Emerson Hall

Ramana M. Pidaparti

2 September 2025

Baseline Trajectory Optimization

Introduction

The purpose of this week's assignment was to implement and evaluate a baseline trajectory planner for an unmanned aerial vehicle (UAV) navigating an environment with static obstacles. This step builds on previous weeks, which defined the problem scope, identified evaluation metrics and created a MATLAB simulation environment. The goal was to design a simple but reliable method for collision free navigation that will serve as a benchmark for comparison with more advanced machine learning and optimization approaches in future work.

Methodology

A grid-based A* planner was selected as the baseline. The simulation environment was constructed in MATLAB to represent a bounded area with two primary obstacles (a vertical wall and a square block). Both obstacles were inflated by a safety margin of 0.25 meters to ensure that the UAV maintained clearance from their surfaces.

The environment was discretized into an occupancy grid with a resolution of 0.20 meters. Each grid cell was marked as free or occupied based on whether its center point lay inside the inflated obstacle polygons. The start location was set at coordinates [-8, -8] while the goal location was [10,4]. The planner used an eight connected grid structure meaning diagonal movements were allowed in addition to horizontal and vertical steps. The A* algorithm expanded nodes based on the lowest estimated total cost $f = g + h$, where g was the path cost from the start and h was the Euclidean heuristic estimate to the goal.

Results

The A* planner successfully produced a collision-free path from the start to the goal. The algorithm expanded 2,247 nodes during the search. The final smoothed path had a total length of 23.67 meters. The path avoided both the wall and the block while maintaining the required clearance and demonstrating that the inflation margin was effective.

The computational cost that is measured in the number of nodes expanded, reflects the relatively fine grid resolution and conservative safety margin. Although the search effort was high, the planner consistently converged to a feasible solution and the resulting trajectory was direct and efficient.

Figure 1 shows the environment layout which includes the inflated wall (blue), the square block (grey), the start (green dot) and goal positions (red X), and the planned A* trajectory. The path avoids obstacles while maintaining clearance which demonstrates the effectiveness of the inflation margin.

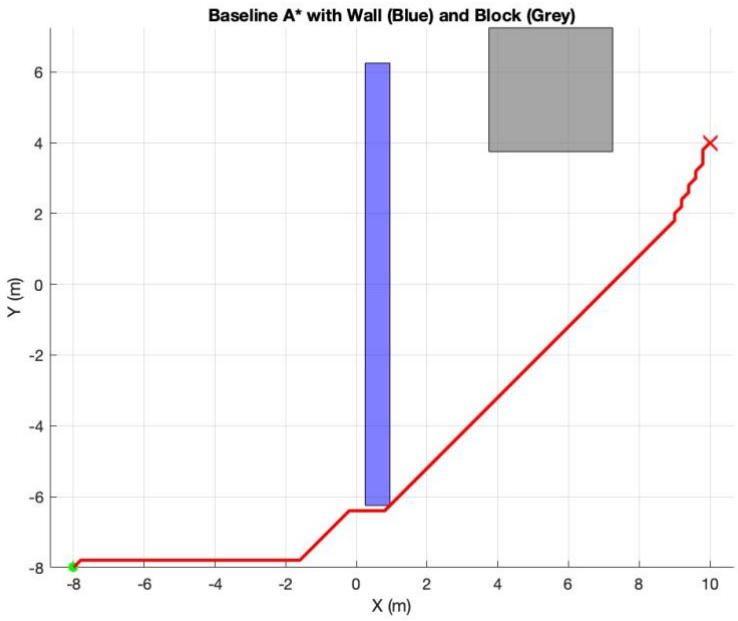

Figure 1: Planned trajectory using A with inflated obstacles*

Discussion

The A* planner proved effective as a baseline method. Its advantages are reliability, determinism and ease of implementation. The produced path length of 23.67 meters is reasonable given the obstacle layout and the solution demonstrates the algorithm's ability to balance path optimality with collision avoidance. However, the cost of 2,245 expanded nodes indicates that grid resolution strongly influences performance. Using a coarser grid could reduce computational effort at the expense of path precision.

The baseline highlights key tradeoffs in classical path planning that finer resolution increases safety and accuracy but requires more computation. Future comparisons with randomized planners such as RRT or optimization-based methods will show how these alternatives balance efficiency and adaptability.

Conclusion

A baseline trajectory planner was implemented using the A* algorithm on an occupancy grid representation of the environment. The planner successfully generated a collision free path that avoided static obstacles with a total length of 23.67 meters and 2,247 nodes expanded. These results provide a benchmark for evaluating the effectiveness of more advanced trajectory optimization techniques in subsequent project phases.

Appendix

A) MATLAB code

```
%% Week 4 — Baseline A*

clear; clc; close all;

% -------------------- Parameters --------------------
gridRes      = 0.20;
safetyMargin = 0.25;
xyMin = [-10 -10];
```

```
xyMax = [ 10  10];

startXY = [-8, -8];
goalXY  = [10,  4];

% -------------------- Obstacles --------------------
wall = [0.5 -6; 0.7 -6; 0.7 6; 0.5 6];
bldg = [4 4; 7 4; 7 7; 4 7];

inflateRect = @(poly,r) [ min(poly(:,1))-r, min(poly(:,2))-r;
                          max(poly(:,1))+r, min(poly(:,2))-r;
                          max(poly(:,1))+r, max(poly(:,2))+r;
                          min(poly(:,1))-r, max(poly(:,2))+r ];
wallInf = inflateRect(wall, safetyMargin);
bldgInf = inflateRect(bldg, safetyMargin);

% ----------------- Occupancy Grid -------------------
res = gridRes;
xv = xyMin(1):res:xyMax(1);
yv = xyMin(2):res:xyMax(2);
[XC, YC] = meshgrid(xv, yv);
occ = inpolygon(XC, YC, wallInf(:,1), wallInf(:,2)) | ...
     inpolygon(XC, YC, bldgInf(:,1), bldgInf(:,2));

mapRows = size(occ,1); mapCols = size(occ,2);

% -------------- Start/Goal to grid idx --------------
world2grid = @(xy) [ round((xy(2)-xyMin(2))/res)+1, ...
                     round((xy(1)-xyMin(1))/res)+1 ];
grid2world = @(rc) [ xyMin(1) + (rc(:,2)-1)*res, ...
                     xyMin(2) + (rc(:,1)-1)*res ];

sRC = world2grid(startXY); gRC = world2grid(goalXY);

% --------------------- A* Setup ---------------------
gScore = inf(mapRows, mapCols); gScore(sRC(1),sRC(2)) = 0;
fScore = inf(mapRows, mapCols);
heur = @(r,c) norm([r c] - gRC);
fScore(sRC(1),sRC(2)) = heur(sRC(1),sRC(2));

cameFrom = zeros(mapRows, mapCols, 2, 'int32');
openSet  = false(mapRows, mapCols);
openSet(sRC(1), sRC(2)) = true;

neighbors = [ -1 -1; -1 0; -1 1; 0 -1; 0 1; 1 -1; 1 0; 1 1 ];
stepCost  = [ sqrt(2); 1; sqrt(2); 1; 1; sqrt(2); 1; sqrt(2) ];
```

```
nodesExpanded = 0;

% ---------------------- A* Loop ----------------------
while true
   ftmp = fScore; ftmp(~openSet) = inf;
   [minVal, idxLin] = min(ftmp(:));
   if isinf(minVal), error('No path found.'); end
   [r, c] = ind2sub(size(occ), idxLin);
   openSet(r,c) = false;

   if r == gRC(1) && c == gRC(2)
      pathRC = [r c];
      while ~(r == sRC(1) && c == sRC(2))
         prev = double(squeeze(cameFrom(r,c,:))');
         if all(prev==0), break; end
         r = prev(1); c = prev(2);
         pathRC(end+1,:) = [r c]; %#ok<AGROW>
      end
      pathRC = flipud(pathRC);
      break;
   end

   nodesExpanded = nodesExpanded + 1;

   for k = 1:8
      rr = r + neighbors(k,1);
      cc = c + neighbors(k,2);
      if rr<1 || rr>mapRows || cc<1 || cc>mapCols, continue; end
      if occ(rr,cc), continue; end
      tentative = gScore(r,c) + stepCost(k);
      if tentative < gScore(rr,cc)
         cameFrom(rr,cc,:) = int32([r c]);
         gScore(rr,cc) = tentative;
         fScore(rr,cc) = tentative + heur(rr,cc);
         openSet(rr,cc) = true;
      end
   end
end

% ------------------- Path World ----------------------
pathXY = grid2world(pathRC);

% ---------------------- Metrics ----------------------
seg = diff(pathXY,1,1);
pathLen = sum( sqrt(sum(seg.^2,2)) );
```

```
fprintf('Nodes expanded: %d\n', nodesExpanded);
fprintf('Path length (m): %.2f\n', pathLen);

% ---------------------- Plot -------------------------
figure; hold on; axis equal;
patch(wallInf(:,1), wallInf(:,2), 'b', 'FaceAlpha',0.5, 'EdgeColor','k');   % wall blue
patch(bldgInf(:,1), bldgInf(:,2), [0.5 0.5 0.5], 'FaceAlpha',0.7, 'EdgeColor','k'); % block
grey
plot(startXY(1), startXY(2), 'go', 'MarkerFaceColor','g');
plot(goalXY(1),  goalXY(2),  'rx', 'LineWidth',2, 'MarkerSize',10);
plot(pathXY(:,1), pathXY(:,2), 'r-', 'LineWidth', 2);
xlabel('X (m)'); ylabel('Y (m)');
title('Baseline A* with Wall (Blue) and Block (Grey)');
grid on;
```