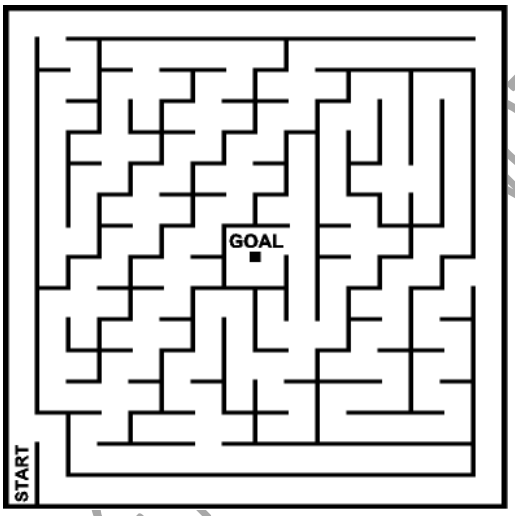
Để biết thêm những thông tin chi tiết các bạn vui long vào Website sau để tìm hiểu thêm: http://www.ieee.ucla.edu/

1.1. Tìm hiểu về các cuộc thi micromouse

1.2.1 Giải thích nguyên tắc thi Micromouse

Về cơ bản, Micromouse là một robot thông minh tự hành có hai hoặc nhiều bánh xe có thể tự động di chuyển mà không cần đến tác động hay điều khiển nào từ con người, nó được lập trình trước.

Micromouse sẽ di chuyển trong một mê cung gồm các ô vuông được ngăn cách bởi các bức tường, và mê cung sẽ không biết trước.



Hình 1.2a: Minh họa mê cung 16*16

Micromouse chạy trong một mê cung có kích thước tiêu chuẩn cho trước gồm 16*16 hoặc 32*32 ô, người lập trình chỉ được biết trước ô đích và ô bắt đầu, còn lại các bức tường trong mê cung thì không biết trước. Mục đích là phải làm sao để Micromouse có thể tìm được đến đích, Có nhiều lần chạy khác nhau, làm sao cho Micromouse di chuyển nhanh nhất đến đích có thể, lấy điểm của lần chạy có thời gian ngắn nhất để chấm điểm.

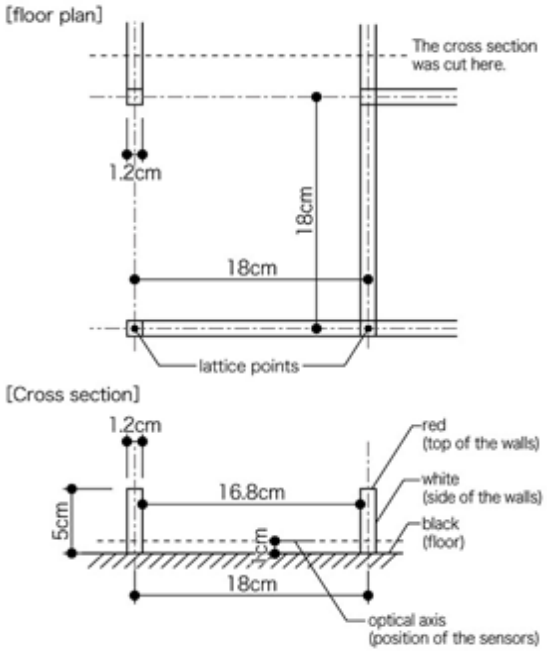
1.2.2 Nguyên tắc thiết kế mê cung, các tiêu chuẩn quốc tế và cách di chuyển của Micromouse.

Hình 1.2a minh họa một loại cấu trúc mê cung 16*16. Trong các cuộc thi quốc tế. Mới đây, Nhật Bản phát triển thêm một loại mê cung mới có kích trước 32*32 ô, quy định quốc tế của mê cung như sau:

Đối với mê cung 16*16

+ Tổng cộng gồm 16*16 = 255 ô vuông
+ Nền được sơn màu đen.

+ Tường được sơn màu trắng
+ Phía trên của bức tường được sơn màu đỏ
+ Khoảng cách giữ 2 bức tường là 16.8cm, bức tường dày 1.2cm và cao 5cm
+ Điểm bắt đầu chỉ có 1 bức tường mở duy nhất và được biết trước vị trí ô đó trong mê cung (Chú ý là không biết ngõ vào đích hướng nào) .
+ Ô đích gồm 4 ô vuông không ngăn cách bởi bức tường nào và chỉ có 1 lối vào duy nhất (thường nằm ở giữa mê cung). Ô đích là được biết trước.



Hình 1.2b. Mô tả kích thước tiêu chuẩn của mê cung

Đối với mê cung 32*32: Có tất cả 32*32 = 1024 ô vuông, Tất cả các kích thước được giảm đi ½ so với mê cung 16*16 và vẫn giữ nguyên kích thước tổng thể.

- Quy tắc di chuyển cơ bản:
  - Micromouse sẽ di chuyển từ điểm bắt đầu (start) đến đích là 4 ô nằm ở giữa mê cung (goal), nhiệm vụ là đi đến đích trong thời gian ngắn nhất. (minh họa mê cung hình 1.2a)
  - Trong lúc di chuyển micromous không được vượt ngang qua các bức tường, không được làm đổ các bức tường.
  - Người tham gia chỉ được chỉnh sửa phần cứng và không được chỉnh sửa chương trình trong lúc cuộc thi bắt đầu diễn ra đến hết.
- Quy định thiết kế micromouse

- Micrmouse Được quy định như sau: Chiều cao không giới hạn, ngang không vượt quá kích thước 25x25 cm.
- Robot Micomouse không giới hạn số lượng và các loại cảm biến sử dụng.
- Micromuse có thể có 2 hoặc nhiều bánh xe túy ý.
- Mê cung là một ma trận không được biết trước, Khi cuộc thi bắt đầu mê cung có thể được thay đổi tùy vào giám khảo và ban tổ chức, trong suốt cuộc thi mê cung sẽ giữ nguyên không thay đổi, và Micromouse cũng không được phép sửa đổi code thuật toán di chuyển khi đang trong cuộc thi.

Ở trên là những quy tắc cơ bản, còn nhiều quay tắc chi tiết khác các bạn vui lòng vào Trang WEB sau để tham khảo thêm: http://www.ieee.ucla.edu/

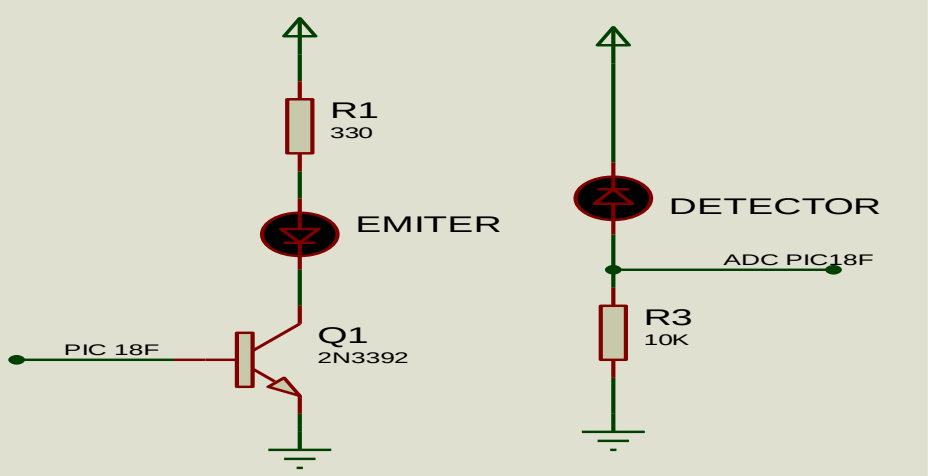
CHƯƠNG 2 THIẾT KẾ PHẦN CỨNG CHO MICROMOUSE

2.1. Cảm biến.

Trong Quy tắc thiết kế Micromouse không giới hạn số lượng cảm biến, do đó có thể đặt bao nhiêu cảm biến tùy ý, ở đây Micromouse cần phải tự tránh bật cản được và cập nhật thông tin các bức tường dựa vào cảm biến. Do đó có thể sử dụng cảm biến siêu âm, cảm biến hồng ngoại hoặc cảm biến ảnh…

Hầu hết mọi người đều ưa chuộng sử dụng cảm biến hồng ngoại cho việc phát hiện và cập nhật các thông tin liên quan đến bức tường. Có thể đặt ít nhất 3 cặp cảm biến thu - phát hồng ngoại: 1 trước, 1 trái và 1 phải, tùy nhiên tùy vào người lập trình và yêu cầu cao để có thể bố trí thêm nhiều cảm biến. Thường chúng ta nên bố trí 2 cặp cảm biến 2 bên và 2 cặp phía trước song song (vì khi chạy phải sử dụng 2 cảm biến trước để hiệu chỉnh, chi tiết đọc phần tiếp theo).

Mạch nguyên lý gợi ý như sau:



Hình 2.1c.Mạch nguyên lý thu phát hồng ngoại.

Các thông số và cách mắc:

+ Đối với LED phát (Emiter): Cực dương (A) được qua trở 330R lên 5V, cực âm (K) được nối vào ngõ C (Colecter) của Transistor ngược NPN (thường sử dụng loại C1815) được mắc như hình 2.1c.

- Thực tế nếu các bạn muốn làm mạch nhỏ gọn và sử dụng IC dán và linh kiện dán (SMD) thì nên sử dụng con IC ULN2303 hoặc ULN2803 thay thế (xem datasheet để biết cách sử dụng).

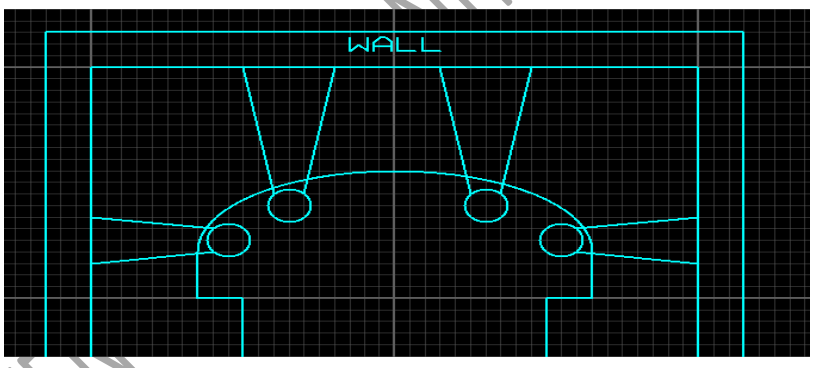
+ LED thu (Detector): Khi có ánh sáng hồng ngoại chiếu vào điện trở của LED thu sẽ giảm xuống dần tới một giá trị rất bé gần như nối tắt, và mức độ tùy vào cường độ ánh sáng hồng ngoại nó nhận được. Lưu ý LED thu được lắp ngược lại với LED phát: cực âm LED thu được nối trực tiếp lên nguồn dương, **cực dương được nối qua điện trở 10k xuống đất**. Ngõ ra Analog là giá trị điện áp đưa trực tiếp vào chân ADC của vi xử lí (xem hình 2.1c ở trên).

- Chú ý vì lý do trong ánh sáng mặt trời chứa rất nhiều tia hồng ngoại nên việc thiết kế đòi hỏi phải chống nhiễu tốt, khi cho micromouse chạy tránh môi trường ngoài trời hay có ánh sáng trực tiếp. Đồng thời sử dụng ngôn ngữ lập trình để giảm nhiễu cho LED hồng ngoại (chi tiết phần hướng dẫn lập trình).

Giá trị điện trở cho LED phát là 220R hoặc 330R, LED thu là 10k (hoặc từ 5 đến 50K)

Cách bố trí cảm biến cho micromouse:

+ Sử dụng 4 cặp thu phát hồng ngoại, mỗi cặp thu phát đặt cạnh nhau sao cho ánh sáng phản xạ từ tường và LED thu nhận được. 1 cặp trái, cặp phải và 2 cặp phía trước, hình minh họa như sau:



Hình 2.1d. Cách bố trí cảm biến hồng ngoại cho micrmouse.

Trong chương trình chúng ta sẽ dựa vào giá trị cảm biến hồng ngoại đọc về để xác định có hay không có bức tường. Đồng thời so sánh giá trị ADC đọc được từ cảm biến 2 bên để giúp micromouse chạy ở giữa mà không lệch sang 1 bên khi di chuyển.

Cách mắc cảm biến với Vi xử lí:

Các LED thu hồng ngoại đều phải nối vào ADC của vi xử lí một cách độc lập, mỗi một LED vào một chân ADC. Đối với LED phát ta cũng phải mắc vào chân vi xử lí một cách độc lập, **KHÔNG ĐƯỢC CHO các LED phát hồng ngoại phát cùng một lúc** (ngoại trừ 2 LED phía trước), mục đích là để chống nhiễu cho LED hồng ngoại (rất quan trọng).

2.2. Động cơ.

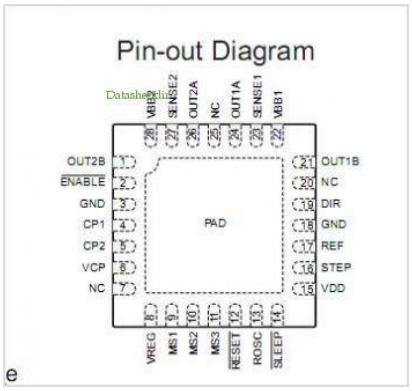
Người ta thường yêu cầu động cơ có tốc độ cao để có thể tham gia các cuộc thi. Do đó động cơ DC servo được sử dụng rất phổ biến, tuy nhiên đòi hỏi kỹ thuật lập trình rất khó. Động cơ DC được điều khiển PWM (điều khiển bằng điện áp) vị trí và tốc độ (theo logic mờ và 2 bộ PID kết hợp).

Vì lý dó khó khăn trong điều khiển DC với người mới bắt đầu nên Tôi khuyên các bạn sử dụng loại động cơ thay thế là động cơ bước, loại động cơ này thích hợp với những người mới nghiên cứu về micromouse mà chưa đòi hỏi yêu cầu cao cho việc đi thi (thực tế thì các cuộc thi MIcromouse của các trường ở Việt Nam chưa có ai cần phải dùng đến động cơ DC cả, do đó với động cơ bước bạn đã đủ sức để đi thi các cuộc thi trong các trường ở Việt nam). Sau khi các bạn đã thấu hiểu, lập trình Micromouse chạy tốt rồi và nếu muốn đi thi và đoạt giải chắc chắn thì mới tìm hiểu động cơ DC.

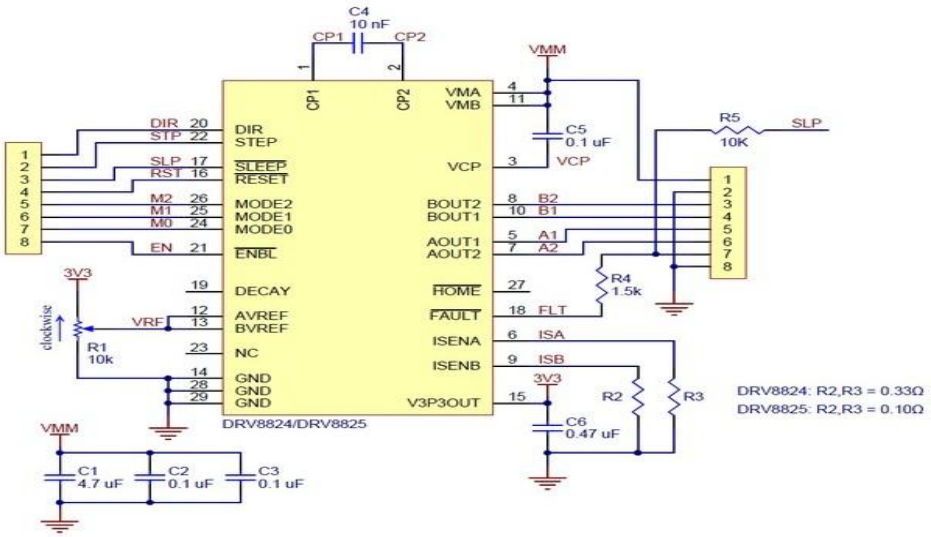
Hướng dẫn điều khiển động cơ bước:

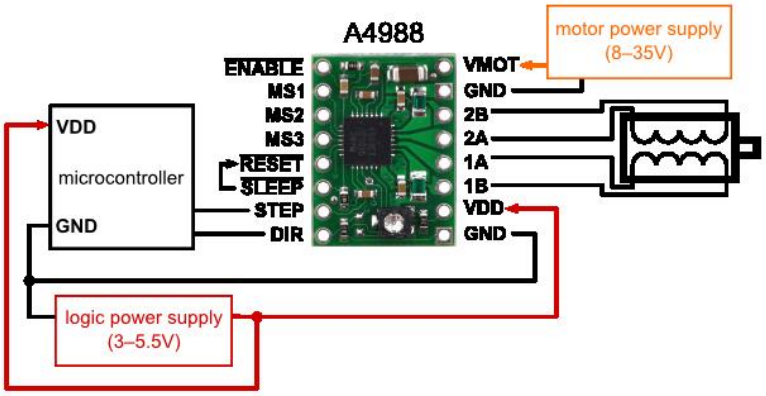
Đối với Micromouse là mạch công suất bé, do đó chúng ta chọn loại động cơ bước nhỏ và công suất bé, loại 4 hoặc 6 dây đều được (cả 2 đều sử dụng điều khiển theo phương pháp 4 dây) và nên sử dụng Module A4988 cho việc điều khiển. Các bạn hãy đi tìm mua một loại động cơ bước nhỏ thôi, loại 4 hoặc 6 dây nhé.

2.3. Hướng dẫn sử dụng Modull A4988



Hình 2.3a Modull A4988 mà sơ đồ chân chip A4988





Hình 2.3b. Sơ đồ của modull A4988 và cách sử dụng.

A4988 là chip do hãng Allegro sản xuất, Hình 2.3a là sơ đồ chân của IC 4988 chuyên dụng điều khiển động cơ bước loại đơn cực nam châm vĩnh cữu 4 dây. Hình 2.3b là sơ đồ chân của modull A4988 sau khi đã lắp ráp theo datasheet để điều khiển động cơ bước. Sử dụng điều khiển bao gồm:

+ MS1, MS2, MS3: Chọn bước điều khiển, có bảng chân trị như sau (mặc định là full step, xem bảng 3.0):

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Bảng 3.0: Bảng tùy chọn bước điều khiển

+ DIR: (direction) Chọn hướng quay của động cơ

+ STEP: chân vào xung clock, chu kì nhỏ nhất cho phép là 4uS

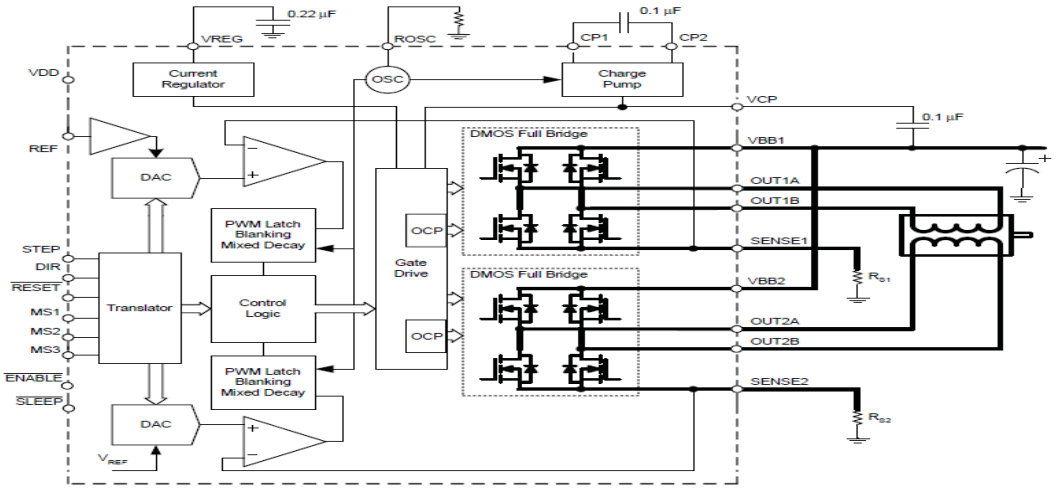
+ SLEEP & RESET: chân này thường không sử dụng. trong modull được nối chung với nhau và thả nổi. do cấu tạo mặc định chúng được đưa xuống 0 qua điện trở.

+ OUT 1A, 2A, 1B, 2B: chân ra đi vào cuộn dây động cơ, dòng cấp tối đa cho động cơ là 2A

+ ENABLE: chân cho phép ngõ ra hoạt động, chân này được kích lên 1 thì tất cả ngõ ra bị khóa. Mặc định đang là "0"

+ VMOT: Chân cấp nguồn cho động cơ: điện áp hoạt động từ 8 -> 35V DC

+ VDD: điện áp logic cấp cho A4988 từ 3 -> 5.5V DC



Hình 2.3c. Cấu tạo chi tiếp bên trong modull A4988

Modull có sẵn chức năng chống thấp áp và quá dòng, quá nhiệt. Nếu điện áp cấp cho động cơ dưới 5V modull sẽ tự động chuyển sang chế độ SLEEP tắt mọi hoạt động của modull cho đến khi điện áp cấp đủ điều kiện. Ngõ ra được kích bởi các mosfet có tốc độ cực cao đã có sẵn loại FAST DIODE bảo vệ bên trong.

Ở đây chúng ta sử dụng 3 chân nối trực tiếp vào vi xử lý bao gồm: ENABLE, STEP và DIR, các chân còn lại chúng ta nên thiết kế trên 1 board tùy chọn có công tắc gạt.

Chú ý khi modull đang hoạt động không được tháo và lắp động cơ trực tiếp vào cho thể sẽ dẫn đến chết FET bên trong chip. Cần ngắt điện cấp cho FET trước khi tháo hay lắp động cơ vào modull.

Đọc kỹ datasheet của Module này trước khi sử dụng. Có thể mua Module này ở HShop.vn hoặc thienminhelectronic.

Lưu ý đối với động cơ Bước loại 6 dây: Các bạn hãy dùng đồng hồ kim đo (thang đo điện trở) để xác định 2 cuộn dây và xác định dây giữa (Nếu không biết các thì lên mạng có bài chỉ). Sau đó bỏ 2 dây giữa của 2 cuộn dây đi ta còn lại 4 dây, mắc 4 dây như datasheet của 4988 để điều khiển.

2.4. Hướng dẫn thiết kế mê cung.

Ở phần trước tôi đã ghi thông tin cụ thể của một mê cung có kích thước như thế nào rồi, do đó các bạn hãy thiết kế mê cung chuẩn như thế.

Một mẹo nhỏ về thiết kế mê cung cho các bạn tự làm tại nhà Mê cung thử nghiệm:

- Tường của mê cung các bạn có thể chọn tấm Format loại dày nhất là 10 li hoặc loại 8 li để tiết kiệm chi phí thay vì làm bằng gỗ, khi đi mua các bạn nói với người bán cắt cho các bạn mỗi tấm có kích thước như mong muốn luôn, về đỡ phải cắt lại.
- Giả sử các bạn chọn loại Format dày 10mm thì: Đối với mê cung chuẩn thì các bạn sẽ cắt các miếng nhỏ làm tường tại mỗi ô có kích thước là 5x17cm. Với mê cung 16*16 các bạn cần ít nhất khoảng 100 tấm như vậy. Tuy nhiên tự test ở nhà thì chúng ta chỉ cần một mê cung có kích thước khoảng 5*4 là vừa đủ.
- Cách làm mê cung như sau: sau khi cắt các tấm nhỏ đó ra các bạn lưu ý, tại mỗi điểm góc giao nhau giữa các bức tường có một khoảng trống 1cm, tại đó chúng ta sẽ cắm 1 cái trụ có kích thước tương ứng để ghép các tầm làm tường vào với nhau. Tấm nền chúng ta sử dụng gỗ ép hoặc tấm giấy ép chống thấm nước để tiết kiệm chi phí (Các bạn vào Google gõ tìm "**design maze in Micromouse**" là sẽ có các

bài hướng dẫn, các bạn quan sát thật kĩ thì sẽ hình dung ra cách để tự làm mê cung test tại nhà).

Những thông tin về bức tường và mê cung đều rất quan trọng để tính toán trong chương trình, chúng ta cần tính số xung (sử dụng Encoder hoặc cảm biến Hall-E đối với động cơ DC và tính xung đối với động cơ bước), mục đích tính toán làm thế nào để biết micromouse di chuyển đi hết một ô, rẽ đúng 45 độ hay 90 độ.... Như vậy ở bị trí ban đầu đặt micromouse ở giữa ô đầu tiên, hoặc có một mẹo nhỏ là để tránh sai lệch trong ô đầu tiên, các bạn hãy tìm cách đặt Micromouse sao cho phần đuôi của Micromouse chạm vào bức tường của ô đầu tiên, sau đó tính toán đủ số xung từ vị trí đó đến vị trí ô thứ 2 thì Micromouse nằm ở giữa ô thứ 2, như vậy sẽ tránh được trường hợp sai sót do vị trí đặt đầu tiên, chú ý là nó cũng khá quan trọng nhé!

2.5. Bánh xe và tính toán bước.

Đối với bánh xe gắn trên động cơ cũng cần phải tính toán cẩn thận, tính toán chu vi bánh xe để biết quãng đường đi được. Giải pháp là chúng ta sẽ kiểm tra thử trong một ô vuông, các bạn hãy vẽ một ô vuông với các đường chéo đầy đủ, sau đó đặt Micromouse vào và Kiểm tra các góc quay cho thật chính xác, càng chính xác càng tốt. Thực tế chúng ta có 2 cách để tính góc quay cho Micromouse, đơn giản nhất là tính toán xung của động cơ, cách thứ 2 là sử dụng cảm biến góc quay trong các Module cảm biến từ trường ….

Một trong những hướng ưa chuộng hiện nay là sử dụng cảm biến góc quay, các Module đó rất phổ biến bên ngoài, ví dụ M6050 … có rất nhiều tài liệu trên website hướng dẫn về Module M6050, do đó hãy tìm hiểu lập trình cho nó, thật sự thì việc lập trình cho Module này khá khó vì cần có bộ lọc Kallman (hoặc bộ lọc bổ sung), và tính toán góc quay quanh trục Z (trục đứng) rất khó khăn.

2.6. Nguồn

Năng lượng cấp cho động cơ và chip hoạt động cũng rất quang trọng. Năng lượng phải đảm bảo micromouse chạy hoàn tất trong thời gian thi đấu. Đối với động cơ DC thì chỉ cần nguồn năng lượng bé vì công suất tiêu thụ khá thấp, tuy nhiên với động cơ bước đòi hỏi công suất nguồn phải cao hơn.

Để đảm bảo an toàn và nguồn năng lượng đảm bảo cho Micromouse các bạn nên chọn sử dụng pin LIPO:

Hình 2.7d. pin LIPO

Loại pin LIPO là loại có cấu tạo khác so với pin Lithium bình thường, dòng cấp rất lớn và chống cháy nổ tốt hơn, tuy nhiên giá cả khá cao so với pin lithium. Loại trên hình là loại Wild Scorpion 7.4V 2200mAh 35C, thực tế chúng ta không sử dụng hết công suất của nó.

Sử dụng trực tiếp điện áp 7.4V nối vào làm nguồn cấp cho động cơ bước.

2.7. Các loại vi xử lí nên sử dụng.

Tùy vào nhu cầu hay sự yêu thích của mỗi người mà các bạn có thể chọn cho mình một loại vi xử lí phù hợp như: dsPIC, AVR,ARM hoặc Arduino…

Thực tế trong các cuộc thi hầu hết các cá nhân tham gia đều sử dụng ARM dòng ATmega324. Vì dòng chip này có tốc độ xử lí rất cao (>80MHz), bộ nhớ lớn và nhiều tài liệu hướng dẫn. Tuy nhiên chúng ta có thể chọn loại khác, nhiều bạn lại chọn Arduino (thực tế bên trong nó là chip ARM), nhưng chúng ta nên sử dụng ARM riêng để lập trình chứ không nên chọn Arduino.

Một lưu ý quan trọng là vi xử lí phải đủ bộ nhớ chương trình để thực hiện thuật toán, Chúng ta có thể tính toán đối với bộ nhớ cần thiết như sau:

Đối với mê cung 16*16 chúng ta cần ít nhất:

- 256 Byte lưu thông tin thuật toán Flood-Fill.
- 256 Byte lưu thông tin bức tường của mê cung.
- 512 Byte làm Stack để thực hiện thuật toán.

Đó là những con số bắt buộc, chúng ta cần ít nhất như thế, tuy nhiên còn rất nhiều thứ khác phải xử lí, đo đó trước khi chọn vi xử lí để lập trình chúng ta nên lưu ý chọn loại cho đủ bộ nhớ cần thiết. Chương trình rất dài do đó các dòng vi xử lí thấp chắc chắn sẽ không đủ bộ nhớ và tốc độ xử lí nếu muốn đem Micromouse đi thi :D.

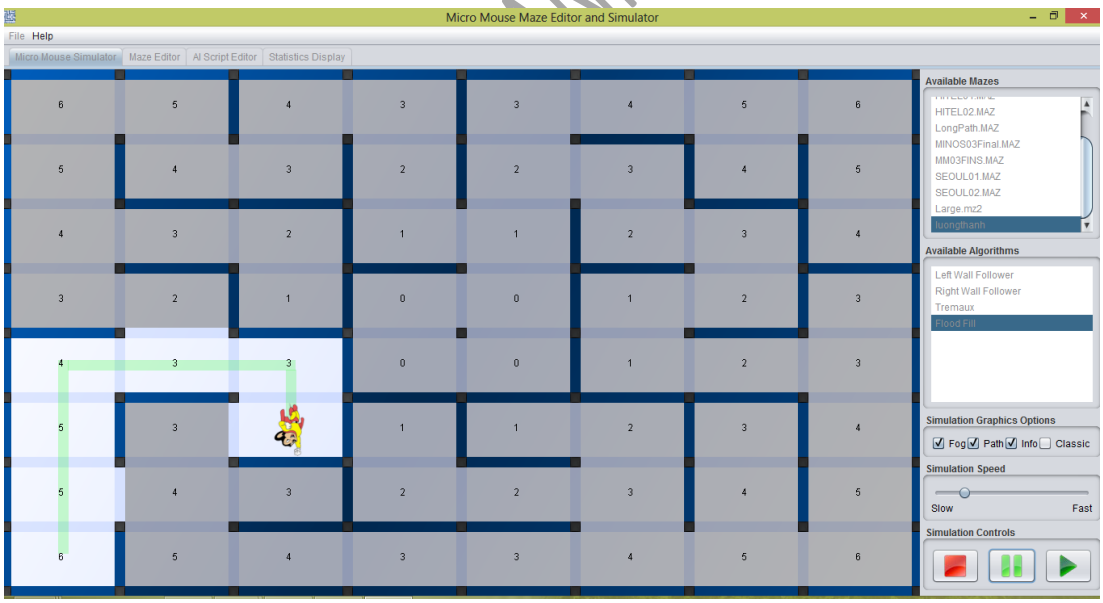
CHƯƠNG 3 LẬP TRÌNH MICROMOUSE

3.1. Làm gì trước lập trình cho Robot micromouse?

3.1.1. Lập trình mô phỏng trên máy tính.

Trong thực tế, để một micromouse chạy được không hề đơn giản, thậm chí một quá trình nghiên cứu dài hạn trên 1 năm đối với người mới bắt đầu (nếu tự tìm hiểu), do đó, việc nghiên cứu phải bắt đầu từ mô phỏng chúng ta mới có thể hiểu rõ được bản chất và nguyên lý, những gì xảy ra khi micromouse chạy trong mê cung. Một trong những chương trình ứng dụng phổ biến nhất đó là java, chương trình cho phép chúng ta lập trình tương tự như ngôn ngữ C++, có thể tạo ra ma trận, tạo ra chuột và viết chương trình cho micromouse chạy trên nền java. Riêng đối với việc nghiên cứu này cũng mất một thời gian khá dài đối với người mới bắt đầu, do đó nên chọn những ngôn ngữ nào gần nhất với ngôn ngữ vi xử lý chọn cho micromouse, sau khi nó đã chạy trên máy tính rồi thì bạn rõ ràng đã hiểu bản chất của nó, do đó việc đưa vào vi xử lí tương đối dễ dàng.

Hình ảnh sau đây mô tả một chương trình micromouse được lập trình từ chương trình java scrip:



Hình 3.1. Mô phỏng micromouse bằng JAVA

Chương trình cho phép ta hình dung được những gì sảy ra khi micromouse đang di chuyển trong mê cung.

Ngoài ra chúng ta có thể tìm hiểu thêm về lập trình mô phỏng trên Matlab hoặc visual studio của Microsoft.

Và một trong những ứng dụng quen thuộc là Excell, các bạn hãy tận dụng nó như một mê cung ảo trên máy tính, điền các số để tính toán vào đó, chạy trên nền CMD mô phỏng (khá rắc rối).

3.1.2. Lập trình ngôn ngữ C cho vi xử lý.

Mỗi loại vi xử lý có một chương trình và ngôn ngữ riêng, có thể sử dụng ASM hoặc C, tuy nhiên về cơ bản chúng ta đều có thể sử dụng ngôn ngữ C để lập trình và chúng gần như tương tự nhau không có nhiều khác biệt. Phần lớn các lập trình viên micromouse hiện nay đều sử dụng dòng vi xử lý ATMEGA32 32bit của ATMEL để lập trình.

Chú ý khi lập trình C cho micromouse cần những kiến thức cơ bản liên quan đến mảng 1 chiều và nhiều chiều và stack trong C++.

3.2. Các thuật toán sử dụng trong Micromouse.

Có rất nhiều thuật toán để lập trình, tuy nhiên phổ biến thì có xử lí ảnh, Flood-Fill hoặc tọa độ. Hiện nay thuật toán Flood-Fill được sử dụng nhiều nhất trên hầu hết các Micromouse, nó dễ lập trình, tương đối tối ưu và hiệu quả cao.

3.2.1. Thuật toán Flood Fill

Flood Fill là thuật toán được sáng tạo dựa trên thuật toán tô màu trong các trò chơi truyền thống có tên Bellmen-Ford, nó được áp dụng vào Micrmouse đầu tiên năm 2009 bởi MR. Lee một Sinh Viên Đài Loan lúc bấy giờ và được mô phỏng bởi phần mềm JAVA, sau đó nó được rất nhiều người nghiên cứu và tối ưu lại. Hiện tại hầu hết các nhà lập trình đều sử dụng thuật toán này cho các cuộc thi micromouse của họ.

Hãy tưởng tượng đơn giản như sau: Một mê cung được chia ra làm nhiều ô và thấp dần từ ngoài vào trong cùng mê cung, khi đổ nước vào từ ô ngoài cùng, nước sẽ chảy đến ô thấp hơn nó và dần đi đến trung tâm của mê cung. Như vậy thuật toán này được lấy ý tưởng từ đó. Ta sẽ điền các giá trị tại mỗi ô tượng trưng cho khoảng cách ngắn nhất từ nó đến ô trung tâm (giả sử không có bức tường nào trong mê cung). Thuật toán được mô tả trong lập trình như sau:

- **Tạo một mảng chứa các giá trị khoảng cách của ô hiện tại với ô trung tâm, mỗi 1 ô là 1 byte dữ liệu lưu trữ giá trị đó**
- **Đặt giá trị nhỏ nhất là 0 cho ô mà chúng ta muốn đi đến đó.**
- **Cho chuột thăm dò đi qua tất cả những ô đó, nó sẽ kiểm tra giá trị các ô xung quanh mà nó đi qua để gán giá trị sao cho phải có "ít nhất một ô bên cạnh nó có giá trị nhỏ hơn nó một đơn vị".**

- **Thực hiện lặp đi lặp lại các bước kiểm tra và điền giá trị cho các ô cho đến khi không còn ô nào cần cập nhất giá trị mới thì dừng lại (thõa điều kiện ở trên).**

3.2.2. Một số định nghĩa sử dụng trong giải thuật toán:

+ Ô mở: là ô không có bức tường nào ngăn giữ 2 ô với nhau.

+ Ô hàng xóm mở: gồm tất cả những ô xung quanh không có bức tường ngăn với ô hiện tại đang xét, nơi mà con trỏ stack đang đặt tại đó.

+ kí hiệu "**!=**" là lệnh so sánh "Khác nhau" trong C.

+ **Stack** là mảng theo quý tắc vào trước ra sau (**FILO** (fist in - last out), giá trị được đưa vào thứ 2 sẽ lấp vào chỗ của giá trị đưa vào trước đó, đẩy giá trị trước đó xuống dưới 1 ô. Đỉnh stack là phần tử thứ 0, tiếp theo là 1 cho đến cuối cùng của stack. Cần tạo 1 biến làm con trỏ cho stack để trỏ đến vị trí trong mảng stack khi lấy và đọc giá trị từ mảng stack.

3.3. Phân tích thuật toán flood fill

Bài viết này được tôi nghiên cứu từ nhiều tài liệu khác nhau và được viết lại tối ưu một số ưu điểm giúp rút ngắn thời gian tính toán.

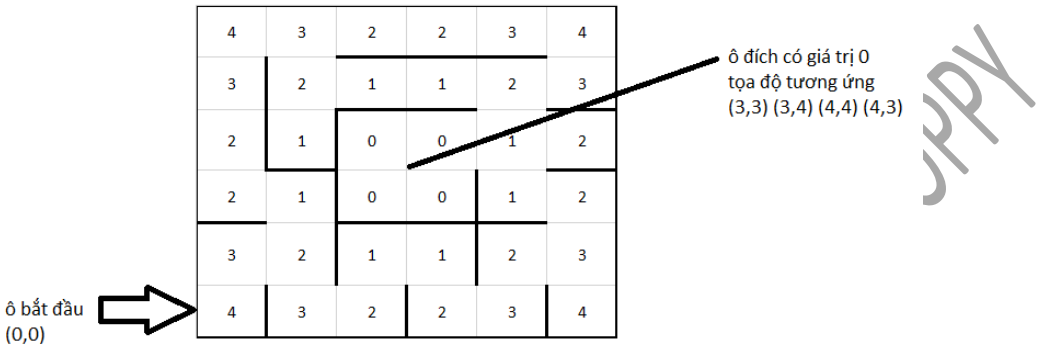
Giả sử ta cần giải quyết một mê cung gồm 6*6=36 ô như hình 3.3

Ban đầu ta tạo một mảng minh họa như hình 3.3, Ô trung tâm là 4 ô có giá trị đặt trước là 0, ô bắt đầu là ô ở góc bên trái phía dưới, ô bắt đầu có giá trị lớn nhất tương ứng với khoảng cách từ ô đích đến ô bắt đầu, giả sử một mảng được khởi tạo mô phỏng như hình 3.3, chuột sẽ ở vị trí bắt đầu và luôn luôn hướng về phía Bắc, tọa độ đường đánh dấu theo (hàng,cột). Những con số sau đây tôi sẽ gọi nó là giá trị Flood (nó được khởi tạo trước như hình, trong quá trình di chuyển các giá trị Flood này sẽ thay đổi sao cho nó thõa điều kiện). Bài ví dụ sau tôi lấy từ một ví dụ của tác giả **George Law.**

- **Hãy lưu ý là phần này chỉ mô tả quá trình xảy ra khi thực hiện thuật toán Flood-Fill như thế nào trong khi Micromouse di chuyển để các bạn hiểu rõ bản chất của nó, chứ trên thực tế khi viết chương trình, các bạn không cần quan tâm đến các giá trị Flood thay đổi là bao nhiêu. Hầu hết các bạn mới bắt đầu tìm hiểu thuật toán Flood-Fill thì khi đọc phần mô tả thuật toán Flood-Fill sau sẽ chẳng hiểu gì cả, do đó các bạn nên đọc phần hướng dẫn lập trình bằng ngôn ngữ C ở chương sau rồi quay lại chương này để hình dung quá trình thực hiện thuật toán và lập trình nó liên quan đến nhau như thế nào, các bạn nên đọc phần lập trình cho Stack và phần mô tả sau đây song**

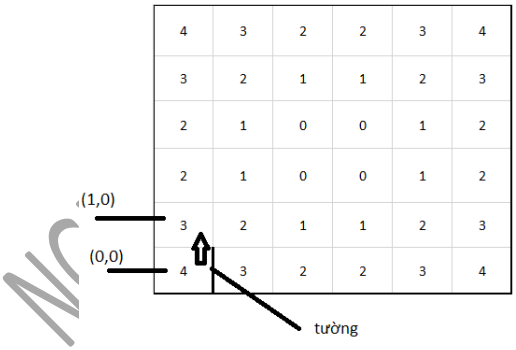
song với nhau, vừa đọc vừa nhìn phần mô tả thuật toán các bạn sẽ dễ hiểu hơn rất nhiều.

- **Thật ra thuật toán này không khó nhưng mới bắt đầu sẽ thấy khó khăn, các bạn hãy kiên trì đọc kĩ hướng dẫn của tôi từ đầu đến cuối, đừng bỏ chi tiết nào hết vì nó không thừa đâu nhé, chỉ có thiếu thôi.**



Hình 3.3. Mảng mê cung cơ bản ban đầu

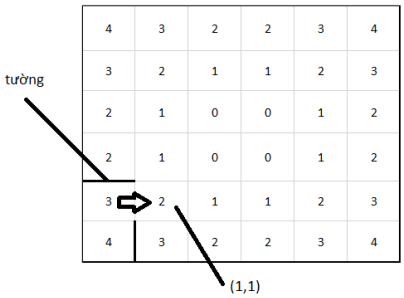
Ngay ban đầu chuột không hề biết bất kì bức tường nào, do đó, khi bắt đầu điều đầu tiên là cần phải cập nhật bức tường tại ô nó đang đứng trước, cách cập nhật thông tin bức tường được giải thích chi tiết phần chương trình. Sau đây là thuật toán Flood Fill thực hiện như sau: xem thêm hình 3.4a.

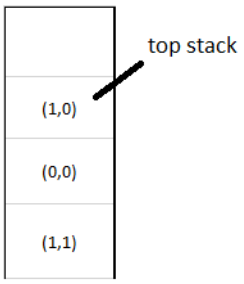


Hình 3.4a: Flood

- Lần đầu tiên khi bắt đầu di chuyển chỉ cần cho chuột chạy thẳng tới một ô, chủ động viết mã cập nhật hướng và giá trị Flood mới cho Micromouse. Không cần thực hiện thuật toán Flood cho ô đầu tiên. Khi đã di chuyển tới một ô thì bắt đầu thực hiện theo các bước như sau

Khi chuột đang ở vị trí (1,0)





Hình 3.5a: Flood

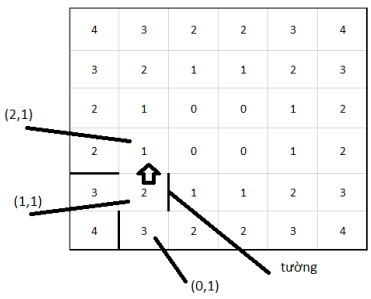
Hình 3.5b: Stack

- Tại tọa độ (1,0) đẩy giá trị ô hiện tại và hàng xóm mở với nó vào stack kiểm tra gồm ô (1,1) và ô (0,0) và (1,0) xem hình 3.5b. Mỗi lần đẩy một giá trị vào stack thì tăng con trỏ stack lên một đơn vị (xem hướng dẫn lập trình C ở chương tiếp theo).
- Kiểm tra giá trị nhỏ nhất ô hàng xóm mở với ô hiện tại so sánh với giá trị Flood của ô hiện tại.

Ta thấy giá trị nhỏ nhất ô hiện tại là ô (1,1), thực hiện kiểm tra: (1,0)=3, (1,1) = 2, vậy (1,0)=(1,1)+1 = 2+1 = 3, do đó không cần cập nhật giá trị mới cho ô hiện tại nữa (vì điều kiện là bất kì ô nào cũng phải có ít nhất một ô hàng xóm mở với nó nhỏ hơn nó một đơn vị, lưu ý xem lại định nghĩa ô hàng xóm mở để dễ hình dung). Giảm con trỏ stack xuống 1 đơn vị, tiếp tục kiểm tra đến ô (0,0), thực hiện tương tự ta thấy (0,0) = 4 = (1,0) + 1 = 3+1, do đó không cần cập nhất giá trị ô (0,0) nữa. Giảm con trỏ Stack xuống 1 đơn vị, lúc này con trỏ stack là 0, do đó kiểm tra stack đã rỗng, thực hiện di chuyển đến ô hàng xóm mở nhỏ nhất là ô (1,1).

- Quá trình trên thực hiện tương tự cho ô tiếp theo (1,1).

Khi chuột đã đến ô (1,1) hình 3.6



(1,0)
(0,1)
(2,1)
(1,1)

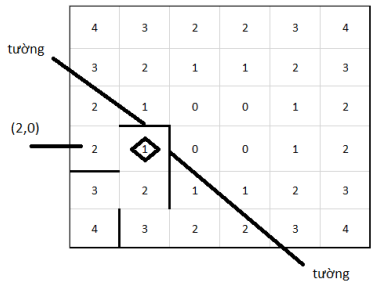
Hình 3.6: Flood

Hình 3.6a: Stack

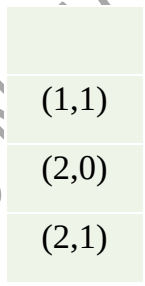
Khi chuột đang ở vị trí (1,1) thực hiện flood fill như sau:

- Tại vị trí (1,1) thực hiện tương tự như bước 1, đẩy giá trị (1,1), (2,1), (0,1), (1,0) và stack thực hiện kiểm tra. Lấy từng giá trị từ đỉnh stack xuống để kiểm tra.
- Ta thấy (1,0) = 3 = (1,1) + 1 = 2+1 = 3, không cần cập nhật ô (1,0).
- Giảm stack xuống 1 đơn vị, kiểm tra (0,1) = 3 = (0,2) =2+1, không cần cập nhật ô (0,1).
- Tiếp tục ô (2,1) = 1 = (2,2) +1 = 0+1, không cần cập nhật ô (2,1) nữa.
- Còn lại ô (1,1) trong stack thực hiện kiểm tra cuối cùng (1,1) = 2 = (2,1) + 1 = 1+1 do đó không cần cập nhật nữa, thực hiện di chuyển đến ô (2,1).

Khi chuột đến vị trí (2,1) hình 3.7



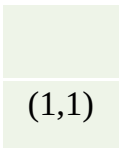
Hình 3.7: Flood

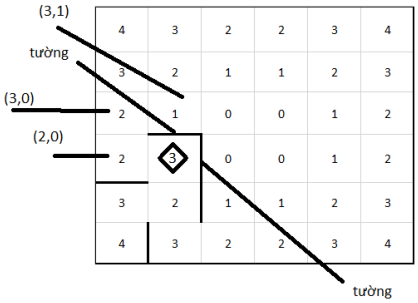


Hình 3.7a: Stack

Tại vị trí ô (2,1) = 1. Tiếp tục thực hiện flood fill:

- Tại ô (2,1) Tiếp tục lặp lại thuật toán tương tự. Đẩy giá trị (2,1) vào stack tiếp tục đẩy các giá trị còn lại (2,0), (1,1) vào stack kiểm tra.
- Lấy giá trị từ đỉnh stack là ô (1,1) thực hiện kiểm tra, ô hàng xóm mở nhỏ nhất là ô (1,1) ta thấy (1,1) = 2 = (2,1) +1 = 1+1, không cần cập nhật nữa.
- Lấy giá trị (2,0) ra kiểm tra với ô hàng xóm nhỏ nhất tương tự ô (2,0) = (2,1) + 1, không cần cập nhật.
- Lấy giá trị ô (2,1) cuối cùng trong stack ra kiểm tra, ô hàng xóm nhỏ nhất với ô (2,1) có giá trị là 2, ta thấy (2,1) != 2+1, do đó cần cập nhật giá trị mới cho ô (2,1) như sau:
- (2,1) = 2 + 1 = 3
- Lưu giá trị 3 vào ô (2,1). Đẩy các giá trị hàng xóm mở với ô (2,1) vào stack kiểm tra tiếp.





(2,0)

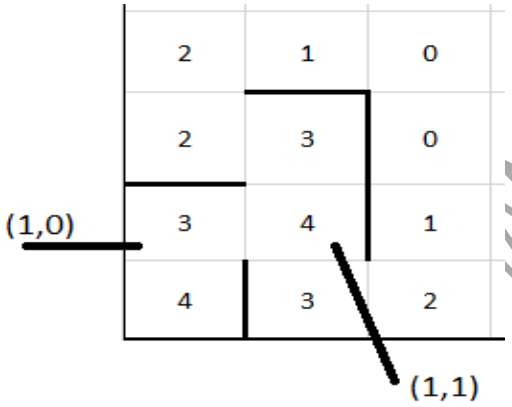
(2,1)

Hình 3.7b: Flood

Hình 3.7c: stack

- Làm tương tự, stack lúc này như hình 4.1a. Lấy giá trị từ đỉnh stack (1,1) kiểm tra với ô hàng xóm mở nhỏ nhất có giá trị là 3. ta thấy 2!= 3+1, do đó cần cập nhật giá trị mới: (1,1) = 3+1 =4.

Lưu giá trị này vào ô (1,1), đẩy các giá trị hàng xóm mở với ô (1,1) vào stack kiểm tra gồm ô (0,1), (1,0) và (2,1). Lúc này stack như hình 3.7e.



(2,1)

(1,0)

(0,1)

(2,0)

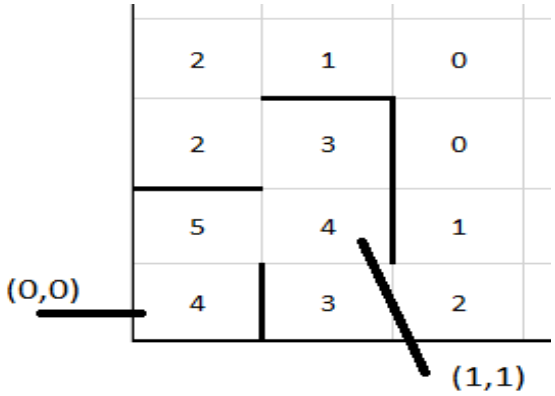
(2,1)

Hình 3.7d: Flood

Hình 3.7e: Stack

- Quan sát Stack hình 3.7e, lấy giá trị từ đỉnh stack (2,1) kiểm tra thấy (2,1) = 3 = 2+1 do đó không cần cập nhật ô (2,1).
- Kiểm tra tiếp (1,0) = 3, ô nhỏ nhất mở có giá trị là 4, do đó cần cập nhật giá trị mới cho ô (1,0): (1,0) = 4+1 = 5
- Lưu giá trị này vào ô (1,0), đẩy ô hàng xóm mở của nó vào stack kiểm tra gồm (0,0) và (1,1).
- Lúc này Stack sẽ như hình 3.7g

(1,1)

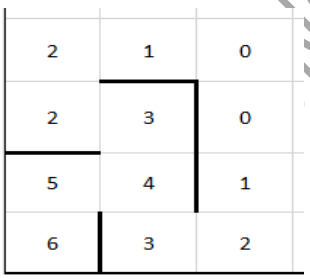


Hình 3.7f: Flood

(0,0)
(0,1)
(2,0)
(2,1)

Hình 3.7g: Stack

- Lấy ô (1,1) kiểm tra với giá trị hàng xóm nhỏ nhất ta thấy (1,1) = 4 = (0,1) + 1 = 3+1, do đó không cần cập nhật nữa.
- Tiếp tục kiểm tra giá trị ô (0,0) ta thấy 4!= 5+1, do đó cập nhật giá trị mới cho ô (0,0): (0,0) = 5+1 = 6
- Lưu giá trị này vào ô (0,0), đẩy các ô hàng xóm mở với nó là ô (1,0) kiểm tra.
- Ta thấy (1,0) = 5 = (1,1) + 1 = 4+1, do đó không cần cập nhật giá trị mới cho ô (1,0) nữa.
- Lúc này Stack chỉ còn các giá trị như hình 3.8b. Thực hiện tương tự.



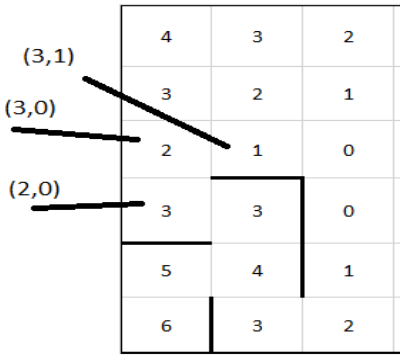
Hình 3.8a: Flood

(2,0)
(2,1)

Hình 3.8b: Stack

- Lấy giá trị tử đỉnh stack ra kiểm tra (2,0) ta thấy
- (2,0) != 2+1 =3, do đó cần cập nhật giá trị mới cho ô này: (2,0) = 2+1 = 3, lưu vào ô (2,0), các ô hàng xóm mở với ô này vào stack để kiểm tra gồm (3,0) và (2,1). Lúc này stack sẽ như hình 3.8f.

(2,1)



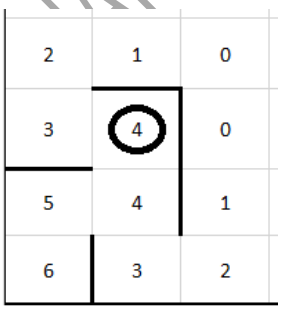
(3,0)

(2,1)

Hình 3.8c: Flood

Hình 3.8d: Stack

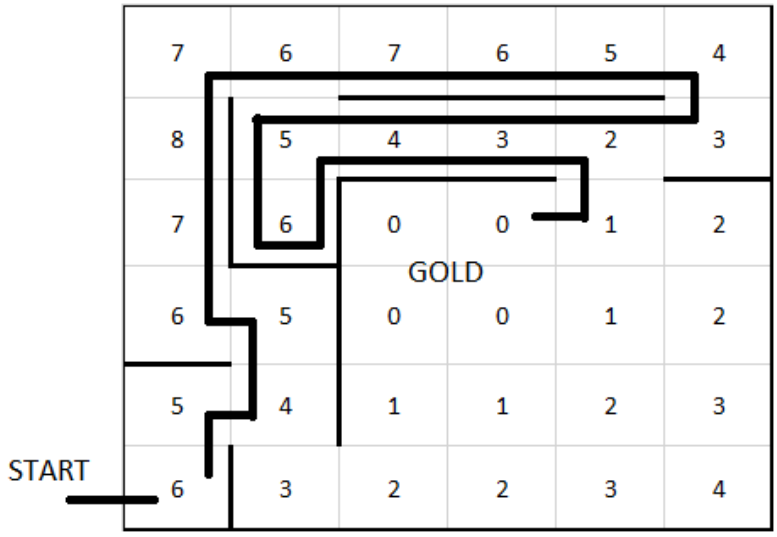
- Lấy giá trị từ đỉnh stack ra kiểm tra ta thấy:
- (2,1) = 3 != 3+1, do đó cập nhật giá trị mới
- (2,1) = 3+1 =4, lưu giá trị 4 vào ô (2,1)
- Đẩy các giá trị hàng xóm vào stack để kiểm tra gồm (1,1) và (2,0). Thực hiện kiểm tra tương tự ta thấy tất cả các ô này đã đảm bảo điều kiện tại mỗi ô có ít nhất một ô hàng xóm mở với nó có giá trị nhỏ hơn nó một đơn vị.
- Lúc này stack rỗng và thoát vòng lặp FloodFill. Các giá trị Flood sẽ như hình 3.8e.



Hình 3.8e: Flood

- lặp lại thuật toán tương tự như bước đầu tiên ta được lần đâu tiên chuột sẽ đi về đích xem hình 3.9 (nếu ưu tiên chạy thẳng trước)

- Lưu ý có một số vị trí chuột sẽ kiểm tra có 2 giá trị hàng xóm bằng nhau cùng nhỏ hơn 1 so với ô hiện tại, tức chuột phải chọn 1 hướng đi lúc đó tùy vào người lập trình, có thể chọn đi thẳng hoặc rẽ. Tất nhiên việc rẽ hay đi thẳng sẽ dẫn đến những giá trị flood fill trong lúc di chuyển khác nhau nhưng kết quả cuối cùng thì tương tự nhau. Nếu các con chuột lập trình cùng thuật toán này và cùng mức ưu tiên rẽ giống nhau thì khi di chuyển trong mê cung chúng có đường đi giống nhau.



Hình 3.9: kết quả lần di chuyển đầu tiên

Hình 3.9 trình bày kết quả trong lần di chuyển đầu tiên sử dụng thuật toán Flood Fill và giá trị Flood được điền trong lần đầu tiên khi đến đích, ở đây ưu tiên đi thẳng trước. Lần đầu tiên chạy vẫn còn những bức tường chưa được cập nhật nơi mà con chuột vẫn chưa đi qua, do đó có thể vẫn còn con đường khác ngắn hơn, chuột cần phải đi đến những ô còn lại để tìm đường ngắn hơn.

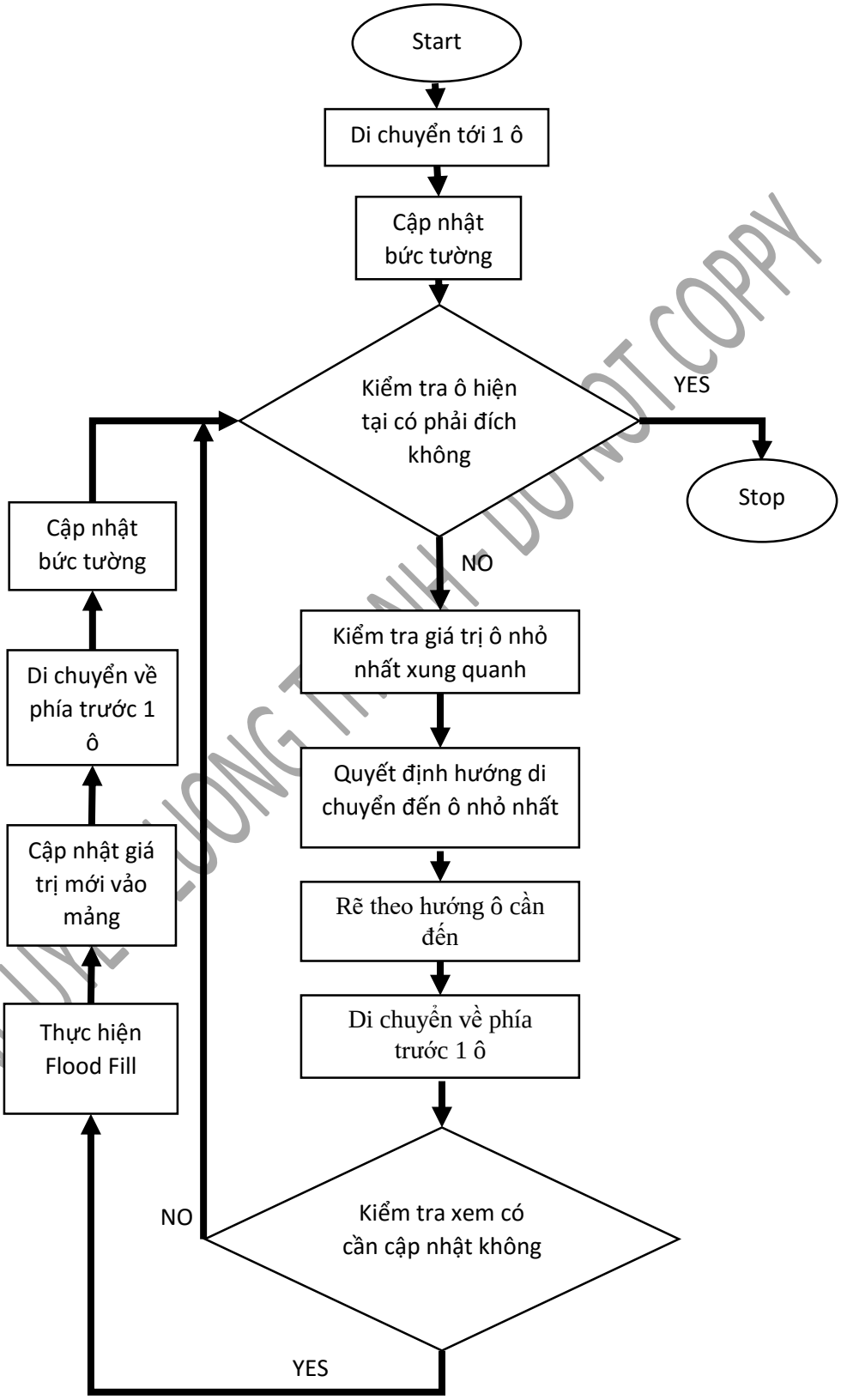
Sau khi đến đích, tạo một mảng mới như sau:

```
mazeflood[36]={
        0, 1, 2, 3, 4, 5,
        1, 2, 3, 4, 5, 6,
        2, 3, 4, 5, 6, 7,
        3, 4, 5, 6, 7, 8,
        4, 5, 6, 7, 8, 9,
        5, 6, 7, 8, 9, 10
        };
```

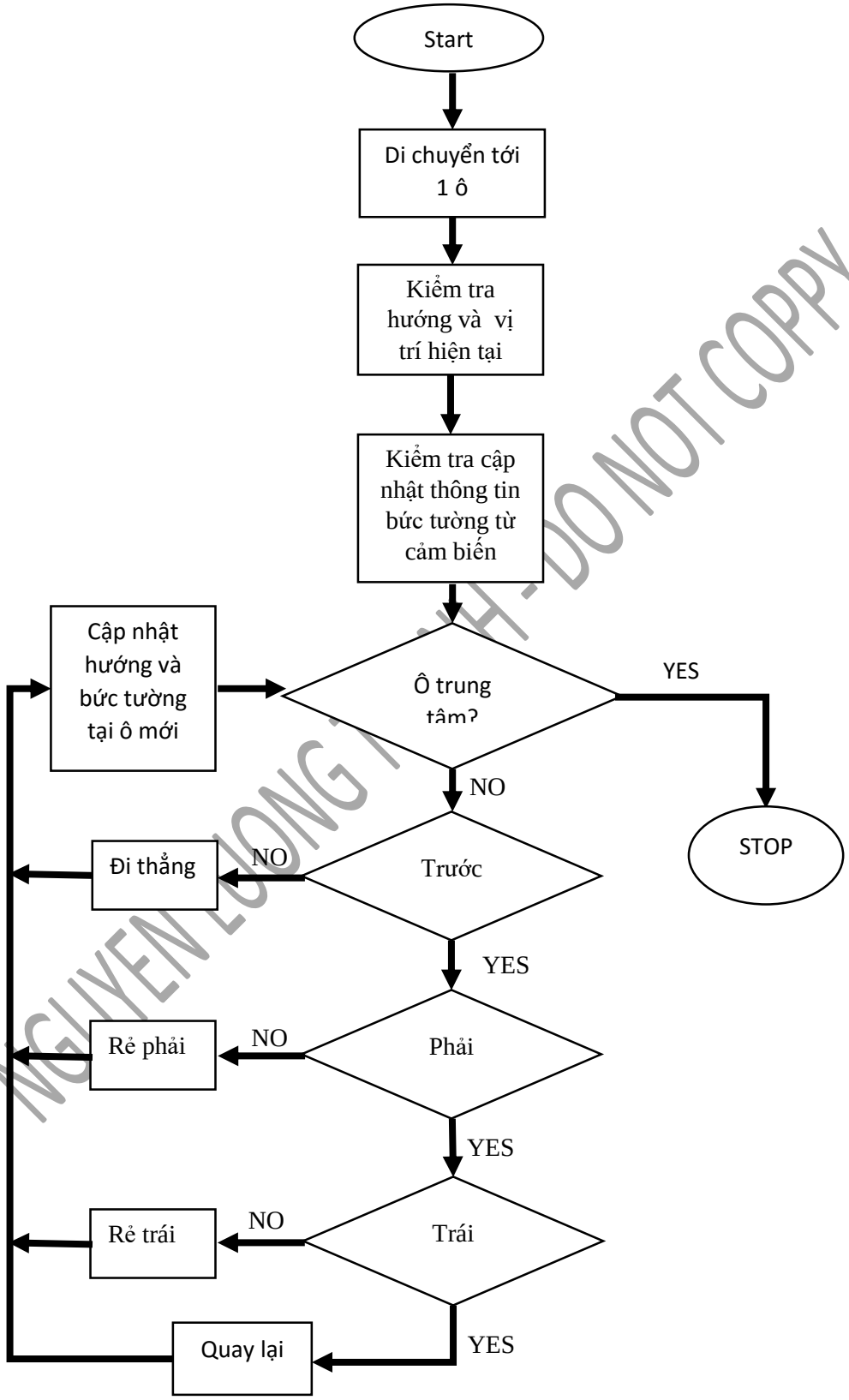
Lúc này thuật toán được thực hiện lại từ đầu, xem ô "đích" ban đầu là ô "bắt đầu", ô bắt đầu là ô đích. Những ô đã đi qua đã có thông tin bức tường nên chắc chắn Micromouse sẽ đi theo đường khác để tìm đường về ô khởi đầu (vì những ô nó chưa đi qua và chưa có thông tin nào thì coi như không có bức tường), sau khi lặp lại khoảng 2 đến 3 lần thì chúng ta sẽ tìm ra đường ngắn nhất, đường ngắn nhất này là các số đánh đánh số từ ô khởi đầu đến ô đích trong thuật toán Flood-Fill.

3.4. Lưu đồ giải thuật tham khảo.

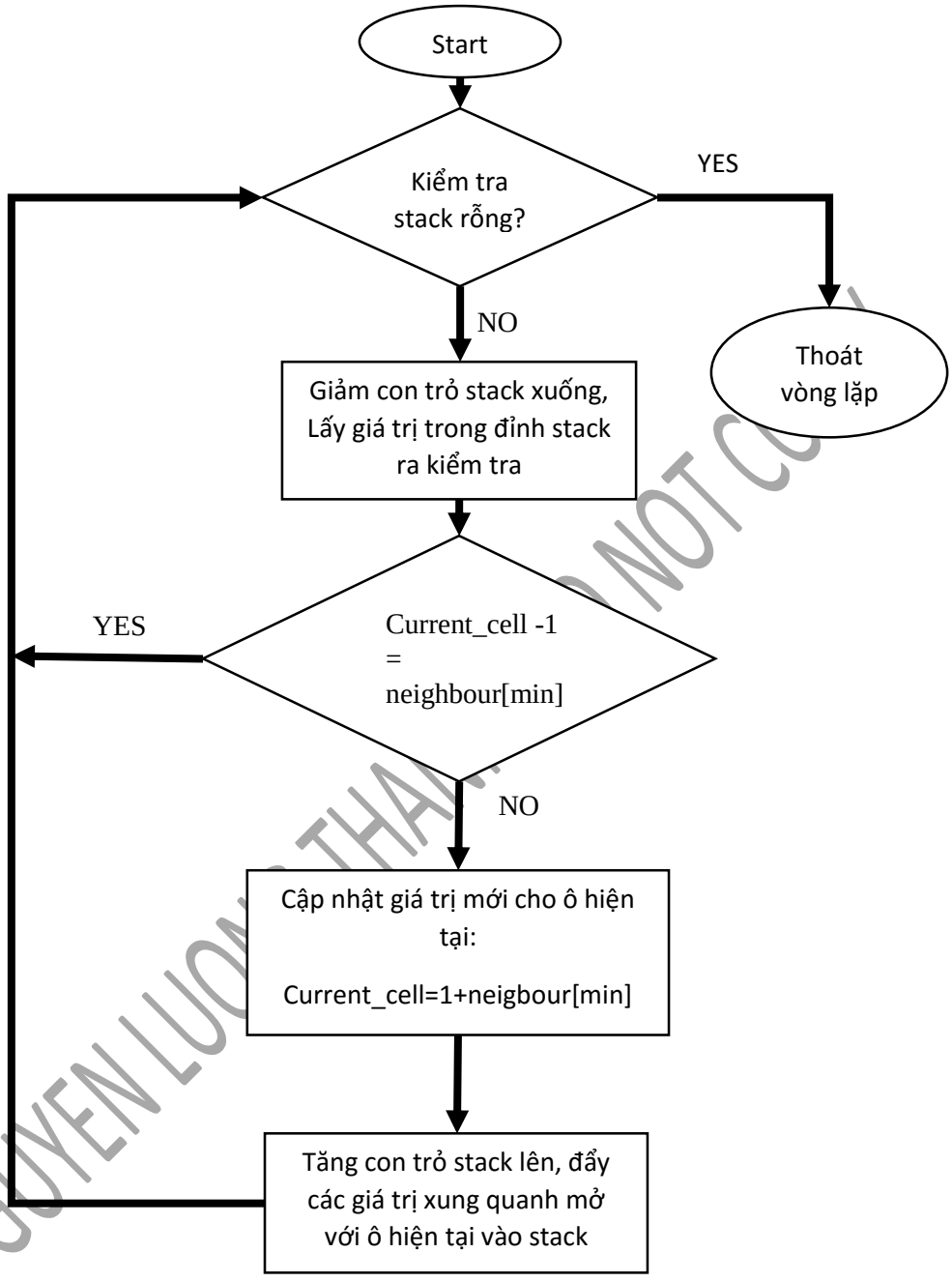
3.3.1. Lưu đồ tổng quát.



3.3.2. Lưu đồ giả thuật chọn hướng đi. Ưu tiên đi thẳng trước.



3.3.3. Lưu đồ thuật toán Flood Fill.



Ghi chú:

- Current_cell: là vị trí hiện tại của Micromouse
- neighbour[min] là ô hàng xóm mở có giá trị nhỏ nhất.

HƯỚNG DẪN LẬP TRÌNH BẰNG NGÔN NGỮ C

Ở phần hướng dẫn sau đây tôi sẽ hướng dẫn ngôn ngữ C từ phần mềm hỗ trợ PIC C COMPLIER của microchip và lập trình cho PIC 18F4550, thạch anh sử dụng 20MHz. Đây là phần hướng dẫn chi tiết từng phần một và là phần quan trọng nhất, các bạn phải đọc toàn bộ mới có thể hiểu được và áp dụng cho các dòng vi xử lí khác.

3.4. Chương trình CCS C

Ngôn ngữ được sử dụng là C từ phần mềm hỗ trợ PIC C COMPLIER của microchip và lập trình cho PIC 18F4550, thạch anh sử dụng 20MHz.

3.4.1. Các bước lập trình thuật toán Flood Fill.

Các bước cần thiết lập tình theo trình tự như sau trong vòng lặp while trong main:

Bước 1: Xác định hướng hiện tại của chuột và đọc giá trị từ cảm biến để cập nhật thông tin bức tường tương ứng với hướng của chuột tại ô hiện tại.

- Kiểm tra bức tường phía Bắc: YES: cập nhật tường vào mảng walldata. NO: không làm gì

- Kiểm tra bức tường phía Đông: YES cập nhật tường vào mảng. NO: không làm gì.

- Kiểm tra tường phía Nam: YES: cập nhật tường vào mảng walldata. No: Không làm gì.

- Kiểm tra tường phía Tây: YES: cập nhật tường vào mảng. NO: Không làm gì.

Bước 2: Thực hiện thuật toán Flood Fill

- Kiểm tra stack có rỗng hay không, phải đảm bảo stack rỗng thì thực hiện.

- Đẩy giá trị flood tại ô hiện tại chuột đang đứng vào stack

=> Vòng lặp:

- Kiểm tra stack rỗng hay không.

- Lấy giá trị từ đỉnh stack ra kiểm tra giá trị. Nếu là ô đích thì dừng lại. Nếu không phải ô đích thì thực hiện bước tiếp theo.

- Đọc các giá trị hàng xóm mở với ô hiện tại kiểm tra để tìm ra giá trị flood nhỏ nhất trong các ô hàng xóm mở đó.

- Kiểm tra xem giá trị flood ô hiện tại trừ đi 1 có bằng giá trị flood của ô hàng xóm mở nhỏ nhất hay không:

+ Đúng: Không làm gì cả

+ Sai: cập nhật giá trị mới cho ô hiện tại bằng cách: lấy giá trị flood của ô hàng xóm bé nhất cộng với 1 và gán vào ô hiện tại. cập nhật vào mảng Mazeflood.

- Đẩy các giá trị ô hàng xóm mở vào stack để kiểm tra lại

- Lặp lại cho đến khi stack rỗng.

Bước 3: Kiểm tra lại các bức tường xung quanh ô hiện tại để quyết định hướng đi:

- Kiểm tra bức tường phía Bắc: YES. Bỏ qua. NO: đẩy vào stack kiểm tra

- Kiểm tra bức tường phía Đông: YES. Bỏ qua. NO: đẩy vào stack kiểm tra

- Kiểm tra bức tường phía Nam: YES. Bỏ qua. NO: đẩy vào stack kiểm tra

- Kiểm tra bức tường phía Tây: YES. Bỏ qua. NO: đẩy vào stack kiểm tra

- Quyết định hướng đi đến ô nhỏ nhất hàng xóm mở.

=> Khi đã đi đến ô mới thì tiếp tục vòng lặp while trong chương trình main.

Lưu ý: Cần tính toán góc quay và bước di chuyển. Để biết chuột đi qua một cell bắt buộc phải đếm số xung (đối với động cơ DC) hoặc đếm số bước đối với động cơ bước, số bước được lấy từ thực tế dựa vào tính toán kết hợp lại. Các bước rẽ trái, phải, đi thẳng hay quay lại đều được thực hiện rời rạc, do đó khi di chuyển nếu tốc độ xử lý của vi xử lý chậm sẽ khiến cho chuột bị dừng lại một lúc tại mỗi ô để thực hiện xong các lệnh tính toán kiểm tra mới có thể đi tiếp được. Do đó để chuột có thể di chuyển một cách nhanh và mượt đòi hỏi tốc độ xử lý của vi xử lý phải nhanh đủ để xử lý toàn bộ mê cung trong nhiều nhất là 0.25 giây để ta không thể nhìn thấy chuột bị dừng lại trong lúc di chuyển.

Tạo một mảng khai báo gồm 8*8 = 64 (từ 0 đến 63) phần tử chứa các giá trị flood (value)

Khai báo trong PIC C:

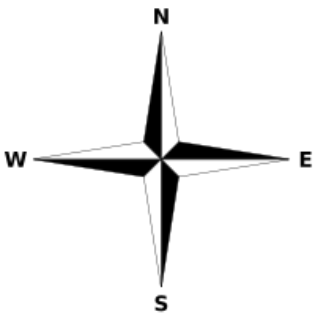
unsigned char mazeflood[64]=

```
{
  6,  5,  4,  3,  3,  4,  5,  6,
  5,  4,  3,  2,  2,  3,  4,  5,
  4,  3,  2,  1,  1,  2,  3,  4,
  3,  2,  1,  0,  0,  1,  2,  3,
  3,  2,  1,  0,  0,  1,  2,  3,
  4,  3,  2,  1,  1,  2,  3,  4,
  5,  4,  3,  2,  2,  3,  4,  5,
  6,  5,  4,  3,  3,  4,  5,  6,
};
```

Mazeflood: là mảng chứa số phần tử của mảng được đánh số từ 0 đến 63 tương ứng với vị trí của nó, phần tử đầu tiên là phần tử thứ 0. Gán một biến current_cell để đại diện cho vị trí của con chuột.

Hướng thực tế ngược lại so với mảng, mô tả vị trí mazeflood[vị trí] như sau:

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	18	20	21	22	23
8	9	10	11	12	13	14	15



0 1 2 3 4 5 6 7

Bảng 3.4.1. mô tả vị trí mảng maze và hướng

Value: là giá trị của phần tử thứ x của mảng mazeflood. Gán một biến là Flood_cell để truy cập vào từng phần tử của mảng.

unsigned char current_cell = 0;

unsigned char flood_cell=0;

Tiếp tục tạo một mảng gồm 64 phần từ để lưu trữ thông tiin bức tường tại từng ô

```
unsigned char Walldata[64] ;
#define CHECKBIT(x,b)   (x&b)
#define BIT0   0x01
#define BIT1   0x02
#define BIT2   0x04
#define BIT3   0x08
#define BIT4   0x10
#define BIT5   0x20
#define BIT6   0x40
#define BIT7   0x80
```

#define là định nghĩa địa chỉ các bit trong một byte dữ liệu, ví dụ: bit 0 (MSB) có địa chỉ là 0x01 (0b00000001). Tương tự định nghĩa cho các bit còn lại. Thông tin bức tường được mô tả như bảng 3.4.2

WALLDATA	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
value	V	0	0	0	W	S	E	N

Bảng 3.4.2. Thông tin byte dữ liệu từ bức tường.

E (East): lưu thông tin bức tường hướng Đông.

W (West): Lưu thông tin bức tường hướng Tây.

S (South): Lưu thông tin bức tường hướng Nam.

N (North): Lưu thông tin bức tường hướng Bắc.

V (Visited): Lưu thông tin đánh dấu ô đã đi qua.

Lệnh*: #define* CHECKBIT(x,b) (x&b): sử dụng để kiểm tra bit x trong byte dữ liệu b, trả về kết quả là TRUE nếu bit =1, FALSE nếu bít khác 1.

Mỗi một byte trong mảng Walldata gồm 8 bit chứa những thông tin của bức tường tại ô đó. Ban đầu được gán =0 vì được xem như không có bức tường, khi di chuyển dựa vào cảm biến ta sẽ đọc được bức tường để lưu vào mảng. Bân đầu bức tường xung quanh phải được đặt giá trị trước để giới hạn mê cung.

Tạo một biến current_dir để chỉ hướng hiện tại của micromouse, giả sử ban đầu ta gán cho nó là 0: tương ứng với hướng Bắc, 1 tương ứng với hướng Đông … gán số tương ứng là tuy vào mỗi người lập trình, tuy nhiên để tiện cho việc sử dụng thao tác sau này, chúng ta có thể gán biến theo bảng 3.4.3

DIREC	BIT 7	**BIT6**	BIT5	**BIT4**	BIT3	**BIT2**	BIT1	**BIT 0**
Binary	0	**W**	0	**S**	0	**E**	0	**N**
DEC	128	**64**	32	**16**	8	**4**	2	**1**

Bảng 3.4.3. Byte dữ liệu hướng của micromouse

Như vậy từ bảng 3.4.2 ta lấy giá trị hệ cơ số thập phân để sử dụng tính toán:

Hướng West: current_dir=01000000b = 64

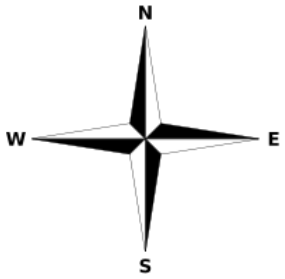
Hướng South: current_dir=00010000b = 16

Hướng East: current_dir=00000100b = 4

Hướng North: current_dir=00000001b = 1

Tại mỗi vị trí phải đọc hướng của chuột trước khi lưu thông tin bức tường vào mảng walldata để tránh bị mất dữ liệu bức tường trước đó.

Hướng quy định luôn được giữ nguyên không được thay đổi khi chuột đổi hướng, chỉ có hướng của cảm biến là thay đổi trong quá trình di chuyển của chuột, vấn đề này được giải thích rõ như sau:



Hình 3.5.1

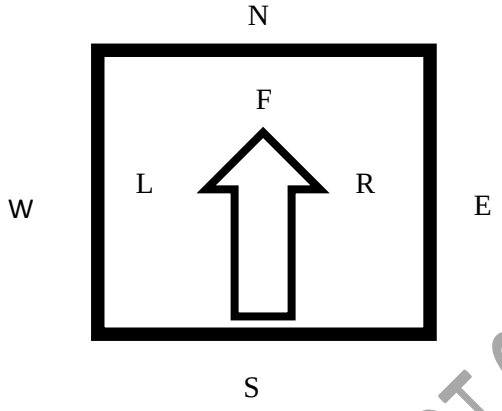
Đặt hướng ban đầu theo quy định như hình 3.4.2

Các bit lưu thông tin bức tường được đặt trong byte dữ liệu của mảng walldata như bảng 3.4.3

Quy đinh bit x: bit = 1: có bức tường, bit = 0: không có bức tường. Các hình ảnh minh họa dưới đây quy đinh: mũi tên là hướng hiện tại của chuột tại ô đó, kí hiệu W,E,S,N là hướng quy định không thay đổi được lưu trong byte. L,R,F tương ứng với hướng của cảm biến trên con chuột (L:left, R: right, F:front)

Kiểm tra hướng hiện tại của chuột

- *If (current_dir ==1) // chuột hướng về phía Bắc (N):*



Hình 3.4.2. Mô tả hướng của chuột

+ kiểm tra cảm biến bên trái: lưu thông tin x vào bit hướng Tây (W) trong byte dữ liệu tường (4 bit thấp trong byte dữ liệu tường: LSB=x000):

lệnh CCS:

left<<3 // dịch "left" đến vị trí BIT3 trong LSB, xem // bảng 3.5b

// "left" là "1": có tường" hoặc "0": k có tường

+ Kiểm tra cảm biến phải: lưu thông tin vào bit hướng Đông (E) trong byte dữ liệu tường (LSB: 00x0)

Lệnh CCS: *Right<<1*

+ Kiểm tra cảm biến trước: lưu thông tin bit hướng Bắc vào byte dữ liệu tường (LSB: 000x)

Lệnh CCS: *front<<0*

+ Phía sau chuột là hướng Nam không có bức tường: *0000*

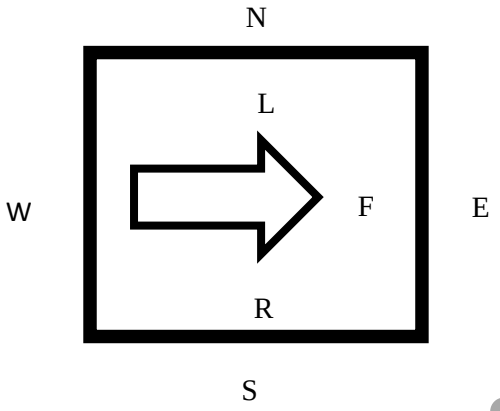
⇨ Cộng tất cả chúng lại gán vào biến wallstring bằng lênh:

Wallstring=(left<<3 | right<<1 | front);

⇨ Thực hiện lưu vào mảng thông tin bức tường của ô hiện tại:

Walldata[current_cell] = wallstring;

- *If (current_dir == 4) // chuột đang hướng sang hướng Đông (E)*



Hình 3.4.3 Mô tả hướng của chuột

+ Kiểm tra cảm biến trái: lưu thông tin bit hướng Bắc (N)
+ Kiểm tra cảm biến phải: lưu thông tin vào bit hướng Nam (S)
+ Kiểm tra cảm biến trước: lưu thông tin vào bit hướng Đông (E).
+ Không có bức tường nào hướng Tây (W)
Lưu tất cả thông tin vào byte của ô hiện tại.

Làm tương tự cho các hướng còn lại, bắt buộc khi đọc cảm biến phải xét điều kiện hướng trước rồi mới đọc cảm biến.

Cách tạo mảng và thực hiện Flood Fill

Tạo mảng strack gồm 128 giá trị số nguyên không dấu:

unsigned char stk[128];

Tạo biến để làm con trỏ cho strack:

unsigned char stkprt=0;

Tạo một cờ điều kiện để thoát khỏi vòng lặp Flood, biến này = 0 nghĩa là strack chưa rỗng, khi nào biến này đặt bằng một thì thực hiện xong vòng lặp và thoát khỏi vòng lặp.

stk_empty_flag = 0;

Tạo một mảng chứa 4 phần tử, mỗi phần tử đại diện cho một ô hàng xóm (một ô có nhiều nhất 4 ô hàng xóm mở với nó).

unsigned char neighbour_val[] = {255,255,255,255};

Tạo một vòng lặp while (điều kiện thoát) để thực hiện thuật toán Flood, luôn luôn lấy giá trị ô hiện tại con chuột đang đứng (current_cell) đưa vào strack trước, nó sẽ được thực hiện kiểm tra sau cùng trong thuật toán.

Bắt đầu tăng con trỏ lên 1 đơn vị:

```
Stkprt++;
stk[stkptr] = current_cell; // đẩy giá trị ô chuột đang đứng vào trước
While (stk_empty_flag = 0)
{
        // gán biến giá trị flood bằng giá trị flood của ô mà con trỏ
        stack đang trỏ tới
floodcell = stk[stkptr];
if (stkptr == 1) stk_empty_flag = 1;  // đặt điều kiện thoát vòng lặp
    else
    stkptr--; // giảm con trỏ xuống 1 đơn vị
// đọc thông tin bức tường
wallinfo = walldata[floodcell];
//kiểm tra các giá trị flood của các ô hàng xóm mở
if (!(CHECKBIT(wallinfo, BIT0) )))   // kiểm tra bức tường hướng Bắc
      neighbour_val[0] = mazeflood[floodcell + 8];
if (!(CHECKBIT(wallinfo, BIT1) ) )   // kiểm tra bức tường hướng Đông
      neighbour_val[1] = mazeflood[floodcell + 1];
if (!(CHECKBIT(wallinfo, BIT2) ) ) )   // kiểm tra bức tường hướng Nam
    neighbour_val[2] = mazeflood[floodcell - 8];
if (!(CHECKBIT(wallinfo, BIT3) ) )   )  // kiểm tra bức tường hướng Tây
      neighbour_val[3] = mazeflood[floodcell - 1];
```

// sau khi gán giá trị Flood các ô hàng xóm vào các biến tương ứng ta thực hiện kiểm tra giá trị nhỏ nhất trong 4 ô hàng xóm:

```
x = 0;
    while (x < 3)
    {   // gán giá trị nhỏ nhất vào biến neighbour_val[0]
      if (neighbour_val[0] > (neighbour_val[x + 1]))
```

```
            neighbour_val[0] = neighbour_val[x + 1];
          x+=1;
        }
```

// sau khi tìm được giá trị nhỏ nhất gán vào biến neighbour_val[0] , ta thực hiện kiểm tra xem ô đó có phải là ô đích hay không và có thõa điều kiện thuật toán Flood Fill hay không:

```
if ((!(floodcell == destination )) && (!(mazeflood[floodcell] == (1+neighbour_val[0])))) // nếu không phải là ô đích và không thõa điều kiện thì thực hiện thuật toán Flood Fill (điền lũ)
    {
      mazeflood[floodcell] = 1 + neighbour_val[0];     // Đặt giá trị ô hiện tại bằng ô hàng xóm nhỏ nhất công với 1.
      if (!(CHECKBIT(wallinfo, BIT0))) // kiểm tra bức tường phía bắc
      {
// Mỗi lần đọc một ô hàng xóm lấy vào stack thì phải tăng con trỏ lên 1 đơn vị rồi gán nó vào con trỏ stack đó
        stkptr++;
        stk[stkptr] = floodcell + 8;        ///Lấy giá trị ô hàng xóm phía bắc đưa bào stack để kiểm tra
        stk_empty_flag = 0; // đặt cờ stack =0 để biết stack đang không rỗng
      }
      // Thực hiện tương tự cho các ô hàng xóm hướng tương ứng còn lại
      if (!(CHECKBIT(wallinfo, BIT1)))  // phia đông
      {
        stkptr++;
        stk[stkptr] = floodcell + 1;
        stk_empty_flag = 0;
      }
      if (!(CHECKBIT(wallinfo, BIT2))) // phía Nam
```

```
{
  stkptr++;
  stk[stkptr] = floodcell - 8;
  stk_empty_flag = 0;
}
if (!(CHECKBIT(wallinfo, BIT3))) // Phía Tây
{
  stkptr++;
  stk[stkptr] = floodcell - 1;
  stk_empty_flag = 0;
}
// Sau khi thực hiện điền lũ xong thuật toán quay lại vòng lặp while kiểm tra điều kiện stack rỗng hay không và thực hiện tiếp đến khi đã kiểm tra hết các ô đều đã thõa điều kiện.
}
  }
```

Rõ ràng cách lập trình và lý thuyết về thuật toán hoàn toàn giống nhau, thực hiện đúng các bước theo lý thuyết, kiểm tra điều kiện thuật toán và thực hiện thuật toán.

Cách chọn hướng đi:

Sau đây là bài giải thích vì sao chọn bit lưu hướng của Micromouse như bảng sau:

DIREC	BIT 7	**BIT6**	BIT5	**BIT4**	BIT3	**BIT2**	BIT1	**BIT 0**
Binary	0	**W**	0	**S**	0	**E**	0	**N**
DEC	128	**64**	32	**16**	8	**4**	2	**1**

Có nhiều cách để chọn hướng đi khác nhau tùy vào mỗi người lập trình, có thể chọn đi thẳng, rẽ trái, rẽ phải hay di chuyển theo hướng nào thuật toán Flood thực hiện trước. Ở đây chúng ta chọn hướng theo cách tùy thuộc vào giá trị nhỏ nhất của ô hàng xóm nào được xét trước.

Trước tiên kiểm tra thông tin bức tường của các ô hàng xóm

```
unsigned char neighbour_val[] = {255,255,255,255, 1};
unsigned char wallinfo = 0, x = 0, x2 = 0;
wallinfo = walldata[current_cell]; // lấy thông tin bức tường ô hiện tại
// kiểm tra thông tin bức tường các ô hàng xóm mở
if (!(CHECKBIT(wallinfo, BIT0)))
  neighbour_val[0] = mazeflood[current_cell + 8];
if (!(CHECKBIT(wallinfo, BIT1)))
  neighbour_val[1] = mazeflood[current_cell + 1];
if (!(CHECKBIT(wallinfo, BIT2)))
  neighbour_val[2] = mazeflood[current_cell - 8];
if (!(CHECKBIT(wallinfo, BIT3)))
  neighbour_val[3] = mazeflood[current_cell - 1];
// sau khi kiểm tra xong ta thực hiện tương tự trong thuật toán Flood Fill là kiểm tra giá trị Flood nhỏ nhất của các ô hàng xóm mở:
x = 0;
x2 = 0;
while (x < 3)
{
  if (neighbour_val[0] > neighbour_val[x + 1])
  {
  x2 = x + 1;
    neighbour_val[4] = pow(2, ( 2 * x2 )) ;
neighbour_val[0] = neighbour_val[x + 1];        // gán giá trị Flood nhỏ nhất vào biến  neighbour_val[0]
  }
  x += 1;
}
```

```
 if (current_dir == neighbour_val[4])
 {
   neighbour_val[4] = 'M';
   return neighbour_val[4];
 }

 if ( ((current_dir == 64) && (neighbour_val[4] == 1))
   || (current_dir == (neighbour_val[4]/4)) )
 {
   neighbour_val[4] = 'R';
   return neighbour_val[4];
 }
 if ( ((current_dir == 1) && (neighbour_val[4] == 64))
   || (current_dir == (4 * neighbour_val[4])) )
 {
   neighbour_val[4] = 'L';
   return neighbour_val[4];
 }
 neighbour_val[4] = 'F';
 return neighbour_val[4];
}
```

Giải thích một số câu lệnh:

Câu lệnh : *neighbour_val[4] = pow(2, (2 * x2))*

- Ở đây ta thấy, trong lệnh while (x<3) kiểm tra nêu ngay ban đầu mà ô hướng bắc (biến *neighbour_val[0]*) là ô nhỏ nhất thõa điều kiện thì trong vòng lặp if không thực hiện kiểm tra các ô còn lại nữa, nghĩa là biến *neighbour_val[4]* nhỏ nhất là 1, *if (current_dir == neighbour_val[4])*

```
 {
```

```
    neighbour_val[4] = 'M';
    return neighbour_val[4];
  }
```

Dòng lệnh trên xét khi hướng *current_dir* =1 tương ứng với hướng bắc (cùng hướng hiện tại của chuột) thì cho chuột đi thẳng.

- Tương tự khi xét đến biến *neighbour_val[1])* thõa điều kiện thì thoát vòng lặp while (x<3) lúc này x2=x+1=1+1=2, biến *neighbour_val[4] = 2^2 = 4 (nghĩa là ô nhỏ nhất nằm hướng đông)*

```
  if ( ((current_dir == 64) && (neighbour_val[4] == 1))
    || (current_dir == (neighbour_val[4]/4)) )
  {
    neighbour_val[4] = 'R';
    return neighbour_val[4];
  }
```

(current_dir == (neighbour_val[4]/4))) Tương ứng với *current_dir* =4/4 =1, nghĩa là nếu hướng con chuột đang hướng về hướng bắc (1) thì cho chuột rẽ phải sẽ hướng về hướng đông, nơi có ô hàng xóm có giá trị Flood nhỏ nhất.

Giải thích tương tự cho các hướng còn lại, mỗi một hướng chọn đi ta thực hiện lệnh chia bội số với 4 ta được các giá trị tương ứng với hướng của con chuột gồm 1,4,16 và 64.

3.4.2. Chống nhiễu cảm biến

Để chống nhiễu cho cảm biến bởi các yếu tố ngoại cảnh như ánh sáng đèn điện, ánh sáng mặt trời … chúng ta sử dụng timer của PIC để thực hiện xuất xung cho LED phát.

- . Sử dụng ngắt timer0.

Trong PIC 18F4550 có 4 timer khác nhau gồm timer0,timer1,timer2 và timer3, cả 4 timer đều có thể cấu hình để hoạt động ở chế độ 8bit hoặc 16bit tùy vào người sử dụng.

Ở đây chúng ta sử dụng timer0 để cấu hình hoạt động ngắt cho timer0 ở chế độ 16bit, Bảng 3.4.3 mô tả cấu trúc thanh ghi timer0.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Bảng 3.4.4. Cấu trúc thanh ghi T0CON

Bit7 (TMR0ON):

1: cho phép timer0 hoạt động.

0: Không cho phép timer0 hoạt động.

Bit6 (T08BIT):

1: Cấu hình timer0 hoạt động chế độ 8bit

0: Cấu hình timer0 hoạt động chế độ 16bit

Bit5 (T0CS):

1: Sử dụng ngắt ngoài timer0 ở chân T0CKI

0: Sử dụng bộ chia tần số thạch anh làm ngắt.

Bit4 (T0SE) để cấu hình ngắt cho pin T0CKI

Bit3 (PSA)

1: Không sử dụng bộ chia tần

0: Sử dụng bộ chia tần trong PIC.

Bit 2-0 (T0PS2, T0PS1, T0PS0): ĐỂ cấu hình bộ chia tần cho timer0 như bảng 3.4.4.

111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Bảng 3.4.5. Bộ chia tần trong timer

Từ cấu hình của thanh nghi timer0 như trên ta lập trình CCS như sau:

Setup_timer0(87); // (10000111)

Trong bộ chia tần số timer đều qua bộ chia 4. ở đây ta sử dụng timer0 chế độ 16bit, bộ chia tần 256. Do đó ta tính toán thời gian ngắt cho timer như sau:

T = (4*256)/20Mhz = 0.051 (ms)

Như vậy cứ sau 0.051ms bộ đếm của timer0 sẽ tăng lên 1 đơn vị, có tổng cộng 2^{16} = 65536 giá trị, ngắt tràn timer0 sẽ sảy ra khi đếm đến 65536.

Mục đích sử dụng timer0 là để xuất xung chống nhiễu cho cảm biến, do đó ở đây cứ 20ms ta sẽ ngắt timer0 1 lần để bật LED phát, sau 20ms lại tắt. như vậy chu kì cần là 40ms. Từ bộ timer0 ta tính giá trị cho bộ đếm để ngắt sau 20ms như sau:

0.051ms => 1

20ms => 20/0.051 = 390

Do đó ta cấu hình cho bộ đếm của timer0 = 65536-390 = 65145

Set_timer0(65145);

- . Chống nhiễu cho LED hồng ngoại.

Cứ sau 20ms ta tắt các LED phát và đọc giá trị ADC trả về

Led_off= read_adc();

Sau 20ms ta lại bật LED phát lên và đọc giá trị ADC trả về

Led_on = read_adc();

Như vậy ta sẽ biết được giá trị thực của cảm biến là:

ADC = led_on-led_off;

Thời gian ngắt timer0 ta có thể tùy chọn, tuy nhiên tránh ngắt quá nhiều lần để chương trình có thể chạy xong mà không bị dừng liên tục do ngắt và thời gian bật LED phát cũng phải đủ để LED thu đọc giá trị ADC trả về.

Ngoài ra để tránh ảnh hưởng của các cảm biến với nhau chúng ta thực hiện bật từng cảm biến một để đo sau đó tắt đi.

3.4.3. Căn chỉnh động cơ

Khi Micromouse chạy trên 1 đường thẳng có thể sẽ khiến micromoue bị lệch qua một bên do tích lũy sai số, do đó cần phải điều khiển cho Micromouse chạy ở giữa đường trong mê cung. Việc điều khiển cũng tương tự như động cơ

DC, tuy nhiên động cơ bước là loại động cơ điều khiển vòng hở, do đó ta phải dựa vào cảm biến để biết Micromouse đang lệch sang hướng nào.

Ta thực hiện kiểm tra giá trị ADC bên trái và bên phải để so sánh

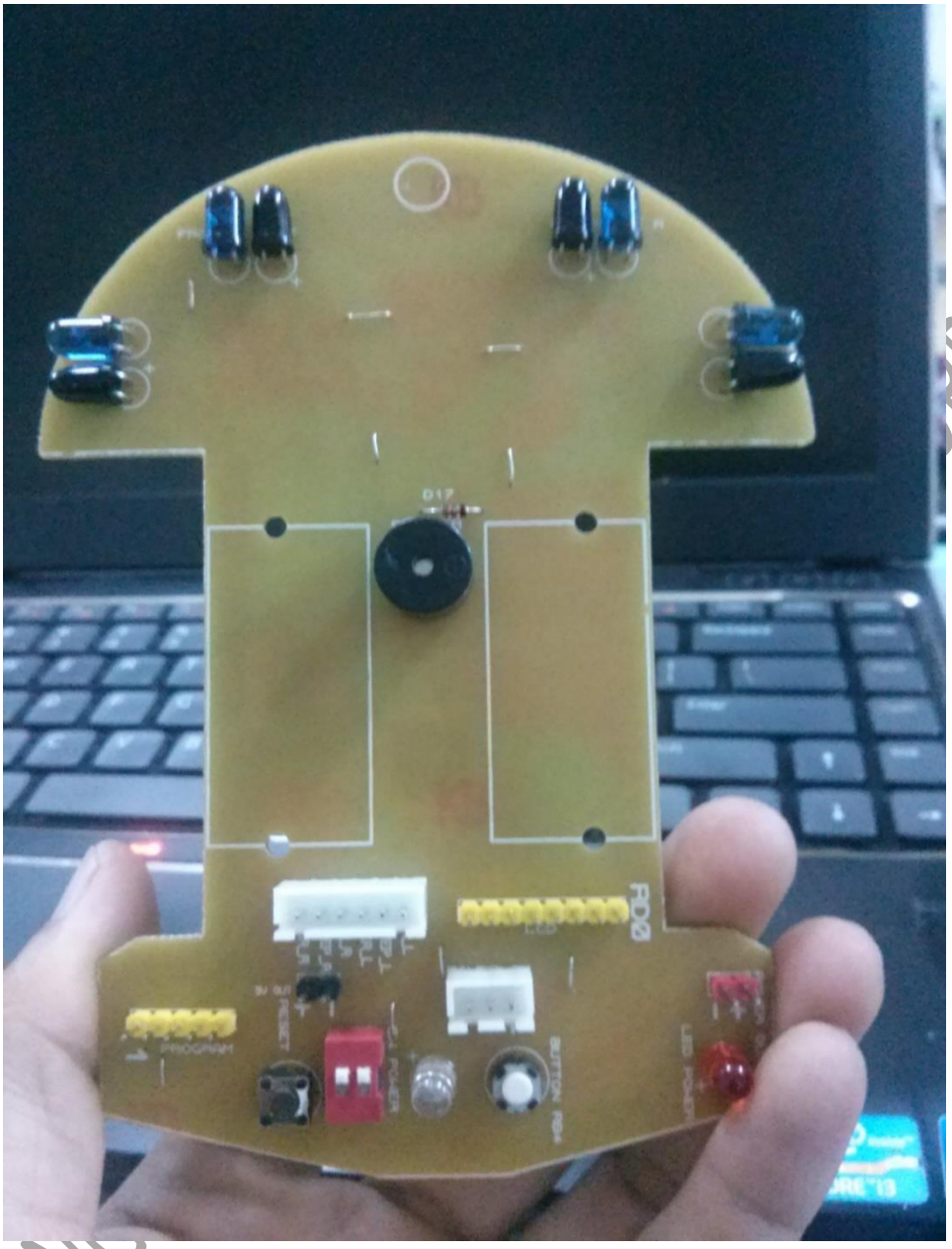
```
If (left>=right)
{
Error_l = left – right;
}
Else

Error_r = right – left;
```

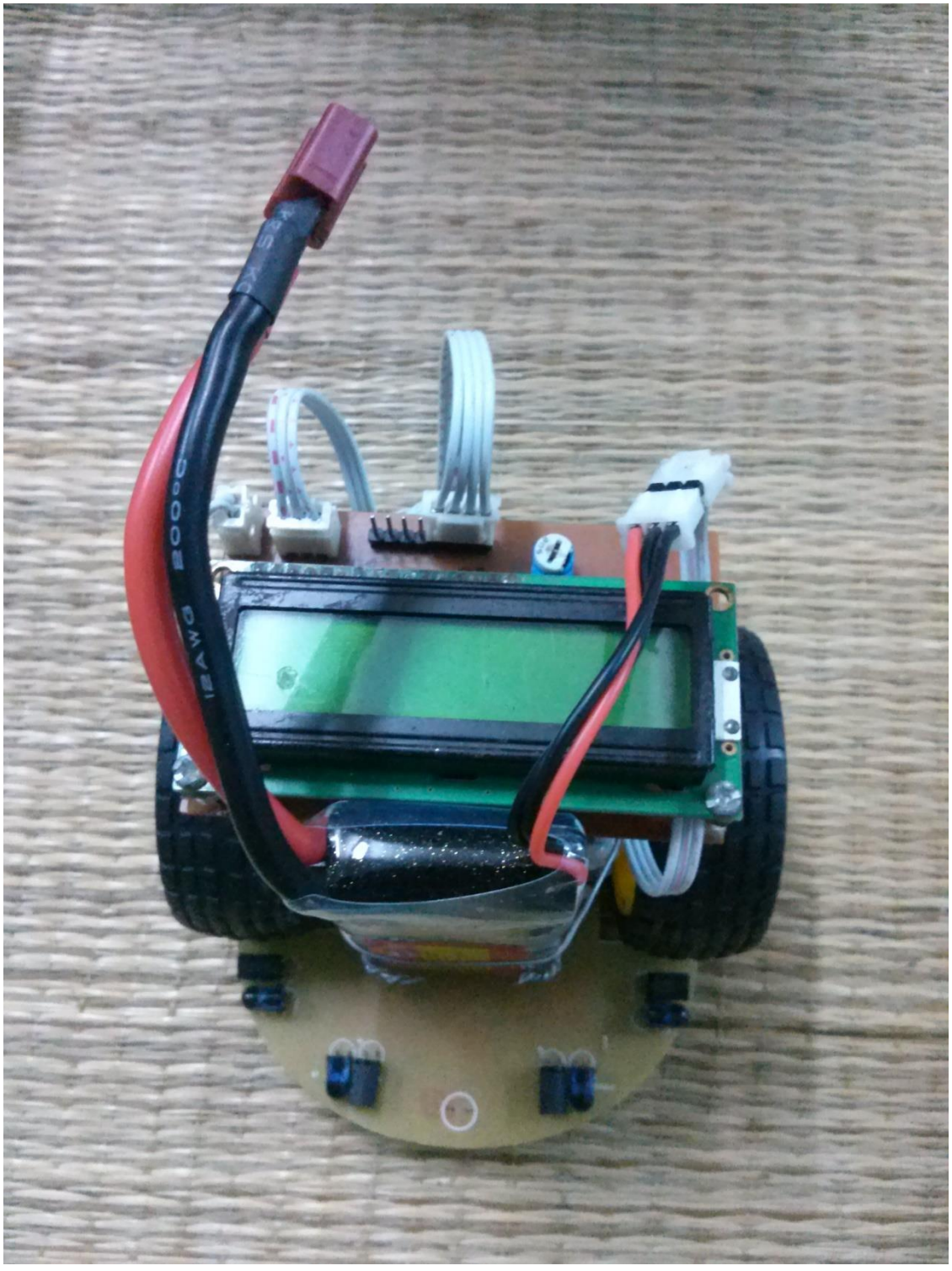
Từ giá trị sai số của 2 bên cảm biến ta sẽ biết được Micromouse đang lệch sang bên nào, lúc đó chúng ta sẽ điều khiển Micromouse rẽ qua hướng ngược lại cho đến khi đúng điều kiện. Ta sẽ cho phép điều kiện sai số 1 giá trị (value_error) nào đó đủ để micromouse không bị chạm bức tường.

Ngoài ra chúng ta có thể vận dụng hai cảm biến phía trước để căn chỉnh khi nó gặp bức tường phía trước, Đọc giá trị ADC trả về từ 2 cảm biến để biết sai số. Khi gặp bức tường phía trước giá trị ADC trả về từ 2 cảm biến phải bằng nhau, kiểm tra sai số để có điều chỉnh cho micromouse vuông góc với tường.

4.1. Một số hình ảnh

Hình 4.1.1 Mạch hoàn thành





Hình ảnh chạy thử:

Hình 4.2.1. Hình ảnh chạy thử mê cung 6 x 5

4.3. Tài liệu tham khảo

TIẾNG VIỆT:

1. Hoàng Thị Điệp: "*Lập trình C++, 2010*" Giảng viên khoa Công Nghệ Thông Tin trường ĐH Công Nghệ.
2. Đoàn Hiệp: "*Điều khiển động cơ bước*" sách dịch từ tài liệu hướng dẫn động cơ bước của giáo sư *Douglas W. Jones, Đại học IOWA.*

TIẾNG ANH:

1. David Hannaford: "*Tuning Flood algorithms"* MINOS 2010
2. George Law: "*Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms*" *International Journal of Computer Theory and Engineering, Vol. 5, No. 3, June 2013.*
3. Mr.Peter Harrison: admin website *micromouseonline.com.*
4. Mr. Ching-Hsing Pei & Jing Lee: *"Micro Mouse Maze Solving 2011" Center for General Education, Tung Fang Design University, Department of Electronics Engineering, Southern Taiwan University.*
5. Chutisant Kerdvibulvech & Andrew Junmee: *"A New Method for a Micromouse to Find its Way Through a Maze Unaided" Proceedings of the 2013 International Conference on Systems, Control and Informatics.*
6. Ibrahim Elshamarka & Abu Bakar Sayuti Saman: "*Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm" - International Journal of Computer Applications (0975 – 8887) Volume 56– No.5, October 2012.*
7. Garima & Vipul Aggarwal - "*Optimization of Flood fill algorithm using Look-Ahead Technique"- IRACST – Engineering Science and Technology: An International Journal (ESTIJ), ISSN: 2250-3498, Vol.3, No.4, August 2013.*

WEBSITE:

Http://ieeecharusat.wordpress.com/

Http://micromouseonline.com/

Http://ieee.ucsd.edu/

Http://micromouseusa.com/

Http://www.tic.ac.uk/micromouse/

Http://www.picaxe.com/

Http://picvietnam.com

THỰC RA MICROMOUSE CÒN RẤT NHIỀU THỨ PHẢI TÌM HIỂU, Ở TRÊN CHỈ LÀ PHẦN LẬP TRÌNH CƠ BẢN CHO MICROMOUSE TÌM ĐƯỢC ĐƯỜNG ĐI NGẮN NHẤT (KHÔNG PHẢI NHANH NHẤT) TRONG MÊ CUNG. CHỈ CẦN THAY ĐỔI PHẦN KÍCH THƯỚC MÊ CUNG VÀ MỘT VÀI DÒNG LỆNH BẠN CÓ THỂ MỞ RỘNG MÊ CUNG LÊN ĐẾN BAO NHIÊU TÙY THÍCH, MIỄN LÀ BỘ NHỚ VI SỬ LÝ BẠN ĐỦ NHIỀU VÀ ĐỦ NHANH.

ĐỂ MICROMOUSE TÌM ĐƯỢC ĐƯỜNG NHANH NHẤT THÌ PHẢI TÌM HIỂU THUẬT TOÁN CHO MICROMOUSE DI CHUYỂN CHÉO, THAY VÌ CUA 90^0 THÌ CHỈ CẦN CUA 45^0, HÃY XEM CÁC VIDEO TRÊN YOUTOBE BẠN SẼ THẤY. NÓ TƯƠNG ĐỐI KHÓ.

THÔNG TIN SINH VIÊN

Họ và tên: NGUYỄN LƯƠNG THÀNH

Ngày sinh: 05/02/1991

Sinh Viên Khóa 13

Trường Đại Học Tôn Đức Thắng

Khoa: Điện - Điện tử

Ngành: Tự Động Điều Khiển

Lớp: 09040003

MSSV: 40900040

SDT: 01648911181

Email: thanhnl0502@gmail.com