Adaptive and Autonomous Aerospace Systems

School of Industrial and Information Engineering - Aeronautical Engineering
Davide Invernizzi – Department of Aerospace Science and Technology

Part 3: Autonomous systems

Lab experience 3: Trajectory generation for quadrotors with obstacles

POLITECNICO
MILANO 1863

ASCL
AEROSPACE SYSTEMS & CONTROL LABORATORY

Outline

- This laboratory shows how to plan a minimum snap trajectory (minimum control effort) for a multirotor unmanned aerial vehicle (UAV) from a start pose to a goal pose on a 3D map by using the optimized rapidly exploring random tree (RRT*) path planner.

- The first step is to set up a 3D map of a given enviroment, provide a start pose and goal pose.

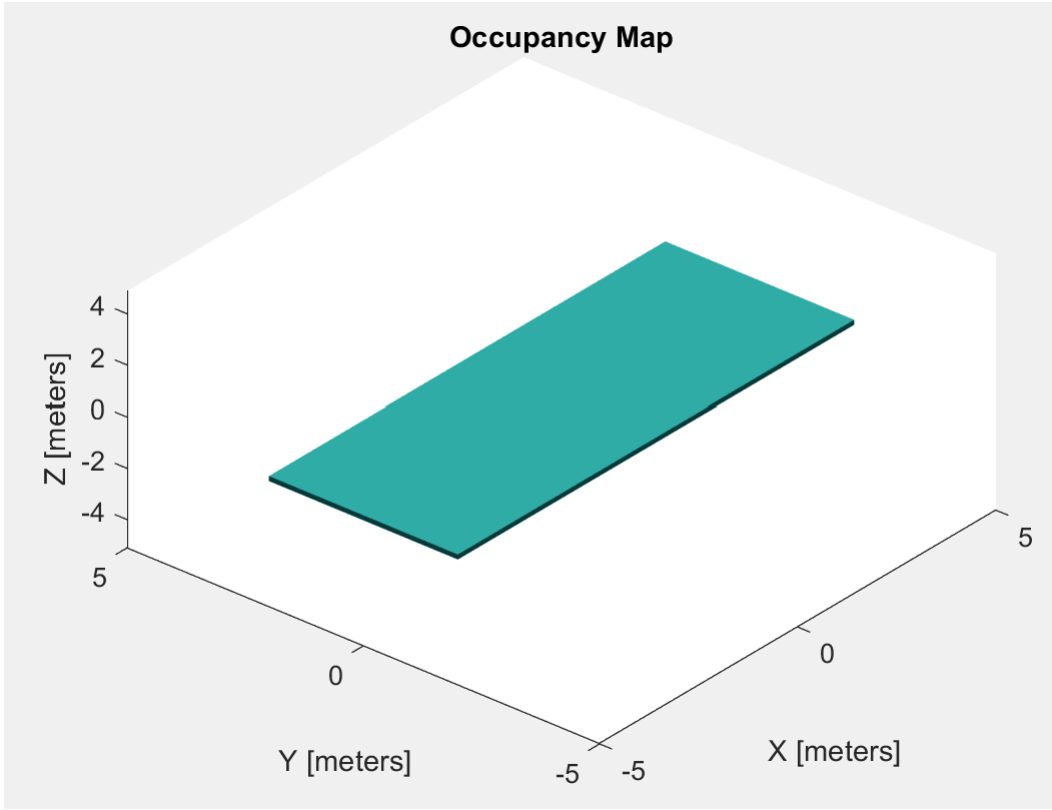
- Then, to plan a path with RRT* using 3D straight line connections.

- Finally, to fit a minimum snap trajectory through the obtained waypoints.

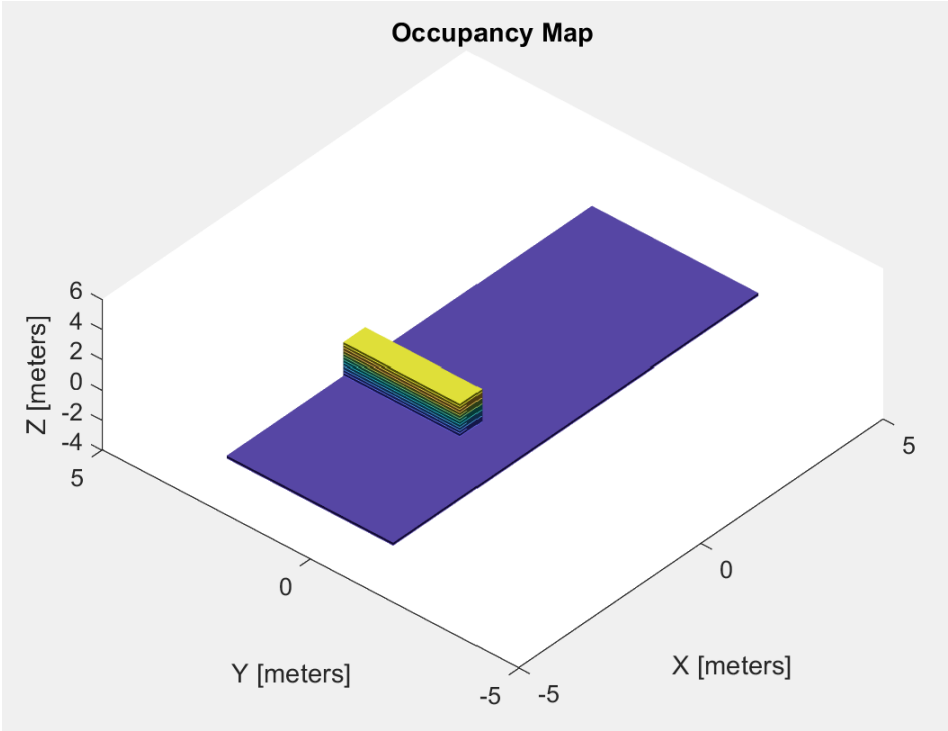
TASK: implement in MATLAB the code reported in the next slides and try tuning the RRT* algorithm to achieve **fast** but also **smooth** trajectories.

```
addpath map_tools

%Configure the random number generator for repeatable result.

rng(“default")

%% Map geometry and grid resolution

r = 0.2; % drone radius: parameter to be used in inflate map routine

% Mapsize
mapsize.x = 10; %[m]
length_ground = mapsize.x;

mapsize.y = 4; %[m]
width_ground = mapsize.y;


% Map resolution
res = 0.2; %[m]
grid_size = res;
height_ground = res;
```

```
%% Create map

map3D = occupancyMap3D(1/res);  %1/res cells per meter

map3D.FreeThreshold = map3D.OccupiedThreshold; % set to 0.65; unknown spaces (occupancy 0.5) considered as free

% Define Ground Plane
for x = -length_ground/2+res/2:res:length_ground/2-res/2
for y = -width_ground/2+res/2:res:width_ground/2-res/2
for z = -height_ground
xyz = [x y z];
setOccupancy(map3D, xyz, 1)
end
end
end

show(map3D)
```

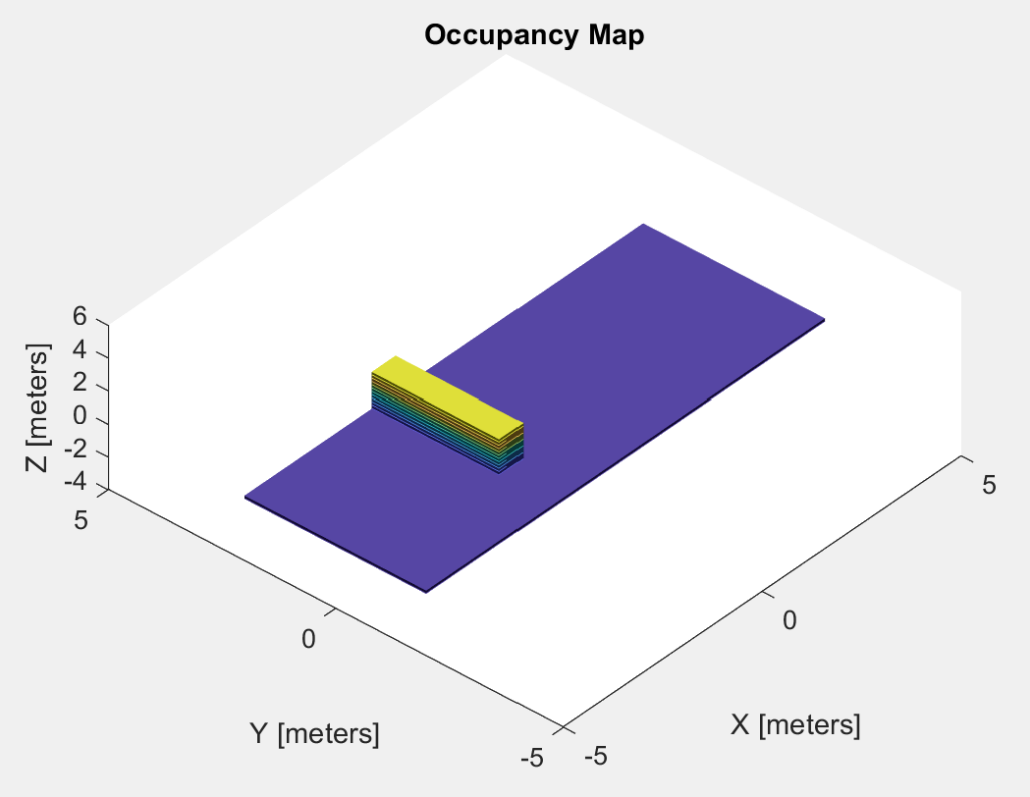


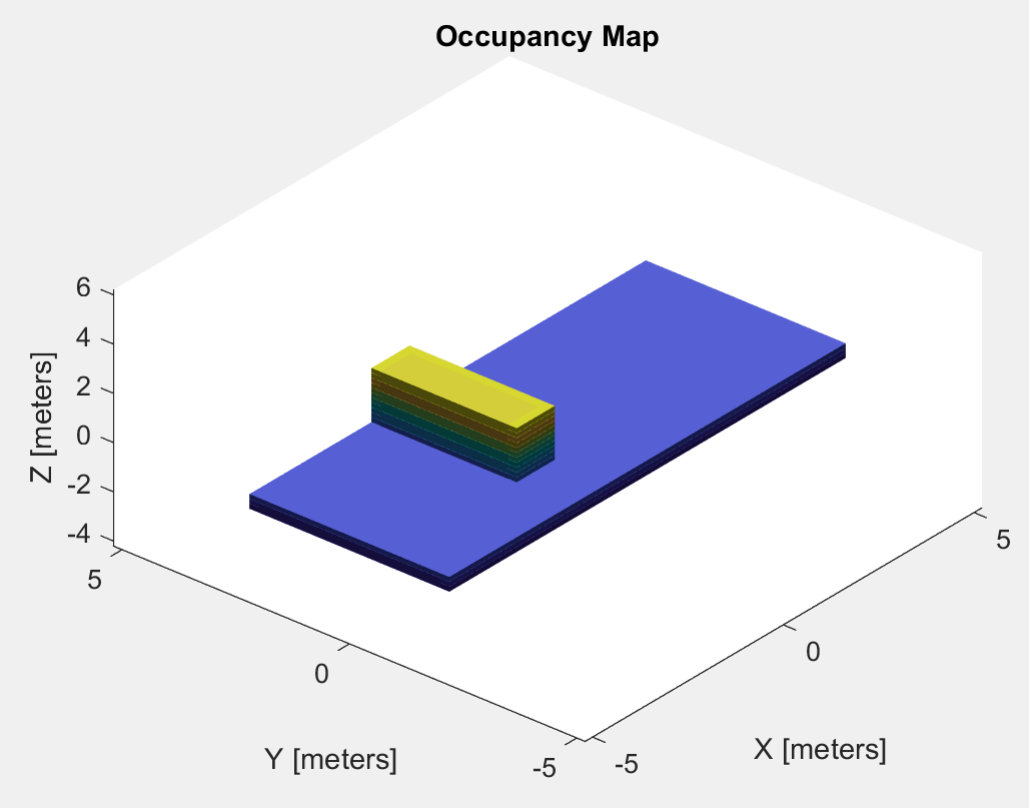
```
% Define Ground Plane
for x = -length_ground/2+res/2:res:length_ground/2-res/2
        for y = -width_ground/2+res/2:res:width_ground/2-res/2
                for z = -height_ground
                        xyz = [x y z];
                        setOccupancy(map3D, xyz, 1)
                end
        end
end

show(map3D)

%% Insert boxes
length_box = 0.4; % use multiple of the cell dimensions
width_box = 2.4;
height_box = 2;


% Box 1
pos_x1 = -1.6; % use multiple of the cell dimensions
pos_y1 = 0.4;
pos_z1 = 1;


create_box(res, pos_x1, pos_y1, pos_z1, length_box, width_box, height_box, map3D)


show(map3D)
```



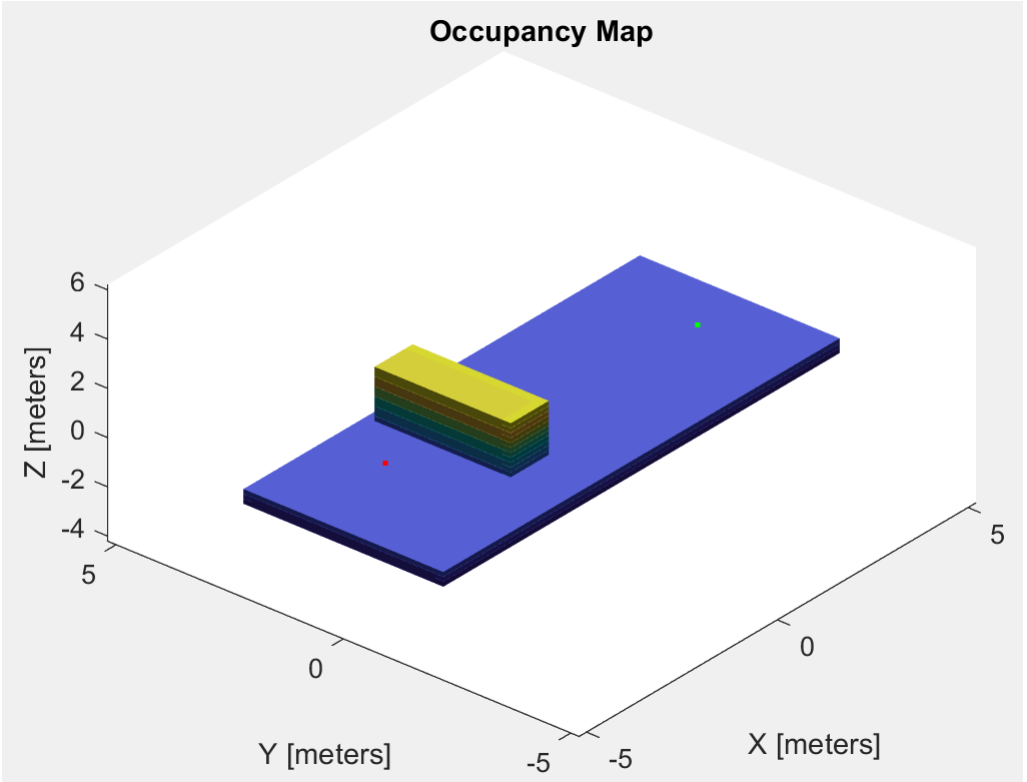
%% inflate the map with the drone radius

inflate(map3D,r)
show(map3D)





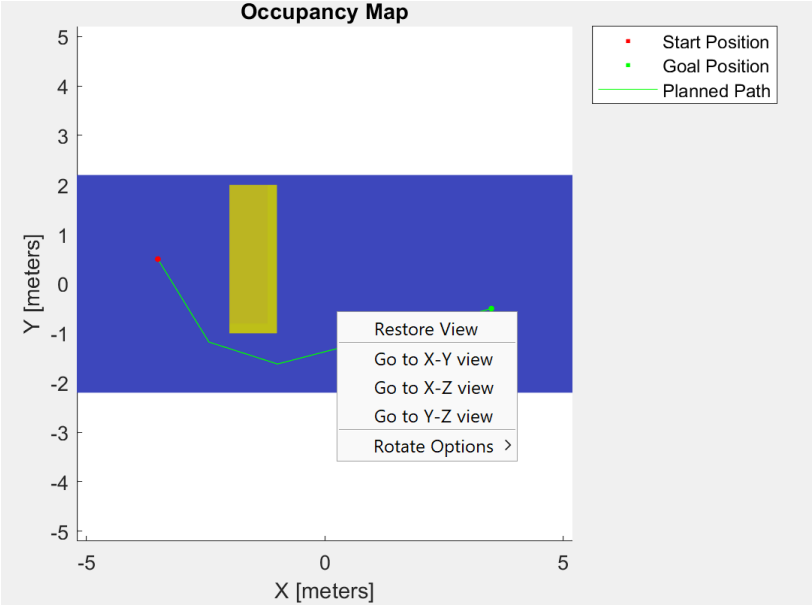
```
%% Starting position
startState = [-3.5, 0.5, 1];
% Target position
goalState = [3.5, -0.5, 1];


% Plot map, start pose, and goal pose
show(map3D)
hold on
scatter3(startState(1),startState(2),startState(3),30,".r")
scatter3(goalState(1),goalState(2),goalState(3),30,".g")
```

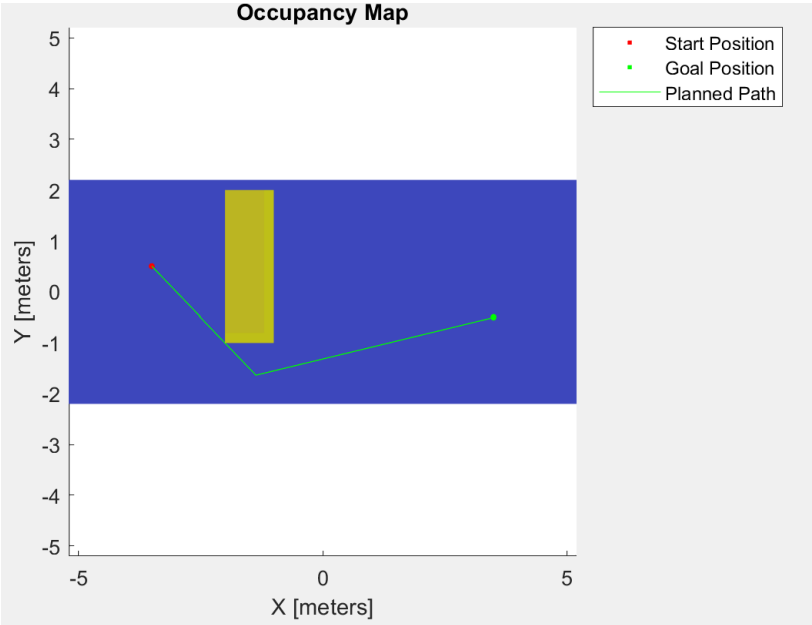


```
%% Define state space

xmax = mapsize.x/2;
ymax = mapsize.y/2;
zmax = 2;
ss = E3space([-xmax xmax;-ymax ymax;0 zmax]);

% Define validator occupancy object

sv = validatorOccupancyMap3D(ss);
sv.ValidationDistance = res;
sv.Map = map3D;
```

```
%% Plan the path

planner = plannerRRTStar(ss,sv);

%tunable parameters
planner.ContinueAfterGoalReached = true;
planner.MaxConnectionDistance = TO BE SET;
planner.GoalBias = TO BE SET; %between 0 and 1
planner.MaxIterations = TO BE SET;
planner.MaxNumTreeNodes = TO BE SET;
%planner.BallRadiusConstant = 100 ; % (default) read the documentation

[pthObj,solnInfo] = plan(planner, startState, goalState);

if (~solnInfo.IsPathFound)
disp("No Path Found by the RRT, terminating example")
return
end
```

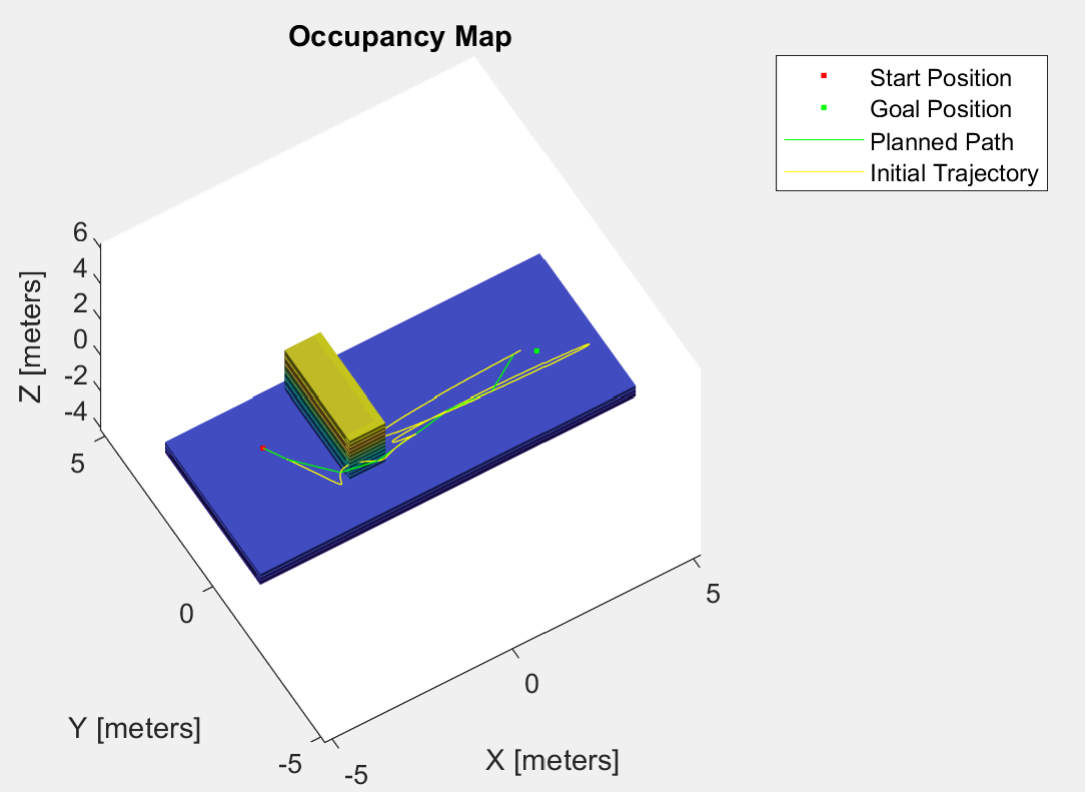
```
% Plot the waypoints
plot3(pthObj.States(:,1),pthObj.States(:,2),pthObj.States(:,3),"-g")
view([-31 63])
legend("","Start Position","Goal Position","Planned Path")
```



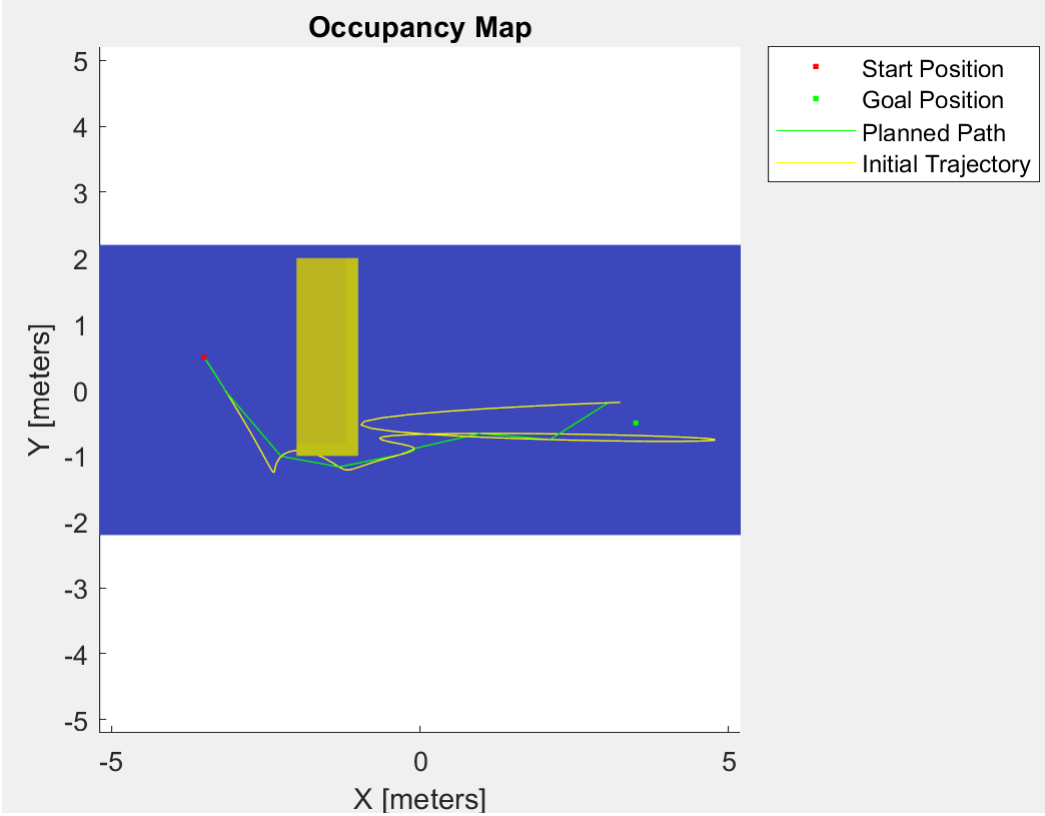


%% Generate trajectory

```
waypoints = pthObj.States(:,1:3);
nWayPoints = pthObj.NumStates;

% Calculate the distance between waypoints

distance = zeros(1,nWayPoints);
for i = 2:nWayPoints
distance(i) = norm(waypoints(i,1:3) - waypoints(i-1,1:3));
end

% Assume a UAV speed of 1 m/s and calculate time taken to reach each waypoint

UAVspeed = 1;
timepoints = cumsum(distance/UAVspeed);
nSamples = 100;

% Compute states along the trajectory (minsnappolytraj )
[y,yd,ydd,yddd,ydddd,pp,tPoints,tSamples] = minsnappolytraj(waypoints',timepoints,nSamples);

initialStates = y';

% Plot the minimum snap trajectory
plot3(initialStates(:,1),initialStates(:,2),initialStates(:,3),"-y")
legend("","Start Position","Goal Position","Planned Path","Initial Trajectory")
```



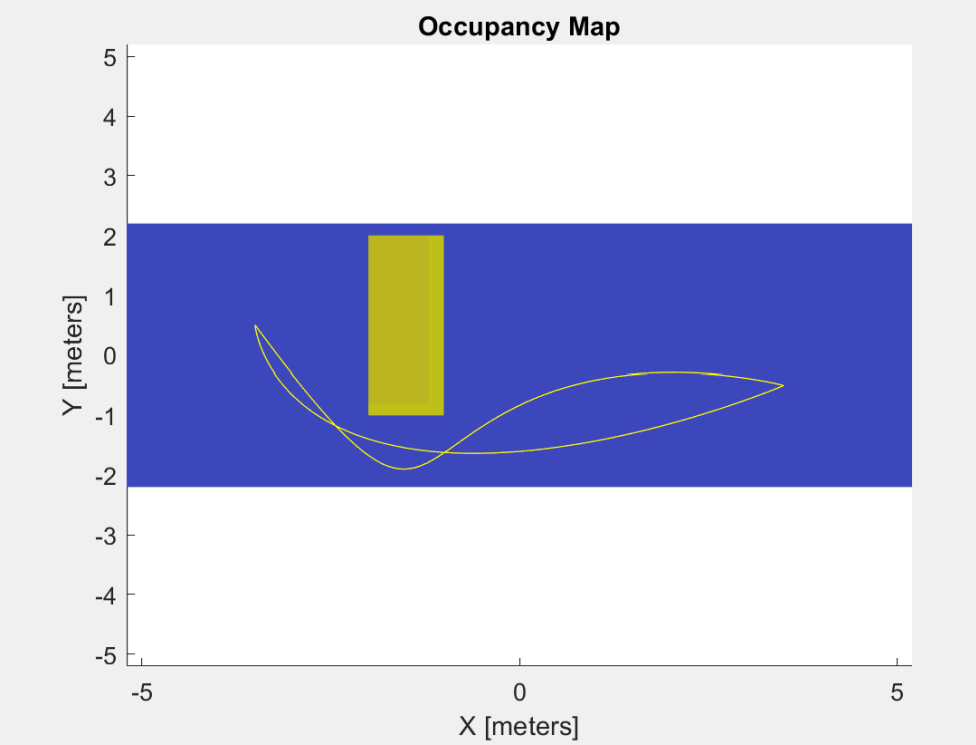
```
% check that the trajectory does not hit the obstacles

states = y';
valid = all(isStateValid(sv,states));

if (~valid)
disp("Trajectory not feasible, adding intermediate waypoints")
end
```



%% Generate trajectory: Alternative solution using time weight

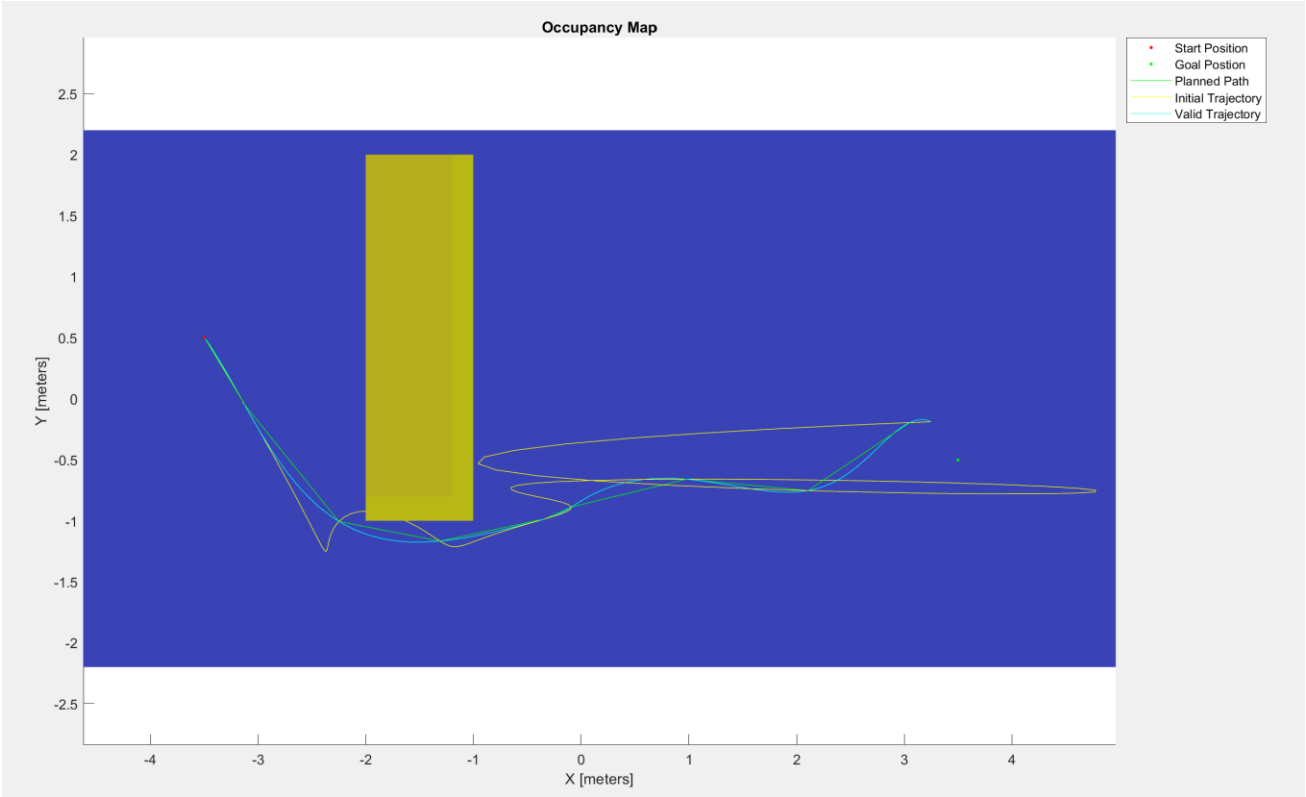
% Compute states along the trajectory (minsnappolytraj with time weight)

```
[y,yd,ydd,yddd,ydddd,pp,tPoints,tSamples] =
minsnappolytraj(waypoints',timepoints,nSamples,MinSegmentTime=0.1,MaxSegmentTime=20,TimeAllocation=true,TimeWeight=1);
```



%% Generate feasible minimum snap trajectory by adding intermediate waypoints

```
while(~valid)
% Check the validity of the states
validity = isStateValid(sv,states);

% Map the states to the corresponding waypoint segments
segmentIndices = MapStatesToPathSegments(waypoints,states);

% Get the segments for the invalid states
% Use unique, because multiple states in the same segment might be invalid
invalidSegments = unique(segmentIndices(~validity));

% Add intermediate waypoints on the invalid segments
for i = 1:numel(invalidSegments)
segment = invalidSegments(i);
% Take the midpoint of the position to get the intermediate position
midpoint = (waypoints(segment,:) + waypoints(segment+1,:))/2;
end

nWayPoints = size(waypoints,1);
distance = zeros(1,nWayPoints);
for i = 2:nWayPoints
distance(i) = norm(waypoints(i,1:3) - waypoints(i-1,1:3));
end

% Calculate the time taken to reach each waypoint
timepoints = cumsum(distance/UAVspeed);
nSamples = 100;
[y,yd,ydd,yddd,ydddd,pp,tPoints,tSamples] = minsnappolytraj(waypoints',timepoints,nSamples,MinSegmentTime=0.1,MaxSegmentTime=20,TimeAllocation=true,TimeWeight=1);
states = y';
% Check if the new trajectory is valid
valid = all(isStateValid(sv,states));

end
```



```
save('poly_coeff','pp')


% Plot the final valid trajectory

plot3(states(:,1),states(:,2),states(:,3),"-c")
view([-31 63])
legend("","Start Position","Goal Postion","Planned Path","Initial Trajectory","Valid Trajectory")
hold off

%% plot velocity and acceleration to check dynamic feasibility

figure
plot(tSamples,yd)
grid on
xlabel('Time [s]')
ylabel("Velocity [m/s]")

figure
plot(tSamples,ydd)
grid on
xlabel('Time [s]')
ylabel("Acceleration [m/s2]")
```