

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - CƠ - TIN HỌC**



**BÁO CÁO CUỐI KỲ
THỊ GIÁC MÁY TÍNH - MAT3562 1**

Giảng viên: TS. Hà Mạnh Toàn

Đề tài:

Nhận diện bi-a và tìm đường đi trong game 8 Ball Pool

Thành viên

Bùi Khánh Duy – 20001898

HÀ NỘI - 12/2023

Giới thiệu

Bi-a đã và đang là một trò chơi thu hút rất nhiều người tham gia chơi và học hỏi. Bản thân em cũng không ngoại lệ. Ngay từ khi bắt đầu học môn này, em đã quyết định làm một sản phẩm có liên quan đến nó.

Tuy vậy, với hạn chế về mặt kĩ thuật cũng như khó khăn khi thực hiện với dữ liệu thực tế, cuối cùng em lựa chọn thực hiện dự án trên môi trường game 2D có tên: **8 Ball Pool** của MINICLIP.

Mục đích của dự án là xây dựng một hệ thống hỗ trợ người chơi game 8 Ball Pool, chủ yếu là hướng dẫn cho các người chơi mới (điều mà đang chưa quá được coi trọng trong tựa game này).

Các kiến thức để làm ra dự án phần lớn được sử dụng từ việc học môn Thị giác máy tính của thầy Hà Mạnh Toàn. Ngoài ra các công cụ và nguồn tham khảo đều được trích dẫn đầy đủ ở mục tham khảo của bài báo cáo.

Cuối cùng, em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn - Thầy Hà Mạnh Toàn. Thầy đã giảng dạy, truyền đạt, chia sẻ cho em những kiến thức quý báu trong suốt học kỳ qua. Nhờ sự giúp đỡ của thầy mà em được tiếp thu những kiến thức rất hữu ích cả trong và ngoài chuyên ngành. Đây sẽ là hành trang giúp em vững bước trên những chặng hành trình sắp tới.

Mục lục

Nội dung	3
Phát biểu bài toán	3
Circle Hough Transform	3
Phát hiện các góc	4
Phát hiện các viên bi	5
Tìm đường đi	8
Hạn chế	12
Tổng thể	12
Các hình ảnh cho việc nhận diện sai	12
Kết quả	13
Hướng phát triển	14
Tài liệu tham khảo	15

Nội dung

Phát biểu bài toán

Input

- Video quay lại quá trình chơi game bi-a 8 Ball Pool

Output

- Video sau khi thêm các hình tròn và đường đi.

Công cụ chính được sử dụng trong dự án: cv2.HoughCircles

Circle Hough Transform

Theo [1], đây là một thuật toán trong xử lý ảnh được sử dụng để phát hiện các đường tròn trong hình ảnh, thuộc họ các thuật toán Hough Transform, được đặt tên theo nhà toán học Paul Hough

Thuật toán Circle Hough Transform hoạt động bằng cách chuyển đổi hình ảnh từ không gian tọa độ thông thường sang không gian tham số (không gian Hough). Không gian Hough là không gian các tham số hình học của đường tròn: tâm (x, y) và bán kính r . Mỗi điểm trong ảnh ban đầu là một đường tròn trong không gian này. Các điểm tập trung trong không gian Hough cho biết có một vòng tròn tương ứng trong ảnh ban đầu. Vùng tập trung nhất chỉ ra vị trí và kích thước vòng tròn. Sau khi chuyển đổi xong, ta dễ dàng tìm ra các vòng tròn bằng cách tìm theo các giá trị cực đại trong ma trận tích lũy.

Các tham số của cv2.HoughCircles:

- `images` = Ảnh xám
- `method` = `cv.HOUGH_GRADIENT`
- `dp` = 1
- `minDist` = `r1`
- `param1` = `p1`

- `param2 = p2`
- `minRadius = r1`
- `maxRadius = r2`

Ý nghĩa của các tham số:

- `p1`: Giá trị gradient được sử dụng trong phát hiện cạnh.
- `p2`: Nguồn tích lũy cho phương pháp HOUGH_GRADIENT, giá trị càng bé càng cho phép phát hiện nhiều vòng tròn hơn.

Phát hiện các góc

Trong quá trình thử nghiệm, các tham số được chọn như sau:

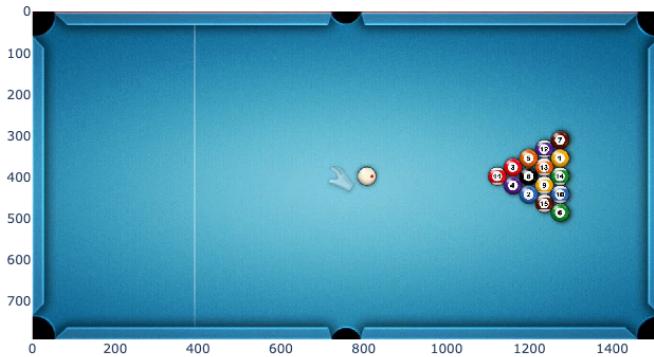
- `r1 = HOLE_RADIUS - 2 = 46`
- `p1 = 150`
- `p2 = 16`
- `r2 = HOLE_RADIUS = 48`

Quá trình này sẽ tìm được tọa độ 4 lỗ góc của bàn bi-a. 2 lỗ ở giữa được tính bằng tổng tọa độ chia 2 theo chiều ngang.



Hình 1: Các lỗ góc được phát hiện

Từ các góc, ta cắt được phần bàn chơi và tập trung vào xử lý ở đây:



Hình 2: Phần bàn chơi

Phát hiện các viên bi

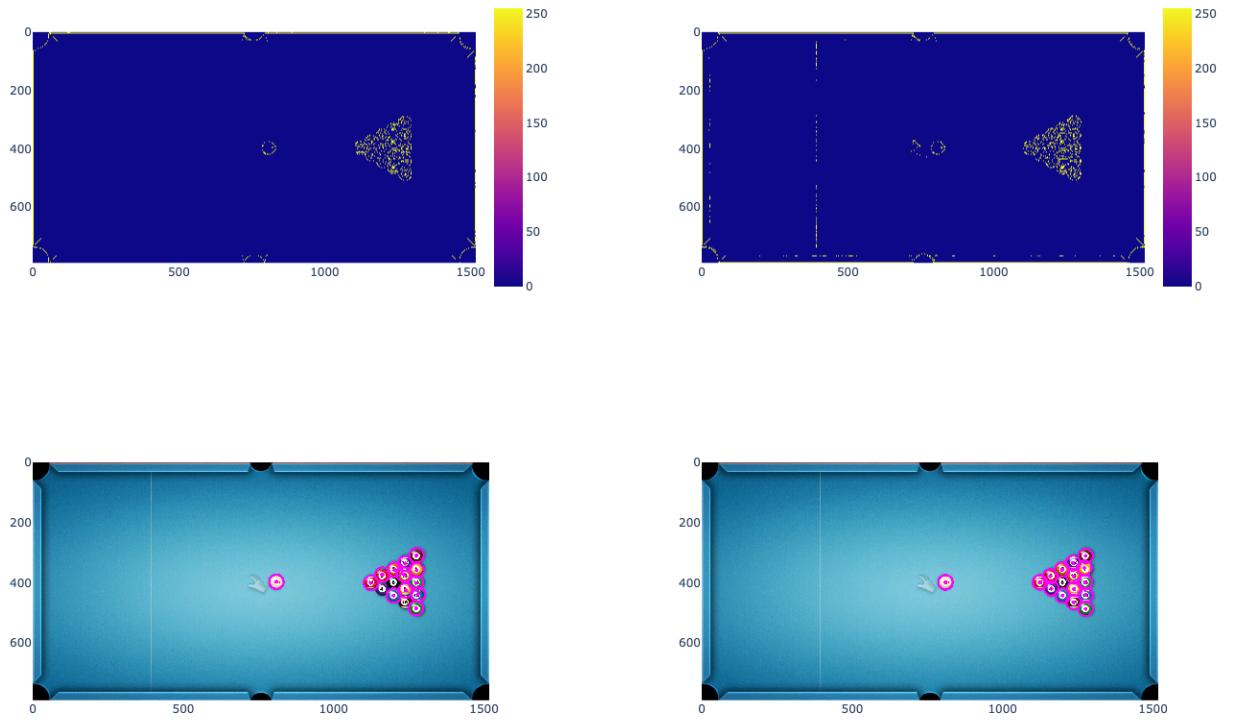
Tham số cho cv2.HoughCircle:

Các ý nghĩa của p1, p2 đã được đề cập ở phần Circle Hough Transform:

- p1 = 300: vì các viên bi đã có sẵn viền (một phần của trò chơi) nên cần tăng giá trị này so với việc phát hiện các lỗ góc, tương ứng là độ nhạy cho việc phát hiện tăng lên.
- p2 = 11: vì các viên bi có kích cỡ nhỏ hơn lỗ góc, nên giảm giá trị này giúp cho việc phát hiện tốt hơn. Tuy nhiên khi giảm xuống < 11 có thể dẫn đến hiện tượng lệch tâm (thuật toán nhận diện thêm cả vòng tròn đánh số viên bi, thậm chí là vệt sáng do bóng).

Làm sắc

Trước khi thực hiện tìm kiếm các viên bi, chương trình có sử dụng thêm một bước xử lý ảnh: tăng độ sắc cạnh. Kết quả trước và sau khi áp dụng như sau:



Trước khi làm sắc

Sau khi làm sắc

Hình 3: So sánh kết quả trước/sau khi làm sắc

Phân loại dựa trên việc đếm số điểm ảnh

Hệ màu được sử dụng là hệ RGB.

- Đếm số điểm pixel màu trắng:

```

1   white_count = 0
2   for pixel in ball_pixels:
3       is_b_valid = 192 <= pixel[0] <= 255
4       is_g_valid = 192 <= pixel[1] <= 255
5       is_r_valid = 192 <= pixel[2] <= 255
6
7       if is_b_valid and is_g_valid and is_r_valid:
8           white_count += 1
9
10      return white_count

```

- Đếm số điểm pixel màu đen:

```

11     black_count = 0
12     for pixel in ball_pixels:
13         is_b_valid = 0 <= pixel[0] <= 64
14         is_g_valid = 0 <= pixel[1] <= 64
15         is_r_valid = 0 <= pixel[2] <= 64
16
17         if is_b_valid and is_g_valid and is_r_valid:
18             black_count += 1
19
20     return black_count

```

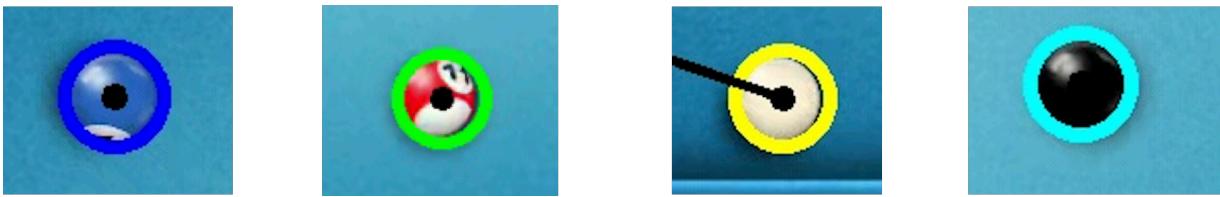
- Tổng điểm ảnh total_count = số lượng điểm ảnh trong phạm vi đường tròn là viên bi
- Các điểm ảnh màu color_count = total_count - white_count - black_count
- Phân loại bi sọc (striped) và bi trơn (solid)

Kết hợp với các tinh chỉnh trong quá trình thực hiện, đưa ra được công thức hiện tại như sau:

- Solid_ball: color_count / total_count ≥ 0.8
- Stripped_ball: color_count / total_count ≥ 0.2
- Black_ball: black_count / total_count ≥ 0.5
- White_ball: white_count / total_count ≥ 0.89

Viên bi đen (black ball) thực chất là viên bi số 8

Các công thức này chỉ đảm bảo việc nhận biết các viên bi một cách tương đối và chấp nhận được. Hiện vẫn chưa tìm được công thức nào vừa đơn giản vừa đảm bảo độ chính xác cao hơn.



Theo thứ tự từ trái qua phải: bi tròn, bi sọc, bi trắng, bi đen

Output

Tập toạ độ các viên bi và phân loại của nó

```
[(926, 898, <BallColour.Solid: 1>), (1192, 616, <BallColour.White: 4>), (1632, 694, <BallColour.Black: 3>), (1724, 640, <BallColour.Strip: 2>), (1148, 662, <BallColour.Solid: 1>), (1002, 754, <BallColour.Strip: 2>), (1562, 430, <BallColour.Solid: 1>), (852, 662, <BallColour.Solid: 1>), (1726, 346, <BallColour.Solid: 1>), (1606, 580, <BallColour.Solid: 1>), (1600, 376, <BallColour.Solid: 1>), (1662, 596, <BallColour.Solid: 1>), (676, 626, <BallColour.Strip: 2>), (1910, 794, <BallColour.Solid: 1>)]
```

Tìm đường đi

Xây dựng đồ thị

Tập đỉnh của đồ thị là tập các viên bi.

Xây dựng tập các cạnh của đồ thị như sau:

- Duyệt qua các lỗ mục tiêu (các góc trên bàn bi-a) và các viên bi mục tiêu.
- Đối với cặp bi mục tiêu và lỗ, tính toán vị trí đánh trúng bi. Đó là vị trí mà bi trắng nên đánh trúng vào bi mục tiêu để ăn điểm.
- Nếu tìm thấy vị trí đánh trúng bi mục tiêu hợp lệ (đường đến vị trí đánh trúng không bị chắn bởi bất kỳ viên bi nào khác, hoặc cạnh bàn), nó sẽ kiểm tra xem cú đánh có khả thi hay không.
- Nếu cú đánh khả thi, nó sẽ thêm một cạnh vào đồ thị từ quả bi trắng đến vị trí đánh trúng bi mục tiêu và từ bi mục tiêu đến lỗ mục tiêu. Trọng số của cạnh là khoảng cách giữa các viên bi
- Nếu cú đánh không khả thi, nó sẽ thêm một cạnh vào đồ thị từ bi trắng đến bi mục tiêu. Trọng số của cạnh là khoảng cách giữa bi trắng và bi mục tiêu +

hàng số là 10000 (một giá trị đặc biệt lớn). Điều này khiến các đường đi không dẫn đến lỗ ít được ưu tiên hơn khi tìm đường ngắn nhất trong đồ thị.

Ở mỗi khung hình, thuật toán Dijkstra được thực hiện với điểm xuất phát là bi trắng và đỉnh kết lần lượt là 6 lỗ góc.

Kiểm tra đường đi hợp lệ

Kiểm tra xem vị trí của bi trắng có nằm giữa bi mục tiêu và lỗ hay không.

Công thức như sau:

- Cận dưới và cận trên của bi mục tiêu: target_ball - ball_diameter, target_ball + ball_diameter
- Cận trên và dưới của lỗ: target_hole - ball_diameter, target_hole + ball_diameter

Toạ độ (x, y) của bi trắng thuộc trong các cận hay không:

- Theo trục x: ($\text{lower_target_ball_x} < \text{white_x} < \text{upper_target_hole_x}$)
or ($\text{lower_target_hole_x} < \text{white_x} < \text{upper_target_ball_x}$)
- Theo trục y: ($\text{lower_target_ball_y} < \text{white_y} < \text{upper_target_hole_y}$)
or ($\text{lower_target_hole_y} < \text{white_y} < \text{upper_target_ball_y}$)

Và nếu một trong 2 điều kiện trên đúng \Rightarrow đường đi là không hợp lệ.

Dijkstra để tìm đường đi

Input

- Đỉnh xuất phát
- Đỉnh kết thúc

Ý tưởng thực hiện thuật toán

1. Khởi tạo từ điển `shortest_paths` với cặp khoá-giá trị là đỉnh và trọng số.
Bắt đầu với (None, 0) tương ứng: (đỉnh đã đi qua, tổng trọng số của đường đi)

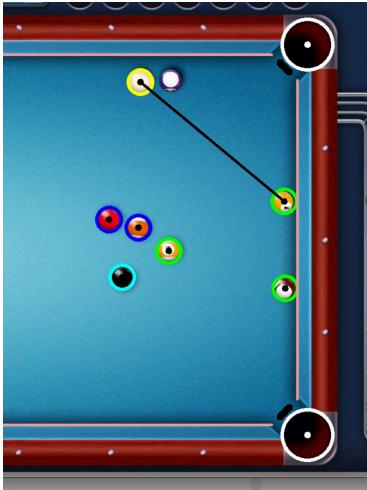
2. Tập `visited` để đánh dấu các đỉnh đã đi qua
3. Thực hiện vòng lặp:
 - (a) Thêm đỉnh hiện tại vào `visited`
 - (b) Duyệt tất cả các đỉnh lân cận (có cạnh với đỉnh hiện tại):
 - Lấy ra tổng trọng số của đỉnh hiện tại từ `shortest_paths`
 - Nếu tổng trọng số mới bé hơn giá trị hiện tại thì cập nhật lại tổng trọng số cho đỉnh lân cận từ vị trí hiện tại,
4. Sau khi kết thúc vòng lặp, xây dựng đường đi ngắn nhất bằng cách duyệt ngược từ đỉnh cuối đến đỉnh bắt đầu. Kết quả là danh sách đường đi sau khi đảo ngược lại.

Output

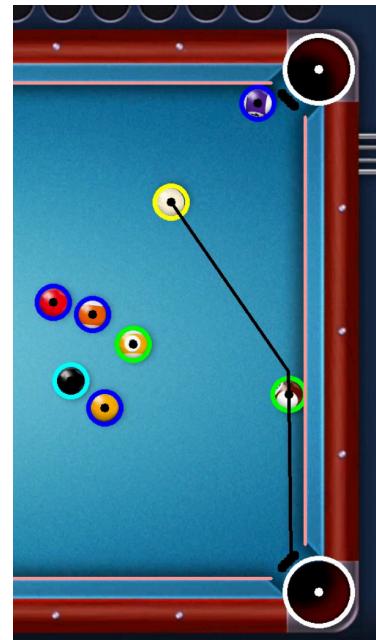
- Danh sách tập đỉnh (2 hoặc 3)

Kết quả lý tưởng

Khi tìm được đường đi, các đường màu hồng được vẽ để khoanh vùng khu vực bàn chơi. Trước cửa mỗi lỗ góc có một vùng màu đen, là điểm kết thúc của đường đi dùng cho thuật toán Dijkstra.

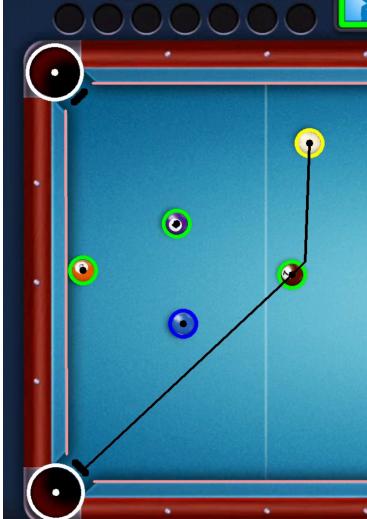


Tìm thấy viên bi gần nhất mà không thấy lỗ
mục tiêu



Tìm thấy bi gần nhất và đường đến lỗ mục
tiêu gần nhất

Hình 4: Mục tiêu ở cạnh bàn



Trường hợp điển hình 1



Trường hợp điển hình 2

Hình 5: Các trường hợp điển hình

Hạn chế

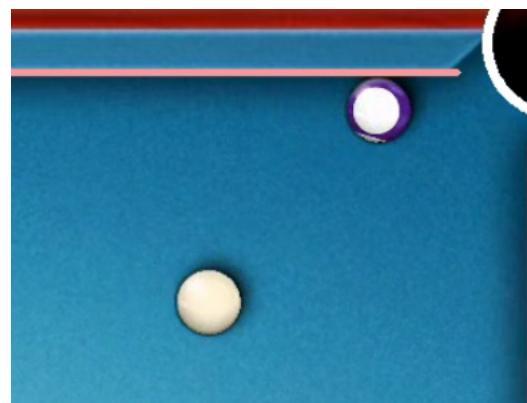
Tổng thể

- Tốc độ xử lý chưa đủ nhanh, dù một số phần tính toán với các vector được thêm thư viện numba, thể hiện ở video kết quả
- Độ chính xác vẫn chưa được như kì vọng, hiện tượng nhận diện nhầm xảy ra khá nhiều.
- Nếu các viên bi nằm ở trước cửa lỗ thì không nhận diện được.
- Phụ thuộc vào kích thước của video (kích thước ảnh) đầu vào. Bắt buộc phải nhập các tham số ứng với các kích thước khác nhau.

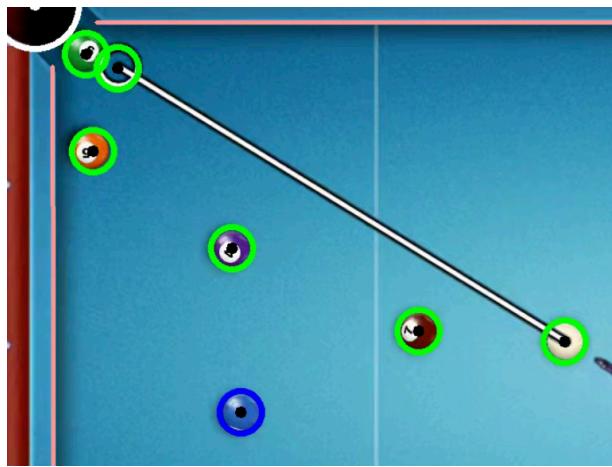
Các hình ảnh cho việc nhận diện sai



Hình 6: Nhận diện sai loại



Hình 7: Không nhận dạng được viên bi



Hình 8: Nhận dạng sai bi trắng và nhận diện nhầm phần đầu thành bi

Kết quả

Một số hình ảnh lý tưởng thu được từ quá trình chạy, độ chính xác không quá phụ thuộc vào số bi còn lại trên bàn.



Hình 9: Thấy bi gần nhất nhưng không thấy lỗ



Hình 10: Tìm được bi gần nhất và đường đến lỗ gần nhất

Hướng phát triển

Vì gốc rễ của dự án này có mã nguồn mở, vậy nên dự án này sẽ được công khai mã nguồn để hướng tới việc phát triển hơn.

Các hướng có thể cân nhắc:

- Sử dụng Xác suất để phân loại chính xác các loại bi
- Sử dụng các thuật toán tìm đường đi có Heuristic để tăng tốc độ tìm kiếm
- Áp dụng các công thức phức tạp trong bi-a để giống với thực tế hơn.

Link git của dự án: <https://github.com/kidclone3/8-Ball-Pool-Analysis>

Tài liệu tham khảo

- [1] Wikipedia. *Circle Hough Transform*. https://en.wikipedia.org/wiki/Circle_Hough_Transform
- [2] Brandon Abela. *8 Ball Pool Analysis*. Open Source. <https://github.com/brandonabela/8-Ball-Pool-Analysis>
- [3] Stack Overflow. *Detecting Balls on a Pool Table (Stripes and Solids)*. <https://stackoverflow.com/questions/51792919/detecting-balls-on-a-pool-table-stripes-and-solids>
- [4] Stack Overflow. *Improving Flood Fill Result Under Sensible Light Conditions with OpenCV*. <https://stackoverflow.com/questions/67123784/improving-flood-fill-result-under-sensible-light-conditions-with-opencv>
- [5] OpenCV Documentation. *OpenCV - Tutorial: Hough Circle Detection*. https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html
- [6] Prishita Kapoor. *Circle Detection with Hough Circles*. <https://prishitakapoor2.medium.com/circle-detection-with-hough-circles-cc2a0140db88>
- [7] Stack Overflow. *How Can I Sharpen an Image in OpenCV?*. <https://stackoverflow.com/questions/4993082/how-can-i-sharpen-an-image-in-opencv>