ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO
ĐỒ ÁN MÔN HỌC THIẾT KẾ LUẬN LÝ

ỨNG DỤNG PHÁT HIỆN VẠCH KẺ ĐƯỜNG, VẠCH KẺ Ô ĐỖ XE, VẠCH TRẮNG CHỈ ĐƯỜNG,... THÔNG QUA HÌNH ẢNH THU ĐƯỢC TỪ CAMERA

Ngành: Kỹ Thuật Máy Tính

HỘI ĐỒNG: Đồ án thiết kế luận lý 5
GVHD: Huỳnh Phúc Nghị
TKHĐ: Huỳnh Hoàng Kha

---o0o---

SVTH 1: Phạm Duy Quang – 2011899
SVTH 2: Hà Vĩnh Nguyên – 2011698
SVTH 3: Đỗ Thành Minh – 2011610
SVTH 4: Phạm Đức Thắng – 2012080

TP. HỒ CHÍ MINH, 01/2023

Xác nhận của sinh viên	Xác nhận của GVHD

STT	Sinh viên thực hiện	Khối lượng công việc
1	Phạm Duy Quang	Tìm hiểu và đề xuất các giải thuật; tìm hiểu các thiết bị phần cứng phù hợp với đề tài; hiện thực, hoàn thiện và kiểm thử các giải thuật trên phần mềm và phần cứng; viết báo cáo về các bước hiện thực và tổng hợp báo cáo
2	Hà Vĩnh Nguyên	Hỗ trợ hiện thực, chỉnh sửa và hoàn thiện giải thuật trên phần mềm và phần cứng
3	Đỗ Thành Minh	Hỗ trợ tìm hiểu và viết báo cáo về giải thuật; tìm hiểu các thiết bị phần cứng cho đề tài
4	Phạm Đức Thắng	Hỗ trợ tìm hiểu và viết báo cáo về giải thuật; tìm hiểu các thiết bị phần cứng cho đề tài

TP. HỒ CHÍ MINH, 01/2023

MỤC LỤC

PHẦN 1: BÁO CÁO GIAI ĐOẠN 1 – TIẾP CẬN VẤN ĐỀ

I. Báo cáo giai đoạn 1

Link Github về Project của nhóm 3 trong giai đoạn 1: https://github.com/ULTIMATE-Mystery/Logic-Design-Project-Group-3-Semester-221-HCMUT/tree/Phase-1.

1. Nội dung đã làm được trong giai đoạn vừa rồi là gì?

- Phân công, chọn các giải thuật cần thiết để hiện thực project.
- Tìm hiểu các lý thuyết cơ bản về giải thuật phát hiện cạnh/góc như:
  - Gaussian blur / Bilateral filter
  - Canny Edge Detection
  - Hough Line Transform
- Tìm hiểu, học hỏi các project có đề tài liên quan đến phát hiện vạch kẻ đường, vạch kẻ ô đỗ xe, vạch trắng chỉ đường,... thông qua hình ảnh thu được từ camera.
- Tìm hiểu, thảo luận các thiết bị phần cứng được đề xuất để hiện thực project.
- Trao đổi, thực nghiệm các giải thuật nêu trên để xử lý một số ảnh ví dụ từ đó rút ra sai sót và tiến hành chỉnh sửa.

2. Nội dung dự định làm trong giai đoạn tiếp theo là gì?

- Hoàn thiện thêm giải thuật của nhóm để loại bỏ những lỗi sai đang có.
- Từng bước xây dựng sơ đồ khối, hoàn thiện các khối xử lý từ đó tiến đến hiện thực phần cứng.
- Hiện thực phần cứng từ các thiết bị đã được bàn bạc và chọn lựa.
- Kiểm thử phần cứng và tiến hành chỉnh sửa.

3. Khó khăn nào gặp phải trong giai đoạn vừa rồi?

- Khó khăn về nội dung mới lạ, phức tạp của các giải thuật phát hiện cạnh/góc từ hình ảnh.

- Khó khăn trong việc bàn bạc, quyết định chọn lựa phần cứng để từ đó lựa chọn phương pháp, nền tảng hiện thực giải thuật.
- Khó khăn trong việc hiện thực, xây dựng các giải thuật trên phần mềm.

II. Nội dung báo cáo

1. Gaussian blur

1.1. Giới thiệu

Xử lý ảnh ở giai đoạn đầu giúp phân biệt các điểm nổi bật trong ảnh, có vai trò quan trọng trong việc đánh giá cấu trúc và đặc tính của các mục trong một cảnh. Cạnh là một trong những đặc tính đó, cũng thành phần quan trọng để đánh giá hình ảnh. Vì vậy trong đề tài này nhóm sử dụng thuật toán Gaussian blur làm mờ ảnh để loại bỏ nhiễu và tăng độ chính xác khi tìm cạnh.

1.2. Lý thuyết

Trong toán học, việc ứng dụng Gaussian Blur cho một hình cũng chính là tính tích chập (Convolution) hình đó với hàm Gaussian.

Vì phép biến đổi Fourier của một Gaussian là một Gaussian khác, việc áp dụng Gaussian blur có tác dụng làm giảm các thành phần tần số cao của hình ảnh.

Công thức của hàm Gaussian một chiều:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Trong hai chiều, nó là tích của hai hàm Gaussian như vậy, một trong mỗi chiều:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Trong đó x là khoảng cách từ điểm gốc theo trục hoành, y là khoảng cách từ điểm gốc theo trục tung và σ là độ lệch chuẩn của phân bố Gauss. Điểm gốc nằm ở tâm (0, 0). Khi được áp dụng trong hai chiều, công thức này tạo ra một bề mặt có các đường bao là các vòng tròn đồng tâm với phân bố Gauss từ tâm điểm.

Các giá trị từ phân phối này được sử dụng để xây dựng một ma trận tích chập được áp dụng cho hình ảnh ban đầu. Giá trị mới của mỗi pixel được đặt thành giá trị trung

bình trọng số của vùng lân cận pixel đó. Trọng số của sự phụ thuộc được lấy theo hàm Gauss (cũng được sử dụng trong quy luật phân phối chuẩn). Giá trị của pixel gốc nhận được trọng lượng nặng nhất (có giá trị Gaussian cao nhất) và các pixel lân cận nhận được trọng số nhỏ hơn khi khoảng cách của chúng đến pixel gốc tăng lên. Điều này dẫn đến hiệu ứng làm mờ bảo toàn ranh giới và các cạnh tốt hơn các bộ lọc làm mờ khác, đồng nhất hơn.

1.3. Tính chất

Về lý thuyết, hàm Gaussian tại mọi điểm trên hình ảnh sẽ khác 0, có nghĩa là toàn bộ hình ảnh sẽ cần được đưa vào các phép tính cho mỗi pixel. Trong thực tế, khi tính toán xấp xỉ rời rạc của hàm Gaussian, các pixel ở khoảng cách lớn hơn 3 σ có ảnh hưởng đủ nhỏ để được coi là 0 một cách hiệu quả. Do đó, đóng góp từ các pixel bên ngoài phạm vi đó có thể bị bỏ qua. Thông thường, một chương trình xử lý ảnh chỉ cần tính toán một ma trận với các kích thước [6σ] x [6σ] ([...] là hàm trần) để đảm bảo một kết quả đủ gần với kết quả thu được của toàn bộ phân phối Gauss.

Ngoài tính chất đối xứng tròn, hiệu ứng mờ Gaussian có thể được áp dụng cho hình ảnh hai chiều dưới dạng hai phép tính một chiều độc lập, và vì vậy được gọi là bộ lọc tách biệt. Có nghĩa là, hiệu quả của việc áp dụng ma trận hai chiều cũng có thể đạt được bằng cách áp dụng một loạt các ma trận Gaussian đơn chiều theo hướng ngang, sau đó lặp lại quá trình theo hướng dọc. Theo thuật ngữ máy tính, đây là một thuộc tính hữu ích, vì phép tính có thể được thực hiện trong thời gian O (w_kernel * w_image * h_image) + O (h_kernel * w_image * h_image), thay vì O (w_kernel * h_kernel * w_image * h_image).

2. Canny Edge Detection

2.1. Ý tưởng

Được chia thành 4 bước:

- Chọn bộ lọc Gaussian để làm mịn hình ảnh (lấy đạo hàm của ảnh theo chiều ngang và dọc dựa trên phân phối Gaussian - Gaussian Smoothing).

- Tính toán cường độ và hướng của gradient hình ảnh (Calculation of the Image Gradient).

- Sử dụng thuật toán Loại bỏ các giá trị không phải cực đại để loại bỏ đi các bounding box dư thừa của cùng một đối tượng trong ảnh (Non-maximum Suppression of the Grad value).

- Sử dụng threshold để tạo loại bỏ cạnh giả, xác định cạnh thực sự (Image Threshold Setting and Edge Connection).

2.2. Lý thuyết

Gaussian Smoothing - giảm nhiễu:

Việc giảm nhiễu, làm mịn ảnh đầu vào là điều cần thiết, giúp giảm sự nhầm lẫn với biên ảnh, và do hầu hết hình ảnh từ máy chụp sẽ chứa lượng nhiễu ảnh nhất định. Bằng cách áp dụng bộ lọc Gauss. Nhân của bộ lọc Gauss với độ lệch chuẩn σ = 1,4 được thể hiện trong phương trình sau (ở đây ta sử dụng một bộ lọc 5 x 5):

$$S = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Calculation of the Image Gradient - Tìm Gradient:

Hình ảnh được làm mịn sau đó được lọc bằng hạt nhân Sobel theo cả hướng ngang và dọc để có được đạo hàm đầu tiên theo hướng ngang (G_x) và hướng dọc (G_y). Từ hai hình ảnh này, chúng ta có thể tìm thấy độ dốc và hướng của cạnh cho mỗi pixel như sau:

$$Edge\ Gradient\ (G) = \sqrt{G_x^2 + G_Y^2}$$

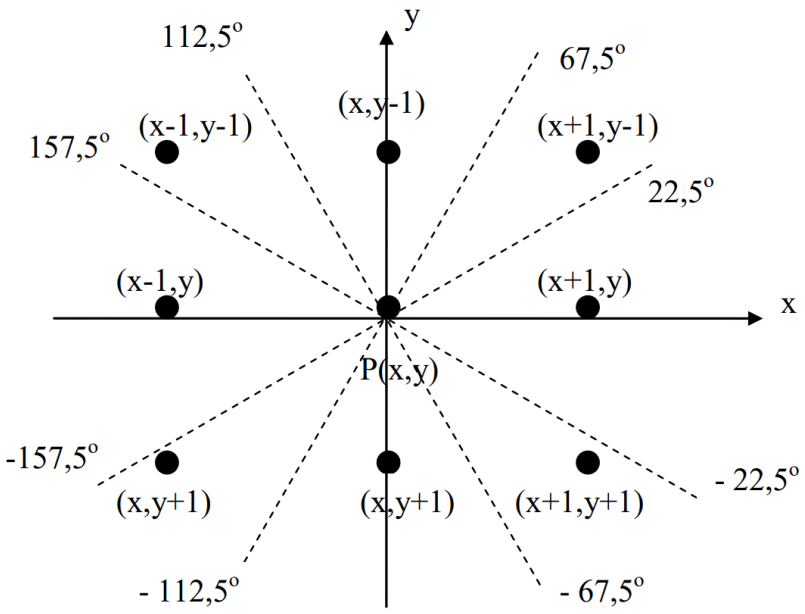
Trong đó G_x và G_y là gradient theo 2 hướng x và y tương ứng và hướng của biên θ như sau:

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Ảnh G tìm được là kết quả của quá trình này.

Non-maximum Suppression of the Grad value

Sau khi nhận được độ lớn và hướng của gradient, quá trình quét toàn bộ hình ảnh sẽ được thực hiện để loại bỏ bất kỳ pixel không mong muốn nào có thể không tạo thành cạnh. Đối với điều này, tại mỗi pixel, pixel được kiểm tra xem nó có phải là cực đại cục bộ trong vùng lân cận của nó theo hướng gradient hay không. Giả sử với điểm biên đang xét tại vị trí P (x, y), ta có 8 điểm biên lân cận điểm biên này:



Hình 2.1: Hình mô tả các điểm biên lân cận của P

Tại điểm biên đó ta tiến hành tính giá trị góc của hướng đường biên θ. Nếu hướng của đường biên $\theta \leq 22.5^o$ hoặc $\theta > 157.5^o$ thì đặt giá trị của $\theta = 0^o$, khi đó hai điểm biên lân cận điểm biên này tại vị trí (x-1, y) và (x+1, y).

Tương tự ta có kết quả hai điểm biên lân cận theo các hướng biên khác nhau như bảng dưới đây:

Giá trị θ	Phương hướng	Điểm ảnh
$\theta \leq 22,5^0$ hoặc $\theta > 157,5^0$	$\theta = 0^0$	(x − 1, y); (x + 1, y)
$22,50 < \theta \leq 67,50$	$\theta = 45^0$	(x − 1, y − 1); (x + 1, y + 1)
$67,5^0 < \theta \leq 112,5^0$	$\theta = 90^0$	(x − 1, y − 1); (x + 1, y − 1)
$112,5^0 < \theta \leq 157,5^0$	$\theta = 135^0$	(x, y + 1); (x, y − 1)

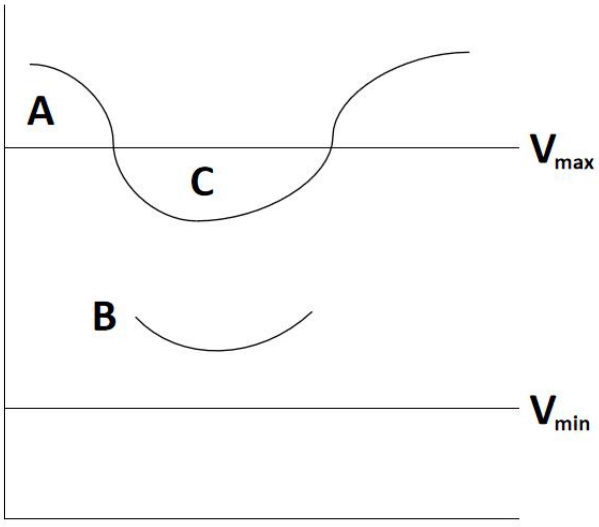
Nếu điểm ảnh P (x, y) có cường độ gradient lớn nhất trong ba điểm ảnh kiểm tra thì được giữ lại điểm biên này. Nếu một trong hai điểm ảnh khác có cường độ gradient cao hơn thì điểm ảnh P (x, y) này không có trong "trung tâm" của biên và không nên được phân loại như là một điểm biên (tức là loại đi – cho giá trị điểm này = 0).

Tóm lại, kết quả nhận được là một hình ảnh nhị phân với "các cạnh mỏng".

Image Threshold Setting and Edge Connection

Bước này sẽ quyết định một cạnh ta dự đoán ở các bước trên nó có phải là một cạnh thật sự hay không. Giá trị threshold ở đây có hai ngưỡng Vmax và Vmin.

Bất kỳ cạnh nào có gradient cường độ lớn hơn Vmax chắc chắn là cạnh và những cạnh dưới Vmin chắc chắn không phải là cạnh, do đó, bị loại bỏ. Những cạnh nằm giữa hai ngưỡng này được phân loại các cạnh hoặc không cạnh dựa trên khả năng kết nối của chúng. Nếu chúng được kết nối với các pixel một cách chắc chắn, chúng được coi là một phần của các cạnh. Nếu không, chúng cũng bị loại bỏ. Ta có ví dụ như hình bên dưới:



Hình 2.2: Ví dụ về strong edge (cạnh chắc chắn), weak edge (không là cạnh) và sự kết nối giữa chúng.

Ví dụ ở hình trên A nằm trên ngưỡng Vmax nên A chắc chắn là cạnh (strong edge). C nằm giữa ngưỡng Vmax và Vmin nên C là cạnh yếu (weak edge) nhưng C kết nối với strong edge là A nên C cũng là một cạnh. Tuy nhiên B cũng nằm trong ngưỡng giống C

nhưng không kết nối với cạnh nào chắc chắn là cạnh (strong edge) nên B được xem là nhiễu.

Vì vậy, những gì cuối cùng nhận được là các cạnh thực sự trong hình ảnh.

3. Hough Line Transform

3.1. Ý tưởng

● Dựa trên kết quả phát hiện cạnh để tiến hành phát hiện đường thẳng. Giải thuật phát hiện cạnh thường được sử dụng cùng với Hough Line Transform là Canny Edge Detection.

● Trên mỗi pixel thuộc cạnh được phát hiện trong ảnh, lần lượt thử các phương trình đường thẳng đi qua pixel đó. Số phương trình đường thẳng được thử càng nhiều, ít bỏ lỡ đường thẳng có trong ảnh sẽ cho ra kết quả phát hiện đường thẳng càng tốt. Pixel cạnh đó sẽ cho thêm 1 giá trị vào ma trận thống kê.

● Duyệt hết tất cả các pixel cạnh, sau đó lọc theo một giá trị ngưỡng đã được xác định trước trên ma trận thống kê, từ đó xác định được các phương trình đường thẳng có trong ảnh.

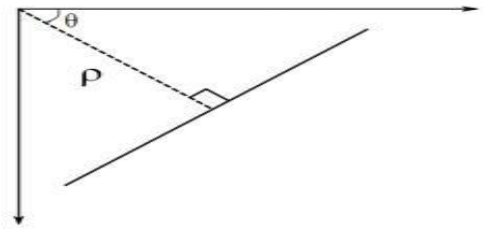
● Sau khi xác định được các đường thẳng, cuối cùng sẽ vẽ các đường thẳng đó lên ảnh.

3.2. Lý thuyết

Hough Transform là một kỹ thuật phổ biến để phát hiện bất kỳ hình dạng nào, có thể biểu diễn hình dạng đó dưới dạng toán học. Nó có thể phát hiện hình dạng ngay cả khi nó bị hỏng hoặc bị bóp méo. Chúng ta sẽ xem nó hoạt động như thế nào đối với một đường thẳng bất kỳ.

Phương trình đường thẳng trong không gian ảnh

Phương trình đường thẳng được biểu diễn với dạng 2 tham số m, c như sau: $y = mx + c$; đối với dạng được biểu diễn trong hệ tọa độ cực: $\rho = x\cos\theta + y\sin\theta$, với ρ là khoảng cách vuông góc từ điểm gốc đến đường thẳng, θ là góc tạo bởi đường vuông góc này và trục hoành (góc được đo ngược chiều kim đồng hồ). Hình ảnh bên dưới thể hiện các thông số của đường thẳng dạng tham số:



Hình 3.1: Đường thẳng dạng tham số

Cách biểu diễn đường thẳng ở dạng tham số gây khó khăn cho việc sử dụng giải thuật Hough Transform, vì yêu cầu đầu vào của giải thuật này đòi hỏi các giá trị m, c phải nằm trong một khoảng xác định (bị chặn trên dưới), trong khi giá trị góc nghiêng m trải dài từ -∞ đến +∞.

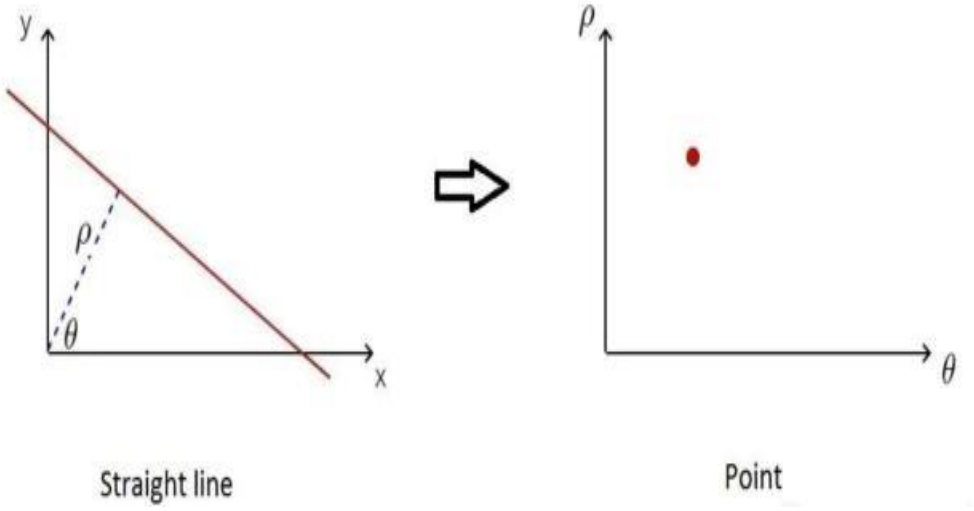
Trong phương trình hệ tọa độ cực, giá trị góc θ có thể được lấy chặn trong khoảng [0, π), dẫn tới giá trị ρ cũng bị chặn (do trên thực tế, không gian ảnh cũng là không gian hữu hạn).

Ngoài ra, phương trình trong hệ tọa độ cực có thể được viết lại như sau (tương đồng với dạng tham số):

$$y = -\frac{\cos\theta}{\sin\theta} \cdot x + \frac{r}{\sin\theta}$$

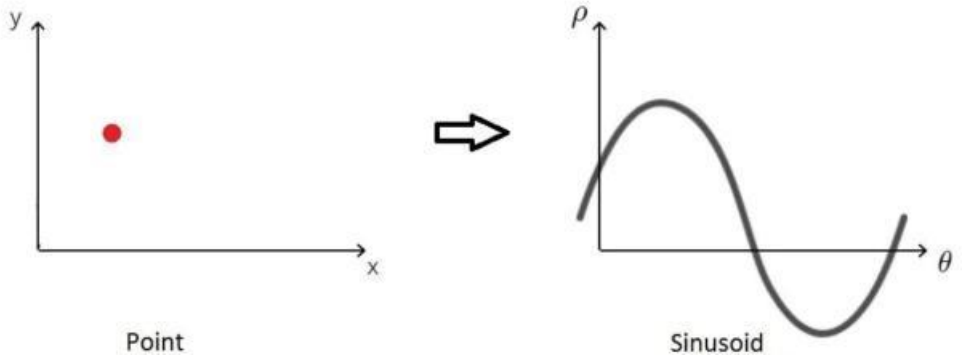
Mapping giữa không gian ảnh và không gian Hough

Với một đường thẳng trong không gian ảnh (kèm theo thông số ρ và θ), map sang không gian Hough sẽ chuyển thành một điểm:



Hình 3.2: Mapping đường sang điểm giữa không gian ảnh và không gian Hough

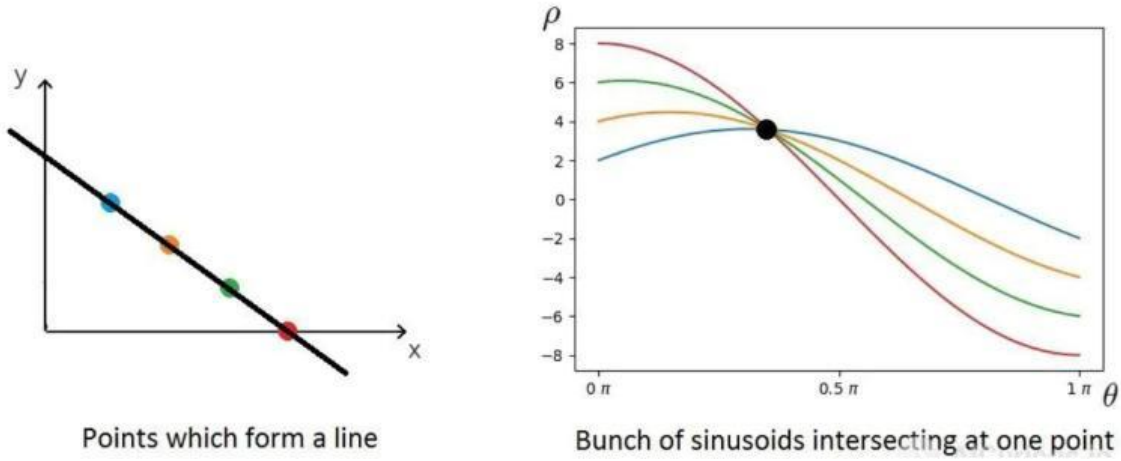
Với một điểm trong không gian ảnh, map sang không gian Hough sẽ có được một hình sin:



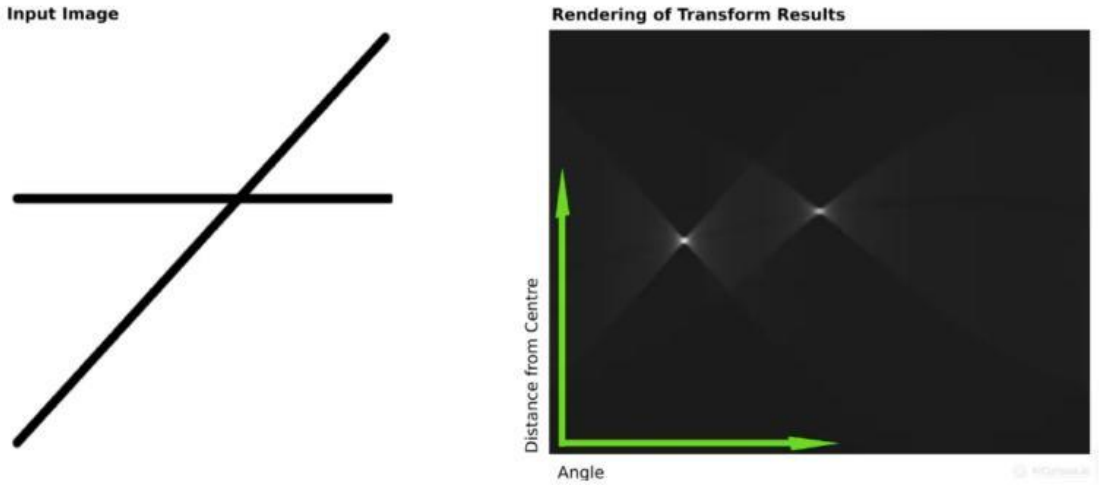
Hình 3.3: Mapping điểm sang đường sóng sin giữa không gian ảnh và không gian Hough

Thực chất, khi biểu diễn một đường thẳng, đường thẳng vốn là tập hợp của vô hạn các điểm nằm trên nó, mà một điểm trong không gian ảnh sẽ tương ứng là một hình sóng sin trong không gian Hough. Từ đó, tập hợp các điểm thuộc đường thẳng đang xét trong không gian ảnh sẽ cho ra tập hợp các đường sóng sin. Đặc biệt, các đường sóng sin đó sẽ có một điểm giao với nhau. Điểm giao này chính là dấu hiệu nhận biết tập hợp các điểm đang xét thuộc về một đường thẳng. Mỗi đường thẳng khác nhau sẽ tạo thành một

điểm giao các đường sóng sin (hay còn gọi là điểm sáng) trong không gian Hough. Từ đó, xác định được đường thẳng trong không gian ảnh.



Hình 3.4: Mapping nhiều điểm thẳng hàng sang nhiều đường sóng sin giữa không gian ảnh và không gian Hough



Hình 3.5: Hai đường thẳng được chuyển thành hai điểm sáng trong không gian Hough

4. Thực nghiệm giải thuật trên phần mềm

4.1. Phân ngưỡng ảnh (threshold)

4.1.1. Các điểm ảnh cô lập có thể đại diện cho các vạch kẻ đường

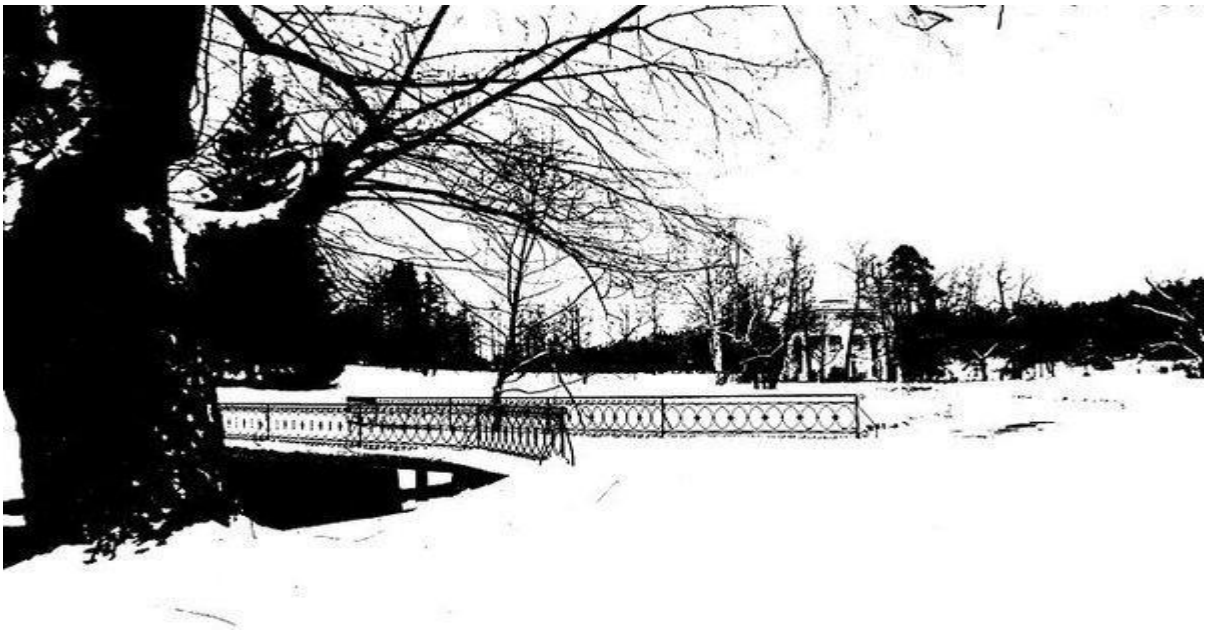
Trong xử lý hình ảnh kỹ thuật số, phân ngưỡng ảnh (thresholding) là phương pháp phân đoạn hình ảnh đơn giản nhất. Từ một hình ảnh thang độ xám, phân ngưỡng ảnh có thể được sử dụng để tạo hình ảnh nhị phân.

Ánh sáng chói từ mặt trời, bóng tối, đèn pha ô tô và những thay đổi của mặt đường đều có thể gây khó khăn cho việc tìm làn đường trong hình ảnh hoặc khung hình video.

Phần đầu tiên của quy trình phát hiện làn đường là áp dụng phân ngưỡng ảnh cho mỗi khung hình video để ta có thể loại bỏ những thứ gây khó khăn cho việc phát hiện làn đường. Bằng cách áp dụng phân ngưỡng ảnh, chúng ta có thể cô lập các điểm ảnh (pixels) đại diện cho các làn đường.

Trước khi phân ngưỡng ảnh:

Sau khi phân ngưỡng ảnh:



4.1.2. Các bước phân ngưỡng ảnh

4.1.2.1. Chuyển đổi khung hình video từ không gian màu BGR (xanh lam, xanh lục, đỏ) sang HLS (sắc độ, độ bão hòa, độ đậm nhạt)

Có rất nhiều cách để thể hiện màu sắc trong hình ảnh. Nếu ta sử dụng chương trình như Microsoft Paint hoặc Adobe Photoshop, ta thấy rằng một cách để biểu thị màu là sử dụng không gian màu RGB (trong thư viện OpenCV, nó là BGR thay vì RGB), trong đó mỗi màu là hỗn hợp của ba màu sắc, đỏ, xanh lá cây và xanh lam.

Không gian màu HLS tốt hơn không gian màu BGR để phát hiện các vấn đề hình ảnh do ánh sáng: chẳng hạn như bóng đổ, ánh sáng chói từ mặt trời, đèn pha, v.v… Nhóm muốn loại bỏ tất cả những thứ này để giúp phát hiện các vạch làn đường dễ dàng hơn. Vì lý do này, nhóm sử dụng không gian màu HLS, chia tất cả các màu thành các giá trị sắc độ, độ bão hòa và độ đậm nhạt.

4.1.2.2. Thực hiện giải thuật Sobel edge detection trên kênh L (độ đậm nhạt) của hình ảnh để phát hiện sự gián đoạn rõ nét trong cường độ pixel dọc theo trục x và y của khung hình video

Những thay đổi rõ nét về cường độ từ một điểm ảnh sang điểm ảnh lân cận có nghĩa là một cạnh có thể xuất hiện. Nhóm muốn phát hiện các đường cạnh có cường độ mạnh nhất trong hình ảnh để có thể cô lập các cạnh có thể là làn đường.

4.1.2.3. Thực hiện phân ngưỡng nhị phân trên kênh S (bão hòa) của khung hình video

Giá trị độ bão hòa cao có nghĩa là màu sắc rõ ràng. Các đường phân làn thường có màu sắc sáng, rõ ràng, chẳng hạn như màu trắng và màu vàng. Cả màu trắng và màu vàng đều có giá trị kênh bão hòa cao.

Ngưỡng nhị phân tạo ra một hình ảnh có đầy đủ các giá trị cường độ từ 0 (đen) đến 255 (trắng). Các điểm ảnh có giá trị bão hòa cao (ví dụ: > 80 trên thang giá trị từ 0 đến 255) sẽ được đặt thành màu trắng, trong khi mọi thứ khác sẽ được đặt thành màu đen.

4.1.2.4. Thực hiện phân ngưỡng nhị phân trên kênh R (sắc độ) của khung hình video BGR ban đầu

Bước này giúp trích xuất các giá trị màu vàng và trắng, là màu đặc trưng của đường phân làn. Màu trắng thuần là bgr (255, 255, 255). Màu vàng thuần là bgr (0, 255, 255). Cả hai đều có giá trị kênh sắc độ cao.

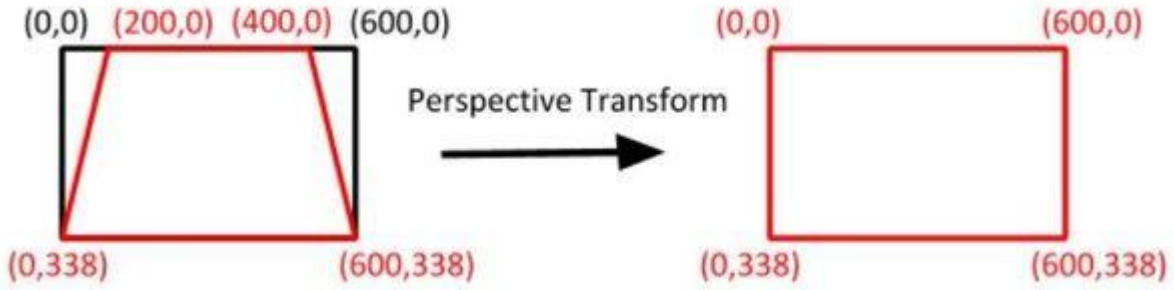
Để tạo hình ảnh nhị phân giai đoạn này, các điểm ảnh có giá trị kênh R cao (ví dụ: > 120 trên thang giá trị từ 0 đến 255) sẽ được đặt thành màu trắng. Tất cả các điểm ảnh khác sẽ được đặt thành màu đen.

4.1.2.5. Thực hiện thao tác bitwise AND để giảm nhiễu trong ảnh do bóng đổ và các biến thể của màu khi xe chạy trên đường

Đường phân làn phải có màu thuần và có giá trị kênh R (sắc độ) cao. Thao tác bitwise AND giúp giảm nhiễu và bôi đen bất kỳ điểm ảnh nào có vẻ không rõ ràng, thuần và đồng nhất.

Phương thức **get_line_markings (self, frame = None)** trong **lane.py** thực hiện tất cả các bước mà nhóm đã đề cập ở trên.

4.2. Cách thức hoạt động của quá trình chuyển đổi phối cảnh

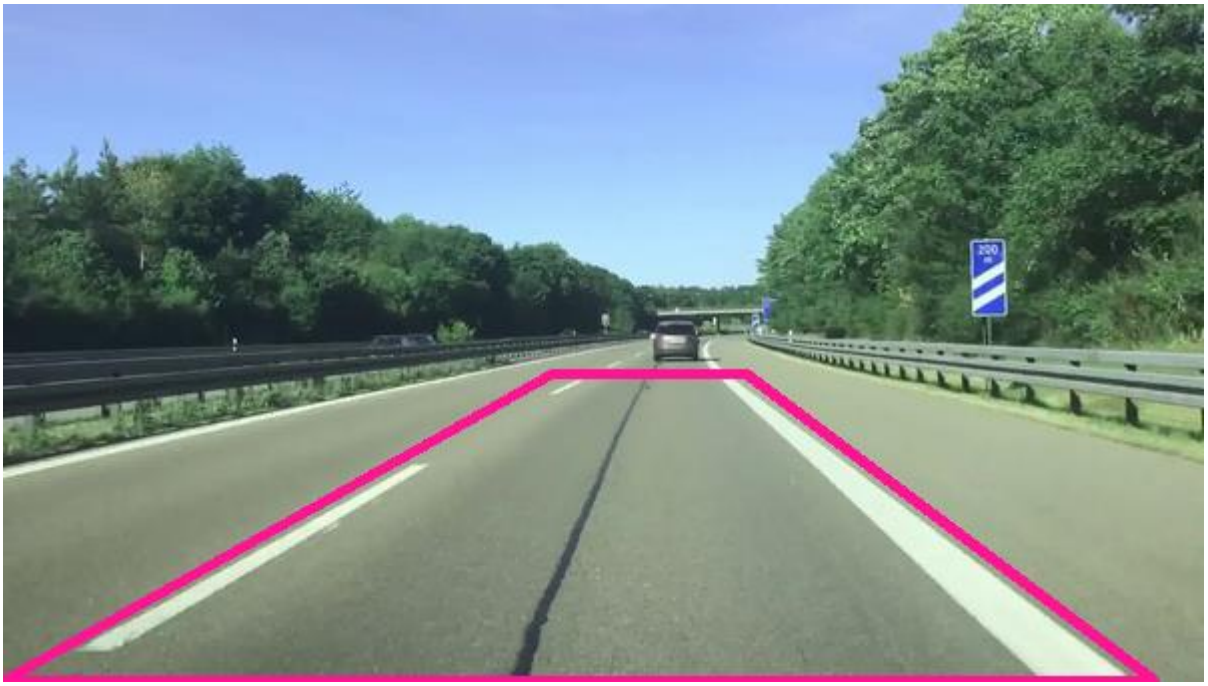


Thư viện OpenCV có các phương pháp giúp chúng ta thực hiện chuyển đổi phối cảnh (tức là biến đổi xạ ảnh hoặc hình học xạ ảnh). Các phương pháp này làm cong góc nhìn của máy ảnh thành chế độ xem góc nhìn cao (tức là chế độ xem từ trên cao nhìn xuống).

Đối với bước đầu tiên của chuyển đổi phối cảnh, chúng ta cần xác định khu vực quan tâm (ROI – Region of Interest). Bước này giúp xóa các phần của hình ảnh mà chúng ta không quan tâm. Chúng ta chỉ quan tâm đến đoạn làn đường ngay phía trước ô tô.

Đoạn code sau sẽ hiện thực nó:

```
lane_obj.plot_roi(plot=True)
```



Ta có thể thấy rằng khu vực ROI là hình thang với bốn góc riêng biệt.

Bây giờ chúng ta đã có khu vực quan tâm (ROI), ta sử dụng các phương pháp **getPerspectiveTransform** và **warpPerspective** của OpenCV để chuyển đổi phối cảnh giống hình thang thành một phối cảnh giống như hình chữ nhật.

```
warped_frame = lane_obj.perspective_transform(plot=True)
```

Đây là một ví dụ về hình ảnh sau quá trình này. Ta có thể thấy phối cảnh bây giờ như thế nào từ chế độ xem góc nhìn cao. Các đường trong khu vực quan tâm hiện song song với các cạnh của hình ảnh, giúp tính toán độ cong của đường và làn đường dễ dàng hơn.

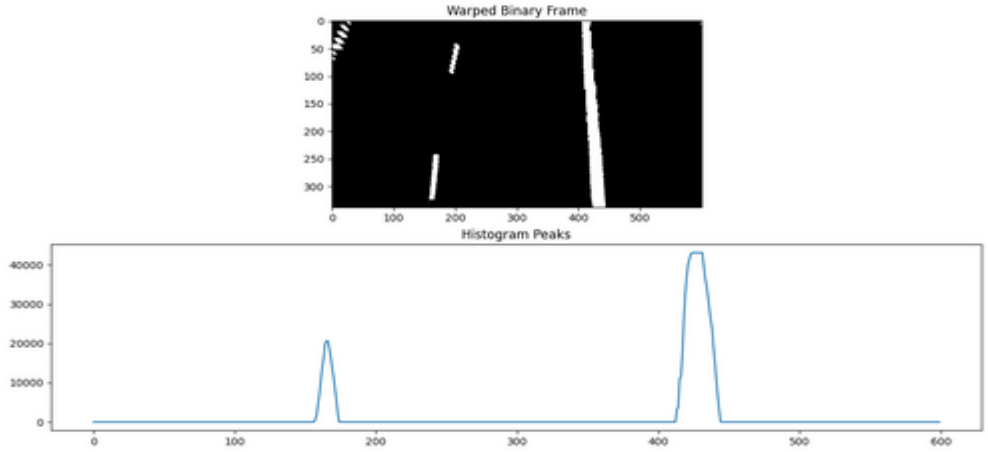
4.3. Xác định các điểm ảnh làn đường

Bây giờ chúng ta cần xác định các điểm ảnh trên hình ảnh bị cong vênh tạo nên các làn đường. Nhìn vào hình ảnh bị cong vênh, chúng ta có thể thấy rằng các điểm ảnh màu trắng đại diện cho các phần của các làn đường.

Chúng ta bắt đầu phát hiện các điểm ảnh làn đường bằng cách tạo biểu đồ để xác định vị trí các khu vực của hình ảnh có cường độ các điểm ảnh màu trắng cao.

Lý tưởng nhất là khi chúng ta vẽ biểu đồ, chúng ta sẽ có hai đỉnh. Sẽ có một đỉnh bên trái và một đỉnh bên phải, tương ứng với vạch của làn đường bên trái và vạch của làn đường bên phải.

```
histogram = lane_obj.calculate_histogram(plot=True)
```

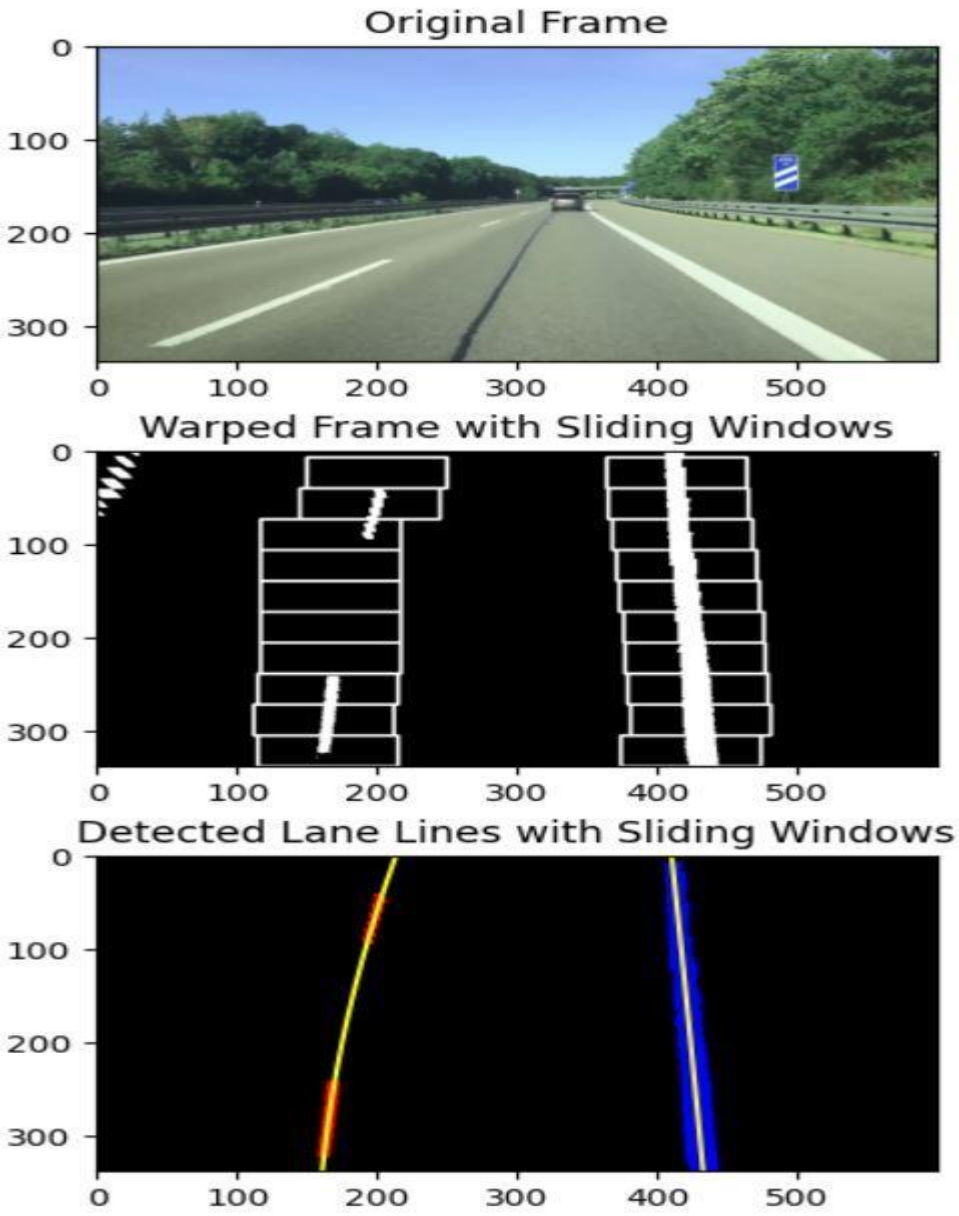


4.4. Kỹ thuật cửa sổ trượt (Sliding Windows) để phát hiện điểm ảnh trắng

Bước tiếp theo là sử dụng kỹ thuật cửa sổ trượt (Sliding Windows), nơi chúng ta bắt đầu ở dưới cùng của hình ảnh và quét tất cả lên trên cùng của hình ảnh. Mỗi lần chúng ta tìm kiếm trong cửa sổ trượt, chúng ta thêm các điểm ảnh có thể là làn đường vào danh sách. Nếu chúng ta có đủ các điểm ảnh làn đường trong một cửa sổ, thì vị trí trung bình của những điểm ảnh này sẽ trở thành tâm của cửa sổ trượt tiếp theo.

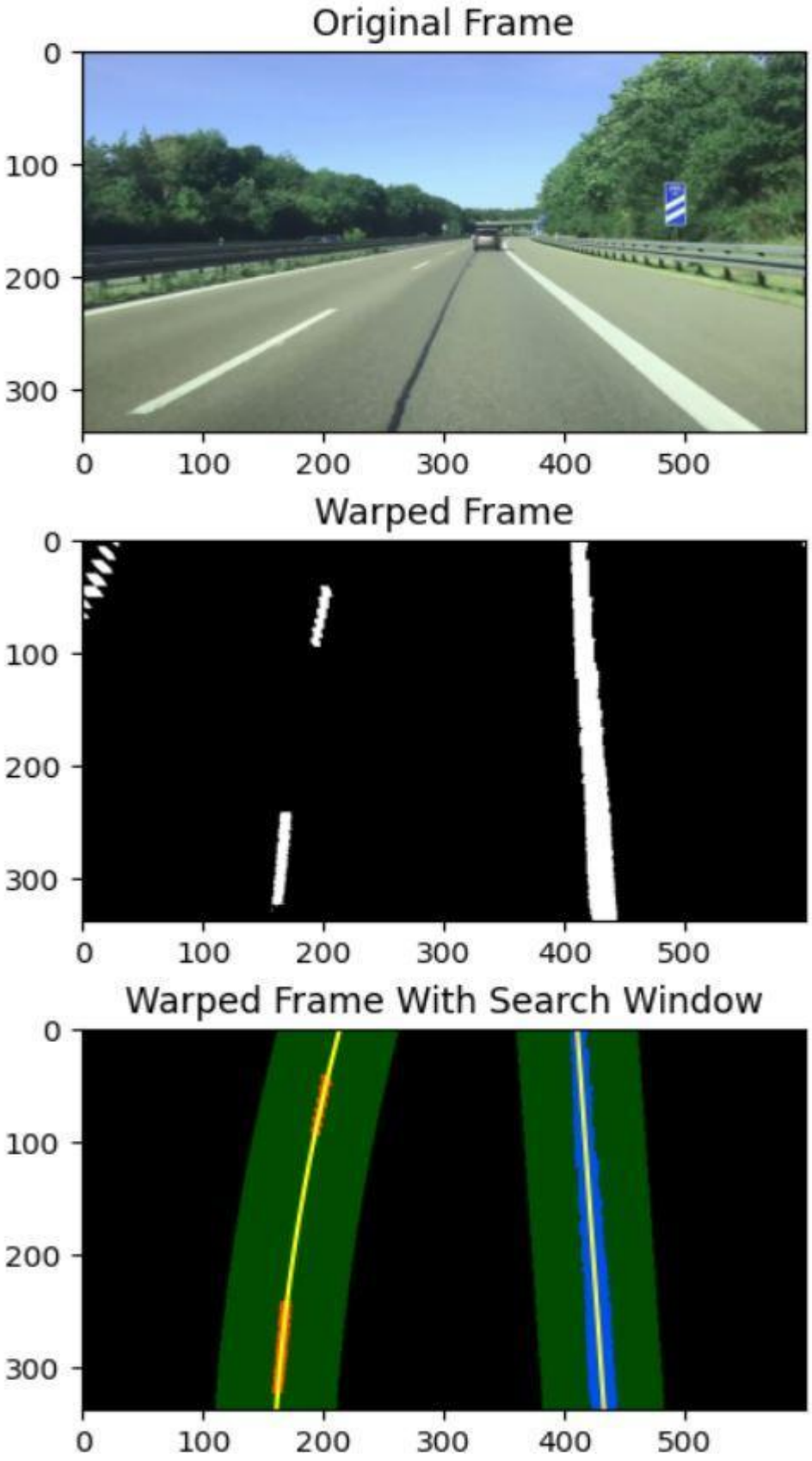
Khi chúng ta đã xác định được các điểm ảnh tương ứng với các làn đường bên trái và bên phải, chúng ta vẽ một đường đa thức phù hợp nhất qua các điểm ảnh. Đường này thể hiện ước tính tốt nhất của nhóm về các làn đường.

```
left_fit, right_fit = lane_obj.get_lane_line_indices_sliding_windows(plot=True)
```



4.5. Phát hiện vạch kẻ đường

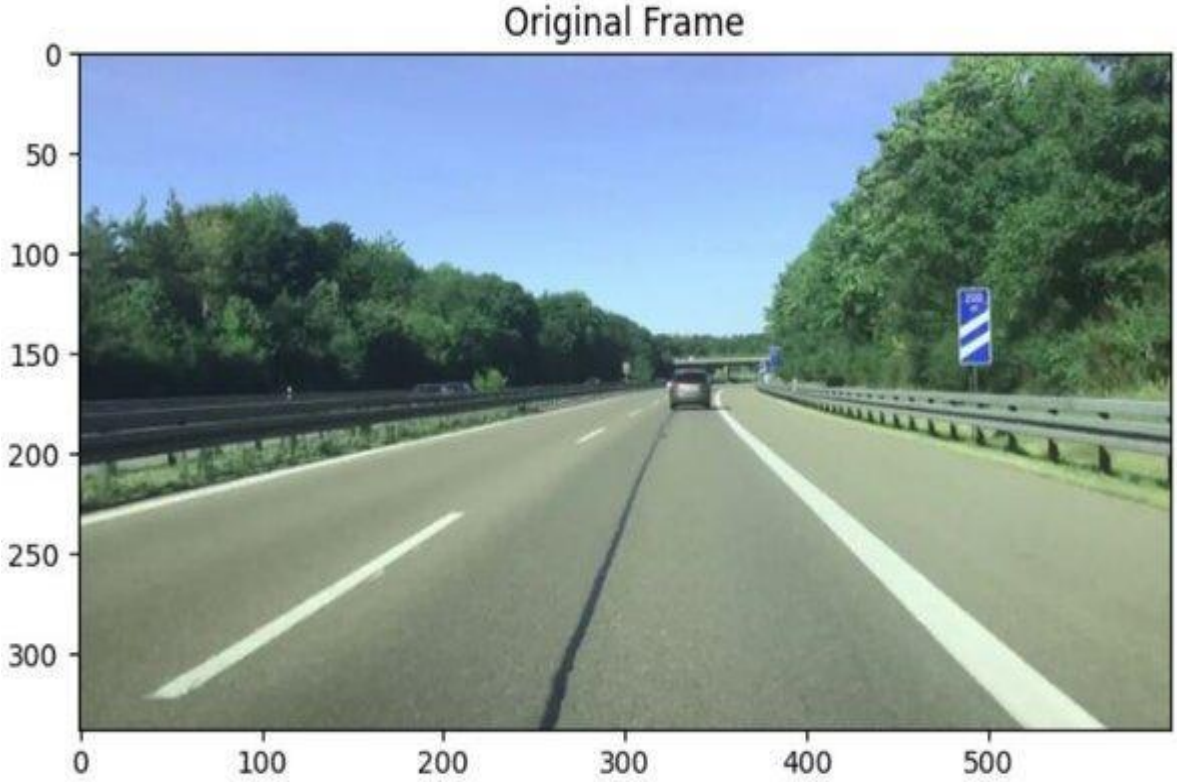
```
lane_obj.get_lane_line_previous_window(left_fit, right_fit, plot=True)
```

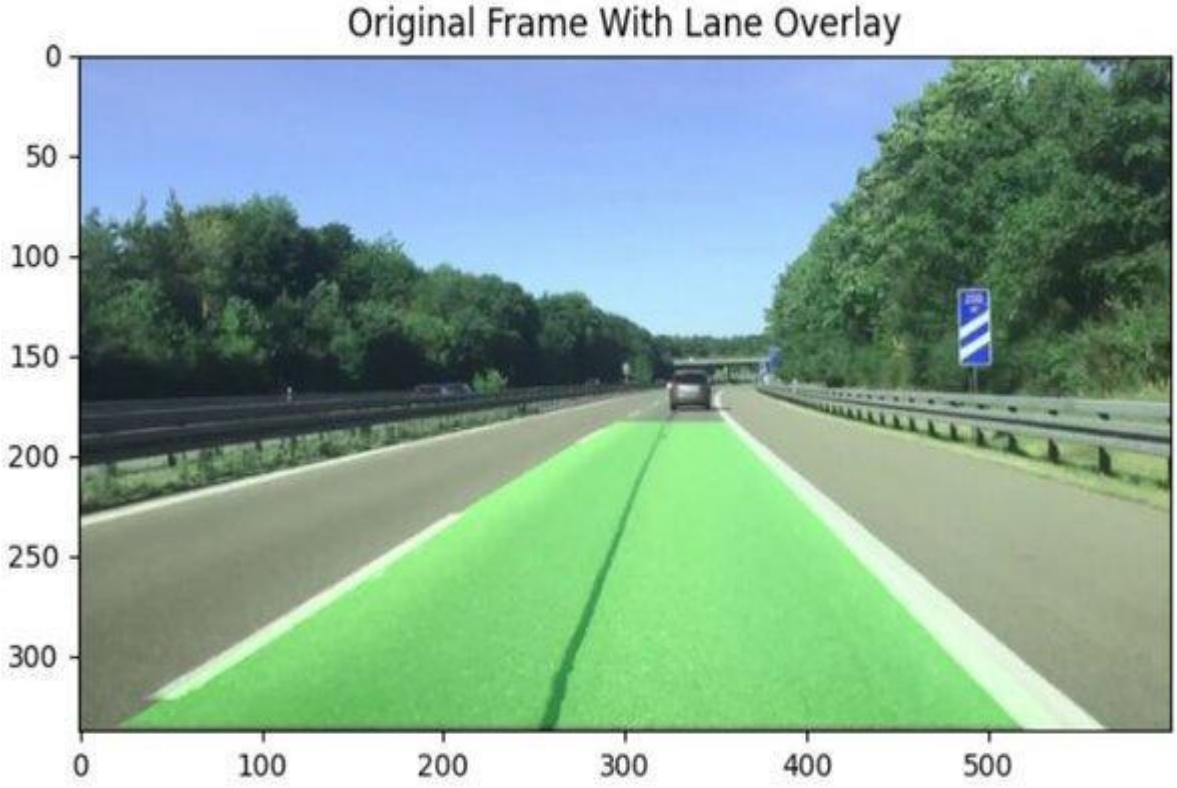

Original Frame
0
100
200
300
0
100
200
300
400
500
Warped Frame
0
100
200
300
0
100
200
300
400
500
Warped Frame With Search Window
0
100
200
300
0
100
200
300
400
500

4.6. Lớp phủ màu làn đường đang đi trên hình ảnh gốc

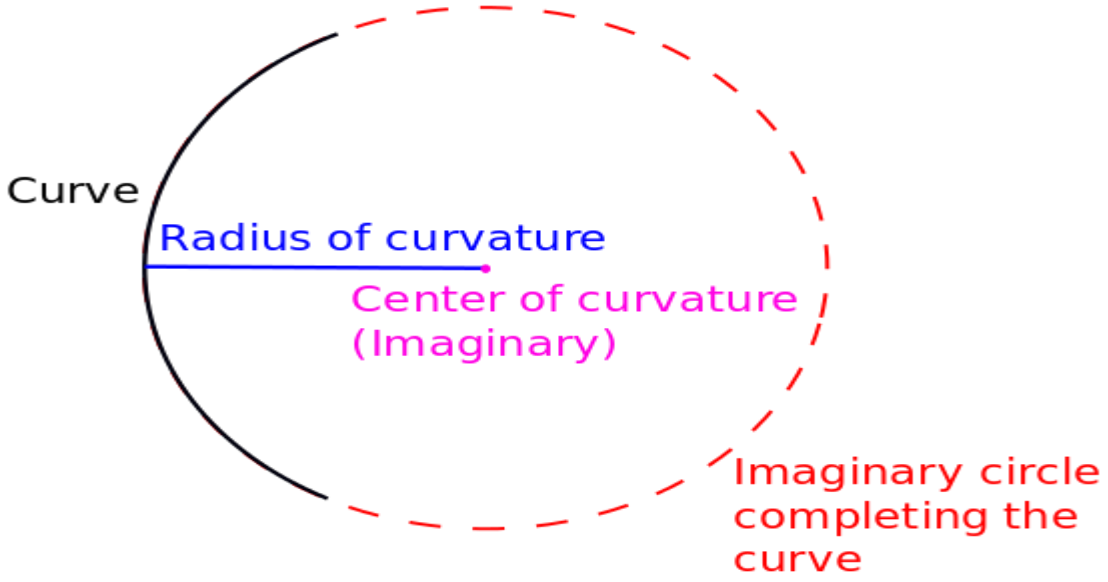
Chúng ta đã xác định được các vạch làn đường, giờ chúng ta cần phủ lớp màu lên làn đường đang đi từ hình ảnh gốc ban đầu.

```
frame_with_lane_lines = lane_obj.overlay_lane_lines(plot=True)
```


Original Frame
0
50
100
150
200
250
300
0
100
200
300
400
500


Original Frame With Lane Overlay
0
50
100
150
200
250
300
0
100
200
300
400
500

4.7. Tính toán độ cong làn đường




```
lane_obj.calculate_curvature(print_to_terminal=True)
```

Đây là kết quả tính toán độ cong làn đường. Ta có thể thấy bán kính cong từ các làn đường bên trái và bên phải:

```
40.31949794887235 m 3772.427228241085 m
```

4.8. Tính độ lệch tâm

Chúng ta cần tính toán xem trọng tâm của chiếc xe cách chính giữa làn đường bao xa (tức là tính toán "Độ lệch của đường tâm").

```
lane_obj.calculate_car_position(print_to_terminal=True)
```

Đây là kết quả đầu ra. Ta có thể thấy độ lệch tâm tính bằng cen-ti-mét:

```
1.435914726167449cm
```

4.9. Hiển thị kết quả cuối cùng

Chúng ta sẽ hiển thị kết quả cuối cùng với các chú thích về độ cong và độ lệch cũng như làn đường đã được đánh dấu khi xe đang chạy trên làn đường đó.

```
frame_with_lane_lines2 =
lane_obj.display_curvature_offset(frame=frame_with_lane_lines, plot=True)
```

Chạy chương trình **lane.py**:

```
python lane.py
```

Kết quả cuối cùng:

Ta nhận thấy rằng bán kính đường cong là giá trị trung bình của bán kính cong cho các làn đường bên trái và làn đường bên phải.

PHẦN 2: BÁO CÁO GIAI ĐOẠN 2 – HIỆN THỰC PHẦN CỨNG XỬ LÝ

I. Báo cáo giai đoạn 2

Link Github về Project của nhóm 3 trong giai đoạn 2: https://github.com/ULTIMATE-Mystery/Logic-Design-Project-Group-3-Semester-221-HCMUT/tree/Phase-2.

1. Nội dung đã làm được trong giai đoạn vừa rồi là gì?

● Chọn ra phần cứng phù hợp với kinh tế và nguồn lực của nhóm để thực hiện đồ án giai đoạn 2, đó là ESP32-CAM:

- ESP32-CAM được đánh giá là module Wifi giá rẻ hoàn toàn phù hợp cho các dự án tự làm trong lĩnh vực Internet of Things (IoT). Các tính năng đi kèm với GPIOs, hỗ trợ cho một loạt các giao thức như SPI, I2C, UART,…

- Giá thành tương đối rẻ, kích thước nhỏ gọn, có camera tích hợp trong module.

- Được sử dụng phổ biến trong nhiều dự án, tạo nên cộng đồng hỗ trợ mạnh và rộng lớn.

- Dòng tiêu thụ ở chế độ deep sleep giúp tiết kiệm điện năng.

- Có thể dễ dàng lắp đặt trên breadboard hoặc tích hợp lên bo mạch sản phẩm một cách chắc chắn.

- Chi tiết đặc tính và thông số kỹ thuật của module cực kỳ tiện dụng này được liệt kê dưới đây:

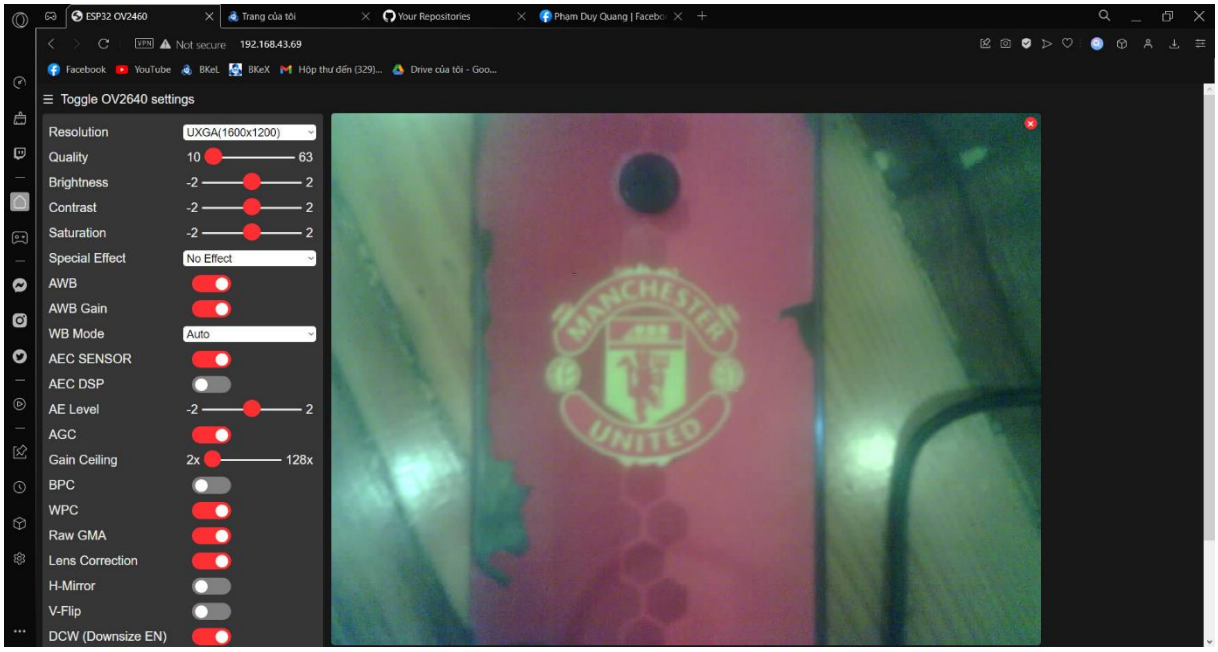
Module	ESP32-CAM AI-Thinker
Bộ xử lý	LX6 32-bit
Kiến trúc vi mạch	Đường dữ liệu 7 lớp
SPI Flash	32-bit
SRAM	520 KB
PSRAM	4 MB

Dãy nhiệt độ hoạt động	$-20℃ - 85℃$
Điện áp hoạt động	5V
UART	1
Tốc độ truyền UART	115200 bps
SPI	Có
I2C	Có
PWM	Có
Wifi	802.11 b / g / n
Bluetooth	Bluetooth 4.2 BR / EDR theo tiêu chuẩn BLE
Hỗ trợ thẻ TF	Hỗ trợ tối đa 4G
Bảo mật dữ liệu	WPA / WPA2 / WPA2-Enterprise / WPS
Định dạng hình ảnh đầu ra	BMP, GRAYSCALE, JPEG (chỉ hỗ trợ OV2640)
Số cổng đầu vào / đầu ra	9
Công suất tiêu thụ	Khi tắt đèn flash tiêu tốn: 180mA ở 5V. Bật đèn flash với độ sáng tối đa: 310mA ở 5V. Chế độ ngủ sâu (sleepmode): có mức tiêu thụ điện năng thấp nhất có thể đạt 6mA ở 5V. Modem-sleep mode: <20mA ở 5V, slight-sleep mode: <6.7mA ở 5V

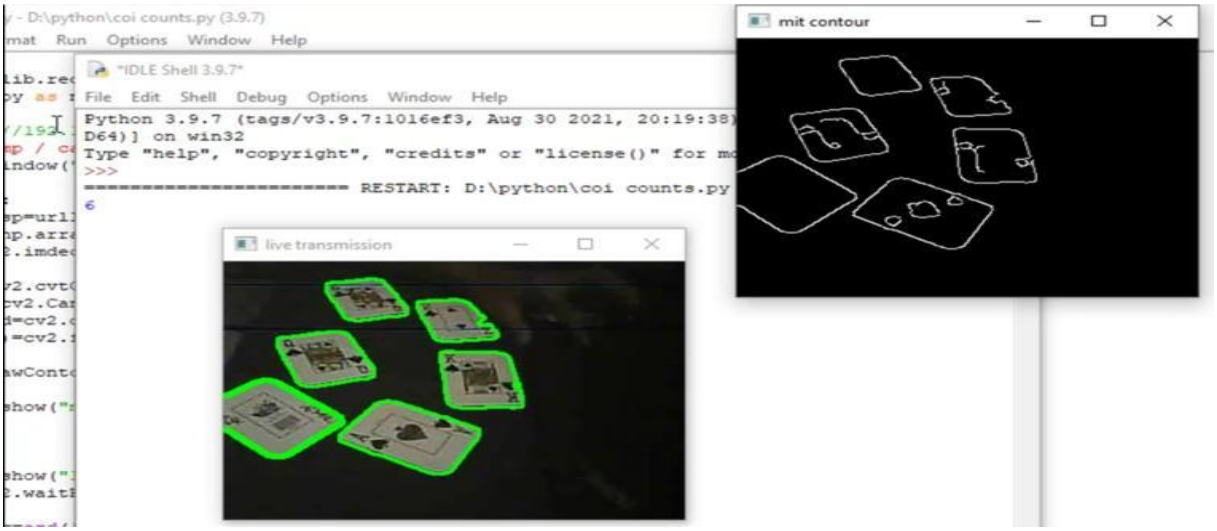
Tần số xung nhịp	240 MHz (tối đa)
Loại package	DIP-16
Kích cỡ package	27 × 40,5 × 4,5 (±0,2) mm

- Truyền được hình ảnh trực tiếp từ camera của ESP32-CAM lên local server, sau đó lấy hình ảnh từ local server về máy để xử lý ảnh thời gian thực bằng Python (tuy nhiên điều này chưa đúng với yêu cầu của đồ án là phải xử lý ảnh trực tiếp trên vi xử lý).

Đưa hình ảnh trực tiếp từ camera ESP32-CAM lên local server:



Dùng Python lấy những hình ảnh trực tiếp từ local server về xử lý ảnh trên thời gian thực:



- Nhận biết rằng việc xử lý hình ảnh trực tiếp trên những vi xử lý là rất khó khăn vì sự giới hạn về thư viện (không có thư viện OpenCV) cùng với tốc độ xử lý và bộ nhớ giới hạn.

- Nhóm chuyển sang thực hiện các thao tác xử lý ảnh trên nền tảng FPGA, mặc dù đã gặp không ít khó khăn nhưng nhóm đã đạt được một số kết quả nhất định.

- Nhận biết rằng với xử lý hình ảnh thì thư viện OpenCV là khó có thể thiếu cũng như những thiết bị phần cứng phù hợp cho các tác vụ xử lý ảnh (Raspberry Pi – một máy vi tính rất nhỏ gọn, FPGA, các vi xử lý hỗ trợ thư viện OpenMV,…).

2. Nội dung dự định làm trong giai đoạn tiếp theo là gì?

- Hoàn thiện hơn nữa các thao tác xử lý ảnh trên nền tảng FPGA để thực nghiệm trên thực tế.

- Đánh giá thực nghiệm và so sánh hiệu năng trên các nền tảng phần cứng khác nhau.

- Hoàn thiện báo cáo tổng kết về đồ án.

3. Khó khăn nào gặp phải trong giai đoạn vừa rồi?

- Với nguồn lực, kinh tế và thời gian giới hạn, nhóm khó tiếp cận được các thiết bị phần cứng với hiệu năng xử lý cao như FPGA mà chỉ có thể chọn những thiết bị vi xử lý với hiệu năng, tốc độ xử lý và bộ nhớ lưu trữ thấp.

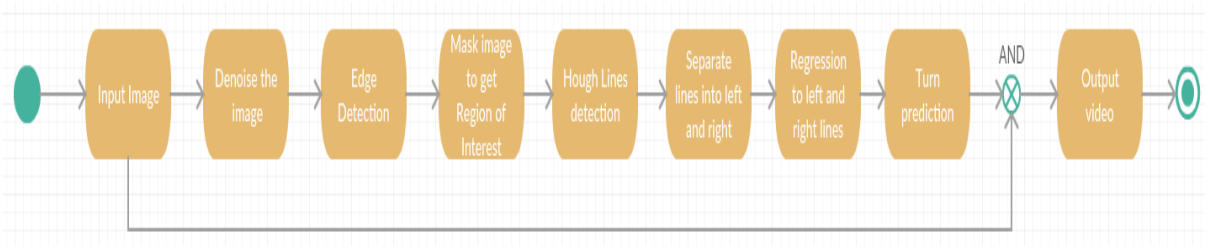
- Việc hiện thực các tác vụ xử lý hình ảnh mà không có thư viện OpenCV (ESP32-CAM không hỗ trợ) khiến cho nhóm gặp rất nhiều khó khăn. Mặc dù đã rất cố gắng,

nhóm chỉ có thể thực hiện một vài tác vụ xử lý hình ảnh mà không thể xử lý toàn bộ trực tiếp trên vi xử lý ESP32-CAM mà nhóm đã chọn.

II. Nội dung báo cáo

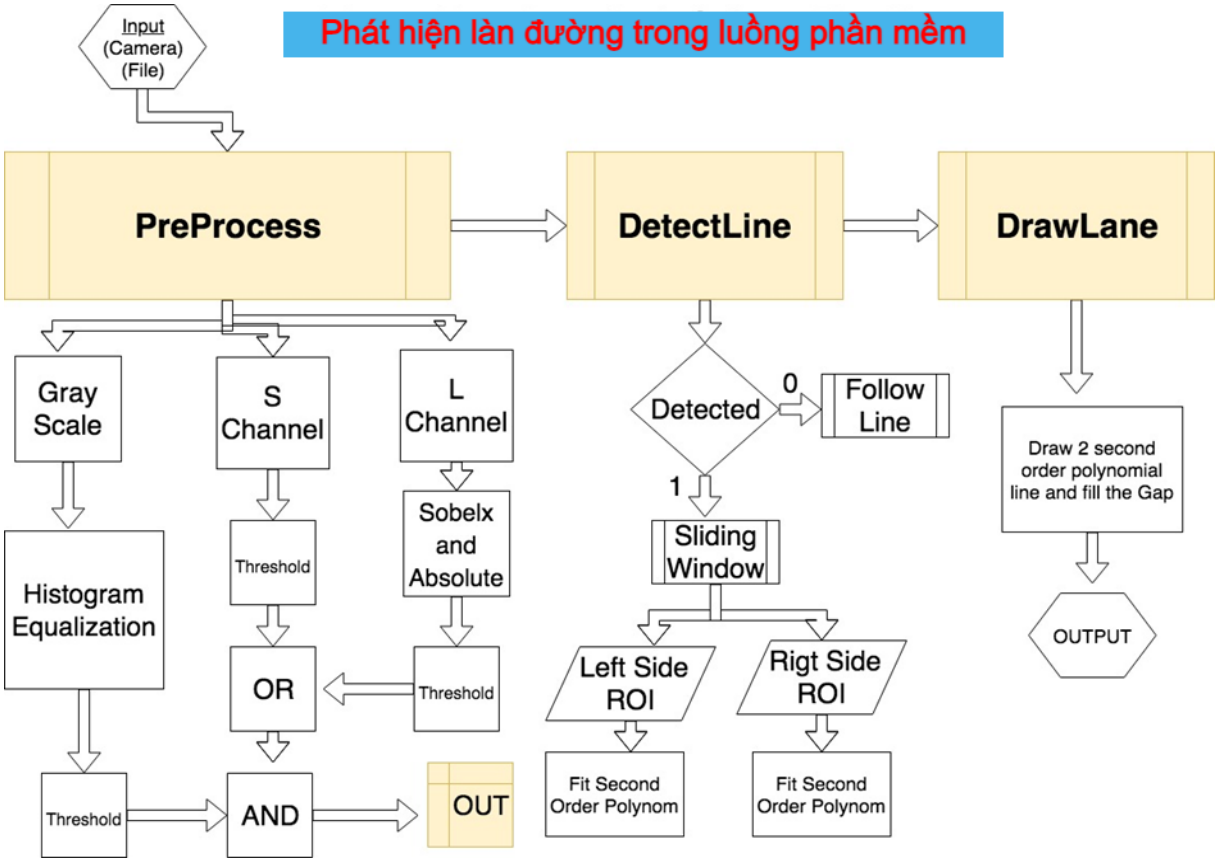
1. Thiết kế sơ đồ khối

1.1. Xử lý trực tiếp trên vi xử lý ESP32-CAM

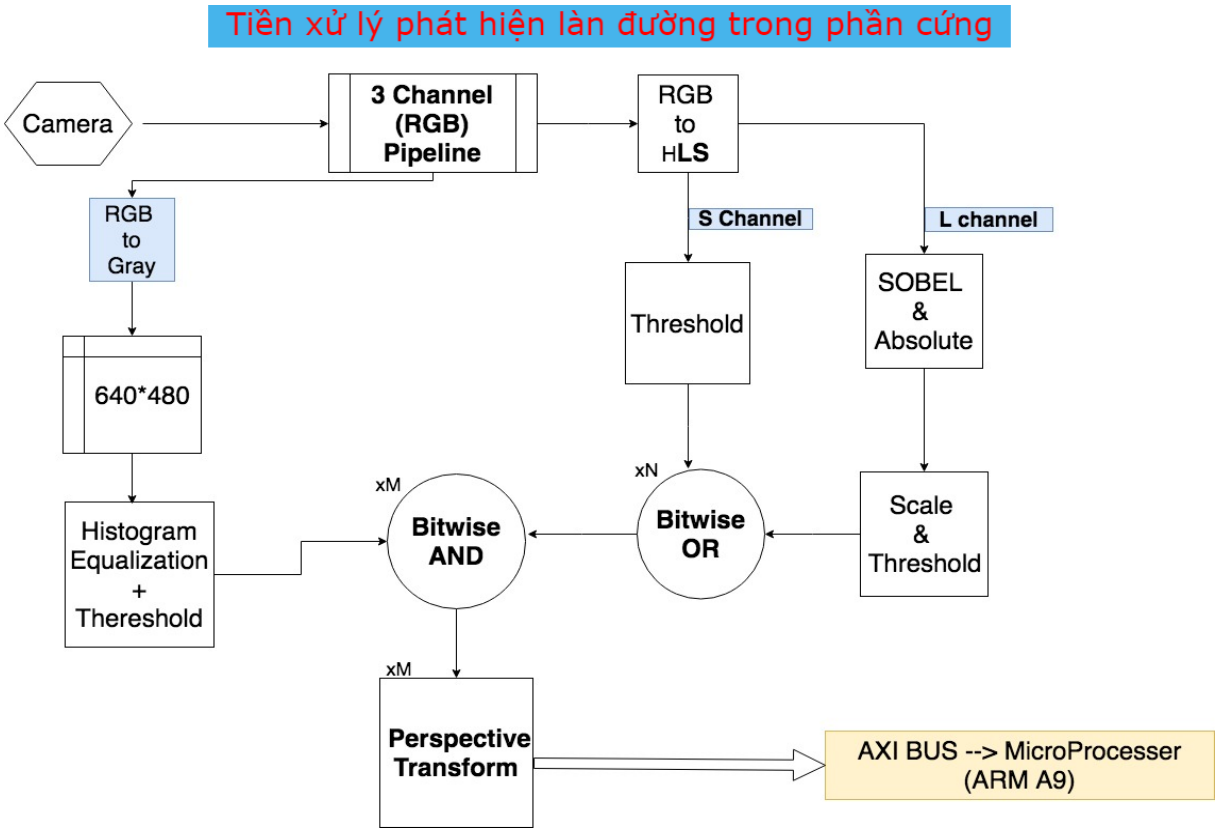


1.2. Xử lý trên nền tảng FPGA

1.2.1. Xử lý trong luồng phần mềm



1.2.2. Tiền xử lý trên phần cứng



2. Hiện thực từng khối xử lý

2.1. Hiện thực trên ESP32-CAM

Dù rất cố gắng nhưng nhóm đã không thể xử lý ảnh toàn bộ trực tiếp trên ESP32-CAM. Những gì nhóm đã thực hiện đó là:

- Tìm cách thêm thư viện OpenCV vào lập trình trên ESP32-CAM nhưng thất bại.

- Tìm cách thực hiện từng bước xử lý ảnh mà không dùng đến thư viện OpenCV nhưng vẫn thất bại, nhóm chỉ có thể thực hiện thao tác xử lý ảnh đơn giản đó là làm xám ảnh (grayscale) với đoạn code sau trên Arduino IDE:

```
#include "esp_camera.h"
#include "soc/soc.h"           // Disable brownour problems
#include "soc/rtc_cntl_reg.h"  // Disable brownour problems
#include "driver/rtc_io.h"
#include "img_converters.h"
```

```
// Pin definition for CAMERA_MODEL_AI_THINKER
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22



// our call back to dump whatever we got in binary format
size_t jpgCallBack(void * arg, size_t index, const void* data, size_t len)
{
  uint8_t* basePtr = (uint8_t*) data;
```

```
  for (size_t i = 0; i < len; i++) {
    Serial.write(basePtr[i]);
  }
  return 0;
}

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

  Serial.begin(115200);

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
```

```
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_GRAYSCALE;

  if (psramFound()) {
    config.frame_size = FRAMESIZE_QQVGA; // FRAMESIZE_ +
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.fb_count = 2;
  } else {
    Serial.println(F("ps RAM not found"));
    return;
  }

  // Init Camera
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
```

```
// disable white balance and white balance gain
  sensor_t * sensor = esp_camera_sensor_get();
  sensor->set_whitebal(sensor, 0);       // 0 = disable , 1 = enable
  sensor->set_awb_gain(sensor, 0);       // 0 = disable , 1 = enable

  camera_fb_t * fb = NULL;

  // Take Picture with Camera
  fb = esp_camera_fb_get();

  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }

  // DUMP THE PIXELS AS ASCII TO SERIAL
  Serial.printf("\n\nWidth = %u, Height=%u\n", fb->width, fb->height);
  for (size_t i = 0; i < fb->len; i++) {
    if (i % 16 == 0) Serial.printf("\n%06u\t", i);
    if (fb->buf[i] < 0x10) Serial.write('0');
    Serial.print(fb->buf[i], HEX);
  }

  Serial.println(F("\n\n---------------------\nPREPARE TO CAPTURE TO FILE\n"));
  delay(6000);
```

```
  frame2jpg_cb(fb, 10, jpgCallBack, NULL);


  esp_camera_fb_return(fb);
}


void loop() {}
```

Khi chạy đoạn code trên, nó sẽ thiết lập ESP32-CAM và chụp ảnh (thang độ xám 160x120) và in ra Serial output với tốc độ 115200 baud tất cả các pixel (ta nhận được 19200 byte với giá trị chính xác cho 160x120 pixel thang độ xám trên 8 bit). Sau đó nó sẽ in ra Serial Monitor một dòng là "PREPARE TO CAPTURE TO FILE" và đợi khoảng 6 giây.

Và sau đó ta gọi một hàm để tạo tệp jpg từ bộ đệm khung (**frame2jpg_cb**). Bằng cách này, ta có thể xem dữ liệu pixel và cũng có thể tạo một tệp mà ta có thể xem trên máy tính của mình.

Mở một Serial software có thể xử lý dữ liệu nhị phân và khi thấy dòng "PREPARE TO CAPTURE TO FILE", kích hoạt tính năng chụp tệp của software đó để lấy luồng byte vào một tập tin. Sau khi hoàn thành, dừng chụp và đổi tên tệp thành định dạng **.jpg**.

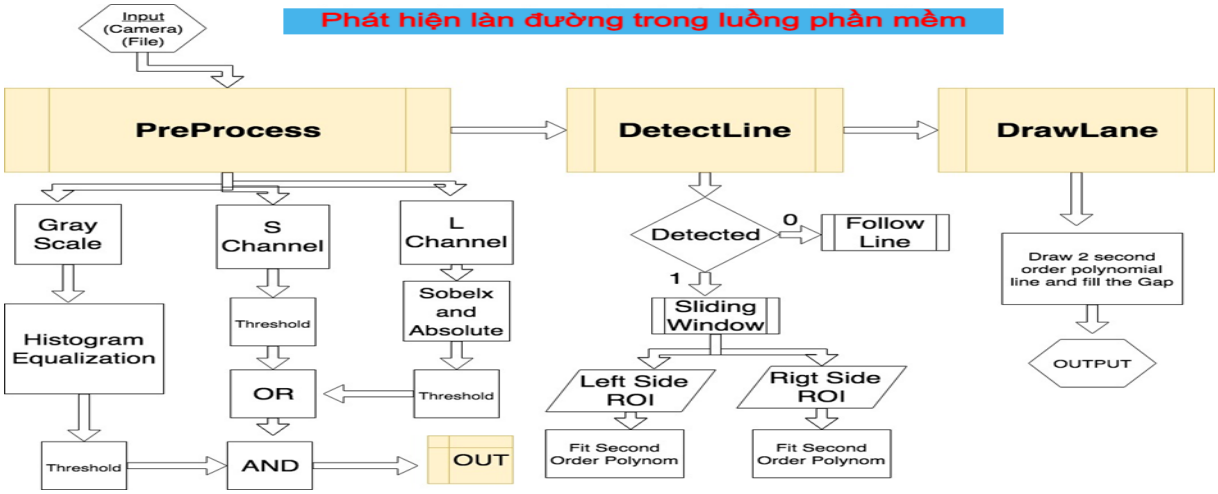
Đây là một ví dụ về hình ảnh nhận được:

2.2. Hiện thực trên nền tảng FPGA (phần cứng kết hợp với phần mềm)

Thuật toán phát hiện làn đường bao gồm ba phần chính: Tiền xử lý, phát hiện làn đường, vẽ làn đường. FPGA Zedboard dòng ZYNQ-7000 của Công ty Xilinx được sử dụng làm "Hệ thống trên Chip" (SoC) để giải quyết vấn đề này. Thuật toán phát hiện làn đường được triển khai trong FPGA sau khi thuật toán được chia ra thực hiện trên phần mềm và phần cứng.

Theo phân tích thời gian của thuật toán phát hiện làn đường, phần tiền xử lý đã được thực hiện trong phần cứng. Các nền tảng SDSoC và Vivado đã được sử dụng để thực hiện thuật toán phát hiện làn đường lập trình trên FPGA. Ngôn ngữ mô tả phần cứng (Verilog) được sử dụng trên nền tảng Vivado trong khi ngôn ngữ lập trình bậc cao (C++) được sử dụng trên nền tảng SDSoC. Trên nền tảng SDSoC, thư viện "xfOpenCV" cũng được sử dụng. Thư viện "xfOpenCV" là một bộ gồm hơn 50 nhân (kernels), được tối ưu hóa cho FPGA và SoC của Xilinx, dựa trên thư viện thị giác máy tính OpenCV. Các nhân trong thư viện xfOpenCV được tối ưu hóa và hỗ trợ trong Bộ công cụ Xilinx SDSoC.

2.2.1. Các bước xử lý giải thuật trên phần mềm



Đây là một số đầu ra trên các bước khác nhau của thuật toán:

a-) Original Frame (RGB)

b-) "L" Channel (HLS Color Space)

e-) "S" Channel (HLS Color Space)

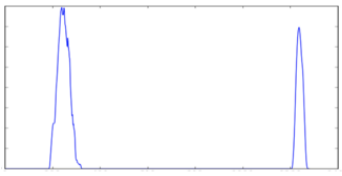
c-) Horizontal derivation

f-) Threshold

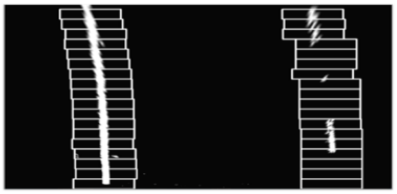
d-) Threshold

g-) Bitwise OR

h-) After Perspective Transform



ı-) Histogram

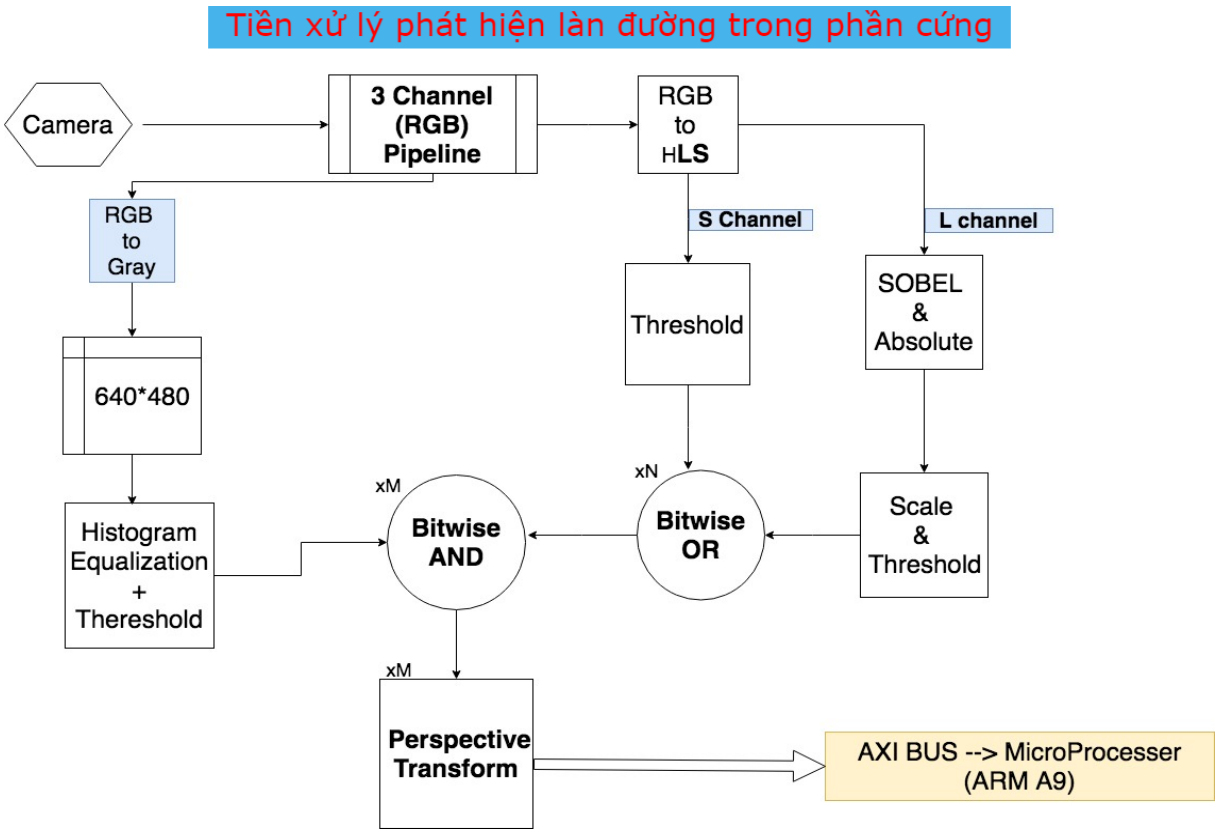


j-) Sliding Windows

2.2.2. Khối phần cứng và phần mềm trên FPGA

Một số phần của thuật toán thuộc hệ thống được triển khai trong phần cứng để hiện thực "Hệ thống phát hiện làn đường" nhanh hơn. Zedboard FPGA dòng ZYNQ-7000 của công ty Xilinx phù hợp với hệ thống vì thuật toán phát hiện làn đường có thể được tách biệt trên đó dưới dạng khối phần cứng và phần mềm. Khối phần mềm có sẵn trên bộ xử lý ARM Cortex-A9 trong Zedboard; khối phần cứng sẽ được triển khai trong các ô logic của Zedboard.

Thuật toán được chia thành 2 phần: Chức năng tiền xử lý được thực hiện trong phần cứng (logic lập trình) và các chức năng khác được thực hiện trong phần mềm (phần mềm lập trình – Arm-A9). Chức năng tiền xử lý nên được triển khai trong phần cứng do khả năng xử lý mạnh mẽ của nó. Sơ đồ khối hiện thực phần cứng của chức năng tiền xử lý trông như sau:



3. Kết nối các khối và kiểm tra tính chính xác

3.1. Trên vi xử lý ESP32-CAM

Nhóm đã thất bại trong việc hiện thực các thuật toán phát hiện làn đường trực tiếp trên ESP32-CAM mà chỉ có thể thực hiện các thao tác xử lý ảnh đơn giản như làm xám ảnh (grayscale)…

3.2. Trên nền tảng FPGA

Sử dụng thư viện "XfOpencv" với ngôn ngữ lập trình C++ thông qua nền tảng SDSoC (tổng hợp phần cứng cấp cao) trên Zedboard FPGA đã đạt được tốc độ xử lý khung hình/giây cao hơn cho "chức năng tiền xử lý" (không chuyển đổi phối cảnh) so với khi khối phần cứng không được sử dụng và triển khai trong bộ xử lý. Ta thấy rằng cấu trúc SoC được triển khai trên Zedboard với nền tảng SDSoC và với thư viện "XfOpencv" sử dụng ngôn ngữ lập trình C++ hiệu suất tốt hơn so với giải pháp phần mềm thuần túy trên bộ xử lý.

Trên nền tảng Vivado với ngôn ngữ lập trình mô tả phần cứng Verilog, chức năng "tiền xử lý" của hệ thống phát hiện làn đường đã được triển khai trên card phát triển Zedboard FPGA.

Kết luận: Khi thiết kế phần cứng được thực hiện, ngôn ngữ lập trình bậc cao có thể được sử dụng cho các giải pháp tham chiếu hoặc cho các giải pháp ngắn hạn. Mặt khác, ngôn ngữ lập trình bậc thấp phù hợp với các giải pháp dự kiến sẽ nâng cao hiệu suất sau này.

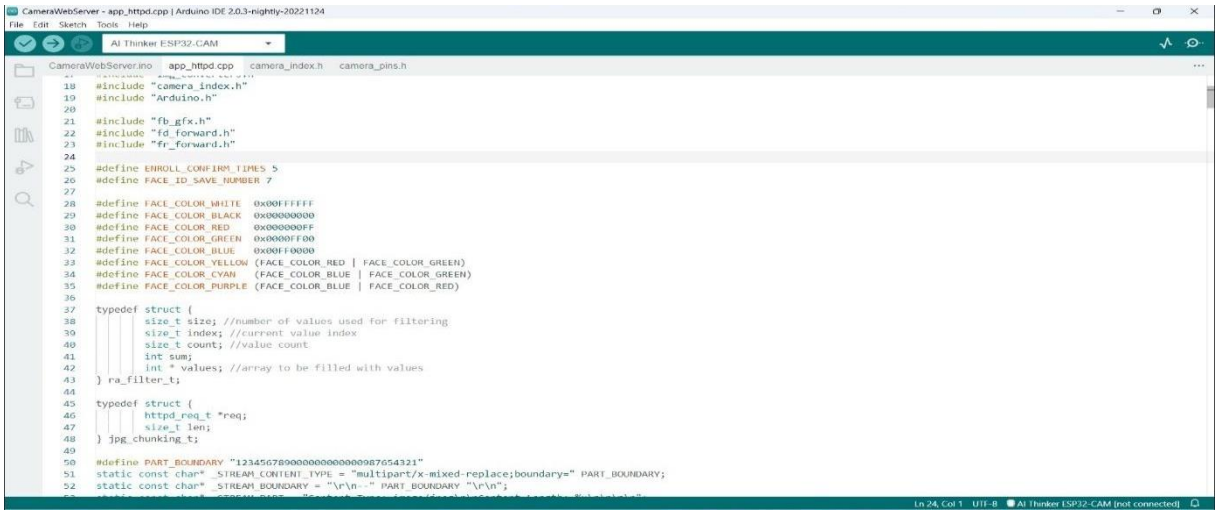
PHẦN 3: BÁO CÁO GIAI ĐOẠN 3 (TỔNG KẾT) – HOÀN THIỆN VÀ KIỂM THỬ

I. Báo cáo giai đoạn 3 (tổng kết)

Link Github về Project của nhóm 3 trong giai đoạn 3 (tổng kết): https://github.com/ULTIMATE-Mystery/Logic-Design-Project-Group-3-Semester-221-HCMUT.

1. Nội dung nhóm đã làm được trong các giai đoạn vừa rồi

- Sau rất nhiều cố gắng, cuối cùng nhóm cũng đã có thể hiện thực các giải thuật phát hiện cạnh/góc trực tiếp trên ESP32-CAM với Arduino IDE và hiển thị kết quả lên máy với sự hỗ trợ từ Python:



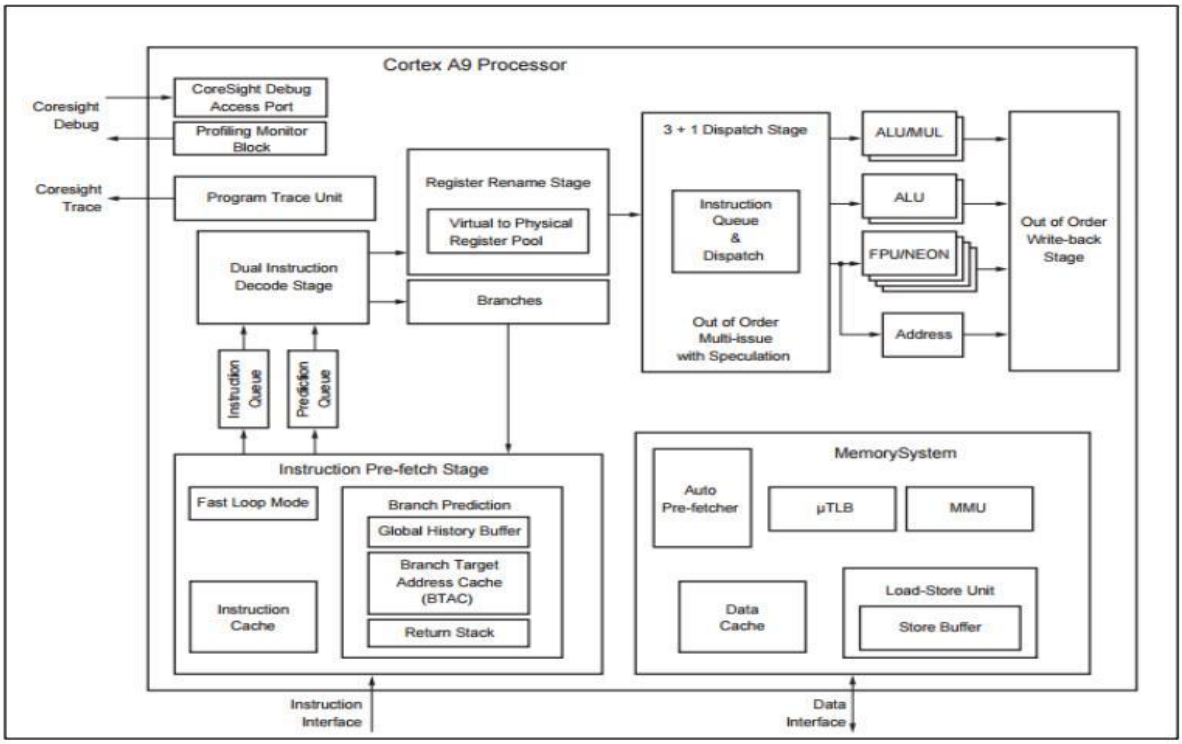
Tuy nhiên điều này khó có thể áp dụng ra thực tiễn. ESP32-CAM với bộ nhớ giới hạn (520 KB SRAM và 4 MB PSRAM) và khả năng xử lý hạn chế cũng như không có sự hỗ trợ từ thư viện OpenCV, nhóm chỉ có thể xử lý một ảnh (kích thước nhỏ) trong mỗi lần xử lý và không thể thực hiện các giải thuật phát hiện làn đường trên thời gian

thực. Đối với những ảnh có kích thước lớn và mật độ điểm ảnh dày, việc xử lý ảnh trên ESP32-CAM còn gây ra quá tải và bắt buộc phải reset thiết bị.

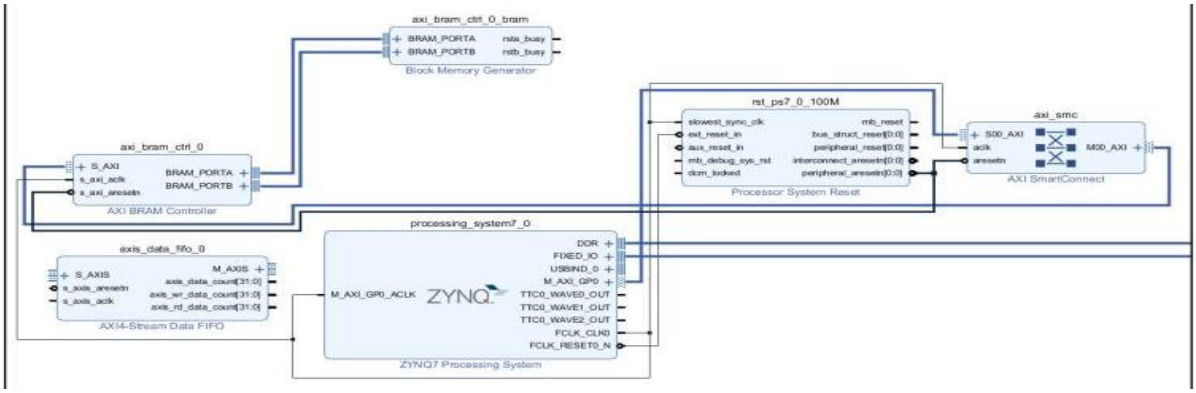
- Nhóm cũng đã hoàn thiện các giải thuật phát hiện làn đường hỗ trợ cho hệ thống xe tự lái trên FPGA ZedBoard Zynq-7000 với Module Camera OV7670:

- Bo mạch phát triển FPGA ZedBoard Zynq-7000 của hãng Xilinx:

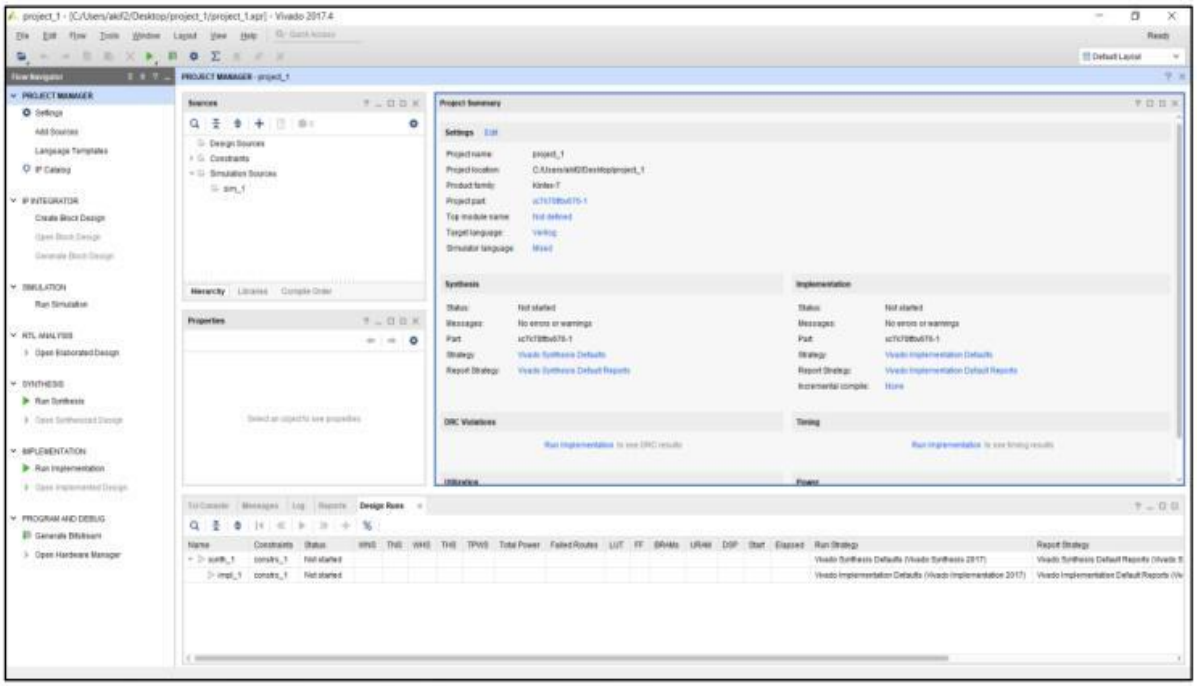
- Cấu trúc ARM Cortex A9:



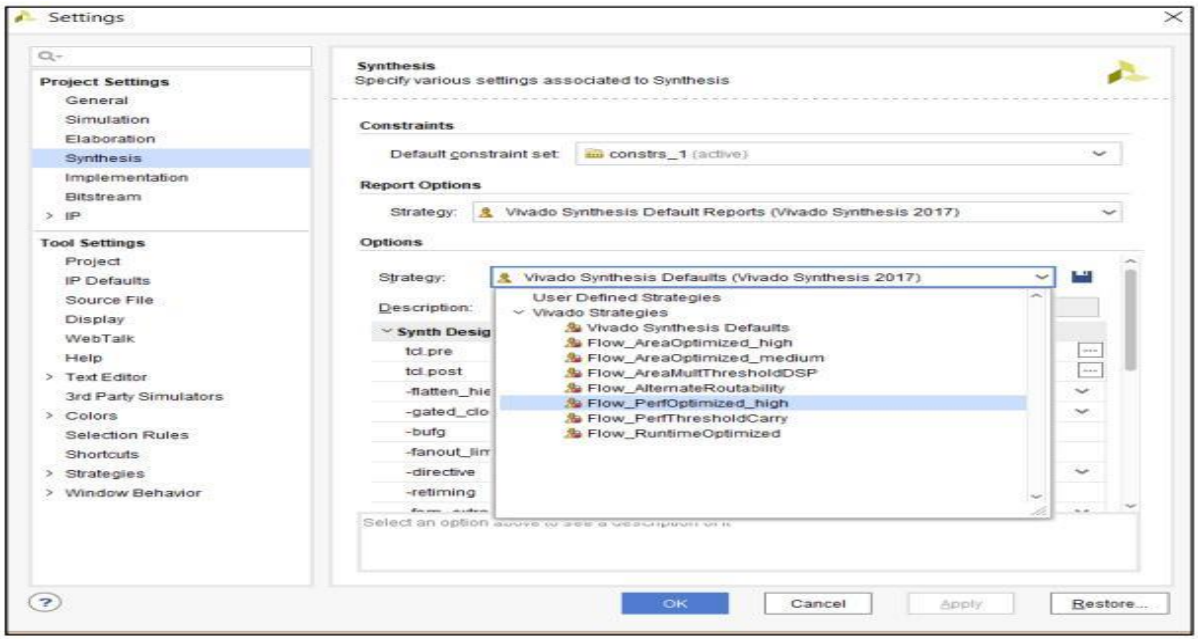
- Thiết kế các khối trên nền tảng VIVADO (phần cứng FPGA):



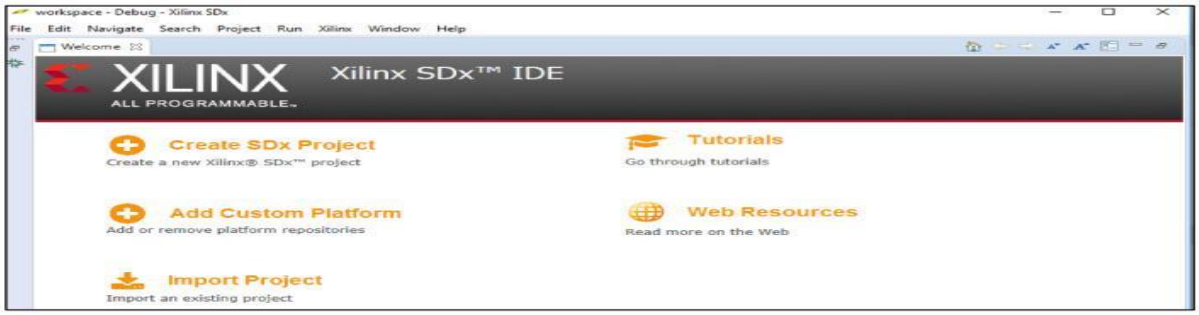
- Lập trình trên nền tảng VIVADO:



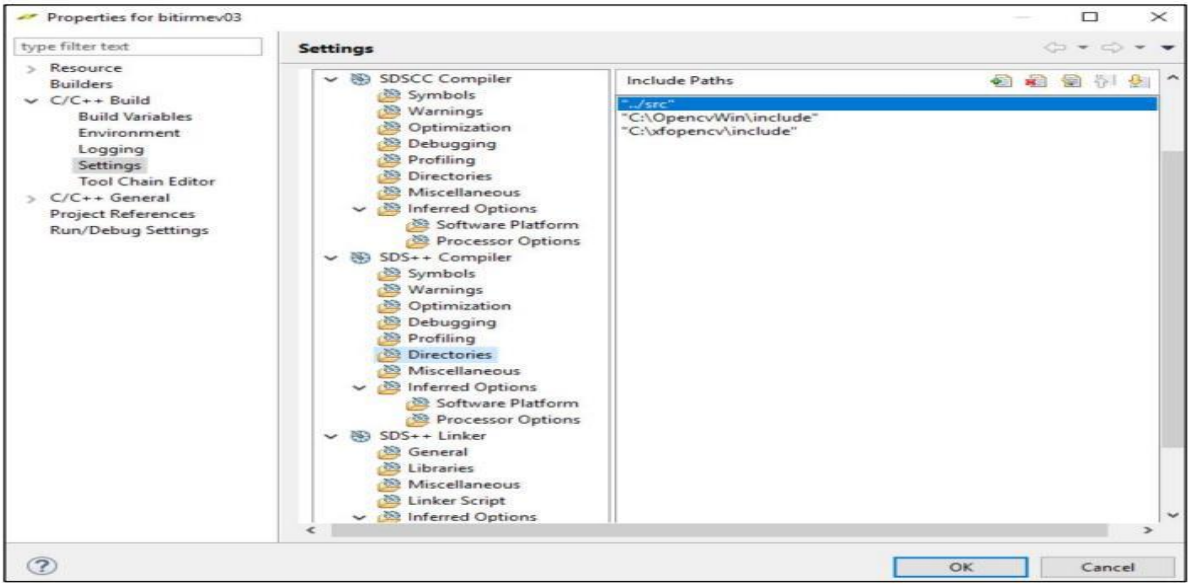
- Thiết lập cài đặt trên nền tảng VIVADO:

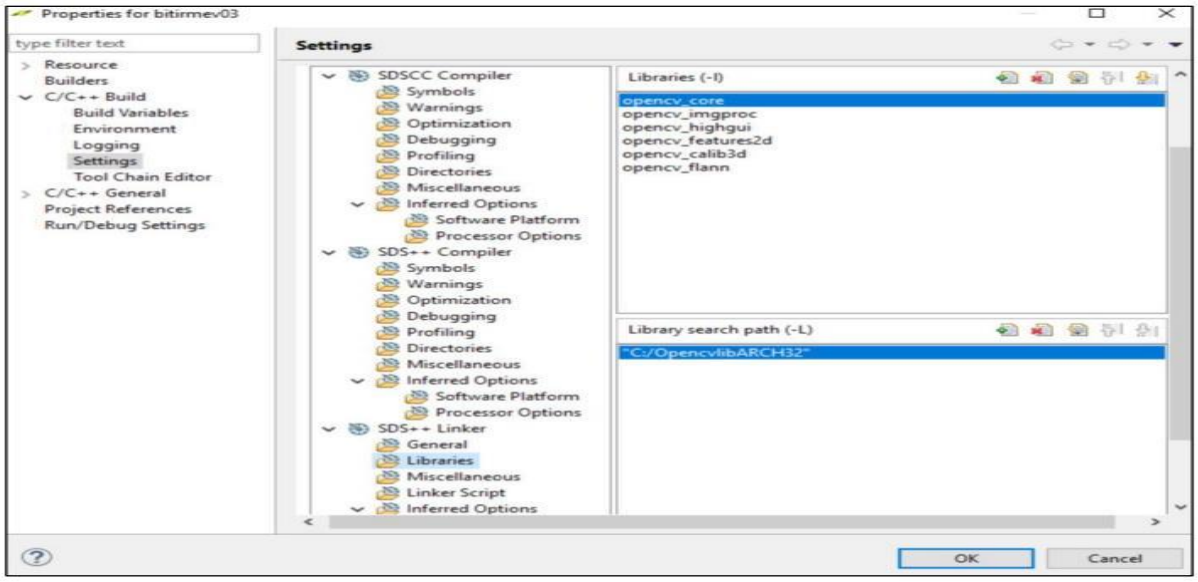


- Lập trình trên nền tảng SDSoC (kết hợp hiện thực trên phần cứng và phần mềm FPGA):

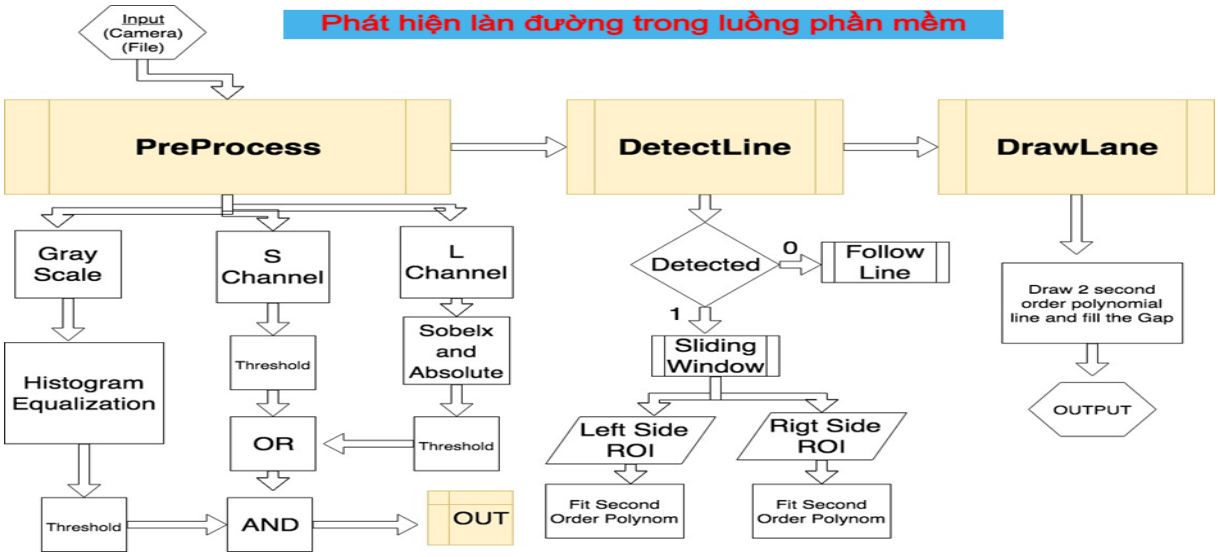


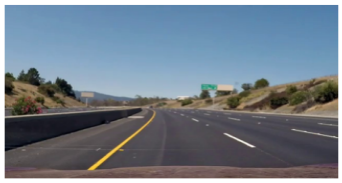
- Liên kết thư viện OpenCV và xfOpenCV:





- Các bước xử lý giải thuật trên phần mềm (C++ với thư viện OpenCV trên CPU):





a-) Original Frame (RGB)

b-) "L" Channel (HLS Color Space)

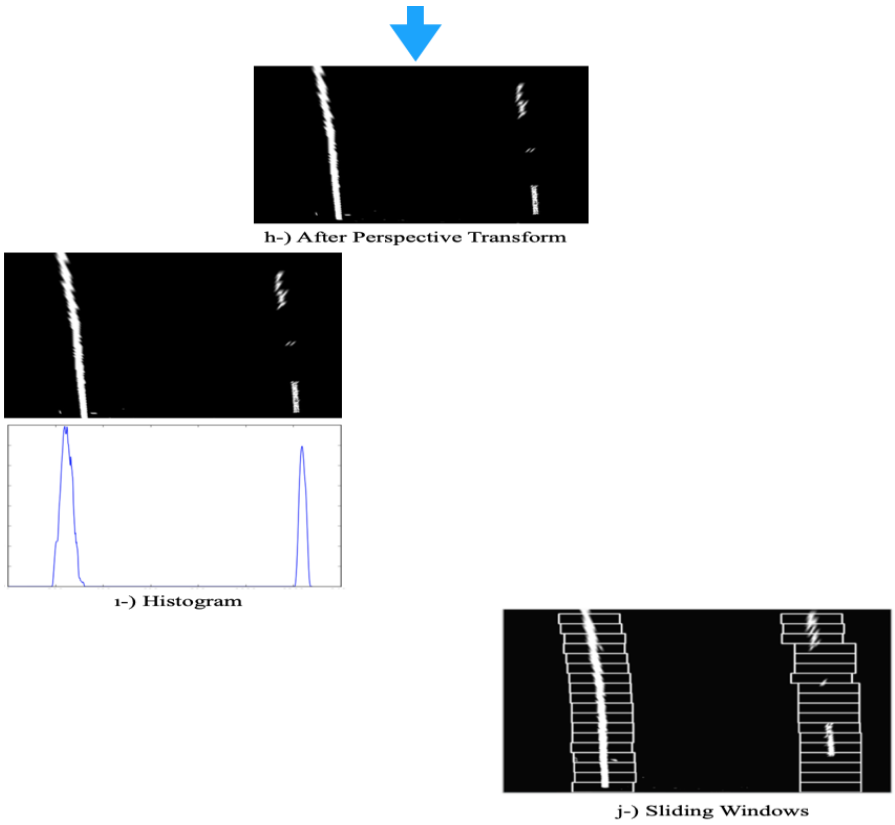
e-) "S" Channel (HLS Color Space)

c-) Horizontal derivation

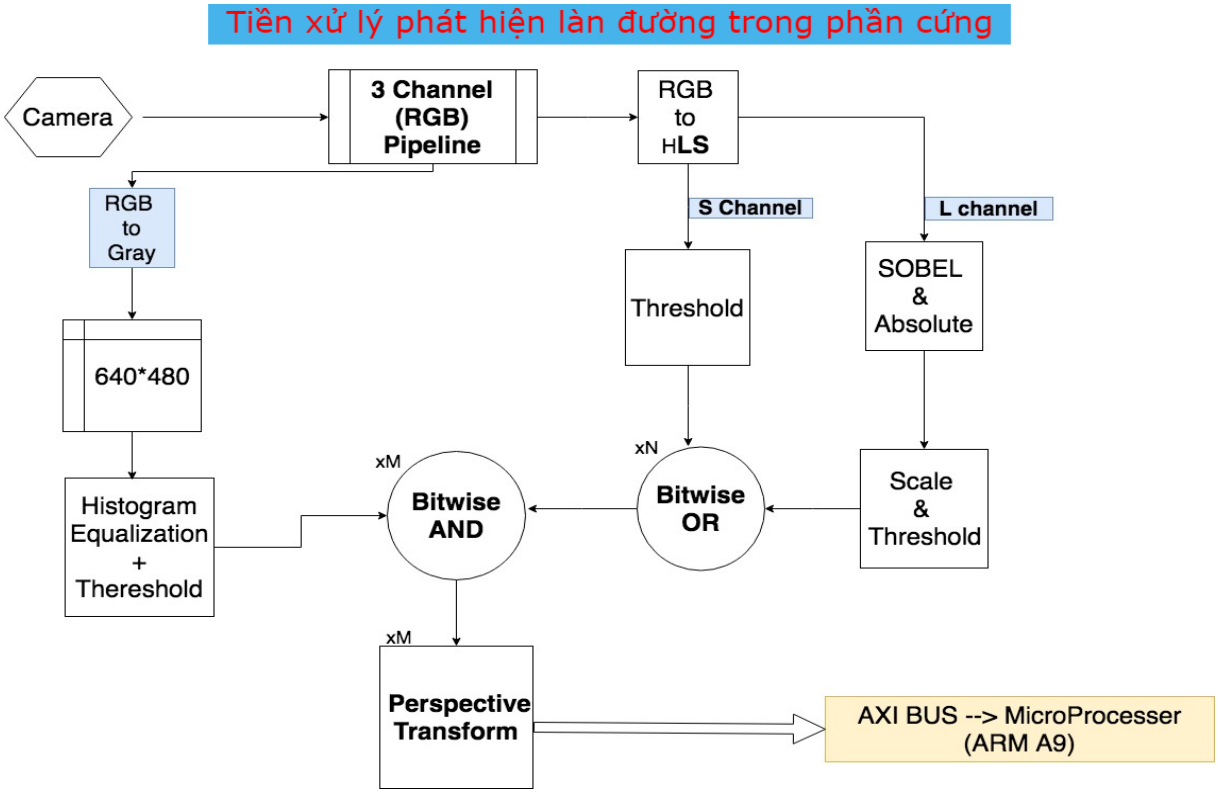
f-) Threshold

d-) Threshold

g-) Bitwise OR



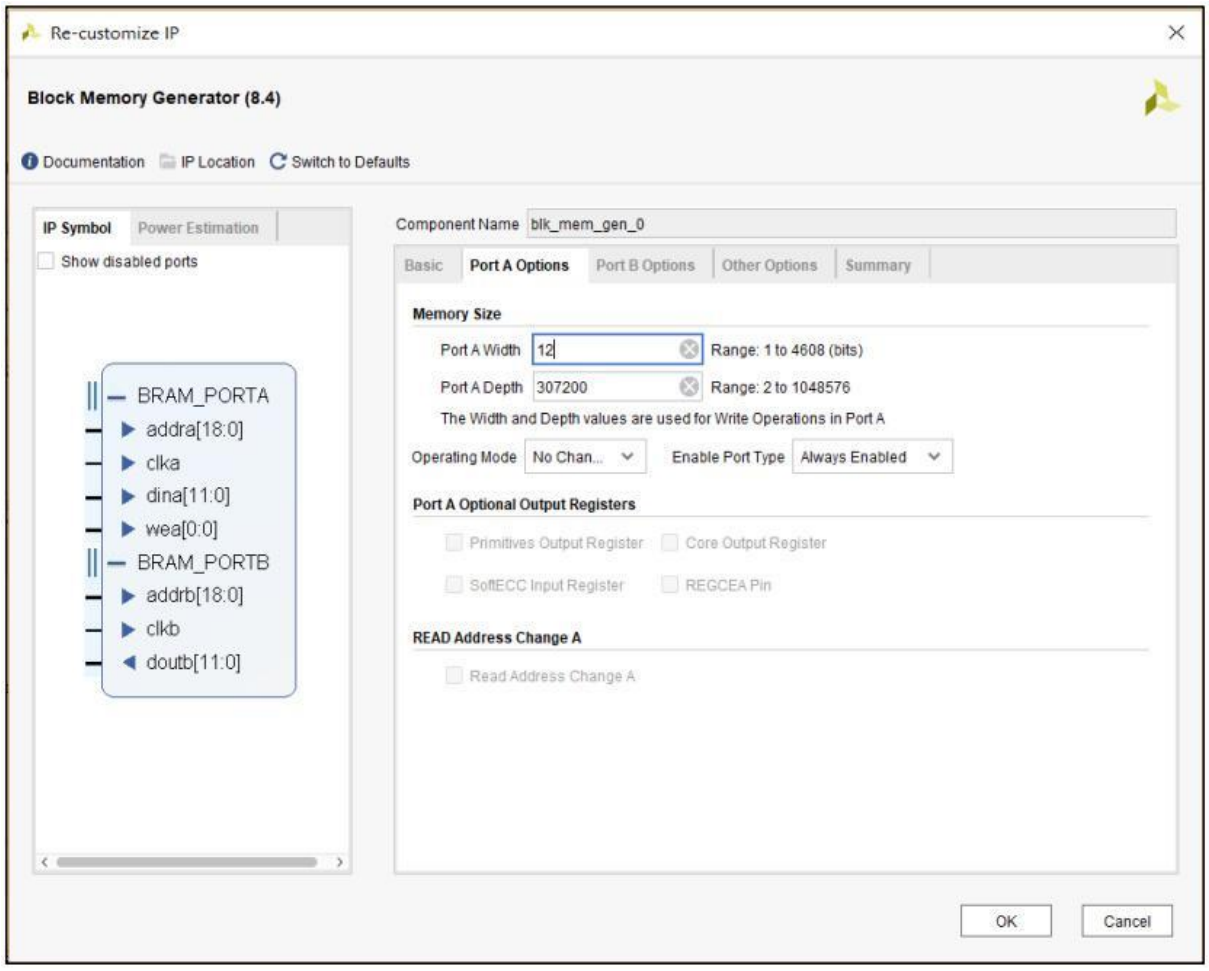
- Tiền xử lý phát hiện làn đường trong phần cứng FPGA:



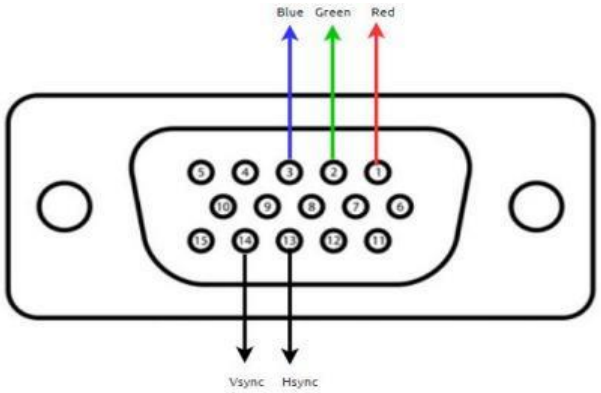
- Module Camera OV7670:

- Kết nối Module Camera OV7670 với FPGA ZedBoard Zynq-7000:

- Điều chỉnh kích thước của khối RAM:



- Kết nối VGA để hiển thị ra màn hình:



2. Nội dung nhóm dự định làm trong tương lai

● Hoàn thiện và cải tiến hơn nữa để các giải thuật phát hiện làn đường càng trở nên chính xác, và nếu có thể sẽ tiến tới đưa vào áp dụng trong thực tiễn.

● Áp dụng các kỹ thuật xử lý ảnh trong thị giác máy tính mà nhóm đã học được không chỉ trong nội dung đồ án môn học mà còn trong các dự án tương lai.

3. Khó khăn mà nhóm gặp phải trong các giai đoạn vừa rồi

● Nhóm gặp nhiều khó khăn khi tiếp cận về nội dung mới lạ, phức tạp của các giải thuật phát hiện cạnh/góc từ hình ảnh. Nhóm cũng gặp khó khăn trong việc bàn bạc, quyết định chọn lựa phần cứng để từ đó lựa chọn phương pháp, nền tảng hiện thực các giải thuật.

● Với nguồn lực, kinh tế và thời gian giới hạn, nhóm khó tiếp cận được các thiết bị phần cứng với hiệu năng xử lý cao như FPGA. Việc hiện thực các tác vụ xử lý hình ảnh mà không có thư viện OpenCV (ESP32-CAM không hỗ trợ) khiến cho nhóm gặp rất nhiều khó khăn.

II. Nội dung báo cáo

1. Đánh giá thực nghiệm

Với việc các làn đường đã được biểu thị theo các phương trình bậc 2, thuật toán phát hiện làn đường của nhóm đã có thể phát hiện các làn đường ở các khúc cua. Nhưng khi tới những khúc cua gắt, chẳng hạn như đường núi, thì thuật toán phát hiện làn đường của nhóm cũng có thể phát hiện một phần làn đường. Thuật toán cũng phát hiện được làn đường trong các tình huống ngoại cảnh khác nhau như trời râm, trời mưa, các làn đường được vẽ bằng các màu khác nhau,...

1.1. Phần mềm (C++ với thư viện OpenCV trên CPU)

- Hình ảnh trước khi xử lý trên phần mềm:

- Hình ảnh sau khi xử lý trên phần mềm:

1.2. Tiền xử lý trên phần cứng FPGA (thư viện xfOpenCV) kết hợp với C++ trên bộ xử lý ARM (thư viện OpenCV)

- Hình ảnh trước khi đưa vào xử lý trên FPGA:

- Hình ảnh sau khi đưa vào xử lý trên FPGA:

2. So sánh hiệu năng

Ta tiến hành so sánh hiệu năng của thuật toán phát hiện làn đường trên các nền tảng khác nhau cho Video lái xe trên đường cao tốc với độ phân giải 640x480.

2.1. Trên phần mềm

Chức năng	**Số lượng khung hình (frame) trong video**	**Thời gian xử lý chậm nhất (trên một khung hình)**	**Thời gian xử lý nhanh nhất (trên một khung hình)**	**Thời gian xử lý trung bình (trên một khung hình)**
Tiền xử lý	1260	81,71 ms	8,21 ms	36,99 ms
Tất cả các hàm	1260	153,31 ms	25,47 ms	77,84 ms

Tiền xử lý / Tất cả các hàm (%)	-	53,3%	32,2%	47,5%

2.2. Trên nền tảng SDSoC (kết hợp hiện thực trên phần cứng và phần mềm FPGA)

Chức năng	**Số khung hình (frame) xử lý**	**Thời gian xử lý (phần mềm)**	**Thời gian xử lý (phần cứng)**	**Tỉ lệ giảm thời gian xử lý (hiệu suất)**
Tiền xử lý (chuyển đổi phối cảnh)	1	157,07 ms	113,76 ms	Giảm khoảng 28%
Tốc độ khung hình (FPS)	1	Khoảng 6,37 FPS	Khoảng 8,79 FPS	Hiệu suất tăng khoảng 1,38 lần

2.3. Nhận xét

Sử dụng thư viện "xfOpenCV" với ngôn ngữ lập trình C++ thông qua nền tảng SDSoC (tổng hợp phần cứng mức cao) trên FPGA Zedboard đạt được tốc độ khoảng 8,78 khung hình/giây (FPS) cho chức năng tiền xử lý (không bao gồm chuyển đổi phối cảnh). Nếu khối phần cứng không được sử dụng và triển khai (không bao gồm chuyển đổi phối cảnh) trong bộ xử lý, hiệu suất là khoảng 6,37 khung hình/giây (FPS). Kết quả là cấu trúc SoC được triển khai trên Zedboard với nền tảng SDSoC và thư viện "xfOpenCV" sử dụng ngôn ngữ lập trình C++ đạt hiệu suất tốt hơn khoảng 1,38 lần so với giải pháp phần mềm thuần túy trên bộ xử lý.

Trên nền tảng VIVADO kết nối một Module Camera OV7670 đầu ra 25 MHz với ngôn ngữ đặc tả phần cứng Verilog. Chức năng tiền xử lý của thuật toán phát hiện làn đường được hiện thực trên bo mạch phát triển FPGA Zedboard và xử lý trong khoảng 12,3 ms tức là với tốc độ khung hình khoảng 81,3 khung hình/giây. Kết quả này tốt hơn 3,01 lần so với giải pháp trên phần mềm và 18,4 lần so với giải pháp trên ARM Cortex A9 (667 MHz).

3. Hoàn thiện báo cáo

Thuật toán phát hiện làn đường bao gồm ba phần chính: Tiền xử lý, phát hiện làn đường, vẽ làn đường. FPGA Zedboard dòng ZYNQ-7000 của hãng Xilinx được sử dụng làm “Hệ thống trên Chip” (SoC) để giải quyết vấn đề này. Thuật toán phát hiện làn đường được triển khai trên FPGA sau khi thuật toán được chia ra thực hiện trên phần mềm và phần cứng.

Theo phân tích thời gian của thuật toán phát hiện làn đường, phần tiền xử lý đã được thực hiện trong phần cứng. Các nền tảng SDSoC và VIVADO đã được sử dụng để thực hiện thuật toán phát hiện làn đường lập trình trên FPGA. Ngôn ngữ đặc tả phần cứng (Verilog) được sử dụng trên nền tảng VIVADO trong khi ngôn ngữ lập trình bậc cao (C++) được sử dụng trên nền tảng SDSoC. Trên nền tảng SDSoC, thư viện "xfOpenCV" cũng được sử dụng. Thư viện “xfOpenCV” là một bộ gồm hơn 50 nhân (kernels), được tối ưu hóa cho FPGA và SoC của Xilinx, dựa trên thư viện thị giác máy tính OpenCV. Các nhân trong thư viện xfOpenCV được tối ưu hóa và hỗ trợ trong bộ công cụ Xilinx SDSoC.

Một số phần của thuật toán thuộc hệ thống được triển khai trong phần cứng để hiện thực “Hệ thống phát hiện làn đường” nhanh hơn. Zedboard FPGA dòng ZYNQ-7000 của hãng Xilinx phù hợp với hệ thống vì thuật toán phát hiện làn đường có thể được tách biệt trên đó dưới dạng khối phần cứng và phần mềm. Khối phần mềm có sẵn trên bộ xử lý ARM Cortex-A9 trong Zedboard; khối phần cứng sẽ được triển khai trong các ô logic của Zedboard.

Thuật toán được chia thành 2 phần: Chức năng tiền xử lý được thực hiện trong phần cứng (logic lập trình) và các chức năng khác được thực hiện trong phần mềm (phần mềm lập trình – Arm-A9). Chức năng tiền xử lý được triển khai trong phần cứng do khả năng xử lý mạnh mẽ của nó.

Kết luận: Khi thiết kế phần cứng được thực hiện, ngôn ngữ lập trình bậc cao có thể được sử dụng cho các giải pháp tham chiếu hoặc cho các giải pháp ngắn hạn. Mặt khác, ngôn ngữ lập trình bậc thấp phù hợp với các giải pháp dự kiến sẽ nâng cao hiệu suất sau này.

TÀI LIỆU THAM KHẢO

1. *OpenCV | Real Time Road Lane Detection*, truy cập từ: https://www.geeksforgeeks.org/opencv-real-time-road-lane-detection.
2. *Edge Detection with Gaussian Blur*, truy cập từ: https://www.projectrhea.org/rhea/index.php/Edge_Detection_with_Gaussian_Blur.
3. *OpenCV - Image Thresholding*, truy cập từ: https://docs.opencv.org/5.x/d7/d4d/tutorial_py_thresholding.html.
4. *OpenCV - Canny Edge Detection*, truy cập từ: https://docs.opencv.org/5.x/da/d22/tutorial_py_canny.html.
5. *OpenCV - Hough Line Transform*, truy cập từ: https://docs.opencv.org/5.x/d3/de6/tutorial_js_houghlines.html.
6. *[Tìm hiểu] ESP32-CAM AI-Thinker Board là gì?*, truy cập từ: https://blog.mecsu.vn/esp32-cam-ai-thinker-board-la-gi.
7. *Image processing on FPGA using Verilog HDL*, truy cập từ: https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html.