ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Bài Báo Cáo Giữa Kỳ Trí Tuệ Nhân Tạo
Đề tài
Lập trình A.I nhận dạng đường đi trong Game

Giáo viên hướng dẫn: Nguyễn Đình Hiển
Sinh viên thực hiện:
+ Nguyễn Hoàng Trung MSSV: 17521176
+ Lê Thanh Phước Hiếu MSSV: 17520474
+ Lê Hoàng Ân MSSV: 17520208
+ Nguyễn Chí Bảo MSSV: 17520271

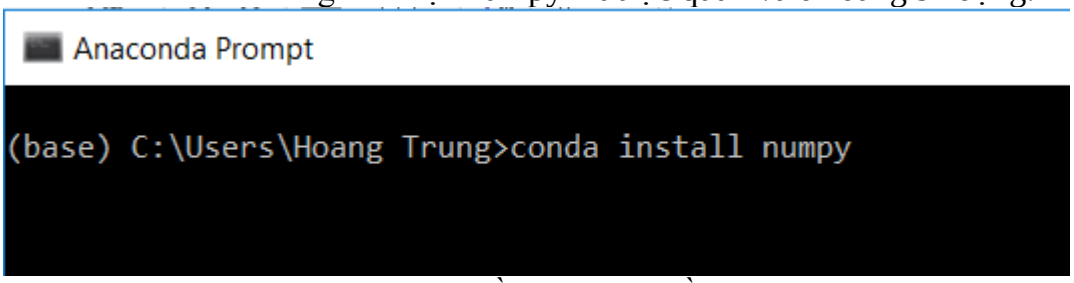
1. Bài Toán

Trong cuộc sống hiện nay, trí tuệ nhân tạo (A.I) xuất hiện ở khắp mọi nơi, và trò chơi điện tử (game) cũng không ngoại lệ. Ở mỗi trò chơi đều có các NPC (Non-Player Character) tức là những nhân vật được nhà phát triển thổi hồn vào chúng, cho chúng khả năng di chuyển, cử động. Các NPC còn có thể di chuyển cùng với người chơi như một người bạn đồng hành, và thuật toán thông dụng nhất được các nhà phát triển game áp dụng để NPC tìm đường đi chính là A*. Sau khi tìm hiểu, chúng em muốn tiếp cận vấn đề này theo một hướng khác, để thử nghiệm xta liệu có thể tìm được giải pháp tối ưu hơn để giải quyết vấn đề di chuyển của NPC hay không, đồng thời cũng là một cách để chúng em ôn tập lại những kiến thức đã được học ở trường và tìm tòi thêm những kiến thức mới.

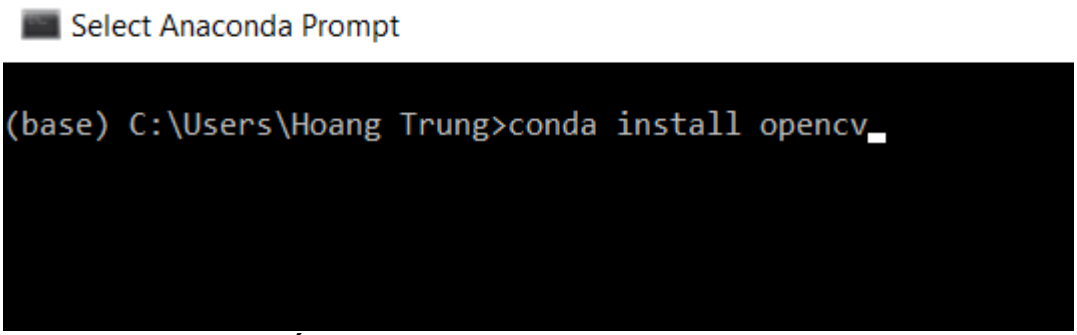
2. Các Công Cụ Hỗ Trợ

Các thư viện chính được nhóm chúng em sử dụng bao gồm:

+ Numpy (viết tắt của Nummerical Python) là một thư viện không thể thiếu khi chúng ta xây dựng các ứng dụng Máy học trên Python. Numpy cung cấp công cụ hỗ trợ mạnh mẽ cho việc xử lý mảng N-chiều; các phép toán trên ma trận; các thuật toán về đại số tuyến tính, phép biến đổi Fourier và khả năng tạo số ngẫu nhiên. Numpy có thể được cài đặt đơn giản thông qua SciPy distribution hoặc sử dụng lệnh từ Anaconda console như hình dưới. Các hàm trong thư viện numpy rất trực quan và dễ dàng sử dụng.



+ OpenCV: là một thư viện mã nguồn mở hàng đầu cho thị giác máy tính (computer vision), xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực. OpenCV cung cấp hơn 2500 các thuật toán đã được tối ưu, bao gồm các thuật toán cổ điển lẫn hiện đại nhất về Computer Vision và Machine Learning. OpenCV được nhiều công ty công nghệ lẫn start-up vì tính đơn giản và hỗ trợ mạnh mẽ của nó. OpenCV có thể được cài đặt thông qua dễ dàng tương tự Numpy vậy.



3.Ý Tưởng Giải Quyết

Thuật ngữ FPS được dùng để diễn tả số khung hình (frame) mà trò chơi đang diễn tả trong 1 giây. Ở mỗi frame cảnh vật trong game sẽ chuyển động từng chút, và tập hợp 60 frames/s sẽ cho khung cảnh trong game chuyển động mượt mà trong mắt người chơi. Dựa vào điều này, nhóm chúng em đã đưa ra ý tưởng để tiếp cận vấn đề như sau:

- Dùng hàm grab_screen thuộc module grabscreen, module này sử dung cách thư viện win32 trên nền tảng Python nên xử lý nhanh hơn so với dùng hàm ImageGrab của thư viện Pillow, để “chụp” lại hình ảnh của frame mỗi 1/60s (giả sử game đang chạy

với tốc độ 60 frame/s). Hàm grab_screen sẽ trả về một ma trận một chiều của frame game vừa chụp được.

-

```
screen = grab_screen(region=(0,40,800,640))
```

**Ở đây chúng ta đang thiết lập cho game chạy ở độ phân giải 800x640, mục đích là để tiện việc theo dõi quá trình nhận diện làn đường của A.I, đồng thời để có thể vừa chạy game vừa chạy code một cách mượt mà.*

- Phân tích trong frame đó, chúng ta dùng thuật toán Canny để nhận diện các cạnh có trong ảnh (Edge Detection). Do ở đây chúng ta cần tìm những đoạn thẳng có khả năng là 2 bên làn đường, nên giá trị threshold1 sẽ được đặt ở mức khá cao, nhằm loại bỏ các pixel không liên quan.

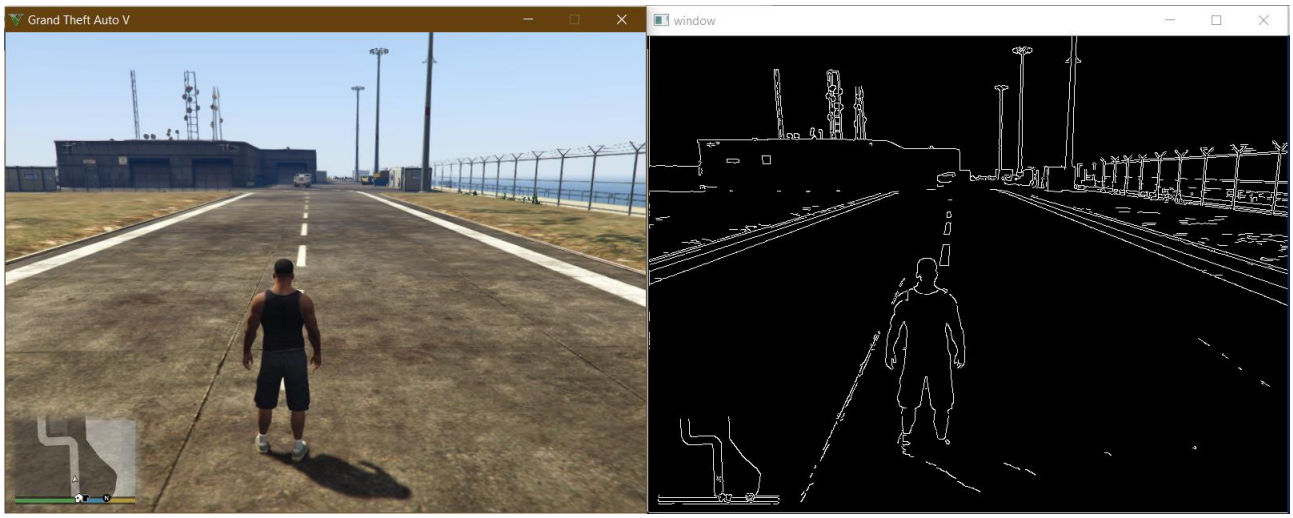
```
processed_img =  cv2.Canny(image, threshold1 = 200, threshold2=300)
```

Xác định cạnh theo điều kiện bằng giải thuật Canny.

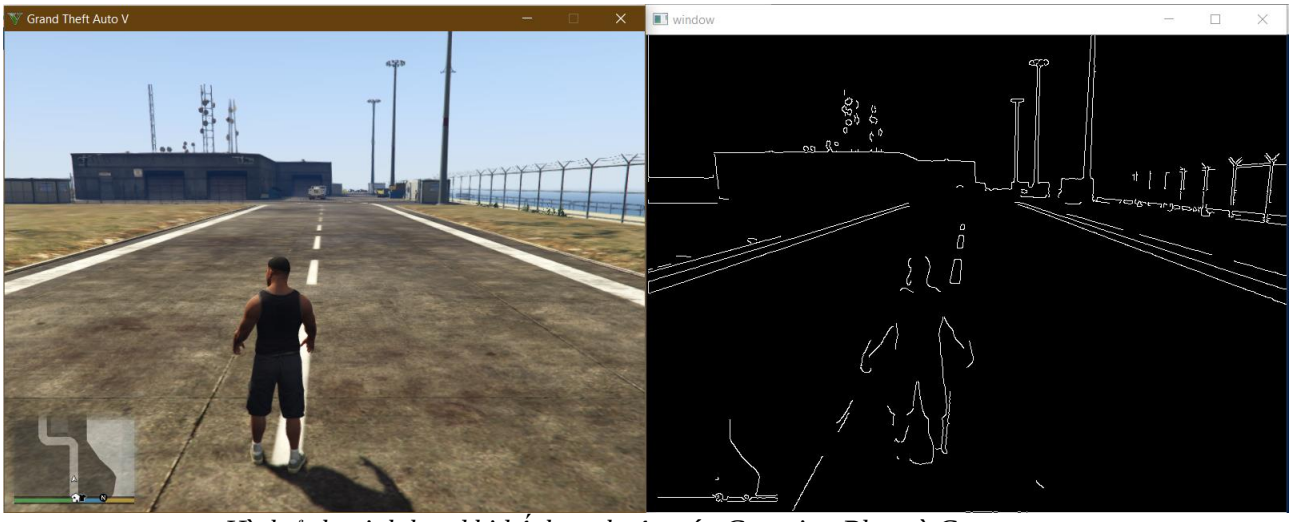
Tuy nhiên, để tăng độ chính xác của thuật toán Canny, chúng ta làm mờ bức ảnh (Blur) trước bằng thuật toán Gaussian Blur, được gọi thông qua hàm GaussianBlur() trong thư viện OpenCV2, sau đó mới thực hiện xác định các cạnh (edge detection) có trong frame. Kernel được áp dụng có kích thước là 5x5.

```
processed_img = cv2.GaussianBlur(processed_img,(5,5),0)
processed_img =  cv2.Canny(image, threshold1 = 200, threshold2=300)
```

Kết hợp cả hai thuật toán để cho ra kết quả tốt.

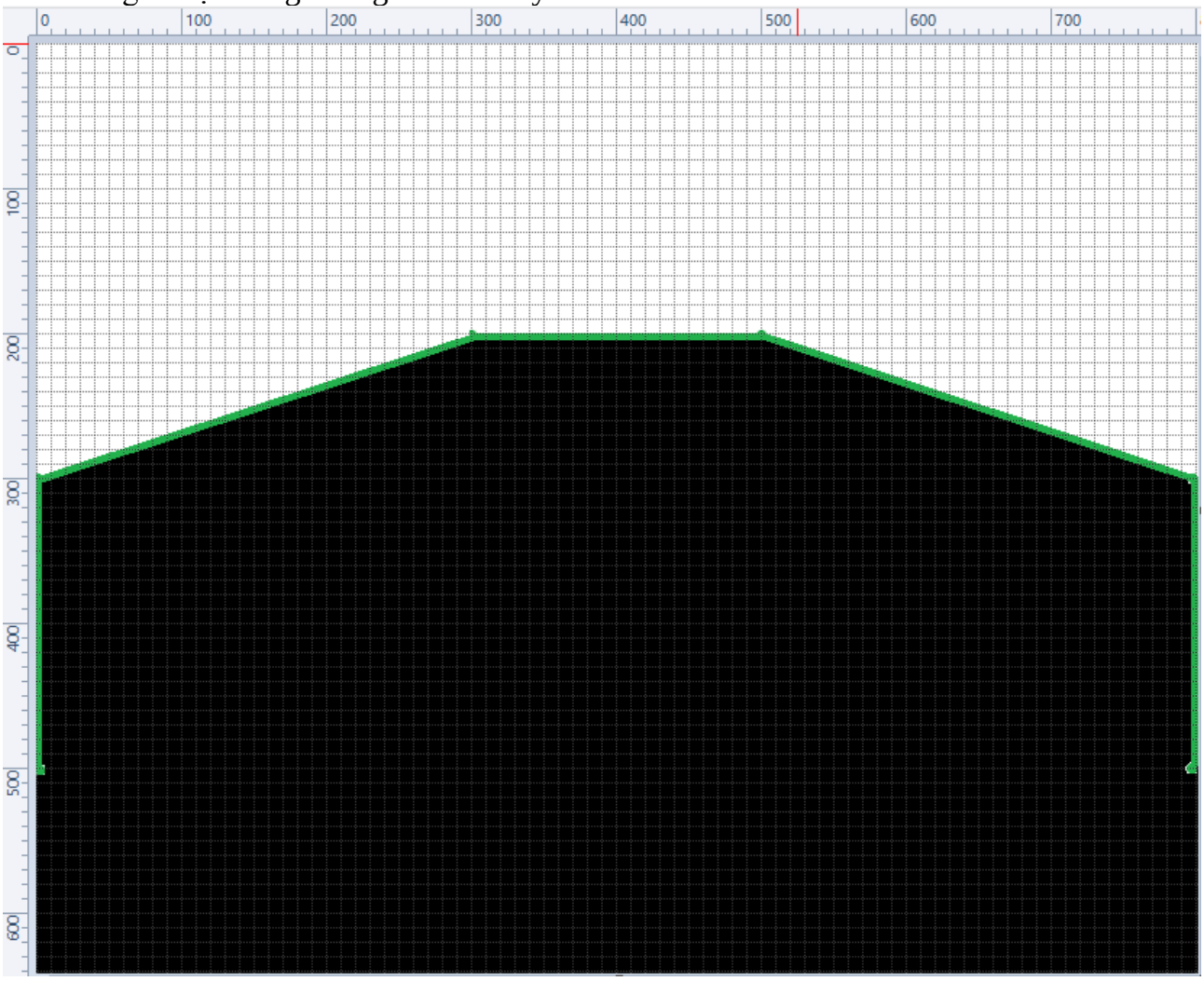


Hình ảnh minh họa khi chỉ sử dụng thuật toán Canny.



Hình ảnh minh họa khi kết hợp thuật toán Gaussian Blur và Canny.

- Xác định vùng quan tâm (Region Of Interest – ROI). Chúng ta đang lập trình cho A.I đi trên đường bộ, nên mục tiêu cần xác định chính là 2 bên làn đường. Do đó, chúng ta xem hình ảnh con đường sẽ chỉ xuất hiện trong 1 vùng hình thang như hình dưới, từ đó giới hạn vùng chúng ta cần xử lý.



Hình ảnh vùng ROI

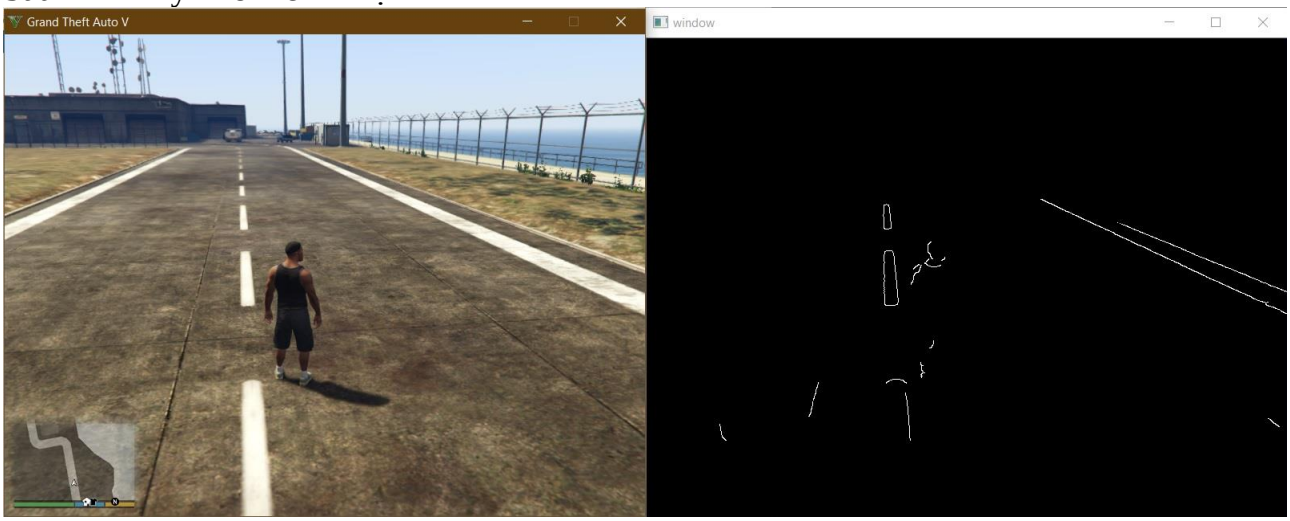
```python
def roi(img, vertices):

    #blank mask:
    mask = np.zeros_like(img)

    #filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, vertices, 255)

    #returning the image only where mask pixels are nonzero
    masked = cv2.bitwise_and(img, mask)
    return masked
```

Đánh dấu và xuất ra hình ảnh chỉ có những pixel đã được đánh dấu.

```python
vertices = np.array([[0,500],[0,300],[300,200],[500,200],[800,300],[800,500],
                     ], np.int32)

processed_img = roi(processed_img, [vertices])
```

Tọa độ các điểm dùng để xác định ROI.

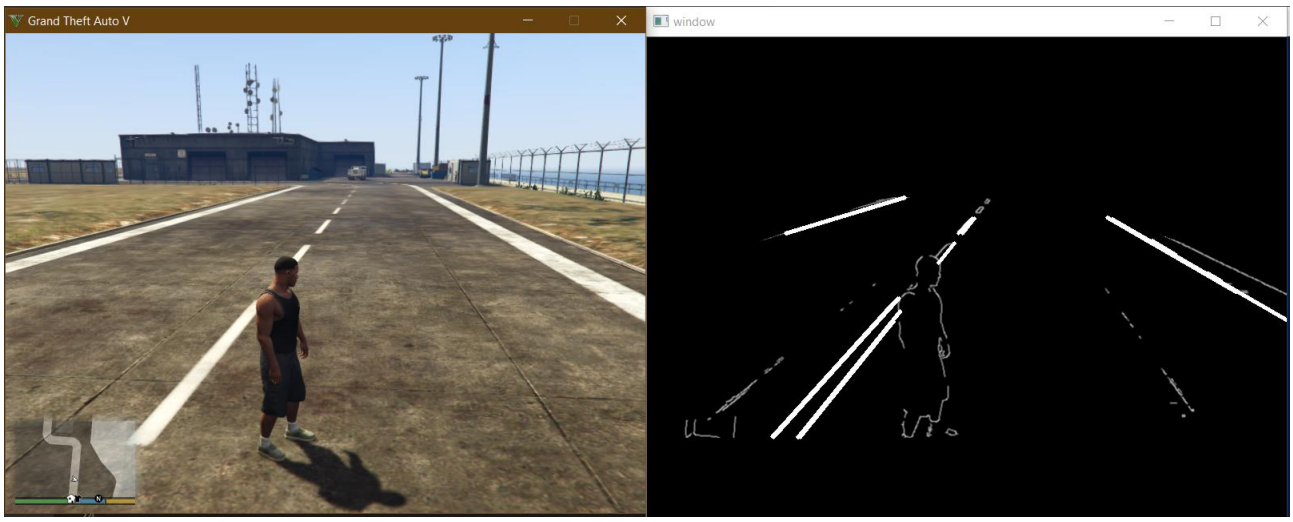
Sau khi xử lý nhóm em được như sau:



- Để làm cho kết quả chính xác hơn, chúng ta dùng Probabilistic Hough Line Transform được gọi thông qua hàm HoughLinesP() của OpenCV để lọc lại những đoạn thẳng phù hợp với yêu cầu, loại bỏ các cạnh quá ngắn hoặc đứt khúc (khả năng cao không phải là 2 bên lề đường).

```python
lines = cv2.HoughLinesP(processed_img, 1, np.pi/180, 180, 20, 15)
```

Tìm ra các đường thẳng thỏa yêu cầu với thuật toán HoughLinesP



- Tiếp theo chúng ta xây dựng 1 hàm để xác định đâu là 2 bên đường, hàm này sẽ trả về cặp giá trị (x1,y1), (x2,y2) thuộc làn đường. dựa vào đó chúng ta sẽ xấp sỉ làn đường cong thành những đường thẳng. Bên cạnh đó, hàm này cũng trả về độ dốc (lane_id) của 2 làn đường.

```python
def draw_lanes(img, lines, color=[0, 255, 255], thickness=3):

    # if this fails, go with some default line
    try:

        # finds the maximum y value for a lane marker
        # (since we cannot assume the horizon will always be at the same point.)

        ys = []
        for i in lines:
            for ii in i:
                ys += [ii[1],ii[3]]
        min_y = min(ys)
        max_y = 600
        line_dict = {}
        #idx = count, i = elements
        #Tìm phương trình y = mx + b dựa vào các điểm x, y đã có bằng cách tính tập vector x minimize Euclide norm2
        #https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.lstsq.html

        for idx,i in enumerate(lines):
            for xyxy in i:
                # Used to calculate the definition of a line, given two sets of coords.
                x_coords = (xyxy[0],xyxy[2])
                y_coords = (xyxy[1],xyxy[3])
                A = vstack([x_coords,ones(len(x_coords))]).T
                m, b = lstsq(A, y_coords)[0]

                # Calculating our new, and improved, xs
                x1 = (min_y-b) / m
                x2 = (max_y-b) / m

                line_dict[idx] = [m,b,[int(x1), min_y, int(x2), max_y]]
```

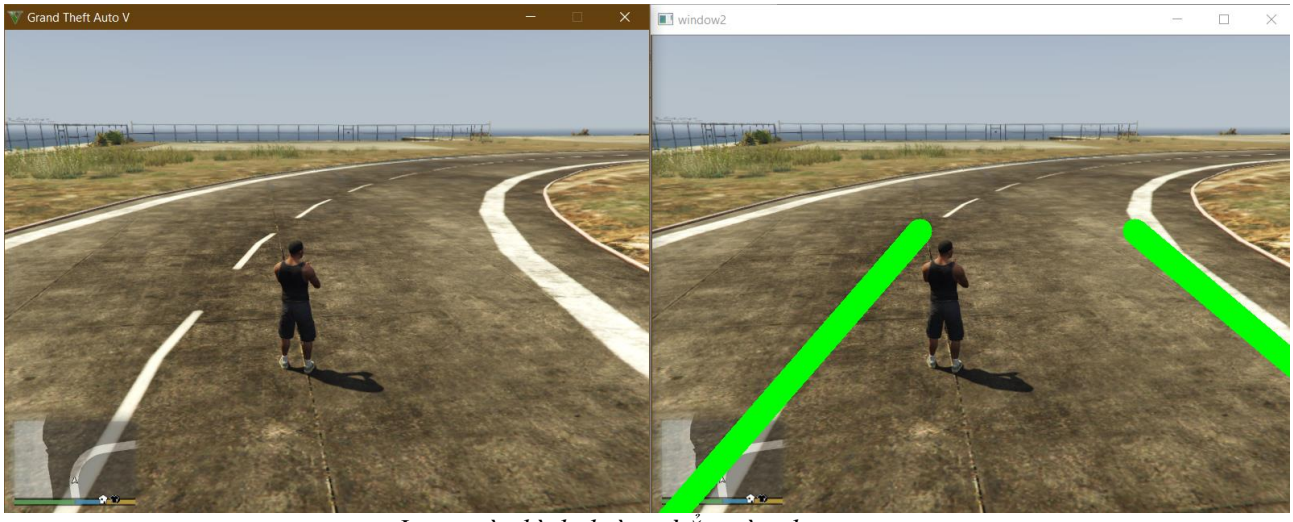
Tìm thông số m,b từ cặp (x,y).

```
for other_ms in final_lanes_copy:

    if not found_copy:
        if abs(other_ms*1.2) > abs(m) > abs(other_ms*0.8):
            if abs(final_lanes_copy[other_ms][0][1]*1.2) > abs(b) > abs(final_lanes_copy[other_ms][0][1]*0.8):
                final_lanes[other_ms].append([m,b,line])
                found_copy = True
                break
        else:
            final_lanes[m] = [ [m,b,line] ]
```

Gom nhóm những đường thẳng có độ dốc (m) gần giống nhau.

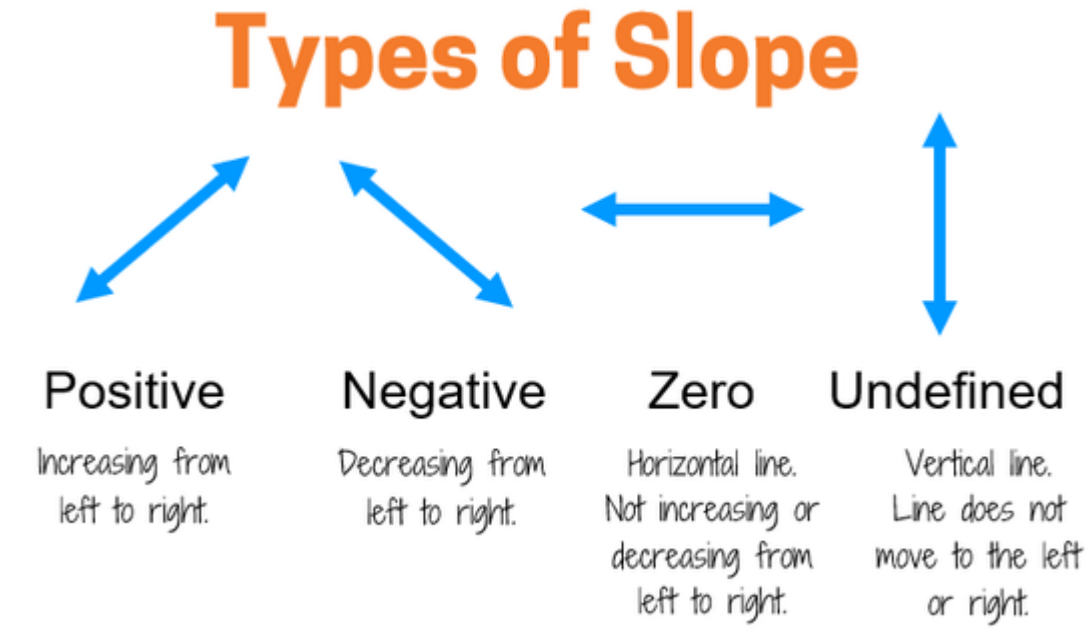
```
def average_lane(lane_data):
    x1s = []
    y1s = []
    x2s = []
    y2s = []
    for data in lane_data:
        x1s.append(data[2][0])
        y1s.append(data[2][1])
        x2s.append(data[2][2])
        y2s.append(data[2][3])
    return int(mean(x1s)), int(mean(y1s)), int(mean(x2s)), int(mean(y2s))

l1_x1, l1_y1, l1_x2, l1_y2 = average_lane(final_lanes[lane1_id])
l2_x1, l2_y1, l2_x2, l2_y2 = average_lane(final_lanes[lane2_id])
# chọn ra những lanes có cùng slope, lấy giá trị trung bình để chọn ra đường thẳng cuối cùng
return [l1_x1, l1_y1, l1_x2, l1_y2], [l2_x1, l2_y1, l2_x2, l2_y2], lane1_id, lane2_id
```

Lấy giá trị trung bình để ra được đường thẳng cần tìm.



In ra màn hình đường thẳng tìm được.

- Dựa vào độ dốc của 2 đường thẳng mà chương trình suy ra được mà A.I sẽ quyết định xem sẽ đi thẳng, quẹo trái hay quẹo phải.



Trên hình là hình ảnh độ dốc của 1 đoạn thẳng trong không gian 2D, tuy nhiên, hệ tọa độ trong OpenCV lại có chiều của trục Oy hướng xuống, ngược lại với hệ tọa độ Oxy chúng ta thường sử dụng nên khi độ dốc < 0 A.I sẽ quẹo phải và tương tự. Còn lại A.I vẫn sẽ đi thẳng.

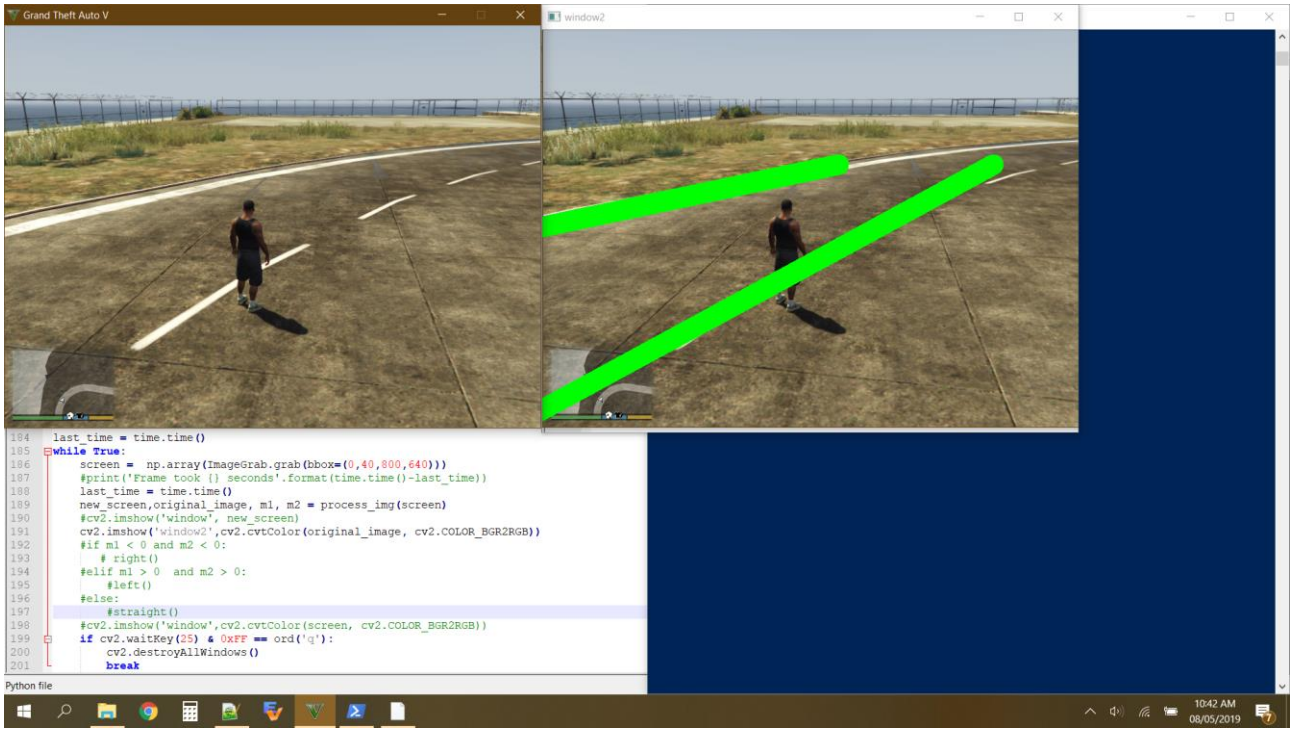
```
if m1 < 0 and m2 < 0:
    right()
elif m1 > 0  and m2 > 0:
    left()
else:
    straight()
```

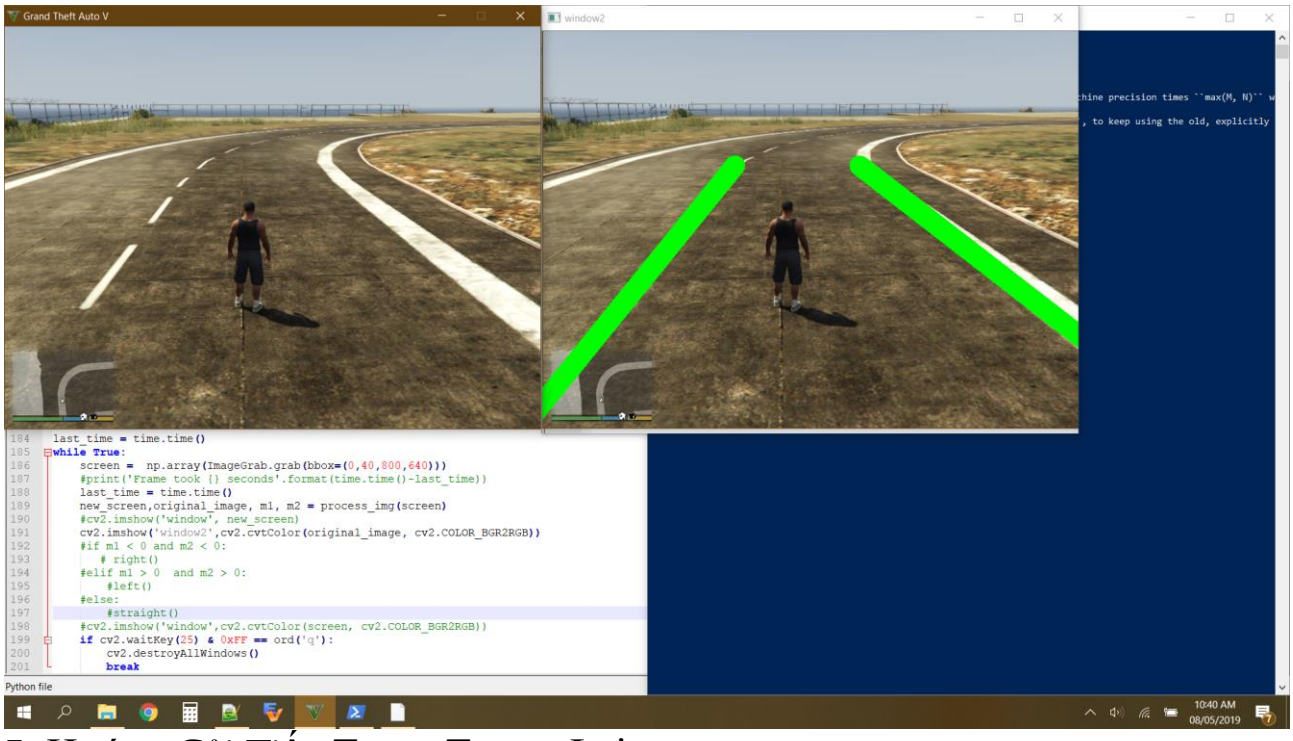
Để thực hiện việc đi thẳng hay quẹo chỉ đơn giản là việc kết hợp giữa việc nhấn và thả các nút di chuyển:

```
def straight():
    PressKey(W)
    ReleaseKey(A)
    ReleaseKey(D)

def left():
    PressKey(A)

def right():
    PressKey(D)
```

4. Kết quả thu được

A.I hiện tại đã có thể di chuyển mượt mà trên đường, có thể quẹo trái, quẹo phải và đi thẳng. Tuy nhiên, do việc xử lý đang thực hiện trên CPU nên cần một phần cứng đủ mạnh để chạy song song với trò chơi. Do cấu hình máy của nhóm đang dùng bị hạn chế nên hiện chỉ có thể cho A.I đi bộ mới cho kết quả chính xác được, vì khi dùng các phương tiện giao thông khác sẽ khiến tốc độ di chuyển tăng, CPU không thể xử lý kịp.





5. Hướng Cải Tiến Trong Tương Lai

Hiện nhóm chúng ta đã thảo luận và đề xuất ra hai hướng để cải thiện tốc độ xử lý của chương trình. Hướng đầu tiên đó là lập trình song song, sử dụng Thread để chia khung hình ra làm 4 phần, xử lý từng phần song song với nhau. Hướng thứ hai là sử dụng phương pháp GPGPU (General-purpose computing on graphics processing units). Ở phương pháp này chúng ta sẽ chuyển tất cả các tác vụ tính toán qua cho GPU xử lý, CPU sẽ chỉ đảm nhiệm việc điều khiển nhân vật. Do chuyên môn chính của GPU là xử lý các tác vụ về hình ảnh, tính toán đồ họa nên chắc chắn tốc độ nhận diện sẽ nhanh hơn so với khi làm việc trên CPU.

6. Tài liệu tham khảo

- https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

- https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
- https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#gaussianblur
- https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html
- https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
- https://www.programcreek.com/python/example/89434/cv2.HoughLinesP
- https://minhng.info/tutorials/opencv-region-of-interest-roi-la-gi.html
- https://www.learnopencv.com/how-to-select-a-bounding-box-roi-in-opencv-cpp-python/
- https://stackoverflow.com/questions/21565994/method-to-return-the-equation-of-a-straight-line-given-two-points
- https://github.com/mhammond/pywin32
- http://timgolden.me.uk/pywin32-docs/contents.html
- https://stackoverflow.com/questions/19695214/python-screenshot-of-inactive-window-printwindow-win32gui
- https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.lstsq.html