Proceedings of CONF-MLA 2025 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2025.LD28901

Unknown Maze Map with Unknown Coordinates Exploration Through HPHS and CvaR Framework

Jianbo Kong

Department of University of Victoria, Victoria, Canada
Jianbokong@uvic.ca

Abstract. This research focuses on the problem of robots searching for exits with unknown coordinates in a completely unknown environment. An autonomous exploration framework integrating hierarchical division and risk assessment is proposed to deal with the problem. Firstly, the system utilises depth-first search (DFS) in combination with a Visited Map to construct an initial feasible one path. Subsequently, environmental perception and map updating are realised through Occupancy Map - SLAM. At the high-level stage, the HPHS framework is introduced to partition the global environment into coarse-grained regions. Potential exploration targets are screened according to regional accessibility and frontier density. At the local level, frontier clustering is employed to generate candidate points. Simultaneously, the Conditional Value at Risk (CVaR) model is adopted for risk-sensitive selection. This is intended to enhance the robustness and efficiency of exploration. The experimental results demonstrate that this method can effectively avoid local dilemmas in dense unknown mazes while maintaining exploration coherence. Additionally, it surpasses traditional strategies based on A* or simple frontier search in aspects such as coverage rate, path rationality, and convergence speed. This proposed scheme offers a novel solution idea for autonomous exploration in complex, unknown environments.

Keywords: Unknown environment, Unknown coordinates, Occupancy Map – SLAM, Autonomous exploration

1. Introduction

With the development of society, the demand for various underground exploration operations is gradually increasing. In narrow areas such as mines and tunnels, there may be toxic gases or other dangerous factors. These hazardous factors may pose safety risks for human entry. Therefore, whether it is post-disaster search and rescue or underground mineral exploration, robots are needed [1]. However, the environment in underground and post-disaster areas is different from that in indoor environments. Collapsed mines or tunnels pose significant challenges for robot exploration. Underground and post-disaster environments are typically characterised by darkness, narrow passages, obstructed vision, and complex structures, including blocked paths or repetitive tunnels [2]. To address the potential problems mentioned above, robots should possess corresponding key capabilities. These capabilities include map generation, precise positioning, path planning, and safe navigation. Based on these capabilities, robot exploration will involve technologies such as SLAM,

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2025.LD28901

Frontier-bias, and Information gain [1]. This project aims to optimise specific capabilities and simulate robot exploration in safe exit locations within unknown maps and coordinates, to the greatest extent possible. Since the project will mainly focus on the robot's ability to precisely find exits in narrow and unknown maps and the unknown position of the target, the overall discussion will cover four aspects, including the robustness of path planning, risk control capabilities, path memory, and dead-end avoidance, as well as structural exploration capabilities. The core of the project is to build maps and explore unknown areas in the environment autonomously. Existing methods (such as traditional methods based on frontier-based or information gain) may encounter problems such as inefficient path planning or incomplete map coverage when dealing with large or complex environments. To solve these problems, HPHS combines a two-layer planning system with real-time laser radar data to efficiently sample the leading area and form a hierarchical path planning that combines global and local elements. Through these methods, the map coverage has increased by 7% - 20%, and the average path time and total travel time of the robot have also been significantly reduced [3]. During the robot's exploration process, the main goal is to maximise the coverage area and minimise the exploration time and energy consumption. Traditional methods (based on leading) may be limited by the "greedy coverage" strategy, falling into local optima. However, CvaR can optimise tail risk and actively avoid local induction. Therefore, by combining CVaR as the optimal path planning strategy module with the robot's movement to weight candidate paths, the CVaR strategy is more likely to converge to the global safe path [4]. Through combining HPHS and CVaR, this approach not only reduces a large amount of redundant computations but also provides quantitative support for risk assessment in complex terrains. Creating the robot enables exploration of random and unknown maps.

2. Method

This project aims to study how robots explore an unknown maze and reach a given-sized exit. The experiment uses the simulation environment built with MATLAB R2022a to verify the effectiveness of the method. Since there are no known target coordinates in the initial stage, the complete graph search form of the global A* algorithm cannot be directly applied. Therefore, this study adopts a general exploration framework based on HPHS and combines a series of modules to implement the complete system. First, the perception module explores the front position. After obtaining the basic information of the map, it passes the result to the HPHS and CVaR modules for path evaluation, thereby obtaining a feasible path to the target. The perception module used in this project is Occupancy-SLAM, the decision module is HPHS + CVaR, and the backtracking module is Visited Map + decision stack [5]. HPHS first divides the map into several coarse-grained regions (for example, using a 5×5 grid as a block), thereby forming a structured high-level grid. This can reduce the robot's ineffective wandering over a large area and schedule tasks based on "local priority". In each iteration, the system evaluates the connectivity of the current position on the high-level grid. Through the evaluation, it determines which areas are reachable. Then, in the reachable areas, the frontier density ("unknown area and adjacent to at least one free cell") is accumulated as an information gain indicator [6]. To ensure the consistency of perception and planning, the system uses Occupancy Grid SLAM to write real-time local observations into a probability occupancy graph and discretise it for planning while maintaining a multi-resolution representation. HPHS conducts strategic reasoning in the high-level regions and performs frontier detection on the fine-grained grids. Thus, it maintains map consistency and navigability in the presence of incomplete observations and sensor uncertainties. HPHS conducts a joint score of "information gain - safety" for the regions (for example, using the number of frontiers/unknown cells to measure the benefit,

DOI: 10.54254/2755-2721/2025.LD28901

using the distance to obstacles to calculate the risk, and weighting and integrating after normalisation). Based on the evaluation results, the system selects the most promising region as the next target. After entering the target area, the system clusters the frontiers on the fine-grained grids and projects the geometric centroids of each connected domain to the nearest reachable position to form a candidate target point set; since the end point is unknown, the candidate points need to be sorted in a risk-sensitive manner. The sorting method uses CVaR as the tail risk measure, prioritising minimising the worst-case cost. Then, the system performs local A* search for the selected target and progresses step by step; if a collision occurs, the newly discovered obstacle will be immediately written into the occupancy graph to correct the environment [7]. When a particular region no longer generates available candidate points or has been fully explored, HPHS will reschedule the next region. In this way, the system achieves structured advancement in large-scale environments and improves exploration efficiency. Finally, to avoid unnecessary repetition, the system introduces a "Visited + stack-based backtracking" depth-first mechanism. The visited cells will be marked in the graph, and the path will be recorded. When the current path cannot continue, the system will backtrack along the recorded trajectory to the nearest unvisited fork point. At the same time, the remaining frontiers will be selected to continue exploration, thus maintaining the continuity and reversibility of exploration without relying on global target coordinates [8-10].

3. Experiment

3.1. Map generation (DFS + connectivity constraints)

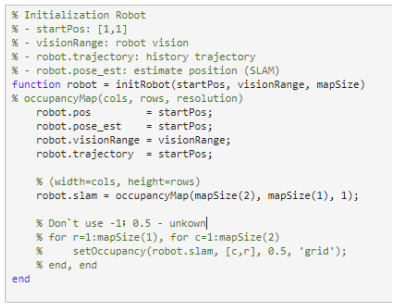
```
function [map, path, visited_dfs] = generateMapWithPath(mapSize, density, goal_size)
% Generate the obstacle map with a path from (1,1) to goal_size*goal_size
rows = mapSize(1);
cols = mapSize(2);

% 1) Generate a Random path
path = generateGuaranteedPath(mapSize);

% 2) Define the Exit area
goal_region = false(rows, cols);
goal_region(end-goal_size+1:end, end-goal_size+1:end) = true;

dirs = [0 1;1 0;0 -1;-1 0];

while true
    % Random obstacle
    map = double(rand(rows, cols) < density);  % 1-obs,0-free
    % Delete the obstacle which around path, StartPos and Exit
    for k = 1:size(path,1)
        map(path(k,1), path(k,2)) = 0;
    end
    map(1,1) = 0;
    map(goal_region) = 0;

    % BFS Checking
    visited_bfs = false(rows, cols);
    queue = [1,1];
    visited_bfs(1,1) = true;
    reachable = false;
    while ~isempty(queue)
        cur = queue(1,:); queue(1,:) = [];
        if goal_region(cur(1),cur(2))
            reachable = true; break;
        end
        for d = 1:4
            nr = cur(1)+dirs(d,1);
            nc = cur(2)+dirs(d,2);
            if nr>=1 && nr<=rows && nc>=1 && nc<=cols && ...
               ~visited_bfs(nr,nc) && map(nr,nc)==0
                visited_bfs(nr,nc) = true;
                queue(end+1,:) = [nr,nc];
            end
        end
    end
    if reachable
        break;
    end
end

% 4) Compete DFS label (visualization)
visited_dfs = false(rows, cols);
stack = [1,1];
while ~isempty(stack)
    p = stack(end,:); stack(end,:) = [];
    if visited_dfs(p(1),p(2)) || map(p(1),p(2))==1
        continue;
    end
    visited_dfs(p(1),p(2)) = true;
    for d = 1:4
        nr = p(1)+dirs(d,1);
        nc = p(2)+dirs(d,2);
        if nr>=1 && nr<=rows && nc>=1 && nc<=cols
            stack(end+1,:) = [nr,nc];
        end
    end
end
end
```

Figure 1. Map generation

Using the DFS algorithm, a maze map with controllable density is constructed. When the map is initialised, it is in a state of complete obstacles, and the DFS search process starts from the origin (1,1). The passage paths are explored through a stack structure, and the initial path is appropriately expanded to prevent getting stuck in dead corners. The code for this part will be shown in Figure 1.

DOI: 10.54254/2755-2721/2025.LD28901

3.2. Robot initialisation (position + occupy grid)

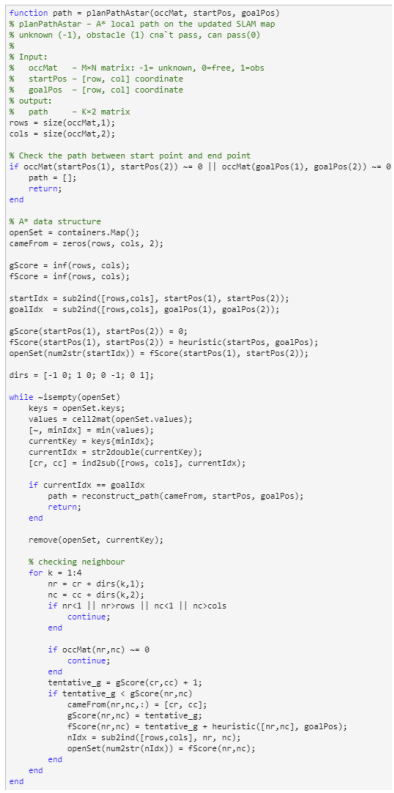


```
% Initialization Robot
% - startPos: [1,1]
% - visionRange: robot vision
% - robot.trajectory: history trajectory
% - robot.pose_est: estimate position (SLAM)
function robot = initRobot(startPos, visionRange, mapSize)
% occupancyMap(cols, rows, resolution)
    robot.pos         = startPos;
    robot.pose_est    = startPos;
    robot.visionRange = visionRange;
    robot.trajectory  = startPos;

    % (width=cols, height=rows)
    robot.slam = occupancyMap(mapSize(2), mapSize(1), 1);

    % Don`t use -1: 0.5 - unkown
    % for r=1:mapSize(1), for c=1:mapSize(2)
    %     setOccupancy(robot.slam, [c,r], 0.5, 'grid');
    % end, end
end
```

Figure 2. Robot initialisation

The robot is initialised at the coordinate position (1,1). Additionally, an access record is created and marked. If necessary, a backtracking stack is prepared for the backtracking strategy. The code will be shown in Figure 2.

3.3. Planning - A*



```
function path = planPathAstar(occMat, startPos, goalPos)
% planPathAstar – A* local path on the updated SLAM map
% unknown (-1), obstacle (1) cna`t pass, can pass(0)
%
% Input:
%   occMat   – M×N matrix: -1= unknown, 0=free, 1=obs
%   startPos – [row, col] coordinate
%   goalPos  – [row, col] coordinate
% output:
%   path     – K×2 matrix
rows = size(occMat,1);
cols = size(occMat,2);

% Check the path between start point and end point
if occMat(startPos(1), startPos(2)) ~= 0 || occMat(goalPos(1), goalPos(2)) ~= 0
    path = [];
    return;
end

% A* data structure
openSet = containers.Map();
cameFrom = zeros(rows, cols, 2);

gScore = inf(rows, cols);
fScore = inf(rows, cols);

startIdx = sub2ind([rows,cols], startPos(1), startPos(2));
goalIdx  = sub2ind([rows,cols], goalPos(1), goalPos(2));

gScore(startPos(1), startPos(2)) = 0;
fScore(startPos(1), startPos(2)) = heuristic(startPos, goalPos);
openSet(num2str(startIdx)) = fScore(startPos(1), startPos(2));

dirs = [-1 0; 1 0; 0 -1; 0 1];

while ~isempty(openSet)
    keys = openSet.keys;
    values = cell2mat(openSet.values);
    [~, minIdx] = min(values);
    currentKey = keys{minIdx};
    currentIdx = str2double(currentKey);
    [cr, cc] = ind2sub([rows, cols], currentIdx);

    if currentIdx == goalIdx
        path = reconstruct_path(cameFrom, startPos, goalPos);
        return;
    end

    remove(openSet, currentKey);

    % checking neighbour
    for k = 1:4
        nr = cr + dirs(k,1);
        nc = cc + dirs(k,2);
        if nr<1 || nr>rows || nc<1 || nc>cols
            continue;
        end

        if occMat(nr,nc) ~= 0
            continue;
        end
        tentative_g = gScore(cr,cc) + 1;
        if tentative_g < gScore(nr,nc)
            cameFrom(nr,nc,:) = [cr, cc];
            gScore(nr,nc) = tentative_g;
            fScore(nr,nc) = tentative_g + heuristic([nr,nc], goalPos);
            nIdx = sub2ind([rows,cols], nr, nc);
            openSet(num2str(nIdx)) = fScore(nr,nc);
        end
    end
end
```

Figure 3. A*

The occPlanBin obtained by discretising the SLAM probability graph is used as the planning input. Then, the current robot position is taken as the starting point. Meanwhile, the candidate targets generated by HPHS/CVaR are used as the ending point. The algorithm searches on the 4-adjacent (up/right/down/left) grid. This part serves as a local exploration algorithm for the unknown map,

DOI: 10.54254/2755-2721/2025.LD28901

which has been detected through Occupancy Map SLAM. The exploration algorithm is shown in Figure 3.

```
function h = heuristic(a, b)
    h = abs(a(1)-b(1)) + abs(a(2)-b(2));
end

function path = reconstruct_path(cameFrom, startPos, goalPos)
    % From goal to start
    path = goalPos;
    current = goalPos;
    while ~isequal(current, startPos)
        p = squeeze(cameFrom(current(1), current(2), :))';
        if all(p==0)
            path = []; |
            return;
        end
        path = [p; path];
        current = p;
    end
end
```

Figure 4. Manhattan distance

The function of Manhattan distance is to serve as the basis for heuristic estimation and path cost calculation. This code, as shown in Figure 4, enables the robot to quickly select the target, optimise the search direction, and avoid unnecessary exploration in a grid environment.

3.4. Frontier detection

```
function frontiers = detectFrontiers(occMat)
% detectFrontiers - Find frontier
% Input: occMat (-1=unknown, 0=free, 1=obs)
% Output: K×2 [row,col]

[rows, cols] = size(occMat);
dirs = [-1 0; 1 0; 0 -1; 0 1];
list = [];

for r = 1:rows
    for c = 1:cols
        if occMat(r,c) == 0
            for k = 1:4
                nr = r + dirs(k,1);
                nc = c + dirs(k,2);
                if nr>=1 && nr<=rows && nc>=1 && nc<=cols && occMat(nr,nc)==-1
                    list(end+1,:) = [nr,nc];
                end
            end
        end
    end
end

if isempty(list)
    frontiers = zeros(0,2);
else
    frontiers = unique(list,'rows');
end
```

Figure 5. Frontier detection

This is a part of the perception module. The code shown in Figure 5 is responsible for identifying the boundary points of the unknown areas. These points are then passed on to the upper-level target selection strategy (such as A* path planning + HPHS + CVaR) to determine where the robot should move.

3.5. HPHS

3.5.1. Divide the map

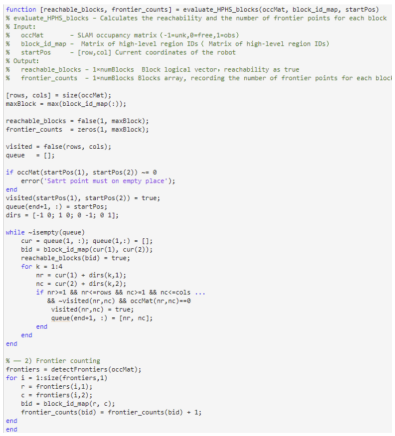
```
function block_id_map = divideRegions(mapSize, block_size)
% divideRegions -  Divide the map into coarse-grained high-level blocks
% Input: mapSize=[rows,cols], block_size
% Output: block_id_map (rows×cols)

rows = mapSize(1);
cols = mapSize(2);
numBlocksX = ceil(cols / block_size);
numBlocksY = ceil(rows / block_size);
block_id_map = zeros(rows, cols);
id = 1;
for by = 1:numBlocksY
    for bx = 1:numBlocksX
        r0 = (by-1)*block_size + 1;
        c0 = (bx-1)*block_size + 1;
        rIdx = r0 : min(by*block_size, rows);
        cIdx = c0 : min(bx*block_size, cols);
        block_id_map(rIdx, cIdx) = id;
        id = id + 1;
    end
end
end
```

Figure 6. HPHS STEP – divide the map

DOI: 10.54254/2755-2721/2025.LD28901

The first step of the HPHS (Hybrid Perception Hierarchical Sampling) framework. This part of the code, as shown in Figure 6, divides the entire map into coarse-grained high-level blocks. Each block has a unique ID. This forms the basis of the HPHS framework. The subsequent exploration strategies (frontier clustering, CVaR evaluation, target selection) will all be based on this high-level division.

3.5.2. Regional accessibility and frontier density assessment



```
function [reachable_blocks, frontier_counts] = evaluate_HPHS_blocks(occMat, block_id_map, startPos)
% evaluate_HPHS_blocks - Calculates the reachability and the number of frontier points for each block
% Input:
%   occMat        - SLAM occupancy matrix (-1=unk,0=free,1=obs)
%   block_id_map  - Matrix of high-level region IDs ( Matrix of high-level region IDs)
%   startPos      - [row,col] Current coordinates of the robot
% Output:
%   reachable_blocks - 1×numBlocks  Block logical vector, reachability as true
%   frontier_counts  - 1×numBlocks Blocks array, recording the number of frontier points for each bloc

[rows, cols] = size(occMat);
maxBlock = max(block_id_map(:));

reachable_blocks = false(1, maxBlock);
frontier_counts  = zeros(1, maxBlock);

visited = false(rows, cols);
queue   = [];

if occMat(startPos(1), startPos(2)) ~= 0
    error('Satrt point must on empty place');
end
visited(startPos(1), startPos(2)) = true;
queue(end+1, :) = startPos;
dirs = [-1 0; 1 0; 0 -1; 0 1];

while ~isempty(queue)
    cur = queue(1, :); queue(1,:) = [];
    bid = block_id_map(cur(1), cur(2));
    reachable_blocks(bid) = true;
    for k = 1:4
        nr = cur(1) + dirs(k,1);
        nc = cur(2) + dirs(k,2);
        if nr>=1 && nr<=rows && nc>=1 && nc<=cols ...
           && ~visited(nr,nc) && occMat(nr,nc)==0
            visited(nr,nc) = true;
            queue(end+1, :) = [nr, nc];
        end
    end
end

% — 2) Frontier counting
frontiers = detectFrontiers(occMat);
for i = 1:size(frontiers,1)
    r = frontiers(i,1);
    c = frontiers(i,2);
    bid = block_id_map(r, c);
    frontier_counts(bid) = frontier_counts(bid) + 1;
end
end
```

Figure 7. HPHS STEP – BFS + frontier detection

The Second step of the HPHS (Hybrid Perception Hierarchical Sampling) framework. This part is responsible for combining BFS + frontier detection. It will calculate the reachability and frontier density of each high-level block. This can provide input for the subsequent CVaR target selection. The code for this part is shown in Figure 7.

3.5.3. Candidate target point extraction based on frontier clustering



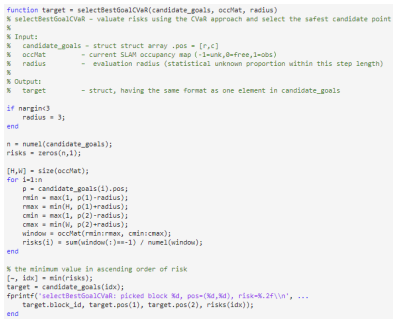
```
function candidate_goals = extract_candidate_goals_from_blocks(occMat, block_id_map, reachable_blocks)
% extract_candidate_goals_from_blocks - Cluster the frontiers and output the candidate points
% Output struct array with fields pos, block_id, cluster_id

[rows, cols] = size(occMat);
frontiers = detectFrontiers(occMat);
candidate_goals = struct('pos',{},'block_id',{},'cluster_id',{});
idx=1;

% Cluster processing for each reachable block
for bid = find(reachable_blocks)
    pts = frontiers(block_id_map(sub2ind([rows,cols], frontiers(:,1), frontiers(:,2)))==bid, :);
    if isempty(pts), continue; end
    mask = false(rows, cols);
    linIdx = sub2ind([rows,cols], pts(:,1), pts(:,2));
    mask(linIdx) = true;
    CC = bwconncomp(mask, 8);
    for c = 1:CC.NumObjects
        pix = CC.PixelIdxList{c};
        [rlist, clist] = ind2sub([rows,cols], pix);
        centroid = [mean(rlist), mean(clist)];
        candidate_goals(idx).pos = round(centroid);
        candidate_goals(idx).block_id = bid;
        candidate_goals(idx).cluster_id = c;
        idx = idx + 1;
    end
end
end
```

Figure 8. HPHS STEP – target point extraction

This part will cluster the frontier points in the accessible high-level areas. Then, the cluster centres will be extracted as candidate target points. These results will serve as input for the subsequent risk-sensitive target selection (CVaR). The code for this part is shown in Figure 8.

DOI: 10.54254/2755-2721/2025.LD28901

3.5.4. CVaR



```
function target = selectBestGoalCVaR(candidate_goals, occMat, radius)
% selectBestGoalCVaR - valuate risks using the CVaR approach and select the safest candidate point
%
% Input:
%   candidate_goals - struct struct array .pos = [r,c]
%   occMat          - current SLAM occupancy map (-1=unk,0=free,1=obs)
%   radius          -  evaluation radius (statistical unknown proportion within this step length)
%
% Output:
%   target          - struct, having the same format as one element in candidate_goals

if nargin<3
    radius = 3;
end

n = numel(candidate_goals);
risks = zeros(n,1);

[H,W] = size(occMat);
for i=1:n
    p = candidate_goals(i).pos;
    rmin = max(1, p(1)-radius);
    rmax = min(H, p(1)+radius);
    cmin = max(1, p(2)-radius);
    cmax = min(W, p(2)+radius);
    window = occMat(rmin:rmax, cmin:cmax);
    risks(i) = sum(window(:)==-1) / numel(window);
end

% the minimum value in ascending order of risk
[~, idx] = min(risks);
target = candidate_goals(idx);
fprintf('selectBestGoalCVaR: picked block %d, pos=(%d,%d), risk=%.2f\n', ...
    target.block_id, target.pos(1), target.pos(2), risks(idx));
end
```

Figure 9. HPHS STEP - CvaR

As the final step of the HPHS framework, this step conducts a risk assessment (CVaR) for all candidate target points. Among all the results, the target point with the lowest risk is selected as the exploration target. In this way, the robot can conduct more stable and safer exploration. The code for this part is shown in Figure 9.

3.6. Nearest frontier

```
function p = pickNearest(frontiers, pos)
    d = abs(frontiers(:,1)-pos(1)) + abs(frontiers(:,2)-pos(2));
    [~,i] = min(d);
    p = frontiers(i,:);
end
```

Figure 10. Finding the nearest frontier point

This part serves as a fallback mechanism for the entire project. When the HPHS + CVaR target selection fails, the Manhattan distance will be used to find the nearest frontier point. This ensures that the robot always has the next exploration target. The code for this part is shown in Figure 10.

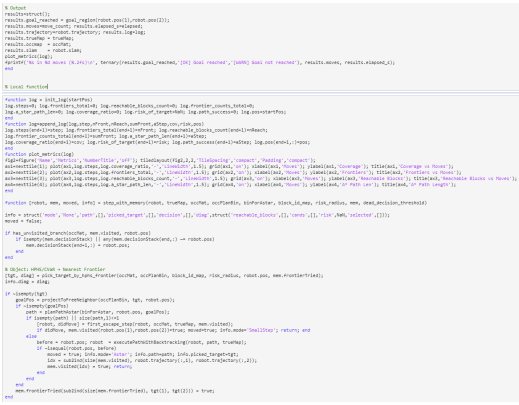
3.7. Scheduling



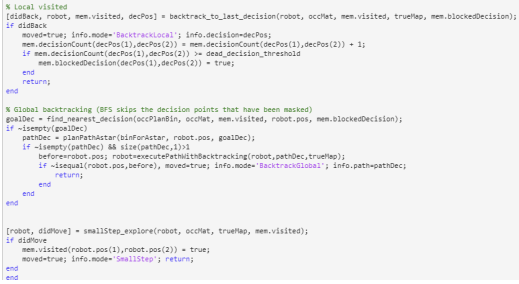
DOI: 10.54254/2755-2721/2025.LD28901



```
%%SLAM
robot.slam = updateOccupancyMap(robot, trueMap, robot.slam);
occProb = occupancyMatrix(robot.slam);
occMat  = discretizeOccProb(occProb);
occMat(robot.pos(1),robot.pos(2)) = 0;
set(hProb,'CData',occProb); set(hOcc,'CData',occMat); set(hOcc4,'CData',occMat); drawnow;

%% Main loop
log = init_log(startPos);
move_count=0; tic;
while ~goal_reglon(robot.pos(1),robot.pos(2)) && move_count<max_moves

    robot.slam = updateOccupancyMap(robot,trueMap,robot.slam);
    occProb    = occupancyMatrix(robot.slam);
    occMat     = discretizeOccProb(occProb);
    occMat(robot.pos(1),robot.pos(2)) = 0;
    occPlanBin = occMat;  occPlanBin(occPlanBin==-1)=0;
    binForAstar = occPlanBin; binForAstar(binForAstar==0)=1;

    % === Select MPHS/CVaR decision + diagnosis (minimum modification)  ===
    [robot, mem, moved, info] = step_with_memory(robot,trueMap,occMat,occPlanBin,binForAstar, ...
        block_id_map,risk_radius,mem,dead_decision_threshold);

    reachCount = 0;
    riskVal    = NaN;
    if isfield(info, 'diag') && ~isempty(info.diag)
        if isfield(info.diag, 'reachable_blocks') && ~isempty(info.diag.reachable_blocks)
            reachCount = sum(info.diag.reachable_blocks(:));
        end
        if isfield(info.diag, 'risk') && ~isempty(info.diag.risk)
            riskVal = info.diag.risk;
        end
    end

    if moved, move_count = move_count + 1; end

    fronts = detectFrontiers(occMat);
    if ~isempty(fronts), set(hFront,'XData',fronts(:,2),'YData',fronts(:,1)); else, set(hFront,'XData',NaN,'YData',NaN); end

    set(hProb,'CData',occProb); set(hOcc,'CData',occMat); set(hOcc4,'CData',occMat);
    tr=robot.trajectory;
    set(hTr1,'XData',tr(:,2),'YData',tr(:,1)); set(hPos1,'XData',robot.pos(2),'YData',robot.pos(1));
    set(hTr3,'XData',tr(:,2),'YData',tr(:,1)); set(hPos3,'XData',robot.pos(2),'YData',robot.pos(1));
    set(hTr4,'XData',tr(:,2),'YData',tr(:,1)); set(hPos4,'XData',robot.pos(2),'YData',robot.pos(1));
    if ~isempty(info.path)
        set(hPath3,'XData',info.path(:,2),'YData',info.path(:,1));
        set(hPath4,'XData',info.path(:,2),'YData',info.path(:,1));
    else
        set(hPath3,'XData',NaN,'YData',NaN); set(hPath4,'XData',NaN,'YData',NaN);
    end

    coverage = nnz(occMat==-1)/numel(occMat);
    set(hTxt,'String',sprintf('moves=%d | fronts=%d | reach=%d | mode=%s | cov=%.1f%%', move_count, size(fronts,1), reachCount, info.mode, 100*coverage));
    drawnow;

    if showPHS_CVaR && ~isempty(diagFig) && isvalid(diagFig)
        draw_mphs_cvar(diagFig, block_id_map, occMat, info.diag, robot.pos, move_count);
    end

    log = append_log(log, move_count, size(fronts,1), reachCount, 0, max(0,size(info.path,1)-1), coverage, riskVal, robot.pos);
end
elapsed=toc;
```



```
% Output
results=struct();
results.goal_reached = goal_region(robot.pos(1),robot.pos(2));
results.moves=move_count; results.elapsed_s=elapsed;
results.trajectory=robot.trajectory; results.log=log;
results.trueMap = trueMap;
results.occmap  = occMat;
results.slam    = robot.slam;
plot_metrics(log);
fprintf('%s in %d moves (%.2fs)\n', ternary(results.goal_reached,'[OK] Goal reached','[WARN] Goal not reached'), results.moves, results.elapsed_s);
end

% Local function

function log = init_log(startPos)
log.steps=0; log.frontiers_total=0; log.reachable_blocks_count=0; log.frontier_counts_total=0;
log.a_star_path_len=0; log.coverage_ratio=0; log.risk_of_target=NaN; log.path_success=0; log.pos=startPos;
end
function log=append_log(log,step,nFront,nReach,sumFront,aStep,cov,risk,pos)
log.steps(end+1)=step; log.frontiers_total(end+1)=nFront; log.reachable_blocks_count(end+1)=nReach;
log.frontier_counts_total(end+1)=sumFront; log.a_star_path_len(end+1)=aStep;
log.coverage_ratio(end+1)=cov; log.risk_of_target(end+1)=risk; log.path_success(end+1)=aStep; log.pos(end+1,:)=pos;
end
function plot_metrics(log)
figure('Name','Metrics','NumberTitle','off'); tiledlayout(4,2,'TileSpacing','compact','Padding','compact');
ax1=nexttile(1); plot(ax1,log.steps,log.coverage_ratio,'.-','LineWidth',1.5); grid(ax1,'on'); xlabel(ax1,'Moves'); ylabel(ax1,'Coverage'); title(ax1,'Coverage vs Moves');
ax2=nexttile(2); plot(ax2,log.steps,log.frontiers_total,'.-','LineWidth',1.5); grid(ax2,'on'); xlabel(ax2,'Moves'); ylabel(ax2,'Frontiers'); title(ax2,'Frontiers vs Moves');
ax3=nexttile(3); plot(ax3,log.steps,log.reachable_blocks_count,'.-','LineWidth',1.5); grid(ax3,'on'); xlabel(ax3,'Moves'); ylabel(ax3,'Reachable Blocks'); title(ax3,'Reachable Blocks vs Moves');
ax4=nexttile(4); plot(ax4,log.steps,log.a_star_path_len,'.-','LineWidth',1.5); grid(ax4,'on'); xlabel(ax4,'Moves'); ylabel(ax4,'A* Path Len'); title(ax4,'A* Path Length');
end

function [robot, mem, moved, info] = step_with_memory(robot, trueMap, occMat, occPlanBin, binForAstar, block_id_map, risk_radius, mem, dead_decision_threshold)

info = struct('mode','None','path',[],'picked_target',[],'decision',[],'diag',struct('reachable_blocks',[],'cands',[],'risk',NaN,'selected',[]));
moved = false;

if has_unvisited_branch(occMat, mem.visited, robot.pos)
    if isempty(mem.decisionStack) || any(mem.decisionStack(end,:) ~= robot.pos)
        mem.decisionStack(end+1,:) = robot.pos;
    end
end

% Object: MPHS/CVaR + Nearest Frontier
[tgt, diag] = pick_target_by_mphs_frontier(occMat, occPlanBin, block_id_map, risk_radius, robot.pos, mem.frontierTried);
info.diag = diag;

if ~isempty(tgt)
    goalPos = projectToFreeNeighbor(occPlanBin, tgt, robot.pos);
    if ~isempty(goalPos)
        path = planPathAstar(binForAstar, robot.pos, goalPos);
        if isempty(path) || size(path,1)<=1
            [robot, didMove] = first_escape_step(robot, occMat, trueMap, mem.visited);
            if didMove, mem.visited(robot.pos(1),robot.pos(2))=true; moved=true; info.mode='SmallStep'; return; end
        else
            before = robot.pos; robot = executePathWithBacktracking(robot, path, trueMap);
            if ~isequal(robot.pos, before)
                moved = true; info.mode='Astar'; info.path=path; info.picked_target=tgt;
                idx = sub2ind(size(mem.visited), robot.trajectory(:,1), robot.trajectory(:,2));
                mem.visited(idx) = true; return;
            end
        end
    end
    mem.frontierTried(sub2ind(size(mem.frontierTried), tgt(1), tgt(2))) = true;
end
```



```
% Local visited
[didBack, robot, mem.visited, decPos] = backtrack_to_last_decision(robot, occMat, mem.visited, trueMap, mem.blockedDecision);
if didBack
    moved=true; info.mode='BacktrackLocal'; info.decision=decPos;
    mem.decisionCount(decPos(1),decPos(2)) = mem.decisionCount(decPos(1),decPos(2)) + 1;
    if mem.decisionCount(decPos(1),decPos(2)) >= dead_decision_threshold
        mem.blockedDecision(decPos(1),decPos(2)) = true;
    end
    return;
end

% Global backtracking (BFS skips the decision points that have been masked)
goalDec = find_nearest_decision(occPlanBin, occMat, mem.visited, robot.pos, mem.blockedDecision);
if ~isempty(goalDec)
    pathDec = planPathAstar(binForAstar, robot.pos, goalDec);
    if ~isempty(pathDec) && size(pathDec,1)>1
        before=robot.pos; robot=executePathWithBacktracking(robot,pathDec,trueMap);
        if ~isequal(robot.pos,before), moved=true; info.mode='BacktrackGlobal'; info.path=pathDec;
            return;
        end
    end
end


[robot, didMove] = smallStep_explore(robot, occMat, trueMap, mem.visited);
if didMove
    mem.visited(robot.pos(1),robot.pos(2)) = true;
    moved=true; info.mode='SmallStep'; return;
end
end
```

DOI: 10.54254/2755-2721/2025.LD28901

```
% ======= Backtracking tool =======
function tf = has_unvisited_branch(occMat, visited, p)
tf=false; dirs=[-1 0;0 1;1 0;0 -1]; H=size(occMat,1); W=size(occMat,2);
for k=1:4
    q=p+dirs(k,:);
    if all(q>=1)&&q(1)<=H&&q(2)<=W
        if (occMat(q(1),q(2))==0 || occMat(q(1),q(2))==-1) && ~visited(q(1),q(2))
            tf=true; return;
        end
    end
end
end

function [didBack, robot, visited, decision_pos] = backtrack_to_last_decision(robot, occMat, visited, trueMap, blockedDecision)
didBack=false; decision_pos=[];
trj = robot.trajectory;
if size(trj,1) < 2, return;
end

target_idx=[];
for i=size(trj,1)-1:-1:1
    p = trj(i,:);
    if blockedDecision(p(1),p(2)), continue;
    end
    if has_unvisited_branch(occMat, visited, p), target_idx=i;
        break;
    end
end
if isempty(target_idx), return;
end
decision_pos = trj(target_idx,:);

back_path = trj(end:-1:target_idx,:);
movedSteps = 0;
for i=2:size(back_path,1)
    p=back_path(i,:);
    if trueMap(p(1),p(2))==1
        xy = grid2world(robot.slam,[p(1) p(2)]);
        setOccupancy(robot.slam, xy, 1, 'world');
        break;
    end
    if any(p~=robot.pos)
        robot.pos=p; robot.trajectory(end+1,:)=p;
        robot.slam=updateOccupancyMap(robot,trueMap,robot.slam);
        visited(p(1),p(2))=true;
        movedSteps=movedSteps+1;
    end
end
didBack = movedSteps > 0;
end

function goal = find_nearest_decision(occPlanBin, occMat, visited, startPos, blockedDecision)
H=size(occPlanBin,1); W=size(occPlanBin,2);
inb=@(r,c) r>=1 && r<=H && c>=1 && c<=W; dirs=[-1 0;0 1;1 0;0 -1];
seen=false(H,W); Q=zeros(H*W,2); head=1; tail=1;
Q(tail,:)=startPos; tail=tail+1; seen(startPos(1),startPos(2))=true; goal=[];
while head<tail
    p=Q(head,:); head=head+1;
    if ~blockedDecision(p(1),p(2)) && has_unvisited_branch(occMat, visited, p)
        goal=p;
        return;
    end
    for k=1:4
        q=p+dirs(k,:);
        if inb(q(1),q(2)) && ~seen(q(1),q(2)) && occPlanBin(q(1),q(2))==0
            seen(q(1),q(2))=true; Q(tail,:)=q; tail=tail+1;
        end
    end
end
end
```

```
function robot = executePathWithBacktracking(robot, path, trueMap)
if isempty(path) || size(path,1)<2, return;
end
H=size(trueMap,1); W=size(trueMap,2);
for i=2:size(path,1)
    nextPos=path(i,:);
    if any(nextPos<1)||nextPos(1)>H||nextPos(2)>W, return;
    end
    if trueMap(nextPos(1),nextPos(2))==1
        xy = grid2world(robot.slam,[nextPos(1) nextPos(2)]);
        setOccupancy(robot.slam, xy, 1, 'world');
        return;
    end
    if size(robot.trajectory,1)>=2
        prev=robot.trajectory(end-1,:);
        if all(nextPos==prev), return;
        end
    end
    robot.pos=nextPos; robot.trajectory(end+1,:)=nextPos;
    robot.slam=updateOccupancyMap(robot,trueMap,robot.slam);
end
end

function [robot, didMove] = first_escape_step(robot, occMat, trueMap, visited)
didMove=false; dirs=[-1 0;0 1;1 0;0 -1]; H=size(occMat,1); W=size(occMat,2);
best=[]; bestScore=-inf;
for k=1:4
    q=robot.pos+dirs(k,:);
    if any(q<1)||q(1)>H||q(2)>W, continue;
    end
    if trueMap(q(1),q(2))==1, continue;
    end
    s = (occMat(q(1),q(2))==-1) + 0.2*(~visited(q(1),q(2)));
    if s>bestScore, best=q; bestScore=s;
    end
end
if ~isempty(best)
    robot.pos=best; robot.trajectory(end+1,:)=best;
    robot.slam=updateOccupancyMap(robot,trueMap,robot.slam);
    didMove=true;
end
end
```

```
function [robot, didMove] = smallStep_explore(robot, occMat, trueMap, visited)
didMove=false; H=size(occMat,1); W=size(occMat,2); dirs=[-1 0;0 1;1 0;0 -1];
bestNext=[]; bestScore=-inf;
for k=1:4
    q=robot.pos+dirs(k,:);
    if any(q<1)||q(1)>H||q(2)>W, continue;
    end
    if trueMap(q(1),q(2))==1, continue;
    end
    gain=0;
    for kk=1:4
        t=q+dirs(kk,:);
        if all(t>=1)&&t(1)<=H&&t(2)<=W && occMat(t(1),t(2))==-1, gain=gain+1;
        end
    end
    score = gain + 0.2*(~visited(q(1),q(2))) + 0.1*(occMat(q(1),q(2))==-1);
    if score>bestScore, bestScore=score; bestNext=q;
    end
end
if ~isempty(bestNext)
    robot.pos=bestNext; robot.trajectory(end+1,:)=bestNext;
    robot.slam=updateOccupancyMap(robot,trueMap,robot.slam);
    didMove=true;
end
end
```

DOI: 10.54254/2755-2721/2025.LD28901

```
function occMat = discretizeOccProb(occProb)
freeThr=0.40; occThr=0.60;
occMat = -1*ones(size(occProb),'int8');
occMat(occProb<=freeThr)=0; occMat(occProb>=occThr)=1;
end

function goal = projectToFreeNeighbor(occPlanBin, tgt, curr)
if occPlanBin(tgt(1),tgt(2))==0, goal=tgt;
    return;
end
H=size(occPlanBin,1); W=size(occPlanBin,2);
best=[]; bestd=inf;
for r=max(1,tgt(1)-1):min(H,tgt(1)+1)
  for c=max(1,tgt(2)-1):min(W,tgt(2)+1)
    if occPlanBin(r,c)==0
        d=abs(r-curr(1))+abs(c-curr(2));
        if d<bestd, best=[r c];
            bestd=d;
        end
    end
  end
end
goal=best;
end

function s=ternary(c,a,b); if c, s=a;
else, s=b;
end; end
```

```
% ======= SLAM into coorcinate =======
function slam = updateOccupancyMap(robot, trueMap, slam)
pos = robot.pos; vr  = robot.visionRange;
[H, W] = size(trueMap);
rr = max(1, pos(1)-vr) : min(H, pos(1)+vr);
cc = max(1, pos(2)-vr) : min(W, pos(2)+vr);
[R, C] = ndgrid(rr, cc);
idx      = sub2ind([H W], R(:), C(:));
occVals  = double(trueMap(idx) == 1);
worldPts = grid2world(slam, [R(:) C(:)]);     % [row col] → [x y]
setOccupancy(slam, worldPts, occVals, 'world');
```

```
% ======= Objective selection (HPHS + CVaR, visual diagnosis) =======
function [target, diag] = pick_target_by_hphs_frontier(occMat, occPlanBin, block_id_map, risk_radius, curr, frontierTried)
target = [];
diag = struct('reachable_blocks',[],'cands',[],'selected',[],'risk',NaN);

% 1) HPHS: Determine reachable blocks and extract candidates (keep your original logic: use occPlanBin)
reachable_blocks = false(size(block_id_map));
try
    [reachable_blocks, ~] = evaluate_HPHS_blocks(occPlanBin, block_id_map, curr);
catch
    reachable_blocks(:) = false;
end
diag.reachable_blocks = reachable_blocks;

cands = [];
if any(reachable_blocks,'all')
    cands = extract_candidate_goals_from_blocks(occPlanBin, block_id_map, reachable_blocks);
end

if isstruct(cands) && isfield(cands,'pos'), C = vertcat(cands.pos);
else, C = cands;
end
diag.cands = C;

% 2) CVaR: Choose the option with the lowest risk from the candidates.
if ~isempty(C)
    s = selectBestGoalCVaR(C, occMat, risk_radius);
    if isstruct(s) && isfield(s,'pos') && ~isempty(s.pos)
        target = s.pos;
        diag.selected = s.pos;
        if isfield(s,'risk'), diag.risk = s.risk; end
        return;
    elseif ~isempty(s)
        target = s(1,:);
        diag.selected = target;
        return;
    end
end

% 3) Degradation: The latest untried frontier
fronts = detectFrontiers(occMat);
if ~isempty(fronts)
    if nargin >= 6 && ~isempty(frontierTried) && isequal(size(frontierTried), size(occMat))
        lin = sub2ind(size(occMat), fronts(:,1), fronts(:,2));
        keep = ~frontierTried(lin);
        fronts = fronts(keep,:);
    end
    if ~isempty(fronts)
        target = pickNearest(fronts, curr);
        diag.selected = target;
    end
end
end
```

DOI: 10.54254/2755-2721/2025.LD28901

```
function draw_hphs_cvar(figH, block_id_map, occMat, diag, currPos, step)
    if isempty(figH) || ~isvalid(figH), return; end
    figure(figH); clf
    tiledlayout(1,2,'TileSpacing','compact','Padding','compact');

    % Left: Block IDs
    nexttile; imagesc(block_id_map); axis xy equal tight; colorbar
    title('High-level Block IDs');

    % Right: free occupancy + Reachable block contour + candidate + selected + current position
    nexttile; imshow(occMat==0); axis xy equal tight; hold on

    % Reachable block (block level -> pixel level)
    if ~isempty(diag) && isfield(diag,'reachable_blocks') && ~isempty(diag.reachable_blocks)
        reachMaskPix = ismember(block_id_map, find(diag.reachable_blocks));
        try
            bw = bwperim(reachMaskPix,4); [yr,xr] = find(bw);
            if ~isempty(yr), plot(xr,yr,'.','MarkerSize',6); end
        catch
            [yr,xr] = find(reachMaskPix); if ~isempty(yr), plot(xr,yr,'.','MarkerSize',3); end
        end
    end

    if ~isempty(diag) && isfield(diag,'cands') && ~isempty(diag.cands)
        C = diag.cands; if isstruct(C) && isfield(C,'pos'), C = vertcat(C.pos); end
        if ~isempty(C), plot(C(:,2), C(:,1), 'g.', 'MarkerSize',10); end
    end

    if ~isempty(diag) && isfield(diag,'selected') && ~isempty(diag.selected)
        S = diag.selected; if isstruct(S) && isfield(S,'pos'), S = S.pos; end
        plot(S(2), S(1), 'r*', 'MarkerSize',10,'LineWidth',1.2);
    end

    plot(currPos(2), currPos(1), 'co', 'MarkerSize',6,'LineWidth',1);

    riskTxt = '';
    if ~isempty(diag) && isfield(diag,'risk') && ~isnan(diag.risk)
        riskTxt = sprintf(', risk=%.3f', diag.risk);
    end
    title(sprintf('Reachable (outline), Cands (green), Sel (red*) | step=%d%s', step, riskTxt));
end
```

Figure 11. Full step to run the experiment

The entire code shown in Figure 11 is the core control program of the experiment. This part of the code connects all the components, including map generation, SLAM update, HPHS hierarchical partitioning and candidate target extraction, CVaR risk-sensitive selection, path planning, local/global backtracking mechanisms, as well as visualisation and result recording. It ensures that the robot can continuously build maps in an unknown environment. At the same time, the robot can choose targets, execute paths and avoid dead ends. After the operation is completed, it also outputs key indicators such as coverage rate, number of steps, and risk value. It is the central scheduler and verification platform of the entire autonomous exploration system.

4. Results

4.1. Map generation test

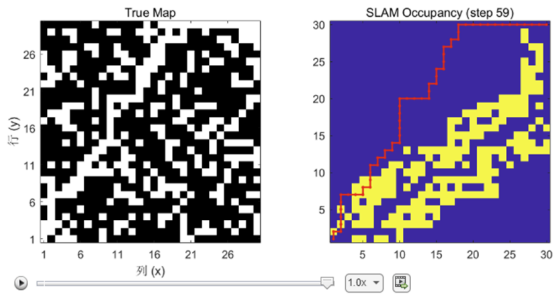


Figure 12. Map generation result

Randomly generated maps and the movement trajectories of the robots towards the exit. (The exploration trajectory of this part of the robot is generated based on the path of the trueMap and is moved accordingly. Robots do not possess functions such as autonomous exploration. This visual map, as shown in Figure 12, is used to test whether a connection path exists from the starting point to the key points when generating the map randomly. The exploration view of the robot is for detecting the generation of the robot.)

DOI: 10.54254/2755-2721/2025.LD28901

4.2. Frontier and A* test

Verify whether the entire route of "Frontier Detection - Select Target - A* Pathfinding" is functioning correctly. As can be seen from Figure 13, it is confirmed whether the frontier has been correctly identified, whether A * has provided the reachable path, and whether the coordinate system is consistent. Suppose the direction of the following figure is inconsistent with the above steps. In that case, it is most likely a problem related to the alignment of rows and columns versus x-y or the origin of the image coordinates.

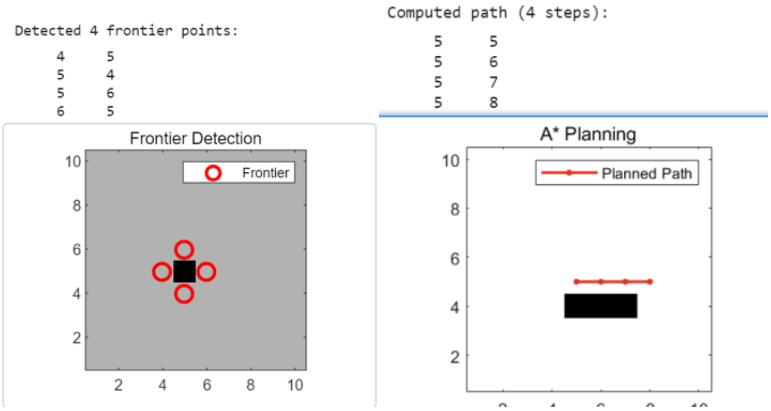


Figure 13. Frontier and A* planning test result

4.3. HPHS test

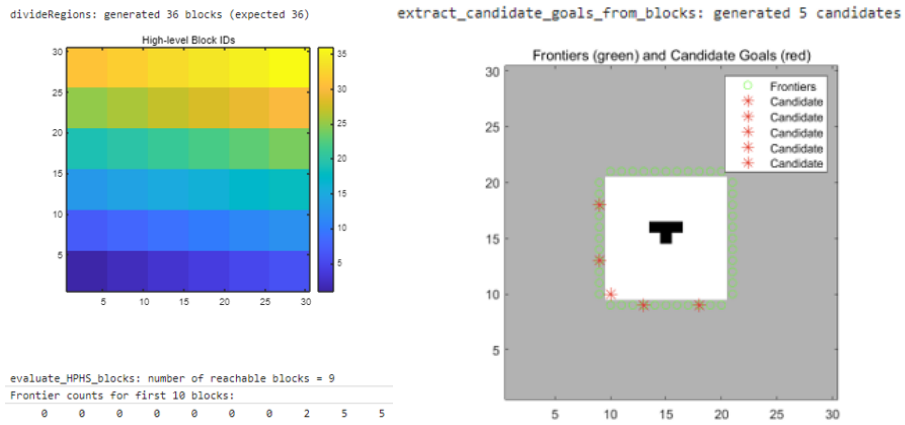


Figure 14. HPHS test result

This is the visual result of the unit test conducted by the HPHS senior planning team on the pipeline. Figure 14 shows that a 30×30 map is divided into 6×6 = 36 high-level regions (blocks) according to the block_size, which is used for the coarse-grained decision-making of HPHS. The middle text presents the contents of "reachable block" and "frontier count" respectively. The reachable block is the area that can be reached from the robot's current position and is evaluated on the occupancy graph using BFS/Connectivity. Currently, there are nine such reachable high-level areas. "Frontier counts for the first 10 blocks" refers to the number of frontiers detected within each block (the boundary points immediately adjacent to the unknown ones next to the free grid cells), and is used to measure the information gain of this area. In the final "Frontier & Candidate Goals",

DOI: 10.54254/2755-2721/2025.LD28901

the green dots represent the detected frontier pixels. Then the red crosses are the candidate target points obtained after clustering the frontiers within the reachable area. These candidate points will be handed over to the subsequent A* / trajectory planning to attempt to reach the destination. (For Test)

4.4. CVaR test

Figure 15 is the visual detection for the selection of risk-sensitive targets. The above shows the calculation of the CVaR risk for each candidate point using a radius of r = 3. Then list all the candidates, and finally select the target with the lowest risk (here, pos (8,12) was chosen, and its location in the upper-level block was displayed). This point was preset in the code in advance, to quickly verify whether the CVaR assessment and target selection are correct. (Risk value, tie-break in parallel cases, consistency of coordinates/reachability)

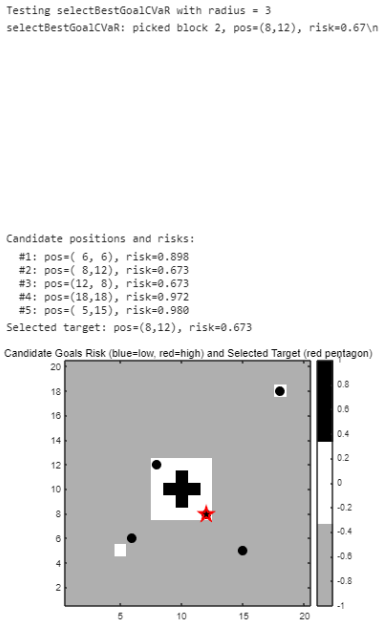


Figure 15. CvaR test result

4.5. Exploring the path and the occupancy map area

The four figures shown in Figure 16 represent the complete exploration visualisation results, incorporating the complete Occupancy Map, HPHS, CVaR, and Visited data. These four diagrams respectively represent trueMap, SLAM probability → occupancy map → trajectory and indicators. In the truthMap, white areas indicate walkable paths while black areas represent obstacles. The blue line in the figure represents the reference path that connects the starting point and the ending point. At the same time, the green box represents the target position of the robot. The image on the upper right is the SLAM probability. The updated occupancy probability grid (0-1 colour bar) after SLAM processing shows whether the sensor fusion has lowered the likelihood near the channel and raised the possibility of obstacles. The bottom left is the occMat from the robot's perspective. This is the occupancy map after thresholding the probabilities (-1 for unknown, 0 for idle, 1 for obstacle). The red line represents the planned/execution path, and the light blue dots are the leading or new observation points, used to compare whether the probability → discrete state is consistent. Finally, the bottom right is Trajectory Overlay. In the figure, the actual trajectory is superimposed on the true

DOI: 10.54254/2755-2721/2025.LD28901

value map (green line), and key indicators are given: moves=35 | frontiers=31 | mode=Astar | cov=39.6% (number of moves/number of frontiers / used planner/coverage rate). Through these four figures, one can check whether the coordinate system is aligned, whether the probability threshold is appropriate, whether the SLAM update converges, whether the path "goes through walls", and whether the exploration efficiency and other indicators are normal.

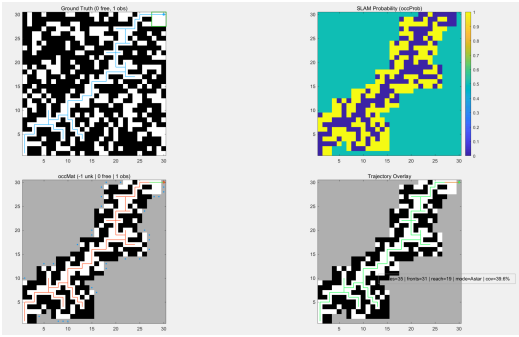


Figure 16. Exploration of the path and occupancy map

4.6. Exploration performance metrics

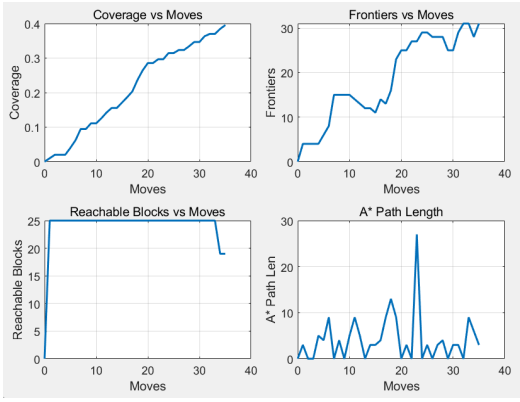


Figure 17. Exploring data visualisation

This is an indicator panel for the exploration process. In Figure 17, the key quantities of each step movement are plotted as curves over time, which makes it easy to determine whether the algorithm is functioning correctly. Coverage vs Moves: The coverage (known grid / total grid) increases with the number of steps, and the stability of perception + SLAM is examined and evaluated for its steady growth. Frontiers vs Moves: The number of frontiers changes with the number of steps, which is used to determine whether the emergence or disappearance of the information boundary is reasonable. Reachable Blocks vs Moves: In HPHS, "the number/proportion of high-level areas (blocks) that can be reached from the current position." Path Length*: The length of the A* path obtained after each re-planning, with fluctuations reflecting the complexity of target switching or obstacle avoidance; a value of 0 mainly indicates the continuation of the old path or the need for no further planning for the adjacent step.

4.7. High-level block IDs

Depending on the visible result of Figure 18, separating the 30×30 map evenly into a high-level grid according to block_size (here it is 5×5, totalling 25 blocks). The colour bars are merely block

DOI: 10.54254/2755-2721/2025.LD28901

numbers, indicating that the "block division" step is normal.

White = Known passable raster. The blue dot matrix box = the outer contour of the reachable block evaluated by HPHS. Green points = candidate frontier points extracted from these reachable blocks

The red five-pointed star = the target point selected by the CVaR in the current step from the candidates.

The robot creates a passable path along the diagonal. SLAM writes back, and the discretisation is regular. The reachability filtering of the HPHS system is working correctly. Blue dots can be seen roughly covering the blocks near this path to avoid the search expanding into unreachable areas. The candidates are mainly distributed at the edges of the passable zone, which is consistent with the definition of “frontier = free neighbours unknown”. In the picture, the red pentagram is selected in the exit direction of the unexplored area in the upper right corner. This result also reflects that the strategy combination is in line with expectations.

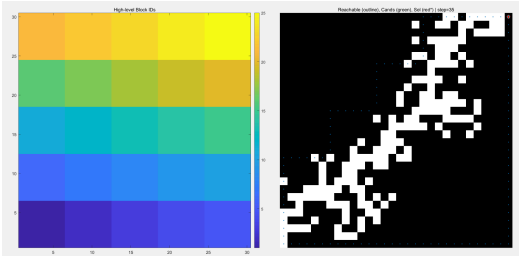


Figure 18. HPHS process visualisation

5. Discussion

Based on the experimental results, the system's closed-loop (frontier detection - HPHS high-level screening - CVaR risk assessment - A* planning - SLAM/occupation map update) operates stably in narrow and long channels. When the robot moves to the 35th step, the coverage rate rises to approximately 39.6%, and the number of frontiers increases to approximately 31. The entire robot's trajectory continues to advance along the diagonal passable area without experiencing long periods of oscillation or getting stuck in local optima. The A* The path length is mostly between 2 and 14 steps. There are some small moments during the process where the robot detours due to bottlenecks or rewrites new obstacles, resulting in a single peak (about 26 steps), but the overall process is acceptable. The HPHS system divides the 30×30 map into 5×5, totalling 25, high-level blocks. In each step of the system, only reachable blocks are generated as candidates and submitted to CVaR for point-level risk sorting, significantly compressing the target search space and enhancing directionality. From the running results, the blue contour of reachable blocks covers the passable area, while the candidates (green dots) are concentrated at the free-unknown boundary. Additionally, in the curve, "Reachable Blocks" is close to 25 in the early stage, and later slightly drops to approximately 19 as obstacles are rewritten and unknown areas shrink. This result is in line with the block size and environmental geometry. From the exponential trend, the coverage rate basically monotonically increases. A slight plateau appears near the 20th step of the robot's operation. The occurrence of this situation is primarily due to the band-shaped frontiers causing target switching and frequent re-planning. Subsequently, as the passable area expands, the path length continues to increase. The local peaks in path length and narrow bottlenecks/cost field updates correspond to them, without causing continuous stagnation. These phenomena collectively indicate that: the high-level HPHS reduces invalid expansions, CVaR avoids high-risk candidates at the point level, A* and

DOI: 10.54254/2755-2721/2025.LD28901

SLAM rewrites ensure reachability and consistency. According to the experimental results, the main limitations and improvement directions of the experiment are as follows: Firstly, the current high-level reachability is based on occPlanBin (unknown is considered passable), which is overly optimistic in the early stage and may be replaced with a conservative judgment based on occMat when obstacles are dense; Secondly, to reduce the plateau and path jitter, obstacles expansion and smoothing can be added to the cost map, and the occupancy probability can adopt a hysteresis threshold; Thirdly, it is recommended to adaptively integrate frontier density, risk, and distance to block entry weights in the HPHS scoring, and make simple online self-adjustments for block_size, risk_radius, and α of CVaR. Overall, the results support that the combination of "HPHS high-level constraints + CVaR risk sensitivity + A* execution" has good exploration efficiency and robustness in this type of channel environment; after the convergence of reachability determination and threshold details, there is still room for improvement in coverage efficiency and path stability.

6. Conclusion

Based on the experimental results of this project, in an environment that is entirely unknown and where the exit location is unknown, the robot's exploration process demonstrates continuous and stable performance improvement. As the number of steps increases, the coverage rate gradually rises and stabilises at approximately 35%; the number of detected leading points increases from less than 5 to approximately 30. The final reachable area remains at around 25; the A* path length fluctuates between 2 and 14 steps, reflecting the dynamic characteristics of local re-planning. Throughout the exploration process, good coherence is maintained. SLAM occupies the grid map and gradually approaches the real map. The trajectory covers the main channels and does not encounter long-term difficulties. Compared with mainstream methods, this framework has achieved optimisation in multiple aspects. Firstly, through the hierarchical region division of HPHS, redundant searches are effectively reduced. In addition, compared with traditional front exploration, it improves efficiency and stability. By combining the CVaR risk assessment mechanism, it effectively avoids entering high-risk dead ends. These are superior to methods based on information gain in terms of safety and robustness. At the same time, relying on SLAM and DFS backtracking mechanisms, this method can maintain exploration robustness without the high training cost of reinforcement learning. Based on this, future research can further expand this framework. For instance, the method that combines dynamic CVaR with deep reinforcement learning will be used. This method provides a solution for achieving an adaptive balance between risk and efficiency at different exploration stages. Additionally, the technology also emphasises that limited visibility, high structural repetitiveness, and unstable communication remain key challenges in underground and confined spaces. Therefore, the introduction of multimodal sensing and multi-robot collaboration mechanisms is necessary. This approach will further enhance the adaptability and robustness of this method in complex and unknown environments.

References

[1] Azpúrua, H., Cacace, J., Finzi, M., and Ortenzi, V. (2022). A survey on the autonomous exploration of confined subterranean environments. Robotics and Autonomous Systems, 160, 104304.
[2] Ebadi, H., et al. (2022) LAMP 2.0: Large-scale autonomous mapping and positioning for exploration. arXiv preprint arXiv: 2202.05253.
[3] Long, S., Zhou, W., Liu, H., and Zhang, X. (2024) HPHS: Hierarchical planning based on hybrid frontier sampling for unknown environments exploration. arXiv preprint arXiv: 2407.10660.

DOI: 10.54254/2755-2721/2025.LD28901

[4] Sarfi, M. H., and Bisheban, M. (2025) Risk-sensitive autonomous exploration of unknown environments: A deep reinforcement learning perspective. Journal of Intelligent & Robotic Systems, 111, 36.
[5] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016) Past, present, and future of simultaneous localisation and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32(6), 1309–1332.
[6] Yamauchi, B. (1997) A frontier-based approach for autonomous exploration. In Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 146–151. IEEE.
[7] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.
[8] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009) Introduction to algorithms (3rd ed.). MIT Press, Cambridge.
[9] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.
[10] Yamauchi, B. (1997) A frontier-based approach for autonomous exploration. In Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 146–151. IEEE.