TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG

KHOA ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO CUỐI KỲ

Môn: Chuyên Đề 2 – Nhóm 19

Giảng viên hướng dẫn : Nguyễn Văn Hiếu

Sinh viên thực hiện : Mai Đình Tuấn Đạt - 106210209
: Nguyễn Ngọc Huy - 106210217
: Lê Tấn Duy - 106210212

Lớp : 21KTMT

Đà Nẵng,ngày 13 tháng 12 năm 2025

Mục Lục

Tên thành viên	Phân công công viêc	Tỉ lệ
Mai Đình Tuấn Đạt (Trưởng nhóm)	Thiết kế bản đồ kho hàng (Grid Map/Occupancy Grid). Xây dựng Class mô phỏng Robot (kích thước, vận tốc). Code phần hiển thị trực quan	33.5%
Nguyễn Ngọc Huy	Mô hình hóa toán học: Thiết lập hàm chi phí f(n), hàm heuristic h(n). Code thuật toán 1: A* (A-Star) Code thuật toán 2: RRT*. Đảm bảo code chạy đúng logic, không đâm vật cản.	33.5%
Lê Tấn Duy	Xây dựng module Data Logger: Ghi lại thời gian chạy, độ dài đường đi. Chạy các trường hợp test (Test cases) với các điểm Start/Goal, vật cản khác nhau. Vẽ biểu đồ so sánh và viết báo cáo phân tích.	33%

I. MÔ TẢ CÁC NỘI DUNG CHÍNH CỦA DỰ ÁN

1. Yêu Cầu Bài Toán & Tính Cấp Thiết:

Cuộc cách mạng Thương mại điện tử (E-commerce) đã dẫn đến sự ra đời và mở rộng quy mô của các trung tâm logistics và kho hàng tự động khổng lồ (ví dụ: Amazon, Tiki, Shopee). Nhu cầu vận chuyển hàng hóa từ điểm nhận về đến các kệ chứa và từ kệ chứa đến điểm đóng gói là liên tục với cường độ cao.

Trong lĩnh vực robot, việc tìm đường đi an toàn và hiệu quả trong môi trường có chướng ngại vật là một trong những thách thức quan trọng. Để giải quyết vấn đề này, các thuật toán điều hướng đã và đang được phát triển mạnh mẽ với mục tiêu giúp robot có khả năng tự chủ trong việc khám phá không gian, tránh vật cản và di chuyển đến vị trí mong muốn một cách tối ưu.

Bài toán đặt ra: Cần phát triển một hệ thống điều khiển Robot Tự Hành (Autonomous Mobile Robot - AMR) có khả năng:

- Tự lập kế hoạch đường đi (Path Planning): Tìm đường đi từ điểm xuất phát (Start - điểm nhận hàng) đến điểm đích (Goal - vị trí kệ hàng) một cách tối ưu (ngắn nhất về quãng đường hoặc thời gian).
- Tránh vật cản (Obstacle Avoidance): Bản đồ môi trường chứa cả vật cản cố định (tường, cột, kệ hàng) và vật cản động (công nhân, các robot khác). Đường đi tìm được phải an toàn, không xảy ra va chạm.

- Hoạt động hiệu quả: Thuật toán phải có thời gian tính toán nhanh để robot có thể phản ứng kịp thời với sự thay đổi của môi trường.

Tính cấp thiết:

- Tiết kiệm chi phí: Đường đi tối ưu giúp robot tiêu hao ít năng lượng (pin), giảm chi phí vận hành và bảo trì.
- Tăng năng suất: Thời gian di chuyển được rút ngắn tối đa, tăng số lượt vận chuyển trong một ca làm việc, từ đó nâng cao hiệu suất toàn kho.
- Giảm sai sót: Thay thế con người trong công việc di chuyển lặp đi lặp lại, giảm thiểu rủi ro nhầm lẫn vị trí và tai nạn lao động.
- Khả năng mở rộng: Là nền tảng cho việc điều phối đa robot (Multi-robot Coordination) trong tương lai.

2. Mô Hình Hệ Thống & Cách Thức Hoạt Động:

Mô hình hóa môi trường:

- Bản đồ lưới chiếm giữ (Occupancy Grid Map): Môi trường được rời rạc hóa thành một lưới các ô vuông. Mỗi ô có giá trị biểu thị trạng thái: 0 (trống), 1 (bị chiếm/vật cản), hoặc giá trị xác suất. Đây là mô hình phổ biến, dễ hiểu và tương thích với dữ liệu từ cảm biến (Lidar, Camera).
- Đồ thị (Graph): Các vị trí quan trọng (giao lộ, trước kệ hàng) được coi là các đỉnh (node). Các đường đi an toàn giữa chúng là các cạnh (edge) có trọng số là khoảng cách hoặc thời gian di chuyển.

Cách thức hoạt động của hệ thống:

- Đầu vào (Input): Tọa độ điểm đầu Start (x_s, y_s), điểm đích Goal (x_g, y_g), và bản đồ vật cản (dưới dạng ma trận lưới hoặc danh sách tọa độ).
- Xử lý (Process - Path Planning Module): Module lập kế hoạch đường đi, sử dụng một trong hai thuật toán A* hoặc RRT*, sẽ tính toán và tạo ra một lộ trình (path).
- Đầu ra (Output):
    + Dạng rời rạc: Tập hợp các tọa điểm liên tiếp [(x1,y1), (x2,y2), ..., (xn,yn)] mà robot cần đi qua.
    + Dạng liên tục (nâng cao): Có thể kết hợp thêm module làm trơn đường đi (Path Smoothing) (sử dụng Spline, Bezier curve) để biến đổi đường đi rời rạc, gấp khúc thành quỹ đạo mượt mà (x(t), y(t)), từ đó tính toán vận tốc dài v và vận tốc góc ω phù hợp cho robot.

Điểm mới đề xuất (Novelty Suggestions):

Tích hợp cơ chế làm trơn đường đi (Smooth Path): Sau khi có đường đi từ A* hoặc RRT*, áp dụng thuật toán làm trơn (ví dụ: Gradient Descent cho trajectory optimization) để tạo ra đường cong mượt, giúp robot di chuyển ổn định, giảm hao mòn phần cơ và tiết kiệm năng lượng khi vào cua.

II. MÔ TẢ VÀ GIẢI QUYẾT BÀI TOÁN

1. Mô Hình Hóa Bài Toán:

1.1. Không gian làm việc và ràng buộc:

Không gian cấu hình (Configuration Space):

- Không gian 2D: $C = [0, W] \times [0, H]$, với W, H là chiều rộng và chiều cao khu vực kho hang
- Trong triển khai code: $W = H = 800$ pixel, tương ứng với SCREEN_WIDTH và SCREEN_HEIGHT

Không gian vật cản (Obstacle Space):

- Tập hợp các vị trí bị chiếm bởi vật cản: $C_{obs} \subset C$
- Vật cản bao gồm: tường, kệ hàng, thiết bị cố định
- Trong code: Vật cản được biểu diễn bằng các ô màu đen (BLACK) trong lưới

Không gian tự do (Free Space):

- Không gian robot có thể di chuyển: $C_{free} = C \setminus C_{obs}$
- Điểm bắt đầu: $q_{start} \in C_{free}$ (màu cam - ORANGE)
- Điểm đích: $q_{goal} \in C_{free}$ (màu ngọc lam - TURQUOISE)

1.2. Mô hình hóa đường đi:

Một đường đi σ từ start đến goal là một ánh xạ liên tục:

$$\sigma: [0,1] \rightarrow C_{free} \qquad \text{sao cho} \quad \sigma(0) = q_{start}, \sigma(1) = q_{goal}$$

Trong mô hình rời rạc (A*):

- Không gian được lượng tử hóa thành lưới $N \times N$ ô vuông
- Mỗi ô là một đỉnh trong đồ thị:

$$v_{i,j} \in V, \text{ với } i, j \in \{0,1,\dots,N-1\}$$

- Các cạnh nối giữa các ô kề nhau (4 hướng):

$$E = \{(v_{i,j}, v_{i',j'}) \mid\mid i - i' \mid + \mid j - j' \mid = 1\}$$

- Trọng số cạnh:

$$w(v_{i,j}, v_{i',j'}) = 1 \text{ (khoảng cách Manhattan)}$$

$$\text{hoặc } \sqrt{(i - i')^2 + (j - j')^2} \text{ (khoảng cách Euclid)}$$

Trong mô hình liên tục (RRT*):

- Không gian làm việc liên tục:

$$(x, y) \in \mathbb{R}^2$$

- Đường đi được biểu diễn bằng chuỗi các điểm liên tục:

$$\sigma = \{q_0, q_1, \ldots, q_k\} \text{ với } q_0 = q_{start}, q_k = q_{goal}$$

1.3. Hàm mục tiêu (Objective Function):

Bài toán cần cực tiểu hóa hàm chi phí tổng:

$$J(\sigma) = \sum_{i=0}^{k-1} c(q_i, q_{i+1})$$

với $c(q_i, q_{i+1})$ là chi phí di chuyển từ q_i đến q_{i+1}.

Trong triển khai:

- A*: Sử dụng hàm heuristic Manhattan:

$$c_{\text{A}*}(q_i, q_j) = \mid x_i - x_j \mid + \mid y_i - y_j \mid$$

- RRT*: Sử dụng khoảng cách Euclid:

$$c_{\text{RRT}*}(q_i, q_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- Độ dài đường đi cuối cùng được tính chính xác bằng khoảng cách Euclid giữa các điểm liên tiếp

2. Trình Bày Các Phương Pháp Giải Quyết:

a. Thuật Toán A* (A-Star):

A* tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A* xây dựng tất cả các đường đi có thể đi dùng hàm ước lược khoảng cách (hàm Heuri stic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

Hàm đánh giá tổng thể cho mỗi bước di chuyển được tính bằng:

$$f(n) = g(n) + h(n)$$

Trong đó:

- g(n): biểu diễn chi phí thực tế từ điểm xuất phát đến nút n.
- h(n): Hàm heuristic ước lượng chi phí từ n đến đích.

$$h(n) = \sqrt{(x_n - x_{goal})^2 + (y_n - y_{goal})^2}$$

Thuật toán không xem xét các vật cản nằm giữa hai điểm khi tính heuristic, mà chỉ sử dụng khoảng cách trực tiếp làm cơ sở ước lượng. Robot sẽ lựa chọn bước đi có giá trị f(n) nhỏ nhất, tức là vị trí gần đích nhất theo đánh giá tổng thể, và di chuyển đến đó.

Cách thức hoạt động:

Khởi Tạo

- Đặt điểm bắt đầu vào danh sách mở (open list) và gán giá trị f ban đầu cho nó.
- Đặt danh sách đóng (closed list) rỗng.

Lặp Lại

- Chọn nút trong danh sách mở có giá trị f nhỏ nhất và di chuyển nó sang danh sách đóng.
- Nếu nút này là điểm đích, thì đường đi ngắn nhất đã được tìm thấy.
- Nếu không, đối với mỗi nút kề cận:
    + Tính giá trị g và f cho nút kề.

+ Nếu nút này chưa có trong danh sách mở hoặc có giá trị f nhỏ hơn giá trị hiện tại trong danh sách mở, thì cập nhật giá trị và đặt nút này vào danh sách mở.

b. Thuật Toán RRT (Rapidly-exploring Random Tree Star)*

Thuật toán RRT* (RRT Star) là một phiên bản tối ưu hóa của thuật toán RRT (Rapidly-exploring Random Tree), một phương pháp lập kế hoạch chuyển động (motion planning) phổ biến, sử dụng cây ngẫu nhiên để tìm đường đi trong không gian có chướng ngại vật; điểm khác biệt cốt lõi của RRT* là nó liên tục cải thiện chất lượng đường đi bằng cách "rewiring" (nối lại) các nhánh cây khi tìm thấy đường ngắn hơn, đảm bảo đường đi tìm được sẽ tiến tới tối ưu (ngắn nhất) khi số lượng mẫu tăng lên.

Là thuật toán lập kế hoạch dựa trên mẫu ngẫu nhiên (sampling-based), xây dựng một cây trong C_free hướng về phía mục tiêu. RRT* có tính chất hội tụ tiệm cận đến đường đi tối ưu (asymptotically optimal).

Cách thức hoạt động :

- Khởi tạo cây T chỉ có node gốc q_start.
- Lặp lại N lần:
  + Lấy mẫu (Sample): Lấy một điểm ngẫu nhiên q_rand trong C_free.
  + Tìm node gần nhất (Nearest): Tìm node q_near trong T gần q_rand nhất.
  + Hướng về mẫu (Steer): Từ q_near, tạo một điểm mới q_new trên đoạn thẳng nối q_near và q_rand với một khoảng cách bước (step_size) cố định.
  + Kiểm tra va chạm (Collision Check): Kiểm tra xem đoạn (q_near, q_new) có nằm trong C_free không.
  + Tìm các node lân cận (Near): Tìm tập Q_near các node trong T nằm trong bán kính r(n) xung quanh q_new.
  + Chọn cha tối ưu (Choose Parent): Duyệt Q_near để tìm node q_min sao cho đường đi từ q_start qua q_min đến q_new là ngắn nhất và không va chạm.
  + Thêm node vào cây (Insert): Thêm q_new vào T, nối q_new với q_min.
  + Tối ưu hóa lại cây (Rewire): Với mỗi q_near trong Q_near, kiểm tra xem nếu đường đi từ q_start qua q_new đến q_near ngắn hơn đường

đi hiện tại của q_near, thì cập nhật lại cha
của q_near thành q_new (tối ưu hóa cây).

Điểm mới áp dụng: Triển khai cơ chế Goal Biasing (ví dụ: 5% xác suất lấy q_rand trùng với q_goal) để tăng tốc độ hội tụ của cây về phía mục tiêu. So sánh hiệu quả với RRT* cơ bản.

3. Đánh Giá Độ Phức Tạp & Tốc Độ Hội Tụ:

Thuật toán A*:

- Độ phức tạp thời gian: O(b^d) trong trường hợp xấu nhất, với b là hệ số nhánh (số node con trung bình) và d là độ sâu của lời giải. Tuy nhiên, với heuristic tốt (như Euclid), hiệu suất được cải thiện rất lớn.
- Tốc độ hội tụ: Hội tụ nhanh đến lời giải tối ưu toàn cục nếu heuristic là chấp nhận được (không overestimate) và nhất quán.
- Nhược điểm: Hiệu suất giảm mạnh khi kích thước bản đồ (số node) rất lớn. Việc lưu trữ toàn bộ OPEN_LIST và CLOSED_LIST có thể tốn bộ nhớ (O(n)).

Thuật toán RRT*:

- Độ phức tạp: Mỗi vòng lặp chính yêu cầu tìm kiếm láng giềng (Near), với cấu trúc dữ liệu thông thường là O(n). Sử dụng cấu trúc kd-tree có thể giảm xuống còn O(log n).
- Tốc độ hội tụ: Hội tụ chậm hơn A* trong không gian nhỏ và rời rạc. Tuy nhiên, ưu điểm lớn là khả năng hội tụ tiệm cận đến tối ưu khi số lượng mẫu N -> ∞. Hiệu suất vượt trội trong không gian liên tục, có chiều cao và không cần rời rạc hóa toàn bộ không gian.

III. KẾT QUẢ VÀ ĐÁNH GIÁ

1. Các Metrics và Thông số đánh giá hệ thống:

Để đảm bảo tính khách quan trong việc so sánh hiệu năng giữa thuật toán tìm kiếm dựa trên đồ thị (A*) và thuật toán dựa trên lấy mẫu (RRT*), hệ thống sử dụng 03 chỉ số đo lường (metrics) chính:

- Thời gian thực thi (Avg Execution Time - ms): Đo lường thời gian CPU cần thiết để tìm ra đường đi từ điểm bắt đầu đến điểm đích. Chỉ số này càng thấp càng tốt, thể hiện khả năng đáp ứng thời gian thực.
- Độ dài đường đi (Avg Path Length - pixels): Tổng chiều dài quãng đường robot phải di chuyển. Chỉ số này càng thấp càng tốt, thể hiện tính tối ưu về năng lượng và chi phí vận hành.
- Tỷ lệ thành công (Success Rate - %): Tỷ lệ số lần thuật toán tìm được đường đi thành công trên tổng số lần thử nghiệm (n=10 lần/kịch bản). Chỉ số này đánh giá độ tin cậy của thuật toán.

2. Mô tả dữ liệu kết quả:

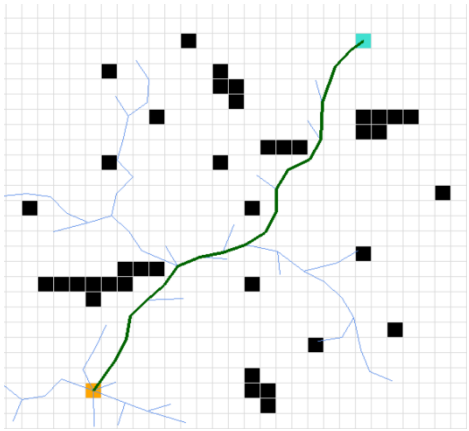
Hệ thống không chỉ hiển thị trực quan quá trình tìm đường trên giao diện đồ họa (GUI) mà còn lưu trữ và tổng hợp dữ liệu sau mỗi lần chạy thử nghiệm:

- Dữ liệu Runtime (Lịch sử chạy): Hệ thống lưu trữ mảng danh sách history bao gồm: thời gian xử lý, độ dài đường đi và trạng thái (Thành công/Thất bại) của n lần chạy gần nhất.
- Biểu đồ trực quan (Visual Charts): Kết quả được xuất ra dưới dạng biểu đồ cột so sánh trực tiếp (như hình minh họa), cập nhật theo thời gian thực (Real-time plotting) giúp người vận hành dễ dàng nhận diện sự chênh lệch hiệu năng.

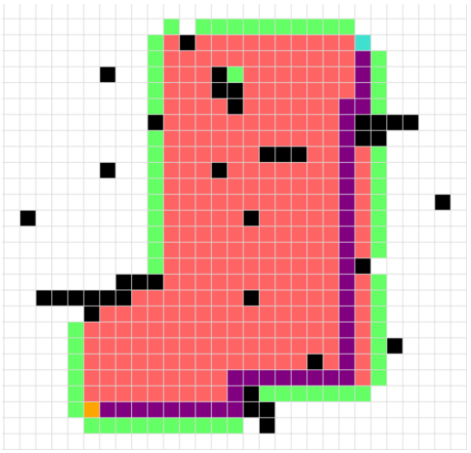
3. Phân tích kết quả chi tiết:

Dựa trên dữ liệu thực nghiệm từ 10 lần chạy (n=10) cho mỗi kịch bản, ta có các phân tích sau:

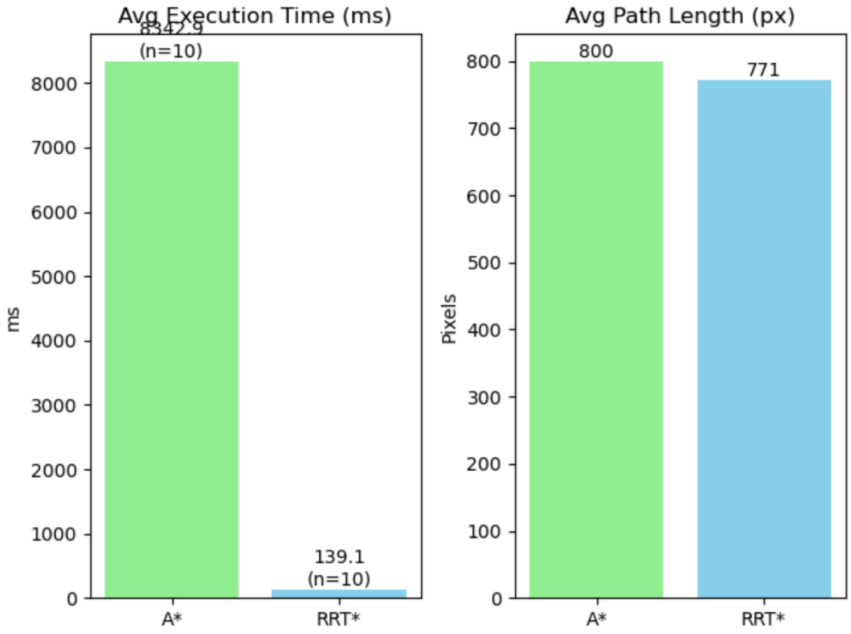
3.1. Trường hợp 1: Môi trường Ngẫu nhiên/Thoáng (Random Case):



*Thuật toán RRT**



*Thuật toán A**



Biểu đồ so sánh thời gian và độ dài đường đi của 2 thuật toán trong trường hợp Random Obstacle

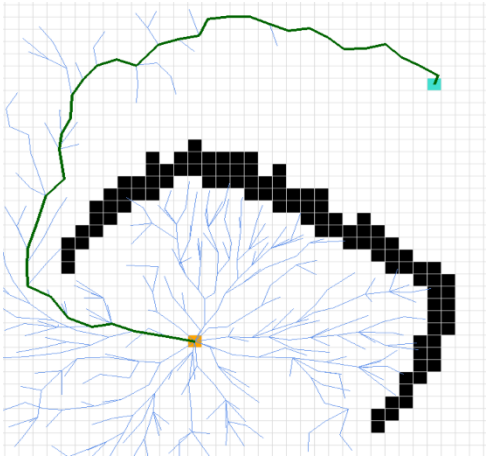
Thời gian: Sự chênh lệch là cực kỳ lớn. A* mất trung bình 8342.9 ms, trong khi RRT* chỉ mất 139.1 ms. RRT* nhanh hơn A* khoảng 60 lần.

Độ dài đường đi: RRT* (771 px) có đường đi ngắn hơn A* (800 px) một chút.

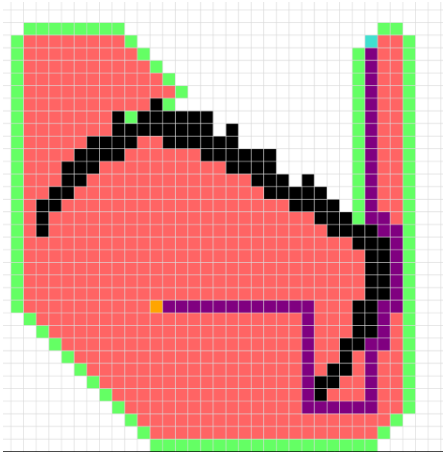
Lý Do: Vì A* bị ràng buộc bởi các ô lưới (Grid-based) nên đường đi thường bị gấp khúc theo góc 45 hoặc 90 độ. Ngược lại, RRT* hoạt động trong không gian liên tục nên có thể tạo ra các đường cắt chéo tối ưu hơn.

Kết luận: Trong không gian rộng, ít vật cản, RRT* vượt trội hoàn toàn về tốc độ và vẫn đảm bảo đường đi tối ưu.

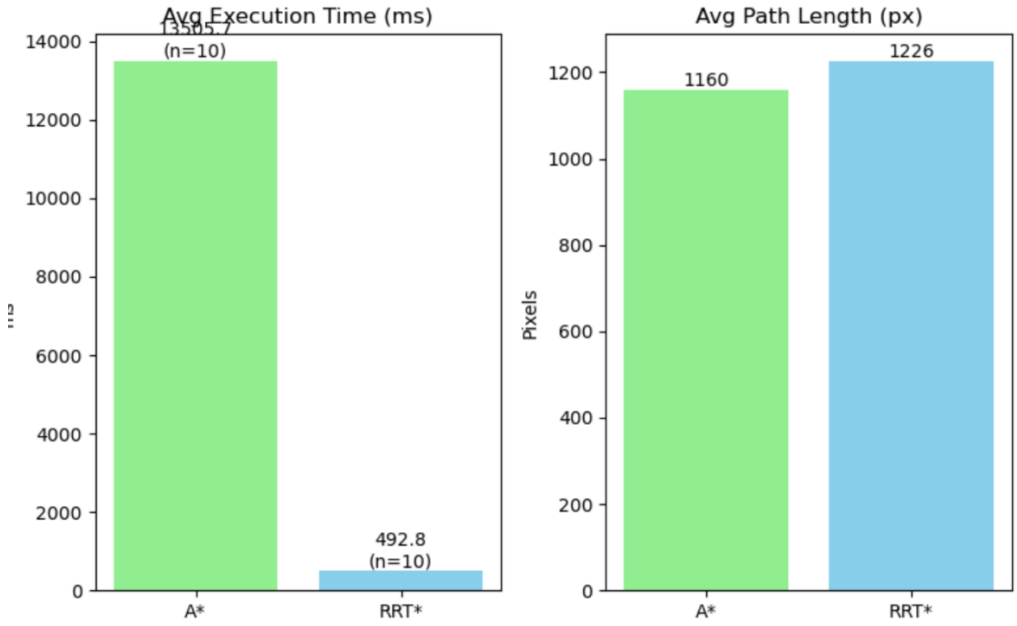
3.2. Trường hợp 2: Vật cản hình chữ U (Local Minima Trap):



*Thuật toán RRT**



*Thuật toán A**



Biểu đồ so sánh thời gian và độ dài đường đi của 2 thuật toán trong trường hợp Local Minima Trap

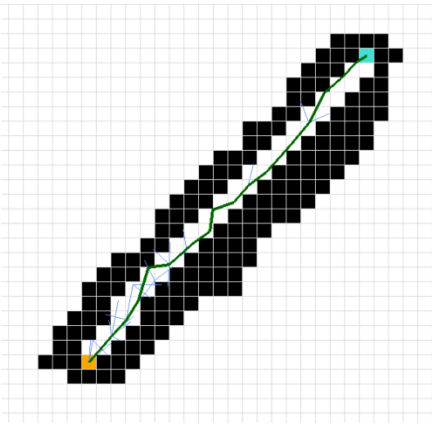
Thời gian: A* tăng vọt lên 13,505.7 ms do phải duyệt loang (flood fill) toàn bộ khu vực "bẫy" bên trong chữ U trước khi thoát ra ngoài. RRT* vẫn giữ tốc độ ấn tượng 492.8 ms.

Độ dài đường đi: A* (1160 px) tốt hơn RRT* (1226 px).

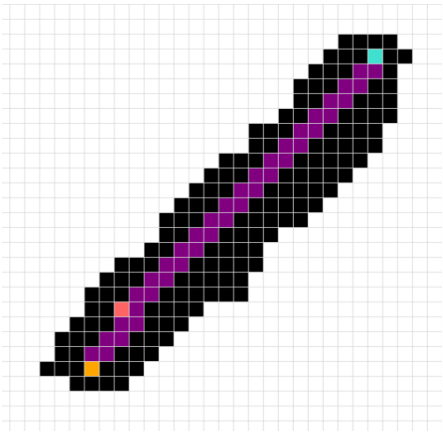
Lý do: RRT* do tính chất ngẫu nhiên nên đường đi có thể bị vòng vèo, không bám sát chướng ngại vật tốt như A*. Tuy nhiên, sự chênh lệch này không đáng kể (khoảng 5%).

Kết luận: A* đảm bảo tìm ra đường ngắn nhất nhưng chi phí tính toán quá lớn khi gặp vật cản phức tạp. RRT* cân bằng tốt giữa tốc độ và độ dài đường đi.

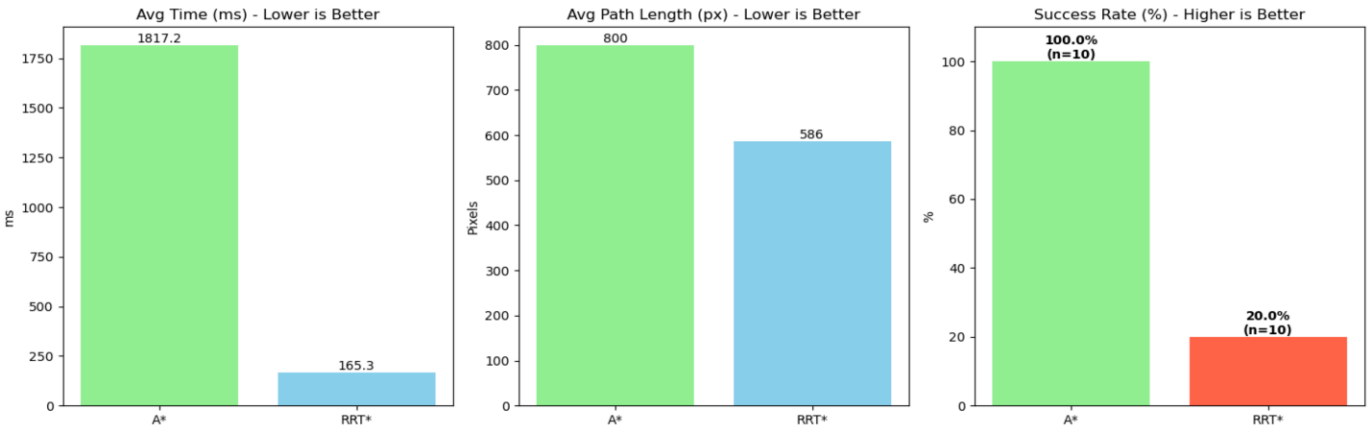
3.3. Trường hợp 3: Khe hẹp (Narrow Passage):



*Thuật toán RRT**



*Thuật toán A**



Biểu đồ so sánh thời gian, độ dài đường đi và tỉ lệ thành công của 2 thuật toán trong trường hợp Narrow Passage

Tỷ lệ thành công (Success Rate): Đây là chỉ số quan trọng nhất.

- A* đạt 100%:* Luôn luôn tìm thấy đường đi nếu tồn tại.
- RRT* chỉ đạt 20%:* Thất bại 8/10 lần thử nghiệm.

Phân tích: RRT* dựa trên xác suất lấy mẫu ngẫu nhiên. Xác suất để "bắn" được một điểm mẫu (sample) lọt chính xác vào khe hẹp là cực kỳ thấp. Do đó, thuật toán thường xuyên bị kẹt hoặc hết số vòng lặp tối đa mà chưa qua được khe.

Kết luận: RRT* không ổn định và không phù hợp cho môi trường có các khe hẹp hoặc hành lang nhỏ, trong khi A* thể hiện sự ổn định tuyệt đối.

4. Đánh giá tính hợp lý, ổn định và so sánh phương pháp:

4.1. Tính hợp lý và Ổn định:

A* (A-Star):

- Tính ổn định: Cao (Deterministic). Với cùng một đầu vào, kết quả luôn giống nhau 100%.
- Tính đầy đủ: Có (Complete). Luôn tìm ra đường đi ngắn nhất nếu tồn tại.
- Nhược điểm: Tốn kém tài nguyên tính toán và bộ nhớ khi không gian tìm kiếm lớn (thể hiện qua thời gian chạy > 13s ở kịch bản U-shape).

RRT* (RRT-Star):

- Tính ổn định: Thấp (Stochastic). Mỗi lần chạy cho ra một đường đi và thời gian khác nhau.
- Tính đầy đủ: Theo xác suất (Probabilistically Complete). Khi số lần lặp tiến tới vô cùng, xác suất tìm thấy đường tiến tới 1. Tuy nhiên trong thực tế giới hạn số vòng lặp, thuật toán dễ thất bại ở các khe hẹp.

4.2. So sánh Môi trường & Ngôn ngữ:

Ngôn ngữ Python: Việc sử dụng Python giúp triển khai nhanh (Rapid Prototyping) và dễ dàng trực quan hóa dữ liệu. Tuy nhiên, Python là ngôn ngữ thông dịch (interpreted), dẫn đến tốc độ thực thi của A* khá chậm (lên tới hàng chục giây).

Để ứng dụng vào Robot thực tế (Real-time AMR), cần chuyển đổi mã nguồn sang C++ và tích hợp vào hệ điều hành Robot ROS 2. Điều này sẽ giảm thời gian tính toán của A* từ hàng nghìn ms xuống chỉ còn vài chục ms.

5. Kết luận chung:

Dự án đã thực hiện thành công việc mô phỏng và so sánh hai thuật toán:

Chọn A* khi: Môi trường chật hẹp, nhiều mê cung, yêu cầu bắt buộc phải tìm được đường đi ngắn nhất và sự ổn định cao (ví dụ: Robot hút bụi, Robot trong kho hàng lối đi nhỏ).

Chọn RRT* khi: Môi trường rộng lớn, không gian mở, yêu cầu phản ứng nhanh và chấp nhận đường đi gần đúng (ví dụ: Drone bay ngoài trời, Xe tự hành ngoài xa lộ).