Tạp chí Khoa học Trường Đại học Cần Thơ
website: sj.ctu.edu.vn

ỨNG DỤNG ĐỒ THỊ TẦM NHÌN VÀO BÀI TOÁN TÌM ĐƯỜNG CHO ROBOT

Trần Thị Như Nguyệt[1], Vũ Đức Lung[1] và Trần Văn Hoài[2]

[1] *Khoa Kỹ thuật Máy tính, Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Hồ Chí Minh*
[2] *Khoa Khoa học và Kỹ thuật Máy tính, Trường Đại học Bách Khoa, Đại học Quốc gia Hồ Chí Minh*

Thông tin chung:
Ngày nhận: 18/03/2014
Ngày chấp nhận: 30/06/2014

Title:
Applying visibility-graph to the shortest path problem for robots

Từ khóa:
Đồ thị tầm nhìn, đường dẫn ngắn nhất, vật cản, hoạch định tối ưu toàn cục, Lego-Mindstorm NXT, bộ điều khiển PID

Keywords:
Visibility-graph, shortest-path, obstacles, global optimization planning, Lego-Mindstorm NXT, PID controller

ABSTRACT

This paper presents a solution for global optimized path planning with respect to finding the shortest distance for autonomous robotic system, particularly in two-dimensional space with a set of obstacles. The proposed approach is based on visibility-graph and the literature review of path planning is presented in details to explain why this approach is used. Through pros, cons, and complexity in the construction of a visibility-graph, the paper proposed two simple and efficient techniques to significantly reduce computation time in building a visibility-graph in the case of numerous obstacles. Finally, the method of how to control the robot follow exactly the identified path is shown. The experimental results, with a real robot, show that the proposed approach is efficient, feasible and straightforward to apply in practice.

TÓM TẮT

Bài báo trình bày giải pháp cho bài toán xác định đường đi ngắn nhất toàn cục, tránh các vật cản trong không gian hai chiều được áp dụng và kiểm thử trên robot thật. Cách tiếp cận sử dụng là dựa vào đồ thị tầm nhìn (visibility-graph). Thông qua ưu khuyết điểm và độ phức tạp trong việc xây dựng một visibility-graph, bài báo đề xuất hai phương pháp đơn giản, hiệu quả, giúp giảm đáng kể thời gian xây dựng một visibility-graph trong trường hợp môi trường chứa nhiều vật cản. Ngoài ra, bài báo cũng giới thiệu một phương pháp giúp robot di chuyển chính xác theo đường đi đã hoạch định trước. Kết quả thực nghiệm cho thấy giải pháp đề nghị này khá hiệu quả, khả thi và có thể áp dụng dễ dàng trong thực tế.

1 GIỚI THIỆU

Giả sử một robot tự hành (bao gồm tự hành trên mặt đất và trên không) tự do di chuyển trong môi trường với nhiều vật cản, như thế nào để tìm ra đường dẫn tốt nhất với một vài tiêu chuẩn tối ưu nào đó như ngắn nhất về mặt khoảng cách, ngắn nhất về mặt thời gian, tiêu tốn năng lượng ít nhất để robot có thể di chuyển dọc theo đường này từ điểm bắt đầu đến điểm kết thúc mà không va vào các vật cản trên đường đi trong môi trường hai hoặc nhiều chiều. Môi trường hoạt động có thể tĩnh (các vật cản cố định và biết trước) hoặc động (các vật cản có thể không cố định hoặc không biết trước). Đây là vấn đề gặp phải khá thường xuyên trong xây dựng robot tự hành và mục đích của bài toán tìm đường (path-planning) là giải quyết các vấn đề tương tự như thế.

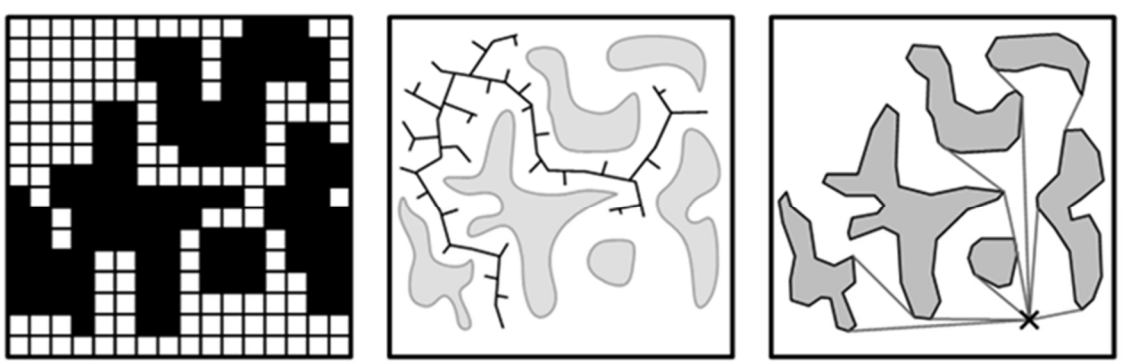
Một trong những giải pháp cơ bản cho vấn đề này thường là đi-và-tránh, tức robot cứ di chuyển và gặp vật cản thì tránh với một góc nào đó rồi sau đó hướng tới đích và đi tiếp. Tuy nhiên, nếu một

đường dẫn tối ưu toàn cục theo một tiêu chuẩn nào đó được tính toán trước và robot di chuyển theo thì kết quả sẽ tốt hơn rất nhiều. Robot không dừng lại ở phần cơ điện mà tính thông minh trong bộ não cũng sẽ phát triển mạnh dựa vào việc kết hợp này.

Các giải pháp đã công bố cho bài toán này nhìn chung thường sử dụng một trong ba hướng tiếp cận: dựa trên tìm kiếm (search-based), dựa trên lấy mẫu (sampling-based), hoặc phương pháp tổ hợp (combinatorial-planning) [S. M. Valle, 2006; S.K. Ghosh, 2007]. Trong ba phương pháp, dựa trên lấy mẫu và dựa trên tìm kiếm được cho là nhanh và dễ dàng thực hiện, còn dựa trên tổ hợp sẽ tiêu tốn một khoảng thời gian tính toán khá lớn nếu môi trường chứa nhiều vật cản. Tuy nhiên, hai phương pháp đầu, do dựa vào xác suất và tính ngẫu nhiên, sau một khoảng thời gian chạy có thể vẫn chưa cho ra lời giải và sẽ không chắc chắn là liệu đang có một đường dẫn tốt nhất tồn tại hay không, thời gian chạy đã đủ chưa. Trong khi đó, ưu điểm lớn nhất của dựa trên tổ hợp là luôn biết được đường đi tốt nhất như mong muốn tồn tại hay không và đó là đường nào sau khi giải thuật hoàn tất, do chỉ sử dụng các lý thuyết hình học. Chính vì đặc tính này mà hầu hết các nhân hoặc bộ não của các hệ thống robot mạnh luôn cố gắng hiện thực một bộ xác định đường đi được hiện thực theo hướng tiếp cận này, phòng trường hợp cần thiết [S.K. Ghosh, 2007]. Ngoài ra, hoạch định đường đi dựa trên tổ hợp có khả năng ứng dụng rộng rãi hơn các phương pháp khác, có thể được áp dụng ở nhiều lĩnh vực khác như: đồ họa máy tính, VLSI, GIS.

Từ lý do này, bài báo trình bày một bài toán tìm đường tối ưu được áp dụng thực tế vào robot thật với cách tiếp cận là dựa theo tổ hợp. Có nhiều mục tiêu tối ưu khác nhau, mục tiêu cụ thể mà bài báo này giải quyết là tìm đường đi ngắn nhất về mặt khoảng cách cho robot điểm; robot di chuyển trong môi trường chứa nhiều vật cản; các vật cản được xem như các đa giác (bao gồm cả lồi và lõm). Nội dung công việc có thể phân làm hai công đoạn chính: (1) như thế nào để tính toán đường đi ngắn nhất cho robot và (2) sau khi đường đi ngắn nhất dự kiến đã tính toán được, như thế nào để robot bám đường một cách chính xác. Về nội dung (1), bài báo sử dụng đồ thị tầm nhìn (visibility-graph) để dựa vào đó tính toán ra đường đi ngắn nhất. Như thế nào để tạo thành một visibility-graph đã được trình bày trong nhiều công bố trước đây, bài báo sử dụng phương pháp của Welzl [S.K. Ghosh, 2007; E. Welzl, 1985] với độ phức tạp $\theta(n^2)$. Và trong trường hợp môi trường hoạt động của robot lớn, với số vật cản quá nhiều, bài báo cũng giới thiệu hai phương pháp đơn giản nhưng hiệu quả giúp giảm đáng kể thời gian tính toán visibility-graph. Về nội dung (2), một giải pháp bám đường dễ dàng áp dụng cho nhiều loại robot thật, sử dụng ý tưởng từ lý thuyết điều khiển PID (proportional-integral-derivative) cũng như các kết quả thực nghiệm được trình bày chi tiết.

Phần còn lại của bài báo bao gồm bốn phần. Phần II trình bày tổng quan về path-planning, ý tưởng của các hướng tiếp cận; khái niệm cũng như chi tiết về các công bố liên quan đến việc sử dụng visibility-graph cũng được liệt kê rõ. Phần III giới thiệu hai phương pháp đóng góp của bài báo giúp giảm thời gian xây dựng một visibility-graph. Phần IV hướng dẫn như thế nào điều khiển việc bám đường chính xác cho robot. Và phần cuối cùng là kết quả thực nghiệm của tất cả các công việc nêu trên.



Hình 1: Hình ảnh mô phỏng ba hướng tiếp cận của path-planning [S. M. Valle, 2006]

2 TỔNG QUAN VỀ CÁC GIẢI THUẬT XÁC ĐỊNH ĐƯỜNG ĐI (PATH-PLANNING) ĐÃ ĐƯỢC NGHIÊN CỨU

2.1 Các hướng nghiên cứu của path-planning

2.1.1 Dựa trên tìm kiếm

Ý tưởng cơ bản của phương pháp này là chia không gian môi trường thành một lưới với từng ô kích thước đều nhau. Đích đến và vị trí robot được biết trong lưới. Một quá trình tìm kiếm (ví dụ như giải thuật D*) được thực hiện trên lưới để giải quyết vấn đề tìm đường đi, như hình 1(a). Ưu điểm của phương pháp này là quá trình tìm kiếm được thực hiện trên lưới, điều này giúp giảm bớt gánh nặng của việc duy trì một cấu trúc đồ thị trong quá trình hiện thực giải thuật. Lưới chính là đồ thị. Tuy nhiên, đây cũng là khuyết điểm của bài toán khi cả không gian bài toán chỉ được thể hiện với một lưới. Việc phân giải lưới phải được chỉ rõ, kết quả bài toán tối ưu đến mức nào phụ thuộc hoàn toàn vào sự phân giải lưới [D. Ferguson and A.Stentz, 2006].

2.1.2 Dựa trên lấy mẫu

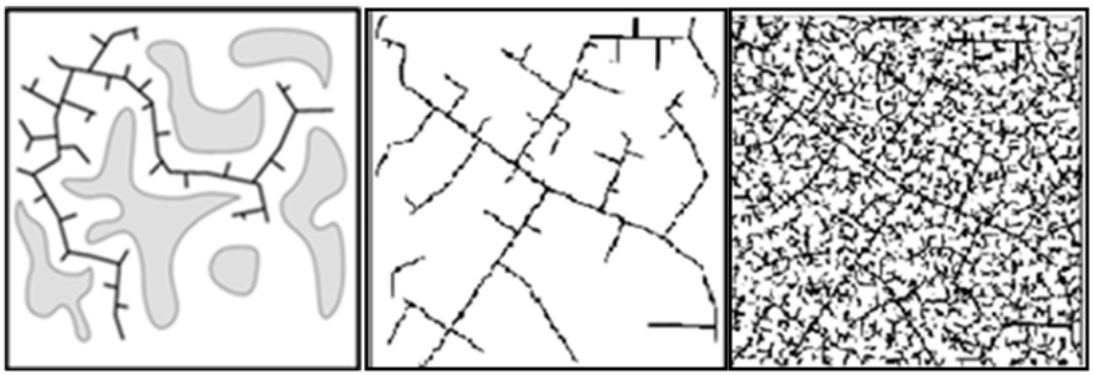
Phương pháp này thường trải qua các bước: Khi sinh ra một đỉnh mới v, kiểm tra xem v có ở bên trong vật cản hay không, nếu đó là một đỉnh có thể đến được thì một cạnh giữa nó và những điểm lân cận trực tiếp của nó được đưa vào đồ thị mới (hay còn gọi là tập các cạnh có thể di chuyển được), cuối cùng một đồ thị các đường đi được được tạo thành giữa điểm bắt đầu và điểm đích và việc tìm đường sẽ được thực hiện trên đồ thị các đường có thể đi được này theo một giải thuật nào đó, như trình bày trong hình 1(b) [R. Bohlin and L. Kavraki, 2000; S. LaValle, 1998].

Khuyết điểm của phương pháp này cũng gần giống như dựa trên tìm kiếm, việc lấy ngẫu nhiên một đỉnh thêm vào và kiểm tra cho sự kết nối này có thỏa mãn không sẽ ảnh hưởng lớn đến thời gian chạy của giải thuật. Lấy như thế nào cho tốt nhất và chạy bao nhiêu vòng lặp cũng là một vấn đề rất khó khăn, tương tự như dựa trên tìm kiếm là chia lưới như thế nào thì tốt nhất. Hình 2 thể hiện hình ảnh kết quả của phương pháp này sau số vòng lặp khác nhau.

2.1.3 Dựa trên tổ hợp

Phương pháp này được xem như giải pháp kinh điển đầu tiên, ra đời trước hai phương pháp dựa trên tìm kiếm và lấy mẫu. Trong khi dựa trên tìm kiếm và lấy mẫu dựa trên tính ngẫu nhiên cũng như xác suất để tìm ra đường dẫn mong muốn thì phương pháp này dựa trên lý thuyết của hình học để tìm một đường đi tốt nhất mong muốn một cách chính xác tuyệt đối. Phương pháp này lấy input vào là những đại diện nhiều mặt trong môi trường đang xét (như những vật cản đa diện) và thêm vào đó những cạnh hoặc đỉnh cần thiết. Dựa trên những yêu cầu bài toán, một đồ thị mới sẽ được tạo ra, gọi là một *roadmap*. Dựa trên *roadmap* này, đường dẫn tốt nhất như yêu cầu sẽ được tính toán. Và visibility-graph là một trong số những *roadmap* của phương pháp này. Định nghĩa cũng như độ phức tạp các giải thuật xây dựng một visibility-graph sẽ được trình bày trong mục 2.2.

Với đặc điểm này, giải pháp này luôn đưa ra một kết quả cụ thể cho bất kỳ bài toán tìm đường nào, hoặc là tìm ra một đường tốt nhất hoặc sẽ báo ra một cách chính xác rằng không một đường dẫn tốt nhất nào như yêu cầu tồn tại. Trái ngược lại với phương pháp này, hai phương pháp đầu, sau một khoảng thời gian chạy có thể không đưa ra bất kỳ giải pháp nào và người sử dụng sẽ không chắc chắn là liệu đang có một đường dẫn tốt nhất đang tồn tại hay không và thời gian chạy đã đủ chưa. Tuy nhiên, trong thực tế, các bài toán tối ưu toàn cục thường đòi hỏi một giải pháp giới hạn về mặt thời gian và phương pháp dựa trên tổ hợp này lại tiêu tốn khá nhiều thời gian trong trường hợp môi trường hoạt động chứa quá nhiều vật cản. Nhưng một hệ thống robot được xem là hoạt động tốt cần kết hợp cả hai chế độ, chế độ tìm đường thông thường với độ phức tạp thấp và nhanh - sử dụng tìm đường dựa trên tìm kiếm hoặc lấy mẫu và một chế độ tìm đường chính xác - sử dụng tìm đường dựa trên tổ hợp [S.K. Ghosh, 2007]. Ngoài ra phương pháp này còn hữu dụng trong nhiều lĩnh vực như CAD/CAM, xử lý ảnh, VLSI.



Hình 2: Giải thuật dựa trên lấy mẫu với số vòng lặp tăng dần [18]

2.2 Visibility-graph cho bài toán tìm đường đi ngắn nhất

Môi trường di chuyển của robot được xem như một đồ thị $G = (V, E_s)$ với:

– V là tập hợp những đỉnh tương ứng với vị trí các đỉnh của vật cản và điểm đầu, cuối mà robot cần di chuyển.

– E_s là tập con của $V \times V$, là tập hợp những cạnh trong đồ thị, tức đường bao quanh của vật cản.

Visibility-graph là một trong số những *roadmap* theo hướng tiếp cận dựa trên tối ưu tổ hợp, phù hợp nhất cho bài toán tìm đường đi ngắn nhất về mặt khoảng cách. Visibility-graph được xây dựng bằng cách thêm vào đồ thị G tập các cạnh E_v với:

$$E_v = \{(v_i, v_j) \in V \times V \mid visible(v_i, v_j, E_s)\}$$

Hàm $visible(v_i, v_j, E_s)$ trả về true nếu những cạnh trong E_s không cắt cạnh (v_i, v_j). Nói cách khác (v_i, v_j) tạo thành một cạnh nếu có thể đi được từ v_i tới v_j mà không bị cản trở bởi bất kì vật cản nào. Một visibility-graph được tạo thành:

$$G_{visibility-graph} = (V, E, W_E)$$

với: $E = E_s \bigcup E_v$

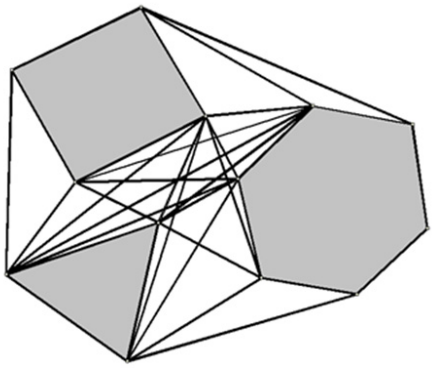
$W_E : E \to R^+$ là trọng số chi phí tương ứng với mỗi cạnh.

Với một visibility-graph như trên, một đường dẫn chi phí nhỏ nhất theo tiêu chí nào đó sẽ được tìm thấy giữa đỉnh bắt đầu và những đỉnh cần đến của robot.

Như vậy, việc tìm đường đi ngắn nhất trong không gian nhiều vật cản cần 2 bước chính khi giải bài toán theo dạng này: (1) Tạo visibility-graph và (2) dựa vào trong số chi phí tương ứng mỗi cạnh, một giải thuật tìm đường đi ngắn nhất có thể áp dụng như Dijkstra, Bellman-Ford, A* search, Floyd-Warshall. Với bước (2), bài toán trở thành bài toán tìm đường ngắn nhất kinh điển trên đồ thị liên thông có trọng số, các giải thuật cũng như độ phức tạp của chúng đã trở nên rất quen thuộc. Vì vậy, độ phức tạp của toàn bộ quá trình phụ thuộc vào bước (1), như thế nào để tạo ra một visibility-graph ít tốn thời gian nhất.

Giải thuật đơn giản nhất là lần lượt kiểm tra từng đoạn thẳng có thể trong đồ thị xem đoạn thẳng này có bị cắt bởi bất kì vật cản nào không. Giải thuật này chạy trong thời gian $\theta(n^3)$. Lee là người đầu tiên cải tiến từ giải thuật đơn giản để đạt được độ phức tạp tốt hơn $\theta(n^2 logn)$; Lee sử dụng phương pháp quét xoay tròn và một danh sách lưu lại thứ tự quét lần trước để làm giảm độ phức tạp của giải thuật [S. M. Valle, 2006]. Sau Lee, một nhóm các giải thuật đạt độ phức tạp $\theta(n^2)$ được giới thiệu. Tiêu biểu là một số giải thuật của Welzl và các nhóm tương tự [S.K. Ghosh, 2007; E. Welzl, 1985; John E. Hershberger and Subhash Suri, 1993]. Sau khi nhóm giải thuật Output-sensitive được công bố, tiêu biểu là giải thuật của Ghosh and Mount thời gian tính toán visibility-graph xuống $\theta(nlogn)$. Tuy nhiên, phương pháp này sử dụng lý thuyết tam giác hóa và cấu trúc dữ liệu đặc biệt phức tạp. Về mặt lý thuyết phương pháp này có thể đạt được độ phức tạp tốt nếu các cạnh được tạo ra trong quá trình cấu tạo visibility-graph là thưa thớt. Nhưng về mặt hiện thực, giải thuật này được khuyến cáo là không đạt được độ phức tạp trên vì cấu trúc dữ liệu quá phức tạp [S.K. Ghosh, 2007].

Một số điều cần lưu ý thêm, nếu dùng visibility-graph, đường đi tìm được là dọc theo biên của vật cản. Nếu robot có kích thước, khoảng cách hành lang phải được tính toán để robot có thể di chuyển tốt. Vấn đề chặt góc khi cua cũng cần xem xét đến. Việc mở rộng visibility-graph lên nhiều hơn hai chiều là điều không thể. Point-to-point planning trong không gian cấu hình ba chiều với những vật cản đa diện là bài toán NP-hard và một số phương pháp giải quyết đã được trình bày trong [S.K. Ghosh, 2007].



Hình 3: Một ví dụ với ba vật cản đa giác và visibility-graph

Như đã trình bày ở trên, bài báo sử dụng giải thuật của Welzl [S.K. Ghosh, 2007; E. Welzl, 1985] để hiện thực việc xây dựng visibility-graph (chi tiết của hiện thực giải thuật không trình bày ở đây). Tuy nhiên, một khuyết điểm của phương pháp này là với môi trường chứa nhiều vật cản, thời gian tính toán sẽ rất lớn. Vì vậy, một số kỹ thuật có thể được sử dụng thêm vào cùng nhằm giảm bớt thời gian tính toán cho visibility-graph. Phần tiếp theo sẽ giới thiệu hai kỹ thuật đơn giản nhưng giúp giảm thời gian tính toán này một cách hiệu quả.

3 CÁC PHƯƠNG PHÁP GIÚP GIẢM THỜI GIAN XÂY DỰNG MỘT VISIBILITY-GRAPH

3.1 Gom nhóm, tiền xử lý vật cản

Ý tưởng chính của phương pháp này xuất phát từ một số môi trường thực tế mà trong đó một số vật cản là rất nhỏ so với các vật cản khác. Các nhóm vật cản nhỏ này thường đứng lận cận nhau, vì vậy nếu xem xét vai trò các vật cản rất nhỏ ngang bằng với các vật cản rất lớn, thì thời gian tiêu tốn là lãng phí. Kỹ thuật này trình bày cách gom nhóm các vật cản nhỏ lại với nhau, sau đó hợp lại để tạo thành một vật cản lớn hơn.

3.1.1 Tổng quát bài toán

Định nghĩa 1. Vật cản được gọi là nhỏ nếu diện tích của nó nhỏ hơn $1/\alpha$ tổng diện tích của môi trường mà vật cản di chuyển trong đó

Định nghĩa 2. Hai vật cản nhỏ được xem là gần nhau nếu khoảng cách giữa hai tâm của chúng nhỏ hơn ε. Tập các vật cản nhỏ gần nhau được tạo thành một cụm.

Định nghĩa 3. Bao đóng lồi của một tập hữu hạn các điểm *S* trong mặt phẳng là đa giác lồi *P* nhỏ nhất mà bao xung quanh *S*, nhỏ nhất theo nghĩa là không có một đa giác *P'* nào khác sao cho $P \supset P' \supseteq S$ [J. O'Rourke, 1998]

Khoảng cách giữa hai đa giác được định nghĩa như sau:

$$d(P_i, P_j) \cong d(c(P_i), c(P_j)) = \left|c(P_i), c(P_j)\right|$$

Với $c(P_i)$, $c(P_j)$ theo thứ tự là tâm của đa giác P_i và P_j

Tâm của một đa giác có thể được xem như tâm của tập hợp các đỉnh trong đa giác, hoặc cải tiến hơn theo công thức Surveyor

- Tâm của một tập hữu hạn của *n* điểm x_1, x_2, ... x_n trong R^2: $C = \frac{1}{n}\sum_{k=1}^{n} x_k$
- Công thức Surveyor:

$$C_x = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$A = \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)$$

Với các đỉnh phải xếp theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ; nếu ngược chiều kim đồng hồ, giá trị tính được theo công thức sẽ âm, và giá trị đúng lấy trị tuyệt đối. (Lưu ý, đỉnh đầu tiên và cuối cùng là một $(x_n, y_n) = (x_0, y_0)$)

3.1.2 Phương pháp thực hiện

Việc gom nhóm được thực hiện khi:

$(S(p_i) < 1/\alpha)$ và $(\exists x \in c : d(p_i, x) \le \varepsilon)$ thì

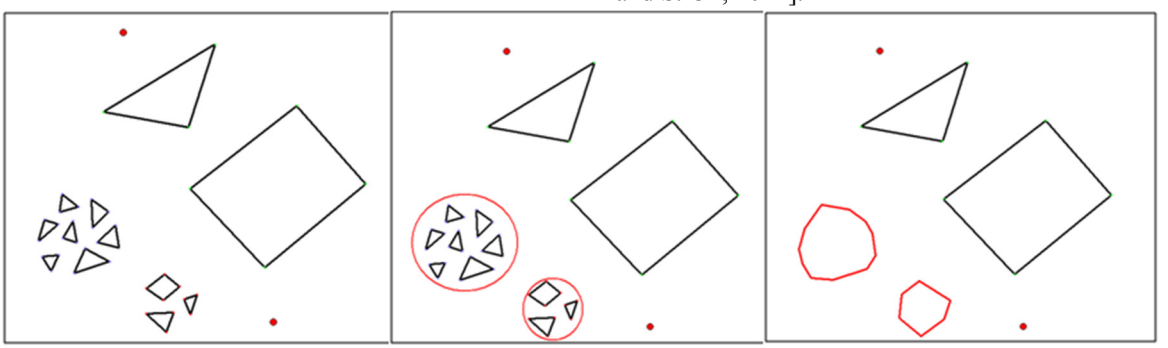
$c = c \cup p_i$ và $center_c = \frac{1}{n}\sum_{i=1}^{n} c(p_i)$

với c là một cụm, p_i là một đa giác, $S(p_i)$ là diện tích p_i và $c(p_i)$ trong R^2

Mỗi đa giác trước khi đưa vào danh sách, nếu là đa giác nhỏ, sẽ được kiểm tra với các đa giác trước đó để xác định đa giác này có thuộc về cụm đa giác nào không. Đa giác nhỏ được xếp vào một cụm đa giác chỉ cần được xem là gần với một phần tử bất kì trong cụm đó. Mỗi cụm đa giác được xác định bằng một ID. Và tâm của nhóm sẽ được tính toán lại mỗi khi các đa giác trong nhóm thay đổi.

Sau khi các đa giác nhỏ được đưa vào từng nhóm, các đa giác trong cùng một nhóm được kết hợp lại với nhau tạo thành một đa giác lớn bao xung quanh. Giải thuật kết hợp các đa giác lấy ý tưởng từ giải thuật tìm bao đóng lồi (convex hull) hoặc bao đóng lõm (concave hull) của tập điểm. Các đỉnh của các polygon trong một cụm được đưa vào tập điểm cần tìm bao đóng. Kết quả tìm được là một đa giác bao xung quanh, thay thế cho cụm các đa giác đó như được trình bày trong Hình 4. Nếu quá trình kết hợp các vật cản chỉ dựa hoàn toàn vào giải thuật tìm convex hull, trong một vài trường hợp, quá trình kết hợp này sẽ không hiệu quả, lãng phí một số không gian. Để khắc phục vấn đề này, giải thuật concave hull có thể được sử dụng thay thế.

Để cấu trúc một bao đóng lồi cho tập điểm trong 2D có rất nhiều giải thuật, độ phức tạp từ $\theta(n^3)$ tới $\theta(nlogn)$. Các giải thuật tiêu biểu như Gift Warpping $\theta(n^2)$, QuickHull $\theta(n^2)$, O'Graham $\theta(nlogn)$, Incremental Algorithm $\theta(nlogn)$ [J. O'Rourke, 1998]. Và một vài giải thuật tìm concave hull đã được giới thiệu như SwingArm $\theta(n^3)$, KNN-based $\theta(n^3)$, Shape $\theta(nlogn)$ [J.Park and S. Oh, 2012].



Hình 4: Các bước gom nhóm vật cản

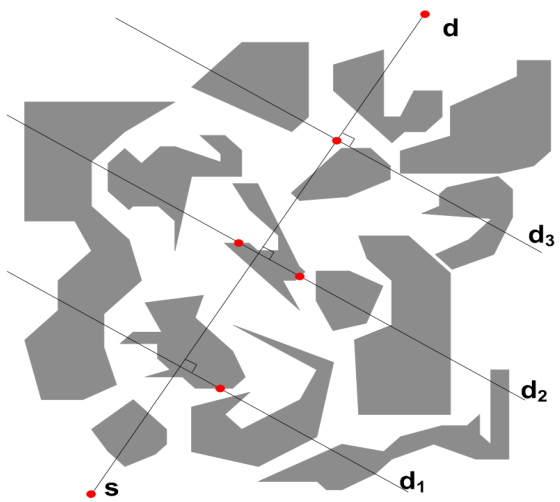
3.2 Chia vùng môi trường hoạt động kết hợp với xử lý song song

Ý tưởng chính của kỹ thuật này là vùng hoạt động được chia thành nhiều vùng nhỏ và visiblity-graph cho mỗi phần được tạo thành một cách song song.

Định nghĩa 1. Giả sử vùng hoạt động được chia thành $n + 1$ phần, gọi d_i là đường vuông góc với đường sd (với s là điểm đầu, d là điểm đích của đường đi ngắn nhất cần tìm). Tập hợp của tất cả các đường d_i, với i từ 1 tới n được gọi là tập hợp của các đường chia. Khoảng cách từ s tới d_i là kết quả phép nhân k và i, với k là kết quả của sd chia cho n.

Định nghĩa 2. Một phần chia được gọi là S_i nếu nó được tạo thành từ đường chia d_{i-1} và d_i

Đường dẫn ngắn nhất từ điểm nguồn đến điểm đích có thể được tính toán bằng cách: hoặc tìm ra các đoạn đường ngắn nhất cục bộ trong từng phần; hoặc tích hợp các visibility-graph trong từng phần tạo một visibility-graph hoàn chỉnh và đường dẫn ngắn nhất được tính toán dựa trên graph mới này. Hình 6 và 7 trình bày code giả của hai cách tính như trình bày ở trên.



Hình 5: Vùng hoạt động được chia thành nhiều phần

Với dạng đầu tiên, để tìm ra đường dẫn ngắn nhất trong mỗi phần chia, điểm bắt đầu và điểm đích cục bộ của từng phần phải được chỉ rõ. Do điểm đích toàn cục là cố định nên việc chọn lựa các điểm đầu và điểm đích cục bộ cố gắng hướng theo điểm đích toàn cục này. Kỹ thuật được sử dụng trong phần này định nghĩa các điểm bắt đầu và điểm đích cục bộ như giao điểm của *sd* và các đường chia. Tuy nhiên, trong một vài trường hợp đặc biệt như Hình 8 những giao điểm này rơi vào trong các đa giác và một số điểm thay thế cần được sử dụng sao cho đảm bảo tối ưu nhiều nhất có thể. Những điểm thay thế có thể là một trong hai giao điểm của đa giác và đường chia (Hình 8(a), 8(b)). Trường hợp giao điểm của đa giác và *sd* không rơi vào bên trong của đa giác mà trên một cạnh lõm của một đa giác lõm (Hình 8(c)) thì các giao điểm này không được chọn vì các cạnh lõm như thế đã được chứng minh sẽ không rơi vào tập cạnh của đường dẫn ngắn nhất [X. Shen and H. Edelsbrunner, 1987]. Theo đó, A hoặc B trong trường hợp này sẽ thích hợp hơn để thay thế. Tương tự, trong Hình 8(d) chỉ một giao điểm bên phải nên được sử dụng. Hình 7 trình bày code giả của cách tiếp cận này và hàm replace_point được sử dụng để kiểm tra và thay thế các giao điểm trong các trường hợp đặc biệt vừa nêu trên. Và trong hàm shortest_path, visibility-graph cho mỗi phần chia sẽ được tạo và giải thuật Dijkstra được áp dụng để tính đường ngắn nhất trong mỗi phần. Như trình bày ở trên, một vài giải thuật khác như Bell-man-Ford, A* search, Floyd-Warshall vẫn có thể được sử dụng thay vì Dijkstra.

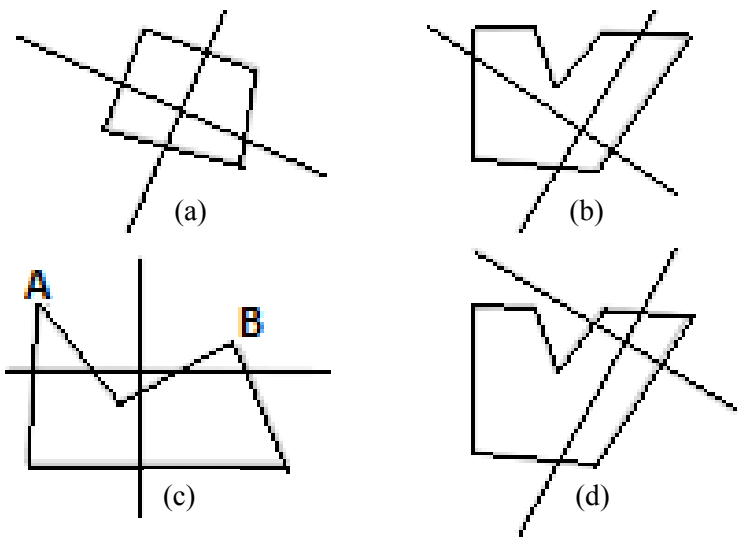
```
//D_local and S_local is a point
for_parallel (int i = 0; i <= n; i++)
  //find S_local
  if (i == 0)
    then S_local = s
    else
      point_k = intersect(sd, d_i)
      if ∃p ∈ P : (inside(point_k, p)) or
            (!inside(point_k, p) & concave_hull(p) & inConcavePart(point_k, p))
        then S_local = replace_point(point_k, p)
        else S_local = point_k
      end
  end
  //find D_local
  if (i == n)
    then D_local = d
    else
      point_k = intersect(sd, d_i+1)
      if ∃p ∈ P : (inside(point_k, p)) or
            (!inside(point_k, p) & concave_hull(p) & inConcavePart(point_k, p))
        then D_local = replace_point(point_k, p)
        else D_local = point_k
      end
  end
  shortest_path_local = shortest_path(S_local, D_local)
end
```

Hình 6: Code giả - Đường dẫn ngắn nhất tìm được dựa trên kết hợp các đường ngắn nhất trong từng phần

Với dạng thứ hai, các visibility-graph cục bộ trong từng phần được tích hợp lại tạo thành một visibility-graph hoàn chỉnh và đường dẫn ngắn nhất được tính toán dựa trên visibility-graph cuối cùng này. Code giả lập cho cách tiếp cận này được trình bày trong Hình 7. Lưu ý, hàm shortest_path trong code giả lập ở Hình 6 và Hình 7 chứa các tham số khác nhau. Với hai tham số như trong Hình 6, hàm shortest_path sẽ phải cấu trúc một visibility-graph trước khi sử dụng Dijkstra, trong khi với ba tham số như trong Hình 7, hàm này chỉ cần sử dụng Dijkstra ngay lập tức.

```
- Initial visibility_graph_global
- for_parallel (int i = 0; i <= n; i++)
    set_points_local = get_points_in_division_i(P, di)
        visibility_graph_i = build_visibility_graph(set_points_local)
        visibility_graph_global.append(visibility_graph_i)
    end
- shortest_path(s,d, visibility_graph_global)
```

Hình 7: Code giả - Đường dẫn ngắn nhất tìm được dựa trên kết hợp các visibility-graph trong từng phần, tạo ra visibility-graph toàn cục



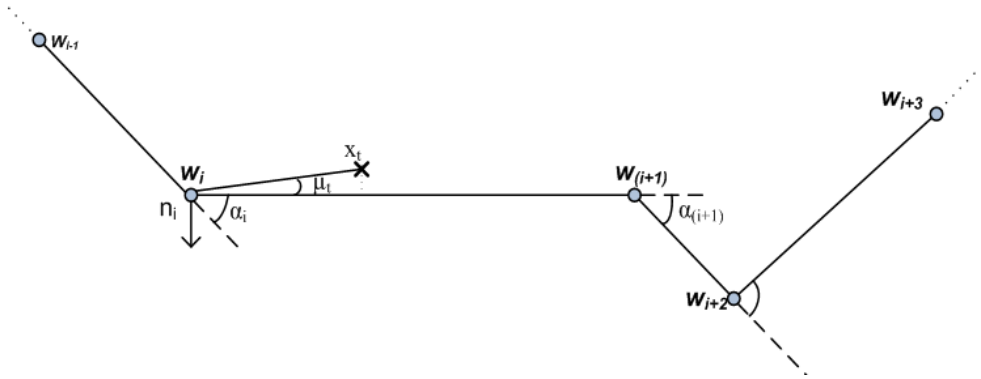
Hình 8: Giao giữa đường chia và sd trong một vài trường hợp

Việc chia môi trường hoạt động thành nhiều phần và thực hiện song song hóa cần phân thành hai dạng như trên do ưu cũng như khuyết điểm của từng dạng. Dạng thứ nhất sẽ thích hợp trong môi trường động hơn tĩnh. Trái ngược lại, dạng thứ hai phù hợp với các hoạt động tĩnh và không đòi hỏi khắt khe về giới hạn thời gian. Các robot có thời gian đủ để chờ một visibility-graph hoàn chỉnh được tạo thành và sau đó mới tìm một đường dẫn ngắn nhất có thể. Trong khi đó, với dạng đầu tiên, robot có thể di chuyển theo đường đi ngắn nhất cục bộ trong phần chia thứ n đồng thời tính toán lại đường đi ngắn nhất của phần chia $n+k$ nào đó nếu không may xảy ra sự thay đổi trong phần chia $n+k$ này.

4 GIẢI PHÁP BÁM ĐƯỜNG CHÍNH XÁC CHO ROBOT

Như đã trình bài trong phần giới thiệu, sau khi tìm ra một đường dẫn tốt nhất để di chuyển (trong trường hợp này là ngắn nhất về khoảng cách), một danh sách các đỉnh nối tiếp nhau cần di chuyển để đạt đích đến sẽ phải được chuyển tới robot. Các đỉnh này gọi là các waypoint. Việc di chuyển từ điểm A đến một điểm B nào đó của robot trong

thực tế chính xác như thế nào bị phụ thuộc bởi rất nhiều yếu tố khác nhau như cấu tạo robot, ma sát, gió,... Do đó, robot có thể bị lệch đường trong khi di chuyển và chỉ đến một điểm nào đó gần B. Việc robot di chuyển trong một tập các điểm nối tiếp nhau càng dẫn đến độ lệch lớn do tích lũy sai nhiều dần lên trong từng đoạn nhỏ. Phần này trình bày phương pháp giúp robot bám theo một đường đi mong muốn một cách chính xác, gọi là point-to-point-tracking.



Hình 9: Đường di chuyển mong muốn của robot với các waypoint theo thứ tự w_{i-1}, w_i , w_{i+1}, w_{i+2}, w_{i+3}

Dạng điều khiển	K_p	K_i	K_d
P	$K_u/2$	-	-
PI	$K_u/2.2$	$1.2K_p/T_u$	-
PID	$0.60K_u$	$2K_p/T_u$	$K_pT_u/8$

Hình 10: Các thông số PID tính toán theo phương pháp Ziegler–Nichols

Định nghĩa 1: $P \in N \times R^2$ là một đường dẫn được định nghĩa bởi một chuỗi N waypoint mong muốn và mỗi đường P_i kết nối một waypoint w_i tới w_{i+1} được gọi là một đường con trong chuỗi các đường được tạo thành từ đường đi toàn cục mong muốn.

Như trong Hình 9, n_i là vector đơn vị định hướng di chuyển mong muốn, x_t định vị trí hiện tại của robot, vector lỗi của robot được tính:

$$e_t = x_t \times \tan(\mu_t) n_i$$

Bộ điều khiển bám đường chính xác của robot sử dụng một vòng lặp đóng lấy ý tưởng từ bộ điều khiển P trong điều khiển dùng PID: $u_t = K_p e_t$. Vị trí hiện tại x_t của robot có thể được xác định bởi một camera nếu trong phòng thí nghiệm, hoặc một bộ định vị nội bộ nhỏ nếu trong tòa nhà, hoặc có thể dùng GPS nếu trong môi trường ngoài trời. Và K_P được xác định theo phương pháp của Ziegler–Nichols, một trong những giải pháp kinh điển khi sử dụng điều khiển PID [J. B. Ziegler and N. B. Nichols, 1942]. Đầu tiên, K_i và K_d được gán giá trị 0 để bộ điều khiển chỉ hoạt động với tham số P. K_p được tinh chỉnh cho đến khi đạt được một giá trị tạm chấp nhận, gọi là K_u. Với K_u, output của vòng lặp điều khiển robot rung lắc với một biên độ gần cố định nào đó, biên độ này được gọi là T_u. Giá trị T_u được sử dụng để gán các giá trị cho K_p, K_i, K_d tùy vào bộ điều khiển sử dụng là loại P, PI hay PID theo công thức như trong Hình 10. Vào cuối mỗi đoạn di chuyển từ i tới $i + i$, trước khi robot xoay để thực hiện việc di chuyển tiếp theo, vận tốc di chuyển cần được giảm xấp xỉ về 0; tức robot cần dừng lại tại các waypoint trung gian để xoay một góc chính xác theo hướng di chuyển tiếp theo. Như trong Hình 9, robot sẽ xoay một góc α về bên trái tại waypoint w_i và một góc α_{i+1} về bên phải tại waypoint w_{i+1}.

5 KẾT QUẢ THỰC NGHIỆM

5.1 Kết quả thực nghiệm khi áp dụng gom nhóm, tiền xử lý vật cản và chia vùng môi trường hoạt động kết hợp với xử lý song song

Để kiểm tra độ hiệu quả khi áp dụng phương pháp tiền xử lý gom nhóm vật cản và chia vùng môi trường hoạt động kết hợp xử lý song song, chúng tôi đã hiện thực một chương trình tính đường đi ngắn nhất từ điểm đầu đến điểm cuối, tránh vật cản tĩnh trong môi trường 2D, sử dụng visibility-graph, dùng Visual C++ và thư viện Leda

hỗ trợ. Chương trình có thể lựa chọn: dùng visibility-graph thông thường để tính đường ngắn nhất, hoặc kết hợp với tiền xử lý gom nhóm vật cản, hoặc chia vùng môi trường hoạt động song song hóa. Tập testcase từ 5 đến 50 vật cản được tạo ra với số đỉnh trung bình từ 30 đến 300 đỉnh. Tập testcase chứa các vật cản lớn nhỏ với nhiều kích thước khác nhau. Các vật cản được mô phỏng trong môi trường 2D trên hệ trục tọa độ *Oxy*. Tùy vào kích thước môi trường hoạt động thật, một đơn vị trên tọa độ sẽ tương ứng với một đơn vị đo lường khoảng cách trong thực tế (*mm, cm, m, km ...*).

Như lý thuyết trình bày trong phần 3.1, hai tham số α (tham số xác định vật cản như thế nào là nhỏ) và ε (tham số xác định hai vật cản gần nhau) có thể nhận các giá trị khác nhau, và các giá trị khác nhau này sẽ ảnh hưởng đến thời gian chạy của giải thuật khi sử dụng phương pháp tiền xử lý gom nhóm vật cản nhỏ. Chúng tôi thực hiện việc đo đạc bằng cách cho tập testcase chạy với giải thuật tính đường đi ngắn nhất dùng visibility-graph thông thường trước; sau đó, gom cụm các vật cản nhỏ được áp dụng (theo kiểu tạo bao đóng lồi) và việc tìm đường đi ngắn nhất được thực hiện lại để tính độ chênh lệch về thời gian, với $\alpha = 100$ và ε lần lượt nhận các giá trị *20, 30, 40, 50* (Bảng 1). Với $\varepsilon = 20$, tức khi tâm hai vật cản cách nhau dưới 20 đơn vị thì hai vật cản này được xem là gần nhau và sẽ được gom vào một cụm. Thời gian tìm ra đường đi ngắn nhất khi áp dụng tiền xử lý, gom nhóm các vật cản so với áp dụng đơn thuần visibility-graph giảm xấp xỉ 45%. Tham số ε càng lớn, tức khoảng cách hai vật được xem là gần nhau càng nhỏ thì rõ ràng các cụm được gom lại ít hơn. Bảng 1 cho thấy, khi môi trường hoạt động chứa nhiều vật cản nhỏ, việc áp dụng gom nhóm là rất cần thiết, giúp giảm thời gian tính toán so với thông thường trung bình gần 30% (Giả sử đường đi ngắn nhất thật sự xuyên qua nơi các vật cản nhỏ phân bố thì đường đi ngắn nhất trong trường hợp gom cụm chắc chắn sẽ dài hơn đường đi ngắn nhất trong trường hợp không gom cụm. Tuy nhiên trong thực tế, nếu robot di chuyển trong khu vực phân bố của các vật cản nhỏ, việc điều khiển sẽ trở nên phức tạp và chậm vì phải rẽ quẹo liên tục; vì vậy, gom cụm các khu vực này lại, đường đi tốt nhất tìm được sau khi gom cụm có thể dài hơn so với việc không gom cụm, nhưng giúp robot không cần di chuyển vào các vùng phức tạp này sẽ tốt hơn).

Tương tự, như lý thuyết trình bày trong phần 3.2, tham số *d* (dùng để xác định bao nhiêu vùng được chia trong môi trường hoạt động) và cấu hình máy tính hỗ trợ song sẽ ảnh hưởng đến kết quả giải thuật. Chúng tôi thực hiện việc đo đạc bằng cách cho tập testcase chạy với giải thuật tính đường đi ngắn nhất dùng visibility-graph thông thường trước; sau đó chia vùng song song hóa với *d* lần lượt bằng *2, 3, 4, 5, 6* (theo code giả trong Hình 7) để tính độ chênh lệch về thời gian (Bảng 2). Tất cả các thí nghiệm được chạy trên CPU 2.53-GHz-Intel®Core™-i3-dual-core với 4GB RAM. Giá trị của *d* càng tăng, tức vùng hoạt động càng chia nhỏ để song song thì thời gian chạy giảm đáng kể (Thực tế, đường đi ngắn nhất trong trường hợp áp dụng chia vùng môi trường hoạt động, kết hợp xử lý song song sẽ dài hơn đường đi ngắn nhất trong trường hợp không áp dụng phương pháp này. Tuy nhiên, với các môi trường rộng, phức tạp việc áp dụng này giúp giảm thời gian một cách đáng kể trong khi đường đi tốt tìm được chênh lệch không nhiều so với đường đi ngắn nhất thật sự).

Bảng 1: Kết quả đo lường từ tập testcase cho phương pháp gom nhóm, tiền xử lý vật cản

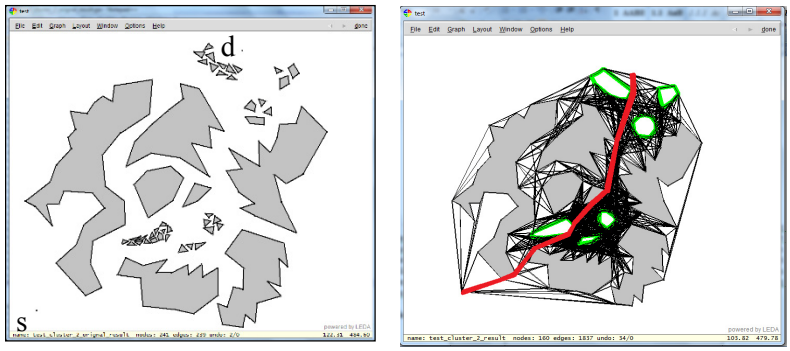
ε	Độ lệch thời gian (%)
20	45
30	37
40	32
50	06
Trung bình	30

Các giá trị của ε thể hiện ở cột thứ nhất; độ lệch thời gian chạy giải thuật khi áp dụng tiền xử lý gom nhóm vật cản giảm so với chỉ áp dụng visibility-graph đơn thuần thể hiện ở cột thứ 2; và α = 100 cho tất cả các trường hợp trong bảng

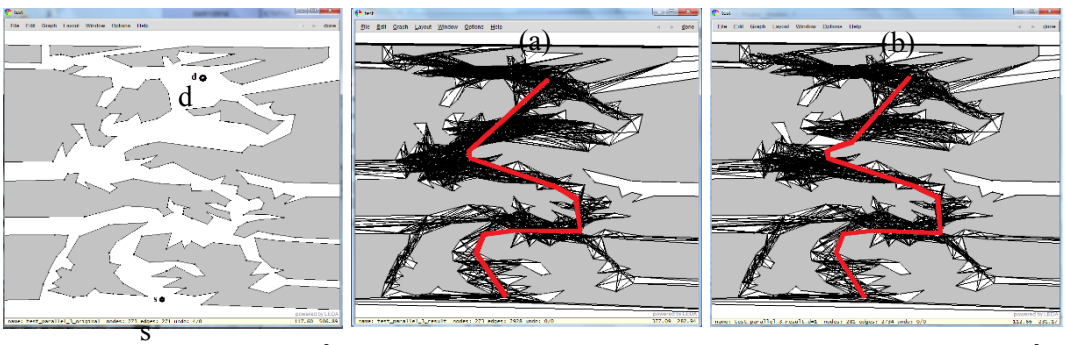
Bảng 2: Kết quả đo lường từ tập testcase cho phương pháp chia vùng môi trường hoạt động, kết hợp xử lý song song

d	Độ lệch thời gian (%)
2	53
3	45
4	59
5	60
6	61

Các giá trị của d thể hiện ở cột thứ nhất; độ lệch thời gian chạy giải thuật khi áp dụng chia vùng môi trường hoạt động, kết hợp xử lý song song giảm so với chỉ áp dụng visibility-graph đơn thuần thể hiện ở cột thứ 2



Hình 11: Hình ảnh một trong số các testcase khi thực hiện tiền xử lý gom cụm (s là điểm đầu, d là đích)



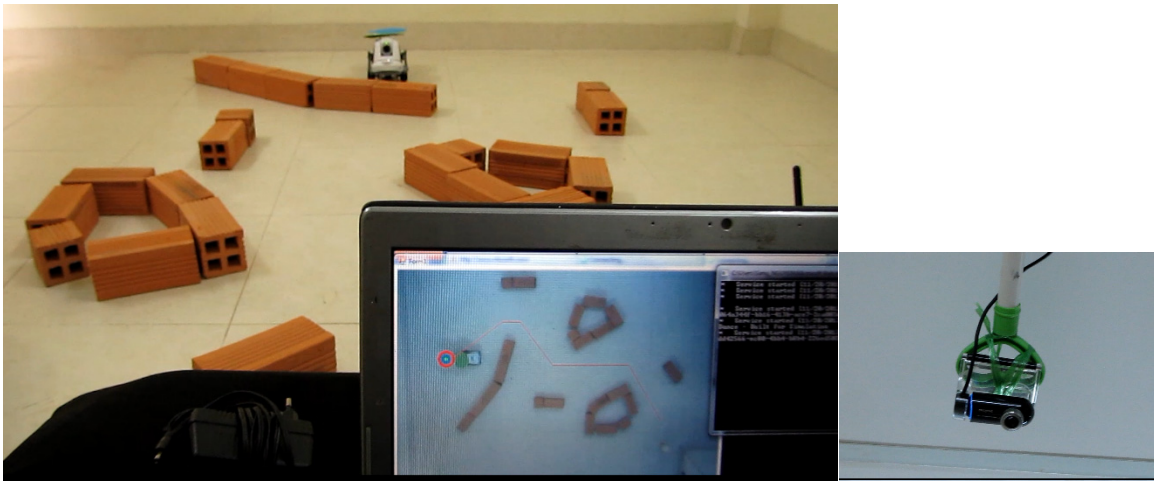
Hình 12: Hình ảnh một trong số các testcase khi thực hiện chia vùng môi trường hoạt động, kết hợp xử lý song song ((a) – đường đi ngắn nhất khi không chia nhỏ môi trường hoạt động, (b) – đường đi ngắn nhất với d = 2 và làm trơn lại)

5.2 Bài toán tìm đường đi ngắn nhất toàn cục và giải pháp bám đường chính xác được áp dụng lên robot thật

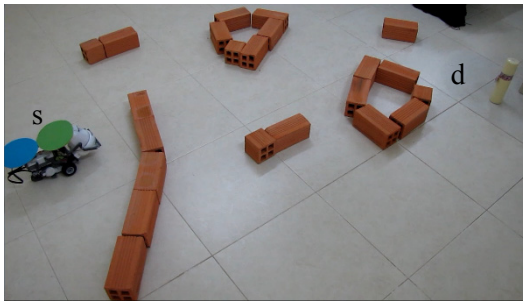
Để đảm bảo tính khả thi của lý thuyết được giới thiệu, một hệ thống robot thật được xây dựng và chạy thử nghiệm trong môi trường phòng thí nghiệm. Hệ thống bao gồm một trung tâm điều khiển (sử dụng máy tính với CPU Intel®Core™-i3-dual-core 2.53 GHz, 4.00-GB RAM), một robot (sử dụng Mindstorm-NXT của hãng Lego, một robot phục vụ cho mục đích giáo dục, có thể lập trình điều khiển với nhiều ngôn ngữ lập trình khác nhau như C, C++, Java, Python ...) và một camera lắp trên trần nhà để định vị robot (Hình 13a).

Quy tắc hoạt động như sau: robot được đặt trong sân với một số vật cản, nhiệm vụ của robot là di chuyển từ điểm "s" tới điểm "d" cho trước theo đường ngắn nhất và không va vào các vật cản tĩnh (Hình 13b). Đầu tiên, bản đồ chứa tập các vật cản cũng như điểm đầu và điểm cuối cần di chuyển của robot được đưa vào chương trình tính đường đi ngắn nhất trong máy tính của trung tâm điều khiển (đây cũng chính là chương trình được sử dụng để đo lường trong phần 5.1). Đường đi ngắn nhất tìm được là chuỗi các điểm cần di chuyển (waypoint); và chuỗi các waypoint này sẽ được truyền không dây tới robot để thực hiện việc di chuyển. Trong quá trình di chuyển, robot cần bám đường chính xác, vì vậy một chương trình bám đường chính xác cho robot như lý thuyết trình bày trong phần 4 được hiện thực. Chương trình bám đường này có hai khối chức năng chính, xác định vị trí hiện tại của robot (dùng camera và chương trình xử lý ảnh, nhận diện khối màu của robot để xác định vị trí) và điều khiển robot di chuyển chính xác theo đường đi hoạch định trước (Hình 13c). Như vậy, người điều khiển tại trung tâm có thể theo dõi tiến trình di chuyển của robot có chính xác không và can thiệp nếu cần.

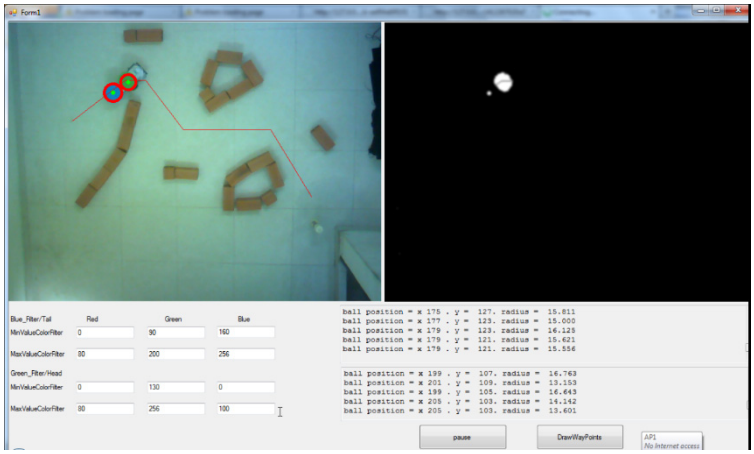
Hình 14 mô tả chương trình tìm đường đi ngắn nhất cho robot trước khi di chuyển (Vì robot có kích thước, nên các đường bao an toàn xung quanh vật cản cần được thiết lập trước khi tính đường đi ngắn nhất). Kết quả thực nghiệm cho thấy robot di chuyển thành công từ điểm đầu đến điểm đích cho trước, tránh các vật cản theo đường đi ngắn nhất toàn cục đã hoạch định trước (Hình 15). Nếu chỉ áp dụng bộ điều khiển P trong PID, sau khoảng trên dưới 100 lần đo đạt, kết quả thực nghiệm cho thấy trung bình khi di chuyển 100 cm, robot chỉ lệch biên xấp xỉ khoảng 5 cm và sau đó quay về đúng vị trí mong muốn.



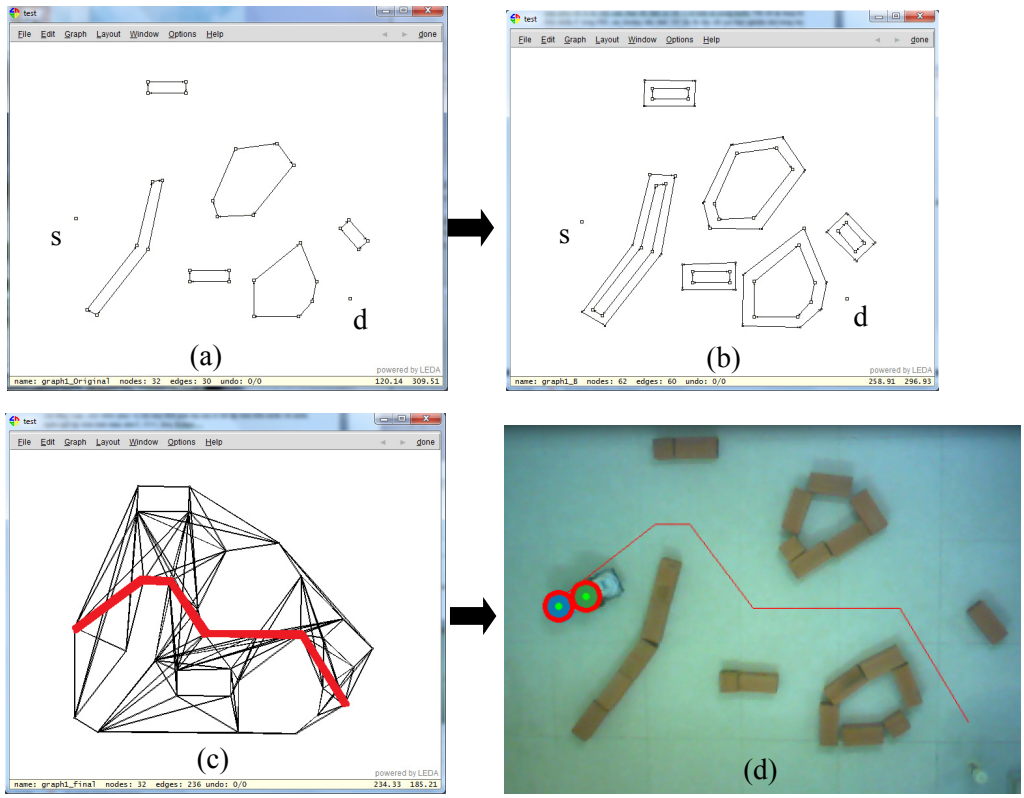
Hình 13a: Robot, vật cản, máy tính làm trung tâm điều khiển và camera gắn trên trần nhà dùng định vị robot



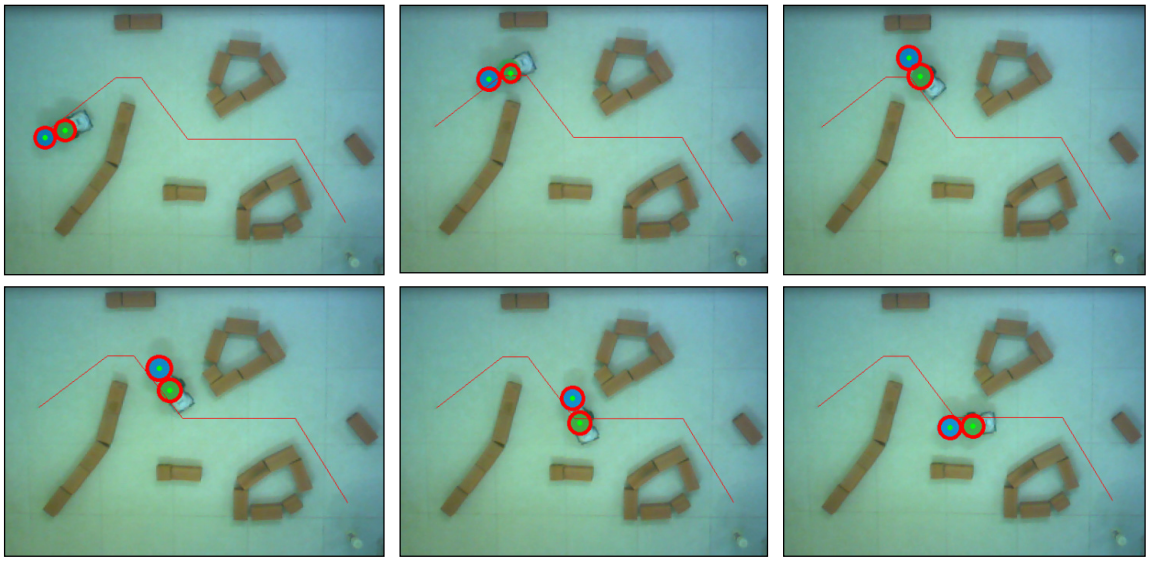
Hình 13b: Robot và môi trường hoạt động, với "s" là điểm đầu và "d" là đích đến

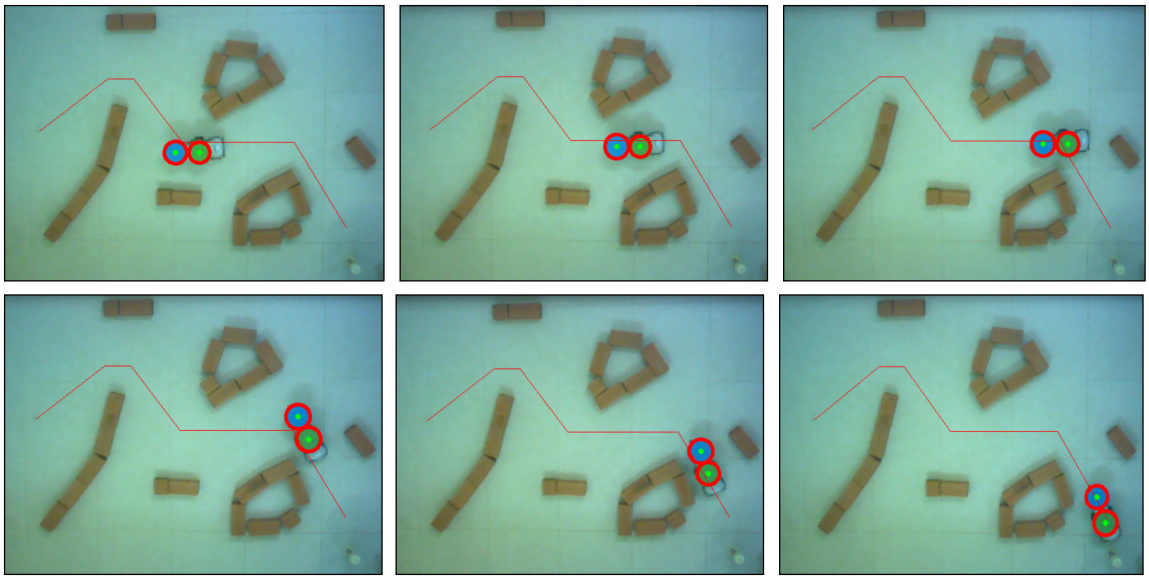


Hình 13c: Giao diện chương trình theo dõi và hỗ trợ robot bám đường chính xác



Hình 14: Chương trình tính đường đi ngắn nhất cho robot ((a) – bản đồ môi trường hoạt động; (b) – Các vật cản được tính thêm biên an toàn; (c) – đường đi ngắn nhất tìm được; (d) – đường đi ngắn nhất hiển thị ra màn hình theo dõi trên môi trường thật, và robot chuẩn bị di chuyển)





Hình 15: Một trong những thử nghiệm robot di chuyển bám theo đường đi ngắn nhất thành công

6 KẾT LUẬN

Bài báo đã trình bày một giải pháp tìm đường đi ngắn nhất từ một điểm đầu đến điểm đích, tránh vật cản tĩnh, trong môi trường 2D sử dụng visibility-graph cho robot. Tổng quan về các hướng tiếp cận của path-planning, ưu khuyết điểm của từng hướng và tại sao bài báo chọn sử dụng visibiliy-graph theo hướng combinatorial-planning cũng đã được trình bày. Bên cạnh những ưu điểm của visibility-graph, thì khuyết điểm lớn nhất của nó là tiêu tốn khá nhiều thời gian tính toán trong trường hợp số lượng vật cản trong môi trường hoạt động khá lớn. Vì vậy, bài báo cũng giới thiệu hai cách, gom nhóm các vật cản nhỏ và chia vùng môi trường hoạt động để xử lý song song, nhằm giảm thời gian tính toán trong thực tế. Kết quả thực nghiệm cho thấy hai kỹ thuật này giúp giảm thời gian đáng kể cũng như việc áp dụng trên robot thật hoàn toàn khả thi.

LỜI CẢM TẠ

Nghiên cứu này được tài trợ bởi Đại học Quốc gia Thành phố Hồ Chí Minh (VNU-HCM) trong đề tài mã số B2012-26-06.

TÀI LIỆU THAM KHẢO

1. S. M. Valle, 2006. Planning Algorithms. Cambridge University Press. 811 pp.
2. S.K. Ghosh, 2007. Visibility algorithms in the plane. Cambridge University Press. 334 pp.
3. E. Welzl, 1985. Constructing the Visibility Graph for n Line Segments in $\theta(n^2)$ Time. *In:* Information Processing Letters, vol. 20, issue 4: 167-171.
4. D. Ferguson and A.Stentz, 2006. Multi-resolution field D*. *In:* Proceedings of the International Conference on Intelligent Autonomous Systems (IAS).
5. R. Bohlin and L. Kavraki, 2000. Path planning using lazy PRM. *In:* IEEE Int. Conf. on Robotics and Automation, vol. 1: 521-528.
6. S. LaValle, 1998. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11. Computer Science Dept., Iowa State University.
7. D. Wooden and M. Egerstedt, 2006. Oriented Visibility Graphs: Low-Complexity Planning in Real-Time Environments. *In:* IEEE Conference on Robotics and Automation, Orlando, FL: 2354-2359.
8. J. Canny, 1987. The Complexity of Robot Motion Planning Cambridge. MA: MIT Press. 196 pp.
9. T. Lozano-Perez, 1987. A simple motion-planning algorithm for general robot manipulators. IEEE Journal of Robotics and Automation, vol. RA-3, No. 3: 224-238.

10. John E. Hershberger and Subhash Suri, 1993. Efficient Computation of Euclidean Shortest Paths in the Plane, *In:* IEEE 34th Annual Foundations of Computer Science: 508-517.
11. J. O'Rourke, 1998. Computational geometry in C. Cambridge University Press. 361 pp.
12. J.Park and S. Oh, 2012. A New Concave Hull Algorithm and Concaveness Measure for n-dimensional Datasets. Journal of Information science and engineering, No. 100295.
13. X. Shen and H. Edelsbrunner, 1987. A Tight Lower Bound on the Size of Visibility Graphs. *In:* Inf. Process. Lett: 61-64.
14. J. B. Ziegler and N. B. Nichols, 1942. Optimum settings for automatic controllers. *In:* ASME Transactions: 759-768.
15. J. Kitzinger and B.Moret, 2003. The Visibility Graph Among Polygonal Obstacles: a Comparison of Algorithms. Tech. Rep. TR-CS-2003-29. University of New Mexico.