ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỆN TỬ

----------o0o----------

ĐỒ ÁN 2

THỬ NGHIỆM HỆ THỐNG CHIẾU LAZER VÀO MỤC TIÊU ĐƯỢC XÁC ĐỊNH TRÊN BỨC ẢNH

GVHD: THS. TRẦN HOÀNG QUÂN

SVTH: NGUYỄN THANH TOÀN

MSSV: 2014777

TP. HỒ CHÍ MINH, THÁNG 5 NĂM 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin trân trọng cảm ơn giảng viên thầy Trần Hoàng Quân - người đã trực tiếp hướng dẫn em trong quá trình hoàn thành đồ án. Em cũng xin được gửi lời cảm ơn đến quý thầy, cô giáo trường đại học Bách Khoa Tp.Hồ Chí Minh, đặc biệt là các thầy, cô khoa Kỹ thuật Điện tử, Truyền thông - những người đã truyền lửa và giảng dạy kiến thức cho em suốt thời gian qua. Em đã cố gắng vận dụng những kiến thức đã học được và tìm tòi thêm nhiều thông tin để hoàn thành đồ án. Tuy nhiên, do kiến thức còn hạn chế và không có nhiều kinh nghiệm trên thực tiễn nên khó tránh khỏi những thiếu sót trong bài làm. Rất kinh mong quý thầy, cô cho em thêm những góp ý để kết quả của em được hoàn thiện hơn. Cuối cùng em xin cảm ơn đến gia đình, bạn bè đã luôn chia sẻ, ủng hộ, động viên và giúp đỡ trong suốt quá trình học tập của bản thân. Em xin trân trọng cảm ơn!

Tp. Hồ Chí Minh, ngày tháng năm .

TÓM TẮT BÁO CÁO ĐỒ ÁN

Đồ án này trình bày về hệ thống có chức năng hướng lazer đến vị trí mục tiêu được xác định từ bức ảnh. Vì là mô hình thử nghiệm nên dữ liệu hình ảnh được lấy từ camera của điện thoại (Ưu điểm của việc này là camera có độ phân giải tốt, góc rộng). Yêu cầu tốt thiểu mà hệ thống cần đạt được là thực hiện được các chức năng sau:

1. Hiển thị ảnh chụp từ camera của điện thoại lên màng hình máy tính.
2. Nhận biết được vị trí click chuột (xác định vị trí tọa độ của mục tiêu trong bức ảnh – tọa độ X, Y trong bức ảnh).
3. Đưa vào mô hình neural network để thực hiện dự đoán góc quay của động cơ servo nhằm hướng servo đến vị trí của mục tiêu cần chiếu).
4. Thực hiện giao tiếp với MCU điều khiển servo.

Các công nghệ, kỹ thuật được sử dụng:

1. Chức năng hiển thị ảnh từ camera của điện thoại, Ta thực hiện bằng thư viện pygame của python. Về phần nhận hình ảnh từ camera của điện thoại ta sẽ xây dựng một web app để upload ảnh từ điện thoại lên server. Về server sử dụng chương trình XAMPP.
2. Nhận biết vị trí click chuột, Sử dụng thư viện pygame.event của python.
3. Mô hình neural network, Sử dụng thư viện hỗ trợ phổ biến là tensoflow.
4. Thực hiện giao tiếp với MCU, Sử dụng giao thức UART.

Mục Lục

Danh mục hình ảnh

1. GIỚI THIỆU

1.1 Tổng quan

Xuất phát từ thực tế hiện nay, Công nghệ AI hay cụ thể hơn và thị giác máy tính phát triển mạnh mẻ và rực rỡ. Dựa vào một số mô hình open source ta có thể dễ dàng phân loại, xác định vị trí của mô vật thể trong bức ảnh. Ví dụ như mô hình Yolo có rất nhiều phiên bản khác nhau và rất phù hợp cho ứng dụng phân loại và xác định vị trí của một vật thể trong bức ảnh. Chính vì vậy, Việc xây dựng một mô hình có khã năng hướng lazer đến một mục tiêu từ tọa độ của mục tiêu đó trong bức ảnh sẽ có rất nhiều ứng dụng trong đời sống.

Một số ví dụ tiêu biểu như hướng vòi phun chửa cháy đến vị trí của ngon lửa được xác định trong khung ảnh, robot chữa cháy, hay hệ thống phòng không tầm thấp (hướng pháo phòng không theo máy bay chiến đầu khi ở độ cao thấp). Hệ thống bám mục tiêu cho flycam.

1.2 Các hệ thống tương tự đã được ứng dụng trong thực tế.

Súng phun nước chữa cháy tự động (Automatic fire fighting water cannon). Hệ thống chữa cháy phun nước tự động trong không gian lớn là một hệ thống phun lửa công nghệ cao thế hệ mới tích hợp hệ thống báo cháy và chữa cháy với nhau. Loạt sản phẩm này có chức năng phát hiện cháy tích cực, định vị tự động và dập lửa bằng tia nước cố định và hệ thống chữa cháy tự động đặt lại. Trong trường hợp cháy nổ lại, hệ thống sẽ tự động khởi động lại để chữa cháy bằng tia nước. Loại sản phẩm này có ưu điểm là lắp đặt thuận tiện, cấp nước và nguồn điện đơn giản, chi phí bảo trì thấp, tiết kiệm nước, giảm tổn thất hoặc hư hỏng thứ hai gây ra do phun nước chữa cháy. Loại hệ thống này có tỷ lệ hiệu suất giá cao. Được sử dụng rộng rãi trong cấu trúc cao hoặc không gian rộng lớn như rạp chiếu phim, nhà kho, nhà máy, phòng tập thể dục, thính phòng, sảnh khởi hành, Trung tâm hội nghị & triển lãm, khách sạn, bãi đỗ xe.

Figure 1: Súng phun nước chữa cháy tự động (Automatic fire fighting water cannon)

1.3 Nhiệm vụ của đồ án.

1.3.1 Tìm hiểu về neural network, xây dựng, huấn luyện và sử dụng mô hình neural network.

Yêu cầu: Tìm hiểu cách hoạt động của mô hình neural network. Xây dựng mô hình có khả năng thực hiện được yêu cầu của đồ án. Huấn luyện mô hình dự đoán được góc quay cho servo.

Kết quả tối thiểu cần đặt được: Thu thập được bộ dữ liệu góc quay servo tương ứng với tọa độ mục tiêu trong ảnh (kích thước bộ dữ liệu tối thiểu đạt 100-200 mẫu dữ liệu). Xây dựng được mô hình dự đoán góc quay cho servo hướng lazer đến mục tiêu có độ chính xác chấp nhận được (sai số không lớn hơn 5% khoảng cách từ mục tiêu đến lazer)

1.3.2 Tìm hiểu về vi điều khiển ATMEGA328P.

Yêu cầu: Thực hiện giao tiếp giữa chương trình python trên máy tính và vi điều khiển ATMEGA328P.

Kết quả tối thiểu cần đạt được: Truyền được thông tin góc quay được dự đoán từ chương trình python trên máy tính đến vi điều khiển ATMEGA328P. Vi điều khiển thực hiện phản hồi xác nhận góc quay nhận được.

1.3.3 Thiết kế hệ thống điều khiển lazer

Yêu cầu: Lazer có thể quay tự do trong hệ tọa độ cầu (gốc quay phi và gốc quay theta)

Yêu cầu tối thiểu cần đạt được: Điều khiển lazer đến mục tiêu được xác định từ chương trình python trên máy tính với độ sai số có thể chấp nhận được (không quá 5% khoảng cách từ lazer đến mục tiêu)

2. LÝ THUYẾT

2.1 Mô hình neural network

Neural Network đọc tiếng việt là Mạng nơ-ron nhân tạo, đây là một chuỗi những thuật toán được đưa ra để tìm kiếm các mối quan hệ cơ bản trong tập hợp các dữ liệu. Thông qua việc bắt bước cách thức hoạt động từ não bộ con người. Nói cách khác, mạng nơ ron nhân tạo được xem là hệ thống của các tế bào thần kinh nhân tạo[1]

Mạng Neural Network là sự kết hợp của các tần hay lớp (layer). Và mỗi một mạng Neural Network cơ bản thường bao gồm 3 loại tầng (layer) là:

Tầng input layer (tầng vào): Tầng này nằm bên trái cùng của mạng, thể hiện cho các đầu vào của mạng. Đây là tầng không thể thiếu của một mô hình neural network.

Tầng output layer (tầng ra): Là tầng bên phải cùng và nó thể hiện cho những đầu ra của mạng. Đây cũng là một tầng không thể thiếu của một mô hình neural network.

Tầng hidden layer (tầng ẩn): Tầng này nằm giữa tầng vào và tầng ra nó thể hiện cho quá trình suy luận logic của mạng. Đây là tầng không bắt buột phải có của một mô hình

[1] Itnavi, Tổng quan về Neural Network(mạng Nơ Ron nhân tạo) là gì?, (13/05/2021) truy cập tại https://itnavi.com.vn/blog/neural-network-la-gi/

neural network, tuy nhiên trong đa phần các trường hợp thì một mô hình neural network đều có ít nhất một tầng hidden layer.

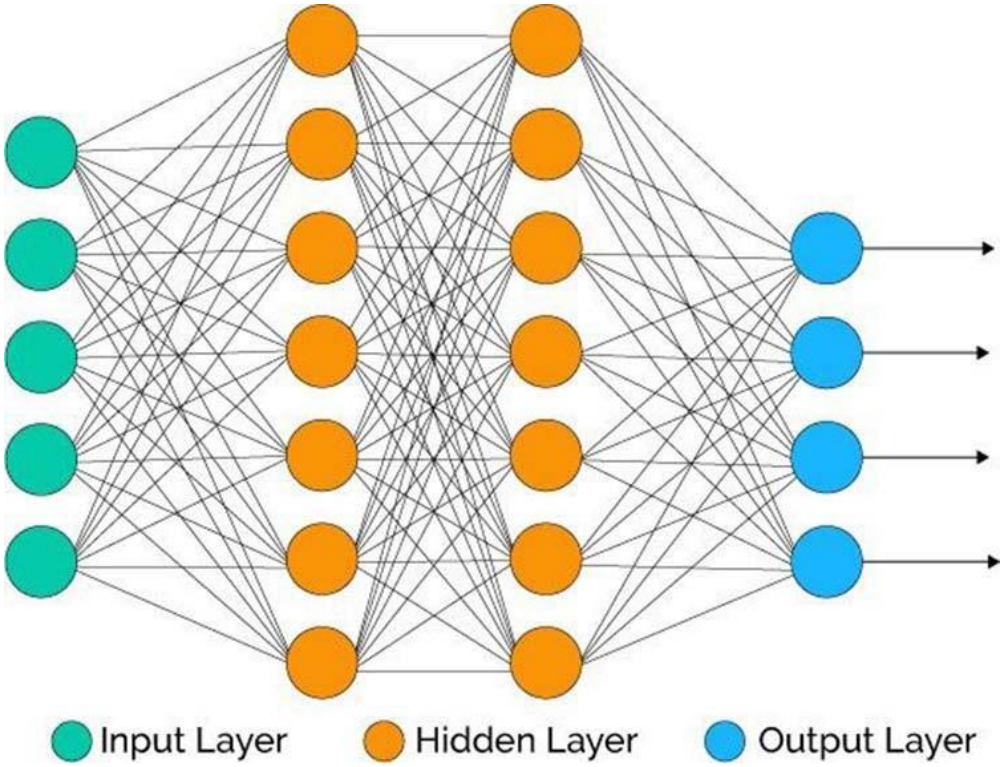

Figure 2: Cấu trúc cơ bản của mạng neural network

Đối với mô hình neural network có 2 quá trình quan trọng nhất là: Feedforward (lan truyền tiến) và Backpropagation (lan truyền ngược)

Quá trình lan truyền tiến (Feedforward):

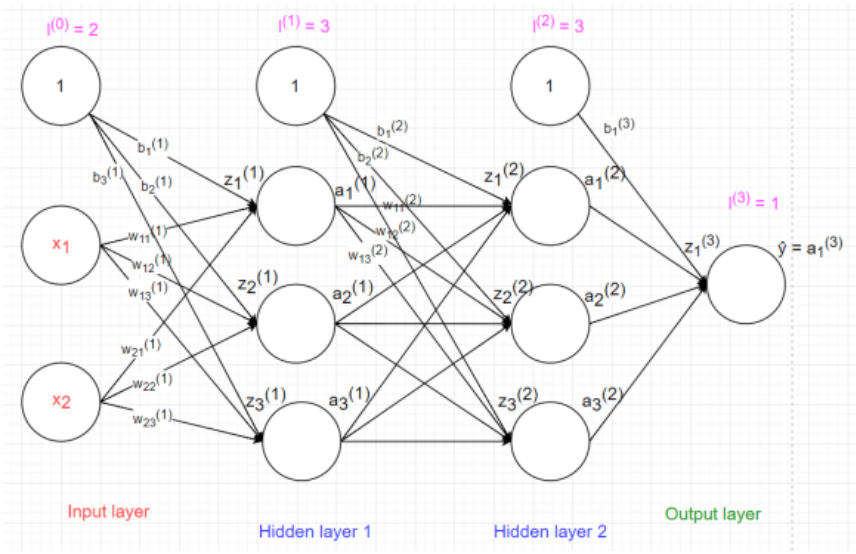

Figure 3: Cấu trúc mạng neural network chi tiết

Về cơ bản quá trình lan truyền tiến dùng để xác định giá trị của các node của các layer trong mạng neural network. Bởi vì các giá trị của node trong một layer sẽ được xác định từ giá trị của các node trong một layer trước đó nên quá trình này được gọi là quá trình lan truyền tiến.

$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} a_1^{(0)} * w_{11}^{(1)} + a_2^{(0)} * w_{21}^{(1)} + a_3^{(0)} * w_{31}^{(1)} + b_1^{(1)} \\ a_1^{(0)} * w_{12}^{(1)} + a_2^{(0)} * w_{22}^{(1)} + a_3^{(0)} * w_{32}^{(1)} + b_2^{(1)} \\ a_1^{(0)} * w_{13}^{(1)} + a_2^{(0)} * w_{23}^{(1)} + a_3^{(0)} * w_{33}^{(1)} + b_3^{(1)} \end{bmatrix}$$

$$= (W^{(1)})^T * a^{(0)} + b^{(1)}$$

$$a^{(1)} = \sigma(z^{(1)})$$

Figure 4: Công thức tính giá trị của một node trong một layer

Như công thưc trên ta có thể thấy giá trị của một node trong một layer sẽ được các định từ tích của ma trận trong số kết nối giữa layer này và layer trước đó và một activation function. Tác dụng của activation function là khử, loại bỏ thành phần tuyến tính của mạng neural network.

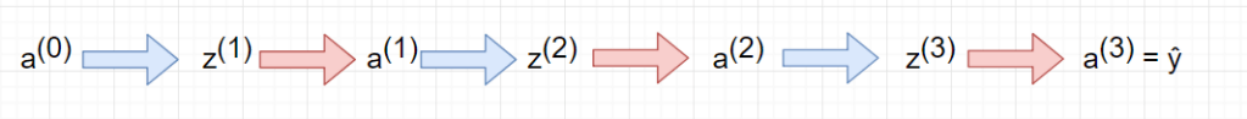

Figure 5: Lan truyền thuật của neural network

Trong đó:

a(n): Là giá trị của các node trong một layer.

z(n): Là kết quả của phép nhân ma trận trong số và giá trị các node của layer trước đó.

Mũi tên màu xanh: Thể hiện cho quá trình nhân ma trận trọng số và giá trị các node của layer trước đó.

Mũi tên màu đỏ: Thể hiện cho quá trình thực hiện activation function.

Đối với quá trình lan truyền ngược (Backpropagation) có một hàm quan trọng là loss function hay cost function (Hàm này đặt trưng cho độ sai khác giữa kết quả dựa đoán và dữ liệu dùng dể huấn luyện)

$$L=-(yi*log(yi\hat{}))+(1-yi)*log(1-yi\hat{}))$$

Trong đó:

L: Loss function hay cost function.

yi: Giá trị đầu ra của dữ liệu huấn luyện.

yi^: Giá trị dự đoán của mô hình.

Thuật toán Gradient descent: Đây là một thuật toán vô cùng quan trọng trong việc cập nhật các trọng số của mô hình neural network. Mục đích của thuật toán gradient descent dùng để thay đổi trọng số của mô hình neural network sau cho giảm loss function.

Như vậy ta có thể nhận xét quá trình lan truyền bắt đầu từ đạo hàm của loss function

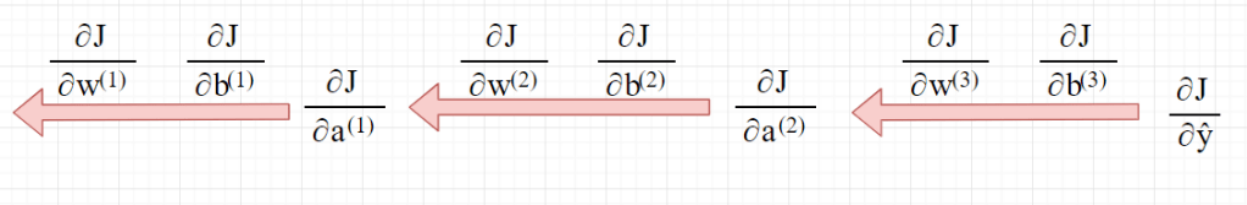

Figure 6: Lan truyền ngược trong mô hình neural network

theo giá trị dự đoán, tiếp đến lần lược đến các trọng số của các lớp lan truyền ngược về phía input layer.

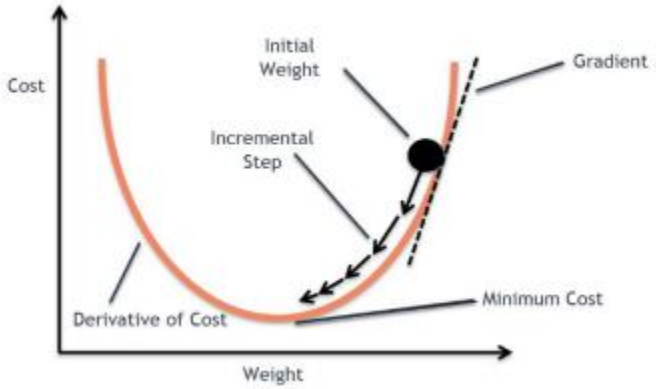

Figure 7: Minh họa thuật toán Gradient descent với một biến

Gradient Descent là một thuật toán tối ưu lặp (iterative optimization algorithm) được sử dụng trong các bài toán Machine Learning và Deep Learning (thường là các bài toán tối ưu lồi — Convex Optimization) với mục tiêu là tìm một tập các biến nội tại (internal parameters) cho việc tối ưu models.

Trong đó:

• Gradient: là tỷ lệ độ nghiêng của đường dốc (rate of inclination or declination of a slope). Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng

với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative thay cho Gradient.

Về cơ bản thì đều được thực thi như sau:

- Khởi tạo biến nội tại.
- Đánh giá model dựa vào biến nội tại và hàm mất mát (Loss function).
- Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát (finding optimal points).
- Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.

Công thức cập nhật cho thuật toán có thể được viết là: $\theta^{(\text{nextStep})} = \theta - \eta\Delta_\theta$

Ở đây ta có thông số rất quan trọng trong quá trình học của model. Đó là hệ số learning rate. Thông số này liên quan mật thiết đến tốc độ học có model.

Minh họa về thuật toán Gradient descent nhiều biến.

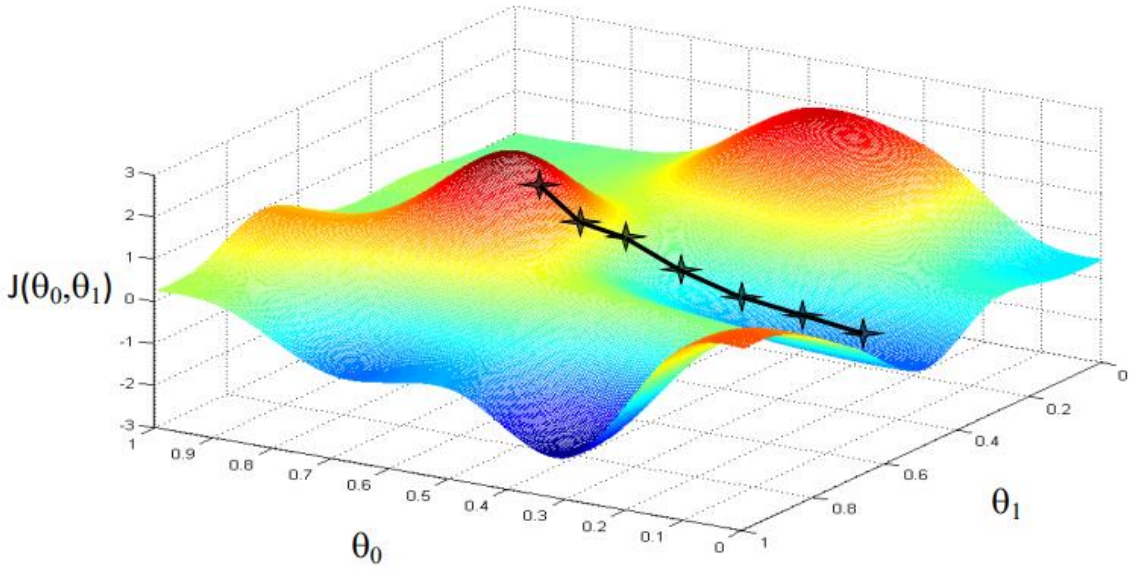

Figure 8: Minh họa thuật toán gradient descent với nhiều biến

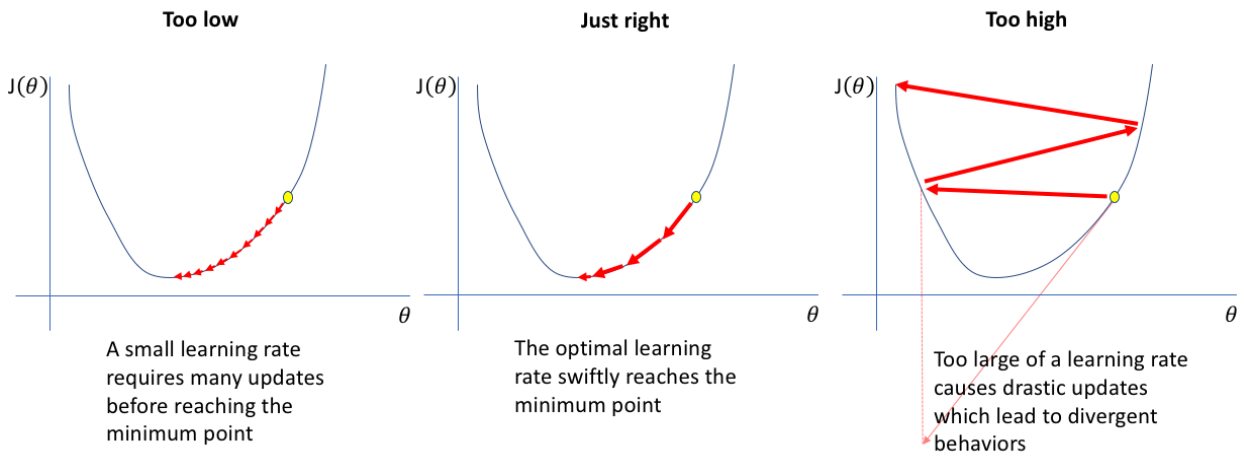

Figure 9: Minh họa việc chọn đúng learning rate

Như ta có thể thấy khi learning rate nhỏ: Thì tốc độ học của mô hình sẽ chậm, Ta sẽ cần nhiều lần học hơn để mô hình có thể dự đoán chính xác.

Khi learning rate vừa đủ: Mô hình sẽ có tốc độ học tốt.

Khi learning rate quá lớn: Thì hàm cost function sẽ dao động qua lại giữa vị trí cực tiểu làm cho quá trình học trở nên khó khăn hoặc sẽ không thành công.

Thông thường ta sẽ chọn learning rate vào khoảng: 0.001

Trên đây là phần trình bày về những kiến thức cơ bản nhất về mô hình neural network, Cũng là phần kiến thức quan trọng nhất, quyết định nhất liên quan đến việc hoàn thành đồ án.

2.2 Thư viện pygame.

Pygame là một thư viện của ngôn ngữ lập trình Python và là một tập hợp các mô-đun Python được thiết kế riêng để lập trình trò chơi. Pygame được viết bởi Pete Shinners thay thế cho chương trình PySDL sau khi quá trình phát triển dự án này bị đình trệ. Chính thức phát hành từ năm 2000, Pygame được phát hành theo phần mềm miễn phí GNU Lesser General Public License.

Pygame có thể chạy trên nhiều nền tảng và hệ điều hành khác nhau. Với thư viện pygame trong Python, các nhà phát triển có thể sử dụng công cụ và chức năng mở rộng để tạo ra các trò chơi nhập vai ấn tượng. Bởi vậy, Pygame đang ngày càng phổ biến với nhà phát triển vì tính đơn giản, linh hoạt, dễ sử dụng.

Dưới đây là một số đặc điểm nổi bật của Pygame:

Pygame sử dụng Simple DirectMedia Layer (SDL), một thư viện phát triển đa nền tảng cho phép các nhà phát triển có thể truy cập vào phần cứng máy tính như đồ họa, âm thanh và thiết bị đầu vào.

Xây dựng các trò chơi trên nhiều nền tảng khác nhau như Windows, Mac, Linux thậm chí là cả các thiết bị di động

Nhà phát triển có thể quản lý tất cả các yếu tố trong quá trình phát triển trò chơi. Đó có thể là các chức năng như xuất đồ họa, xử lý sự kiện, hoạt ảnh, hiệu ứng âm thanh và phát lại nhạc

Cung cấp nhiều chức năng mở rộng hỗ trợ nhà phát triển tập trung phát triển trò chơi

API trực quan và dễ hiểu, hỗ trợ người mới sử dụng hay cả những nhà phát triển có kinh nghiệm đều có thể truy cập được

Nguồn tài nguyên và tài liệu phong phú, các nhà phát triển có thể sử dụng các mã nguồn mở miễn phí để phát triển dự án của mình

Tính đa phương tiện giúp nhà phát triển có thể ứng dụng để xử lí hình ảnh hay video, mô phỏng, công cụ giáo dục….

Thư viện pygame không chỉ gói gọn trong việc thực hiện game. Mà thư viện pygame còn giúp thực hiện đồ họa, xữ lý sự kiện một cách đơn giản và hiệu quả nên thư viện pygame được lựa chọn đển thực hiện chức năng giao tiếp với người dùng.

2.3 Vi điều khiển ATMEGA328P

ATmega328P là một bộ vi điều khiển tiên tiến và nhiều tính năng. Nó là một trong những vi điều khiển nổi tiếng của Atmel vì nó được sử dụng trong bo mạch arduino UNO. Nó là một bộ vi điều khiển thuộc họ vi điều khiển megaMVR của Atmel (Cuối năm 2016, Atmel được Microchip Technology Inc mua lại). Các vi điều khiển được sản xuất trong họ megaMVR được thiết kế để xử lý các bộ nhớ chương trình lớn và mỗi vi điều khiển trong họ này chứa lượng ROM, RAM, các chân I / O và các tính năng khác nhau và được sản xuất với các chân đầu ra khác nhau, từ 8 chân đến hàng trăm chân.

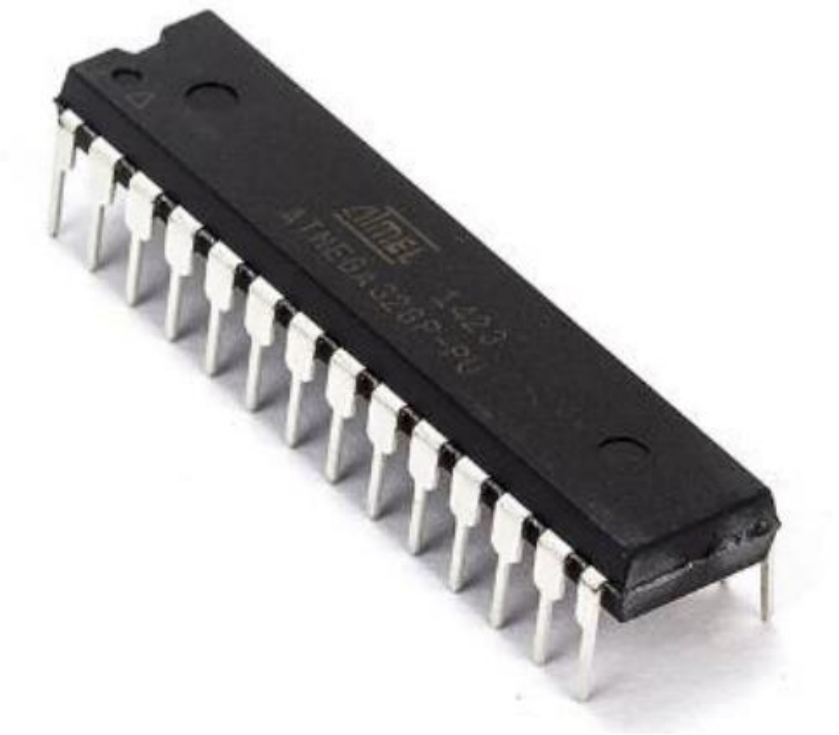

Figure 10: Vi điều khiển ATMEGA328P

Mạch bên trong của ATmega328P được thiết kế với tính năng tiêu thụ dòng điện thấp. Con chip này chứa 32 kilobyte bộ nhớ flash trong dùng để chứa chương trình, 1 kilobyte EEPROM và 2 kilobyte SRAM. EEPROM và bộ nhớ flash là bộ nhớ lưu thông tin và thông tin đó vẫn tồn tại và không bị xóa mỗi khi nguồn điện bị ngắt. Còn SRAM là bộ nhớ chỉ lưu thông tin cho đến khi có điện và khi ngắt nguồn điện tất cả thông tin được

lưu trong SRAM sẽ bị xóa. Vì vậy bộ nhớ EEPROM có thể dùng để ghi dữ liệu mà có thể dùng lại sau đó ngay cả khi đã bị ngắt nguồn cung cấp. (Xem chi tiết trong datasheet của chip ATMEGA328P). Tốc độ xung nhịp 16 megahertz.

Vì để dễ dàng thực hiện đồ án và tránh khó khăn khi thực hiện trên mạch in hoặc breakboard thì ta có thể sử dụng kit Arduino uno. Để nhanh chống thực hiện mà không cần phải làm mạch in hoặc dùng breakboard.

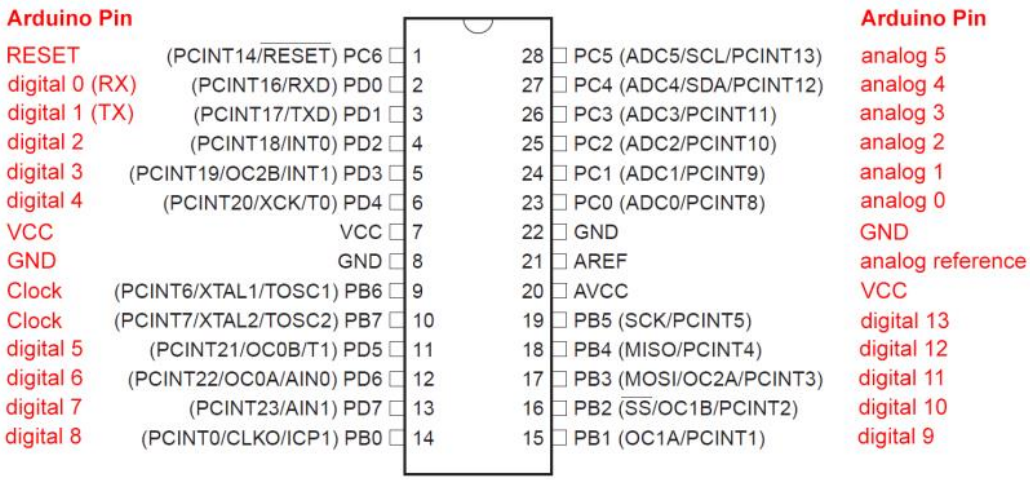

Figure 11: Sơ đồ chân của ATMEGA328P ứng với arduino pin

3 THIẾT KẾ VÀ THỰC HIỆN ĐỒ ÁN.

3.1 Thiết kế và thực hiện chương trình thu thập dữ liệu.

Nhiệm vụ chính của đồ án là từ một tọa độ của mục tiêu trong bứa ảnh, mô hình sẽ trả về góc điều khiển động cơ servo hướng lazer đến mục tiêu đã chọn. Vì vậy nhiệm vụ thu thập dữ liệu để mô hình có thể dự đoán góc cho servo.

Kiến trúc cơ bản của model là có 2 code đầu vào X và Y là vị trí pixel chưa vị trí trung tâm vật thể được chọn trong bức ảnh. Đầu ra của mô hình là 2 góc quay phi1 và phi2 của servo (phi1 lazer quay theo góc phi trong hệ tọa độ cầu, phi2 điều khiển quay lazer theo chiều góc theta trong hệ tọa độ cầu).

Vì vậy ta sẽ lưu dữ liệu với định dạng một mảng dữ liệu 2 chiều [Nx4]

Trong đó:

- N là số hàng: Là số lượng mẫu dữ liệu được thu thập.
- 4 là số cột: Như vậy một hàng sẽ có một cột.
    + Cột 1: Chứa tọa độ X của mục tiêu.
    + Cột 2: Chứa tọa độ Y của mục tiêu.
    + Cột 3: Chứa góc quay Phi1 của servo.
    + Cột 4: Chứa góc quay Phi2 của servo.

```
array([[ 0.61484375,  0.54895833, -0.23333333,  0.07777778],
       [ 0.38828125,  0.52604166,  0.01111111,  0.05555556]])
```

Figure 12: Minh họa tổ chức lưu trữ dataset

Phía trên và ví dụ về 2 mẫu dữ liệu được lưu trữ trong một array.

Lưu ý một vấn đề quan trọng là khi lưu dữ liệu để dễ dàng cho quá trình huấn luyện mô hình và tránh những vấn đề xãy ra trong quá trình huấn luyện ta nên thực hiện ánh xạ dữ liệu input và output về khoảng từ -1 đến 1. Lý do của việc này rất đơn giản vì khi thực hiện thuật toán Gradient descent thì đối với biến đầu vào có giá trị lớn hơn thì sẽ ảnh hưởng nhiều hơn đến thuật toán Gradient descent. Để hiểu hơn về vấn đề này ta sẽ xem xét hai hình minh họa sau đây:

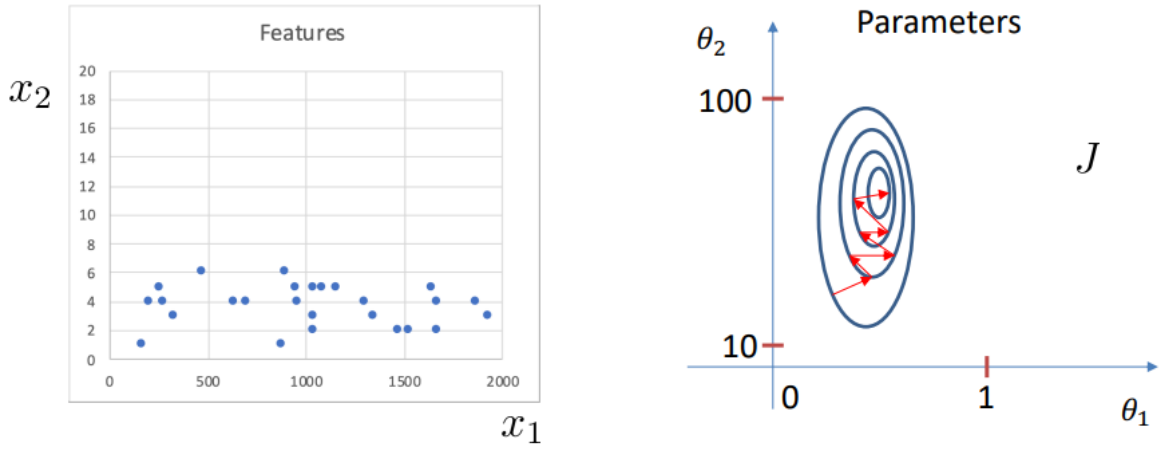

Figure 13: Minh họa khi không scale các đầu vào tương ứng nhau

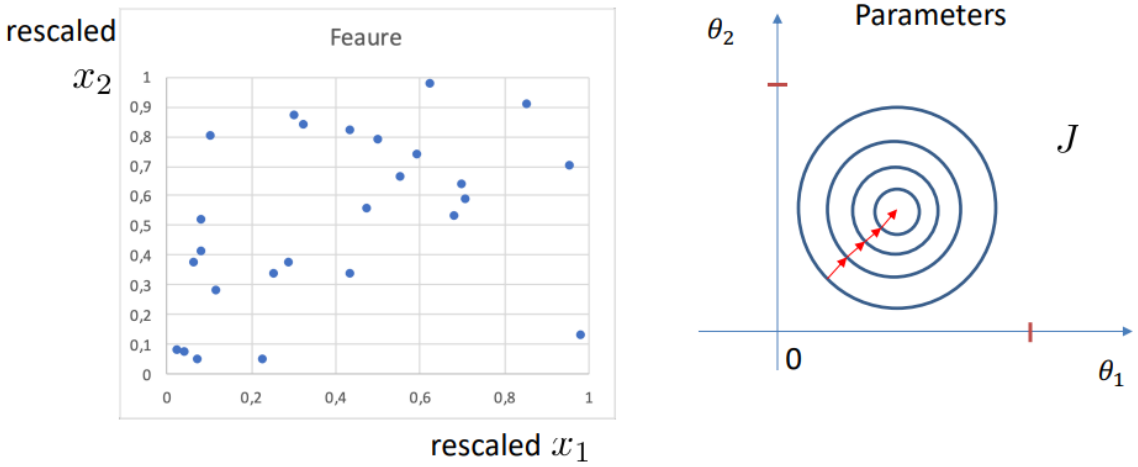

Figure 14: Minh họa khi có scale đầu vào tương ứng nhau

Như vậy ta có thể thấy khi ánh xạ các biến đầu vào về các khoảng giá trí tương ứng nhau thì sự ảnh hưởng của các biến đầu vào lên thuật toán gradient descent tương tự nhau. Điều này là cho quá trình học trở nên ổn định hơn, hội tụ tốt hơn, tiến đến vị trí có hàm loss function nhanh hơn, ổn định hơn. Góp phần làm giảm thời gian học của môt hình.

Sơ đồ khối thực hiện chương trình thu thập dataset.

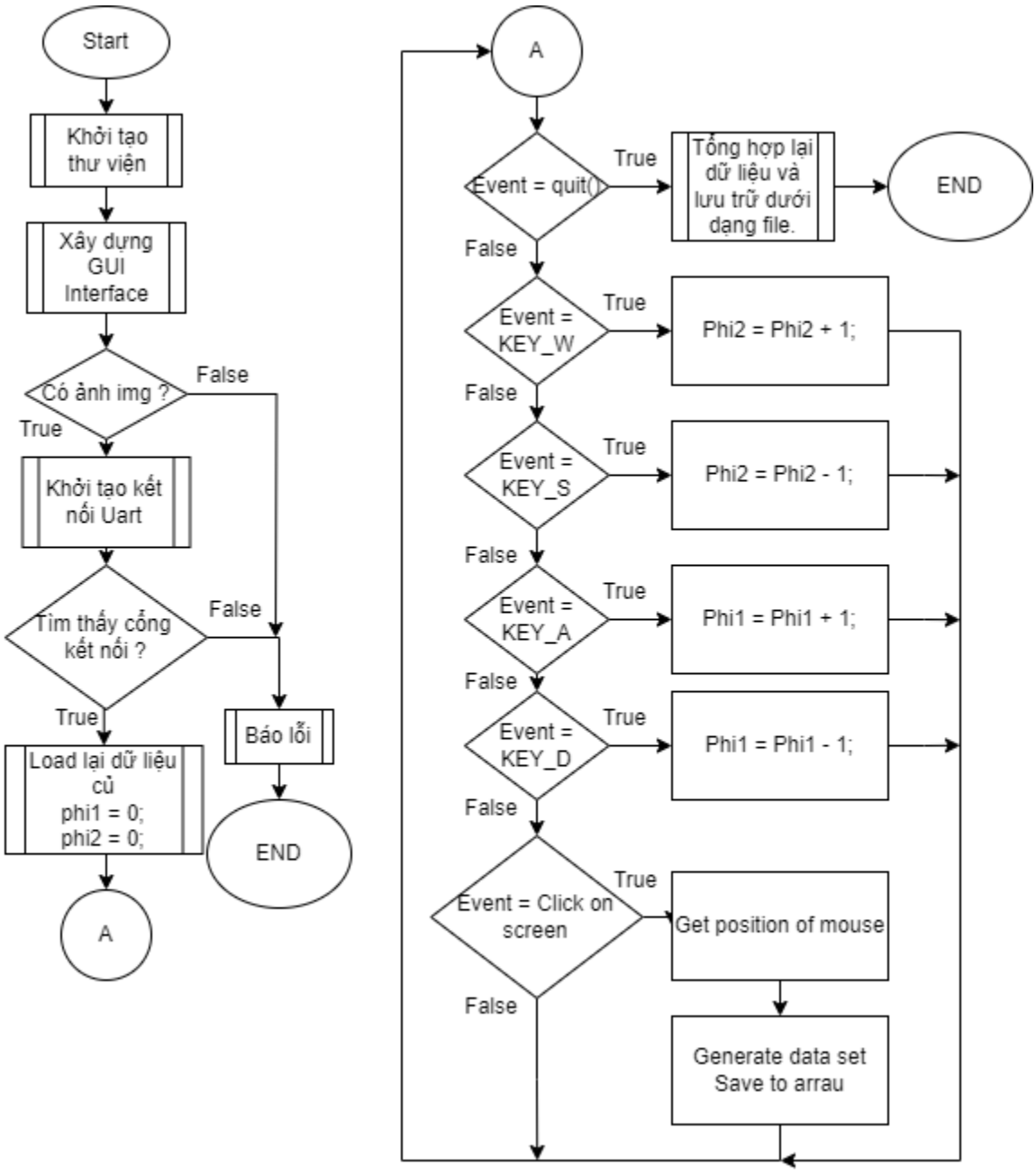

Figure 15: Flowchart chương trình thu thập dữ liệu

Về phần xây dựng GUI và load image vào GUI ta nhờ sự hỗ trợ của thư viên pygame như đã trình bày bên trên.

```
# Pygame for interface
pygame.init()
# setting frame/window/surface with some dimensions
window = pygame.display.set_mode((1040, 780))
# to set title of pygame window
pygame.display.set_caption("Get_dataSet")
clock = pygame.time.Clock()
# creating image object
try:
    image = pygame.image.load('./picture/img.jpg')
    image = pygame.transform.scale(image, (1040, 780))
except:
    print('can find img for this position')
    image = pygame.image.load('./picture/imgNotFound.jpg')
    image = pygame.transform.scale(image, (1040, 780))
```

Tên bức ảnh được load vào GUI là img.jpg. Và được scale về kích thước 1040 x 780 bởi vì hình ảnh được chụp trên điện thoại có kích thước là 4160 x 3120 với kích thường này thì quá lớn đối với màng hình máy tính (Màng hình máy tính sẽ không render đầy đủ bức ảnh trên màng hình sẽ gây ra khó khăn khi sử dụng) Vì vậy ta sẽ scale 1040 x 780 để vừa với màng hình của máy tính đồng thời bảo đảm khung hình cùng tỉ lệ, Không gây méo dạng cho bức ảnh. Đoạn chương trình cũng dự phòng khả năng không tìm được ảnh nguồn hoặc là không thể load được ảnh thì chương trình sẽ load bức ảnh báo lỗi không tìm thấy bứa ảnh nguồn, đồng thời thoát chương trình. Điều này đảm bảo rằng bộ dữ liệu sẽ chính xác tránh làm sai dataset vì lý do load nhầm ảnh nguồn.

```
arduino = serial.Serial(port='COM3', baudrate=9600, timeout=.1)
```

Câu lệnh trên dùng để khởi tạo giao tiếp Uart tại cổng COM3 của laptop. Với baudrate là 9600 bps.

```
positionMouse = pygame.mouse.get_pos()
state = []
x = positionMouse[0] / 1040
y = positionMouse[1] / 780
state.append(x)
state.append(y)
state.append((phi1 - 90) / 90)
state.append((phi2 - 90) / 90)
mainData.append(state)
print('dataSet have been saved: {}', state)
```

Đoạn chương trình trên thực hiện chức năng lấy vị trí của chuột máy tính bởi hàm: pygame.mouse.get_pos(). Khi ta kết hợp với hàm if để bắt lấy sự kiện click chuột thì ta sẽ lấy được vị trí mà người dùng đã click chuột.

Tọa độ của mục tiêu trên bức ảnh sẽ được ánh xạ về doạn [0:1] bằng hàm ánh xạ: positionX / Max wight.

Tương tự như vậy thì tọa độ Y của mục tiêu của mục tiêu cũng được ảnh xạ [0:1].

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_w:
        phi2 = phi2 + 1
        sendArduino()
    if event.key == pygame.K_s:
        phi2 = phi2 - 1
        sendArduino()
    if event.key == pygame.K_a:
        phi1 = phi1 + 1
        sendArduino()
    if event.key == pygame.K_d:
        phi1 = phi1 - 1
        sendArduino()
```

Đoạn chương trình trên bắt và xử lý các sự kiện nhấn phím để khiển khiển vị trí chiếu của lazer. Có các phím sự kiện được bắt và xử lý bao gồm:

+ Phím W: Tăng góc phi 2.

+ Phím S: Giảm góc phi 2.

+ Phím A: Tăng góc phi 1.

+ Phím D: Giảm góc phi 1.

Sau khi thực hiện thay đổi góc của phi1 hoặc phi2 thì chương trình thực hiện gửi góc quay của phi1, phi2 đến Arduino.

```
def sendArduino():
    dataSend = phi1 * 1000 + phi2
    dataSend = str(dataSend)
    print(f'Current phi1: {phi1}')
    print(f'Current phi2: {phi2}')
    arduino.write(bytes(dataSend, 'utf-8'))
    time.sleep(0.1)
    data = arduino.readline()
    data = data.decode('utf-8')
    print(f'Phi send back from arduino: {data}')
```

Đoạn chương trình trên là chương trình con dùng để gửi dữ liệu góc quay đến arduino. Vì để dễ dàng giao tiếp, dễ dàng truyền nhận thì dữ liệu về góc phi1 và góc phi2 được mã hóa thành một chuỗi trước khi được gửi đi. Dữ liệu được mã hóa thành chuỗi ký tự với 3 ký tự đầu tiên chứa thông tin của phi1 và 3 ký tự cuối chứa thông tin của góc phi2. Điều này được mã hóa bằng một hàm đơn giản:

B1: Chuyển 2 góc thành một số duy nhất, dataSend = phi1 * 1000 + phi2

B2: Chuyển dataSend thành chuỗi ký tự, dataSend = str(dataSend)

B3: Gửi chuỗi ký tự này đến arduino, được mã hóa thành từng byte theo chuẩn tiêu chuẩn utf-8.

B4: Cho chương trình tạm ngừng 0.1s mục đích việc này giúp có thời gian giản cách giữ các lần gửi hoặc gửi và nhận dữ liệu giúp giảm lỗi dữ liệu.

B5: Nhận gói tin xác nhận được gửi lại từ arduino và in lên màng hình.

```
# to end the event/loop
if event.type == pygame.QUIT:
    mainData = np.array(mainData)
    np.save('dataSet', mainData)
    print("Your data set that you colect have been save with file name is dataSet.npy")
    # it will deactivate the pygame library
    pygame.quit()
    quit()
```

Đoạn chương trình xử lý khi người dùng thoát chương trình.

B1: Chuyển dữ liệu đã lưu được đang ở kiểu list thành kiểu numpy array để dễ dàng lưu trữ và sử dụng lại.

B2: lưu dữ liệu vào file dataSet.npy

B3: Thoát chương trình.

3.2 Thiết kế và thực hiện chương trình xây dựng và huấn luyện model.

Nhiệm vụ của nhiệm vụ này là thực hiện một chương trình python nhằm xây dựng và huấn luyện model từ nguồn dữ liệu đã được góp nhặt phía trên. Dữ liệu sau khi được thu thập được biểu diễn thành các đồ thị như sau:

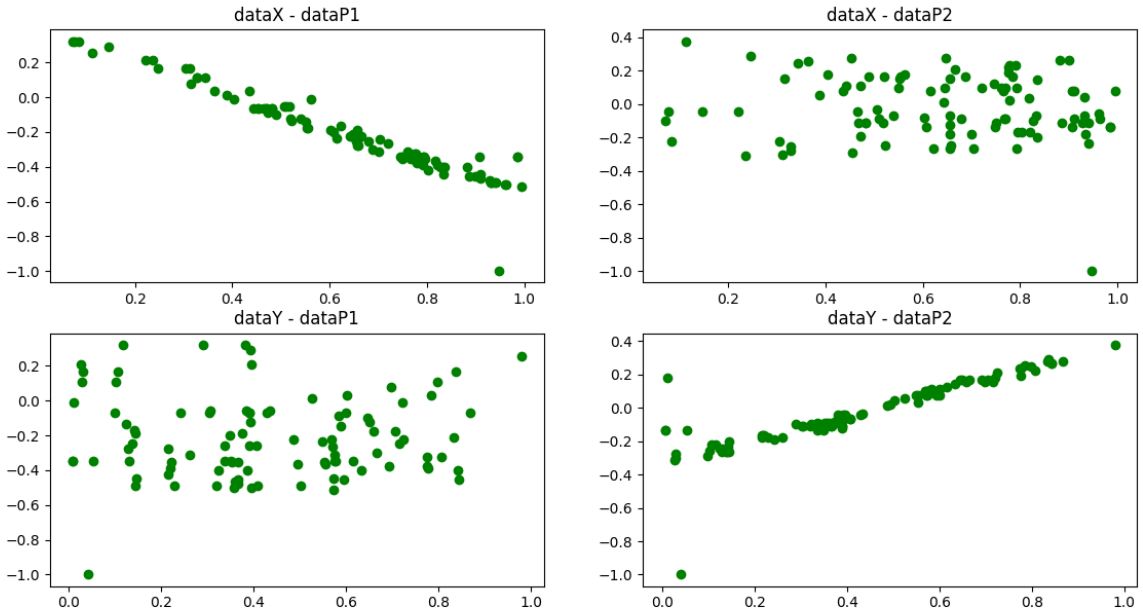

Figure 16: Đồ thị tương qua của bộ dữ liệu đã thu thập được

Ta dễ dàng nhận thấy được rằng, dữ liệu có sự tương quan rất chặt chẻ giữa DataX và phi1 Hoặc giữa DataY và Phi2. Dữ liệu tương đối đơn giản không quá phức tạp. Như vậy ta có thể xây dựng một mô hình có cấu trúc như sau.

```
def get_nn(state_size, output_size):
    model  = Sequential()
    model.add (Dense(8, activation='relu', input_dim=state_size))
    model.add (Dense(8, activation='relu'))
    model.add (Dense(output_size))
    model.compile( loss="mse", optimizer=Adam(learning_rate=0.01))
    return model
```

Mô hình ta xây dựng bao gồm 4 lớp:

+ Lớp 1: Input layer bao gồm 2 node (X, Y)

+ Lớp 2: Hidden layer 1 bao gồm 8 node.

+ Lớp 3: Hidden layer 2 bao gồm 8 node.

+ Lớp 4: Output layer bao gồm 2 node (Phi1, Phi2)

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 8)	24
dense_1 (Dense)	(None, 8)	72
dense_2 (Dense)	(None, 2)	18

Total params: 114 (456.00 B)
Trainable params: 114 (456.00 B)
Non-trainable params: 0 (0.00 B)

Figure 17: Tóm tắc mô hình đã được xây dựng

Như vậy ta có tổng số thông số (trọng số) từ layer input đến hidden layer 1 là 24 trọng số. Từ hidden layer 1 đến hidden layer 2 có 72 trọng số. Từ hidden layer 2 đến output layer là 18 trọng số vì vậy tổng trọng số của cả model là 114 trọng số bao gồm cả trọng số bias.

Ở hidden layer 1, ta dùng activation function là Relu,

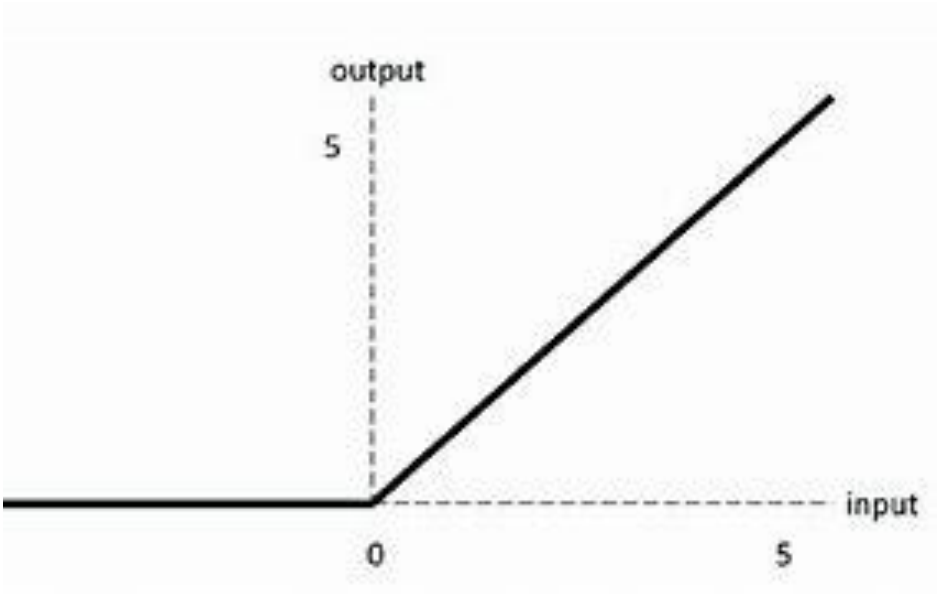

Figure 18: Minh họa hàm activation ReLU

Activation ReLU đơn giản, có tốc độ tính toán rất nhanh. Nhưng vẫn đảm bảo được nhiệm vụ khử tuyến tính của model.

Hàm activation ReLU cũng được sử dụng tiếp tục đối với hidden layer 2.

Ở Output layer, trong đoạn chương trình không định nghĩa hàm activation. Thì mặc định activation ở layer này sẽ là linear.

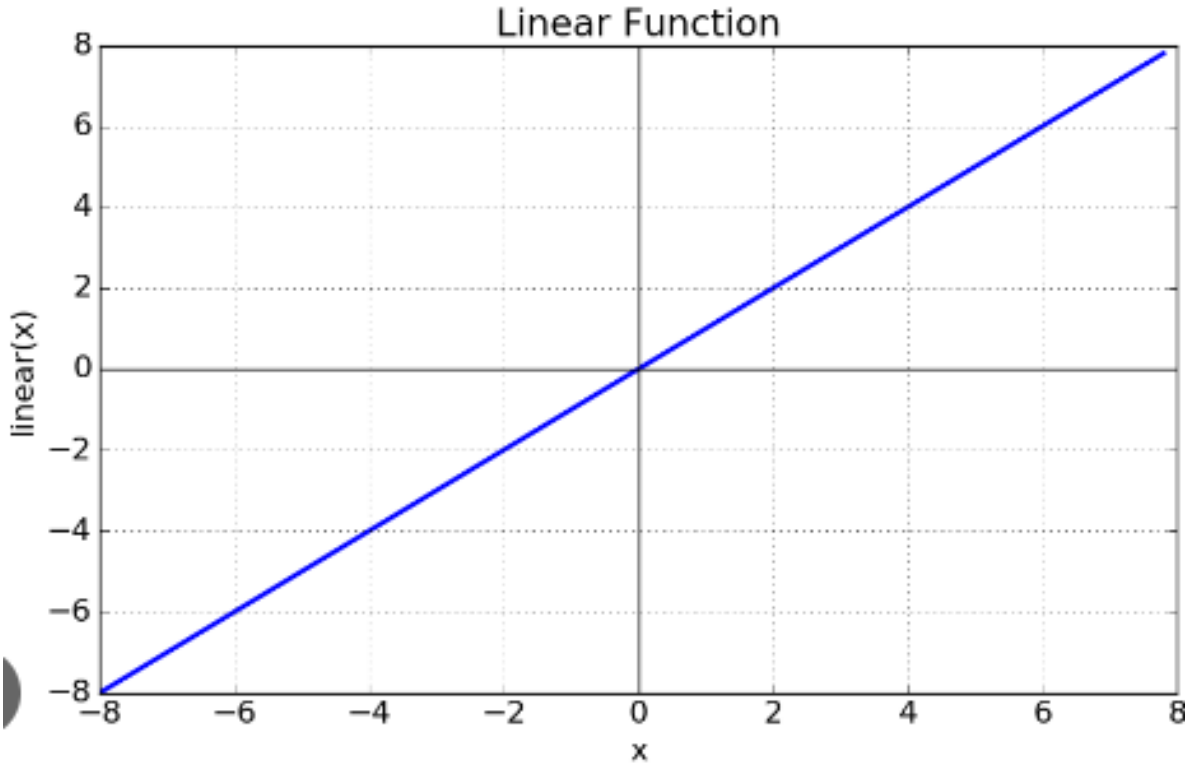

Figure 19: Minh họa hàm activation linear

Đối với hàm activation linear thì sẽ không có chức năng khử tuyến tính cho model, tuy nhiên điều này không quan trọng nữa vì đây là output layer. Quan trọng hơn output của model có giá trị trải dài từ [-1; 1] vì vậy hàm activation linear sẽ phù hợp hơn so với các hàm activation khác.

```
inModel = DataSet[:, 0:2]
outModel = DataSet[:, 2:4]
myModel = get_nn(2,2)
myModel.summary()
```

Đây là đoạn chương trình dùng để chuẩn bị dữ liệu và thành lập mô hình.

Dữ liệu ban đầu là mảng hai chiều (Nx4). Ta chia dữ liệu ban đầu thanh input cho model và output của model.

+ Input của model: Là một mãng 2 chiều (Nx2) là hai cột đầu của bộ dữ liệu chung. Ta lấy dữ liệu bằng cách lấy tấc cả các hành và 2 cột đầu tiên của bộ dữ liệu chung. DataSet[: , 0:2] trong đó DataSet là bộ dữ liệu chung, dấu ":" đầu tiên giúp lấy toán bộ các

hàng của bộ dữ liệu chung, “0:2” nghĩa là lấy từ cột 0 đến cột 1 theo nguyên tắc của python [start: stop: step].

+ Output của model: Là một mãng 2 chiều (Nx2) là hai cột cuối của bộ dữ liệu chung.

Bước cuối cùng của chương trình xây dựng và huấn luyện model là bước huấn luyện và lưu trọng số của model cho lần dùng tiếp theo hoặc lần học tiếp theo.

```
history = myModel.fit(inModel, outModel, epochs=200, batch_size = 20, validation_split = 0.2, verbose=1)
print('success')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Ta thực hiện tranning model bao gồm các thông số như sau:

+ InModel: Mảng chứa dữ liệu đầu vào

+ outModel: Mảng chứa đầu ra của model.

+ epochs: Số lần học.

+ batch_size: Số mẫu dữ liệu đầu vào mỗi lần học.

+ validation_split = 0.2 (0.8 dữ liệu dùng cho tranning và 0.2 dữ liệu dùng cho validation, dữ liệu này sẽ không được dùng để traning)

Kết quả huấn luyện model:

Ta xem xét loss fuction như sau:

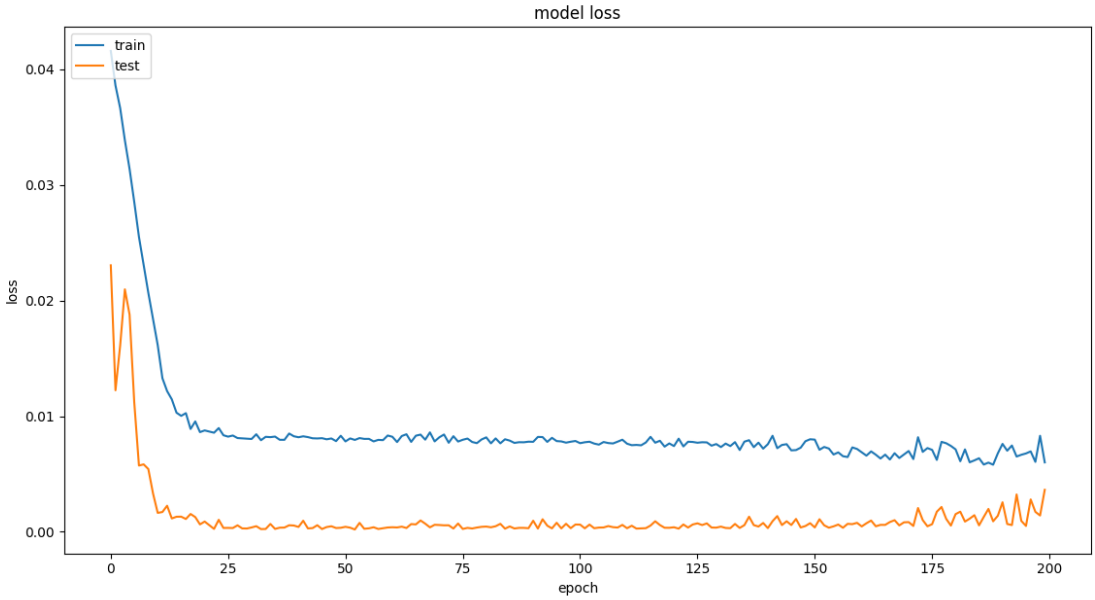

Figure 20: Đồ thị hàm cost thay đổi sau những lần học

Ta nhận thầy rằng sau khoảng 15 lần học thì mô hình bắt đầu học tụ tốt.

Bước cuối cùng là lưu các trọng số của mô hình đã được huấn luyện:

```
myModel.save_weights('myweight.weights.h5')
```

Trọng số của mô hình sẽ được lưu vào file: myweight.weights.h5

3.3 Thiết kế và thực hiện chương trình chính.

Nhiệm vụ chính của chương trình chính là xây dựng model để thực hiện dự đoán góc quay của servo để hướng lazer đến mục tiêu được xác định trên ảnh.

Ta có sơ đồ khối thực hiện hệ thống như sau:

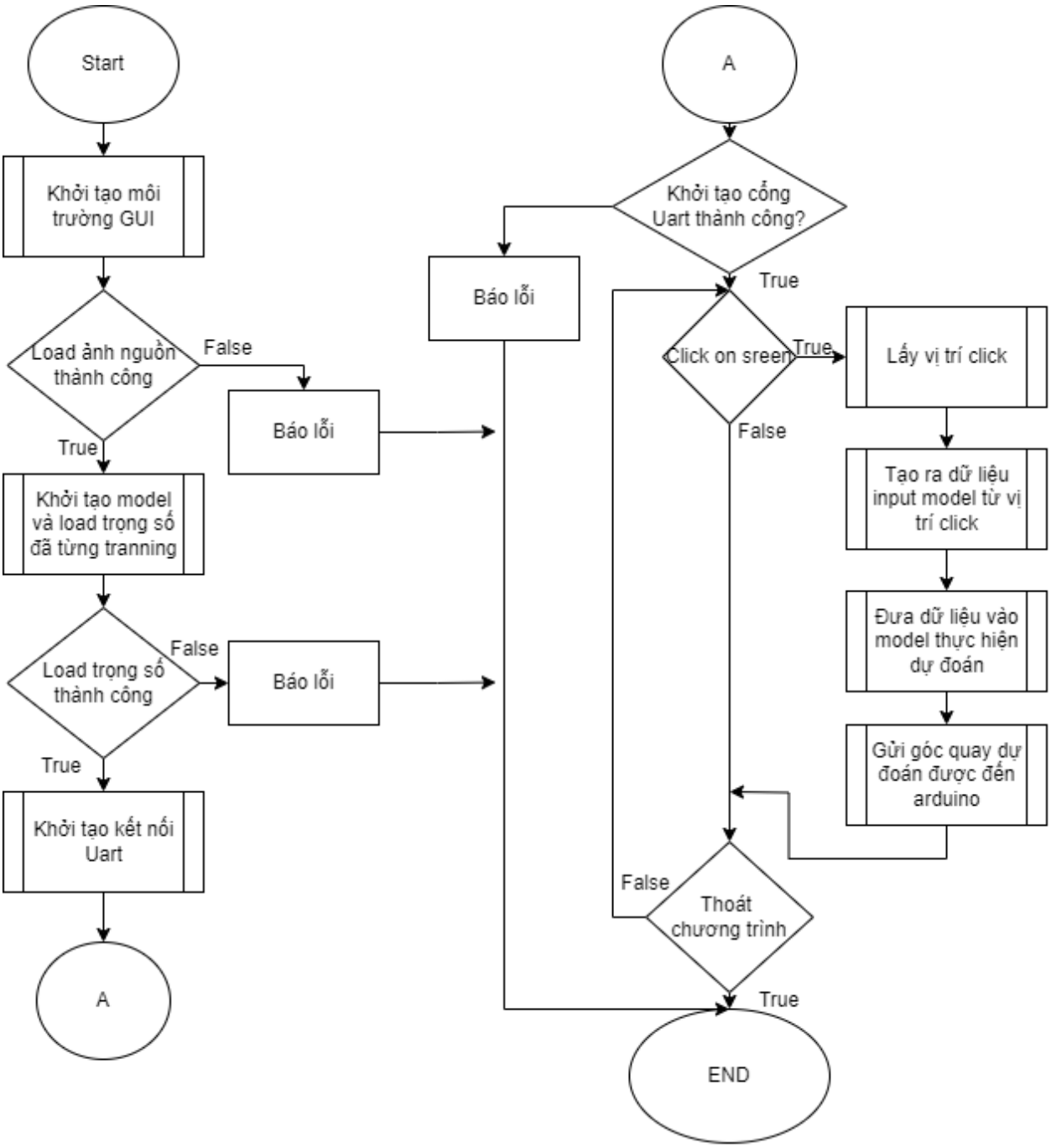

Figure 21: Flowchart chương trình chính

Khối đầu tiên khởi tạo GUI cho hệ thống, chức năng chính là render bức ảnh nguồn, cũng như tạo ra mô trường giúp bắt và xử lý các sự kiện xảy ra.

```
pygame.init()
window = pygame.display.set_mode((1040, 780))
pygame.display.set_caption("Using model")
clock = pygame.time.Clock()
try:
    image = pygame.image.load('./picture/img.jpg')
    image = pygame.transform.scale(image, (1040, 780))
except:
    print('can find img for this position, The program will be close,
    running = False
    image = pygame.image.load('./picture/imgNotFound.jpg')
    image = pygame.transform.scale(image, (1040, 780))
```

Cũng như lúc thu thập dữ liệu ta cũng scale ảnh nguồn về kích thước (1040x780). Đặt title cho của sổ là "Using model" để phân biệt với chương trình thu thập dữ liệu. Thực hiện kiểm tra nên không load được ảnh nguồn thì thông báo cho user biết để kiểm tra lại ảnh nguồn. Đồng thời gán biến running = false. Ngừng và thoát chương trình nhằm kiểm tra lại ảnh nguồn.

```
model  = Sequential()
model.add (Dense(8, activation='relu', input_dim=2))
model.add (Dense(8, activation='relu'))
model.add (Dense(2))
try:
    model.load_weights('myweight.weights.h5')
except:
    print('can not load weight of model which has been tranned,
    running = False
```

Khởi tạo model. Modle này phải có cấu trúc giống hoàn toàn với model lúc tranning. Thực hiện load lại trọng số đã được tranning trước đó. Nếu không tồn tại file chứa trọng số củ hoặc load không thành công thì báo lỗi đến user đông thời gán biến running = false. Vì khi không load được trọng số đã tranning thì lúc này trọng số của model được random, điều này làm cho model không dự đoán được chính xác góc quay của servo.

```
try:
    arduino = serial.Serial(port='COM3', baudrate=9600, timeout=.1)
except:
    print('can not open port to connect with arduino, Please check your connection and try again')
    running = False
```

Khởi tạo cổng giao tiếp Uart. Cổng giao tiếp Uart được khởi tạo tại cổng COM3 của máy tính, với tốc boadrate là 9600 bps. Với tốc độ này tuy không quá cao nhưng đáp ứng được tốc độ mà hệ thống yêu cầu, đồng thời giảm được lỗi truyền nhận dữ liệu. Đồng thời trên đoạn chương trình trên khi khởi tạo không thành công cổng kết nối Uart vì lý do công

bận hoặc cổng bị hỏng, chương trình sẽ thông báo đến người dùng, đồng thời gán biến running = false. Nhằm ngừng và thoát chương trình. Vì nếu không có cổng kết nối Uart thì không thể tạo kênh giao tiếp giữa máy tính và arduino, vì vậy model cũng không thể điều khiển, hướng lazer đến mục tiêu cho dù đã dự đoán chính xác góc quay.

```
if event.type == pygame.MOUSEBUTTONUP:
    print("[On processing]")
    positionMouse = pygame.mouse.get_pos()
    state = []
    x = positionMouse[0] / 1040
    y = positionMouse[1] / 780
    state.append(x)
    state.append(y)
    state = np.array(state)
    state.resize((1,2))
    outModel = model.predict(state, verbose=1)
    phi1 = 90*outModel[0][0] + 90
    phi2 = 90*outModel[0][1] + 90
    phi1 = int(phi1)
    phi2 = int(phi2)
    print('Model predict angle of servo Phi1: {}, Phi2: {}'.format(phi1, phi2))
    sendArduino()
```

Đoạn chương trình trên bắt và xử lý sự kiện click chuột trên màng hình. Đầu tiên chương trình sẽ thông báo cho người dùng [On process] Nhằm tránh người dùng click liên tục vào màng hình gây ra lỗi hoặc gây quá tải hệ thống. Tiếp theo chương trình thực hiện lấy vị trí click chuột và tạo ra dữ liệu đầu vào cho model tương tự như chương trình thu thập dữ liệu – đã tìm hiểu phía trên. Tiếp theo từ dữ liệu input model ta sẽ cho model thực hiện dựa đoán và trả về góc quay vào biến outModel. Từ dữ liệu dự đoán này chương trình thực hiện mã hóa và gửi đến Arduino. Đồng thời hiển thị lên màng hình giá trị dự đoán của model để user dễ nắm bắt và chủ động hơn.

```
def sendArduino():
    dataSend = phi1 * 1000 + phi2
    dataSend = str(dataSend)
    print(f'Current phi1: {phi1}')
    print(f'Current phi2: {phi2}')
    arduino.write(bytes(dataSend, 'utf-8'))
    time.sleep(0.1)
    data = arduino.readline()
    data = data.decode('utf-8')
    print(f'Phi send back from arduino: {data}')
```

Chương trình con thực hiện mã hóa và giao tiếp với Arduino tương tự như chương trình thu thập dữ liệu đã được hiểu trước đó.

3.4 Thiết kế và thực hiện chương trình điều khiển trên arduino.

Nhiệm vụ chính của phần này là nhận dữ liệu góc quay được model trên máy tính dự đoán, gửi dữ liệu xác nhận trở lại máy tính, Thực hiện điều khiển động cơ servo từ góc được dựa đoán từ máy tính gửi đến.

Sơ đồ schematic của hệ thống:

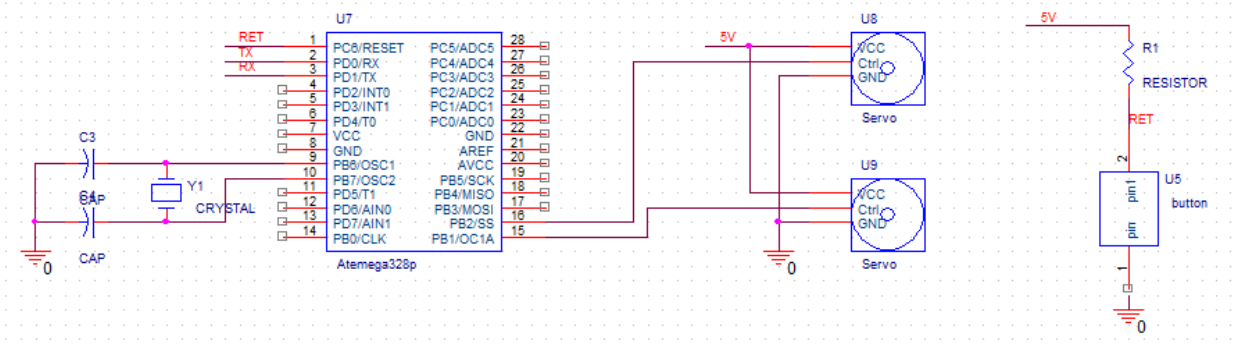

Figure 22: Sơ đồ phần cứng của hệ thống

Hệ thống gồm các phần quang trọng như:

+ Node MCU: Vi điều khiển được lựa chọn là ATMEGA328P được đặt sẵn trên board Arduino Uno. Chức năng thực hiện điều khiển 2 động cơ servo, Thực hiện giao tiếp với máy tính.

+ Phần cấp dao động: Chức năng tạo xung clock cho MCU hoạt động. Phần này ta không cần phải thiết bởi nên phần này được xây dựng sẵn trên board arduino với tần số hoạt động 16MHz.

+ Phần tạo tín hiệu reset: Chức năng reset MCU khi hệ thống bị treo hoặc, khi khởi động lại hệ thống. Phần này ta cũng không cần phải thiết kế vì phần này cũng có sẵn trên board arduino. (Reset trên vi điều khiển ATMEGA328P tích cực mức thấp).

+ Phần giao tiếp Uart: Chức năng tạo nên kênh giao tiếp Uart giữ máy tính và arduino. Phần này ta cũng không cần thực hiện vì trên board Arduino đã có sẵn chip chuyển đổi từ giao thức Uart sang USB và ngược lại.

+ Động cơ servo: Phần này có chức năng vận hành điều khiển lazer. Động cơ servo được lựa chọn sử dụng là SG9g. Được điều khiển góc độ bằng tín hiệu PWM.

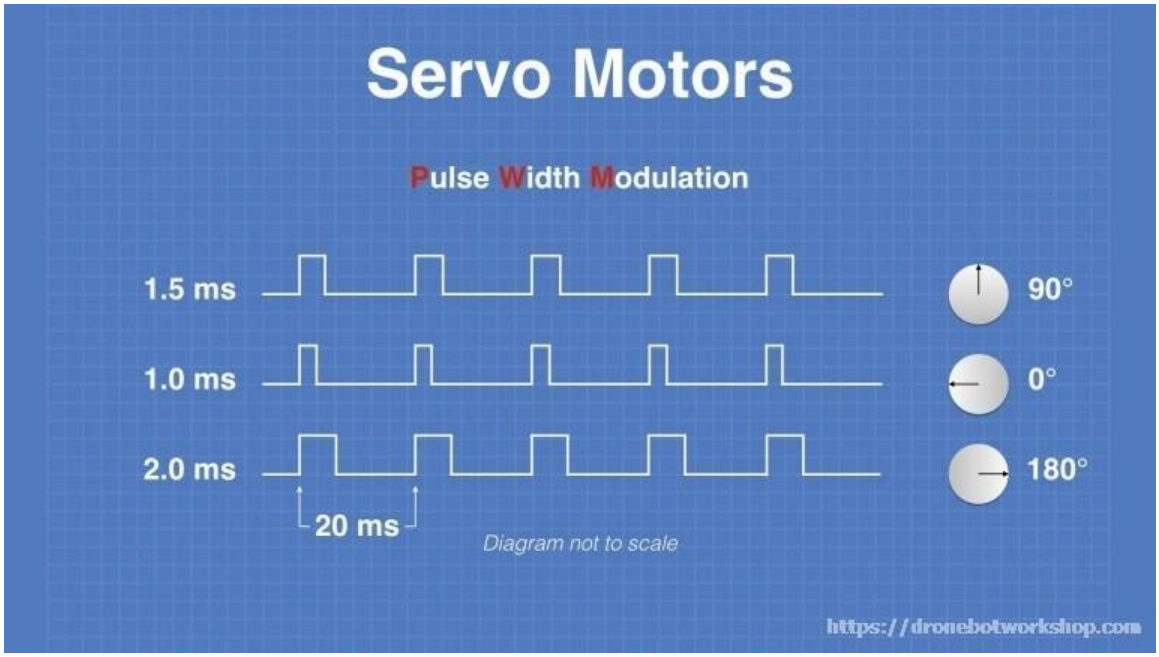

Figure 23: Minh họa giao thức điều khiển động cơ servo

Sơ đồ khối hoạt động của hệ thống:

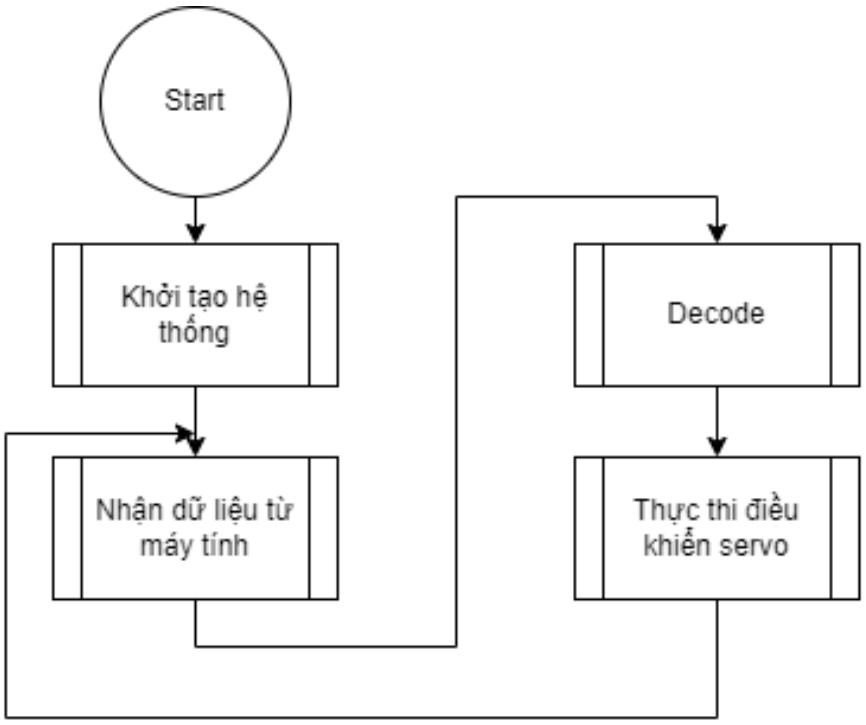

Figure 24: Sơ đồ khối chương trình trên arduino

Khởi tạo hệ thống: có chức năng định nghĩa các chân inout, các biến global.

```
Servo servo1;
Servo servo2;
void setup(){
  servo1.attach(9);
  servo2.attach(10);
  servo1.write(90);
  servo2.write(90);
  Serial.begin(9600);
  Serial.setTimeout(100);
}
int phi1 = 90;
int phi2 = 90;
```

Servo servo1 : Khởi tạo oject điều khiển động cơ servo 1. Đồng thời gán chân điều khiển cho động cơ servo 1 là chân số 9 (Hay tương ứng với Pin PB1 của Atemega328p)
Servo servo2 : Khởi tạo oject điều khiển động cơ servo 2. Đồng thời gán chân điều khiển cho động cơ servo 2 là chân số 10 (Hay tương ứng với Pin PB2 của Atemega328p)
Khởi tạo giao tiếp Uart: Với boadrate tương tự với kênh Uart trên máy tính là 9600bps.
Khởi tạo góc ban đầu cho servo là 90 độ, cho cả 2 servo.

```
if (Serial.available() > 0){
  command = Serial.readString();
```

Nhận dữ liệu truyền đến từ máy tính quá kênh Uart dưới dạng một chuỗi ký tự.

```
long i = command.toInt();
phi1 = i / 1000;
phi2 = i % 1000;
String retu = String(phi1) + '-' + String(phi2) + '\n';
Serial.print(retu);
```

Tiến hành chuyển kiểu chuỗi thành số nguyên và giải mã thành 2 góc là phi1 và phi2. Sau khi giải mã xong thì tổng hợp lại và gửi xác nhận đến máy tính.

```
servo1.write(phi1);
servo2.write(phi2);
delay(10);
```

Thực hiện điều khiển servo từ hai góc phi1 và phi2 đã nhận được.

3.5 Thiết kế phần cứng, bệ đỡ lazer.

Bệ đỡ có chức năng giữa bệ lazer, Tạo không gian chứa board arduino, breakboard nếu có, nguồn nếu có. Đồng thời tạo ra vị trí tuyệt đối cố định giữa camera điện thoại và ví trí bệ đỡ lazer. Bởi vì ví trí tuyệt đối giữa camera điện thoại và bệ đỡ lazer bắt buộc không được thay đổi, bắc buộc phải giống nhau giữa lúc thu thập dữ liệu và lúc sử dụng model. Vì nếu thay đổi vị trí tuyệt đối này thì mô hình sẽ dự đoán sai góc quay dẫn đến lazer sẽ không hướng đúng vị đúng vị trí mục tiêu.

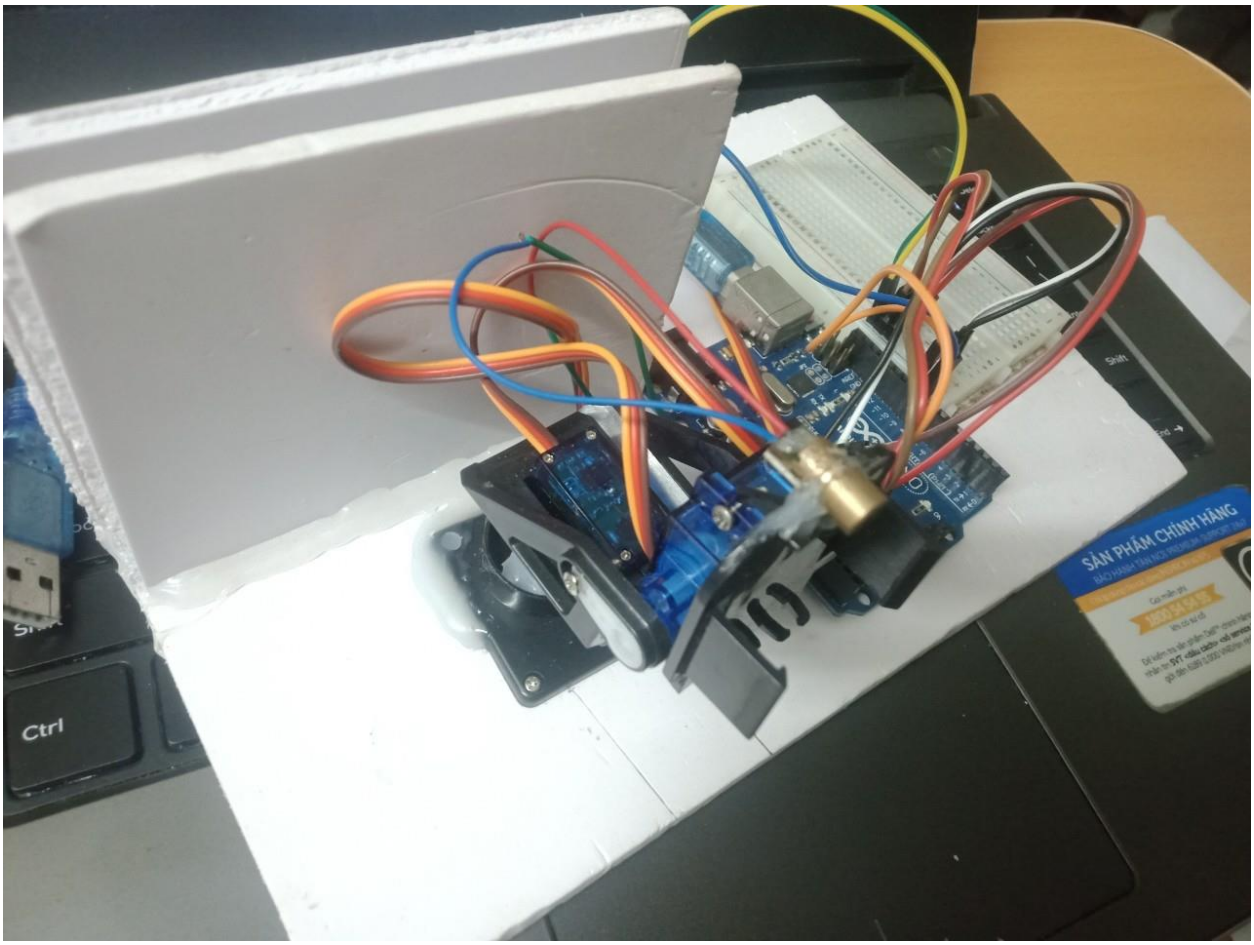

Figure 25: Hình ảnh hệ thống được cây dựng trên giấy bìa cứng.

Hai mặt phẳng vuông gốc dùng để cố định camera, dùng để cố định vị trí tuyệt đối giữa camera điện thoại và bệ đỡ camera.

4 KẾT QUẢ THU ĐƯỢC

Thiết kế thành công hệ thống có khả năng nhận vào tọa độ vật thể và hướng lazer đến mục tiêu được xác định trên ảnh.

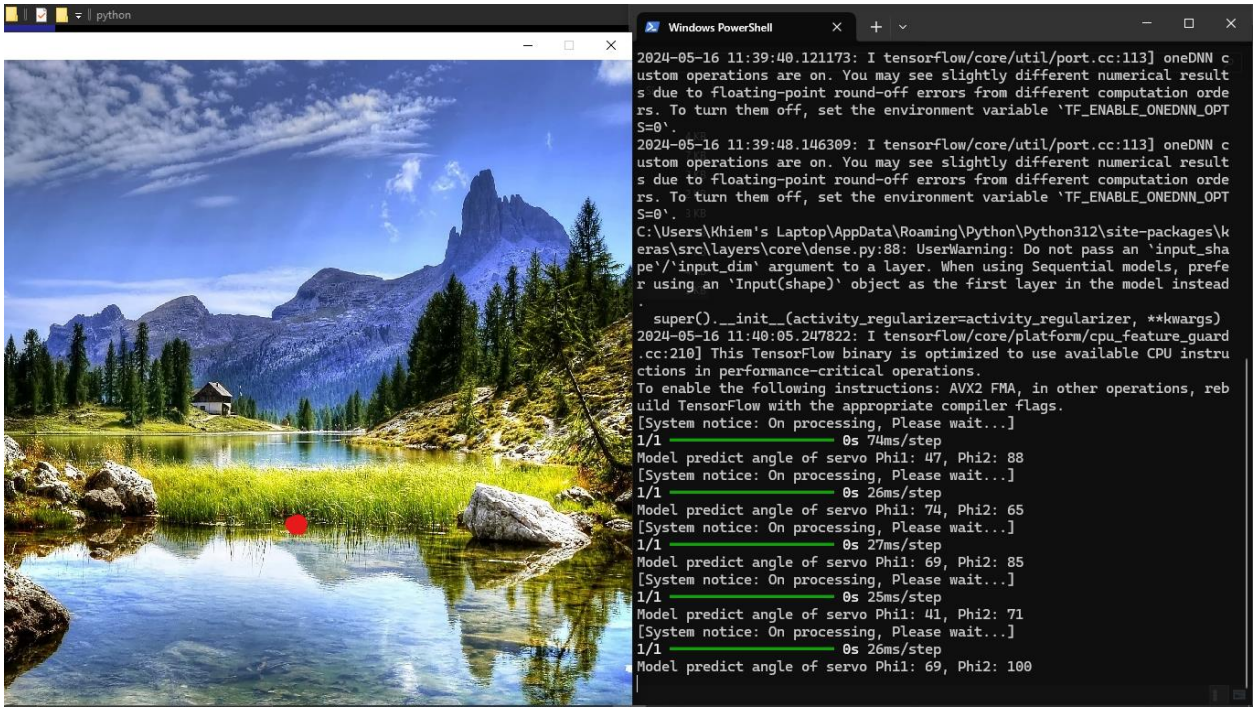

Figure 27: Hệ thống hoạt động trên chương trình test lỗi vẫn hành.

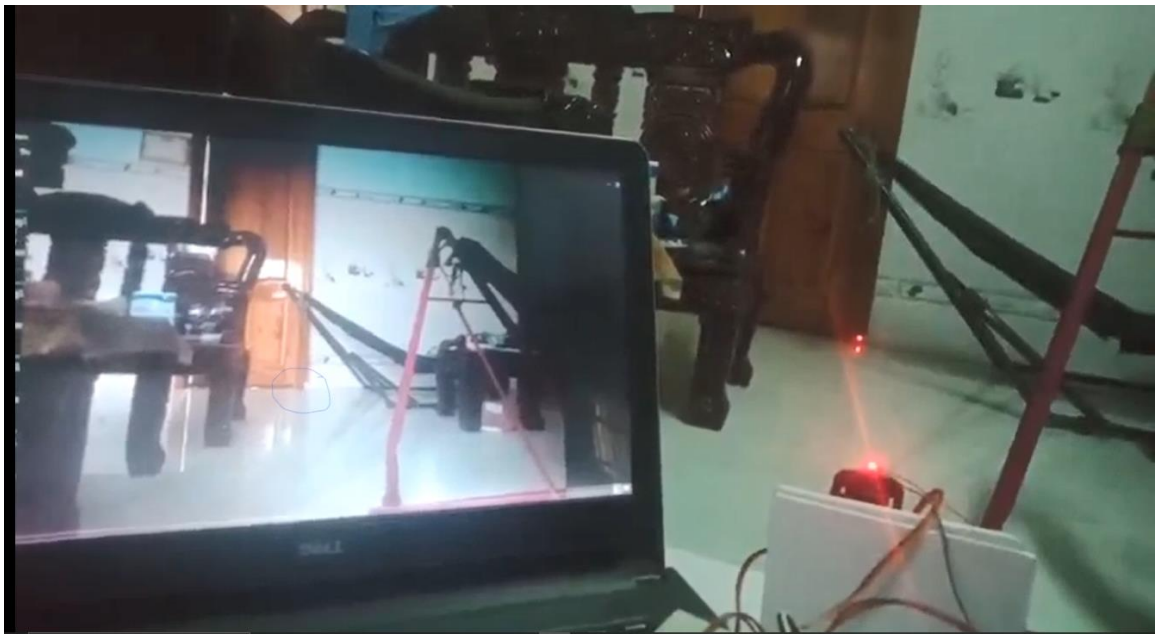

Figure 26: Hệ thống test vận hành thực tế.

Ví dụ đối với chương trình test trên thì khi nhấn chuột vào vị trí chấm đỏ thì model sẽ dự đoán góc quay phi1 bằng 69, góc quay phi2 bằng 100. Vì mô hình khá nhỏ vì thế tốc độ dự đoán của mô hình cũng khá cao tầm 26ms mỗi mẫu dữ liệu đầu vào.

Khi trí click chuột như trong màng hình thì lazer hướng đến mục tiêu tương tự ngoài thực tế với độ chính xác khá cao (chấp nhận được sai số).

5 Phần mở rộng của đồ án

5.1 Giới thiệu về FPGA DE0-Nano

Figure 28: Kit FPGA DE-0 Nano

Kit DE0-Nano có FPGA Altera Cyclone® IV EP4CE22F17C6N với tối đa 153 FPGA I/O pins. Trên Kit có 2 bộ 40-pin Headers (GPIOs) cung cấp 72 I/O pins, 5V power pins, hai 3.3V power pins và các chân đất.

Bộ nhớ của kit bao gồm:

- 32MB SDRAM
- 2Kb I2C EEPROM

Dao động nội có trên KIT kit: On-board 50MHz clock oscillator

Power Supply: USB Type mini-AB port (5V); DC 5V pin for each GPIO header (2 DC 5V pins); 2-pin external power header (3.6-5.7V)

5.2 Xây dựng mô hình neural network trên FPGA

Phép toán quan trọng nhất trong mô hình neural networt là phép toán tích chập (convolution). Tuy nhiên phép toàn này bao gồm nhiều phép nhân và phép cộng. Vì vậy với một bộ xữ lý trung tâm (CPU) thực hiện các phép toán một cách tuần tự thì việc thực hiện các phép tích chập sẽ mất khá nhiều thời gian, Đối với lĩnh vực cần nguồn dữ liệu lớn như máy học thì việc tính toán tuần tự như CPU sẽ là một trở ngại lớn. Vì vậy chúng ta cần một bộ xữ lý tính toán có thể tính toán song song nhiều phép tính cùng lúc và có tốc độ

tính toán cao để đáp ứng nhu cầu của các ứng dụng máy học. FPGA có thể đáp ứng được nhu cầu về tính toán song song và tốc độ xữ lý cao.

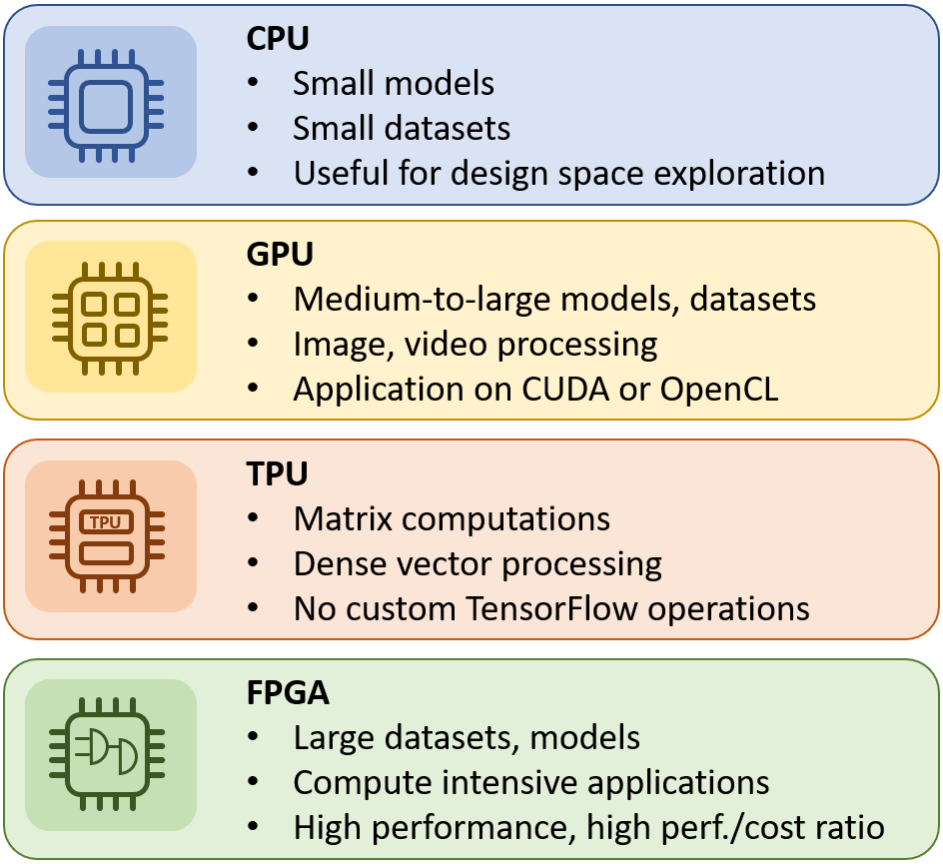

Figure 29: Minh họa ứng dụng Neural network trên CPU, GPU, TPU, FPGA

Các Thành phần quang trọng của neural network được mang lên FPGA.

- Một node trong neural network.

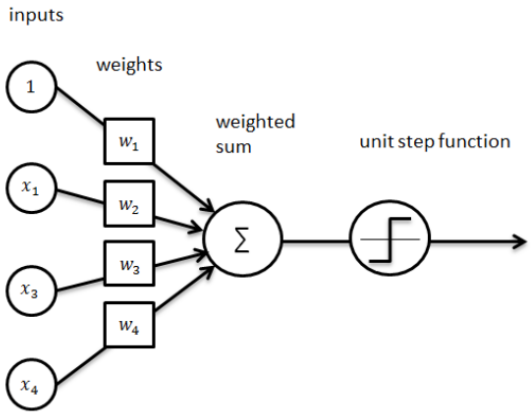

Figure 30: Node của mạng neural network

Thiết kết node trên FPGA:

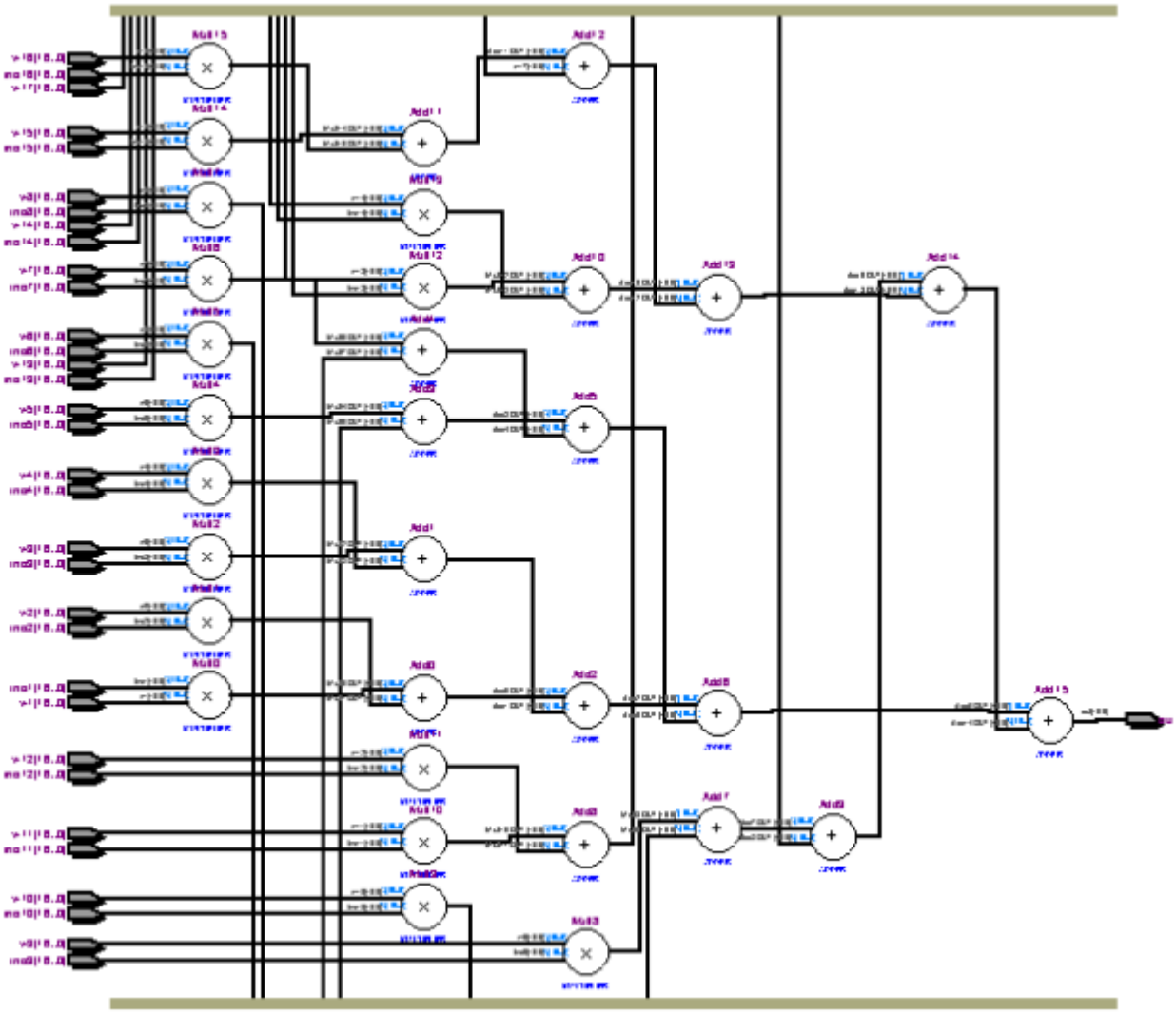

Figure 31: Thiết kế node của neural network trên FPGA

```
module
neural17b(ino1,ino2,ino3,ino4,ino5,ino6,ino7,ino8,ino9,ino10,ino11,ino12,ino13,ino14
,ino15,ino16,
w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17, out);
input [16:0] ino1,ino2,ino3,ino4,ino5,ino6,ino7,ino8,ino9, ino10,ino11,ino12,ino13,
ino14, ino15,ino16;
input [16:0] w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17;
```

```
output [16:0] out;
assign out = (((ino1*w1 + w2*ino2) + (w3*ino3+ w4*ino4)) + ((w5*ino5+ w6*ino6) +
(w7*ino7 + w8*ino8))) +
(((w9*ino9 + w10*ino10) + (w11*ino11 + w12*ino12)) + ((w13*ino13 + w14*ino14)
+ (w15*ino15 + w16*ino16 + w17)));
endmodule
```

Phía trên là đoạn code verilog mô ta hoạt động của một node neural network có 17 ngõ vào, 17 trọng số (weight) và 1 hệ số bias w17. Tấc các ngõ vào và trọng số đều là số nguyên 16 bit có dấu. Tuy nhiên mô hình neural network thực hiện trên số thực nên số 16 bit có dấu sẽ được tự hiểu là chia cho 256 (dịch sang phải 8 bit) như vậy độ chính xác của biểu diễn số trên là 1/256 = 0.004

- Một thành phần quang trọng tiếp theo là hàm activation function: Hàm được xữ dụng nhiều nhất là hàm ReLU. Hàm được mô tả bằng max{0,x}. Hiểu đơn giản nếu x bé hơn hoặc bằng không thì giá trị trả về là 0 và nếu x lớn hơn không thì trả về là x.

```
module relu(in, out);
        input [16:0] in;
        output [16:0] out;
        assign out[0]=in[0]&(~in[16]);
        assign out[1]=in[1]&(~in[16]);
        assign out[2]=in[2]&(~in[16]);
        assign out[3]=in[3]&(~in[16]);
        assign out[4]=in[4]&(~in[16]);
        assign out[5]=in[5]&(~in[16]);
        assign out[6]=in[6]&(~in[16]);
        assign out[7]=in[7]&(~in[16]);
        assign out[8]=in[8]&(~in[16]);
        assign out[9]=in[9]&(~in[16]);
        assign out[10]=in[10]&(~in[16]);
```

```
        assign out[11]=in[11]&(~in[16]);
        assign out[12]=in[12]&(~in[16]);
        assign out[13]=in[13]&(~in[16]);
        assign out[14]=in[14]&(~in[16]);
        assign out[15]=in[15]&(~in[16]);
        assign out[16]=in[16]&(~in[16]);
endmodule
```

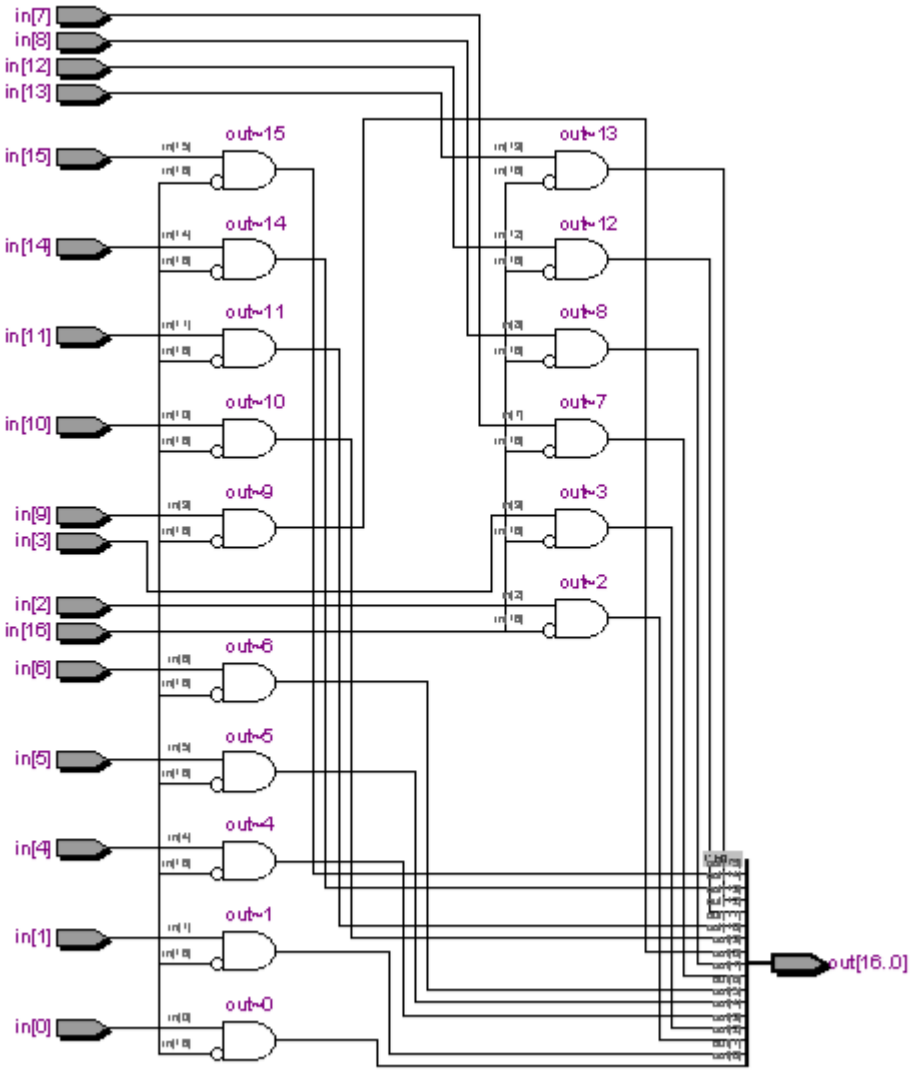

Figure 32: Cấu trúc hàm ReLU logic gate level

5.3 Hệ thống số của mô hình neural network trên FPGA.

Kết quả quả mỗi một node trong mô hình neural network là một số thực, Có hai loại là số chấm động và số chấm tĩnh.

Đặc điểm chính của số chấm tĩnh:

+ Vị trí dấu phẩy cố định: Vị trí của dấu phẩy được xác định trước và không thay đổi trong quá trình biểu diễn số.

+ Độ chính xác hữu hạn: Số chấm tĩnh chỉ có thể biểu diễn một số lượng hữu hạn các giá trị thực.

+ Sai số làm tròn: Do độ chính xác hữu hạn, có thể xảy ra sai số làm tròn trong các phép toán số học trên số chấm tĩnh.

+ Dễ dàng triển khai: Số chấm tĩnh dễ dàng được triển khai trên phần cứng máy tính.

Số chấm tĩnh được biểu diễn theo định dạng sau:

| Sign bit | Exponent | Significand |

+Bit dấu (sign bit): Biểu thị dấu của số (0 là dương, 1 là âm).

+Mũ (exponent): Biểu thị vị trí của dấu phẩy thập phân.

+Phần thập phân (significand): Biểu thị giá trị của số.

Đặc điểm chính của số chấm động:

+ Có độ chính xác cao: Số chấm động có thể biểu diễn các số với độ chính xác cao hơn so với số chấm tĩnh.

+ Có phạm vi biểu diễn rộng: Số chấm động có thể biểu diễn các số có phạm vi giá trị rộng hơn so với số chấm tĩnh.

+ Có thể biểu diễn các số rất lớn và rất nhỏ: Số chấm động có thể biểu diễn các số rất lớn (ví dụ: 1.2345e+100) và rất nhỏ (ví dụ: 1.2345e-100).

+ Có thể biểu diễn các số vô tỉ: Số chấm động có thể biểu diễn các số vô tỉ (ví dụ: π).

Cấu trúc của số chấm động:

Số chấm động được biểu diễn theo định dạng sau:

| Sign bit | Exponent | Significand |

+ Bit dấu (sign bit): Biểu thị dấu của số (0 là dương, 1 là âm).

+ Mũ (exponent): Biểu thị vị trí của dấu phẩy thập phân.

+ Phần thập phân (significand): Biểu thị giá trị của số.

Ưu điểm của số chấm động so với số chấm tĩnh:

1. Độ chính xác cao hơn:

Số chấm động có thể biểu diễn các số với độ chính xác cao hơn nhiều so với số chấm tĩnh.

Lý do là vì số chấm động sử dụng phần mũ (exponent) để biểu diễn vị trí của dấu phẩy thập phân, cho phép biểu diễn các số có phạm vi giá trị rộng hơn và độ chính xác cao hơn.

Với cùng độ dài bit, số chấm động có thể biểu diễn các số từ -10^308 đến 10^308, trong khi số chấm tĩnh chỉ có thể biểu diễn các số từ 0 đến 2^n - 1 (n là số bit).

Do đó, số chấm động có thể biểu diễn các số vô cùng lớn hoặc vô cùng nhỏ với độ chính xác cao, trong khi số chấm tĩnh không thể biểu diễn được các số này.

2. Phạm vi biểu diễn rộng hơn:

Như đã đề cập ở trên, số chấm động có thể biểu diễn các số có phạm vi giá trị rộng hơn so với số chấm tĩnh.

Điều này giúp cho số chấm động phù hợp hơn với các ứng dụng đòi hỏi phải xử lý các số rất lớn hoặc rất nhỏ.

3. Có thể biểu diễn các số rất lớn và rất nhỏ:

Số chấm động có thể biểu diễn các số rất lớn (ví dụ: 1.2345e+100) và rất nhỏ (ví dụ: 1.2345e-100).

Khả năng này giúp cho số chấm động phù hợp với các ứng dụng khoa học, kỹ thuật và tài chính, nơi thường xuyên phải xử lý các số có giá trị lớn hoặc nhỏ.

4. Dễ dàng thực hiện các phép toán phức tạp:

Các phép toán trên số chấm động thường được thực hiện dễ dàng hơn so với số chấm tĩnh.

số chấm tĩnh sẽ dễ thực hiện hơn so với số chấm động cho các phép toán cơ bản như:

+ Cộng: Phép cộng trên số chấm tĩnh chỉ cần cộng từng cặp bit tương ứng của hai số.

+ Trừ: Phép trừ trên số chấm tĩnh cũng chỉ cần trừ từng cặp bit tương ứng của hai số.

+ So sánh: Phép so sánh trên số chấm tĩnh chỉ cần so sánh các bit tương ứng của hai số.

Tuy nhiên, số chấm động sẽ dễ thực hiện hơn so với số chấm tĩnh cho một số phép toán khác như:

+ Nhân: Phép nhân trên số chấm động có thể được thực hiện nhanh hơn so với số chấm tĩnh bằng cách sử dụng các thuật toán nhân nhanh.

+ Chia: Phép chia trên số chấm động có thể được thực hiện nhanh hơn so với số chấm tĩnh bằng cách sử dụng các thuật toán chia nhanh.

+ Lũy thừa: Phép lũy thừa trên số chấm động có thể được thực hiện nhanh hơn so với số chấm tĩnh bằng cách sử dụng các thuật toán lũy thừa nhanh.

6 MỞ RỘNG DỰ ÁN TRONG TƯƠNG LAI

6.1 Phát triển phần camera gắng trực tiếp trên mô hình.

Lời ích: Thêm phần camera không dùng camera của điện thoại có thể realtime gửi hình ảnh đến vi xữ lý và ví xữ lý sẽ gửi đến máy tính. Việc này sẽ tránh được việc thay đổi vị trí của hệ thống sau khi đã chụp ảnh từ điện thoại. Vì nếu sau khi đã chụp ảnh nguồn mà thay đổi vị trí mà ảnh nguồn không thay đổi theo thì model sẽ dự đoán sai và không thể hướng lazer đúng mục tiêu được.

Sơ đồ khối hệ thống mới:

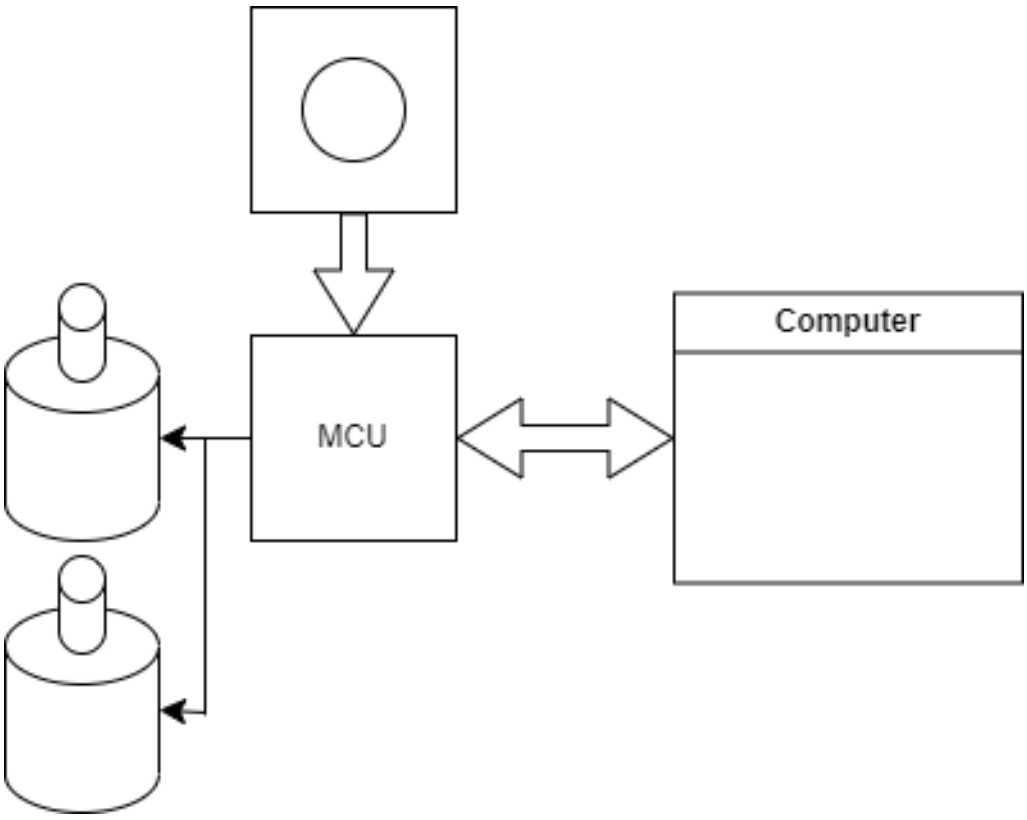

Figure 33: Sơ đồ khối của hệ thống khi thêm realtime camera

Ta có thể dùng một số giao tiếp protocol giao tiếp như sau:

MCU giao tiếp Computer: Uart, dữ liệu hình ảnh cũng sẽ được gửi cũng qua giao thức Uart. Gửi dữ liệu điều khiển cũng qua Uart. Điều này cũng sẽ giảm giảm fps của hệ thống vì Uart có tốc độ không cao. Tuy nhiên hệ thống không cần quá nhanh, băng thông dữ liệu cũng không cần quá nhiều.

6.2 Kết hợp với mô hình Yo-Lo.

Lợi ích: Khi kết hợp với mô hình Yo-lo nhằm xác định tự động tâm mục tiêu của vật thể. Nhằm tăng thêm tính ứng dụng cho hệ thống tự động xác định mục tiêu và hướng lazer đến mục tiêu.

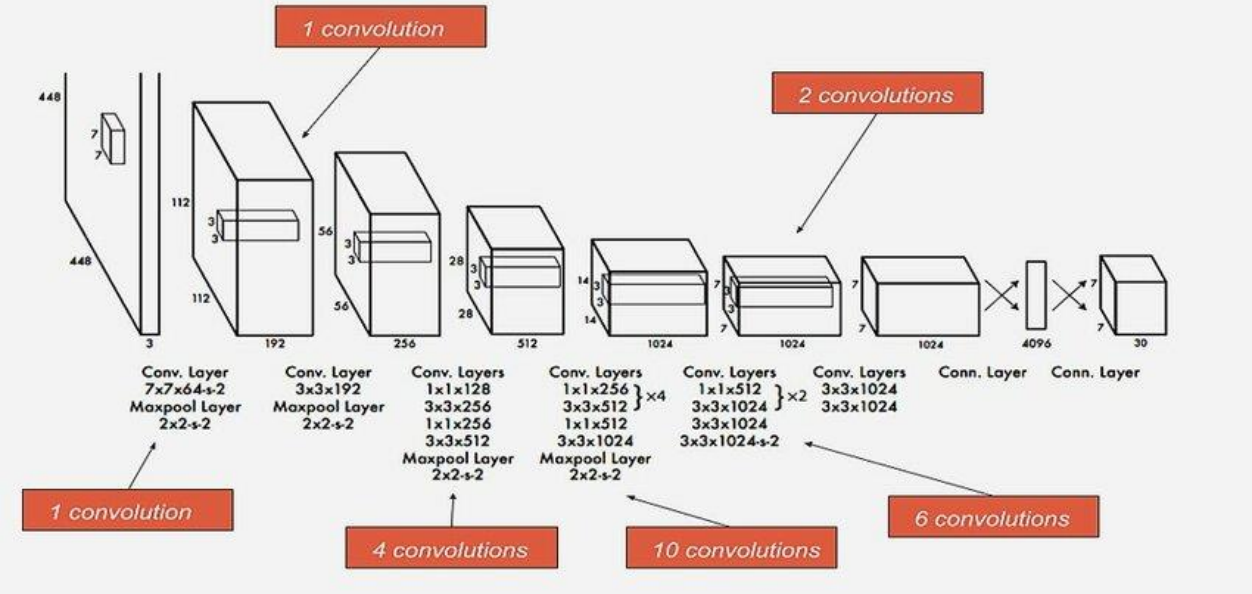

Figure 35: Cấu trúc mô hình Yo-lo v1

Figure 34: Minh họa đầu ra của mô hình Yo-lo

Phía trên là mô hình Yo-Lo v1. Output của mô hình Yo-lo có thể trích xuất được tọa độ mục tiêu của vật thể được huấn luyện sẵn từ model.

TÀI LIỆU THAM KHẢO

1. CODELREARN, *Các Yếu Tố Lập Trình Game Cơ Bản Với Pygame*, (27/10/2020), truy cập tại https://codelearn.io/sharing/lap-trinh-game-co-ban-voi-pygame
2. Machine Learning cơ bản, *Gradient Descent (phần 1/2)*, (12/01/2017), truy cập tại https://machinelearningcoban.com/2017/01/12/gradientdescent/
3. Machine Learning cơ bản, *Gradient Descent (phần 2/2)*, (16/01/2017), truy cập tại https://machinelearningcoban.com/2017/01/16/gradientdescent2/
4. Nguyễn Thanh Tuấn, *Mô hình neural network*, (09/03/2019), truy cập tại https://nttuan8.com/bai-3-neural-network/

PHỤ LỤC

Phụ lục 1: code thu thập dữ liệu

```
import serial
import pygame
import os
import time
import numpy as np
# Pygame for interface
pygame.init()
# setting frame/window/surface with some dimensions
window = pygame.display.set_mode((1040, 780))
# to set title of pygame window
pygame.display.set_caption("Get_dataSet")
clock = pygame.time.Clock()
# creating image object
try:
    image = pygame.image.load('./picture/img.jpg')
    image = pygame.transform.scale(image, (1040, 780))
except:
    print('can find img for this position')
    image = pygame.image.load('./picture/imgNotFound.jpg')
    image = pygame.transform.scale(image, (1040, 780))
# to display size of image
print("size of image is (width,height):", image.get_size())
phi1 = 0
phi2 = 0

def sendArduino():
    dataSend = phi1 * 1000 + phi2
    dataSend = str(dataSend)
    print(f'Current phi1: {phi1}')
    print(f'Current phi2: {phi2}')
    arduino.write(bytes(dataSend, 'utf-8'))
    time.sleep(0.1)
    data = arduino.readline()
    data = data.decode('utf-8')
    print(f'Phi send back from arduino: {data}')

#Main function ----------Start
arduino = serial.Serial(port='COM3', baudrate=9600, timeout=.1)
os.system('cls')
mainData = []
try:
```

```
    mainData = np.load('dataSet.npy')
    mainData = np.array(mainData).tolist()
    print("Load old date set success")
except:
    print("Cant find old data set. Creating new dataset")

# loop to run window continuously
while True:
    window.blit(image, (0, 0))
    # loop through the list of Event
    for event in pygame.event.get():
        # to end the event/loop
        if event.type == pygame.QUIT:
            mainData = np.array(mainData)
            np.save('dataSet', mainData)
            print("Your data set that you colect have been save with file name is
dataSet.npy")
            # it will deactivate the pygame library
            pygame.quit()
            quit()
        # to display when screen update
        pygame.display.flip()
        if event.type == pygame.MOUSEBUTTONUP:
            positionMouse = pygame.mouse.get_pos()
            state = []
            x = positionMouse[0] / 1040
            y = positionMouse[1] / 780
            state.append(x)
            state.append(y)
            state.append((phi1 - 90) / 90)
            state.append((phi2 - 90) / 90)
            mainData.append(state)
            print('dataSet have been saved: {}', state)
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_w:
                phi2 = phi2 + 1
                sendArduino()
            if event.key == pygame.K_s:
                phi2 = phi2 - 1
                sendArduino()
            if event.key == pygame.K_a:
                phi1 = phi1 + 1
                sendArduino()
```

```
            if event.key == pygame.K_d:
                phi1 = phi1 - 1
                sendArduino()
    # Xây dựng một class Agent
```

Phụ lục 2: Code tranning dữ liệu

```
    import serial
    import keyboard
    import os
    import time
    import random
    import numpy as np
    import tensorflow
    import matplotlib.pyplot as plt
    from collections import deque

    from keras.models import Sequential
    from keras.layers import Dense
    from keras.optimizers import Adam

    def get_nn(state_size, output_size):
        model  = Sequential()
        model.add (Dense(8, activation='relu', input_dim=state_size))
        model.add (Dense(8, activation='relu'))
        model.add (Dense(output_size))
        model.compile( loss="mse", optimizer=Adam(learning_rate=0.01))
        return model
    try:
        DataSet = np.load('dataSet.npy')
    except:
```

```
    print("Dont have data set, Please prepare dataset in program getData, try again !")
    DataSet = [[0,0,0,0],[1,1,1,1]]
    DataSet = np.array(DataSet)
inModel = DataSet[:, 0:2]
outModel = DataSet[:, 2:4]
myModel = get_nn(2,2)
myModel.summary()
try:
    myModel.load_weights('myweight.weights.h5')
except:
    print("can not find old weight, create new model with random weight. Or something wrong can not load old weight")
history = myModel.fit(inModel, outModel, epochs=200, batch_size = 20, validation_split = 0.2, verbose=1)
print('success')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
myModel.save_weights('myweight.weights.h5')
```

Phụ lục 3: Code chương trình chính

```
import pygame
import os
import time
import numpy as np
```

```
import tensorflow
from collections import deque
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

#global variable
phi1 = 0
phi2 = 0
running = True
#endn define global variable
# Pygame for interface
pygame.init()
window = pygame.display.set_mode((1040, 760))
pygame.display.set_caption("Using model")
clock = pygame.time.Clock()
try:
    image = pygame.image.load('./picture/lake2.jfif')
    image = pygame.transform.scale(image, (1040, 780))
except:
    print('can find img for this position, The program will be close, Please check picture on Picture folder and try again')
    running = False
    image = pygame.image.load('./picture/imgNotFound.jpg')
    image = pygame.transform.scale(image, (1040, 780))
# End define pygame interface

#define and build model
model  = Sequential()
```

```
model.add (Dense(8, activation='relu', input_dim=2))
model.add (Dense(8, activation='relu'))
model.add (Dense(2))
try:
    model.load_weights('myweight.weights.h5')
except:
    print('can not load weight of model which has been tranned, This model have been create with random weight. It cant use for predict angle of servo. Program will be close, Please check file weight myweight.weights.h5 and try again')
    running = False

while running:
    window.blit(image, (0, 0))
    # loop through the list of Event
    for event in pygame.event.get():
        # to end the event/loop
        if event.type == pygame.QUIT:
            print("Thanks for using this program")
            # it will deactivate the pygame library
            pygame.quit()
            quit()
        # to display when screen update
        pygame.display.flip()
        if event.type == pygame.MOUSEBUTTONUP:
            print("[System notice: On processing, Please wait...]")
            positionMouse = pygame.mouse.get_pos()
            state = []
            x = positionMouse[0] / 1040
            y = positionMouse[1] / 780
```

```
            state.append(x)
            state.append(y)
            state = np.array(state)
            state.resize((1,2))
            outModel = model.predict(state, verbose=1)
            phi1 = 90*outModel[0][0] + 90
            phi2 = 90*outModel[0][1] + 90
            phi1 = int(phi1)
            phi2 = int(phi2)
            print('Model predict angle of servo Phi1: {}, Phi2: {}'.format(phi1, phi2))
```

Phụ lục 4: Code sử dụng trên arduino

```
#include <Servo.h>
#include <string.h>
Servo servo1;
Servo servo2;
void setup(){
  servo1.attach(9);
  servo2.attach(10);
  servo1.write(90);
  servo2.write(90);
  Serial.begin(9600);
  Serial.setTimeout(100);
}
int phi1 = 90;
int phi2 = 90;
String command = "";
void loop(){
  if (Serial.available() > 0){
    command = Serial.readString();
```

```
    long i = command.toInt();
    phi1 = i / 1000;
    phi2 = i % 1000;
    String retu = String(phi1) + '-' + String(phi2) + '\n';
    Serial.print(retu);
   }
  //Serial.write("hi");
  servo1.write(phi1);
  servo2.write(phi2);
  delay(10);
}
```