
106 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
```

```
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Simulación 2 Drones: Evasión de Colisión en Paralelo',...
'FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2');

% Ruta de colisión a 45   altura inferior

superficie = [5 5];
dimx = superficie(1); dimy = superficie(2);
x = 0:dimx; y = 0:dimy; z = zeros(length(x),length(y));

posini_1 = [0 0 2]; posini_2 = [0 5 2];

waypoints_1 = [0 0 2; 1 1 2; 2 2 2; 3 3 2; 4 4 2; 5 5 2];
waypoints_2 = [0 5 2; 1 4 2; 2 3 2; 3 2 2; 4 1 2; 5 0 2];
waypoints_2(:,3)=waypoints_2(:,3)+0.5; posini_2(3)=posini_2(3)+0.5;

destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];

sim('quad_control_2drones')

figure(4)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Simulación 2 Drones: Evasión de Colisión a 45   Altura Inferior',...
'FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2');
```

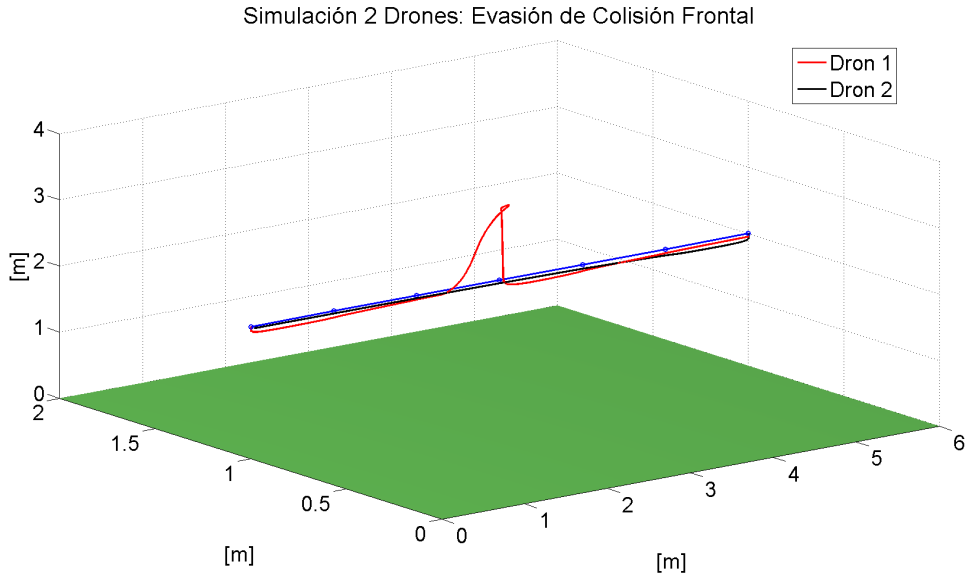

Figura 4.2 Simulación evasión de colisión frontal.

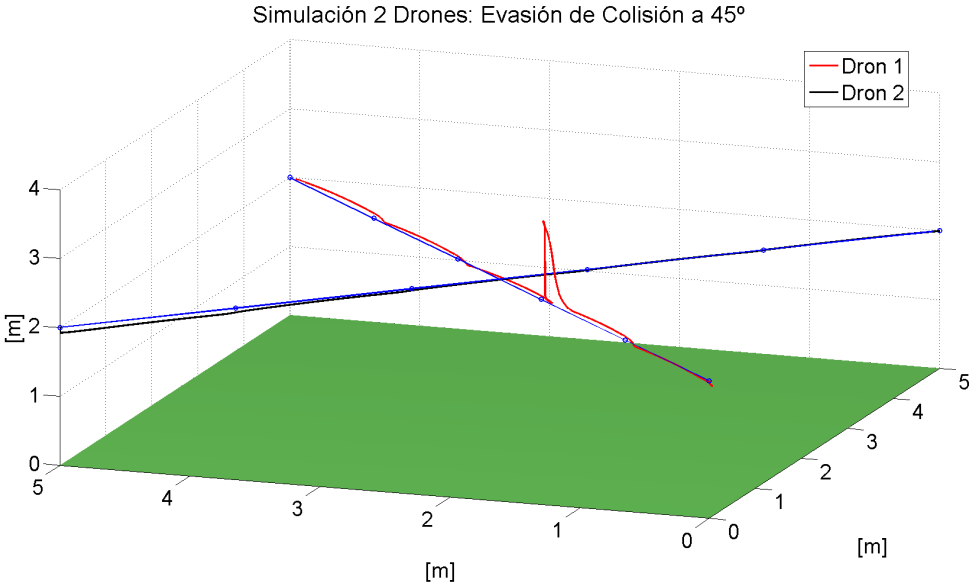

Figura 4.3 Simulación evasión de colisión a 45º.

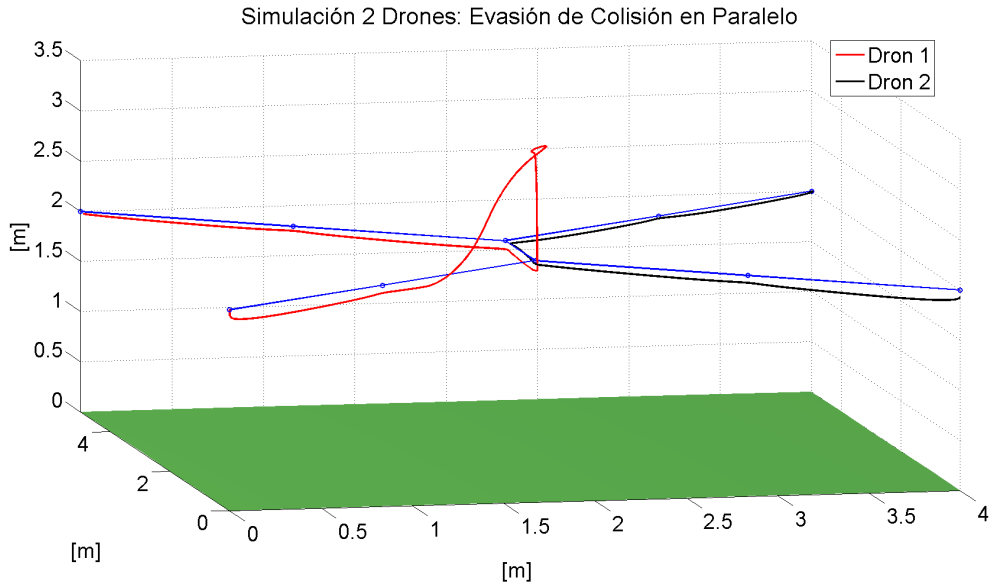

Figura 4.4 Simulación evasión de colisión en paralelo.

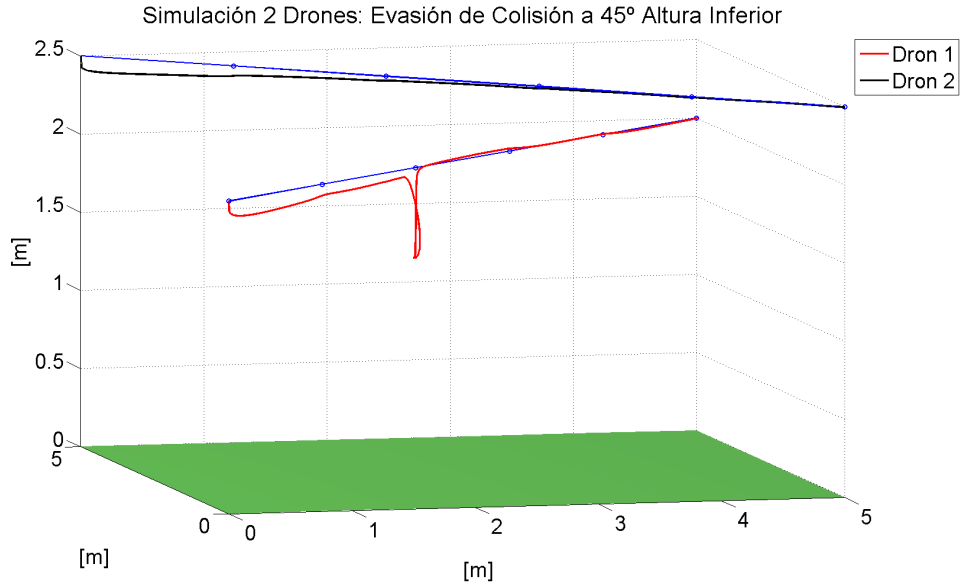

Figura 4.5 Simulación evasión de colisión a 45º para altura inferior.

4.3 División del barrido

La división del barrido se llevará a cabo en las funciones de generación de los waypoints de las trayectorias. Dado que se dispone de dos drones, el modo más intuitivo de hacer la división será separar el terreno en dos partes iguales. Se ha decidido realizar esta división según el eje y. Como base se toman las funciones de generación de waypoints para un único dron y se realizan cambios en la generación de la matriz de waypoints, siendo ahora dos, la primera barrerá los puntos según el eje y desde cero hasta $y/2$ y la segunda comenzará en $y/2+1$ hasta la longitud máxima del terreno según el eje y. Si la decena es par habría que sumar una posición extra recorrido, para ello, se ha decidido que sea el primer dron el que realice esta tarea teniendo un recorrido más largo.

A continuación, se muestran las funciones de generación de waypoints modificadas tanto para una superficie plana como para una montañosa, códigos 4.5 y 4.6.

Código 4.5 Generación de waypoints en superficie plana para dos drones.

```
% ====================================================================
% ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
% ==========           ENRIQUE CARRASCO DOMÍNGUEZ          ==========
% ==========     TRABAJO DE FIN DE GRADO, SEVILLA 2019     ==========
% ====================================================================
% ========== GENERADOR DE WAYPOINTS PLANO PARA DOS DRONES ==========
% ====================================================================

function [waypoints_1,waypoints_2]=generador_waypoints_plano_2drones...
         (superficie,altura_barrido,precision)

dimx=superficie(1); dimy=superficie(2);

precisionx=precision(1); precisiony=precision(2);

puntosx=[0:precisionx:dimx]';
puntosxinv=[dimx:-precisionx:0]';
puntosy=[0:precisiony:dimy]';

% CÁLCULO WAYPOINTS PARA EL PRMIER DRON

p=1-mod(0.1*dimy,2); % Si la decena es par hay que sumar 1.

puntosx_1=puntosx;
puntosxinv_1=puntosxinv;
puntosy_1=puntosy(1:length(puntosy)/2+p);
alpha_1=ones(size(puntosx_1));
e=1; wp_1=[];

for i=1:length(puntosy_1)
    wp_1=[wp_1; abs(((e+1)/2)*puntosx_1+((e-1)/2)*puntosxinv_1) ,...
        alpha_1*puntosy_1(i) ];
    e=-e;
end

wp_1=[wp_1, ones(length(wp_1),1)*altura_barrido];
waypoints_1=wp_1;

% CÁLCULO WAYPOINTS PARA EL SEGUNDO

puntosx_2=puntosx;
puntosxinv_2=puntosxinv;
puntosy_2=puntosy(length(puntosy)/2+1:length(puntosy));
alpha_2=ones(size(puntosx_2));
e=1; wp_2=[];

for i=1:length(puntosy_2)
```

```
48      wp_2=[wp_2; abs(((e+1)/2)*puntosx_2+((e-1)/2)*puntosxinv_2) ,...
49          alpha_2*puntosy_2(i) ];
50      e=-e;
51  end
52
53  wp_2=[wp_2, ones(length(wp_2),1)*altura_barrido];
54  waypoints_2=wp_2;
55
56  end
```

Código 4.6 Generación de waypoints en superficie montañosa para dos drones.

```
1   % ====================================================================
2   % =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
3   % =========            ENRIQUE CARRASCO DOMÍNGUEZ           =========
4   % =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019      =========
5   % ====================================================================
6   % ========= GENERADOR DE WAYPOINTS MONTAÑA PARA DOS DRONES =========
7   % ====================================================================
8
9   function [waypoints_1,waypoints_2] = generador_waypoints_montana_2drones...
10          (superficie,z,altura_barrido,precision)
11
12  dimx=superficie(1); dimy=superficie(2);
13
14  precisionx=precision(1); precisiony=precision(2);
15
16  puntosx=[0:precisionx:dimx]';
17  puntosxinv=[dimx:-precisionx:0]';
18  puntosy=[0:precisiony:dimy]';
19
20  % CÁLCULO WAYPOINTS PARA EL PRMIER DRON
21
22  p=1-mod(0.1*dimy,2); % Si la decena es par hay que sumar 1.
23
24  puntosx_1=puntosx;
25  puntosxinv_1=puntosxinv;
26  puntosy_1=puntosy(1:length(puntosy)/2+p);
27  alpha_1=ones(size(puntosx_1));
28  e=1; wp_1=[];
29
30  for i=1:length(puntosy_1)
31      wp_1=[wp_1; abs(((e+1)/2)*puntosx_1+((e-1)/2)*puntosxinv_1) ,...
32          alpha_1*puntosy_1(i) ];
33      e=-e;
34  end
35
36  wp_1=[wp_1,zeros(length(wp_1),1)];
37
38  for i=1:length(wp_1)
39      wp_1(i,3)=z(wp_1(i,2)+1,wp_1(i(1))+1)+altura_barrido-1;
40  end
41
42  wp_1=wp_1+ones(size(wp_1));
43  waypoints_1=wp_1;
44
45  % CÁLCULO WAYPOINTS PARA EL SEGUNDO
46
47  puntosx_2=puntosx;
48  puntosxinv_2=puntosxinv;
49  puntosy_2=puntosy(length(puntosy)/2+1:length(puntosy));
50  alpha_2=ones(size(puntosx_2));
51  e=1; wp_2=[];
52
53  for i=1:length(puntosy_2)
54      wp_2=[wp_2; abs(((e+1)/2)*puntosx_2+((e-1)/2)*puntosxinv_2) ,...
```

```
55          alpha_2*puntosy_2(i) ];
56      e=-e;
57  end
58
59  wp_2=[wp_2,zeros(length(wp_2),1)];
60
61  for i=1:length(wp_2)
62      wp_2(i,3)=z(wp_2(i,2)+1,wp_2(i(1))+1)+altura_barrido-1;
63  end
64
65  wp_2=wp_2+ones(size(wp_2));
66  waypoints_2=wp_2;
67
68  end
```

Para comprobar el correcto funcionamiento de ambas funciones, se generarán diversas superficies junto con los waypoints de las trayectorias generados, figuras 4.6 y 4.7.

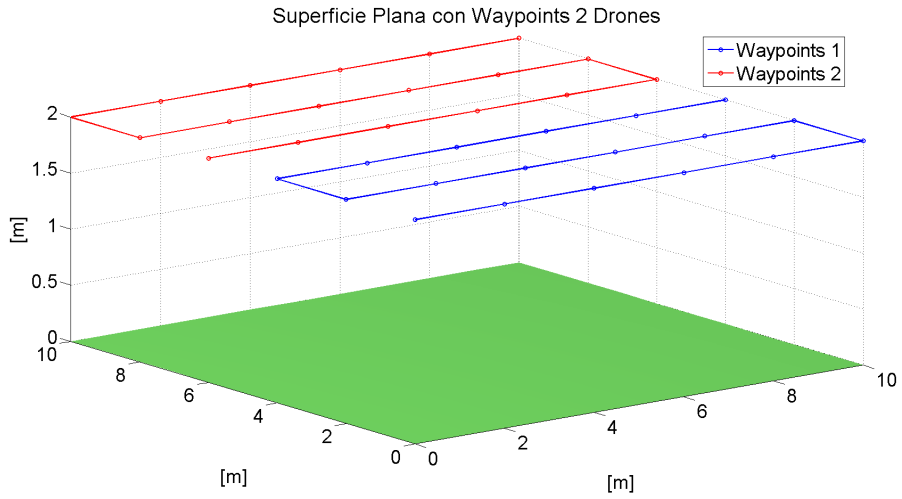

Figura 4.6 Superficie plana con waypoints para dos drones.

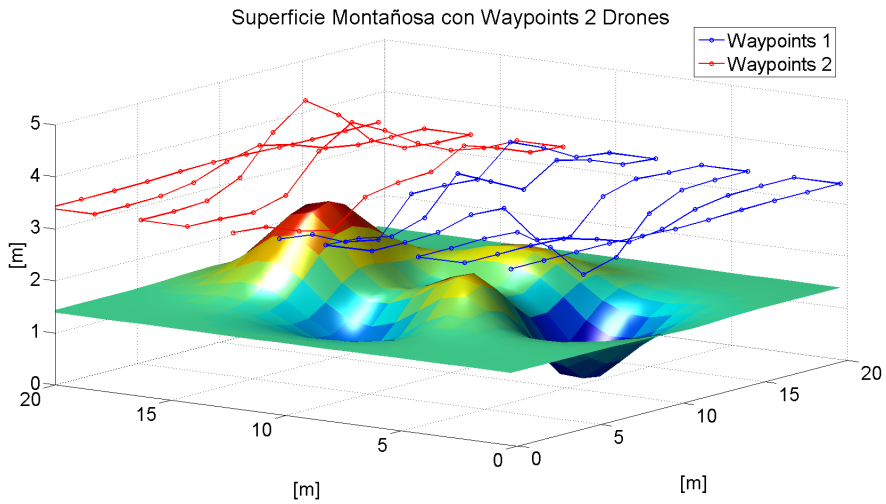

Figura 4.7 Superficie montañosa con waypoints para dos drones.

4.4 Barrido del terreno en una superficie plana

Una vez realizado el desarrollo anterior se simula el modelo de dos drones en un entorno definido por una superficie plana, figura 4.8. Al igual que con el modelo de un único dron, se realizan varias simulaciones anotando el tiempo transcurrido en el barrido del terreno para demostrar lo que era esperado, el tiempo en barrer la superficie es aproximadamente la mitad al hacer uso de dos drones para el barrido, tabla 4.1.

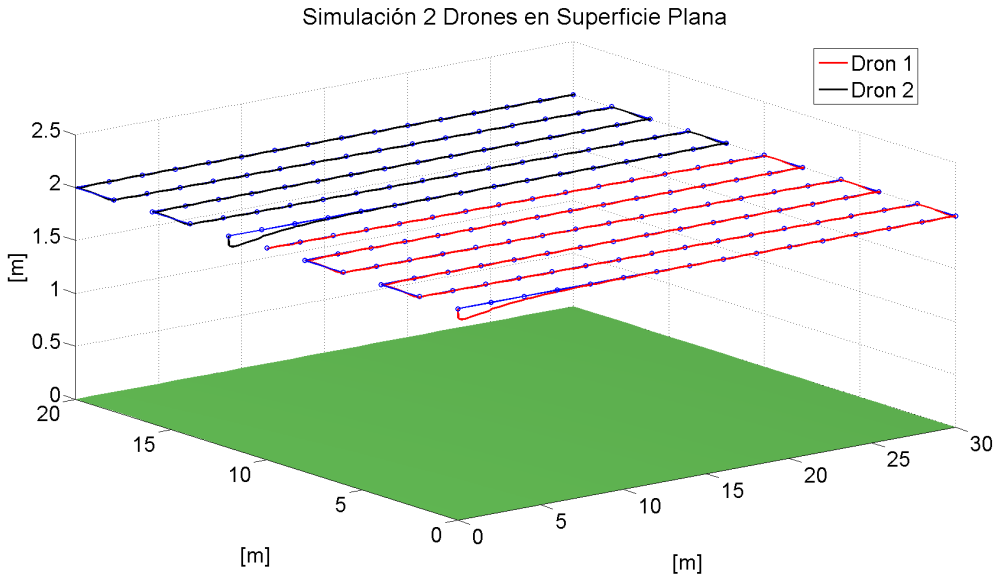

Figura 4.8 Simulación dos drones superficie plana.

Tabla 4.1 Tiempo de barrido en superficie plana con dos drones.

Superficie [m]	Precisión [m]	Tiempo [s]	Reducción de tiempo
20 x 20	2 x 2	89.99	45.8 %
50 x 50	2 x 2	465.55	50.06 %
50 x 50	1 x 2	817.14	50.03 %
40 x 20	1 x 1	554.55	47.66 %

4.5 Barrido del terreno en una superficie montañosa

De forma análoga al apartado anterior, se simula el modelo de dos drones cambiando el tipo de superficie por una con relieve, 4.9. Se toman las medidas pertinentes demostrando así que el tiempo de barrido es aproximadamente la mitad que el de un solo dron en las con el mismo tipo de superficie, 4.2.

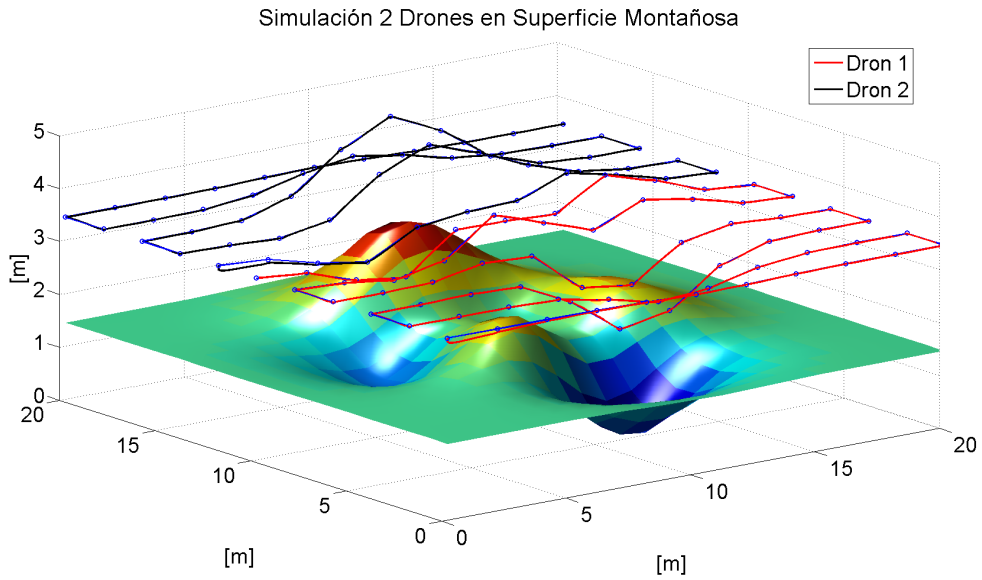

Figura 4.9 Simulación dos drones superficie montañosa.

Tabla 4.2 Tiempo de barrido en superficie montañosa con dos drones.

Superficie [m]	Precisión [m]	Tiempo [s]	Reducción de tiempo
20 x 20	2 x 2	90.05	45.79 %
50 x 50	2 x 2	465.95	50.02 %
50 x 50	1 x 2	820.59	49.97 %
30 x 30	1 x 2	574.66	51.61 %

4.6 Elección de dron para un punto

En algunas aplicaciones, el hecho de realizar un barrido del terreno no es suficiente para completar los objetivos de la misión. El tipo de barrido que se propone en este proyecto es uno espaciado que recorre la trayectoria sin realizar paradas en ninguno de los puntos. Es posible que, ya sea por la morfología del terreno o simplemente por los requisitos del proyecto, sea necesario realizar una parada o medida más exhaustiva en algún punto del terreno, incluso en lugares los que no está la trayectoria de barrido.

Para solucionar este problema, se diseña una función tal que reconozca la posición de cada vehículo y, si está activada la opción, decida cuál de ellos se encuentra más cerca del punto objetivo para así posicionarse adecuadamente. Además, tendrá que realizar la maniobra de una manera adecuada que no ponga en peligro a los demás drones ni al elegido para realizar la medición.

Se diseña la función *Eleccion de dron para un punto*, código 4.7. Ésta toma como entradas las posiciones de los drones, las rutas establecidas según los waypoints, la posición del punto objetivo y, adicionalmente, un "interruptor" que active o desactive la función dependiendo de la posición en la que se encuentre.

Código 4.7 Función elección de dron para un punto.

```
% ==================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========            ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019      =========
% ==================================================================
% =========     FUNCIÓN ELECCIÓN DE DRON PARA UN PUNTO      =========
% ==================================================================

function [ruta_c1,ruta_c2,distancia_1_al_punto,distancia_2_al_punto] = ...
eleccion_dron_punto(posiciones_actuales,ruta_1,ruta_2,punto_objetivo,...
interruptor)

ruta_c1 = ruta_1; ruta_c2 = ruta_2;

posicion_actual_1 = posiciones_actuales(1:3);
posicion_actual_2 = posiciones_actuales(4:6);

x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);

distancia_1_al_punto = abs(sqrt((x1-punto_objetivo(1))^2+(y1-...
                       punto_objetivo(2))^2+(z1-punto_objetivo(3))^2));
distancia_2_al_punto = abs(sqrt((x2-punto_objetivo(1))^2+(y2-...
                       punto_objetivo(2))^2+(z2-punto_objetivo(3))^2));

if interruptor == 1
  if distancia_1_al_punto < distancia_2_al_punto && distancia_1_al_punto<=2
      ruta_c1 = punto_objetivo;
  end
  if distancia_2_al_punto < distancia_1_al_punto && distancia_2_al_punto<=2
      ruta_c2 = punto_objetivo;
  end
end
```

Como se había definido anteriormente, la medición o posicionamiento en el punto objetivo debería realizarse de una manera adecuada sin que generase una perturbación en el desarrollo del proceso de barrido. De este modo, se le impone una condición adicional tal que el dron acuda al punto objetivo únicamente cuando se encuentre a una distancia igual o inferior a los dos metros. Se simula el modelo para dos drones implementando la nueva función mostrándose los resultados en las figuras 4.10 y 4.11.

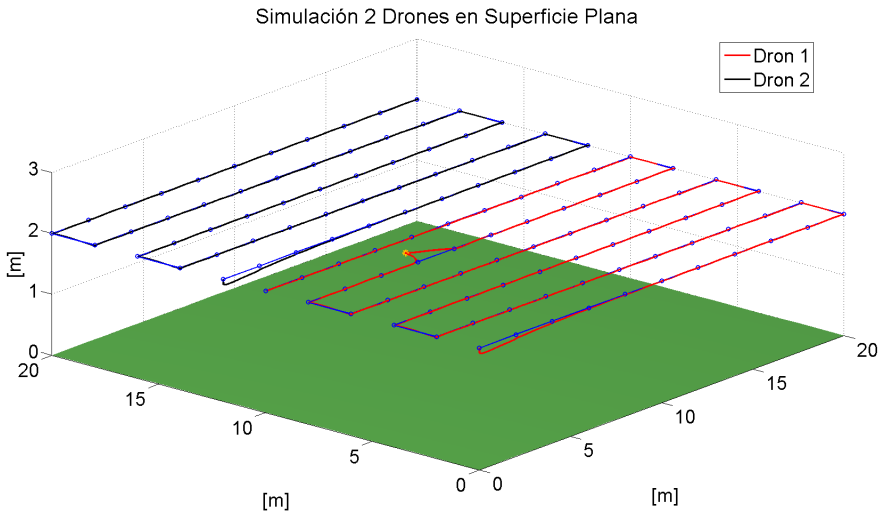

Figura 4.10 Elección de dron para un punto en superficie plana.

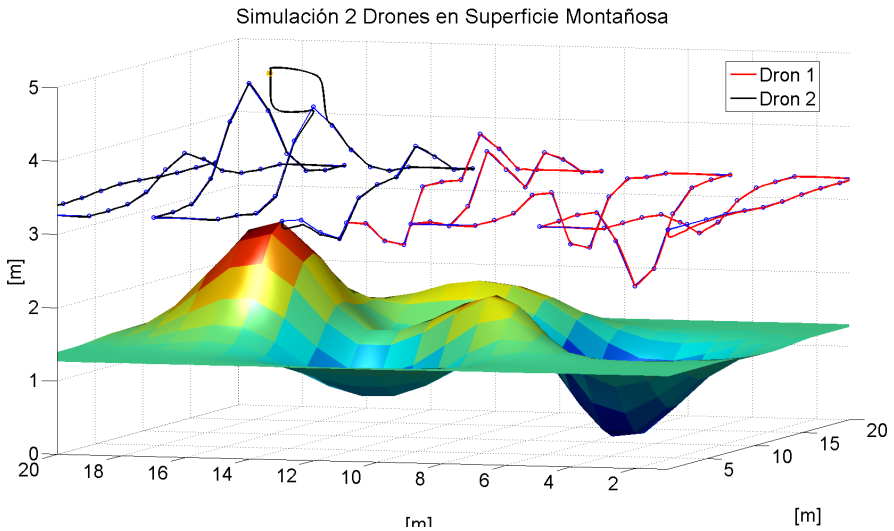

Figura 4.11 Elección de dron para un punto en superficie montañosa.

4.7 Problema del viajante para dos drones

Al igual que en apartado anterior, se realizará un barrido de la superficie según el problema del viajante. Para optimizar las rutas asegurando que cada dron visitará aproximadamente el mismo número de puntos, es necesario generar una función que, dado una serie de puntos aleatorios, tenga la capacidad de repartirlos según se encuentren más próximos a la posición inicial de cada dron. La función *Seleccion de Puntos Salesman*, código 4.8, tiene como entradas las posiciones iniciales de cada dron y la matriz de puntos aleatorios, generando como salida dos matrices de puntos pertenecientes a cada uno de los drones.

Código 4.8 Función selección de puntos para problema del viajante con dos drones.

```
% ====================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========           ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========    TRABAJO DE FIN DE GRADO, SEVILLA 2019       =========
% ====================================================================
% =========  FUNCIÓN SELECCIÓN DE PUNTOS PARA DOS DRONES   =========
% ====================================================================

function [puntos_1,puntos_2]=seleccion_puntos_salesman(posini_1,...
          posini_2,mtxpuntos)

[m,n]=size(mtxpuntos); puntos_1 = []; puntos_2 = []; l1=0; l2=0; e=0;

while e==0

    for i=1:m
        dist_1(i) = sqrt((posini_1(1)-mtxpuntos(i,1))^2+(posini_1(2)-...
                    mtxpuntos(i,2))^2+(posini_1(3)-mtxpuntos(i,3))^2);
    end
    [~,pos]=min(dist_1); puntos_1 = [puntos_1; mtxpuntos(pos,:)];
    mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
    [l1,~]=size(puntos_1);

    if l1+l2==m
    break
    end
```

```
27
28      for i=1:m
29          dist_2(i) = sqrt((posini_2(1)-mtxpuntos(i,1))^2+(posini_2(2)-...
30                      mtxpuntos(i,2))^2+(posini_2(3)-mtxpuntos(i,3))^2);
31      end
32      [~,pos]=min(dist_2); puntos_2 = [puntos_2; mtxpuntos(pos,:)];
33      mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
34      [l2,~]=size(puntos_2);
35
36      if l1+l2==m
37      break
38      end
39
40  end
41
42  end
```

Una vez obtenidas las matrices de puntos, se ejecuta la función que resuelve el problema del viajante 3.11 para calcular la trayectoria óptima de cada uno de los drones y posteriormente se corrige la trayectoria, código 3.12. Se comprueba el correcto funcionamiento de las funciones ejecutando el código 4.9, obteniendo las figuras 4.12 y 4.13.

Código 4.9 Simulación del problema del viajante con dos drones.

```
1  % ======================================================================
2  % ========== MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES ==========
3  % ==========          ENRIQUE CARRASCO DOMÍNGUEZ           ==========
4  % ==========    TRABAJO DE FIN DE GRADO, SEVILLA 2019      ==========
5  % ======================================================================
6  % ==========  MODELO DOS DRONES PROBLEMA DEL VIAJANTE     ==========
7  % ======================================================================
8
9  punto_objetivo = [0 0 0];
10 superficie = [20 20]; dimx=superficie(1); dimy=superficie(2);
11 x=0:dimx; y=0:dimy; z=zeros(length(x),length(y));
12
13 mtxpuntos = [20*rand(50,2), 2+rand(50,1)];
14 posini_1 = [0 0 2]; posini_2 = [20 20 2];
15
16 [puntos_1,puntos_2]=seleccion_puntos_salesman(posini_1, posini_2,mtxpuntos);
17
18 userConfig = struct('xy',puntos_1); [waypoints_1, ~] = tsp_nn(userConfig);
19 userConfig = struct('xy',puntos_2); [waypoints_2, ~] = tsp_nn(userConfig);
20
21 waypoints_1 = [posini_1; waypoints_1];
22 waypoints_2 = [posini_2; waypoints_2];
23
24 waypoints_1 = adiccion_waypoints(waypoints_1);
25 waypoints_2 = adiccion_waypoints(waypoints_2);
26 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
27 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
28
29 sim('quad_control_2drones')
30
31 figure(1)
32 plot3(puntos_1(:,1),puntos_1(:,2),puntos_1(:,3),'*','LineWidth',1.25); hold on;
33 plot3(puntos_2(:,1),puntos_2(:,2),puntos_2(:,3),'*','Color',[1 0 0],...
34 'LineWidth',1.25); hold on; grid on;
35 plot3(mtxpuntos(:,1),mtxpuntos(:,2),mtxpuntos(:,3),'o','Color',[0 0 0],...
36 'LineWidth',1.25);
37 title('Puntos Aleatorios Separados','FontSize', 24);
38 xlabel('[m]','FontSize', 22);
39 ylabel('[m]','FontSize', 22);
40 zlabel('[m]','FontSize', 22);
41 set(gca, 'FontSize', 22);
```

```
xlim([0 20]);
ylim([0 20]);
legend('Puntos 1','Puntos 2','Puntos Aleatorios');

figure(2)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25);
hold on; surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Problema del Viajante por Separado','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2','Waypoints');
```

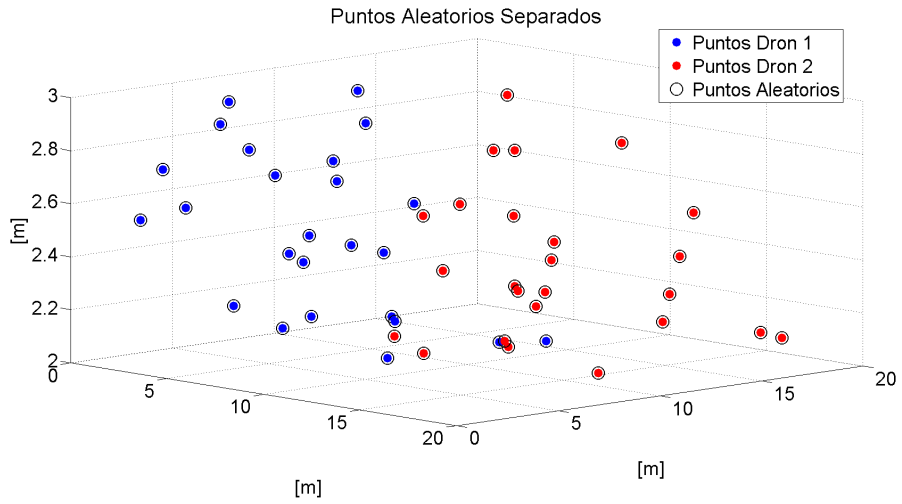

Figura 4.12 Reparto de puntos para el problema del viajante con dos drones.

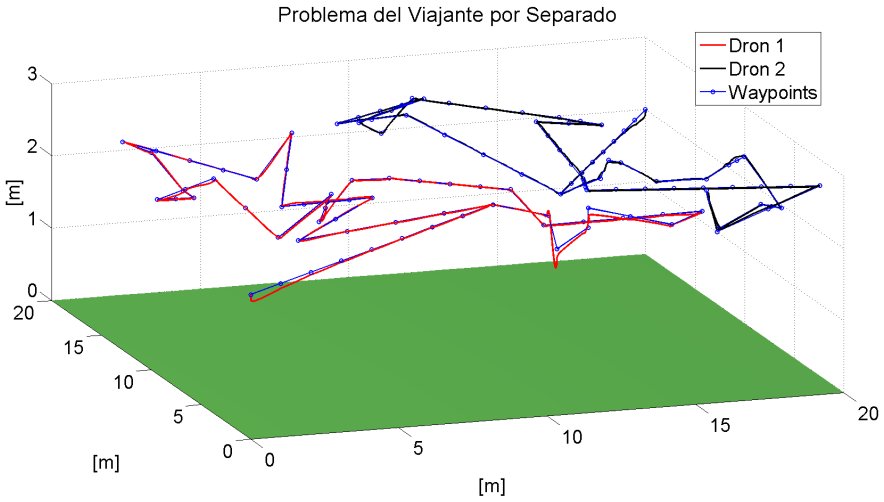

Figura 4.13 Simulación del problema del viajante para dos drones.

El modelo de SimuLink que engloba todo el desarrollo se muestra en la figura 4.14

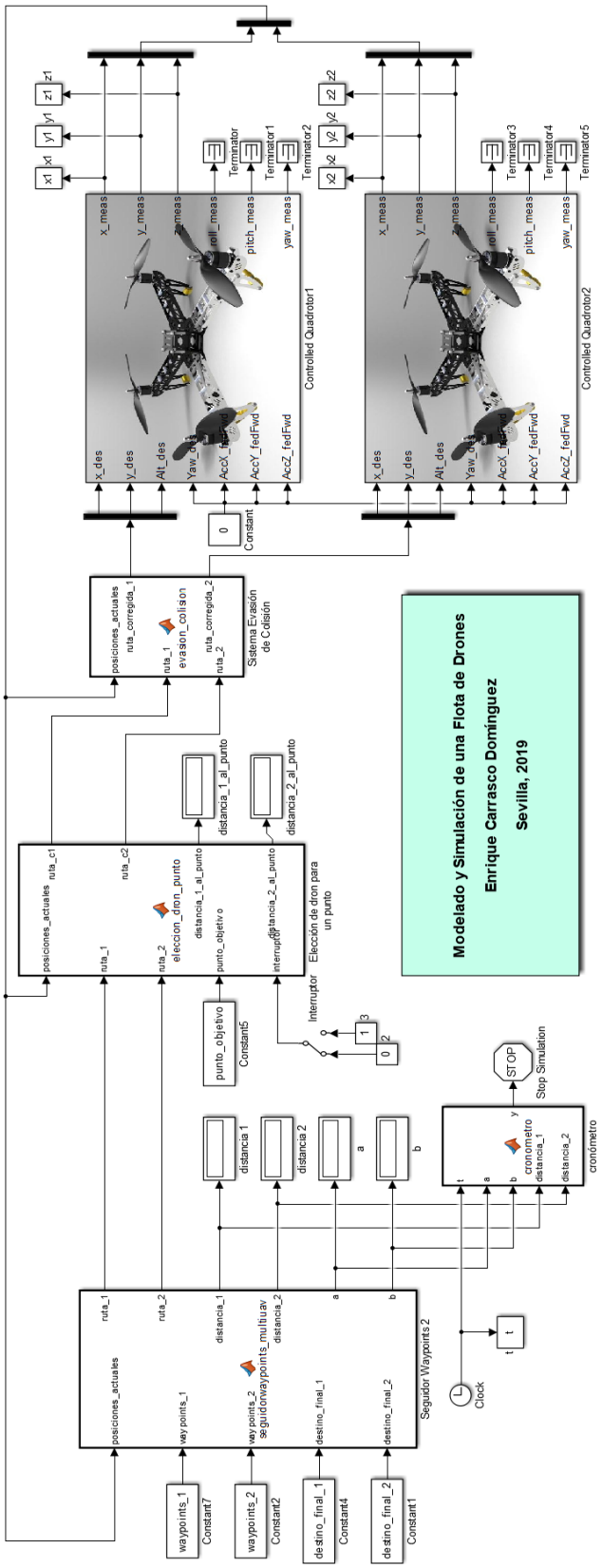

Figura 4.14 Modelo de SimuLink para dos drones.

5 Algoritmos para una flota de ocho drones

El proyecto finaliza con el desarrollo del modelo de una flota compuesta por una cantidad considerable de drones, concretamente se ha decidido que la flota con la que se trabaje sea de ocho. Este número ha sido elegido de modo tal que la explicación de las funciones sea lo suficientemente extensa para que sea fácil entender su funcionamiento y así dar la posibilidad de, si la aplicación lo requiere, añadir o disminuir el número de vehículos por parte de un tercero.

5.1 Control de trayectorias de la flota

De manera similar a lo realizado con dos drones, se modifica el bloque *Seguidor de Waypoints*, código 5.1, que tendrá como entradas las posiciones de todos los drones, sus matrices de waypoints y los vectores que definen la posición de cada uno de ellos tras finalizar el recorrido. Como salida la función dará las rutas de cada uno de ellos en función a su posición actual, un vector con la distancia restante de cada dron hasta alcanzar el próximo waypoint y, además, un vector que contenga el número de waypoints restantes hasta completar la trayectoria.

Cada vehículo circula por su trayectoria de manera independiente al resto por lo que, tal y como se explicó en los apartados anteriores, es necesario disponer de ocho variables globales, una para cada vehículo, que vayan incrementando su valor según se recorran los correspondientes waypoints que conforman la trayectoria de cada dron. Estas variables son del tipo *Simulink.Signal*, A, B, C, D, E, F, G y H.

Código 5.1 Seguidor de Waypints para la flota de ocho drones.

```
% ==================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========           ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019     =========
% ==================================================================
% ========= FUNCIÓN SEGUIDOR DE WAYPOINTS PARA OCHO DRONES =========
% ==================================================================

function [rutas,distancias,waypoints_restantes] = ...
seguidorwaypoints_multiuav(posiciones_actuales,waypoints_1,waypoints_2,...
waypoints_3,waypoints_4,waypoints_5,waypoints_6,waypoints_7,waypoints_8,...
destino_final_1,destino_final_2,destino_final_3,destino_final_4,...
destino_final_5,destino_final_6,destino_final_7,destino_final_8)

posicion_actual_1 = posiciones_actuales(1:3);
posicion_actual_2 = posiciones_actuales(4:6);
```

```
posicion_actual_3 = posiciones_actuales(7:9);
posicion_actual_4 = posiciones_actuales(10:12);
posicion_actual_5 = posiciones_actuales(13:15);
posicion_actual_6 = posiciones_actuales(16:18);
posicion_actual_7 = posiciones_actuales(19:21);
posicion_actual_8 = posiciones_actuales(22:24);

x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);
x3=posicion_actual_3(1); y3=posicion_actual_3(2); z3=posicion_actual_3(3);
x4=posicion_actual_4(1); y4=posicion_actual_4(2); z4=posicion_actual_4(3);
x5=posicion_actual_5(1); y5=posicion_actual_5(2); z5=posicion_actual_5(3);
x6=posicion_actual_6(1); y6=posicion_actual_6(2); z6=posicion_actual_6(3);
x7=posicion_actual_7(1); y7=posicion_actual_7(2); z7=posicion_actual_7(3);
x8=posicion_actual_8(1); y8=posicion_actual_8(2); z8=posicion_actual_8(3);

waypoints_1=[waypoints_1;(destino_final_1)'];
waypoints_2=[waypoints_2;(destino_final_2)'];
waypoints_3=[waypoints_3;(destino_final_3)'];
waypoints_4=[waypoints_4;(destino_final_4)'];
waypoints_5=[waypoints_5;(destino_final_5)'];
waypoints_6=[waypoints_6;(destino_final_6)'];
waypoints_7=[waypoints_7;(destino_final_7)'];
waypoints_8=[waypoints_8;(destino_final_8)'];

[m1,~]=size(waypoints_1); [m2,~]=size(waypoints_2);
[m3,~]=size(waypoints_3); [m4,~]=size(waypoints_4);
[m5,~]=size(waypoints_5); [m6,~]=size(waypoints_6);
[m7,~]=size(waypoints_7); [m8,~]=size(waypoints_8);

global A, global B, global C, global D,
global E, global F, global G, global H;

distancia_1=sqrt((waypoints_1(A,1)-x1)^2+(waypoints_1(A,2)-y1)^2+...
                 (waypoints_1(A,3)-z1)^2);
distancia_2=sqrt((waypoints_2(B,1)-x2)^2+(waypoints_2(B,2)-y2)^2+...
                 (waypoints_2(B,3)-z2)^2);
distancia_3=sqrt((waypoints_3(C,1)-x3)^2+(waypoints_3(C,2)-y3)^2+...
                 (waypoints_3(C,3)-z3)^2);
distancia_4=sqrt((waypoints_4(D,1)-x4)^2+(waypoints_4(D,2)-y4)^2+...
                 (waypoints_4(D,3)-z4)^2);
distancia_5=sqrt((waypoints_5(E,1)-x5)^2+(waypoints_5(E,2)-y5)^2+...
                 (waypoints_5(E,3)-z5)^2);
distancia_6=sqrt((waypoints_6(F,1)-x6)^2+(waypoints_6(F,2)-y6)^2+...
                 (waypoints_6(F,3)-z6)^2);
distancia_7=sqrt((waypoints_7(G,1)-x7)^2+(waypoints_7(G,2)-y7)^2+...
                 (waypoints_7(G,3)-z7)^2);
distancia_8=sqrt((waypoints_8(H,1)-x8)^2+(waypoints_8(H,2)-y8)^2+...
                 (waypoints_8(H,3)-z8)^2);

distancias = [distancia_1,distancia_2,distancia_3,distancia_4,...
              distancia_5,distancia_6,distancia_7,distancia_8];

if distancia_1<0.2, A=A+1; end
if A>m1, A=m1; end
if distancia_2<0.2, B=B+1; end
if B>m2, B=m2; end
if distancia_3<0.2, C=C+1; end
if C>m3, C=m3; end
if distancia_4<0.2, D=D+1; end
if D>m4, D=m4; end
if distancia_5<0.2, E=E+1; end
if E>m5, E=m5; end
if distancia_6<0.2, F=F+1; end
if F>m6, F=m6; end
if distancia_7<0.2, G=G+1; end
if G>m7, G=m7; end
if distancia_8<0.2, H=H+1; end
```

```
if H>m8, H=m8; end

ruta_1=waypoints_1(A,:)'; ruta_2=waypoints_2(B,:)';
ruta_3=waypoints_3(C,:)'; ruta_4=waypoints_4(D,:)';
ruta_5=waypoints_5(E,:)'; ruta_6=waypoints_6(F,:)';
ruta_7=waypoints_7(G,:)'; ruta_8=waypoints_8(H,:)';

a=length(waypoints_1)-A; b=length(waypoints_2)-B;
c=length(waypoints_3)-C; d=length(waypoints_4)-D;
e=length(waypoints_5)-E; f=length(waypoints_6)-F;
g=length(waypoints_7)-G; h=length(waypoints_8)-H;

waypoints_restantes = [a,b,c,d,e,f,g,h];

% Para suavizar las subidas y bajadas, interpolación lineal.

if A~=1 && A~=m1 && waypoints_1(A,1)-waypoints_1(A-1,1)~=0
    ruta_1(3) = waypoints_1(A-1,3)+((waypoints_1(A,3)-waypoints_1(A-1,3))/...
              (waypoints_1(A,1)-waypoints_1(A-1,1)))*(x1-waypoints_1(A-1,1));
end

if B~=1 && B~=m2 && waypoints_2(B,1)-waypoints_2(B-1,1)~=0
    ruta_2(3) = waypoints_2(B-1,3)+((waypoints_2(B,3)-waypoints_2(B-1,3))/...
              (waypoints_2(B,1)-waypoints_2(B-1,1)))*(x2-waypoints_2(B-1,1));
end

if C~=1 && C~=m3 && waypoints_3(C,1)-waypoints_3(C-1,1)~=0
    ruta_3(3) = waypoints_3(C-1,3)+((waypoints_3(C,3)-waypoints_3(C-1,3))/...
              (waypoints_3(C,1)-waypoints_3(C-1,1)))*(x3-waypoints_3(C-1,1));
end

if D~=1 && D~=m4 && waypoints_4(D,1)-waypoints_4(D-1,1)~=0
    ruta_4(3) = waypoints_4(D-1,3)+((waypoints_4(D,3)-waypoints_4(D-1,3))/...
              (waypoints_4(D,1)-waypoints_4(D-1,1)))*(x4-waypoints_4(D-1,1));
end

if E~=1 && E~=m5 && waypoints_5(E,1)-waypoints_5(E-1,1)~=0
    ruta_5(3) = waypoints_5(E-1,3)+((waypoints_5(E,3)-waypoints_5(E-1,3))/...
              (waypoints_5(E,1)-waypoints_5(E-1,1)))*(x5-waypoints_5(E-1,1));
end

if F~=1 && F~=m6 && waypoints_6(F,1)-waypoints_6(F-1,1)~=0
    ruta_6(3) = waypoints_6(F-1,3)+((waypoints_6(F,3)-waypoints_6(F-1,3))/...
              (waypoints_6(F,1)-waypoints_6(F-1,1)))*(x6-waypoints_6(F-1,1));
end

if G~=1 && G~=m7 && waypoints_7(G,1)-waypoints_7(G-1,1)~=0
    ruta_7(3) = waypoints_7(G-1,3)+((waypoints_7(G,3)-waypoints_7(G-1,3))/...
              (waypoints_7(G,1)-waypoints_7(G-1,1)))*(x7-waypoints_7(G-1,1));
end

if H~=1 && H~=m8 && waypoints_8(H,1)-waypoints_8(H-1,1)~=0
    ruta_8(3) = waypoints_8(H-1,3)+((waypoints_8(H,3)-waypoints_8(H-1,3))/...
              (waypoints_8(H,1)-waypoints_8(H-1,1)))*(x8-waypoints_8(H-1,1));
end

rutas = [ruta_1;ruta_2;ruta_3;ruta_4;ruta_5;ruta_6;ruta_7;ruta_8];
```

También se modifica la función *Cronometro* para que detenga la simulación cuando todos los vehículos hayan completado la trayectoria, código 5.2.

Código 5.2 Función cronómetro para la flota.

```
% ================================================================
% ========    MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES    ========
% ========             ENRIQUE CARRASCO DOMÍNGUEZ             ========
```

```
4  % ========       TRABAJO DE FIN DE GRADO, SEVILLA 2019      ========
5  % ========================================================================
6  % ======== FUNCIÓN CRONÓMETRO PARA LA FLOTA DE OCHO DRONES  ========
7  % ========================================================================
8
9  function y  = cronometro(t,distancias,waypoints_restantes)
10 suma=0;
11 for i=1:length(waypoints_restantes), suma=suma+waypoints_restantes(i); end
12
13 if t>0 && suma==0 && distancias(1)<0.05 && distancias(2)<0.05 && ...
14 distancias(3)<0.05 && distancias(4)<0.05 && distancias(5)<0.05 && ...
15 distancias(6)<0.05 && distancias(7)<0.05 && distancias(8)<0.05
16      y=1;
17 else y=0;
18
19 end
```

Se comprueba el funcionamiento de la función creando varias matrices de waypoints, una para cada vehículo, con trayectorias que no presenten peligro de colisión, figura 5.1.

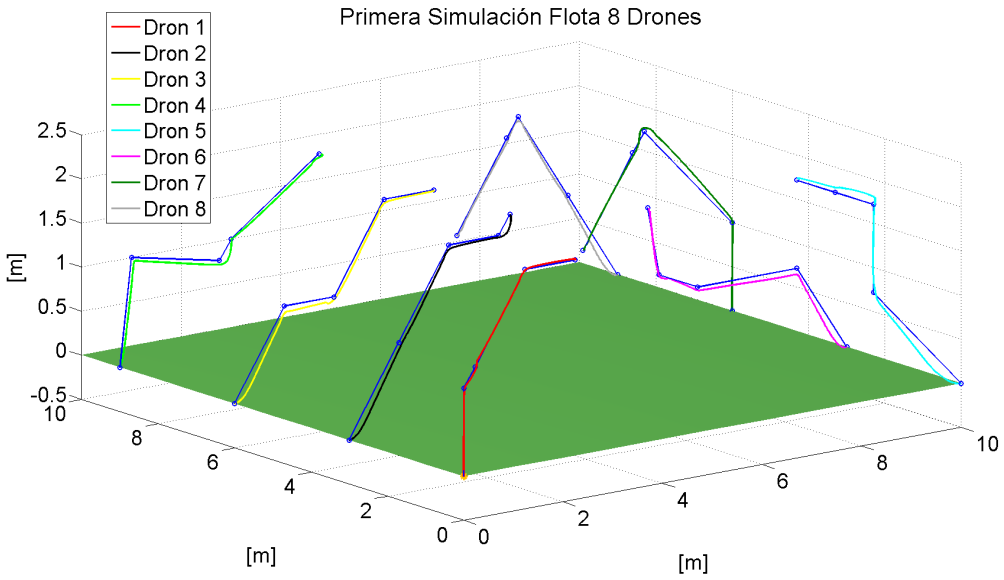

Figura 5.1 Primera simulación de la flota.

5.2 Evasión de colisión

El funcionamiento del sistema de evasión de colisión juega un papel clave cuando se trata de un número considerable de drones. Los problemas derivados de una colisión entre dos o más vehículos podrían perjudicar gravemente el desarrollo de los objetivos del proyecto. Para mitigar este efecto, se modifica la función *Sistema Evasion de Colision* para que actúe de manera semejante a como lo hacía con dos drones. Esta función toma como entrada las posiciones de los vehículos y las rutas, dando como salida las rutas corregidas para evitar una posible colisión, código 5.3.

Este sistema tiene en cuenta todas y cada una de las posibles colisiones entre vehículos comenzando la implementación con las posibles colisiones que podría sufrir el primer dron con todos los demás hasta llegar la posible colisión del séptimo con el octavo. Se ha decidido que sea solo uno de ellos el que varíe su trayectoria incrementando o disminuyendo su altura para evitar así la colisión.

Del mismo modo que con dos vehículos, si dos drones, se encuentran a una distancia inferior a 1.5 *metros*, solo uno de ellos modificará su trayectoria.

Código 5.3 Sistema evasión de colisión para la flota.

```
% ====================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========           ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019     =========
% ====================================================================
% =========      FUNCIÓN SISTEMA EVASIÓN DE COLISIÓN       =========
% ====================================================================

function rutas_corregidas = evasion_colision(posiciones_actuales,rutas)

posicion_actual_1 = posiciones_actuales(1:3);
posicion_actual_2 = posiciones_actuales(4:6);
posicion_actual_3 = posiciones_actuales(7:9);
posicion_actual_4 = posiciones_actuales(10:12);
posicion_actual_5 = posiciones_actuales(13:15);
posicion_actual_6 = posiciones_actuales(16:18);
posicion_actual_7 = posiciones_actuales(19:21);
posicion_actual_8 = posiciones_actuales(22:24);

x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);
x3=posicion_actual_3(1); y3=posicion_actual_3(2); z3=posicion_actual_3(3);
x4=posicion_actual_4(1); y4=posicion_actual_4(2); z4=posicion_actual_4(3);
x5=posicion_actual_5(1); y5=posicion_actual_5(2); z5=posicion_actual_5(3);
x6=posicion_actual_6(1); y6=posicion_actual_6(2); z6=posicion_actual_6(3);
x7=posicion_actual_7(1); y7=posicion_actual_7(2); z7=posicion_actual_7(3);
x8=posicion_actual_8(1); y8=posicion_actual_8(2); z8=posicion_actual_8(3);

distancia_entre_drones = zeros(1,28);
zpos=[z1 z2 z3 z4 z5 z6 z7 z8];

% Distancia 1 respecto a 2,3,4,5,6,7,8
distancia_entre_drones(1) = abs(sqrt((x1-x2)^2+(y1-y2)^2+(z1-z2)^2));
distancia_entre_drones(2) = abs(sqrt((x1-x3)^2+(y1-y3)^2+(z1-z3)^2));
distancia_entre_drones(3) = abs(sqrt((x1-x4)^2+(y1-y4)^2+(z1-z4)^2));
distancia_entre_drones(4) = abs(sqrt((x1-x5)^2+(y1-y5)^2+(z1-z5)^2));
distancia_entre_drones(5) = abs(sqrt((x1-x6)^2+(y1-y6)^2+(z1-z6)^2));
distancia_entre_drones(6) = abs(sqrt((x1-x7)^2+(y1-y7)^2+(z1-z7)^2));
distancia_entre_drones(7) = abs(sqrt((x1-x8)^2+(y1-y8)^2+(z1-z8)^2));

% Distancia 2 respecto a 3,4,5,6,7,8
distancia_entre_drones(8)  = abs(sqrt((x2-x3)^2+(y2-y3)^2+(z2-z3)^2));
distancia_entre_drones(9)  = abs(sqrt((x2-x4)^2+(y2-y4)^2+(z2-z4)^2));
distancia_entre_drones(10) = abs(sqrt((x2-x5)^2+(y2-y5)^2+(z2-z5)^2));
distancia_entre_drones(11) = abs(sqrt((x2-x6)^2+(y2-y6)^2+(z2-z6)^2));
distancia_entre_drones(12) = abs(sqrt((x2-x7)^2+(y2-y7)^2+(z2-z7)^2));
distancia_entre_drones(13) = abs(sqrt((x2-x8)^2+(y2-y8)^2+(z2-z8)^2));

% Distancia 3 respecto a 4,5,6,7,8
distancia_entre_drones(14) = abs(sqrt((x3-x4)^2+(y3-y4)^2+(z3-z4)^2));
distancia_entre_drones(15) = abs(sqrt((x3-x5)^2+(y3-y5)^2+(z3-z5)^2));
distancia_entre_drones(16) = abs(sqrt((x3-x6)^2+(y3-y6)^2+(z3-z6)^2));
distancia_entre_drones(17) = abs(sqrt((x3-x7)^2+(y3-y7)^2+(z3-z7)^2));
distancia_entre_drones(18) = abs(sqrt((x3-x8)^2+(y3-y8)^2+(z3-z8)^2));

% Distancia 4 respecto a 5,6,7,8
distancia_entre_drones(19) = abs(sqrt((x4-x5)^2+(y4-y5)^2+(z4-z5)^2));
distancia_entre_drones(20) = abs(sqrt((x4-x6)^2+(y4-y6)^2+(z4-z6)^2));
distancia_entre_drones(21) = abs(sqrt((x4-x7)^2+(y4-y7)^2+(z4-z7)^2));
distancia_entre_drones(22) = abs(sqrt((x4-x8)^2+(y4-y8)^2+(z4-z8)^2));

```

```
% Distancia 5 respecto a 6,7,8
distancia_entre_drones(23) = abs(sqrt((x5-x6)^2+(y5-y6)^2+(z5-z6)^2));
distancia_entre_drones(24) = abs(sqrt((x5-x7)^2+(y5-y7)^2+(z5-z7)^2));
distancia_entre_drones(25) = abs(sqrt((x5-x8)^2+(y5-y8)^2+(z5-z8)^2));

% Distancia 6 respecto a 7,8
distancia_entre_drones(26) = abs(sqrt((x6-x7)^2+(y6-y7)^2+(z6-z7)^2));
distancia_entre_drones(27) = abs(sqrt((x6-x8)^2+(y6-y8)^2+(z6-z8)^2));

% Distancia 7 respecto a 8
distancia_entre_drones(28) = abs(sqrt((x7-x8)^2+(y7-y8)^2+(z7-z8)^2));

rutas_corregidas = rutas;

for i=1:7
   if distancia_entre_drones(i)< 1.5
    if zpos(i+1)<=z1
         rutas_corregidas(3) = rutas_corregidas(3)+1;
    else rutas_corregidas(3) = rutas_corregidas(3)-0.5;
    end
   end
end

for i=8:13
   if distancia_entre_drones(i)< 1.5
    if zpos(i-5)<=z2
         rutas_corregidas(6) = rutas_corregidas(6)+1;
    else rutas_corregidas(6) = rutas_corregidas(6)-0.5;
    end
   end
end

for i=14:18
   if distancia_entre_drones(i)< 1.5
    if zpos(i-10)<=z3
         rutas_corregidas(9) = rutas_corregidas(9)+1;
    else rutas_corregidas(9) = rutas_corregidas(9)-0.5;
    end
   end
end

for i=19:22
   if distancia_entre_drones(i)< 1.5
    if zpos(i-14)<=z4
         rutas_corregidas(12) = rutas_corregidas(12)+1;
    else rutas_corregidas(12) = rutas_corregidas(12)-0.5;
    end
   end
end

for i=23:25
   if distancia_entre_drones(i)< 1.5
    if zpos(i-17)<=z5
         rutas_corregidas(15) = rutas_corregidas(15)+1;
    else rutas_corregidas(15) = rutas_corregidas(15)-0.5;
    end
   end
end

for i=26:27
   if distancia_entre_drones(i)< 1.5
    if zpos(i-19)<=z6
         rutas_corregidas(18) = rutas_corregidas(18)+1;
    else rutas_corregidas(18) = rutas_corregidas(18)-0.5;
    end
   end
end

```

```
130 if distancia_entre_drones(28)< 1.5
131     if zpos(8)<=z7
132          rutas_corregidas(21) = rutas_corregidas(21)+1;
133     else rutas_corregidas(21) = rutas_corregidas(21)-0.5;
134     end
135 end
136
137 end
```

Para comprobar que el sistema funciona adecuadamente en todas las situaciones, se generan, en siete casos diferentes, las trayectorias que contemplen las colisiones de todos contra todos a diferentes alturas, código 5.4. Los resultados se muestran el las figuras 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 y 5.8.

Código 5.4 Modelo para simulación de las colisiones.

```
1  % ====================================================================
2  % ========   MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES   ========
3  % ========            ENRIQUE CARRASCO DOMÍNGUEZ            ========
4  % ========       TRABAJO DE FIN DE GRADO, SEVILLA 2019      ========
5  % ====================================================================
6  % ========  MODELO FLOTA DE OCHO DRONES EVASIÓN DE COLISIÓN ========
7  % ====================================================================
8
9  superficie = [8 6]; dimx = superficie(1); dimy = superficie(2);
10 x = 0:dimx; y = 0:dimy; z = zeros(length(x),length(y));
11 punto_objetivo = [0 0 0];
12
13 wp_1 = [0 0 2; 2 0 2; 4 0 2; 6 0 2; 8 0 2];
14 wp_2 = [0 2 2.5; 2 2 2.5; 4 2 2.5; 6 2 2.5; 8 2 2.5];
15 wp_3 = [0 4 2.5; 2 4 2.5; 4 4 2.5; 6 4 2.5; 8 4 2.5];
16 wp_4 = [0 6 2; 2 6 2; 4 6 2; 6 6 2; 8 6 2];
17
18 wp_5 = [8 0 2.5; 6 0 2.5; 4 0 2.5; 2 0 2.5; 0 0 2.5];
19 wp_6 = [8 2 2; 6 2 2; 4 2 2; 2 2 2; 0 2 2];
20 wp_7 = [8 4 2; 6 4 2; 4 4 2; 2 4 2; 0 4 2];
21 wp_8 = [8 6 2.5; 6 6 2.5; 4 6 2.5; 2 6 2.5; 0 6 2.5];
22
23 % ========================================================================
24 % ================================ CASO 1 ================================
25 % ========================================================================
26
27 waypoints_1 = wp_4; waypoints_2 = wp_8;
28 waypoints_3 = wp_3; waypoints_4 = wp_7;
29 waypoints_5 = wp_2; waypoints_6 = wp_6;
30 waypoints_7 = wp_1; waypoints_8 = wp_5;
31
32
33 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
34 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
35 destino_final_3 = [waypoints_3(length(waypoints_3),:)];
36 destino_final_4 = [waypoints_4(length(waypoints_4),:)];
37 destino_final_5 = [waypoints_5(length(waypoints_5),:)];
38 destino_final_6 = [waypoints_6(length(waypoints_6),:)];
39 destino_final_7 = [waypoints_7(length(waypoints_7),:)];
40 destino_final_8 = [waypoints_8(length(waypoints_8),:)];
41
42 posini_1 = [waypoints_1(1,:)];
43 posini_2 = [waypoints_2(1,:)];
44 posini_3 = [waypoints_3(1,:)];
45 posini_4 = [waypoints_4(1,:)];
46 posini_5 = [waypoints_5(1,:)];
47 posini_6 = [waypoints_6(1,:)];
48 posini_7 = [waypoints_7(1,:)];
49 posini_8 = [waypoints_8(1,:)];
50
```

```
sim('quad_control_8_drones')

figure(1)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(x3,y3,z3,'y','LineWidth',2); hold on;
plot3(x4,y4,z4,'g','LineWidth',2); hold on;
plot3(x5,y5,z5,'c','LineWidth',2); hold on;
plot3(x6,y6,z6,'m','LineWidth',2); hold on;
plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Evasión de Colisión: 8 Drones (Caso 1)','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
       'Dron 7','Dron 8');

% ========================================================================
% ================================ CASO 2 ================================
% ========================================================================

waypoints_1 = wp_4; waypoints_3 = wp_8;
waypoints_2 = wp_3; waypoints_4 = wp_7;
waypoints_5 = wp_2; waypoints_7 = wp_6;
waypoints_6 = wp_1; waypoints_8 = wp_5;


destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];
destino_final_3 = [waypoints_3(length(waypoints_3),:)];
destino_final_4 = [waypoints_4(length(waypoints_4),:)];
destino_final_5 = [waypoints_5(length(waypoints_5),:)];
destino_final_6 = [waypoints_6(length(waypoints_6),:)];
destino_final_7 = [waypoints_7(length(waypoints_7),:)];
destino_final_8 = [waypoints_8(length(waypoints_8),:)];

posini_1 = [waypoints_1(1,:)];
posini_2 = [waypoints_2(1,:)];
posini_3 = [waypoints_3(1,:)];
posini_4 = [waypoints_4(1,:)];
posini_5 = [waypoints_5(1,:)];
posini_6 = [waypoints_6(1,:)];
posini_7 = [waypoints_7(1,:)];
```

```
posini_8 = [waypoints_8(1,:)];

sim('quad_control_8_drones')

figure(2)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(x3,y3,z3,'y','LineWidth',2); hold on;
plot3(x4,y4,z4,'g','LineWidth',2); hold on;
plot3(x5,y5,z5,'c','LineWidth',2); hold on;
plot3(x6,y6,z6,'m','LineWidth',2); hold on;
plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Evasión de Colisión: 8 Drones (Caso 2)','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
       'Dron 7','Dron 8');


% ================================================================================
% ================================ CASO 3 ========================================
% ================================================================================

waypoints_1 = wp_4; waypoints_4 = wp_8;
waypoints_2 = wp_3; waypoints_3 = wp_7;
waypoints_5 = wp_2; waypoints_8 = wp_6;
waypoints_6 = wp_1; waypoints_7 = wp_5;


destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];
destino_final_3 = [waypoints_3(length(waypoints_3),:)];
destino_final_4 = [waypoints_4(length(waypoints_4),:)];
destino_final_5 = [waypoints_5(length(waypoints_5),:)];
destino_final_6 = [waypoints_6(length(waypoints_6),:)];
destino_final_7 = [waypoints_7(length(waypoints_7),:)];
destino_final_8 = [waypoints_8(length(waypoints_8),:)];

posini_1 = [waypoints_1(1,:)];
posini_2 = [waypoints_2(1,:)];
posini_3 = [waypoints_3(1,:)];
posini_4 = [waypoints_4(1,:)];
```

```
posini_5 = [waypoints_5(1,:)];
posini_6 = [waypoints_6(1,:)];
posini_7 = [waypoints_7(1,:)];
posini_8 = [waypoints_8(1,:)];

sim('quad_control_8_drones')

figure(3)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(x3,y3,z3,'y','LineWidth',2); hold on;
plot3(x4,y4,z4,'g','LineWidth',2); hold on;
plot3(x5,y5,z5,'c','LineWidth',2); hold on;
plot3(x6,y6,z6,'m','LineWidth',2); hold on;
plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Evasión de Colisión: 8 Drones (Caso 3)','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
       'Dron 7','Dron 8');


% ========================================================================
% ================================ CASO 4 ================================
% ========================================================================

waypoints_1 = wp_4; waypoints_5 = wp_8;
waypoints_2 = wp_3; waypoints_6 = wp_7;
waypoints_3 = wp_2; waypoints_7 = wp_6;
waypoints_4 = wp_1; waypoints_8 = wp_5;


destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];
destino_final_3 = [waypoints_3(length(waypoints_3),:)];
destino_final_4 = [waypoints_4(length(waypoints_4),:)];
destino_final_5 = [waypoints_5(length(waypoints_5),:)];
destino_final_6 = [waypoints_6(length(waypoints_6),:)];
destino_final_7 = [waypoints_7(length(waypoints_7),:)];
destino_final_8 = [waypoints_8(length(waypoints_8),:)];

posini_1 = [waypoints_1(1,:)];
```

```
255 posini_2 = [waypoints_2(1,:)];
256 posini_3 = [waypoints_3(1,:)];
257 posini_4 = [waypoints_4(1,:)];
258 posini_5 = [waypoints_5(1,:)];
259 posini_6 = [waypoints_6(1,:)];
260 posini_7 = [waypoints_7(1,:)];
261 posini_8 = [waypoints_8(1,:)];
262
263 sim('quad_control_8_drones')
264
265 figure(4)
266 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
267 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
268 plot3(x3,y3,z3,'y','LineWidth',2); hold on;
269 plot3(x4,y4,z4,'g','LineWidth',2); hold on;
270 plot3(x5,y5,z5,'c','LineWidth',2); hold on;
271 plot3(x6,y6,z6,'m','LineWidth',2); hold on;
272 plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
273 plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
274 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
275 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
276 'LineWidth',1.25); hold on;
277 plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
278 'LineWidth',1.25); hold on;
279 plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
280 'LineWidth',1.25); hold on;
281 plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
282 'LineWidth',1.25); hold on;
283 plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
284 'LineWidth',1.25); hold on;
285 plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
286 'LineWidth',1.25); hold on;
287 plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
288 'LineWidth',1.25); hold on;
289 plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
290 'LineWidth',1.25);
291
292 camlight left; grid on;
293 lighting phong
294
295 title('Evasión de Colisión: 8 Drones (Caso 4)','FontSize', 24);
296 xlabel('[m]','FontSize', 22);
297 ylabel('[m]','FontSize', 22);
298 zlabel('[m]','FontSize', 22);
299 set(gca, 'FontSize', 22);
300 xlim([0 superficie(1)]);
301 ylim([0 superficie(2)]);
302 legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
303        'Dron 7','Dron 8');
304
305
306 % ========================================================================
307 % ================================ CASO 5 ================================
308 % ========================================================================
309
310 waypoints_1 = wp_4; waypoints_6 = wp_8;
311 waypoints_2 = wp_3; waypoints_5 = wp_7;
312 waypoints_3 = wp_2; waypoints_8 = wp_6;
313 waypoints_4 = wp_1; waypoints_7 = wp_5;
314
315
316 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
317 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
318 destino_final_3 = [waypoints_3(length(waypoints_3),:)];
319 destino_final_4 = [waypoints_4(length(waypoints_4),:)];
320 destino_final_5 = [waypoints_5(length(waypoints_5),:)];
321 destino_final_6 = [waypoints_6(length(waypoints_6),:)];
322 destino_final_7 = [waypoints_7(length(waypoints_7),:)];
```

```
destino_final_8 = [waypoints_8(length(waypoints_8),:)];

posini_1 = [waypoints_1(1,:)];
posini_2 = [waypoints_2(1,:)];
posini_3 = [waypoints_3(1,:)];
posini_4 = [waypoints_4(1,:)];
posini_5 = [waypoints_5(1,:)];
posini_6 = [waypoints_6(1,:)];
posini_7 = [waypoints_7(1,:)];
posini_8 = [waypoints_8(1,:)];

sim('quad_control_8_drones')

figure(5)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(x3,y3,z3,'y','LineWidth',2); hold on;
plot3(x4,y4,z4,'g','LineWidth',2); hold on;
plot3(x5,y5,z5,'c','LineWidth',2); hold on;
plot3(x6,y6,z6,'m','LineWidth',2); hold on;
plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25);

camlight left; grid on;
lighting phong

title('Evasión de Colisión: 8 Drones (Caso 5)','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
       'Dron 7','Dron 8');


% ================================================================================
% ================================ CASO 6 ================================
% ================================================================================

waypoints_1 = wp_4; waypoints_7 = wp_8;
waypoints_2 = wp_3; waypoints_8 = wp_7;
waypoints_3 = wp_2; waypoints_5 = wp_6;
waypoints_4 = wp_1; waypoints_6 = wp_5;


destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];
destino_final_3 = [waypoints_3(length(waypoints_3),:)];
destino_final_4 = [waypoints_4(length(waypoints_4),:)];
```

```
391 destino_final_5 = [waypoints_5(length(waypoints_5),:)];
392 destino_final_6 = [waypoints_6(length(waypoints_6),:)];
393 destino_final_7 = [waypoints_7(length(waypoints_7),:)];
394 destino_final_8 = [waypoints_8(length(waypoints_8),:)];
395
396 posini_1 = [waypoints_1(1,:)];
397 posini_2 = [waypoints_2(1,:)];
398 posini_3 = [waypoints_3(1,:)];
399 posini_4 = [waypoints_4(1,:)];
400 posini_5 = [waypoints_5(1,:)];
401 posini_6 = [waypoints_6(1,:)];
402 posini_7 = [waypoints_7(1,:)];
403 posini_8 = [waypoints_8(1,:)];
404
405 sim('quad_control_8_drones')
406
407 figure(6)
408 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
409 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
410 plot3(x3,y3,z3,'y','LineWidth',2); hold on;
411 plot3(x4,y4,z4,'g','LineWidth',2); hold on;
412 plot3(x5,y5,z5,'c','LineWidth',2); hold on;
413 plot3(x6,y6,z6,'m','LineWidth',2); hold on;
414 plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
415 plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
416 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
417 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
418 'LineWidth',1.25); hold on;
419 plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
420 'LineWidth',1.25); hold on;
421 plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
422 'LineWidth',1.25); hold on;
423 plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
424 'LineWidth',1.25); hold on;
425 plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
426 'LineWidth',1.25); hold on;
427 plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
428 'LineWidth',1.25); hold on;
429 plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
430 'LineWidth',1.25); hold on;
431 plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
432 'LineWidth',1.25);
433
434 camlight left; grid on;
435 lighting phong
436
437 title('Evasión de Colisión: 8 Drones (Caso 6)','FontSize', 24);
438 xlabel('[m]','FontSize', 22);
439 ylabel('[m]','FontSize', 22);
440 zlabel('[m]','FontSize', 22);
441 set(gca, 'FontSize', 22);
442 xlim([0 superficie(1)]);
443 ylim([0 superficie(2)]);
444 legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
445        'Dron 7','Dron 8');
446
447
448 % ================================================================================
449 % ================================ CASO 7 ================================
450 % ================================================================================
451
452 waypoints_1 = wp_4; waypoints_8 = wp_8;
453 waypoints_2 = wp_3; waypoints_7 = wp_7;
454 waypoints_3 = wp_2; waypoints_6 = wp_6;
455 waypoints_4 = wp_1; waypoints_5 = wp_5;
456
457
458 destino_final_1 = [waypoints_1(length(waypoints_1),:)];
```

```
459 destino_final_2 = [waypoints_2(length(waypoints_2),:)];
460 destino_final_3 = [waypoints_3(length(waypoints_3),:)];
461 destino_final_4 = [waypoints_4(length(waypoints_4),:)];
462 destino_final_5 = [waypoints_5(length(waypoints_5),:)];
463 destino_final_6 = [waypoints_6(length(waypoints_6),:)];
464 destino_final_7 = [waypoints_7(length(waypoints_7),:)];
465 destino_final_8 = [waypoints_8(length(waypoints_8),:)];
466
467 posini_1 = [waypoints_1(1,:)];
468 posini_2 = [waypoints_2(1,:)];
469 posini_3 = [waypoints_3(1,:)];
470 posini_4 = [waypoints_4(1,:)];
471 posini_5 = [waypoints_5(1,:)];
472 posini_6 = [waypoints_6(1,:)];
473 posini_7 = [waypoints_7(1,:)];
474 posini_8 = [waypoints_8(1,:)];
475
476 sim('quad_control_8_drones')
477
478 figure(7)
479 plot3(x1,y1,z1,'r','LineWidth',2); hold on;
480 plot3(x2,y2,z2,'k','LineWidth',2); hold on;
481 plot3(x3,y3,z3,'y','LineWidth',2); hold on;
482 plot3(x4,y4,z4,'g','LineWidth',2); hold on;
483 plot3(x5,y5,z5,'c','LineWidth',2); hold on;
484 plot3(x6,y6,z6,'m','LineWidth',2); hold on;
485 plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
486 plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
487 surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
488 plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
489 'LineWidth',1.25); hold on;
490 plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
491 'LineWidth',1.25); hold on;
492 plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
493 'LineWidth',1.25); hold on;
494 plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
495 'LineWidth',1.25); hold on;
496 plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
497 'LineWidth',1.25); hold on;
498 plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
499 'LineWidth',1.25); hold on;
500 plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
501 'LineWidth',1.25); hold on;
502 plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
503 'LineWidth',1.25);
504
505 camlight left; grid on;
506 lighting phong
507
508 title('Evasión de Colisión: 8 Drones (Caso 7)','FontSize', 24);
509 xlabel('[m]','FontSize', 22);
510 ylabel('[m]','FontSize', 22);
511 zlabel('[m]','FontSize', 22);
512 set(gca, 'FontSize', 22);
513 xlim([0 superficie(1)]);
514 ylim([0 superficie(2)]);
515 legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
516        'Dron 7','Dron 8');
```

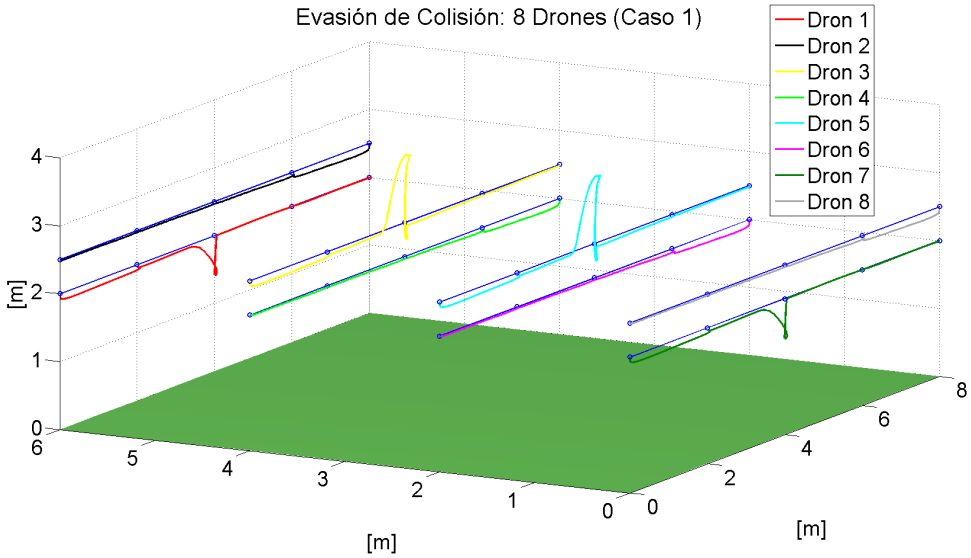

Figura 5.2 Evasión de colisión de la flota (Caso 1).

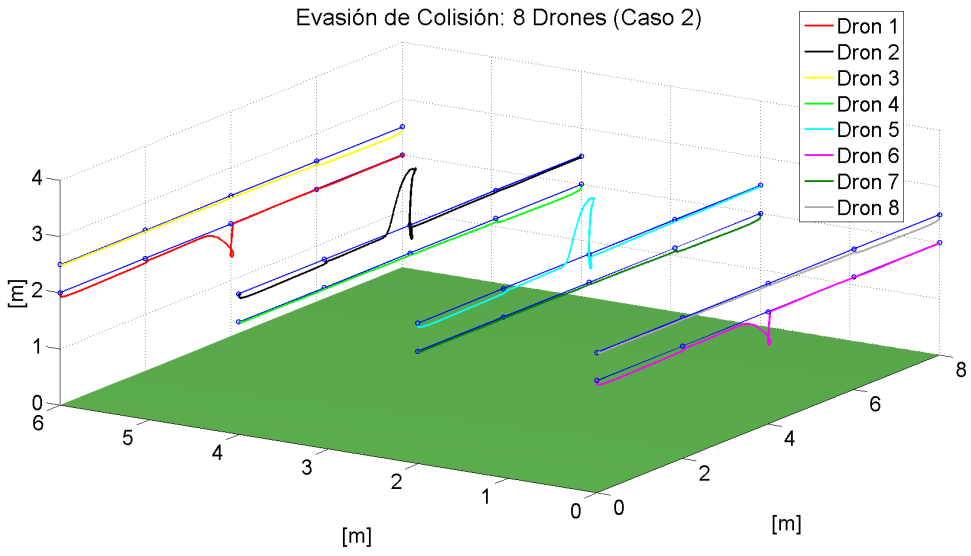

Figura 5.3 Evasión de colisión de la flota (Caso 2).

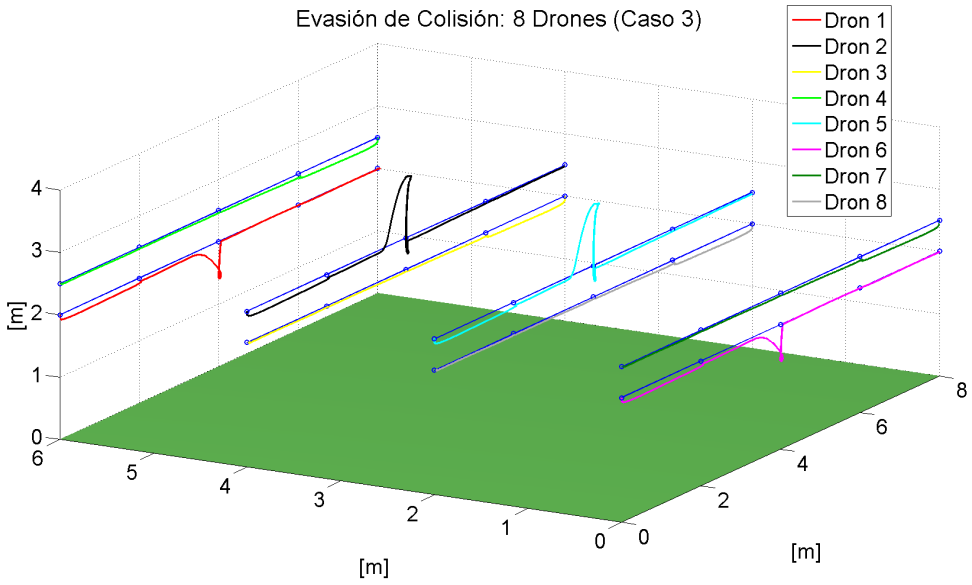

Figura 5.4 Evasión de colisión de la flota (Caso 3).

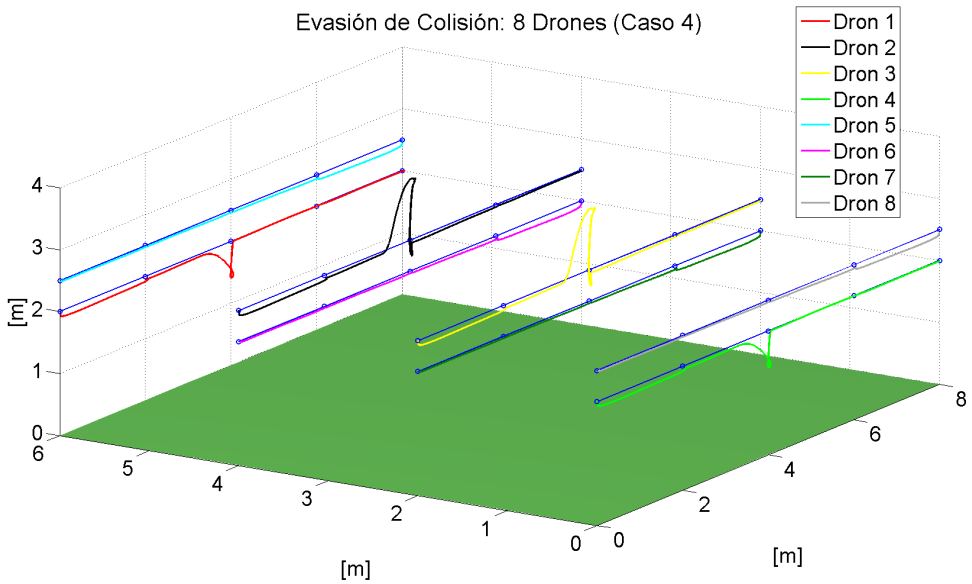

Figura 5.5 Evasión de colisión de la flota (Caso 4).

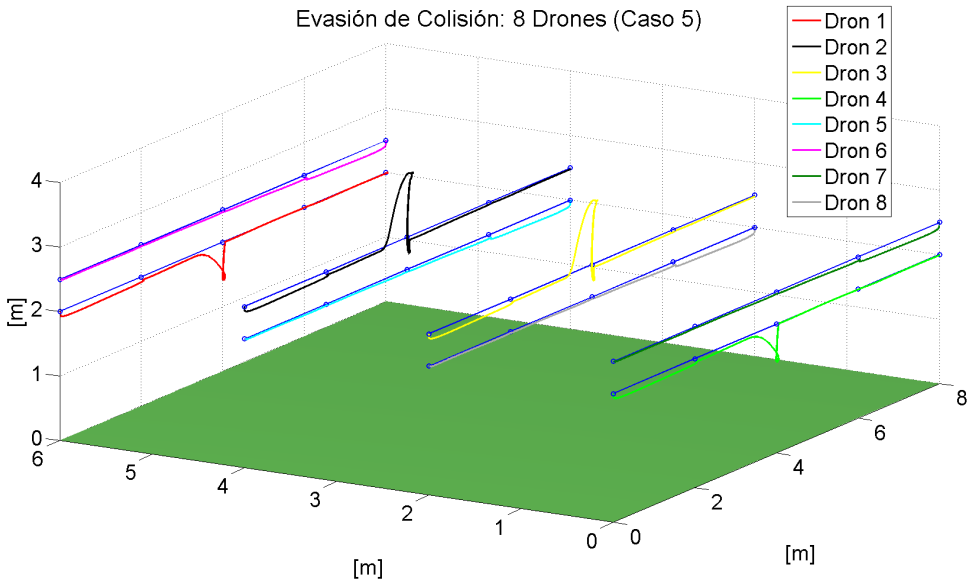

Figura 5.6 Evasión de colisión de la flota (Caso 5).

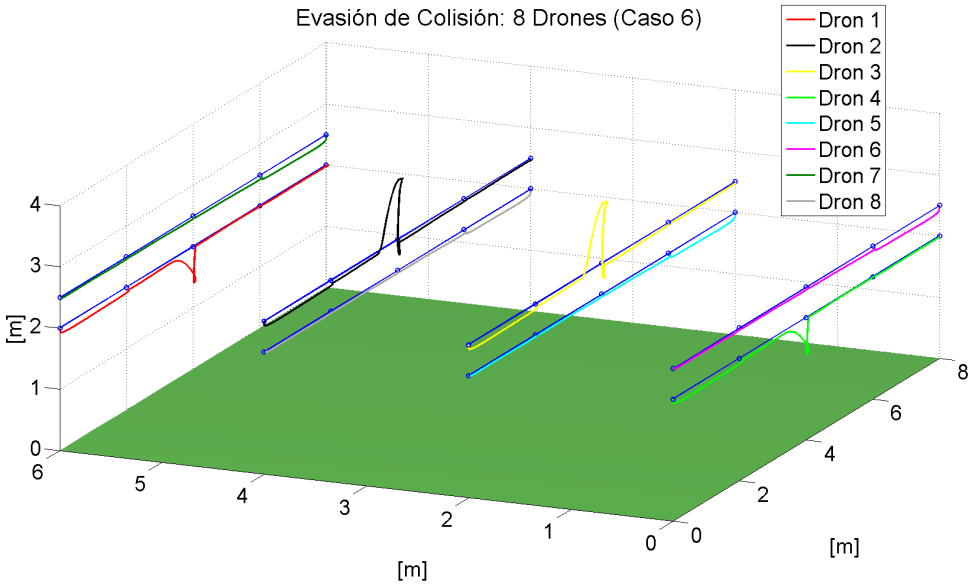

Figura 5.7 Evasión de colisión de la flota (Caso 6).

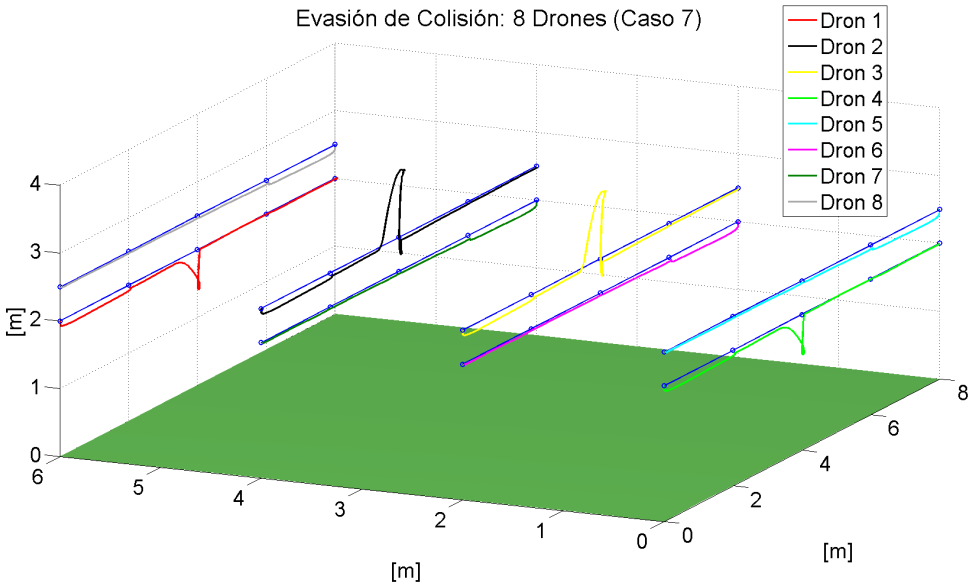

Figura 5.8 Evasión de colisión de la flota (Caso 7).

5.3 División del barrido

La división del barrido para el modelo de ocho drones, al igual que para el modelo con dos vehículos, se llevará a cabo en las funciones de generación de waypoints. Para que ésta sea más intuitiva y eficiente, se ha decidido compartir la división con respecto al eje y, dividiendo el terreno en dos mitades. Posteriormente, cada una de esas dos mitades será dividida en cuatro obteniendo así un total de ocho sectores, uno para cada dron.

La división según el eje x viene establecida según la longitud del terreno respecto a dicho eje. El primer sector comprende desde $x = 0$ a $x = dimx/4$, el segundo desde $x = dimx/4$ a $x = dimx/2$, el tercero desde $x = dimx/2$ a $x = 3\ dimx/4$ y el cuarto desde $x = 3\ dimx/4$ a $x = dimx$, siendo $dimx$ la longitud del terreno respecto al eje x.

A continuación, se muestran las funciones de generación de waypoints modificadas tanto para una superficie plana, código 5.5, como para una montañosa, código 5.6.

Código 5.5 Generación de waypoints en superficie plana para la flota de drones.

```
% ========================================================================
% ========    MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES    ========
% ========              ENRIQUE CARRASCO DOMÍNGUEZ            ========
% ========        TRABAJO DE FIN DE GRADO, SEVILLA 2019       ========
% ========================================================================
% ========     GENERADOR WAYPOINTS PLANO PARA OCHO DRONES    =========
% ========================================================================

function [waypoints_1,waypoints_2,waypoints_3,waypoints_4,waypoints_5,...
waypoints_6,waypoints_7,waypoints_8] = generador_waypoints_plano_8_drones...
(superficie,altura_barrido,precision)

dimx=superficie(1); dimy=superficie(2);

precisionx=precision(1); precisiony=precision(2);

puntosx=[0:precisionx:dimx]';
puntosxinv=[dimx:-precisionx:0]';
```

```
19 puntosy=[0:precisiony:dimy]';
20
21 % ================================================================
22 % ============= CÁLCULO WAYPOINTS PARA EL PRIMER DRON ==============
23 % ================================================================
24
25 p=1-mod(0.1*dimy,2); % Si la decena es par hay que sumar 1.
26
27 puntosx_1 = [0:precisionx:dimx/4]';
28 puntosxinv_1 = [dimx/4:-precisionx:0]';
29 puntosy_1=[0:precisiony:dimy/2-p]';
30
31 alpha_1=ones(size(puntosx_1));
32 e=1; wp_1=[];
33
34 for i=1:length(puntosy_1)
35     wp_1=[wp_1; abs(((e+1)/2)*puntosx_1+((e-1)/2)*puntosxinv_1) ,...
36         alpha_1*puntosy_1(i) ];
37     e=-e;
38 end
39
40 wp_1=[wp_1, ones(length(wp_1),1)*altura_barrido];
41 waypoints_1=wp_1;
42
43 % ================================================================
44 % ============= CÁLCULO WAYPOINTS PARA EL SEGUNDO DRON =============
45 % ================================================================
46
47 puntosx_2 = [dimx/4:precisionx:dimx/2]';
48 puntosxinv_2 = [dimx/2:-precisionx:dimx/4]';
49 puntosy_2=[0:precisiony:dimy/2-p]';
50
51 alpha_2=ones(size(puntosx_1));
52 e=1; wp_2=[];
53
54 for i=1:length(puntosy_2)
55     wp_2=[wp_2; abs(((e+1)/2)*puntosx_2+((e-1)/2)*puntosxinv_2) ,...
56         alpha_2*puntosy_2(i) ];
57     e=-e;
58 end
59
60 wp_2=[wp_2, ones(length(wp_2),1)*altura_barrido];
61 waypoints_2=wp_2;
62
63 % ================================================================
64 % ============= CÁLCULO WAYPOINTS PARA EL TERCER DRON ==============
65 % ================================================================
66
67 puntosx_3 = [dimx/2:precisionx:3*dimx/4]';
68 puntosxinv_3 = [3*dimx/4:-precisionx:dimx/2]';
69 puntosy_3=[0:precisiony:dimy/2-p]';
70
71 alpha_3=ones(size(puntosx_3));
72 e=1; wp_3=[];
73
74 for i=1:length(puntosy_3)
75     wp_3=[wp_3; abs(((e+1)/2)*puntosx_3+((e-1)/2)*puntosxinv_3) ,...
76         alpha_3*puntosy_3(i) ];
77     e=-e;
78 end
79
80 wp_3=[wp_3, ones(length(wp_3),1)*altura_barrido];
81 waypoints_3=wp_3;
82
83 % ================================================================
84 % ============= CÁLCULO WAYPOINTS PARA EL CUARTO DRON ==============
85 % ================================================================
86
```

```
puntosx_4 = [3*dimx/4:precisionx:dimx]';
puntosxinv_4 = [dimx:-precisionx:3*dimx/4]';
puntosy_4=[0:precisiony:dimy/2-p]';

alpha_4=ones(size(puntosx_4));
e=1; wp_4=[];

for i=1:length(puntosy_4)
    wp_4=[wp_4; abs(((e+1)/2)*puntosx_4+((e-1)/2)*puntosxinv_4) ,...
        alpha_4*puntosy_4(i) ];
    e=-e;
end

wp_4=[wp_4, ones(length(wp_4),1)*altura_barrido];
waypoints_4=wp_4;

% ========================================================================
% ============= CÁLCULO WAYPOINTS PARA EL QUINTO DRON ==============
% ========================================================================

puntosx_5 = [0:precisionx:dimx/4]';
puntosxinv_5 = [dimx/4:-precisionx:0]';
puntosy_5 = [dimy/2:precisiony:dimy]';

alpha_5=ones(size(puntosx_5));
e=1; wp_5=[];

for i=1:length(puntosy_5)
    wp_5=[wp_5; abs(((e+1)/2)*puntosx_5+((e-1)/2)*puntosxinv_5) ,...
        alpha_5*puntosy_5(i) ];
    e=-e;
end

wp_5=[wp_5, ones(length(wp_5),1)*altura_barrido];
waypoints_5=wp_5;

% ========================================================================
% ============= CÁLCULO WAYPOINTS PARA EL SEXTO DRON ===============
% ========================================================================

puntosx_6 = [dimx/4:precisionx:dimx/2]';
puntosxinv_6 = [dimx/2:-precisionx:dimx/4]';
puntosy_6 = [dimy/2:precisiony:dimy]';

alpha_6=ones(size(puntosx_6));
e=1; wp_6=[];

for i=1:length(puntosy_6)
    wp_6=[wp_6; abs(((e+1)/2)*puntosx_6+((e-1)/2)*puntosxinv_6) ,...
        alpha_6*puntosy_6(i) ];
    e=-e;
end

wp_6=[wp_6, ones(length(wp_6),1)*altura_barrido];
waypoints_6=wp_6;

% ========================================================================
% ============= CÁLCULO WAYPOINTS PARA EL SÉPTIMO DRON =============
% ========================================================================

puntosx_7 = [dimx/2:precisionx:3*dimx/4]';
puntosxinv_7 = [3*dimx/4:-precisionx:dimx/2]';
puntosy_7 = [dimy/2:precisiony:dimy]';

alpha_7=ones(size(puntosx_7));
e=1; wp_7=[];

for i=1:length(puntosy_7)
```

```
155     wp_7=[wp_7; abs(((e+1)/2)*puntosx_7+((e-1)/2)*puntosxinv_7) ,...
156         alpha_7*puntosy_7(i) ];
157     e=-e;
158 end
159
160 wp_7=[wp_7, ones(length(wp_7),1)*altura_barrido];
161 waypoints_7=wp_7;
162
163 % ======================================================================
164 % ============= CÁLCULO WAYPOINTS PARA EL OCTAVO DRON ==============
165 % ======================================================================
166
167 puntosx_8 = [3*dimx/4:precisionx:dimx]';
168 puntosxinv_8 = [dimx:-precisionx:3*dimx/4]';
169 puntosy_8 = [dimy/2:precisiony:dimy]';
170
171 alpha_8=ones(size(puntosx_8));
172 e=1; wp_8=[];
173
174 for i=1:length(puntosy_8)
175     wp_8=[wp_8; abs(((e+1)/2)*puntosx_8+((e-1)/2)*puntosxinv_8) ,...
176         alpha_8*puntosy_8(i) ];
177     e=-e;
178 end
179
180 wp_8=[wp_8, ones(length(wp_8),1)*altura_barrido];
181 waypoints_8=wp_8;
182
183 end
```

Código 5.6 Generación de waypoints en superficie montañosa para la flota de drones.

```
1  % ======================================================================
2  % ========   MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES   ========
3  % ========            ENRIQUE CARRASCO DOMÍNGUEZ            ========
4  % ========       TRABAJO DE FIN DE GRADO, SEVILLA 2019      ========
5  % ======================================================================
6  % ========   GENERADOR WAYPOINTS MONTAÑA PARA OCHO DRONES   ========
7  % ======================================================================
8
9  function [waypoints_1,waypoints_2,waypoints_3,waypoints_4,waypoints_5,...
10 waypoints_6,waypoints_7,waypoints_8] = generador_waypoints_montana_8_drones...
11 (superficie,z,altura_barrido,precision)
12
13 dimx=superficie(1); dimy=superficie(2);
14
15 precisionx=precision(1); precisiony=precision(2);
16
17 puntosx=[0:precisionx:dimx]';
18 puntosxinv=[dimx:-precisionx:0]';
19 puntosy=[0:precisiony:dimy]';
20
21 % ======================================================================
22 % ============= CÁLCULO WAYPOINTS PARA EL PRIMER DRON ==============
23 % ======================================================================
24
25 p=1-mod(0.1*dimy,2); % Si la decena es par hay que sumar 1.
26
27 puntosx_1 = [0:precisionx:dimx/4]';
28 puntosxinv_1 = [dimx/4:-precisionx:0]';
29 puntosy_1=[0:precisiony:dimy/2-p]';
30
31 alpha_1=ones(size(puntosx_1));
32 e=1; wp_1=[];
33
34 for i=1:length(puntosy_1)
```

```
    wp_1=[wp_1; abs(((e+1)/2)*puntosx_1+((e-1)/2)*puntosxinv_1) ,...
        alpha_1*puntosy_1(i) ];
    e=-e;
end

wp_1=[wp_1,zeros(length(wp_1),1)];

for i=1:length(wp_1)
    wp_1(i,3)=z(round(wp_1(i,2))+1,round(wp_1(i,1))+1)+altura_barrido-1;
end

wp_1=wp_1+ones(size(wp_1));
waypoints_1=wp_1;

% ====================================================================
% ============= CÁLCULO WAYPOINTS PARA EL SEGUNDO DRON =============
% ====================================================================

puntosx_2 = [dimx/4:precisionx:dimx/2]';
puntosxinv_2 = [dimx/2:-precisionx:dimx/4]';
puntosy_2=[0:precisiony:dimy/2-p]';

alpha_2=ones(size(puntosx_1));
e=1; wp_2=[];

for i=1:length(puntosy_2)
    wp_2=[wp_2; abs(((e+1)/2)*puntosx_2+((e-1)/2)*puntosxinv_2) ,...
        alpha_2*puntosy_2(i) ];
    e=-e;
end

wp_2=[wp_2,zeros(length(wp_2),1)];

for i=1:length(wp_2)
    wp_2(i,3)=z(round(wp_2(i,2))+1,round(wp_2(i,1))+1)+altura_barrido-1;
end

wp_2=wp_2+ones(size(wp_2));
waypoints_2=wp_2;

% ====================================================================
% ============= CÁLCULO WAYPOINTS PARA EL TERCER DRON ==============
% ====================================================================

puntosx_3 = [dimx/2:precisionx:3*dimx/4]';
puntosxinv_3 = [3*dimx/4:-precisionx:dimx/2]';
puntosy_3=[0:precisiony:dimy/2-p]';

alpha_3=ones(size(puntosx_3));
e=1; wp_3=[];

for i=1:length(puntosy_3)
    wp_3=[wp_3; abs(((e+1)/2)*puntosx_3+((e-1)/2)*puntosxinv_3) ,...
        alpha_3*puntosy_3(i) ];
    e=-e;
end

wp_3=[wp_3,zeros(length(wp_3),1)];

for i=1:length(wp_3)
    wp_3(i,3)=z(round(wp_3(i,2))+1,round(wp_3(i,1))+1)+altura_barrido-1;
end

wp_3=wp_3+ones(size(wp_3));
waypoints_3=wp_3;

% ====================================================================
% ============= CÁLCULO WAYPOINTS PARA EL CUARTO DRON ==============
```

```
% ====================================================================

puntosx_4 = [3*dimx/4:precisionx:dimx]';
puntosxinv_4 = [dimx:-precisionx:3*dimx/4]';
puntosy_4=[0:precisiony:dimy/2-p]';

alpha_4=ones(size(puntosx_4));
e=1; wp_4=[];

for i=1:length(puntosy_4)
    wp_4=[wp_4; abs(((e+1)/2)*puntosx_4+((e-1)/2)*puntosxinv_4) ,...
        alpha_4*puntosy_4(i) ];
    e=-e;
end

wp_4=[wp_4,zeros(length(wp_4),1)];

for i=1:length(wp_4)
    wp_4(i,3)=z(round(wp_4(i,2))+1,round(wp_4(i,1))+1)+altura_barrido-1;
end

wp_4=wp_4+ones(size(wp_4));
waypoints_4=wp_4;

% ====================================================================
% ============= CÁLCULO WAYPOINTS PARA EL QUINTO DRON ===============
% ====================================================================

puntosx_5 = [0:precisionx:dimx/4]';
puntosxinv_5 = [dimx/4:-precisionx:0]';
puntosy_5 = [dimy/2:precisiony:dimy]';

alpha_5=ones(size(puntosx_5));
e=1; wp_5=[];

for i=1:length(puntosy_5)
    wp_5=[wp_5; abs(((e+1)/2)*puntosx_5+((e-1)/2)*puntosxinv_5) ,...
        alpha_5*puntosy_5(i) ];
    e=-e;
end

wp_5=[wp_5,zeros(length(wp_5),1)];

for i=1:length(wp_5)
    wp_5(i,3)=z(round(wp_5(i,2))+1,round(wp_5(i,1))+1)+altura_barrido-1;
end

wp_5=wp_5+ones(size(wp_5));
waypoints_5=wp_5;

% ====================================================================
% ============= CÁLCULO WAYPOINTS PARA EL SEXTO DRON ================
% ====================================================================

puntosx_6 = [dimx/4:precisionx:dimx/2]';
puntosxinv_6 = [dimx/2:-precisionx:dimx/4]';
puntosy_6 = [dimy/2:precisiony:dimy]';

alpha_6=ones(size(puntosx_6));
e=1; wp_6=[];

for i=1:length(puntosy_6)
    wp_6=[wp_6; abs(((e+1)/2)*puntosx_6+((e-1)/2)*puntosxinv_6) ,...
        alpha_6*puntosy_6(i) ];
    e=-e;
end

wp_6=[wp_6,zeros(length(wp_6),1)];
```

```
171
172 for i=1:length(wp_6)
173     wp_6(i,3)=z(round(wp_6(i,2))+1,round(wp_6(i,1))+1)+altura_barrido-1;
174 end
175
176 wp_6=wp_6+ones(size(wp_6));
177 waypoints_6=wp_6;
178
179 % ======================================================================
180 % ============= CÁLCULO WAYPOINTS PARA EL SÉPTIMO DRON =============
181 % ======================================================================
182
183 puntosx_7 = [dimx/2:precisionx:3*dimx/4]';
184 puntosxinv_7 = [3*dimx/4:-precisionx:dimx/2]';
185 puntosy_7 = [dimy/2:precisiony:dimy]';
186
187 alpha_7=ones(size(puntosx_7));
188 e=1; wp_7=[];
189
190 for i=1:length(puntosy_7)
191     wp_7=[wp_7; abs(((e+1)/2)*puntosx_7+((e-1)/2)*puntosxinv_7) ,...
192         alpha_7*puntosy_7(i) ];
193     e=-e;
194 end
195
196 wp_7=[wp_7,zeros(length(wp_7),1)];
197
198 for i=1:length(wp_7)
199     wp_7(i,3)=z(round(wp_7(i,2))+1,round(wp_7(i,1))+1)+altura_barrido-1;
200 end
201
202 wp_7=wp_7+ones(size(wp_7));
203 waypoints_7=wp_7;
204
205 % ======================================================================
206 % ============= CÁLCULO WAYPOINTS PARA EL OCTAVO DRON ==============
207 % ======================================================================
208
209 puntosx_8 = [3*dimx/4:precisionx:dimx]';
210 puntosxinv_8 = [dimx:-precisionx:3*dimx/4]';
211 puntosy_8 = [dimy/2:precisiony:dimy]';
212
213 alpha_8=ones(size(puntosx_8));
214 e=1; wp_8=[];
215
216 for i=1:length(puntosy_8)
217     wp_8=[wp_8; abs(((e+1)/2)*puntosx_8+((e-1)/2)*puntosxinv_8) ,...
218         alpha_8*puntosy_8(i) ];
219     e=-e;
220 end
221
222 wp_8=[wp_8,zeros(length(wp_8),1)];
223
224 for i=1:length(wp_8)
225     wp_8(i,3)=z(round(wp_8(i,2))+1,round(wp_8(i,1))+1)+altura_barrido-1;
226 end
227
228 wp_8=wp_8+ones(size(wp_8));
229 waypoints_8=wp_8;
230
231 end
```

Para comprobar el correcto funcionamiento de ambas funciones, se generarán diversas superficies junto con los waypoints de las trayectorias generados, figuras 5.9 y 5.10.

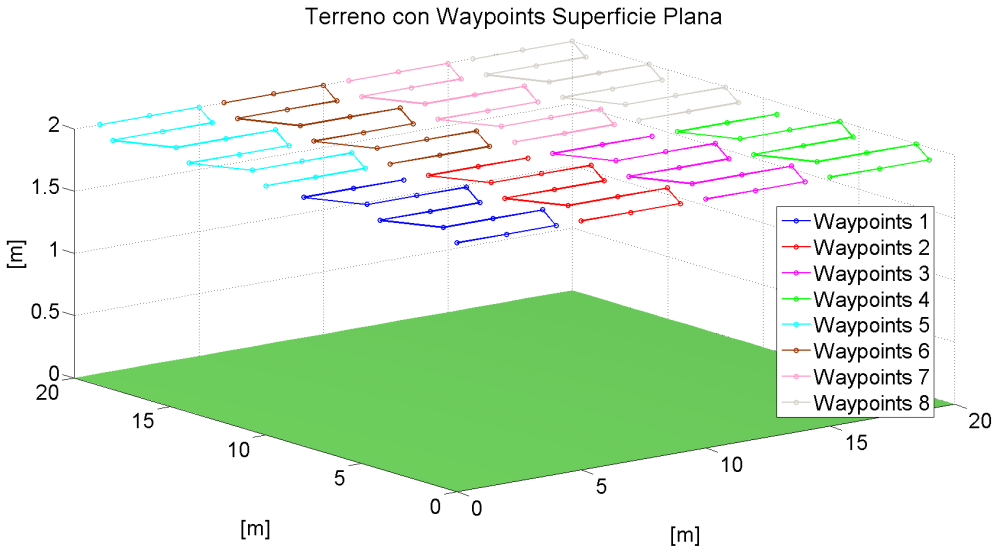

Figura 5.9 Terreno con waypoints para la flota en superficie plana.

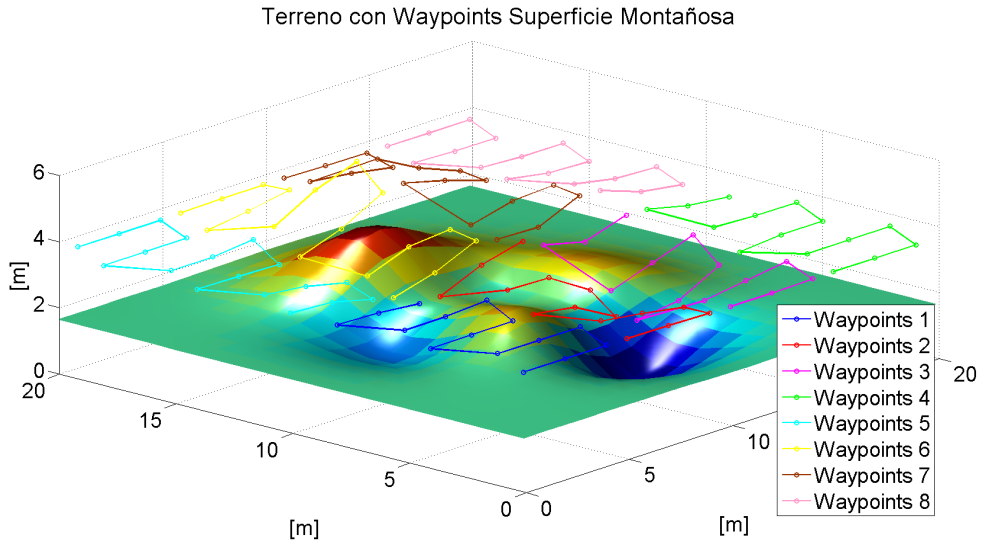

Figura 5.10 Terreno con waypoints para la flota en superficie montañosa.

5.4 Barrido del terreno en una superficie plana

Una vez realizado el desarrollo anterior se simula el modelo de ocho drones en un entorno definido por una superficie plana, figura 5.11. Posteriormente, se realizan varias simulaciones anotando el tiempo transcurrido para demostrar que el tiempo empleado para barrer completamente la superficie es aproximadamente la octava parte del tiempo por parte de un único dron y cuatro veces menor que el transcurrido con dos drones, tabla 5.1.

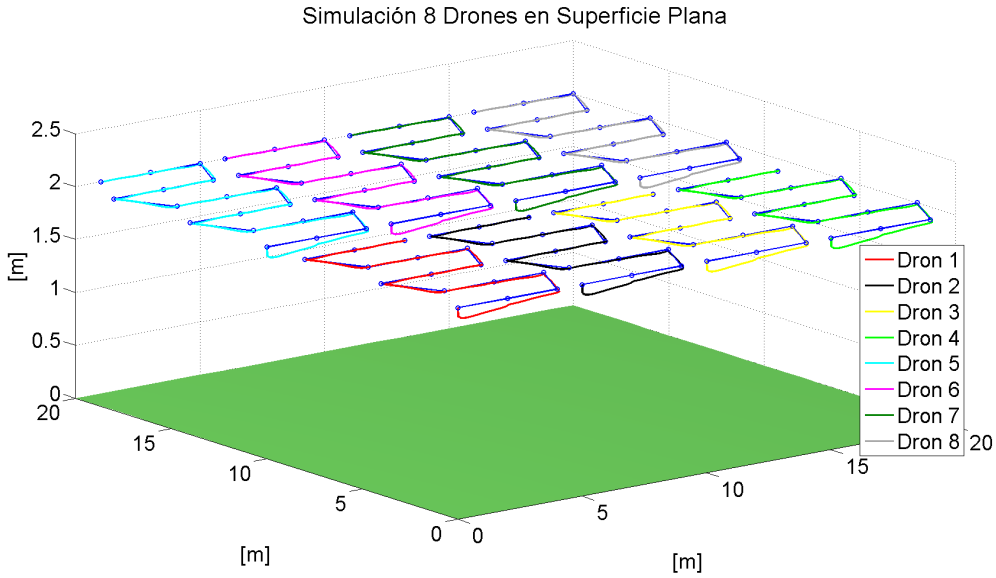

Figura 5.11 Simulación de la flota de drones en superficie plana.

Tabla 5.1 Tiempo de barrido en superficie plana con la flota de drones.

Superficie [m]	Precisión [m]	Tiempo [s]	R. frente un dron	R. frente dos drones
20 x 20	2 x 2	23.73	85.71 %	73.63 %
50 x 50	2 x 2	124.64	86.63 %	73.28 %
50 x 50	1 x 2	209.40	87.20 %	74.27 %
40 x 20	1 x 1	148.65	85.97 %	73.19 %

5.5 Barrido del terreno en una superficie montañosa

De manera similar al apartado anterior, se simula el modelo de ocho drones cambiando el tipo de superficie por una con relieve, figura 5.12. Se toman las medidas pertinentes demostrando la reducción en el tiempo de barrido tabla 5.2.

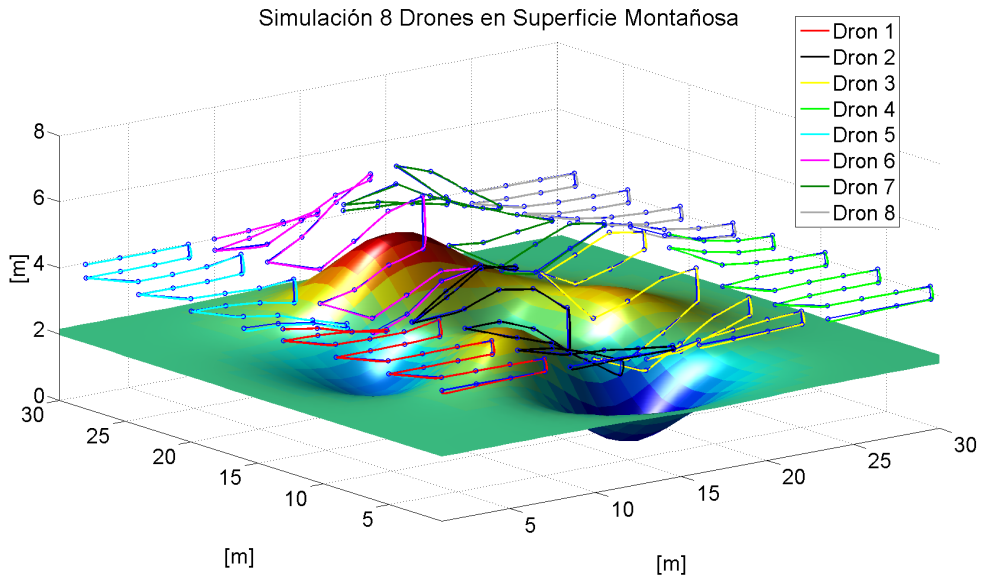

Figura 5.12 Simulación de la flota de drones en superficie montañosa.

Tabla 5.2 Tiempo de barrido en superficie montañosa con la flota de drones.

Superficie [m]	Precisión [m]	Tiempo [s]	R. frente un dron	R. frente dos drones
20 x 20	2 x 2	22.81	85.67 %	73.56 %
50 x 50	2 x 2	125.13	86.58 %	73.15 %
50 x 50	1 x 2	210.88	87.14 %	74.30 %
30 x 30	1 x 2	158.83	86.62 %	72.36 %

5.6 Elección de dron para un punto

El proceso de elección de un dron para un punto con una flota de ocho drones es idéntico al de dos drones. Habría que modificar la función, añadiendo el cálculo de las distancias de cada uno de los vehículos respecto al punto objetivo. Para hacer la selección del dron que se dirigirá al punto deseado, se hace uso de la función *min.m* de MATLAB que, si se ejecuta con dos salidas, da como resultado el valor mínimo y la posición dentro del vector donde se encuentra ese valor. Para ello, una vez calculadas todas las distancias de cada uno de los vehículos respecto al punto objetivo, se disponen en el vector denominado *distancias_al_punto*. Si el interruptor se encuentra en la posición uno, se calcula el valor mínimo del vector y la posición donde se encuentra que corresponde al dron más cercano, dirigiéndolo hasta el punto objetivo.

Al igual que para el modelo de dos drones, para no perturbar en exceso el desarrollo del barrido y no interferir en las trayectorias de los demás vehículos, se impone la condición adicional de que el dron acuda al punto objetivo únicamente si se encuentra a una distancia menor o igual a dos metros, código 5.7.

Código 5.7 Sistema elección de un dron de la flota para un punto.

```
% ====================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========            ENRIQUE CARRASCO DOMÍNGUEZ           =========
% =========      TRABAJO DE FIN DE GRADO, SEVILLA 2019      =========
% ====================================================================
% =========      FUNCIÓN ELECCIÓN DRON PARA UN PUNTO        =========
% ====================================================================

function [rutas_c, distancias_al_punto] = eleccion_dron_punto(...
          posiciones_actuales,rutas,punto_objetivo,interruptor)

rutas_c = rutas;

posicion_actual_1 = posiciones_actuales(1:3);
posicion_actual_2 = posiciones_actuales(4:6);
posicion_actual_3 = posiciones_actuales(7:9);
posicion_actual_4 = posiciones_actuales(10:12);
posicion_actual_5 = posiciones_actuales(13:15);
posicion_actual_6 = posiciones_actuales(16:18);
posicion_actual_7 = posiciones_actuales(19:21);
posicion_actual_8 = posiciones_actuales(22:24);

x1=posicion_actual_1(1); y1=posicion_actual_1(2); z1=posicion_actual_1(3);
x2=posicion_actual_2(1); y2=posicion_actual_2(2); z2=posicion_actual_2(3);
x3=posicion_actual_3(1); y3=posicion_actual_3(2); z3=posicion_actual_3(3);
x4=posicion_actual_4(1); y4=posicion_actual_4(2); z4=posicion_actual_4(3);
x5=posicion_actual_5(1); y5=posicion_actual_5(2); z5=posicion_actual_5(3);
x6=posicion_actual_6(1); y6=posicion_actual_6(2); z6=posicion_actual_6(3);
x7=posicion_actual_7(1); y7=posicion_actual_7(2); z7=posicion_actual_7(3);
x8=posicion_actual_8(1); y8=posicion_actual_8(2); z8=posicion_actual_8(3);

distancias_al_punto = zeros(1,8);

distancias_al_punto(1) = abs(sqrt((x1-punto_objetivo(1))^2+(y1-...
                         punto_objetivo(2))^2+(z1-punto_objetivo(3))^2));
distancias_al_punto(2) = abs(sqrt((x2-punto_objetivo(1))^2+(y2-...
                         punto_objetivo(2))^2+(z2-punto_objetivo(3))^2));
distancias_al_punto(3) = abs(sqrt((x3-punto_objetivo(1))^2+(y3-...
                         punto_objetivo(2))^2+(z3-punto_objetivo(3))^2));
distancias_al_punto(4) = abs(sqrt((x4-punto_objetivo(1))^2+(y4-...
                         punto_objetivo(2))^2+(z4-punto_objetivo(3))^2));
distancias_al_punto(5) = abs(sqrt((x5-punto_objetivo(1))^2+(y5-...
                         punto_objetivo(2))^2+(z5-punto_objetivo(3))^2));
distancias_al_punto(6) = abs(sqrt((x6-punto_objetivo(1))^2+(y6-...
                         punto_objetivo(2))^2+(z6-punto_objetivo(3))^2));
distancias_al_punto(7) = abs(sqrt((x7-punto_objetivo(1))^2+(y7-...
                         punto_objetivo(2))^2+(z7-punto_objetivo(3))^2));
distancias_al_punto(8) = abs(sqrt((x8-punto_objetivo(1))^2+(y8-...
                         punto_objetivo(2))^2+(z8-punto_objetivo(3))^2));

if interruptor == 1
    [valormin,posicion]=min(distancias_al_punto);
    if valormin<=2
       rutas_c(1+(posicion-1)*3:posicion*3) = punto_objetivo;
    end
end
```

Se simula el modelo de la flota de ocho drones implementando la nueva función mostrándose los resultados en las figuras 5.13 y 5.14. En esta última, se puede apreciar como los drones han alterado su trayectoria para evitar estar en proximidad con los demás.

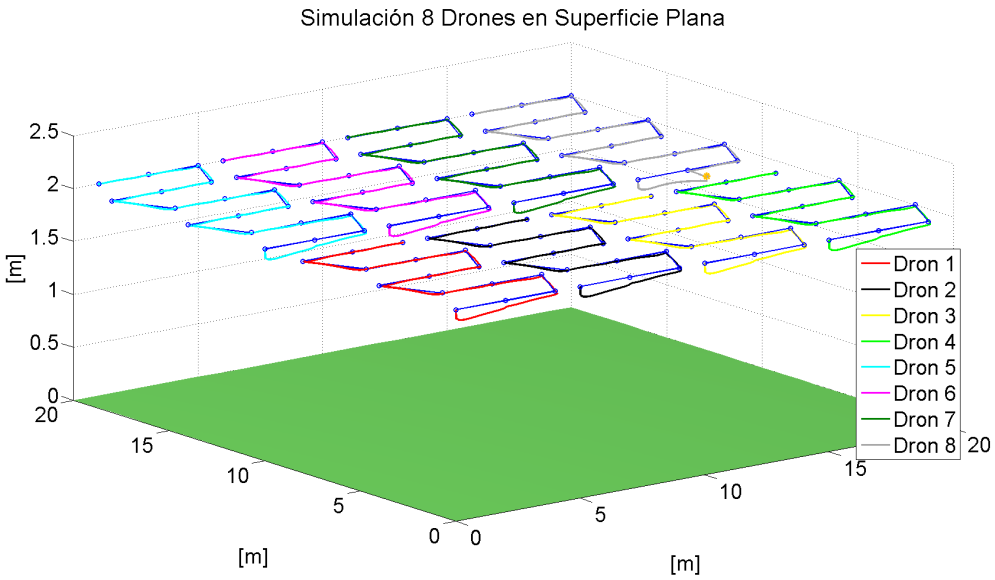

Figura 5.13 Elección de un dron de la flota para un punto en superficie plana.

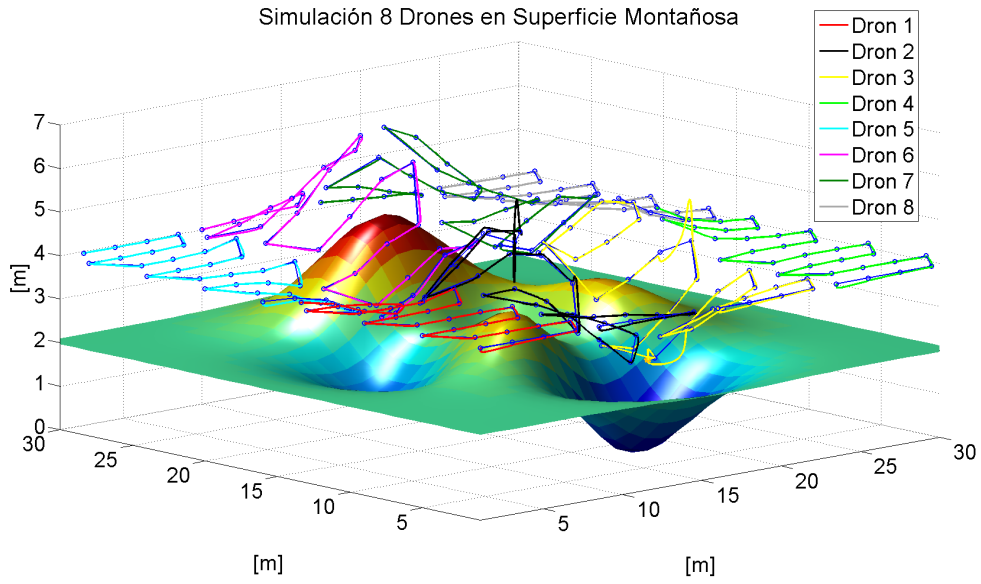

Figura 5.14 Elección de un dron de la flota para un punto en superficie montañosa.

5.7 Problema del viajante para la flota de drones

De manera similar a la simulación con dos drones, se concluirá el desarrollo con un barrido cumpliendo el problema del viajante. Para ello es necesario modificar la función de asignación de puntos a cada dron para que tenga en cuenta la existencia de los nuevos drones y reparta de manera equitativa los puntos aleatorios generados. La función de selección de puntos, código 5.8, tiene como entradas las posiciones iniciales de todos los vehículos de la flota y la matriz de puntos aleatorios. Como salida esta función tiene los puntos a visitar por cada dron.

Código 5.8 Función selección de puntos para el problema del viajante con la flota de drones.

```
% ====================================================================
% =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
% =========           ENRIQUE CARRASCO DOMÍNGUEZ            =========
% =========    TRABAJO DE FIN DE GRADO, SEVILLA 2019       =========
% ====================================================================
% ========        SELECCIÓN DE PUNTOS PARA LA FLOTA        =========
% ====================================================================

function [puntos_1,puntos_2,puntos_3,puntos_4,puntos_5,puntos_6,...
puntos_7,puntos_8]=seleccion_puntos_salesman_8(posini_1,posini_2,...
posini_3,posini_4,posini_5,posini_6,posini_7,posini_8,mtxpuntos)

[m,n]=size(mtxpuntos);
puntos_1 = []; puntos_2 = []; puntos_3 = []; puntos_4 = [];
puntos_5 = []; puntos_6 = []; puntos_7 = []; puntos_8 = [];
l1=0; l2=0; l3=0; l4=0; l5=0; l6=0; l7=0; l8=0; e=0;

while e==0

    for i=1:m
        dist_1(i) = sqrt((posini_1(1)-mtxpuntos(i,1))^2+(posini_1(2)-...
                    mtxpuntos(i,2))^2+(posini_1(3)-mtxpuntos(i,3))^2);
    end
    [~,pos]=min(dist_1); puntos_1 = [puntos_1; mtxpuntos(pos,:)];
    mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
    [l1,~]=size(puntos_1);

    if l1+l2+l3+l4+l5+l6+l7+l8==m
    break
    end

    for i=1:m
        dist_2(i) = sqrt((posini_2(1)-mtxpuntos(i,1))^2+(posini_2(2)-...
                    mtxpuntos(i,2))^2+(posini_2(3)-mtxpuntos(i,3))^2);
    end
    [~,pos]=min(dist_2); puntos_2 = [puntos_2; mtxpuntos(pos,:)];
    mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
    [l2,~]=size(puntos_2);

    if l1+l2+l3+l4+l5+l6+l7+l8==m
    break
    end

    for i=1:m
        dist_3(i) = sqrt((posini_3(1)-mtxpuntos(i,1))^2+(posini_3(2)-...
                    mtxpuntos(i,2))^2+(posini_3(3)-mtxpuntos(i,3))^2);
    end
    [~,pos]=min(dist_3); puntos_3 = [puntos_3; mtxpuntos(pos,:)];
    mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
    [l3,~]=size(puntos_3);

    if l1+l2+l3+l4+l5+l6+l7+l8==m
    break
    end

    for i=1:m
        dist_4(i) = sqrt((posini_4(1)-mtxpuntos(i,1))^2+(posini_4(2)-...
                    mtxpuntos(i,2))^2+(posini_4(3)-mtxpuntos(i,3))^2);
    end
    [~,pos]=min(dist_4); puntos_4 = [puntos_4; mtxpuntos(pos,:)];
    mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
    [l4,~]=size(puntos_4);

    if l1+l2+l3+l4+l5+l6+l7+l8==m
    break
```

```
66      end
67
68      for i=1:m
69          dist_5(i) = sqrt((posini_5(1)-mtxpuntos(i,1))^2+(posini_5(2)-...
70                      mtxpuntos(i,2))^2+(posini_5(3)-mtxpuntos(i,3))^2);
71      end
72      [~,pos]=min(dist_5); puntos_5 = [puntos_5; mtxpuntos(pos,:)];
73      mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
74      [l5,~]=size(puntos_5);
75
76      if l1+l2+l3+l4+l5+l6+l7+l8==m
77      break
78      end
79
80      for i=1:m
81          dist_6(i) = sqrt((posini_6(1)-mtxpuntos(i,1))^2+(posini_6(2)-...
82                      mtxpuntos(i,2))^2+(posini_6(3)-mtxpuntos(i,3))^2);
83      end
84      [~,pos]=min(dist_6); puntos_6 = [puntos_6; mtxpuntos(pos,:)];
85      mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
86      [l6,~]=size(puntos_6);
87
88      if l1+l2+l3+l4+l5+l6+l7+l8==m
89      break
90      end
91
92      for i=1:m
93          dist_7(i) = sqrt((posini_7(1)-mtxpuntos(i,1))^2+(posini_7(2)-...
94                      mtxpuntos(i,2))^2+(posini_7(3)-mtxpuntos(i,3))^2);
95      end
96      [~,pos]=min(dist_7); puntos_7 = [puntos_7; mtxpuntos(pos,:)];
97      mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
98      [l7,~]=size(puntos_7);
99
100     if l1+l2+l3+l4+l5+l6+l7+l8==m
101     break
102     end
103
104     for i=1:m
105         dist_8(i) = sqrt((posini_8(1)-mtxpuntos(i,1))^2+(posini_8(2)-...
106                     mtxpuntos(i,2))^2+(posini_8(3)-mtxpuntos(i,3))^2);
107     end
108     [~,pos]=min(dist_8); puntos_8 = [puntos_8; mtxpuntos(pos,:)];
109     mtxpuntos(pos,:) = mtxpuntos(pos,:).*1e6;
110     [l8,~]=size(puntos_8);
111
112     if l1+l2+l3+l4+l5+l6+l7+l8==m
113     break
114     end
115
116 end
117 end
```

Se ejecuta el código 5.9 para comprobar el correcto funcionamiento de las funciones. Además, se generan como resultado las figuras 5.15, 5.16 y 5.17.

Código 5.9 Simulación del problema del viajante para la flota de drones.

```
1  % ================================================================
2  % =========  MODELADO Y SIMULACIÓN DE UNA FLOTA DE DRONES  =========
3  % =========           ENRIQUE CARRASCO DOMÍNGUEZ            =========
4  % =========    TRABAJO DE FIN DE GRADO, SEVILLA 2019       =========
5  % ================================================================
6  % ========   MODELO FLOTA DE DRONES PROBLEMA DEL VIAJANTE  =========
7  % ================================================================
8
```

```
punto_objetivo = [0 0 0]; altura_barrido = 2;
superficie = [20 20]; dimx=superficie(1); dimy=superficie(2);
x=0:dimx; y=0:dimy; z=zeros(length(x),length(y)); e=1.5;

if dimx>dimy, pt=e*dimx; else pt=e*dimy; end

posini_1 = [0 0 2]; posini_2 = [dimx/4 0 2]; posini_3 = [3*dimx/4 0 2];
posini_4 = [dimx 0 2];posini_5 = [0 dimy 2]; posini_6 = [dimx/4 dimy 2];
posini_7 = [3*dimx/4 dimy 2]; posini_8 = [dimx dimy 2];
mtxpuntos = [posini_1;posini_2;posini_3;posini_4;posini_5;posini_6;...
posini_7;posini_8; round(dimx*rand(pt,1)), round(dimy*rand(pt,1)),...
(altura_barrido+0.1*round(0.5*(10*rand(pt,1))))];

[puntos_1,puntos_2,puntos_3,puntos_4,puntos_5,puntos_6,puntos_7,...
puntos_8]=seleccion_puntos_salesman_8(posini_1,posini_2,posini_3,...
posini_4,posini_5,posini_6,posini_7,posini_8,mtxpuntos);

userConfig = struct('xy',puntos_1); [waypoints_1, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_2); [waypoints_2, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_3); [waypoints_3, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_4); [waypoints_4, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_5); [waypoints_5, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_6); [waypoints_6, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_7); [waypoints_7, ~] = tsp_nn(userConfig);
userConfig = struct('xy',puntos_8); [waypoints_8, ~] = tsp_nn(userConfig);

waypoints_1 = [posini_1; waypoints_1]; waypoints_2 = [posini_2; waypoints_2];
waypoints_3 = [posini_3; waypoints_3]; waypoints_4 = [posini_4; waypoints_4];
waypoints_5 = [posini_5; waypoints_5]; waypoints_6 = [posini_6; waypoints_6];
waypoints_7 = [posini_7; waypoints_7]; waypoints_8 = [posini_8; waypoints_8];

waypoints_1 = adiccion_waypoints(waypoints_1);
waypoints_2 = adiccion_waypoints(waypoints_2);
waypoints_3 = adiccion_waypoints(waypoints_3);
waypoints_4 = adiccion_waypoints(waypoints_4);
waypoints_5 = adiccion_waypoints(waypoints_5);
waypoints_6 = adiccion_waypoints(waypoints_6);
waypoints_7 = adiccion_waypoints(waypoints_7);
waypoints_8 = adiccion_waypoints(waypoints_8);

destino_final_1 = [waypoints_1(length(waypoints_1),:)];
destino_final_2 = [waypoints_2(length(waypoints_2),:)];
destino_final_3 = [waypoints_3(length(waypoints_3),:)];
destino_final_4 = [waypoints_4(length(waypoints_4),:)];
destino_final_5 = [waypoints_5(length(waypoints_5),:)];
destino_final_6 = [waypoints_6(length(waypoints_6),:)];
destino_final_7 = [waypoints_7(length(waypoints_7),:)];
destino_final_8 = [waypoints_8(length(waypoints_8),:)];

posini_1 = [waypoints_1(1,:)]; posini_2 = [waypoints_2(1,:)];
posini_3 = [waypoints_3(1,:)]; posini_4 = [waypoints_4(1,:)];
posini_5 = [waypoints_5(1,:)]; posini_6 = [waypoints_6(1,:)];
posini_7 = [waypoints_7(1,:)]; posini_8 = [waypoints_8(1,:)];

close all

sim('quad_control_8_drones')

figure(1)
plot3(mtxpuntos(:,1),mtxpuntos(:,2),mtxpuntos(:,3),'*',...
'LineWidth',1.25); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none');

camlight left; grid on;
lighting phong

title('Puntos a Visitar Problema del Viajante Flota','FontSize', 24);
xlabel('[m]','FontSize', 22);
```

```
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);

figure(2)
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;
plot3(mtxpuntos(:,1),mtxpuntos(:,2),mtxpuntos(:,3),'*',...
'LineWidth',1.25)

camlight left; grid on;
lighting phong

title('Waypoints Problema del Viajante Flota','FontSize', 24);
xlabel('[m]','FontSize', 22);
ylabel('[m]','FontSize', 22);
zlabel('[m]','FontSize', 22);
set(gca, 'FontSize', 22);
xlim([0 superficie(1)]);
ylim([0 superficie(2)]);
legend('Waypoints 1','Waypoints 2','Waypoints 3','Waypoints 4',...
       'Waypoints 5','Waypoints 6','Waypoints 7','Waypoints 8');

figure(3)
plot3(x1,y1,z1,'r','LineWidth',2); hold on;
plot3(x2,y2,z2,'k','LineWidth',2); hold on;
plot3(x3,y3,z3,'y','LineWidth',2); hold on;
plot3(x4,y4,z4,'g','LineWidth',2); hold on;
plot3(x5,y5,z5,'c','LineWidth',2); hold on;
plot3(x6,y6,z6,'m','LineWidth',2); hold on;
plot3(x7,y7,z7,'Color',[0 0.5 0],'LineWidth',2); hold on;
plot3(x8,y8,z8,'Color',[0.66, 0.66, 0.66],'LineWidth',2); hold on;
surf(x,y,z','FaceColor',[.53 1 .45],'EdgeColor','none'); hold on;...
plot3(waypoints_1(:,1),waypoints_1(:,2),waypoints_1(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_2(:,1),waypoints_2(:,2),waypoints_2(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_3(:,1),waypoints_3(:,2),waypoints_3(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_4(:,1),waypoints_4(:,2),waypoints_4(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_5(:,1),waypoints_5(:,2),waypoints_5(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_6(:,1),waypoints_6(:,2),waypoints_6(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_7(:,1),waypoints_7(:,2),waypoints_7(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(waypoints_8(:,1),waypoints_8(:,2),waypoints_8(:,3),'o-',...
'LineWidth',1.25); hold on;
plot3(mtxpuntos(:,1),mtxpuntos(:,2),mtxpuntos(:,3),'*',...
'LineWidth',1.25)
```

```
145
146 camlight left; grid on;
147 lighting phong
148
149 title('Problema del Viajante Flota','FontSize', 24);
150 xlabel('[m]','FontSize', 22);
151 ylabel('[m]','FontSize', 22);
152 zlabel('[m]','FontSize', 22);
153 set(gca, 'FontSize', 22);
154 xlim([0 superficie(1)]);
155 ylim([0 superficie(2)]);
156 legend('Dron 1','Dron 2','Dron 3','Dron 4','Dron 5','Dron 6',...
157        'Dron 7','Dron 8');
```

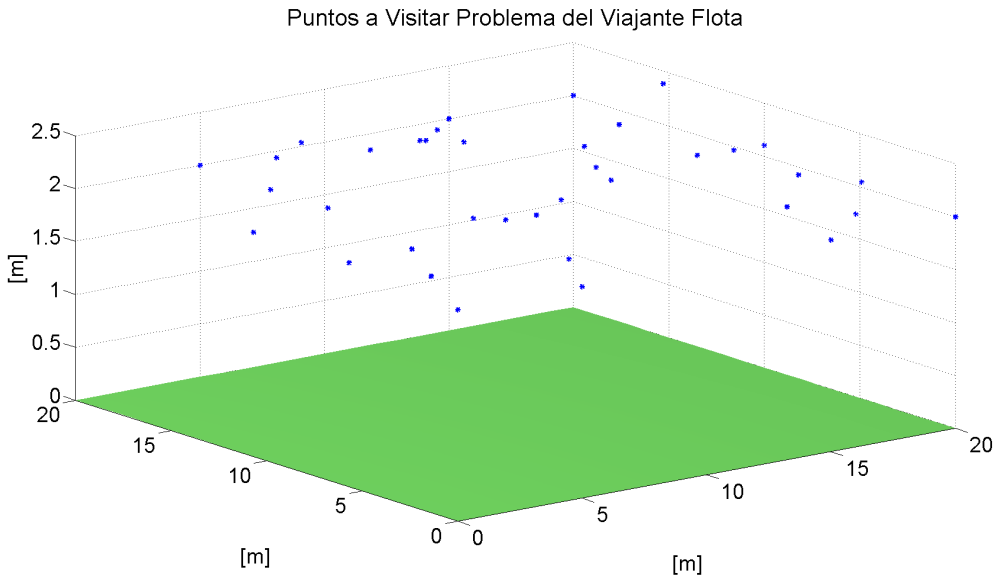

Figura 5.15 Puntos aleatorios a visitar por la flota de drones.

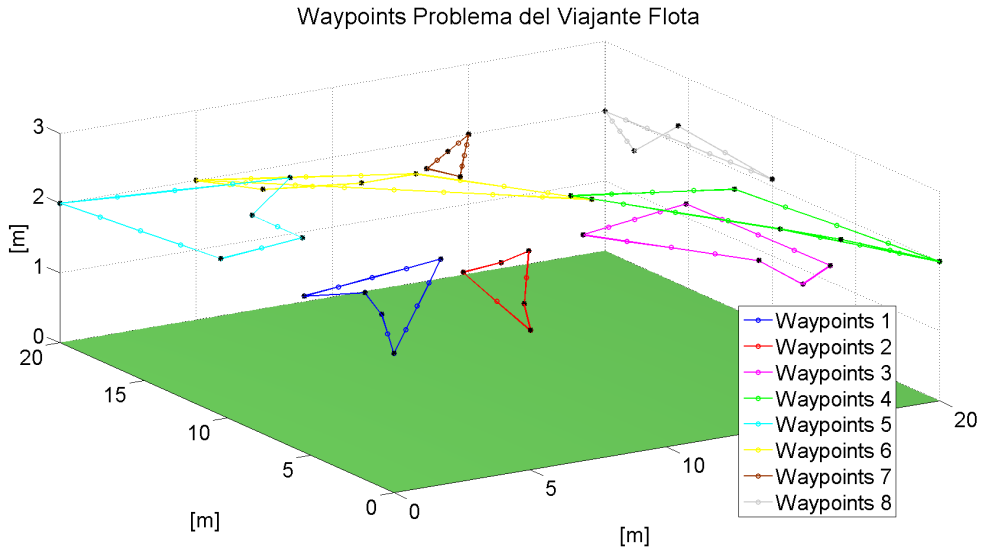

Figura 5.16 Waypoints generados para cada uno de los drones de la flota.

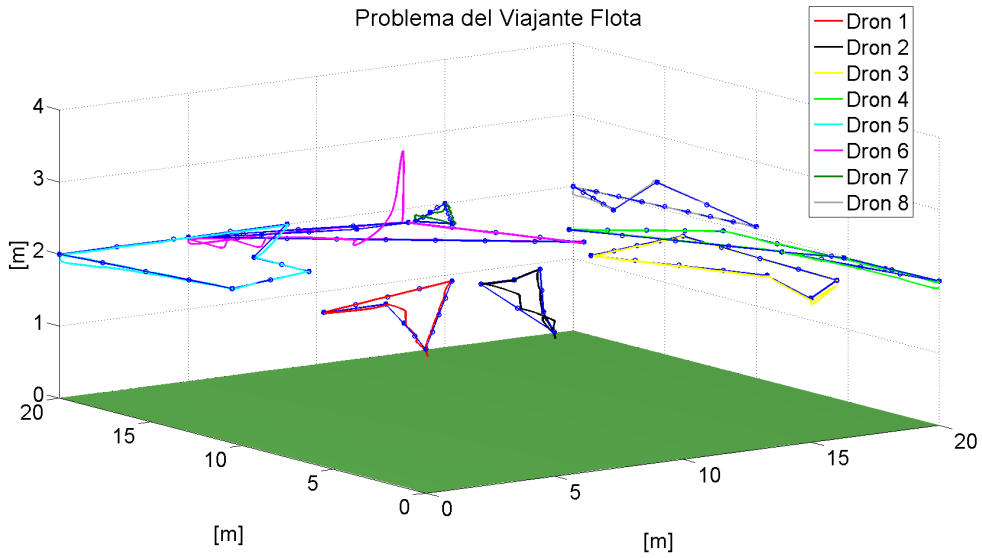

Figura 5.17 Simulación del problema del viajante para la flota de drones.

El modelo de SimuLink que engloba todo el desarrollo de la flota se muestra en la figura 5.18

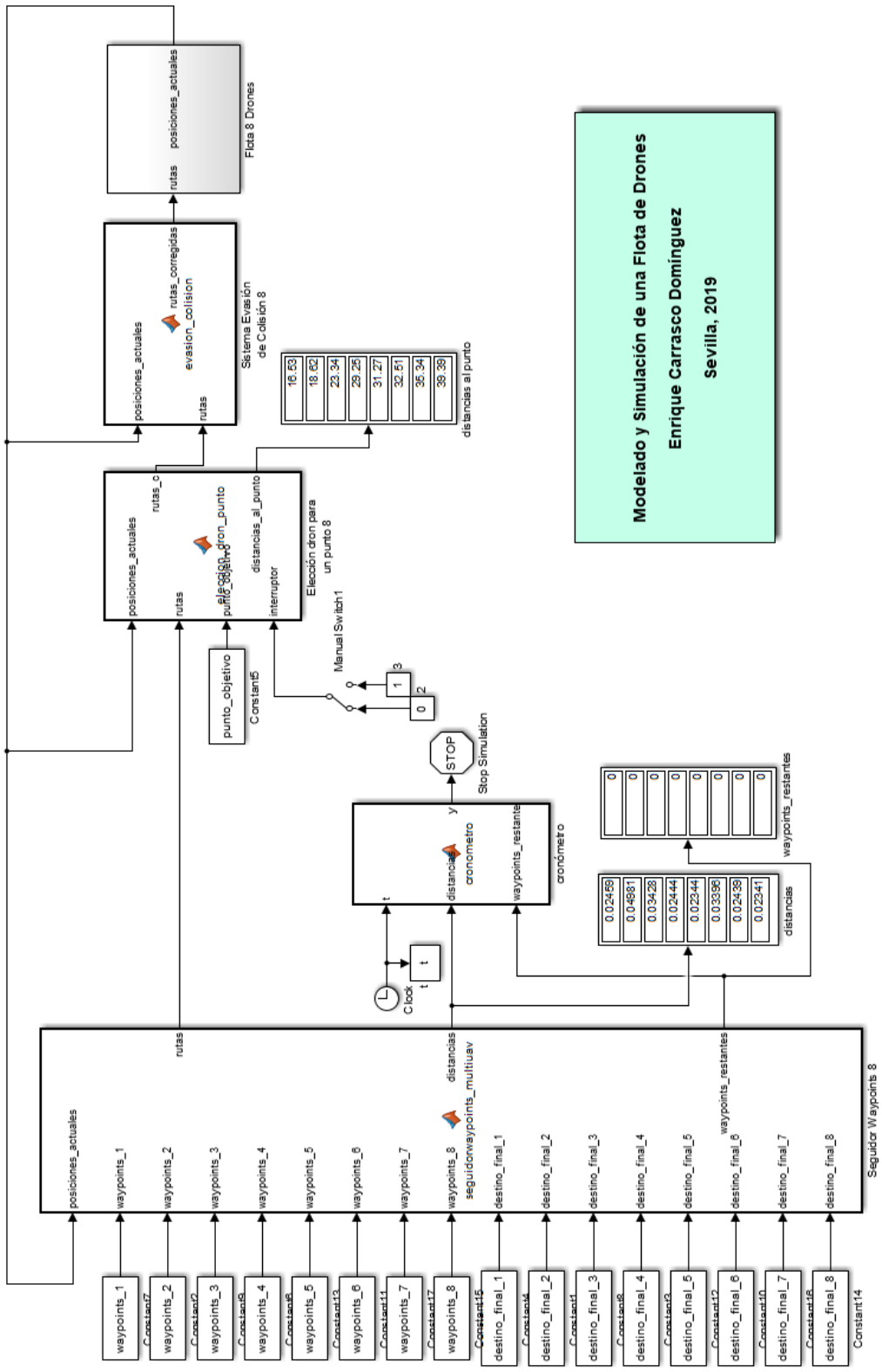

Figura 5.18 Modelo de SimuLink para la flota de ocho drones.

6 Conclusiones

El proyecto concluye afirmando el cumplimiento de los objetivos planteados al comienzo, garantizar la seguridad y la eficacia en las trayectorias de los drones que conforman la flota. Se aprecia que cuanto mayor es el número de drones, menor es el tiempo en recorrer las superficies por lo que se deja la puerta abierta a que, en una futura aplicación real del proyecto, se añadan tantos drones como la actividad así lo requiera.

Partiendo de la base que genera este proyecto, en un futuro se podrían realizar mejoras tanto en el código como en el modelo del dron empleado además de la implementación de datos de geolocalización o de tipo redes móviles como el 5G que permitirían calcular con precisión la posición de cada vehículo y así facilitar la transición del proyecto desde la simulación hasta el mundo real.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] R. M.; Chvátal V.; Cook W. J. Applegate, D. L.; Bixby, *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, 2007.

[2] Kirk J., *Multi-Quadrotor Control using Simulink and SimMechanics*, Mayo 2014.

[3] Abdelkader Zahana M., *Multi-Quadrotor Control using Simulink and SimMechanics*, Marzo 2015.

[4] MathWorks, *Página web oficial de MathWorks*, Junio 2019.

[5] Ferández Camacho E. y Limón Marruedo D., *Contenido de la asignatura Sistemas de Control y Guiado*, Universidad de Sevilla, 2019.