**HỌC VIỆN KỸ THUẬT QUÂN SỰ**

**KHOA KỸ THUẬT ĐIỀU KHIỂN**

****

**BÁO CÁO TỔNG KẾT**

**ĐỀ TÀI NGHIÊN CỨU KHOA HỌC**

|  |  |
| --- | --- |
| Tên đề tài: | Nghiên cứu xây dựng hệ thống lập kế hoạch đường đi cho robot tự hành |

|  |  |
| --- | --- |
| **Sinh viên thực hiện:** | 1. Trịnh Quang Huy – ĐKCN16 (Chủ trì) 2. Nguyễn Tiến Đạt – ĐKCN16 3. Bùi Ngọc Anh – ĐKCN16 |

|  |  |
| --- | --- |
| **Giáo viên hướng dẫn:** | TS. Trương Xuân Tùng – K32 |

**Hà Nội - 2021**

**HÀ NỘI - 2018**

(Cỡ 14-15, đậm, căn giữa)

**UẬT ĐIỀU KHIỂN**

****

**BÁO CÁO TỔNG KẾT**

**ĐỀ TÀI NGHIÊN CỨU KHOA HỌC**

|  |  |
| --- | --- |
| Tên đề tài: | Nghiên cứu xây dựng hệ thống lập kế hoạch đường đi cho robot tự hành |

|  |  |
| --- | --- |
| **Sinh viên thực hiện:**  Contents  [MỤC LỤC 1](#_Toc67827092)  [PHẦN 1: TỔNG QUAN VỀ ROBOT TỰ HÀNH 5](#_Toc67827093)  [1. Mục tiêu 5](#_Toc67827094)  [2. Yêu cầu 5](#_Toc67827095)  [3. Nội dung 5](#_Toc67827096)  [3.1. Giới thiệu sợ lược về Robot 5](#_Toc67827097)  [3.2. Tổng quan về robot tự hành 6](#_Toc67827098)  [4. Nhận xét và Kết luận 9](#_Toc67827099) | 1. Trịnh Quang Huy – ĐKCN16 (Chủ trì) 2. Nguyễn Tiến Đạt – ĐKCN16 3. Bùi Ngọc Anh – ĐKCN16 |

|  |  |
| --- | --- |
| **Giáo viên hướng dẫn:** | TS. Trương Xuân Tùng – K32 |

**Hà Nội - 2021**

**HÀ NỘI - 2018**

(Cỡ 14-15, đậm, căn giữa)

# MỤC LỤC

[MỤC LỤC 1](#_Toc68180323)

[DANH MỤC HÌNH VẼ 3](#_Toc68180324)

[DANH MỤC TỪ VIẾT TẮT 4](#_Toc68180325)

[LỜI NÓI ĐẦU 5](#_Toc68180326)

[CHƯƠNG 1: TỔNG QUAN VỀ ROBOT TỰ HÀNH 6](#_Toc68180327)

[1. Nội dung 6](#_Toc68180328)

[1.1. Giới thiệu sợ lược về Robot 6](#_Toc68180329)

[1.2. Tổng quan về robot tự hành 7](#_Toc68180330)

[1.2.1. Phân loại robot tự hành 7](#_Toc68180331)

[1.2.2. Phương pháp điều hướng cho robot tự hành 9](#_Toc68180332)

[2. Kết luận 10](#_Toc68180333)

[CHƯƠNG 2: HỆ THỐNG DẪN ĐƯỜNG CHO ROBOT TỰ HÀNH 11](#_Toc68180334)

[1. Nội dung 11](#_Toc68180335)

[1.1. Sử dụng ray cứng dẫn đường 11](#_Toc68180336)

[1.2. Sử dụng vạch dán trên sàn nhà máy dẫn đường 11](#_Toc68180337)

[1.3. Sử dụng laser dẫn đường 12](#_Toc68180338)

[1.4. Sử dụng camera dẫn đường 12](#_Toc68180339)

[1.5. Sử dụng hệ định vị GPS 13](#_Toc68180340)

[2. Kết luận 13](#_Toc68180341)

[CHƯƠNG 3: CÁC PHƯƠNG PHÁP LẬP KẾ HOẠCH ĐƯỜNG ĐI CHO ROBOT TỰ HÀNH 14](#_Toc68180342)

[1. Nội dung 14](#_Toc68180343)

[1.1. Lý thuyết đồ thị 14](#_Toc68180344)

[1.2. Thuật toán Dijkstra 16](#_Toc68180345)

[1.2.1. Tổng quan 16](#_Toc68180346)

[1.2.2. Lập trình 16](#_Toc68180347)

[1.2.3. Kết luận 18](#_Toc68180348)

[1.3. Thuật toán A\* 18](#_Toc68180349)

[1.3.1. Tổng quan 18](#_Toc68180350)

[1.3.2. Lập trình 19](#_Toc68180351)

[1.3.3. Kết luận 21](#_Toc68180352)

[CHƯƠNG 4: XÂY DỰNG HỆ THỐNG LẬP KẾ HOẠCH ĐƯỜNG ĐI CHO ROBOT TỰ HÀNH 22](#_Toc68180353)

[1. Nội dung 22](#_Toc68180354)

[1.1. Tổng quan 22](#_Toc68180355)

[1.2. Thiết bị phần cứng 22](#_Toc68180356)

[1.3. Lập trình 24](#_Toc68180357)

[1.1.3. Kết luận 26](#_Toc68180358)

[KẾT LUẬN 27](#_Toc68180359)

[PHỤ LỤC 28](#_Toc68180360)

[I. Code kiểm nghiệm thuật toán Dijkstra viết bằng C 28](#_Toc68180361)

[II. Code kiểm nghiệm thuật toán A\* viết bằng C 30](#_Toc68180362)

[III. Code mô phỏng thuật toán tìm đường đi ngắn nhất cho RBTH trên bản đồ GM bằng ngôn ngữ Python 33](#_Toc68180363)

[TÀI LIỆU THAM KHẢO 41](#_Toc68180364)

# DANH MỤC HÌNH VẼ

Hình 1.1: Cánh tay robot sử dụng trong công nghiệp sản xuất ô tô

Hình 1.2: Robot hút bụi phổ biến trong dân dụng

Hình 1.3. Robot tự hành 8 chân

Hình 1.4. Robot vận chuyển tự hành AGV

Hình 2.1: Xe tự hành sử dụng ray cứng dẫn đường

Hình 2.1: Xe tự hành sử dụng vạch dán dẫn đường

Hình 3.1. Đồ thị có hướng và ma trận kề tương ứng

Hình 3.2. Đồ thị vô hướng có trọng số và ma trận trọng số tương ứng

Hình 4.1. Raspberry Pi 4 Model B (trái) và Module Sim7600G-H (phải)

Hình 4.2. Hệ thống khi kết nối các thiết bị với nhau

Hình 4.3. Ma trận trọng số

Hình 4.4. Kết quả đường đi ngắn nhất sau khi lập trình với GM API

# DANH MỤC TỪ VIẾT TẮT

RBTH: Robot tự hành

AGV: Xe tự hành

RPI: Raspberry Pi 4 Model B

GM: Google Maps

# LỜI NÓI ĐẦU

Cùng với sự phát triển của xã hội loài người, các ngành khoa học – kỹ thuật không ngừng đi đến những thành công mới. Nhiều công trình khoa học, những phát minh của các nhà khoa học đã đi vào cuộc sống, phục vụ lợi ích của con người. Ngày nay, robot đã được ứng dụng rộng rãi trong các ngành công nghiệp, thay thế cho nhiều hoạt động của con người. Một trong số đó là sử dụng robot tự hành (RBTH) trong vận chuyển hàng hóa.

Nhận thấy tính ứng dụng thực tiễn đó, nhóm sinh viên chúng em đã nhận và thực hiện đề tài nghiên cứu khoa học “Nghiên cứu xây dựng hệ thống lập kế hoạch đường đi cho robot tự hành”.

Nội dung nghiên cứu đề tài được chia làm bốn chương:

Chương 1: Nghiên cứu tổng quan về RBTH.

Chương 2: Nghiên cứu hệ thống dẫn đường cho RBTH.

Chương 3: Nghiên cứu các phương pháp lập kế hoạch đường đi cho RBTH

Chương 4: Xây dựng hệ thống lập kế hoạch đường đi cho RBTH

Trong thời gian thực hiện đề tài, nhóm đã nhận được sự giúp đỡ và chỉ bảo tận tình của thầy giáo hướng dẫn *Trương Xuân Tùng*. Tuy nhiên do thời gian có hạn và năng lực còn hạn chế, cùng với sự ảnh hưởng xấu của COVID-19 nên nhóm vẫn còn gặp nhiều khúc mắc dẫn đến chưa hoàn thành ở mức tốt nhất yêu cầu của đề tài. Chúng em rất mong nhận được sự chỉ bảo thêm của thầy cô để đề tài của nhóm được hoàn thiện hơn.

Chúng em xin trân thành cảm ơn.

Sinh viên

Trịnh Quang Huy

Nguyễn Tiến Đạt

Bùi Ngọc Anh

# CHƯƠNG 1: TỔNG QUAN VỀ ROBOT TỰ HÀNH

Chương này trình bày cái nhìn tổng quan về robot, robot tự hành, những khái niệm cơ bản như đặc điểm, phân loại, ứng dụng của robot tự hành.

## 1. Nội dung

### 1.1. Giới thiệu sợ lược về Robot

- Là một loại máy có thể thực hiện được những công việc một cách tự động bằng sự điều khiển của máy tính hoặc các vi mạch điện tử được lập trình.

- Robot thường có 3 phần chính:

* Bộ não (Brain): thường là một máy tính, vi xử lý hoặc vi điều khiển.
* Các bộ phận cơ khí và chấp hành (Actuator): động cơ, piston, cơ cấu kẹp, bánh răng, bánh xe...
* Cảm biến (Sensor): Thị giác, âm thanh, nhiệt độ, ánh sáng...

Với 3 thành phần này, Robot có thể tương tác với môi trường xung quanh.

- Robot được tạo ra chủ yếu để phục vụ mục đích của con người như: giải trí, hỗ trợ trong lao động sản xuất, giảm thiểu thao tác lao động trực tiếp của con người, thực hiện những công việc nặng nhọc, nguy hiểm thay thế con người, mục đích khám phá, thám hiểm...



Hình 1.1: Cánh tay robot sử dụng trong công nghiệp sản xuất ô tô

- Các lĩnh vực ứng dụng robot rất rộng và ngày càng được mở rộng thêm.

### 1.2. Tổng quan về robot tự hành

- Robot tự hành hay Robot tự hoạt động, Robot tự động, là robot thực hiện các nhiệm vụ với hành vi có mức tự chủ cao. Sự can thiệp của con người đến hoạt động của nó là ít hoặc rất ít.

- Tùy theo nhu cầu sử dụng mà người ta có thể chia ra nhiều loại robot tự hành như: tay máy, xe tự hành vận chuyển hàng hóa, xe tự hành trong y tế, robot hút bụi...



Hình 1.2: Robot hút bụi phổ biến trong dân dụng

#### 1.2.1. Phân loại robot tự hành

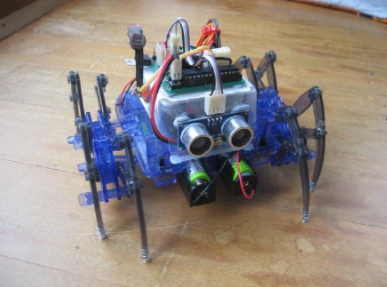
Có nhiều cách phân loại robot tự hành, có thể là theo cấu tạo, theo tính ứng dụng... Trong báo cáo này, em xin trình bày phân loại robot tự hành theo nguyên lý chuyển động.

Robot tự hành được chia làm hai loại chính đó là loại robot tự hành chuyển động bằng chân và loại robot tự hành chuyển động bằng bánh.

Ngoài ra, một số loại robot tự hành hoạt động trong các môi trường đặc biệt như dưới nước hay trên không thì chúng được trang bị cơ cấu di chuyển đặc trưng.

- *Robot tự hành chuyển động bằng chân:*

* Robot tự hành di chuyển bằng chân được mô phỏng theo các loài động vật mà vì thế chúng có loại 2 chân, 4, 6, 8 chân và có thể nhiều hơn nữa.
* Ưu điểm lớn nhất của loại robot này là có thể thích nghi và di chuyển trên các địa hình gồ ghề. Hơn nữa chúng có thể đi qua những vật cản như hố, vết nứt sâu.
* Nhược điểm chính của robot này chính là việc chế tạo quá phức tạp. Chân robot là một kết cấu nhiều bậc tự do, đây là nguyên nhân làm tăng trọng lượng của robot đồng thời làm giảm tốc độ di chuyển. Robot loại này càng linh hoạt thì chi phí chế tạo càng cao.



Hình 1.3. Robot tự hành 8 chân

- *Robot tự hành chuyển động bằng bánh:*

* Bánh xe là cơ cấu chuyển động được sử dụng rộng rãi nhất trong công nghệ Robot tự hành. Vấn đề cân bằng thường không phải là vấn đề được chú ý nhiều trong robot di chuyển bằng bánh. Ba bánh là kết cấu có khả năng duy trì cân bằng nhất, tuy nhiên kết cấu 2 bánh cũng có thể cân bằng được. Khi robot có số bánh nhiều hơn 3 thì thông thường người ta phải thiết kế hệ thống treo để duy trì sự tiếp xúc của tất cả các bánh xe với mặt đất.
* Vấn đề của robot loại này là về lực kéo, độ ổn định và khả năng điều khiển chuyển động.



Hình 1.4: Robot vận chuyển tự hành AGV

#### 1.2.2. Phương pháp điều hướng cho robot tự hành

Kỹ thuật điều hướng sử dụng trí thông minh nhân tạo trong robot tự hành có thể được chia thành 2 loại chính, đó là điều hướng có tính toán và điều hướng theo phản ứng. Đúng như tên gọi, điều hướng có tính toán là phương pháp điều hướng có kế hoạch còn điều hướng theo phản ứng là điều hướng tức thời, là quá trình tự động thực hiện các phản ứng theo môi trường xung quanh.

Ngoài ra còn có phương pháp điều hướng lai ghép là phương pháp kết hợp cả hai phương pháp có tính toán và điều hướng theo phản ứng để xây dựng một bộ điều khiển thông minh hơn.

- *Phương pháp điều hướng có tinh toán:*

* Là phương pháp thực hiện theo trình tự: quan sát – lập kế hoạch – hành động. Thông thường, một hế thống tính toán bao gồm 5 khâu: nhận thức, mô hình thế giới, lập kế hoạch, thực hiện công việc, điều khiển động cơ.
* Trong phương pháp này, khâu lập kế hoạch đường đi cho robot là cực kỳ quan trọng. Việc lập kế hoạch đường đi cho robot tự hành thường có hai giai đoạn đó là lập kế hoạch toàn cục và lập kế hoạch cục bộ. Lập kế hoạch toàn cục có thể được hiểu như là cách di chuyển robot qua một môi trường tùy ý và môi trường này là tương đối lớn. Còn lập kế hoạch cục bộ sẽ đưa ra quyết định khi robot đối mặt với môi trường tĩnh, ví dụ như khi robot gặp phải vật cản, hành lang...

- *Phương pháp điều hướng theo phản ứng:*

* Phương pháp này ra đời nhằm giải quyết các vấn đề có liên quan tới môi trường không biết trước hoặc môi tường thường xuyên thay đổi. Điều hướng theo phản ứng khắc phục được những hạn chế của phương pháp điều hướng tính toán, giúp giảm khối lượng tinh toán, tăng tốc độ xử lý trong môi trường phức tạp.
* Điều hướng theo phản ứng là phương pháp kết hợp các phản ứng thực hiện một cách tự động với các kích thích từ cảm biến để điều khiển robot sao cho an toàn và đạt hiệu suất cao nhất. Phương pháp này đặc biệt phù hợp đối với những ứng dụng mà môi trường là hoàn toàn động hoặc không biết trước, ví dụ như trong không gian hoặc dưới nước.

## 2. Kết luận

- Với sự phát triển nhanh chóng của khoa học – kỹ thuật, công nghệ chế tạo robot cũng phát triển theo. Robot ngày càng gia tăng cả về số lượng lẫn chất lượng nhằm đáp ứng những nhu cầu tự động hóa các quy trình sản xuất trong thời đại 4.0.

- Robot tự hành di chuyển bằng bánh là thông dụng hơn cả.

- Phương pháp điều hướng bằng cách tính toán cho robot tự hành là một phương pháp căn bản. Trong đó có khâu lập kế hoặc đường đi đóng vai trò quan trọng trong khả năng tự hành của robot.

# CHƯƠNG 2: HỆ THỐNG DẪN ĐƯỜNG CHO ROBOT TỰ HÀNH

Chúng ta đều biết để di chuyển từ điểm A đến điểm B, robot tự hành cần một hệ thống dẫn đường. Nói về hệ thống dẫn đường cho robot tự hành, trong phần này em xin đề cập cụ thể về cách xây dựng hệ thống dẫn đường cho xe tự hành AGV sử dụng trong nhà máy công nghiệp .

Xe tự hành AGV (Automated Guided Vehice) là một loại robot tự hành sử dụng trong công nghiệp, nhà xưởng. AGV công nghiệp được sử dụng chủ yếu để xử lý, vận chuyển vật liệu, hàng hóa trong nhà máy hoặc đôi khi là ngoài trời.

## 1. Nội dung

Một số cách để AGV có thể tự động di chuyển từ điểm A sang điểm B như sau:

- Sử dụng ray cứng dẫn đường.

- Sử dụng vạch dán trên sàn nhà máy dẫn đường.

- Sử dụng laser dẫn đường.

- Sử dụng camera dẫn đường.

- Sử dụng hệ định vị GPS.

### 1.1. Sử dụng ray cứng dẫn đường

|  |  |
| --- | --- |
| Mô hình này hiểu một cách đơn giản như các đoàn tàu hoạt động trên đường ray. Ray có thể được lắp nổi hoặc chìm xuống mặt sàn.  Ưu điểm: Đơn giản, ổn định, dễ dàng lắp đặt và vận hành.  Nhược điểm: Chỉ có thể áp dụng với các chuyển động đơn giản, lặp lại và đường di chuyển cố định. | Hình 2.1: Xe tự hành sử dụng ray cứng dẫn đường |

### 1.2. Sử dụng vạch dán trên sàn nhà máy dẫn đường

Với mô hình này chúng ta sẽ lắp đặt các đường dẫn trên sàn để xe AGV có thể theo đó di chuyển. Vạch dẫn ở đây có thể là vạch sơn, vạch băng keo dán, sợi cáp. Chúng có thể được lắp trên mặt sàn hoặc dưới mặt sàn.



Hình 2.2: Xe tự hành sử dụng vạch dán dẫn đường

Ưu điểm: Lắp đặt nhanh, dễ thực hiện thay đổi điều chỉnh dễ dàng, chi phí thấp, dễ thực hiện.

Nhược điểm:

- Vạch dán trên sàn gặp một số vấn đề làm mất sự liền lạc nếu các phương tiện khác có tải trọng lớn di chuyển cùng trên đường đó khiến xe tự hành không thể di chuyển được.

- Đường chạy là cố định.

### 1.3. Sử dụng laser dẫn đường

Với mô hình này xe AGV sử dụng thiết bị đặc biệt thu phát laser và đo khoảng cách đến các điểm cố định được định nghĩa trong nhà máy từ đó xác định được vị trí của xe AGV trong nhà máy và đường di chuyển.

Ưu điểm: Công nghệ này không can thiệp và sàn nhà máy, phù hợp với các khu vực có lưu lượng lưu thông lớn và tải trọng lớn.

Nhược điểm: Chi phí triển khai lớn, công nghệ phức tạp.

### 1.4. Sử dụng camera dẫn đường

Xe tự hành được lắp đặt camera và ứng dụng công nghệ xử lý ảnh để thu thập hình ảnh từ môi trường xung quanh, từ đó xác định được vị trí và đường di chuyển tiếp theo.

Ưu điểm: Xe di chuyển linh hoạt trong các khu vực có mức độ di chuyển ngẫu nhiên của nhiều loại phương tiện và con người do ứng dụng xử lý ảnh để tránh vật cản.

Nhược điểm:

- Chi phí triển khai lớn.

- Công nghệ phức tạp đòi hỏi xử lý lượng dữ liệu lớn.

- Mới chỉ được áp dụng cho một số ứng dụng đặc thù.

### 1.5. Sử dụng hệ định vị GPS

Với mô hình này, xe tự hành được trang bị một bộ định vị GPS vệ tinh để xác định được vị trí hiện tại và đường đi tiếp theo. Hệ thống dẫn đường bằng cách định vị điểm bắt đầu di chuyển và điểm đích cuối, từ đó lập trình thiết lập đường đi phù hợp cho xe.

Ưu điểm: Phù hợp với các robot hoạt động ở phạm vị rộng như ngoài sân bãi.

Nhược điểm: Độ chính xác không cao và tốc độ còn chậm.

## 2. Kết luận

Qua việc nghiên cứu về hệ thống dẫn đường cho xe AGV, ta có thể thấy được hệ thống dẫn đường cho xe tự hành nói riêng và robot tự hành nói chung là đa dạng. Đầu vào cho việc thiết lập một thống dẫn đường như trên là điểm xuất phát ban đầu và điểm đích cuối. Từ đó thiết kế đầu ra là đường đi cho robot di chuyển một cách trực tiếp (sử dụng ray cứng, vạch dán dẫn đường...) hay một cách gián tiếp (sử dụng camera để xử lý ảnh, sử dụng bộ định vị vệ tinh GPS).

Phương pháp lập hệ thống dẫn đường bằng camera hay GPS tạo một hệ thống dẫn đường có phần linh hoạt hơn các phương pháp khác, do đó thuật toán sử dụng trong robot tự hành cũng phức tạp hơn.

# CHƯƠNG 3: CÁC PHƯƠNG PHÁP LẬP KẾ HOẠCH ĐƯỜNG ĐI CHO ROBOT TỰ HÀNH

Từ việc nghiên cứu về các hệ thống dẫn đường cho robot tự hành đã trình bày ở chương trước, ta nhận thấy việc tạo một đường dẫn cho robot tự di chuyển theo có thể chia thành hai cách:

Cách thứ nhất, đường dẫn cho robot là cố định (sử dụng ray cứng, vạch dán) tức con người có thể trực tiếp lắp đặt đường đi cho robot tự hành theo ý muốn.

Cách thứ hai, đường dẫn cho robot là không cố định (sử dụng camera xử lý ảnh, định vị GPS); ở cách này mục tiêu di chuyển của robot từ điểm A đến điểm B sẽ linh hoạt hơn. Kỹ thuật thiết lập đường đi sẽ được xác lập dựa trên các thông tin thu nhận được thông qua các thiết bị cảm biến, định vị và hệ thống thu nhận dữ liệu được lắp trên robot.

Các phương pháp quy hoạch đường đi có thể được chia thành 4 nhóm:

- Lập kế hoạch dựa trên đồ thị tìm kiếm

- Lập kế hoạch dựa trên mẫu

- Lập kế hoạch bằng đường cong nội suy

- Lập kế hoạch bằng phương pháp tối ưu

Trong đề tài này, nhóm nghiên cứu xin tập trung nghiên cứu vào phương pháp lập kế hoạch đường đi dựa trên đồ thị tìm kiếm, cụ thể là hai thuật toán: Dijkstra và A\*.

## 1. Nội dung

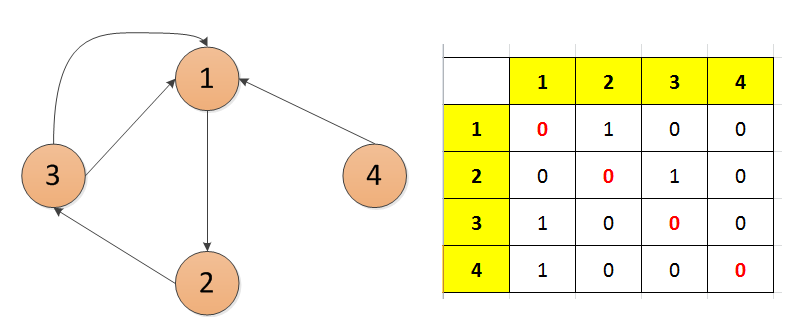
### 1.1. Lý thuyết đồ thị

Trong robot tự hành, ý tưởng cơ bản cho việc di chuyển từ điểm A đến điểm B là diễn tả một không gian trạng thái, không gian trạng thái này thường được biểu diễn dưới dạng lưới hoặc mạng lưới mô tả vị trí của đối tượng trong môi trường.

Từ quan điểm quy hoạch điểm, một giải pháp về thiết lập đường chuyển động được tạo ra bằng thuật toán tìm kiếm trong đồ thị đi qua các trạng thái khác nhau trong lưới.

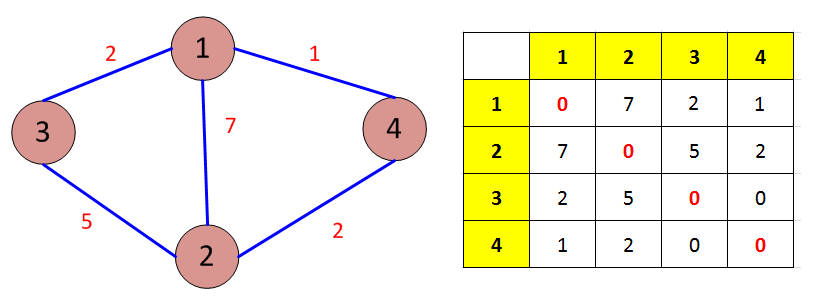
- Đồ thị: là tập hợp các đối tượng gọi là các đỉnh (hoặc nút) nối với nhau bởi các cạnh (hoặc cung). Cạnh có thể có hướng hoặc vô hướng. Đồ thị thường được vẽ dưới dạng một tập hợp các điểm (các đỉnh nối với nhau bằng các đoạn thẳng (các cạnh)).

- Ma trận kề: là ma trận dạng *n x n* biểu diễn cho một đồ thị hữu hạn G gồm *n* đỉnh, trong đó, các ô không nằm trên đường chéo chính *aij* là số cạnh nối hai đỉnh *i* và *j*, còn ô nằm trên đường chéo chính *aii* là hai lần số khuyên tại đỉnh *i*, hoặc chỉ là số khuyên tại đỉnh đó. Mỗi đồ thị có duy nhất một ma trận kề, các đồ thị khác nhau có ma trận kề khác nhau . Trong trường hợp đặc biệt của đồ thị đơn hữu hạn, ma trận kề là một ma trận *(0,1)* với các giá trị 0 nằm trên đường chéo chính. Nếu ma trận là vô hướng, ma trận kề là ma trận đối xứng.



Hình 3.1. Đồ thị có hướng và ma trận kề tương ứng

- Ma trận trọng số: là ma trận kề khi các giá trị 1 được thay bằng trọng số của các cạnh tương ứng.



Hình 3.2. Đồ thị vô hướng có trọng số và ma trận trọng số tương ứng

### 1.2. Thuật toán Dijkstra

#### 1.2.1. Tổng quan

Thuật toán Dijkstra, mang tên của nhà khoa học máy tính người Hà Lan Edsger Dijkstra được ấn bản vào năm 1959, là một thuật toán giải quyết bài toán đường đi ngắn nhất nguồn đơn trong một đồ thị vô hướng hoặc có hướng không có cạnh mang trọng số âm.

Thuật toán này thường được sử dụng trong định tuyến với một chương trình con trong các thuật toán đồ thị hay trong công nghệ hệ thống định vị toàn cầu (GPS).

#### 1.2.2. Lập trình

*Tư duy thuật toán:*

Thuật toán bao gồm một mảng *visited[]* chứa các đỉnh đã xác định được khoảng cách ngắn nhất tới đỉnh nguồn *u*.

Nếu *visited[i]=1* thì khoảng cách từ nguồn tới đỉnh *i* coi như đã được xét. Ở mỗi bước của thuật toán, khoảng cách giữa các điểm được lưu vào trong mảng *distance[].*

1. Tạo ma trận trọng số *cost[][]* từ ma trận kề *G[][]* nhập từ bàn phím hoặc từ file. *cost[i][j]* là mảng 2 chiều có các thành phần là trọng số từ đỉnh *i* đến đỉnh *j*, Nếu không có liên kết giữa đỉnh *i* và *j* thì *cost[i][j]* gán bằng vô cùng.
2. Khởi tạo mảng *visited[]* = 0

*for(i=0; i<n ;i++)*

*visited[i]=0;*

1. Giả sử đỉnh 0 là đỉnh nguồn thì khi đó gán *visited[0]=1*
2. Tạo mảng chứa các giá trị là khoảng cách bằng cách lưu các trọng số của các đỉnh từ đỉnh thứ 0 đến đỉnh thứ *n-1.*

*for(i=0; i<n ;i++)*

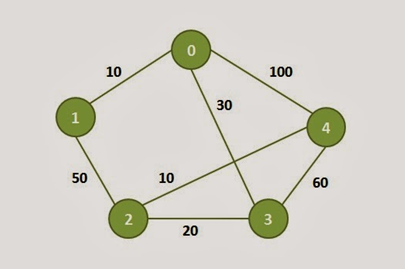
*distance[i]=cost[0][i];*

Ban đầu, khoảng cách của đỉnh nguồn tới đỉnh nguồn lấy bằng 0 (*distance[0]=0*).

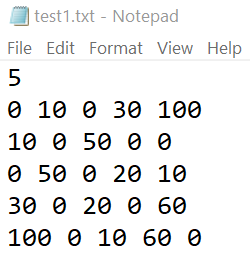
1. Kiểm tra nếu có đường dẫn tốt hơn: tính lại khoảng cách các đỉnh chưa được thăm (*visited[]=0*)

*Kiểm nghiệm thuật toán:*

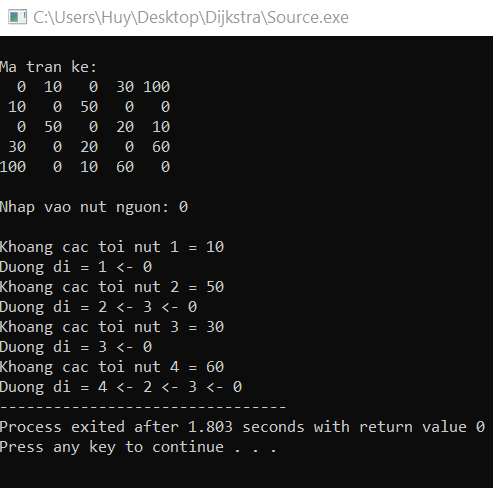
**Cho đồ thị như sau:**



**File .txt để nhập số đỉnh và ma trận trọng số:**



**Kết quả:**



*Chi tiết code viết bằng ngôn ngữ C xem tại Phụ lục I*

#### 1.2.3. Kết luận

- Kết quả sau khi chạy chương trình là chính xác.

- Kế hoạch lập đường đi ngắn nhất bằng phương pháp này có thể tìm được trong một loạt các nút hoặc lưới, thích hợp cho lập kế hoạch toàn cục trong các môi trường hoạt động.

- Tốc độ thực hiện thuật toán chậm ở các khu vực có nhiều nút.

### 1.3. Thuật toán A\*

#### 1.3.1. Tổng quan

Thuật toán A\* (A-star) là một thuật toán tìm kiếm trong đồ thị, cho phép tìm kiếm nút nhanh do sử dụng một đánh giá heuristic (đây là phần mở rộng thuật toán tìm kiếm trên đồ thị Dijkstra).

Heuristic: là phương pháp giải quyết vấn đề dựa trên phỏng đoán, ước chừng, kinh nghiệm, trực giác để tìm ra giải pháp gần như tốt nhất và nhanh chóng. Áp dụng trong thuật toán A\*, các giá trị Heuristic là khoảng cách đường chim bay từ các nút tương ứng tới nút đích.

Từ trạng thái hiện tại, A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài toán 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.

A\* lưu giữ tập các đường đi qua đồ thị từ đỉnh bắt đầu đến đỉnh kết thúc. Tập các đỉnh có thể đi tiếp được lưu trong tập Open. Tập các đỉnh đã xét được lưu trong tập Close.

Thứ tự ưu tiên cho một đường đi *x* được quyết định bởi hàm:

*f(x) = g(x) + h(x)*

Trong đó, *g(x)* là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. *h(x)* là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích.

Hàm *f(x)* có giá trị càng thấp thì độ ưu tiên *x* càng cao.

#### 1.3.2. Lập trình

*Tư duy thuật toán:*

1. Khởi tạo mảng OPEN chỉ chứa nút bắt đầu (startnode), mảng CLOSE rỗng.

Khi đó: *g(startnode) = 0*

*f(startnode) = h(startnode)*

1. Vòng lặp:
2. Nếu OPEN rỗng, suy ra bài toán vô nghiệm và thoát.
3. Nếu OPEN không rỗng, chọn nút *Tmax* trong OPEN sao cho *f(Tmax)* nhỏ nhất

+ Lấy *Tmax* ra khỏi OPEN và đưa vào CLOSE

* Nếu *Tmax* chính là nút đích thì thông báo lời giải là *Tmax*
* Nếu *Tmax* không phải là nút đích thì tạo ra tất cả các trạng thái kế tiếp của *Tmax*. Gọi 1 trạng thái này là *Tk.*

+ Với mỗi *Tk*, thực hiện:

* Tính: *g(Tk) = g(Tmax) + Cost(Tmax, Tk)*
* Nếu trongOPENcó *T’k* mà: *g(Tk) < g(*T’k*)* thì đặt

*g(*T’k*) = g(Tk)*

Tính lại *f(T’k)*

Đặt cha của *T’k* là *Tmax.*

* Nếu trong CLOSE có *T’k* mà: *g(Tk) < g(*T’k*)* thì đặt *g(*T’k*) = g(Tk)*

Tính lại *f(T’k)*

Đặt cha của *T’k* là *Tmax.*

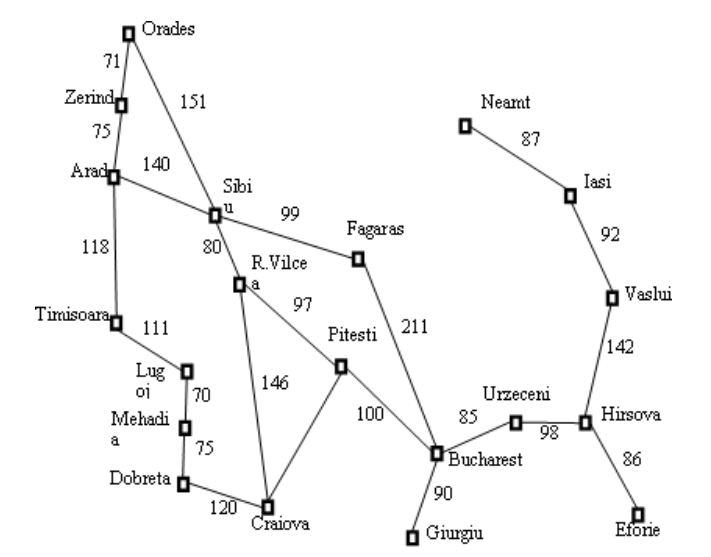
* Nếu *Tk* chưa xuất hiện trong cả OPEN và CLOSE thì:

Thêm *Tk* vào OPEN

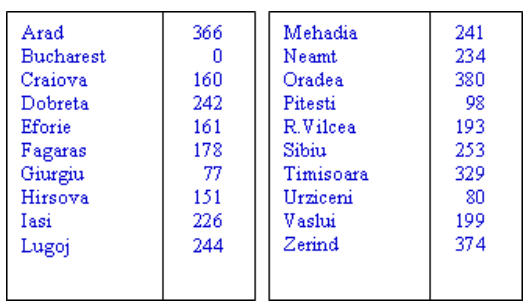
Tính *f(Tk) = g(Tk) + h(Tk)*

*Kiểm nghiệm thuật toán:*

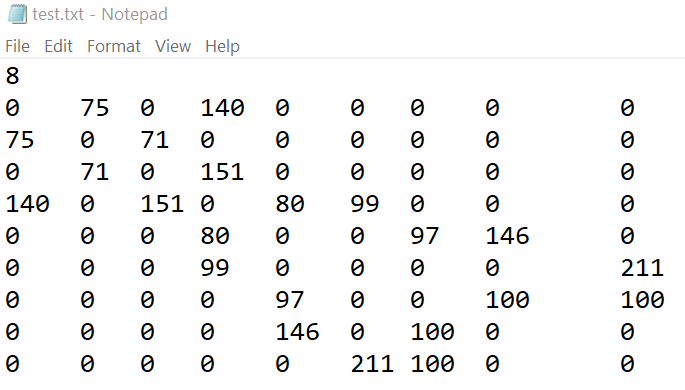
**Xét đồ thị:**



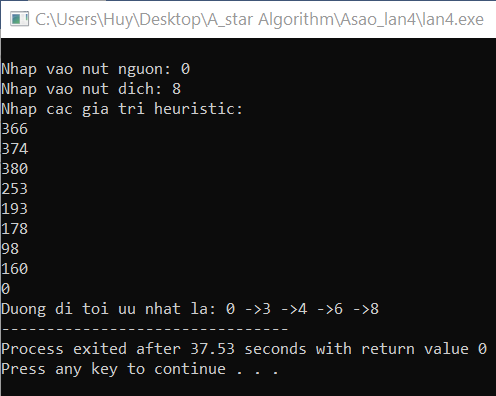
**Bảng giá trị heuristic:**



**Lập ma trận trọng số:**



**Kết quả:**

**

*Chi tiết code viết bằng ngôn ngữ C xem tại Phụ lục II*

#### 1.3.3. Kết luận

- Kết quả xuất ra sau khi chạy chương trình là chính xác.

- Do phát triển từ thuật toán Dijkstra và việc tìm kiếm heuristic đã làm giảm thời gian tính toán.

# CHƯƠNG 4: XÂY DỰNG HỆ THỐNG LẬP KẾ HOẠCH ĐƯỜNG ĐI CHO ROBOT TỰ HÀNH

Trong phần trước, chúng ta đã thực nghiệm hai thuật toán tìm đường đi ngắn nhất qua IDE hỗ trợ lập trình trên máy tính Windows. Bây giờ, để trực quan hóa thuật toán ta sẽ áp dụng lên bản đồ Google Maps (GM). Nhóm nghiên cứu sẽ triển khai nội dung trên máy tính nhúng Raspberry Pi cùng module SIM7600 có định vị GPS.

Mục tiêu trong phần này là lấy tọa độ vị trí hiện tại của robot tự hành nhờ bộ định vị GPS, từ đó lập trình thuật toán xác định đường đi ngắn nhất tới một điểm đích tùy ý với GM API.

## 1. Nội dung

#### 1.1. Tổng quan

- Google Maps là bản đồ trực tuyến miễn phí do Google phát hành, quản lý. Ứng dụng này cho phép người dùng sử dụng các tính năng như: tìm vị trí, tìm đường, hiện thị tuyến đường tối ưu, hướng dẫn bắt xe bus, các địa điểm chỉ định như bệnh viện, cây xăng, ATM...

- API là viết tắt của Application Programming Interface (Giao diện lập trình ứng dụng), là một phần mềm trung gian cho phép hai ứng dụng nói chuyện với nhau.

- GM API là phương pháp sử dụng cho phép web B (cá nhân) có thể dùng dịch vụ hiển thị nội dung ở web A (ở đây là Google Maps) thông qua GM API. Bản đồ tại web A sẽ được nhúng và web B. Sau đó, web B có thể sử dụng mọi dịch vụ, tính năng mà GM đã cung cấp.

- Raspberry Pi (RPI) là một máy tính nhúng có kích thước nhỏ gọn, chỉ có 1 board mạch nhưng được trang bị đầy đủ các chip Wifi, Bluetooth... Ngoài ra có thể kết nối các thiết bị ngoại vi như chuột, bàn phím, màn hình.

#### 1.2. Thiết bị phần cứng

Phần cứng bao gồm:

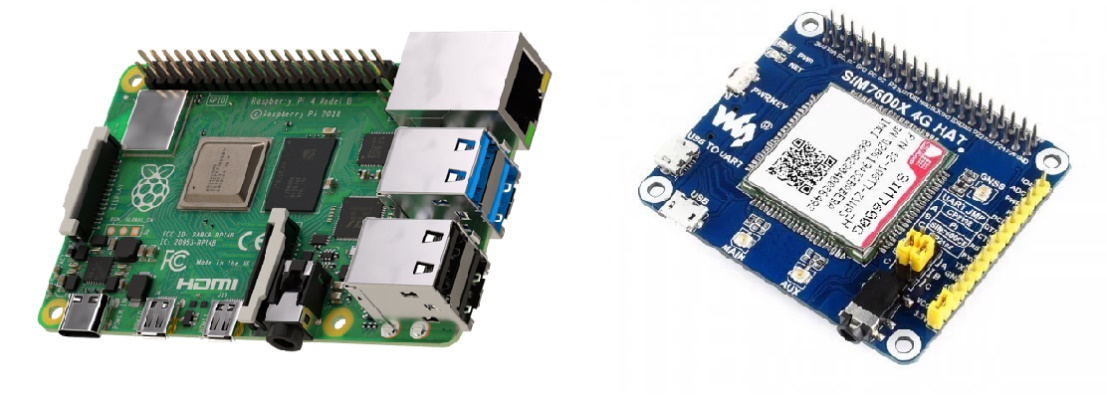
- Raspberry Pi 4 Model B (Máy tính nhúng).

- Module Sim7600G-H (GPS định vị).

- Nguồn cấp cho RPI.

- Dây nối, Ăng-ten.

- Màn hình cảm ứng LCD 3.5 inch.



Hình 4.1. Raspberry Pi 4 Model B (trái) và Module Sim7600G-H (phải)

*Sơ đồ khối kết nối các thiết bị:*

Ăng-ten

Màn hình hiển thị

SIM7600G-H

RPI

Nguồn



Hình 4.2. Hệ thống khi kết nối các thiết bị với nhau

- Nguồn điện cấp vào dùng cho RPI và SIM7600 là 5V – 1,5A, sử dụng adapter: INPUT 220V 50Hz, OUTPUT 5V – 1,5A.

- Kết nối RPI với SIM7600 thông qua các chân có sẵn trên 2 module. Phương thức kết nối là truyền thông nối tiếp:

|  |  |  |
| --- | --- | --- |
| SIM7600 | RPI | MÔ TẢ |
| 5V | 5V | Nguồn cung cấp 5V |
| GND | GND | Đất |
| TXD | RXD (BCM P15) | UART pin |
| RXD | TXD (BCM P14) | UART pin |

- Kết nối SIM7600 với màn hình laptop máy tính thông qua phần mềm VNC Viewer (Quét màn hình các thiết bị dùng chung một địa chỉ IP). Ngoài ra, có thể kết nối RPI với màn hình rời thông qua cổng Micro-HDMI.

- Kết nối Ăng-ten với SIM7600 bằng dây dẫn qua các chân định sẵn.

#### 1.3. Lập trình

Việc lập trình thuật toán từ phần này trở đi nhóm em sẽ viết bằng ngôn ngữ Python do nhận thấy ngôn ngữ này linh hoạt và hỗ trợ tốt hơn khi làm với GM API.

Trên Python có rất nhiều các gói (package) để trực quan hóa (visualization) các bản đồ của GM. Một trong số đó là: folium, gmplot, gmaps. Các gói gmplot, gmaps đều sử dụng các API của GM. Tuy nhiên gần đây việc sử dụng những thông tin này yêu cầu người dùng phải có tài khoản trả phí của Google. Trái lại folium sử dụng thông tin bản đồ từ OpenStreetMap là một database mở được chia sẻ tới người dùng.

Thuật toán nhóm em sử dụng ở đây là thuật toán Dijkstra.

*Tư duy lâp trình như sau:*

- Bước 1: Khởi tạo truyền thông, bật và tắt cho SIM7600G-H

1. Khởi tạo truyền thông, gửi lệnh AT đến SIM7600 để lấy giá trị GPS.

2. Chuyển đổi dữ liệu thô từ SIM7600 trả về chỉ còn kinh độ và vĩ độ.

3. Định nghĩa hàm bật, tắt SIM7600.

- Bước 2: Bật SIM7600

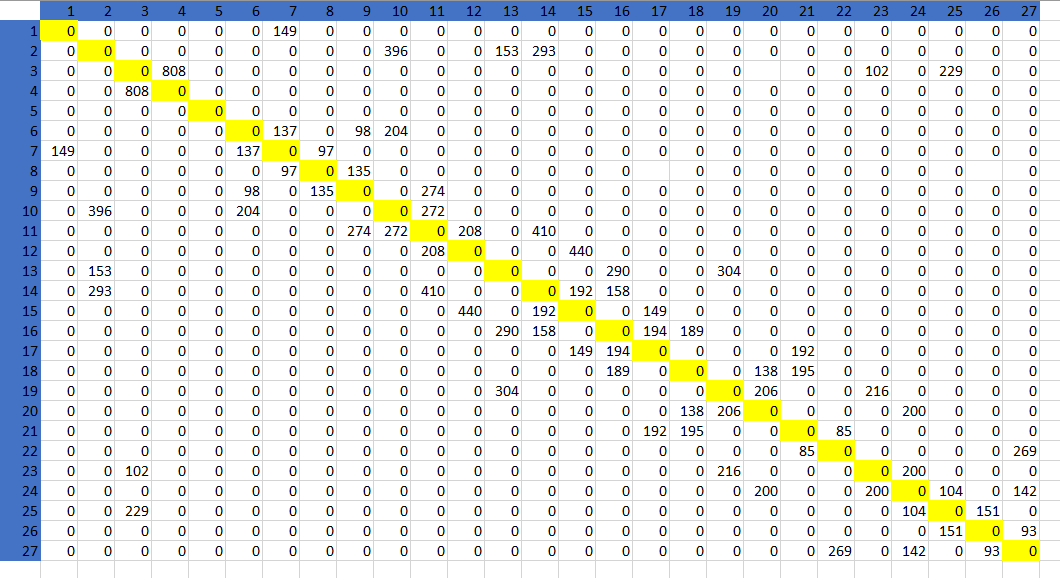
- Bước 3:

1. In ra vị trí các điểm lên bản đồ GM.

2. Từ vị trí lấy ra kinh độ, vĩ độ.

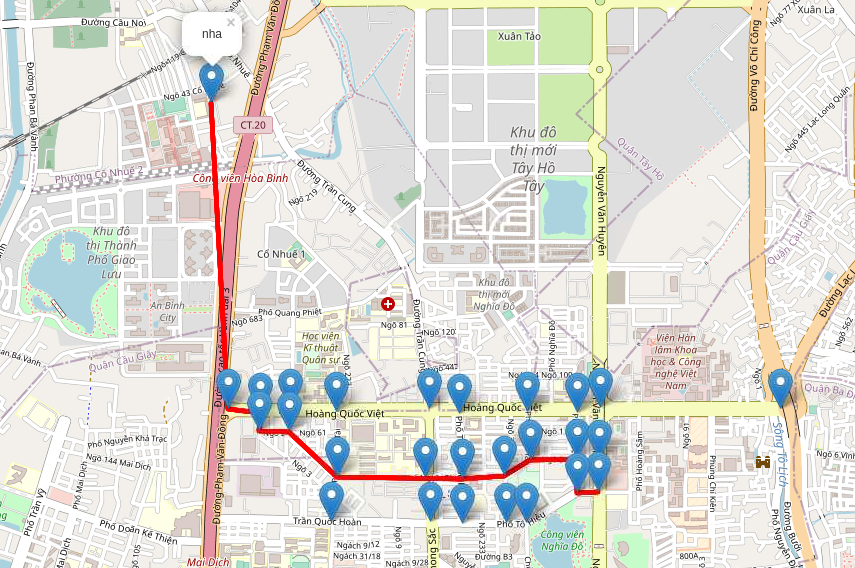
3. Tìm đường đi ngắn nhất từ vị trí của mình đến điểm đích được nhập

- Bước 4: Tắt SIM7600.



Hình 4.3. Ma trận trọng số

Sau khi chạy, chương trình sẽ xuất ra một file định dạng *.html* , mở file này lên ta thu được kết quả như sau:



Hình 4.4. Kết quả đường đi ngắn nhất sau khi lập trình với GM API

- Vị trí “nha” ở đây lấy từ bộ định vị GPS trên SIM7600, điểm đích là điểm nhập bằng tay. Kết quả đưa về là đường đi nối từ vị trí hiện tại qua các điểm trung gian (đặt tại các nút giao trên đường) và tới điểm đích.

- Ở đây, việc chọn các điểm trung gian không phải là ngẫu nhiên. Ta chọn các điểm trung gian sao cho chúng nằm ở đầu mút mỗi con đường, ở các ngã 3, ngữ 4... Lý do cho việc này là vì nhóm dùng hàm Polyline trên thư viện Folium để nối giữa các điểm, đường nối là đường thẳng.

*Chi tiết code viết bằng ngôn ngữ Python xem tại Phụ lục III*

#### 1.1.3. Kết luận

- Từ các điểm đã lấy trên bản đồ, thuật toán đã chạy và cho kết quả chính xác.

- Lấy các điểm càng nhiều, mật độ càng dày thì tăng tính linh động trong việc tạo đường đi sẽ càng cao.

# KẾT LUẬN

Kết thúc đề tài, nhóm nghiên cứu đã học được rất nhiều kiến thức mới bổ ích về robot, robot tự hành, thuật toán tìm đường đi ngắn nhất, API..., học và rèn luyện thêm về kỹ năng lập trình, tiếp xúc và thực hành trên máy tính nhúng... Tuy nhiên, do thời gian có hạn và năng lực còn hạn hẹp, không tránh khỏi những sai sót trong quá trình thực hiện nên đề tài chưa hoàn thành được ở mức tốt nhất.

Cuối cùng nhóm xin được gửi lời cảm ơn đến nhà trường, thầy cô vì đã tạo cơ hội cho sinh viên được tiếp cận và tham gia làm những đề tài khoa học bổ ích.

# PHỤ LỤC

## I. Code kiểm nghiệm thuật toán Dijkstra viết bằng C

#include<stdio.h>

#include<conio.h>

#define INFINITY 99999999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

void XuatMTKe(int G[][MAX], int n);

int main(){

int G[MAX][MAX], i, j, n, u;

FILE \*f = fopen("test1.txt", "rt");

if (f == NULL){

printf("Doc file loi !!!");

return 0; }

fscanf(f, "%d", &n);

for (int i = 0; i<n; i++){

for (int j = 0; j<n; j++){

fscanf(f, "%d", &(G[i][j]));}}

XuatMTKe(G, n);

printf("\nNhap vao nut nguon: ");

scanf("%d", &u);

dijkstra(G, n, u);

return 0;}

void dijkstra(int G[MAX][MAX], int n, int startnode){

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

//pred[] luu cac dinh ma khoang cach ngan nhat tu no den dinh nguon

//count dem so nut ma di qua den den dich

//Tao ma tran trong so

for (i = 0; i<n; i++)

for (j = 0; j<n; j++)

if (G[i][j] == 0)

cost[i][j] = INFINITY;

else

cost[i][j] = G[i][j];

//Khoi tao mang pred[],distance[] va visited[]

for (i = 0; i<n; i++){

distance[i] = cost[startnode][i];

pred[i] = startnode;

visited[i] = 0;}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n-1){

mindistance = INFINITY;

//Nut ke tiep cho khoang cach nho nhat

for (i = 0; i<n; i++)

if (distance[i]<mindistance && visited[i]==0){

mindistance = distance[i];

nextnode = i;}

//Kiem tra neu co duong dan tot hon: tinh lai khoang cach cac nut chua duoc tham

visited[nextnode] = 1;

for (i = 0; i<n; i++)

if (visited[i]==0)

if (mindistance + cost[nextnode][i]<distance[i]){

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;}

count++;}

//In ra duong di va khoang cach den moi nut

for (i = 0; i<n; i++)

if (i != startnode){

printf("\nKhoang cac toi nut %d = %d", i, distance[i]);

printf("\nDuong di = %d", i);

j = i;

do{

j = pred[j];

printf(" <- %d", j);

} while (j != startnode);}}

// Xuat ket qua ma tran ke cua do thi ra man hinh.

void XuatMTKe(int G[MAX][MAX], int n){

printf("\nMa tran ke:\n");

for (int i = 0; i<n; i++){

for (int j = 0; j<n; j++)

printf("%3d ", G[i][j]);

printf("\n");}}

## II. Code kiểm nghiệm thuật toán A\* viết bằng C

#include<stdio.h>

#include<conio.h>

#define INFINITY 99999999

#define MAX 10

void asao(int trongso[MAX][MAX],int khoangcach[MAX], int endnode, int startnode, int n);

int main(){

int trongso[MAX][MAX], khoangcach[MAX], i, j, n, u, m;

FILE \*f = fopen("test.txt", "rt");

if (f == NULL){

printf("Doc file loi !!!");

return 0;}

fscanf(f, "%d", &n);

for (int i = 0; i<=n; i++){

for (int j = 0; j<=n; j++){

fscanf(f, "%d", &(trongso[i][j]));}}

printf("\nNhap vao nut nguon: ");

scanf("%d", &u);

printf("Nhap vao nut dich: ");

scanf("%d", &m);

printf("Nhap cac gia tri heuristic:\n");

for (int i = 0; i<=n; i++) {

scanf("%d", &khoangcach[i]);}

printf("Duong di toi uu nhat la: ");

asao(trongso, khoangcach ,m, u, n);

return 0;}

void asao(int cost[MAX][MAX],int h[MAX], int endnode, int startnode, int n){

int ff[MAX]; // mang chua gia tri ham uoc luong khoang cach f

long int g[MAX],tmax, open[MAX][4], close[MAX][4], tg, tk, tkk, tmin

int cha;

int kq[100];

int inkq[100], chay;

int i, j, k, count;

tg=0;

int tgg=0;

k=0;

chay=0;

cha=0;

tmax=startnode;

open[0][0]=startnode;

g[startnode]=0;

ff[startnode]=0;

open[0][1]=0;

open[0][2]=0;

open[0][3]=0;

open[0][4]=cha;

while(tmax<endnode){ // tmax la dinh dang xet, ban dau dinh tmax=0

close[tg][0]=open[tgg][0];// lay gia tri nho nhat trong open luu vao close

close[tg][1]=open[tgg][1];

close[tg][2]=open[tgg][2];

close[tg][3]=open[tgg][3];

close[tg][4]=open[tgg][4];

open[tgg][0]=tmax;// gan bang 99999 de xoa gia tri vua xet

open[tgg][1]=99999;

open[tgg][2]=99999;

open[tgg][3]=99999;

open[tgg][4]=99999;

tg++; // tg la so phan tu cua mang close

for(tk=0; tk<=n; tk++){//tk la bien chay, n la so phan tu

if (cost[tmax][tk] != 0) { // tinh f(x) cua ca diem dan toi dinh tmax

g[tk]=g[tmax] + cost[tmax][tk];

ff[tk]=g[tk] + h[tk];

cha = tmax;

k++;

open[k][0] = tk; // luu cac gia tri vua tinh vào mang open

open[k][1]=g[tk];

open[k][2]=h[tk];

open[k][3]=ff[tk];

open[k][4]=cha;

kq[k]=cha; }} // k la so phan tu cua mang open

for(tkk=0; tkk<k; tkk++) // kiem tra xem trong mang open co phan tu nao trung voi

if(open[tkk][0] == tk){ // voi phan tu vua xet khong, tkk la bien chay trong open

if(g[tk] < open[tkk][1]) {

open[tkk][1] = g[tk];

g[tkk] = g[tk];

open[tkk][3] = g[tk]+h[tk];

ff[tkk] = g[tk] + h[tk];

open[tkk][4] = open[tk][4];

tmax = open[tk][0];}} for(tkk = 0; tkk <=tg; tkk++) // kiem tra xem trong mang open con phan tu nao trung voi

if(close[tkk][0] == tk) { // phan tu vua xet hay không,tkk la bien chay trong open

if(g[tk] < close[tkk][1]){

close[tkk][1] = g[tk];

g[tkk] = g[tk];

close[tkk][3] = g[tk]+h[tk];

ff[tkk] = g[tk]+h[tk];

open[tkk][4] = open[tk][4];

min = 9999;

for(j = 0; j <= k; j++){ // tim vi tri co f(x) nho nhat trong mang open

if (min >= open[j][3]) {

min = open[j][3];

tmax = open[j][0]; // tmax duoc gan bang dinh co gia tri f(x) nho nhat

tgg = j; // tgg la vi tri cua dinh tmax trong mang open

cha = kq[tgg]; }} // luu lai gia tri cha cua dinh tmax vao mang kq;

inkq[chay] = cha; // luu gia tri cha cua dinh tmax vao mang inkq

chay++;} // so phan tu cua mang inkq

for(i=0; i<chay; i++)

for(j = i+1; j<chay; j++)

if (inkq[i] == inkq[j])

inkq[j]=99999; // de loai bo duong di lap lai

for(i = 0; i<chay; i++)

if (inkq[i] != 99999)

printf("%d ->",inkq[i]); // in ra ket qua

printf("%d",endnode); }

## III. Code mô phỏng thuật toán tìm đường đi ngắn nhất cho RBTH trên bản đồ GM bằng ngôn ngữ Python

import RPi.GPIO as GPIO

import serial

import time

import pandas as pd

import pandas as np

import folium

from   pandas import DataFrame, Series

from math import cos, asin, sqrt

import numpy as np

#khoi tao truyen thong

ser = serial.Serial('/dev/ttyS0',115200)

ser.flushInput()

#khoi tao ban dau

power\_key = 6

rec\_buff = '' #luu gia tri doc ve

time\_count = 0 #thoi gian tra ve ket qua

gps\_latitude = ''

gps\_longitude = ''

data = ''

def transfer\_data(data):

    gps\_D = int(float(data)/100)

    gps\_M = int(float(data)%100/1)

    gps\_S = (float(data)%1)\*60

    gps = gps\_D + round(gps\_M/60,6) + round(gps\_S/3600,6)

    return round(gps,7)

def handle\_data(data):

    answer1 = ''

    answer1 = str(data.decode())

    answer1 = answer1.replace(' ',',')

    answer1 = answer1.replace('\n',',')

    answer1 = answer1.replace('\r','')

    answer1 = answer1.split(',')

    gps\_latitude = transfer\_data(answer1[2])

    gps\_longitude = transfer\_data(answer1[4])

    print('Latitude: ', gps\_latitude)

    print('Longitude: ',gps\_longitude)

    fboiz=[[21.04620428,    105.7816722],

    [21.04620428,   105.790308],

    [21.0462455,    105.7976628],

    [21.04622489,   105.8053931],

    [0,   0],

    [21.04620428,   105.7843447],

    [21.04605999,   105.7830416],

    [21.04531791,   105.7830195],

    [21.04527669,   105.7843005],

    [21.0460806 ,   105.7863104],

    [21.04350394,   105.7863546],

    [21.04166933,   105.7860895],

    [21.04599815,   105.791589],

    [21.04346271,   105.7901534],

    [21.04168995,   105.7903522],

    [21.04342149,   105.7917216],

    [21.04160749,   105.7917436],

    [21.04352456,   105.7935326],

    [21.04610121,   105.7945265],

    [21.04416357,   105.7946591],

    [21.04164872,   105.7935989],

    [21.04164872,   105.7944603],

    [21.04605999,   105.7966468],

    [21.04422541,   105.7966689],

    [21.04418418,   105.7975965],

    [21.04290615,   105.7975745],

    [21.04286492,   105.7966468]

    ]

    name=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]

    lat=gps\_latitude

    long=gps\_longitude

    print(lat,long)

    zoom = 15

    gmap2 = folium.Map(location=(lat, long), zoom\_start=zoom)

    marker = folium.Marker(location=(lat,long), popup = 'Vi tri hien tai')

    marker.add\_to(gmap2)

    n=27

    i=0

#ham danh dau cac diem tren ban ho

    while (i<n):

        marker = folium.Marker(location=(fboiz[i][0],fboiz[i][1]), popup = name[i])

        marker.add\_to(gmap2)

        i=i+1

#nhap ma trang trong so

    graph={'1':{'7':149},

    '2':{'10':396,'13':153,'14':293},

    '3':{'4':808,'23':102,'25':229},

    '4':{'3':808},

    '6':{'7':137,'9':98,'10':2004},

    '7':{'1':149,'6':137,'8':97},

    '8':{'7':97,'9':135},

    '9':{'6':98,'8':135,'11':274},

    '10':{'2':396,'6':204,'11':272},

    '11':{'9':274,'10':272,'12':208,'14':410},

    '12':{'11':208,'15':440},

    '13':{'2':152,'16':290,'19':304},

    '14':{'2':293,'11':410,'15':192,'16':158},

    '15':{'12':440,'14':192,'17':149},

    '16':{'13':290,'14':158,'17':194,'18':189},

    '17':{'15':149,'16':194,'21':192},

    '18':{'16':189,'20':158,'21':195},

    '19':{'13':304,'20':206,'23':216},

    '20':{'18':138,'19':206,'24':200},

    '21':{'17':192,'18':195,'22':85},

    '22':{'21':85,'27':269},

    '23':{'3':102,"19":216,'24':200},

    '24':{'20':200,'23':200,'25':104,'27':142},

    '25':{'3':229,'24':104,'26':151},

    '26':{'25':151,'27':93},

    '27':{'22':269,'24':142,'26':93}}

    lat=gps\_latitude

    long=gps\_longitude

#Chuong trinh con tim duong di ngan nhat

    def dijkstra(graph,start,goal):

#cac diem da xet

        shortest\_distance = {}

        predecessor = {}

#mang chua cac diem tren ban do

        unseenNodes = graph

        infinity = 9999999

        path = []

        'khoi tao mang shortest\_distance'

        for node in unseenNodes:

            shortest\_distance[node] = infinity

        shortest\_distance[start] = 0

        while unseenNodes:

#diem duoc xet tiep theo

            minNode = None

            for node in unseenNodes:

                if minNode is None:

                    minNode = node

                elif shortest\_distance[node] < shortest\_distance[minNode]:

                    minNode = node

#kiem tra xem da phai duong di ngan nhat hay chua, neu chua thi xet lai

            for childNode, weight in graph[minNode].items():

                if weight + shortest\_distance[minNode] < shortest\_distance[childNode]:

                    shortest\_distance[childNode] = weight + shortest\_distance[minNode]

                    predecessor[childNode] = minNode

            unseenNodes.pop(minNode)

        currentNode = goal

 #ham kiem tra dich den

        while currentNode != start:

            try:

                path.insert(0,currentNode)

                currentNode = predecessor[currentNode]

            except KeyError:

                print('Path not reachable')

                break

#path chinh la cac diem thoa man

        path.insert(0,start)

        if shortest\_distance[goal] != infinity:

            print('quang duong phai di ' + str(shortest\_distance[goal]))

            print('duong di ngan nhat ' + str(path))

        return path

#In ra duong di ngan nhat

    def distance(lat1, lon1, lat2, lon2):

        R = 6371 # Ban kinh trai dat (km)

        p = np.pi/180 # gia tri pi/180

        a = 0.5 - cos((lat2 - lat1) \* p)/2 + cos(lat1 \* p) \* cos(lat2 \* p) \* (1 - cos((lon2 - lon1) \* p)) / 2

        return 2\*R\*asin(sqrt(a))

    min=10

    print(min)

#Tim diem gan vi tri hien tai nhat

    v=0

    for i in range(0,26):

        if min > distance( fboiz[i][0],fboiz[i][1], lat, long):

                min = distance(fboiz[i][0],fboiz[i][1], lat, long )

                print(min)

                v=i

    print(min)

    print(v)

    k=str(v+1)

    c=dijkstra(graph,k, '26')

    j=1

    d=[j for j in range(0,30)]

    k=len(c)

    i=0

    m=0

#Chuyen kieu du lieu tu string sang int, roi gan vao list d[i]

    for i in range(0,k):

        m=int(c[i])

        d[i]=m

#lap list gom co k hang va 2 cot

    fbgirl = []

    for i in range(0, k+1):

        fbgirl.append([])

        for j in range(0, 2):

            x =fboiz[i][j]

            fbgirl[i].append(x)

#tao list chua toa do cua cac diem di qua

    j=0

    fbgirl[0][0]=lat

    fbgirl[0][1]=long

    for i in range(1,k+1):

        m=d[j]

        fbgirl[i][0]=fboiz[m-1][0]

        fbgirl[i][1]=fboiz[m-1][1]

        j=j+1

#noi cac diem voi nhau

    for i in range(0,k+1):

        folium.PolyLine(fbgirl, color='red', weight=4.5, opacity=.5).add\_to(gmap2)

    gmap2.save("foliumMarkerMultiple.html")

#Truyen lenh AT den mudule SIM7600

def send\_at(command,back,timeout):

    rec\_buff = ''

    ser.write((command+'\r\n').encode())

    time.sleep(timeout)

    if ser.inWaiting():

        time.sleep(0.01 )

        rec\_buff = ser.read(ser.inWaiting())

    if rec\_buff != '':

        if back not in rec\_buff.decode():

            print(command + ' ERROR')

            print(command + ' back:\t' + rec\_buff.decode())

            return 0

        else:

            print(rec\_buff.decode())

            if '+CGPSINFO: ' in rec\_buff.decode():

                handle\_data(rec\_buff)

                time.sleep(30)

            return 1

    else:

        print('GPS is not ready')

        return 0

def get\_gps\_position():

    answer = 0

    print('Start GPS session...')

    rec\_buff = ''

    send\_at('AT+CGPS=1,1','OK',1)

    time.sleep(2)

    answer = send\_at('AT+CGPSINFO','+CGPSINFO: ',1)

    if 1 == answer:

        answer = 0

        if ',,,,,,' in rec\_buff:

            print('GPS is not ready')

            time.sleep(1)

    else:

        print('error %d'%answer)

        rec\_buff = ''

        send\_at('AT+CGPS=0','OK',1)

        time.sleep(1.5)

def power\_on(power\_key):

    print('SIM7600X is starting:')

    GPIO.setmode(GPIO.BCM)

    GPIO.setwarnings(False)

    GPIO.setup(power\_key,GPIO.OUT)

    time.sleep(0.1)

    GPIO.output(power\_key,GPIO.HIGH)

    time.sleep(2)

    GPIO.output(power\_key,GPIO.LOW)

    time.sleep(20)

    ser.flushInput()

    print('SIM7600X is ready')

def power\_down(power\_key):

    print('SIM7600X is loging off:')

    GPIO.output(power\_key,GPIO.HIGH)

    time.sleep(3)

    GPIO.output(power\_key,GPIO.LOW)

    time.sleep(18)

    print('Good bye')

power\_on(power\_key)

get\_gps\_position()

power\_down(power\_key)

# TÀI LIỆU THAM KHẢO

1. Quách Hải Thọ, Phạm Anh Phương (2019), “Một số đánh giá tổng quan về kỹ thuật thiết lập đường đi cho xe tự hành”, TẠP CHÍ KHOA HỌC XÃ HỘI, NHÂN VĂN VÀ GIÁO DỤC, ISSN: 1859 – 4603.

2. Nguyễn Trường Thịnh (2008), Giáo trình Kỹ thuật Robot, NXB Đại Học Quốc Gia Hồ Chí Minh.

3. Roland Siegwart, Illah R. Nourbakhsh, Davide Scaramuzza. Autonomous Mobile Robots, Published by The MIT Press.

4. Steven M. LaVelle (2006), Planning algorithms, Published by Cambridge University Press.

5. <https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Dijkstra>

6. https://vi.wikipedia.org/wiki/Giải\_thuật\_tìm\_kiếm\_A\*

7. SIM7600E-H-4G-HAT-Manual-EN