

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

Bùi Anh Khoa

KHÓA LUẬN TỐT NGHIỆP
Nghiên cứu và xây dựng chương trình điều khiển xe tự
hành với Deep Learning

Research and Implement Control Program for Autonomous Car
with Deep Learning

KỸ SỰ NGÀNH KỸ THUẬT MÁY TÍNH

TP. HỒ CHÍ MINH, 2019

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH**

Bùi Anh Khoa – 15520364

**KHÓA LUẬN TỐT NGHIỆP
Nghiên cứu và xây dựng chương trình điều khiển xe tự
hành với Deep Learning**

**Research and Implement Control Program for Autonomous Car
with Deep Learning**

KỸ SỰ NGÀNH KỸ THUẬT MÁY TÍNH

**GIẢNG VIÊN HƯỚNG DẪN
Ths. Phạm Minh Quân**

TP. HỒ CHÍ MINH, 2019

DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số
ngày của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. – Chủ tịch.
2. – Thư ký.
3. – Ủy viên.

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc Lập - Tự Do - Hạnh Phúc

TP. HCM, ngày.....tháng.....năm.....

**NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP
(CỦA CÁN BỘ HƯỚNG DẪN)**

MÔ HÌNH MÔ PHỎNG XE TỰ HÀNH

Nhóm sinh viên thực hiện:

Bùi Anh Khoa

15520364

Cán bộ hướng dẫn:

ThS.Phạm Minh Quân

Đánh giá khóa luận

1. Về cuốn báo cáo:
.....
2. Về nội dung nghiên cứu:
.....
3. Về chương trình ứng dụng:
.....
4. Về thái độ làm việc của sinh viên:
.....

Đánh giá chung:
.....

Điểm từng sinh viên:

Bùi Anh Khoa:/10

Người nhận xét

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc Lập - Tự Do - Hạnh Phúc

TP. HCM, ngày.....tháng.....năm.....

**NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP
(CỦA CÁN BỘ PHẢN BIỆN)**

MÔ HÌNH MÔ PHỎNG XE TỰ HÀNH

Nhóm sinh viên thực hiện:

Bùi Anh Khoa

15520364

Cán bộ phản biện:

Ths.Phạm Minh Quân

Đánh giá khóa luận

5. Về cuốn báo cáo:

.....

6. Về nội dung nghiên cứu:

.....

7. Về chương trình ứng dụng:

.....

8. Về thái độ làm việc của sinh viên:

.....

Đánh giá chung:

.....

Điểm từng sinh viên:

Bùi Anh Khoa:/10

Người nhận xét

LỜI CẢM ƠN

Để hoàn thành được khóa luận tốt nghiệp này, chúng em xin chân thành gửi lời cảm ơn đến quý thầy cô Trường Đại học Công nghệ Thông tin – Đại học Quốc gia Thành phố Hồ Chí Minh nói chung và quý thầy cô Khoa Kỹ thuật Máy tính nói riêng đã truyền đạt cho em kiến thức và những kinh nghiệm quý báu trong suốt chặng đường 5 năm học vừa qua.

Xin gửi lời cảm ơn chân thành đến thầy Ths. Phạm Minh Quân, người đã dành những thời gian quý báu và kinh nghiệm của mình, trực tiếp hướng dẫn tận tình cho em hoàn thành khóa luận tốt nghiệp.

Xin chân thành cảm ơn sự giúp đỡ của các bạn bè, anh chị, những người đã giúp đỡ chúng em tìm kiếm, thu thập thông tin trong suốt quá trình thực hiện khóa luận.

Một lần nữa xin chân thành cảm ơn đến tất cả những người đã giành thời gian, công sức giúp đỡ chúng em hoàn thành khóa luận tốt nghiệp. Trong quá trình thực hiện khóa luận, tất nhiên không tránh khỏi những sai lầm, thiếu sót, em mong quý thầy cô, các bạn, anh chị bỏ qua và thứ lỗi.

Sinh viên thực hiện
Bùi Anh Khoa
Khoa Kỹ thuật Máy tính. Lớp KTMT2015

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
TRƯỜNG ĐẠI HỌC Độc Lập - Tự Do - Hạnh Phúc
CÔNG NGHỆ THÔNG TIN

TP. HCM, ngày.....tháng.....năm.....

ĐỀ CƯƠNG CHI TIẾT

MÔ HÌNH MÔ PHỎNG XE TỰ HÀNH

Cán bộ hướng dẫn: ThS.Phạm Minh Quân

Thời gian thực hiện: Từ ngày 09/09/2019 đến ngày 29/12/2019

Sinh viên thực hiện:

Bùi Anh Khoa – 15520364

Nội dung đề tài:

• **Lý do chọn đề tài:**

Năm được tầm quan trọng của tự động hóa trong đời sống và xe tự hành đang là một trong những đề tài nóng hổi của thế giới với sự tham gia của nhiều hãng lớn cho nên nhóm đã quyết định theo hướng này để đón đầu công nghệ xử lý ảnh, máy học, AI. Và board nhúng Nvida Jetson là một trong những board chuyên dùng trong lĩnh vực này

• **Mục tiêu của đề tài:**

Nghiên cứu, thiết kế, mô phỏng và xây dựng mô hình xe tự hành với khả năng đi vào đúng làn đường của mình và tuân thủ theo những biển báo.

Mô phỏng hệ thống chương trình thông qua Unity, socket Communication.

Deep Learning: xây dựng chương trình tự động nhận diện làn đường, tự nhận diện biển báo, tự điều khiển hướng và tốc độ xe đi theo quỹ đạo đã chọn; định vị cục bộ xe trong môi trường outdoor bằng ảnh chiều sâu từ camera 3D.

• **Nội dung chính:**

Viết chương trình mô phỏng xe chạy bằng Unity thông qua giao tiếp Socket. Demo và kiểm thử bằng phần mềm được build ra từ Unity và Hercules (Cho kiểm tra giao thức với data đầu vào nhập bằng tay)

Viết chương trình nhận diện làn đường đi để đưa ra góc bẻ lái. Demo và kiểm thử bằng phần mềm Unity, video, trên xe RC

Viết chương trình phát hiện và nhận diện biển báo. Demo và kiểm thử bằng phần mềm Unity, video và trên xe RC

Demo trên Unity: xe mô phỏng hoàn thành đường chạy. Unity sẽ có vai trò như server tiếp nhận bước đi của xe, phần mềm ngoài sẽ là chương trình tìm làn đường, góc lái, tốc độ, biển báo

Demo trên xe RC: làm sa hình với background màu đen, line đường màu vàng hoặc trắng hoặc cỏ xanh lá, biển báo được thu nhỏ so với thực tế với tỉ lệ 3/10 Nghiên cứu xử lý phát hiện và nhận diện biển báo với phương pháp deep learning và machine learning.

- **Kết quả mong đợi:**

Nghiên cứu và áp dụng thành công xử lý ảnh, deep learning, machine learning vào board Nvidia Jetson. Điều khiển xe RC chạy ổn định trên nhiều loại đường và biển báo, làm tiền đề để tương lai ứng dụng trên xe golf, xe ô tô.

- **Phương pháp thực hiện:**

- Tìm đọc tài liệu và viết chương trình về nhận diện làn đường
- Tìm đọc tài liệu và viết chương trình về phát hiện và nhận diện biển báo
- Nghiên cứu viết chương trình giao tiếp socket C#
- Nghiên cứu vẽ bản đồ mô phỏng trong Unity
- Nghiên cứu chuyển chương trình một luồng thành đa luồng
- So sánh kết quả công nghệ xử lý ảnh bằng opencv với xử lý ảnh bằng deep learning
- Tìm hiểu, thiết kế board điều khiển và viết driver

- **Kế hoạch thực hiện:**

Tuần	Nội dung	Công việc	Người thực hiện

1 (09/09/2019 - 15/09/2019)	Tìm hiểu tổng quan về Unity	Cách vẽ một bản đồ Thư viện xe Thư viện đường đi	Bùi Anh Khoa
2 (16/09/2019 - 22/09/2019)	Tìm hiểu về Socket, Json, Cách điều khiển xe thông qua giao tốc	Tìm hiểu TCP/IP C# Tìm hiểu thư viện Newtonsoft.Json Tìm hiểu script của điều khiển xe	
3 (23/09/2019 - 29/09/2019)	Tìm hiểu Opencv	Tìm đọc và thực hiện tutorial của thư viện OpenCV và những trang khác	
4 (30/09/2019 - 06/10/2019)	Tìm hiểu thuật toán tìm lane line qua thuật toán contour và sliding window	Tìm hiểu lọc màu Tìm hiểu lọc nhiễu Tìm hiểu contour, tâm của đường đi Tìm hiểu phương trình đường cong qua sliding window	
5 (07/10/2019 - 13/10/2019)	Triển khai thực hiện tìm lane line qua Unity	Chạy trương trình cùng với Unity chế độ Manual (Để kiểm tra thông số góc bẻ lái và tốc độ của từng vị trí), chế độ Auto Pilot để thực nghiệm xe tự động chạy	

		trên đường (Không có bất kì điều khiển nào khác can thiệp)	
6 (14/10/2019 - 20/10/2019)	Tìm hiểu keras	Cách sử dụng Các application Các layer Các ví dụ căn bản như mnist, ...	
7 (21/10/2019 - 27/10/2019)	Tìm hiểu thuật toán phát hiện biển báo bằng opencv và nhận diện biển báo bằng keras	Tìm hiểu hàm Inrange Tìm hiểu cách nhận diện hình tròn Thu thập data online và bên ngoài Tìm hiểu các lớp để có độ chính xác nhận diện biển báo tối ưu	
8 (28/10/2019 - 03/11/2019)	Tìm hiểu nhận diện làn đường và góc đánh lái bằng Keras	Thu thập data bằng thuật toán Opencv và bằng tay Tìm hiểu những lớp của mạng Training và thử nghiệm	
9 (04/11/2019 - 10/11/2019)	Tìm hiểu phát hiện và nhận diện biển báo với YOLO	Tìm hiểu cách ứng dụng YOLO bằng keras Train data và đưa vào thử nghiệm với video	
10 (11/11/2019)	Tìm hiểu multithreading và multiprocessing.	Tìm hiểu multithreading và multiprocessing programing qua các ví dụ căn bản	

- 17/11/2019)	Ứng dụng vào chương trình đã viết dạng 1 thread only	Ứng dụng vào phần mềm với 2 luồng	
11 (18/11/2019 - 24/11/2019)	Tìm hiểu board Nvidia Jetson TX2	Lập trình điều khiển GPIO Tìm hiểu ứng dụng, thư viện Cài đặt môi trường cho board (board arm)	
12 (25/11/2019 - 01/12/2019)	Tìm hiểu về xe RC	Tìm hiểu cách điều khiển động cơ servo, góc tối đa của servo mà xe có thể đạt tới Tìm hiểu cách điều khiển động cơ ESC Thiết kế cơ khí để điều khiển cân bằng xe	
13 (02/12/2019 - 08/12/2019)	Thiết kế shield, đặt mạch, kiểm thử	Thiết kế sơ đồ nguyên lý và layout mạch PCB Kiểm thử trên board Nvidia	
14 (09/12/2019 - 15/12/2019)	Viết driver và đo đặc thông số PWM	Tìm hiểu Board PCA9685 Viết driver điều khiển gpio (Đèn để debug), servo, esc bằng python Đo đặc vùng PWM tiến, lùi của ESC để tạo vùng cố định cho	

		công thức tính theo % tốc độ nhập vào và kiểm thử	
15 (16/12/2019 - 22/12/2019)	Thiết kế sa hình	<p>Thiết kế sa hình với góc cong tối đa là 30 độ</p> <p>Nền đen, line đường màu trắng hoặc vàng hoặc cỏ xanh lá</p>	
16 (23/12/2019 - 29/12/2019)	Kiểm thử trên xe thực tế	<p>Cho xe chạy trên sa hình với lúc đầy pin và đến lúc pin giảm 50%</p> <p>Cho xe chạy ổn định tối thiểu 7 vòng vào mức pin đầy</p> <p>Yêu cầu tốc độ của xe là 3/10 tốc độ tối đa trở lên</p> <p>Có thể xử lý được khi leo qua 1 vật dốc nhỏ</p> <p>Thiết kế kỹ thuật giảm rung của ảnh thu vào trong chương trình</p>	

Xác nhận của CBHD

TP. HCM, ngày....thángnăm.....

Sinh viên

MỤC LỤC

TÓM TẮT KHÓA LUẬN	1
CHƯƠNG 1. TỔNG QUAN	2
1.1 Tình hình trong nước	2
1.2 Tình hình ngoài nước	2
1.3 Mục tiêu nghiên cứu	5
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	7
2.1 Tổng quan về Unity [5]	7
2.1.1 Unity là gì	7
2.1.2 Quá trình phát triển	7
2.1.3 Một số thống kê	7
2.1.4 Ưu điểm	9
2.2 Thành phần và khái niệm cơ bản trong Unity [5]	10
2.2.1 Các thành phần trong Unity Editor	10
2.2.2 Các khái niệm cơ bản trong Unity	12
2.3 Socket (TCP/IP) C#	14
2.3.1 Socket là gì	14
2.3.2 Stream Socket	14
2.4 Mô hình xe RGT ROCK HAMMER	15
2.5 Nghiên cứu cấu trúc của những bộ phận trên xe	17
2.6 Tìm hiểu thuật toán tìm đường đi sử dụng thư viện OpenCV	29
2.6.1 Nhận diện màu sắc	29
2.6.1.1 Hệ màu [14]	29

2.6.2	Lọc màu đường và biển báo	33
2.6.3	Thuật toán tìm góc lệch sử dụng contour.....	33
2.6.3.1	Crop image	33
2.6.3.2	Tìm contour	34
2.6.4	Thuật toán tìm góc lệch sử dụng sliding window	34
2.6.4.1	Cân chỉnh camera [17]	34
2.6.4.2	Bẻ thẳng frame ảnh.....	36
2.6.4.3	Biểu đồ những điểm thuộc làn đường	38
2.6.4.4	Chia khung làn đường (Sliding window)	40
2.6.4.5	Phương trình đường cong	41
2.6.4.6	Tính toán dự đoán góc bẻ lái	42
2.6.4.7	Tính toán khoảng cách lệch so với 2 bên làn đường	43
2.6.4.8	Lưu lịch sử cho những frame kế tiếp để dự đoán.....	43
2.6.5	Thuật toán tìm biển báo.....	44
2.6.5.1	Nhận diện biển báo	44
2.6.5.2	Những thành phần trong mạng neural	47
2.6.5.3	Huấn luyện và Dự đoán	50
2.7	YOLO v3 [27]	51
2.7.1	Giới thiệu.....	51
2.7.2	Thuật toán.....	51
2.7.3	Kiến trúc mạng	53
2.7.4	Scales: Xử lý những kích thước khác nhau của vật thể	54
2.7.5	Kiến trúc ResNet-alike: Cách tốt hơn tìm ra những feature riêng biệt	56
2.7.6	Không có lớp softmax: phân loại đa lớp	56

2.8	Board Nvidia Jetson Tx2 và các ngoại vi	57
2.8.1	Thông số kỹ thuật: [29]	57
2.8.2	Camera Orbec Astra	58
2.8.3	Driver Controller shield	60
2.8.3.1	PCA 9685	60
2.8.3.2	Pinout board Nvidia trên shield.....	61
2.8.4	Thư viện	63
2.8.4.1	I2CDEV	63
2.8.4.2	SMBUS2.....	63
2.8.4.3	Tensorflow.....	63
2.8.4.4	Keras.....	66
2.8.4.5	Scikit-image [33]	67
2.8.4.6	Openni2	67
2.9	Lập trình đa luồng.....	67
2.9.1	Luồng (thread). Khác nhau giữa thread và process (tiến trình)	67
2.9.2	Đa luồng (Multithreading)	68
2.9.3	Đồng bộ hóa các thread trong python	69
2.9.4	Ứng dụng đa luồng	69
CHƯƠNG 3. XÂY DỰNG MÔ HÌNH VÀ GIẢI PHÁP		71
3.1	Chương trình mô phỏng Unity	71
3.1.1	Giao diện	71
3.1.2	Flowchart.....	73
3.2	Mô hình xe RC	75
3.2.1	Tổng thể mô hình xe	75

3.2.2	Camera	76
3.2.3	Shield.....	76
3.3	Chương trình điều khiển.....	79
3.3.1	Nhận diện làn đường bằng opencv.....	79
3.3.1.1	Contour	79
3.3.1.2	Sliding window.....	80
3.3.2	Nhận diện làn đường bằng Deep learning (CNN).....	81
3.3.3	Phát hiện và nhận diện biển báo bằng Opencv và CNN [32]	81
3.3.4	Phát hiện và nhận diện biển báo bằng YOLO v3.....	82
3.3.5	Chương trình điều khiển đa luồng.....	82
CHƯƠNG 4.	KẾT QUẢ.....	84
4.1	Chương trình mô phỏng Unity	84
4.1.1	Mô hình chương trình.....	85
4.2	Liên kết giữa Python/C++ (Client) và C# (Unity Server)	85
4.3	Unity Server.....	86
4.4	Client	87
4.5	Mô hình xe RC	89
4.5.1	Tổng thể mô hình xe	89
4.5.2	Camera	90
4.5.3	Driver shield.....	91
4.5.4	Kết quả	91
4.5.5	Mô hình chương trình điều khiển.....	93
4.6	Chương trình điều khiển xe không sử dụng multithreading.....	93
4.6.1	Opencv và CNN	93

4.6.2	YOLO v3.....	94
4.7	Chương trình điều khiển xe sử dụng multithreading.....	96
4.7.1	OpenCV và CNN	96
4.8	So sánh giữa OpenCV + CNN và YOLOv3.....	96
4.9	So sánh Tiny YOLO v3 và Fully-Connected YOLO v3	97
4.10	So sánh chương trình điều khiển non-multithreading và multithreading.....	97
4.11	Stress test trên xe RC.....	97
CHƯƠNG 5.	KẾT LUẬN	99
5.1	Kết luận.....	99
5.2	Kiến nghị	99
TÀI LIỆU THAM KHẢO	100	
PHỤ LỤC	104	

DANH MỤC HÌNH VẼ

Hình 1.1. Xe tự lái của Google [3]	3
Hình 1.2. Công nghệ bên trong xe tự lái Tesla [4].....	3
Hình 2.1. Biểu đồ phân bố những phần mềm làm game [6]	8
Hình 2.2. Biểu đồ thị trường thế giới về thu nhập từ game (di động và PC) [6]	8
Hình 2.3. Những platform Unity hỗ trợ [6]	9
Hình 2.4. Giao diện chính của phần mềm Unity	11
Hình 2.5. Mô hình socket TCP/IP [7].....	15
Hình 2.6. Thiết kế bên trong của xe RGT Rock Hammer.....	16
Hình 2.7. Thiết kế toàn thể của xe RGT Rock Hammer	17
Hình 2.8. Minh họa hệ thống dẫn động 4 bánh	18
Hình 2.9. Các loại hệ thống truyền động 4 bánh	19
Hình 2.10. ESC của xe RGT Rock Hammer	21
Hình 2.11. Động cơ có chổi than.....	22
Hình 2.12. Động cơ không chổi than.....	23
Hình 2.13. Tín hiệu điều khiển động cơ servo	25
Hình 2.14. Phản hồi hệ thống động cơ	25
Hình 2.15. Mạch vòng điều khiển động cơ servo.....	25
Hình 2.16. Pin Lipo	26
Hình 2.17. Phân biệt giữa 3 kênh R, G, B	30
Hình 2.18. Phân biệt giữa 3 kênh L, A, B	31
Hình 2.19. Phân biệt giữa 3 kênh H, S, V	32
Hình 2.20. Vòng tròn hệ màu HSV	32
Hình 2.21. Không gian 3 chiều của không gian màu HSV	33
Hình 2.22. Crop ảnh [14].....	33
Hình 2.23. Ví dụ về contour trong Opencv [16].....	34
Hình 2.24. Ví dụ về radial distortion	35
Hình 2.25. Cân chỉnh frame ảnh.....	36
Hình 2.26. Kết quả sau khi cân chỉnh.....	36

Hình 2.27. Mô hình về birdview	37
Hình 2.28. Birdview với lane đường	38
Hình 2.29. Ví dụ về histogram [19].....	38
Hình 2.30. Histogram của lane đường.....	40
Hình 2.31. Flowchart sliding window	40
Hình 2.32. Sliding window	41
Hình 2.33. Skip sliding window	42
Hình 2.34. Tính góc	42
Hình 2.35. Kết quả tính độ lệch.....	43
Hình 2.36. Chuyển ảnh màu thành xám	44
Hình 2.37. Ví dụ giải thích Gaussian blur bằng toán học	45
Hình 2.38. Gaussian blur	45
Hình 2.39. Lọc màu xanh	46
Hình 2.40. Threshold [20]	46
Hình 2.41. Contour	47
Hình 2.42. Các bước tạo ra map 2D	47
Hình 2.43. Ví dụ về Conv2d [22]	47
Hình 2.44. Ví dụ MaxPooling [18].....	48
Hình 2.45. Ví dụ MaxPooling trong thực tế [23]	48
Hình 2.46. Quá trình biến đổi của flatten [24]	49
Hình 2.47. Ví dụ Zero padding [25]	50
Hình 2.48. Ví dụ Valid Padding [25]	50
Hình 2.49. Chuỗi xử lý của YOLO [27].....	52
Hình 2.50. Ví dụ K-means clustering [28]	53
Hình 2.51. Kiến trúc mạng YOLO v3 [27]	54
Hình 2.52. Minh họa quá trình huấn luyện đặc trưng với nhiều tỉ lệ khác nhau [27]	55
Hình 2.53. Ví dụ về Residual Blocks [27].....	56
Hình 2.54. Board Nvidia Jetson Tx2	57
Hình 2.55. Camera Orbbec Astra [30].....	59

Hình 2.56. PCA9685 [31].....	60
Hình 2.57. Hình minh họa cho Tensorflow [32]	66
Hình 2.58. Minh họa về tìm cạnh trong scikit-image.....	67
Hình 2.59. Sự khác nhau giữa thread và process [34]	68
Hình 3.1. Giao diện lựa chọn độ phân giải trong Unity	71
Hình 3.2. Giao diện hoạt động.....	72
Hình 3.3. Flowchart xử lý của server Tcp/Ip.....	73
Hình 3.4. Flowchart xử lý tốc độ và góc bẻ lái	73
Hình 3.5. Flowchart xử lý LapTime	74
Hình 3.6. Flowchart giao tiếp giữa client và server	75
Hình 3.7. Bản vẽ sơ lược về đế.....	76
Hình 3.8. Schematic của shield Jetson TX	77
Hình 3.9. PCB của shield Jetson TX	78
Hình 3.10. Flowchart xử lý sử dụng contour.....	79
Hình 3.11. Flowchart xử lý sử dụng Sliding window	80
Hình 3.12. Flowchart xử lý sử dụng CNN	81
Hình 3.13. Flowchart xử lý sử dụng OpenCV và CNN	81
Hình 3.14. Flowchar xử lý sử dụng YOLO v3	82
Hình 3.15. Flow chart xử lý sử dụng đa luồng	82
Hình 4.1. Mô hình chương trình điều khiển Unity	85
Hình 4.2. Chương trình điều khiển sử dụng YOLOv3 và phần mềm mô phỏng Unity	87
Hình 4.3. Chương trình sử dụng OpenCV và CNN	88
Hình 4.4. Số liệu khi xử lí sử dụng OpenCV và CNN	88
Hình 4.5. Mô hình xe nhìn từ trên xuống	89
Hình 4.6. Hệ thống lò xo	90
Hình 4.7. Camera Orbbec Astra 3D	90
Hình 4.8. Driver shield	91
Hình 4.9. Mô hình chương trình điều khiển xe RC Car	93
Hình 4.10. Biểu đồ thời gian xử lí với OpenCV và CNN	94

Hình 4.11. Biểu đồ thời gian xử lí với YOLO v3 (đơn vị fps) 95

Hình 4.12. Biểu đồ thời gian xử lí với OpenCV và CNN (Multithreading) 96

DANH MỤC BẢNG

Bảng 2.1. Bảng đầu ra chân của board Nvidia Jetson	62
Bảng 4.1. Bảng kết quả và tốc độ xử lý giữa Tiny Yolo và Full Yolo.....	97
Bảng 4.2. Bảng kết quả độ chính xác và tốc độ xử lý (fps) một luồng và đa luồng	97

DANH MỤC CÔNG THỨC

Công thức 2.1. Cân chỉnh camera với radial distortion.....	35
Công thức 2.2. Cân chỉnh camera với tangential	35
Công thức 2.3. Hệ số biến dạng.....	35
Công thức 2.4. Ma trận camera 3x3	35
Công thức 2.5. Hàm getPerspectiveTransform	36
Công thức 2.6. Hàm warpPerspective	37
Công thức 2.7. Hàm numpy.polyfit.....	41
Công thức 2.8. Thuật toán hàm numpy.polyfit	42
Công thức 2.9. Công thức tính độ lệch (curvature).....	43
Công thức 2.10. Biến đổi ảnh RGB sang GRAY	44
Công thức 2.11. Biến đổi Gaussian trong không gian 1 chiều	44
Công thức 2.12. Biến đổi Gaussian trong không gian 2 chiều	44

DANH MỤC TỪ VIẾT TẮT

RC: Radio controlled

CNN: Convolutional neural network

GPIO: General-purpose input/output

I2C: Inter-Integrated Circuit

PCB: Printed circuit board

PWM: Pulse width modulation

FPS: Frame per second

GPU: Graphics Processing Unit

SLAM: Simultaneous Localization And Mapping

PC: Personal Computer

TCP: Transmission Control Protocol

WD: Wheel Drive

ESC: Electronic Speed Control

LVC: Low Voltage Cut-Off

BEC: Battery Eliminator Circuit

UBEC: Ultimate Battery Eliminator Circuit

IC: Integrated Circuit

FET: Field-effect Transistor

AC: Alternating Current

DC: Direct Current

Lipo: Lithium Polymer

NiCad: Nickel-cadmium battery

NiMH: Nickel-metal hydride battery

YOLO: You Only Look Once

FPN: Feature Pyramid Network

SSD: Single Shot Detector

TTL: Transistor-Transistor Logic

CNTK: Cognitive Toolkit

TPU: Tensor processing unit

API: Application Program Interface

IPC: Inter-Process Communication

CNC: Computer Numerical Control

TÓM TẮT KHÓA LUẬN

Cuộc cách mạng về khoa học công nghệ diễn ra từng ngày đang làm thay đổi toàn diện và sâu sắc cuộc sống cũng như quá trình sản xuất của con người. Trong xu thế phát triển đó, nền công nghệ non trẻ của Việt Nam, vốn đã tụt hậu khá xa, nếu không muốn tụt lại phía sau hơn nữa cần phải vừa tranh thủ thành tựu của các nước tiên tiến, vừa phải có sự đột phá, sáng tạo trong tư duy để theo kịp với công nghệ thế giới. Nhắc đến tự động hóa trong ngành ô tô, chắc hẳn ai cũng nghĩ tới xe tự hành.

Để ứng dụng mô phỏng cho xe tự hành, hãng Nvidia đã cho ra board mạch Nvidia Jetson chuyên dùng cho xử lý ảnh, Deep Learning và Machine Learning. Board này sẽ được gắn trên những mô hình xe điều khiển cỡ lớn và tự động hoàn thành đường đi mà không có bất kỳ can thiệp nào từ con người trong lúc hoạt động. Hiện tại, khá ít công nghệ được phổ biến rộng rãi cho mảng này trên thế giới, vì thế hầu hết mọi người sẽ bắt đầu từ việc mô phỏng trong môi trường thực nghiệm và sau đó sẽ thử nghiệm trong môi trường thực tế và phát triển lên.

CHƯƠNG 1. TỔNG QUAN

1.1 Tình hình trong nước

Hiện tại, ở Việt Nam cũng đã có nhiều công ty như FPT, Ikorn, Bosch, ... đang có xu hướng nghiên cứu và phát triển xe tự hành. Mô hình nghiên cứu được triển khai và có tính lan tỏa mạnh, thu hút nhiều sự quan tâm của giới nghiên cứu khoa học công nghệ, với ước vọng tạo ra và ứng dụng xe tự hành trong tương lai ở đời sống xã hội Việt Nam. Các nghiên cứu xe tự hành tại Việt Nam hiện nay đang tiến lên xe thực tế từ mô hình xe tự hành. Những mô hình demo này sử dụng cơ chế ngoại vi như động cơ servo để bẻ bánh lái và các động cơ điện 1 chiều để mô phỏng xe tự hành ngoài đời thực. Ứng dụng đang hiện thực hóa trên board chuyên biệt cho việc xử lý ảnh như board Nvidia và chỉ dừng lại ở phân đoạn nhận diện đường trắng liền, biển báo, vật cản.

1.2 Tình hình ngoài nước

- Đã có nhiều hãng phát triển và thử nghiệm xe tự hành:
 - o Google: 23 chiếc. [1]
 - o Tesla: 39 chiếc. [2]
 - o Apple: 55 chiếc. [2]
- Các hãng công nghệ lớn đã kể trên và nhiều hãng khác vẫn đang trong quá trình nghiên cứu và phát triển để ứng dụng xe tự hành vào đời sống thực tế. trong đó được tích hợp nhiều công nghệ như camera 360, stereo camera, vi xử lý, GPU tốc độ cao.



Hình 1.1. Xe tự lái của Google [3]



Hình 1.2. Công nghệ bên trong xe tự lái Tesla [4]

Xe tự hành AGV [4] bắt đầu được chế tạo để vận chuyển các phôi gia công vào những năm 70. Vấn đề định hướng của xe tự hành là một trong những vấn đề cốt lõi cần phải giải quyết. Bài toán định hướng hiện nay: định hướng ngoài địa hình (outdoor). Quá trình định hướng gồm 4 bước: thu nhận cảnh quan môi trường, xác định vị trí, thiết kế quỹ đạo và tạo chuyển động. Với môi trường thuận lợi, quá trình nhận biết cho phép tạo ra bản đồ hay mô hình không gian phục vụ cho bài toán định vị và thiết kế quỹ đạo robot. Đối với môi trường phi cấu trúc hay thay đổi, robot cần có khả năng tự học quan sát môi trường để xác định được hướng đi của mình. Do đó lĩnh vực xác định hướng đi cho robot di động là một lĩnh vực mà các phương pháp trí tuệ nhân tạo như quá trình nhận biết môi trường, suy diễn và tìm hướng đi tối ưu có thể được áp dụng. Vấn đề định vị và tạo bản đồ là những vấn đề nghiên cứu trọng tâm ở robot di động thời gian qua. Quá trình định vị là quá trình robot xác định được hiện nó ở đâu trong không gian hoạt động. Để đạt được mục tiêu này, cần sử dụng nhiều cảm biến thu nhận các dữ liệu liên quan đến trạng thái của robot và môi trường xung quanh. Các dữ liệu này thường bị nhiễu và có sai số tích lũy nên cần có các phương pháp lọc động và sử dụng các phương pháp phối hợp cảm biến để có được số liệu đo chính xác hơn. Phương pháp định vị có phương thức là cục bộ hay toàn cục. Giải pháp đơn giản nhất là định vị cục bộ khi robot thường xuyên cập nhật vị trí của nó so với điểm xuất phát. Ngược lại các phương pháp định vị toàn cục không đòi hỏi biết vị trí của điểm xuất phát. Để khắc phục độ bất định của các thông tin đo được từ các cảm biến ta cần sử dụng các phương pháp xác suất. Các phương pháp định vị được sử dụng thường dựa trên nguyên lý lọc Bayes kết hợp với một thuật toán để quy để ước lượng được vị trí và hướng từ phương trình mô tả chuyển động của robot. Thời gian tính toán của bộ lọc Bayes lâu nên nhiều nghiên cứu gần đây tập trung vào tìm các phương pháp đơn giản hóa để giảm khối lượng tính toán. Quá trình đơn giản hóa này dẫn đến nhiều thuật toán định vị khác nhau phân làm 2

loại tùy thuộc vào cách mô tả độ tin cậy của dữ liệu. Nếu dữ liệu được mô tả bằng các hàm phân bố Gauss, ta có thể sử dụng phương pháp lọc Kalman. Nếu dữ liệu được mô tả bằng nhiều hàm phân bố xác suất khác nhau, ta có thể sử dụng các thuật toán định vị dựa trên quá trình Markov. Phương pháp định vị dựa trên phân bố Gauss và lọc Kalman chỉ ứng dụng hiệu quả cho bài toán định vị cục bộ. Các phương pháp định vị Markov có thể là phương pháp topologia, phương pháp lưới và phương pháp sử dụng các mẫu rời rạc của các giá trị trạng thái. Khi các dữ liệu robot được mô tả bằng các mẫu rời rạc ngẫu nhiên ta có thể sử dụng các phương pháp lọc phần tử (particle filter) để xác định được vị trí của robot tốt hơn. Do việc định vị và lập bản đồ cho robot có quan hệ mật thiết với nhau nên từ những năm 90 các nghiên cứu đã tập trung giải quyết hai vấn đề này đồng thời với tên chung là SLAM (Simultaneous Localization And Mapping). Các nghiên cứu gần đây liên quan nhiều đến lập bản đồ cho môi trường động. Bộ Lọc Kalman có thể sử dụng được cho bài toán này nhưng không cho độ chính xác cao. Vấn đề tìm được một thuật toán lập bản đồ cho môi trường động còn là một thách thức lớn. Nhiều vấn đề còn bỏ ngỏ như phân biệt các đối tượng tĩnh, đối tượng chuyển động và mô tả chúng trên bản đồ.

Ngoài các ứng dụng trong công nghiệp và quân sự, robot di động cũng đã đi vào cuộc sống đời thường như robot hút bụi, lau nhà, cọ bể bơi, lau kính, robot viện bào tàng, robot dịch vụ văn phòng, bệnh viện và ở các nơi công cộng. Các robot này yêu cầu mức độ nhận thức nhất định, có khả năng tự định hướng, di chuyển trong môi trường biến động và có giao tiếp người-máy thân thiện. Ứng dụng robot di động trong giám sát, canh gác, cảnh báo về an ninh cũng là một hướng được nhiều công ty bảo vệ an ninh sử dụng.

1.3 Mục tiêu nghiên cứu

Thiết kế một chiếc xe tự hành ở mức nhỏ có thể làm được những công việc sau:

- Nghiên cứu, thiết kế, mô phỏng và xây dựng mô hình xe tự hành với khả năng đi vào đúng làn đường của mình và tuân thủ theo những biển báo.
- Mô phỏng hệ thống chương trình thông qua Unity, socket Communication.
- Deep Learning: xây dựng chương trình tự động nhận diện làn đường, tự nhận diện biển báo, tự điều khiển hướng và tốc độ xe đi theo quỹ đạo đã chọn; định vị xe trong môi trường outdoor.

Công nghệ:

- Nhận diện đường đi bằng OpenCV
- Nhận diện đường đi bằng máy học (Keras)
- Nhận diện biển báo bằng OpenCV
- Nhận diện biển báo bằng máy học (Keras)
- Phân loại biển báo bằng máy học (Keras)
- Hệ điều hành Linux (Ubuntu) trên board Nvidia TX2

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về Unity [5]

2.1.1 Unity là gì

Unity là một “cross- platform game engine” tạm hiểu là công cụ phát triển game đa nền tảng được phát triển bởi Unity Technologies. Game engine này được sử dụng để phát triển game trên PC, consoles, thiết bị di động và trên websites.

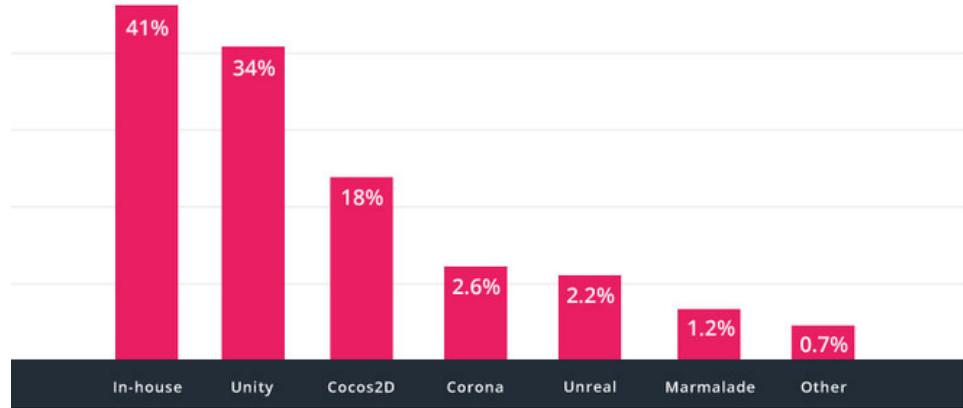
2.1.2 Quá trình phát triển

Ra mắt đầu tiên vào năm 2005 tại sự kiện Apple’s Worldwide Developer Conference bởi nhà sáng lập David Helgason, trải qua hơn 12 năm phát triển, nay Unity đã có version 5.5 hoàn thiện hơn về rất nhiều mặt. Tháng 5-2012 theo cuộc khảo sát Game Developer Megazine được công nhận là Game engine tốt nhất cho mobile. Năm 2014 Unity thắng giải “Best Engine” tại giải UK’s annual Develop Industry Excellence.

2.1.3 Một số thống kê

- Tính đến quý 3 năm 2016 đã có 5 tỉ lượt download game và ứng dụng được phát triển bởi Unity
- 2,4 tỉ thiết bị di động đã từng tải ít nhất 1 ứng dụng bởi unity.
- Trong top 1000 game Mobile miễn phí thì số lượng game tạo ra bởi Unity chiếm tới 34%

34% of top 1000 free mobile games are Made with Unity



Source: SourceDNA, Q1 2016

Hình 2.1. Biểu đồ phân bố những phần mềm làm game [6]

- Số lượng người dùng (gamer) của Unity đạt tới con số 770 triệu, trong khi đó số người thường xuyên sử dụng Twitter là 310 triệu người.
- Sự thay đổi trong cách thức chơi game của người chơi hay nói cách khác là xu hướng mọi người tập trung vào game trên di động nhiều hơn.

Video game market revenue worldwide, 2015-2019, US\$ billions



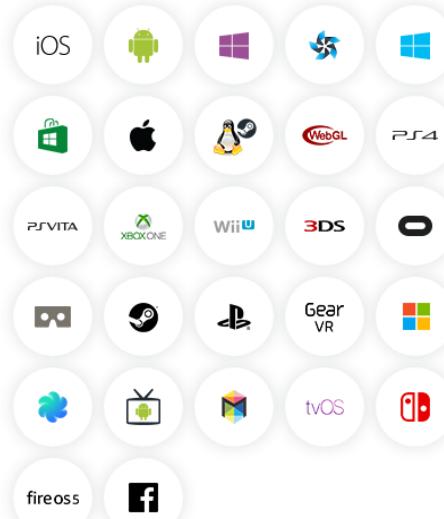
Source: Gartner (October 2013)

Hình 2.2. Biểu đồ thị trường thế giới về thu nhập từ game (di động và PC) [6]

2.1.4 Ưu điểm

- Chức năng cốt lõi đa dạng bao gồm: cung cấp công cụ dựng hình (kết xuất đồ họa) cho các hình ảnh 2D hoặc 3D, công cụ vật lý (tính toán và phát hiện va chạm), âm thanh, mã nguồn, hình ảnh động, trí tuệ nhân tạo, phân luồng, tạo dò ng dữ liệu xử lý, quản lý bộ nhớ, dựng ảnh đồ thị và kết nối mạng. Nhờ có các engine mà công việc làm game trở nên ít tốn kém và đơn giản hơn.
- Hỗ trợ đa nền tảng: Một trong các thế mạnh của Unity3D chính là khả năng hỗ trợ gần như toàn bộ các nền tảng hiện có bao gồm: PlayStation 3, Xbox 360, Wii U, iOS, Android, Windows, Blackberry 10, OS X, Linux, trình duyệt Web và cả Flash. Nói cách khác, chỉ với một gói engine, các studio có thể làm game cho bất kỳ hệ điều hành nào và dễ dàng convert chúng sang những hệ điều hành khác nhau. Đồng thời, đây cũng là giải pháp cho các game online đa nền tảng – có thể chơi đồng thời trên nhiều hệ điều hành, phần cứng khác nhau như Web, PC, Mobile, Tablet....

Platform support

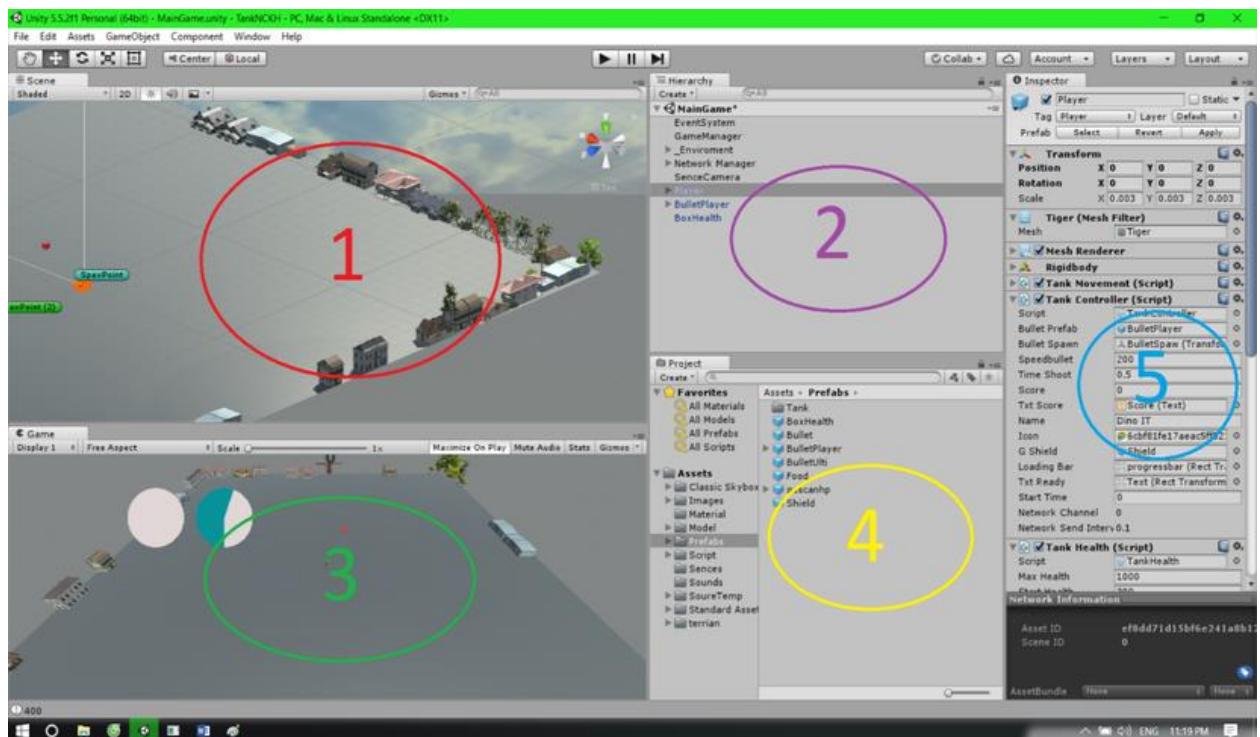


Hình 2.3. Những platform Unity hỗ trợ [6]

- Dễ sử dụng: Unity3D được built trong một môi trường phát triển tích hợp, cung cấp một hệ thống toàn diện cho các lập trình viên, từ soạn thảo mã nguồn, xây dựng công cụ tự động hóa đến trình sửa lỗi. Do được hướng đến đồng thời cả lập trình viên không chuyên và studio chuyên nghiệp, nên Unity3D khá dễ sử dụng. Hơn nữa, đây là một trong những engine phổ biến nhất trên thế giới, người dùng có thể dễ dàng tìm kiếm kinh nghiệm sử dụng của “tiền bối” trên các forum công nghệ.
- Tính kinh tế cao: Unity Technologies hiện cung cấp bản miễn phí engine Unity3D cho người dùng cá nhân và các doanh nghiệp có doanh thu dưới 100.000 USD/năm. Với bản Pro, người dùng phải trả 1.500 USD/năm – một con số rất khiêm tốn so với những gì engine này mang lại.

2.2 Thành phần và khái niệm cơ bản trong Unity [5]

2.2.1 Các thành phần trong Unity Editor



Hình 2.4. Giao diện chính của phần mềm Unity

1. Cửa sổ Scene

- Phần này phần hiển thị các đối tượng trong scenes một cách trực quan, có thể lựa chọn các đối tượng, kéo thả, phóng to, thu nhỏ, xoay các đối tượng ...
- Phần này có để thiết lập một số thông số như hiển thị ánh sáng, âm thanh, cách nhìn 2D hay 3D ... -Khung nhìn Scene là nơi bố trí các Game Object như cây cối, cảnh quan, enemy, player, camera, ... trong game. Sự bố trí hoạt cảnh là một trong những chức năng quan trọng nhất của Unity.

2. Cửa sổ hierarchy

- Tab hierarchy là nơi hiển thị các Game Object trong Sences hiện hành. Khi các đối tượng được thêm hoặc xóa trong Sences, tương ứng với các đối tượng đó trong cửa sổ Hierarchy.
- Tương tự trong tab Project, Hierarchy cũng có một thanh tìm kiếm giúp quản lý và thao tác với các Game Object hiệu quả hơn đặc biệt là với các dự án lớn.

3. Cửa sổ game

- Đây là mạn hình demo Game, là góc nhìn từ camera trong game.
- Thanh công cụ trong cửa sổ game cung cấp các tùy chỉnh về độ phân giải màn hình, thông số (stats), gizmos, tùy chọn bật tắt các component...

4. Cửa sổ project

- Đây là cửa sổ explorer của Unity, hiển thị thông tin của tất cả các tài nguyên (Assets) trong game của bạn.
- Cột bên trái hiển thị assets và các mục yêu thích dưới dạng cây thư mục tương tự như Windows Explorer. Khi click vào một nhánh trên cây thư mục thì toàn

bộ nội dung của nhánh đó sẽ được hiển thị ở khung bên phải. Ta có thể tạo ra các thư mục mới bằng cách Right click -> Create -> Folder hoặc nhấn vào nút Create ở góc trên bên trái cửa sổ Project và chọn Folder. Các tài nguyên trong game cũng có thể được tạo ra bằng cách này.

- Phía trên cây thư mục là mục Favorites, giúp chúng ta truy cập nhanh vào những tài nguyên thường sử dụng. Chúng ta có thể đưa các tài nguyên vào Favorites bằng thao tác kéo thả.
- Đường dẫn của thư mục tài nguyên hiện tại. Chúng ta có thể dễ dàng tiếp cận các thư mục con hoặc thư mục gốc bằng cách click chuột vào mũi tên hoặc tên thư mục.

5. Cửa sổ inspector

- Cửa sổ Inspector hiển thị chi tiết các thông tin về Game Object đang làm việc, kể cả những component được đính kèm và thuộc tính của nó. Bạn có thể điều chỉnh, thiết lập mọi thông số và chức năng của Game Object thông qua cửa sổ Inspector.
- Mọi thuộc tính thể hiện trong Inspector đều có thể dễ dàng tùy chỉnh trực tiếp mà không cần thông qua một kịch bản định trước. Tuy nhiên Scripting API cung cấp một số lượng nhiều và đầy đủ hơn do giao diện Inspector là có giới hạn.
- Các thiết lập của từng component được đặt trong menu. Các bạn có thể click chuột phải, hoặc chọn icon hình bánh răng nhỏ để xuất hiện menu.
- Ngoài ra Inspector cũng thể hiện mọi thông số Import Setting của asset đang làm việc như hiển thị mã nguồn của Script, các thông số animation, ...

2.2.2 Các khái niệm cơ bản trong Unity

1. Game Object

Một đối tượng cụ thể trong game gọi là một game object, có thể là nhân vật, đồ vật nào đó. Ví dụ: cây cối, xe cộ, nhà cửa, người...

2. Component

Một GameObject sẽ có nhiều thành phần cấu tạo nên nó như là hình ảnh (sprite render), tập hợp các hành động (animator), thành phần xử lý va chạm (collision), tính toán vật lý (physical), mã điều khiển (script), các thành phần khác... mỗi thứ như vậy gọi là một component của GameObject.

3. Sprite

Là một hình ảnh 2D của một game object có thể là hình ảnh đầy đủ, hoặc có thể là một bộ phận nào đó.

4. Animation

Là tập một hình ảnh động dựa trên sự thay đổi liên tục của nhiều sprite khác nhau.

5. Key Frame

Key Frame hay Frame là một trạng thái của một animation. Có thể được tạo nên từ 1 sprite hay nhiều sprite khác nhau.

6. Prefabs

Là một khái niệm trong Unity, dùng để sử dụng lại các đối tượng giống nhau có trong game mà chỉ cần khởi tạo lại các giá trị vị trí, tỉ lệ biến dạng và góc quay từ một đối tượng ban đầu. Ví dụ: Các đối tượng là đồng tiền trong game Mario đều có xử lý giống nhau, nên ta chỉ việc tạo ra một đối tượng ban đầu, các đồng tiền còn lại sẽ sử dụng prefabs. Hoặc khi ta lát gạch cho một cái nền nhà, các viên gạch cũng được sử dụng là prefabs.

7. Sounds

Âm thanh trong game.

8. Script

Script là tập tin chứa các đoạn mã nguồn, dùng để khởi tạo và xử lý các đối tượng trong game. Trong Unity có thể dùng C#, Java Script, BOO để lập trình Script.

9. Scenes

Quản lý tất cả các đối tượng trong một màn chơi của game.

10. Assets

Bao gồm tất cả những gì phục vụ cho dự án game như sprite, animation, sound, script, scenes...

11. Camera

Là một game object đặc biệt trong scene, dùng để xác định tầm nhìn, quan sát các đối tượng khác trong game.

12. Transform

Là 3 phép biến đổi tịnh tiến, quay theo các trục, và phóng to thu nhỏ một đối tượng

2.3 Socket (TCP/IP) C#

2.3.1 Socket là gì

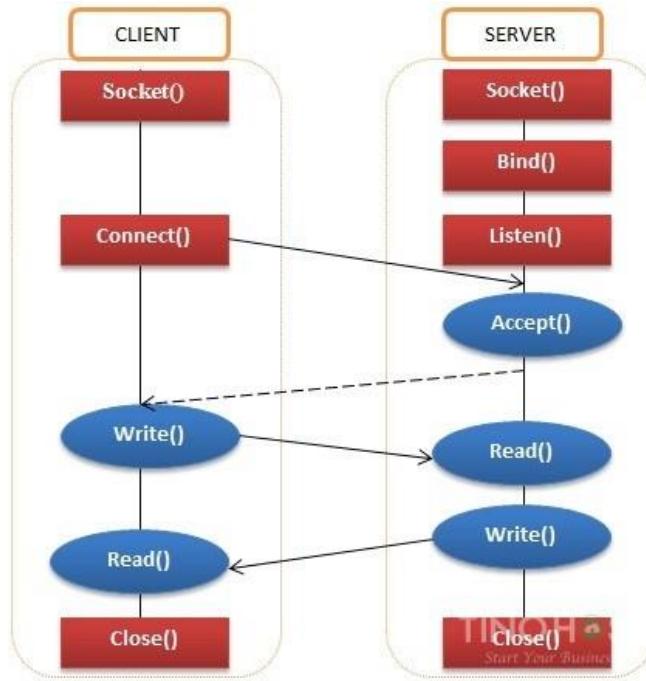
Socket là một điểm cuối (end-point) của liên kết truyền thông hai chiều (two-way communication) giữa hai chương trình chạy trên mạng. Các lớp Socket được sử dụng để biểu diễn kết nối giữa client và server, được ràng buộc với một cổng port (thể hiện là một con số cụ thể) để các tầng TCP (TCP Layer) có thể định danh ứng dụng mà dữ liệu sẽ được gửi tới.

Lập trình socket là lập trình cho phép người dùng kết nối các máy tính truyền tải và nhận dữ liệu từ máy tính thông qua mạng

Hiểu đơn giản, socket là thiết bị truyền thông hai chiều gửi và nhận dữ liệu từ máy khác.

2.3.2 Stream Socket

Dựa trên giao thức TCP(Tranmission Control Protocol), việc truyền dữ liệu chỉ thực hiện giữa 2 quá trình đã thiết lập kết nối. Do đó, hình thức này được gọi là **socket hướng kết nối**. [7]



Hình 2.5. Mô hình socket TCP/IP [7]

Ưu điểm: Có thể dùng để liên lạc theo mô hình client và sever. Nếu là mô hình client /sever thì sever lắng nghe và chấp nhận từ client. Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn. Đồng thời, mỗi thông điệp gửi phải có xác nhận trả về và các gói tin chuyển đi tuần tự.

Hạn chế: Có một đường kết nối (địa chỉ IP) giữa 2 tiến trình nên 1 trong 2 tiến trình kia phải đợi tiến trình kia yêu cầu kết nối.

2.4 Mô hình xe RGT ROCK HAMMER

RGT Rock Hammer [8] là mẫu xe địa hình tỉ lệ 1/10 của hãng RGT Racing. Đây là dòng xe chuyên địa hình đầy sức mạnh, với thiết kế khung sườn kim loại bắt mắt và độ bền cao, hệ thống truyền động 4 bánh (4WD), servo nhông kim loại, bộ hộp số tạo lực kéo mạnh, bánh lớn, gầm cao giúp xe di chuyển ở những địa hình phức tạp (đá, cát, sinh lầy hay cát nước...) với độ dốc lên tới 45 độ. Đây là mẫu xe phù hợp với những người thích

thể loại cơ bắp, leo trèo khỏe, không cần tốc độ cao, thích chế độ các chi tiết trên xe cho giống xe thật.



Hình 2.6. Thiết kế bên trong của xe RGT Rock Hammer

Ngoài những thiết kế như trên, xe được bổ sung:

- Mạch được bọc kín hạn chế nước văng vào
- ESC chống nước 40A
- Servo chống nước E1501 với sức kéo 15kg
- Phuộc nhún

Thông số kỹ thuật đầy đủ:

- Hãng: RGT Racing
- Tên sản phẩm: Rock Hammer
- Tỉ lệ: 1/10
- Kích thước: 470x270x215mm
- Chiều dài cơ sở: 350mm
- Khoảng sáng gầm xe: 80mm
- Bề rộng lốp: 52mm
- Đường kính lốp: 135mm

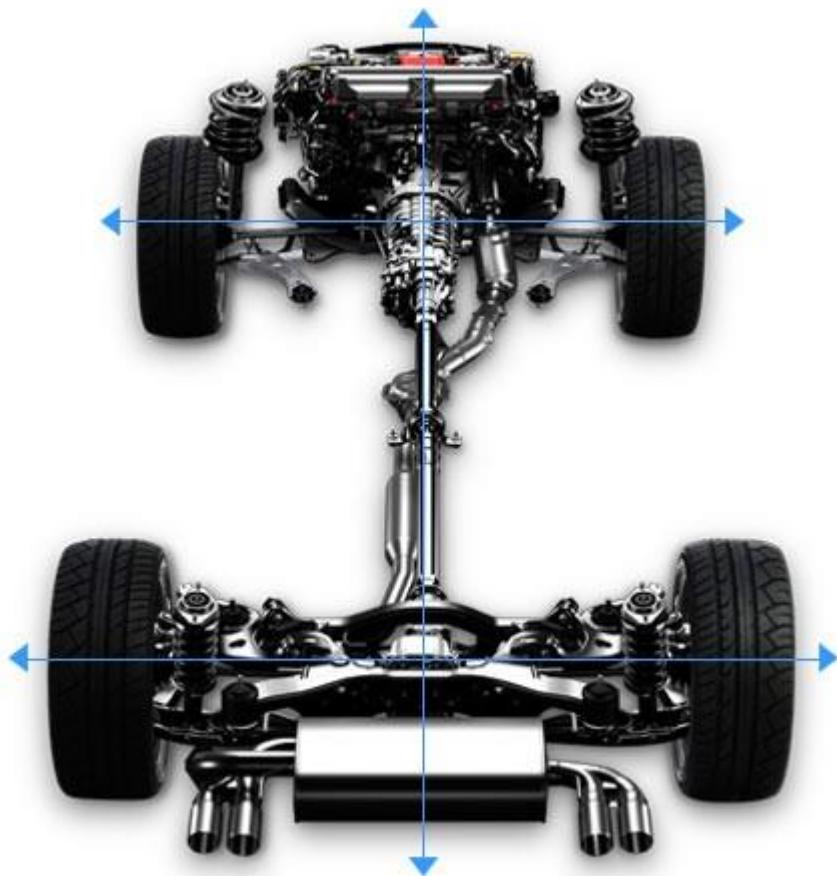
- Hex trực bánh: 12m
- Tần số điều khiển: 2.4GHz
- ESC: Hobbywing WP1040 40A, chống nước
- Motor: Brushed RC540-8020
- Servo: 15Kg
- Pin: Pin sạc 7.2V 2000 mAh
- Khoảng cách điều khiển: khoảng 100-120m
- Thời gian sạc: 3-4 tiếng
- Thời gian sử dụng: khoảng 15-20 phút
- Tốc độ: 25km/h



Hình 2.7. Thiết kế toàn thể của xe RGT Rock Hammer

2.5 Nghiên cứu cấu trúc của những bộ phận trên xe

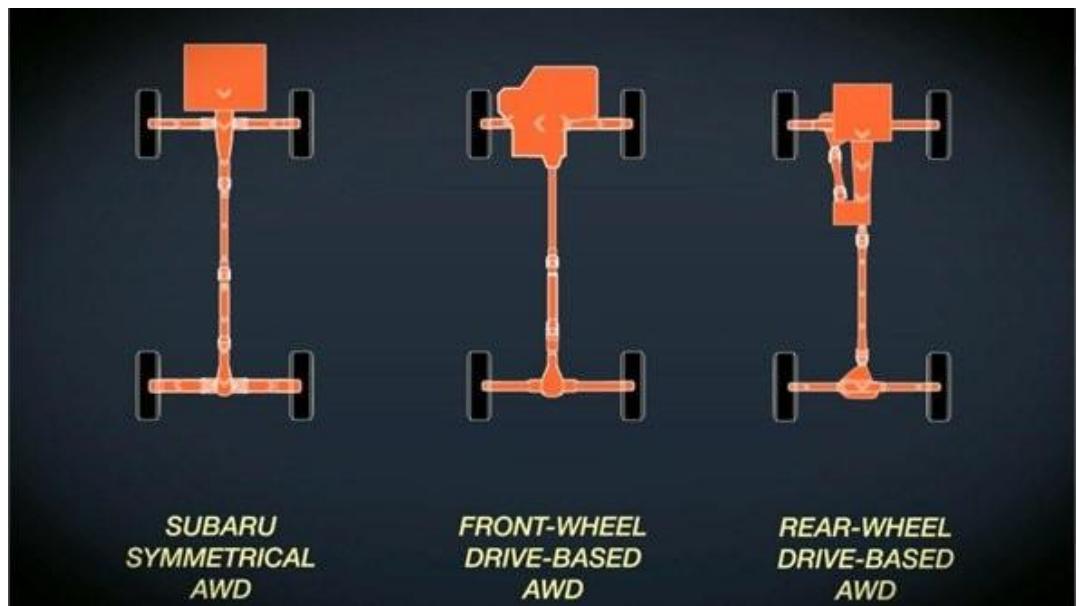
- Hệ thống dẫn động 4 bánh toàn thời gian đối xứng (4WD) [9]:
 - Hệ thống dẫn động được cấu thành từ 3 thành phần: hệ truyền động, động cơ Boxer và hộp số, tất cả đều được bố trí một cách cân bằng và đối xứng qua trục dọc ở giữa thân xe



Hình 2.8. Minh họa hệ thống dẫn động 4 bánh

- Sự kết hợp của cấu trúc truyền động đối xứng, động cơ Boxer dạng phẳng cùng với hộp số đặt theo trực dọc ở giữa mang đến sự phân bố trọng lượng hoàn hảo với trọng tâm của xe được hạn thấp tối đa. Về mặt vật lý, những điều này mang lại sự cân bằng tối ưu và độ đầm chắc mượt mà cho chiếc xe khi vận hành ở tốc độ cao.
- Về mặt sức kéo: Hệ thống sẽ truyền sức kéo liên tục đến cả 4 bánh cùng một lúc, do đó chiếc xe sẽ có được lực kéo và khả năng tăng tốc tối ưu. Trong những điều kiện địa hình phức tạp hoặc có 1 bánh bị trượt, hệ thống sẽ tự động tính toán là ngắt mô-men xoắn ở bánh đó và truyền lực đến những bánh có độ bám tốt hơn, giúp xe không bị mất lái và vượt qua một cách nhanh chóng, an toàn.

- Về mặt hiệu suất và độ bền: Hệ thống được bố trí thẳng hàng và đối xứng qua trục giữa một cách tối ưu. Nhờ vậy, nó có thể truyền lực kéo từ động cơ một cách liên tục và mượt mà đến cả 4 bánh xe. Do phải dùng động cơ với kiểu sắp xếp xy-lanh dạng chữ I hoặc chữ V, các hệ thống dẫn động 4 bánh AWD của những hãng khác phải có thêm nhiều thành phần dẫn động khác, tất nhiên là nó sẽ phức tạp hơn, trọng lượng nặng hơn và chi phí bảo trì cao hơn



Hình 2.9. Các loại hệ thống truyền động 4 bánh

➤ Bộ điều khiển ESC (Electronic Speed Control) [10]

- Trong ESC, bộ phận giới hạn điện áp cho pin lipo có tên là LVC (Low Voltage Cut-Off)
- BEC: BEC là bộ nguồn ổn áp tạo điện áp làm việc (5 volt) cho vi điều khiển nằm trong ESC và cung cấp luôn cho RX cũng như các servo... BEC đi kèm ESC thường là loại ổn áp bù nối tiếp nên rất nóng (điện áp pin càng cao thì càng nóng) và rất hao pin, vì vậy thường người ta ko dùng BEC trong ESC mà dùng BEC rời gọi là UBEC

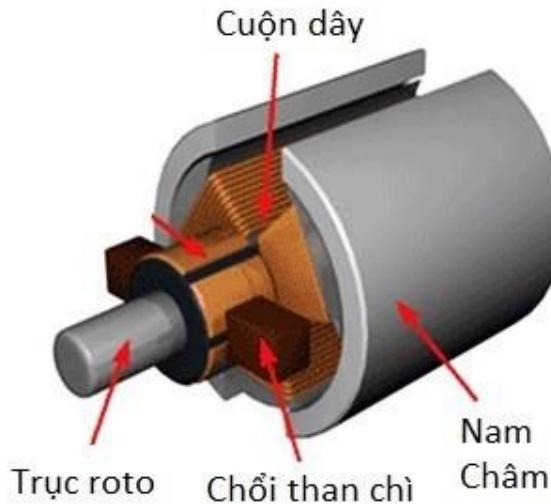
- BEC: là nguồn ôn áp dạng đóng ngắt, sử dụng PWM (kỹ thuật điều biến bìe rộng xung) để biến đổi điện thế DC to DC (1 chiều sang 1 chiều). Thường thì UBEC sử dụng dạng mạch step down để hạ áp (thường điện áp pin lớn hơn điện áp làm việc).
- PWM: kỹ thuật điều biến bìe rộng xung, đây là phần nguyên lý của ESC, kỹ thuật này giúp ta điều chỉnh công suất cấp cho động cơ thông qua độ rộng xung. Giả dụ ta gắn 1 biến trở để điều khiển vận tốc quay của động cơ (như quạt máy chẳng hạn), phương pháp này có ưu điểm là dễ làm, nhưng mà biến trở sẽ tiêu hao 1 phần năng lượng và biến thành nhiệt năng. Ta thay biến trở bằng 1 nút nhấn, và bắt đầu nhấn, ta nhấn nhanh thì động cơ quay mượt, thời gian nhấn lâu hơn thời gian nhả thì động cơ quay nhanh. Đó là ý tưởng về PWM, ta thay 1 nút nhấn bằng 1 IC có khả năng phát xung và có thể điều chỉnh được 2 đại lượng (tần số và độ rộng xung). IC này ngày nay thường dùng các loại vi điều khiển mà điển hình là ATmega8 với khả năng điều chế PWM vượt trội và giá thành rẻ. Do đó, khi nói đến PWM trên ESC ta có thể tham khảo về mặt tần số với đơn vị là Hz (thường từ 6k-8kHz)
- FET: transistor trường ứng, đây là khói công suất của mạch ESC, xung do vi điều khiển điều chế và phát ra không có khả năng cấp dòng cho động cơ vì đơn giản nó chỉ là tín hiệu PWM, vì vậy cần có các FET cấp dòng cho động cơ với sự điều khiển bằng xung PWM. Khả năng chịu dòng của FET cũng là khả năng chịu dòng của ESC



Hình 2.10. ESC của xe RGT Rock Hammer

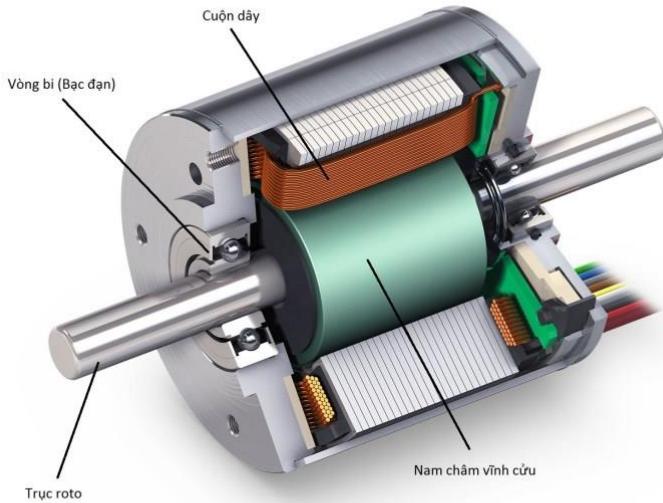
➤ Động cơ (có/không) chổi than [11]

- Chổi than là một vật liệu dẫn điện làm từ carbon có tác dụng tiếp điện, duy trì kết nối điện giữa bộ phận tĩnh và các phần chuyển động của động cơ điện DC hoặc AC được sử dụng trong công nghiệp sản xuất sử dụng động cơ dây quấn



Hình 2.11. Động cơ có chổi than

- Động cơ chổi than là loại mô-tơ dùng chổi than chì. Động cơ chổi than sử dụng trong xe điện RC là loại Mô-tơ dùng điện một pha, gồm có hai dây đỏ và đen. Ưu điểm của Mô-tơ chổi than là giá xe điều khiển từ xa sản xuất thấp cho nên chúng ta có thể thấy những chiếc xe điện RC giá thành thấp có gắn động cơ chổi than.
- Nhược điểm của loại động cơ chổi than được làm bằng chổi than chì nên sau một khoảng thời gian sử dụng sẽ làm mòn chổi than, tuổi thọ kém, mô-tơ tiêu thụ điện lớn, công suất yếu hơn các loại mô-tơ không chổi than có cùng kích cỡ. Vì vậy những chiếc xe mô hình RC chạy loại mô-tơ này thường không có tốc độ cao.
- Nhược điểm của động cơ chổi than khá lớn và gây bất tiện, vì vậy việc sáng tạo ra động cơ không chổi than là một bước đột phá lớn, như việc sáng tạo ra pin Li-Po.



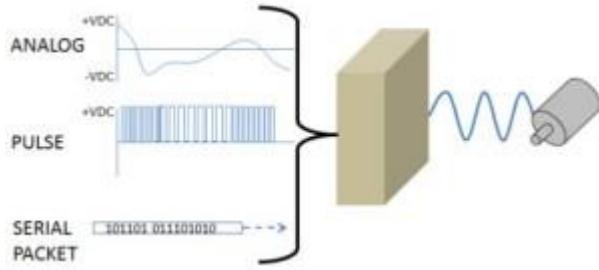
Hình 2.12. Động cơ không chổi than

- Động cơ DC không chổi than có các ưu điểm của động cơ đồng bộ nam châm vĩnh cửu như: tỷ lệ momen/quán tính lớn, tỷ lệ công suất trên khối lượng cao.
- Do máy được kích từ bằng nam châm vĩnh cửu nên trên rotor hiệu suất động cơ cao hơn.
- Động cơ kích từ nam châm vĩnh cửu không cần chổi than và vành trượt nên không tốn chi phí bảo trì chổi than. Ta cũng có thể thay đổi đặc tính động cơ bằng cách thay đổi đặc tính của nam châm kích từ và cách bố trí nam châm trên rotor.
- Một số đặc tính nổi bật của động cơ không chổi than khi hoạt động:
 - Tỷ lệ công suất/khối lượng máy điện cao.
 - Tỷ lệ momen/quán tính lớn (có thể tăng tốc nhanh).
 - Vận hành nhẹ nhàng (dao động của momen nhỏ) thậm chí ở tốc độ thấp (để đạt được điều khiển vị trí một cách chính xác).

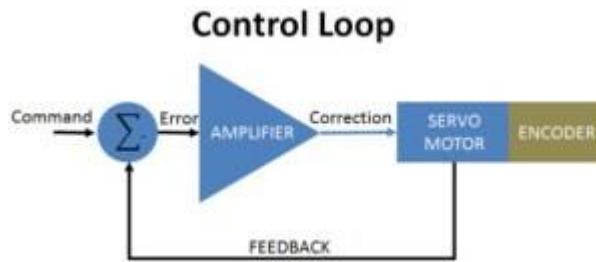
- Mômen điều khiển được ở vị trí bằng không.
- Vận hành ở tốc độ cao.
- Có thể tăng tốc và giảm tốc trong thời gian ngắn.
- Hiệu suất cao.
- Kết cấu gọn.

➤ Động cơ Servo [12]

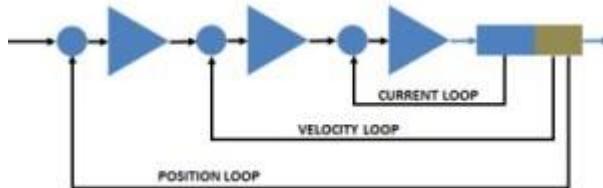
- Động cơ Servo là một bộ phận của hệ thống điều khiển chuyển động của máy móc. Một trong các bộ phận không thể thiếu giúp Động cơ Servo có thể hoạt động đó chính là Driver servo. Tương tự như driver của máy tính. Động cơ Servo cung cấp lực chuyển động cần thiết cho các thiết bị máy móc khi vận hành
- Ở ngành công nghiệp, đa số các động cơ Servo sử dụng động cơ một chiều không chổi than.
- Rotor của động cơ là một nam châm vĩnh cửu có từ trường mạnh và Stator của động cơ được cuốn các cuộn dây riêng biệt, được cấp nguồn theo một trình tự thích hợp để quay rotor.
- Nếu thời điểm và dòng điện cấp tới các cuộn dây là chuẩn xác thì chuyển động quay của rotor phụ thuộc vào tần số và pha, phân cực và dòng điện chạy trong cuộn dây stator.
- Động cơ servo được hình thành bởi những hệ thống hồi tiếp vòng kín. Tín hiệu ra của động cơ được nối với một mạch điều khiển. Khi động cơ vận hành thì vận tốc và vị trí sẽ được hồi tiếp về mạch điều khiển này. Khi đó bất kỳ lý do nào ngăn cản chuyển động quay của động cơ, cơ cấu hồi tiếp sẽ nhận thấy tín hiệu ra chưa đạt được vị trí mong muốn. Mạch điều khiển tiếp tục chỉnh sai lệch cho động cơ đạt được điểm chính xác nhất.



Hình 2.13. Tín hiệu điều khiển động cơ servo



Hình 2.14. Phản hồi hệ thống động cơ



Hình 2.15. Mạch vòng điều khiển động cơ servo

➤ Pin Lipo [13]

- Pin Lipo (viết tắt từ Lithium Polymer) là một loại pin sạc sử dụng chất điện phân polymer khô. Sự ra đời của pin Lipo là một trong những yếu tố khiến cho mô hình điện phát triển nhanh chóng, đặc biệt là mô hình máy bay.
- Những ưu điểm chính khiên Pin LiPo được dân chơi mô hình ưu tiên lựa chọn so với các loại pin sạc khác (NiCad, NiMH) là:
 - Pin RC LiPo nhỏ, nhẹ và có thể làm ở mọi hình dáng kích thước.
 - Pin RC LiPo có dung lượng cao trong khi kích thước, khối lượng nhỏ hơn các loại pin khác.

- Pin RC LiPo có dòng xả cao đảm bảo đủ cung cấp năng lượng cho các động cơ có công suất cao.
- Tóm lại, pin Lipo có tỉ lệ năng lượng lưu trữ/đơn vị khối lượng cao và có thể chế tạo ở kích thước, hình dạng đa dạng phù hợp với đa số mô hình RC.
- Tuy nhiên, pin LIPO cũng có một vài nhược điểm so với các loại pin khác:
 - Pin LiPo vẫn còn đắt tiền so với pin NiCad và NiMH. Tuy nhiên, với sự phát triển của công nghệ, giá thành pin Lipo sẽ giảm dần.
 - Tuổi thọ Pin LiPo không cao, chỉ khoảng 300-400 lần sạc (và sẽ thấp hơn nhiều nếu không được chăm sóc đúng cách). Tuy nhiên nếu sử dụng đúng cách, tuổi thọ pin Lipo có thể lên tới 1000 lần sạc/xả.
 - Pin Lipo dễ bắt lửa và cháy nổ nếu không bảo quản, sử dụng đúng cách
 - Pin LiPo yêu cầu cao về việc tuân thủ các quy tắc khi sử dụng, khai thác để đảm bảo tuổi thọ và an toàn.
- Thông số kỹ thuật pin lipo:



Hình 2.16. Pin Lipo

- Các thông số quan trọng cần biết khi mua pin Lipo gồm có: Điện áp, dung lượng, dòng xả.
- **Điện áp:**

- Một cục pin Lipo được ghép (song song hoặc nối tiếp) từ một hay nhiều cell pin. Mỗi cell pin lipo có điện áp 3,7V (khác với cell pin NiCad&NiMH thường có điện áp 1,2V). Tùy theo mô hình lựa chọn cần điện áp lớn hay nhỏ sẽ dùng pin được ghép bởi 1 cell, 2 cell hay nhiều hơn. Để tăng điện áp của pin lipo, các cell pin được ghép nối tiếp với nhau và được kí hiệu bởi chữ S. Ví dụ:
 - Pin 1 cell (1S): Điện áp 3,7V
 - Pin 2 cell (2S): Điện áp 7,4V
 - Pin 3 cell (3S): Điện áp 11,1V
 - Pin 4 cell (4S): Điện áp 14,8V
 - Pin 5 cell (5S): Điện áp 18,5V
 - Pin 6 cell (6S): Điện áp 22,2V
 - Pin 8 cell (8S): Điện áp 29,6V
 - Pin 10 cell (10S): Điện áp 37V
 - Pin 12 cell (12S): Điện áp 44,4V
 - Để tăng dung lượng pin mà vẫn giữ nguyên điện áp, các cell pin sẽ được mắc song song và được kí hiệu bởi chữ P. Ví dụ pin lipo có kí hiệu 2S2P gồm 2 cục pin 2S được mắc song song với nhau.
- Trên máy bay mô hình, thường dùng động cơ 3 pha không chổi quét. Động cơ này có thông số quan trọng là Kv, tương ứng với số vòng quay được ứng với 1V điện áp. Ví dụ: Động cơ 1000kv trong dải từ 10V đến 25V nghĩa là động cơ này sẽ quay với vận tốc 10.000 vòng/phút khi điện áp pin là 10V

và lên đến 25.000 vòng/phút khi điện áp là 25V. Chính vì vậy, điện áp pin rất quan trọng khi chế tạo máy bay mô hình.

- Dung lượng:

- Dung lượng pin (Capacity – C) là lượng năng lượng lưu trữ trong pin khi nạp đầy, có đơn vị là mAh (mini ampe giờ). Dung lượng pin tương ứng với dòng tối đa mà pin xả (tiêu hao, tính theo mA) để pin chạy được 1 giờ. Ví dụ, một viên LiPo có dung lượng 1000mAh sẽ được xả hoàn toàn trong một giờ (dùng trong 1 giờ) với dòng tải 1000mA trên nó. Vẫn cùng viên pin này nhưng với dòng tải 500mAh thì dùng được khoảng 2 giờ, nhưng nếu tăng dòng tải lên 15000 mA thì thời gian để tiêu hao pin sẽ chỉ được khoảng 4 phút.
- Như vậy, cùng một mô hình, dung lượng pin sẽ quyết định thời gian chạy trước khi hết pin. Đối với máy bay mô hình, dung lượng pin sẽ không tỉ lệ thuận với thời gian bay vì pin dung lượng lớn thì khói lượng pin sẽ lớn và dòng tiêu thụ sẽ cao hơn. Vì vậy cân đối của việc tăng dung lượng pin để tăng thời gian bay của mô hình.

- Dòng xả:

- Thông số quan trọng thứ 3 là dòng xả của pin. Dòng xả hiểu đơn giản là một viên pin có khả năng cung cấp dòng lớn một cách an toàn như thế nào. Dòng xả liên tục (continuous discharge) an toàn trên pin lipo được tính theo bội số của dung lượng pin (C). Một viên pin với một dòng xả 10C có nghĩa là bạn có thể xả một cách an toàn với tốc độ gấp 10 lần so với dung lượng của viên pin đó.

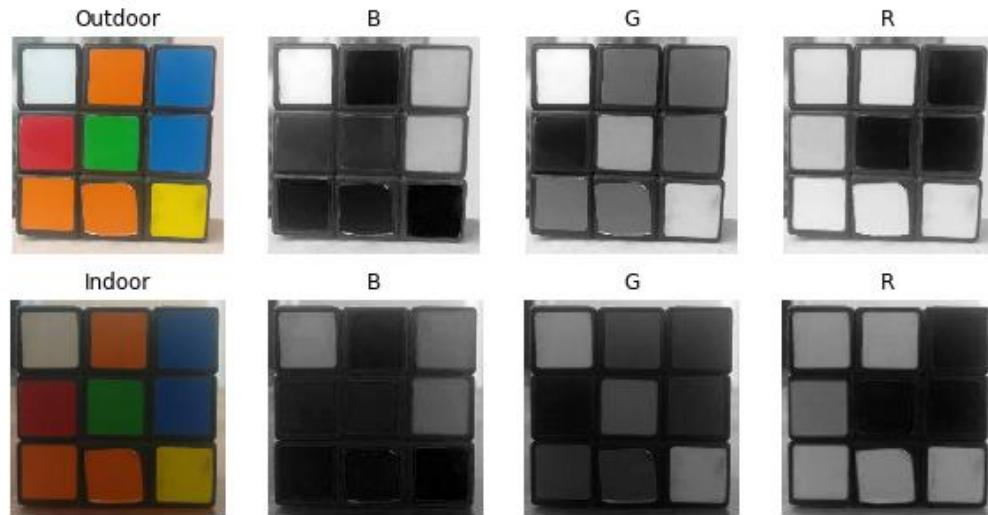
- Ví dụ: Pin lipo có dung lượng 2000mAh; dòng xả 20C tức là dòng xả liên tục an toàn của pin là $20 \times 2000mA = 40000mA = 40A$.
- Ngoài dòng xả liên tục (Continuous Discharge Rate), pin có thể xả với dòng cao hơn trong một thời gian rất ngắn (khoảng vài giây) để gia tăng công suất (tiếng anh là Burst Discharge).
- Thông thường, Pin có dòng xả càng cao thì càng đắt. Vì vậy, nếu bạn không có nhiều tiền, không nên chọn pin có dòng xả quá cao so với nhu cầu sử dụng.
- Vậy làm sao để bạn biết dòng xả bao nhiêu là phù hợp khi mua pin RC Lipo? Câu trả lời dễ dàng nhất sẽ là chọn pin có dòng xả C lớn nhất mà bạn có thể ... Nếu tiền không phải là vấn đề. Tuy nhiên, nếu bạn mới chơi, nên chọn pin có dòng xả vừa phải. Để chọn dòng xả pin, bạn phải ước lượng xem, động cơ bạn sử dụng khi chạy với 100% ga thì dòng tiêu thụ là bao nhiêu. Ví dụ là 30A thì bạn nên chọn pin có dòng xả liên tục <40A (ví dụ pin 2000mAh thì chọn dòng xả <20C).

2.6 Tìm hiểu thuật toán tìm đường đi sử dụng thư viện OpenCV

2.6.1 Nhận diện màu sắc

2.6.1.1 Hệ màu [14]

Hệ màu RGB: Màu RGB thu được từ sự kết hợp tuyến tính (Linear combination) của 3 giá trị Red, Green và Blue. Ba kênh màu là sự tương quan của lượng ánh sáng chiếu vào bề mặt.



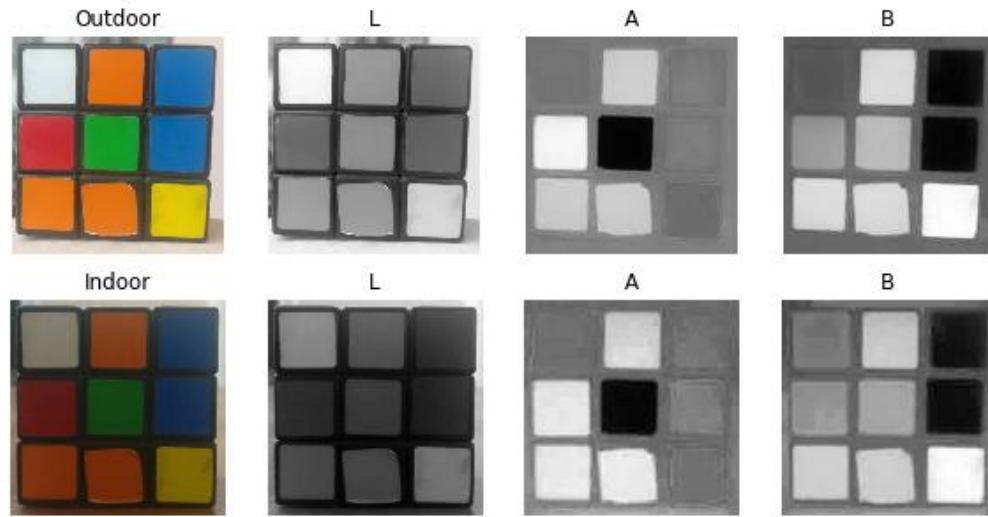
Hình 2.17. Phân biệt giữa 3 kênh R, G, B

Ta rút ra được những đặc điểm từ hệ màu RGB như sau:

- Không đồng nhất đáng kể giữa 3 kênh
- Trộn lẫn giữa sắc độ và độ chói

Hệ màu LAB: Gồm 3 thành phần L – Lightness (Intensity), a – thành phần màu từ xanh lá → đỏ tươi (Green to Magenta), b – thành phần màu từ xanh dương → vàng (Blue to Yellow).

Khác với hệ RGB – 3 kênh riêng biệt được mã hóa dựa trên cường độ sáng. Còn với LAB, L được tách biệt là mã hóa về độ sáng và 2 kênh còn lại dùng cho mã hóa màu.

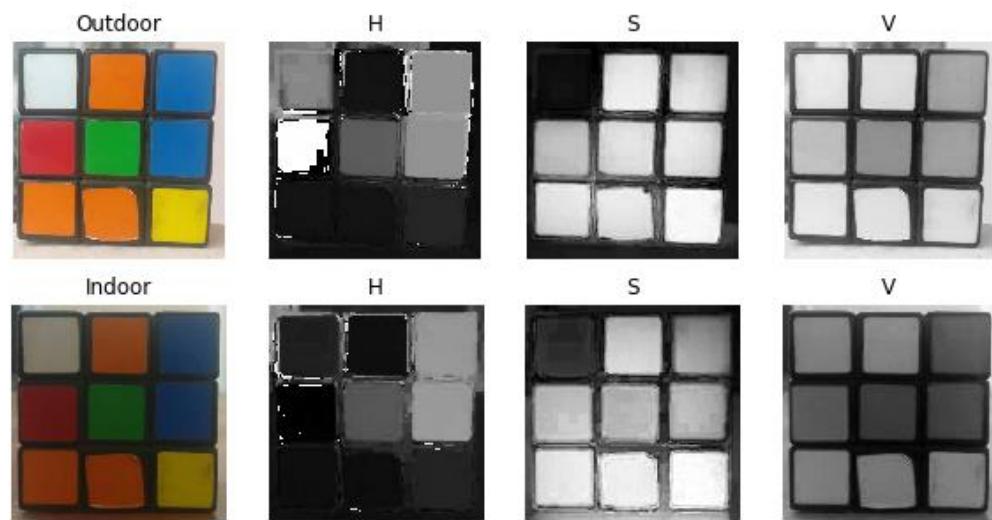


Hình 2.18. Phân biệt giữa 3 kênh L, A, B

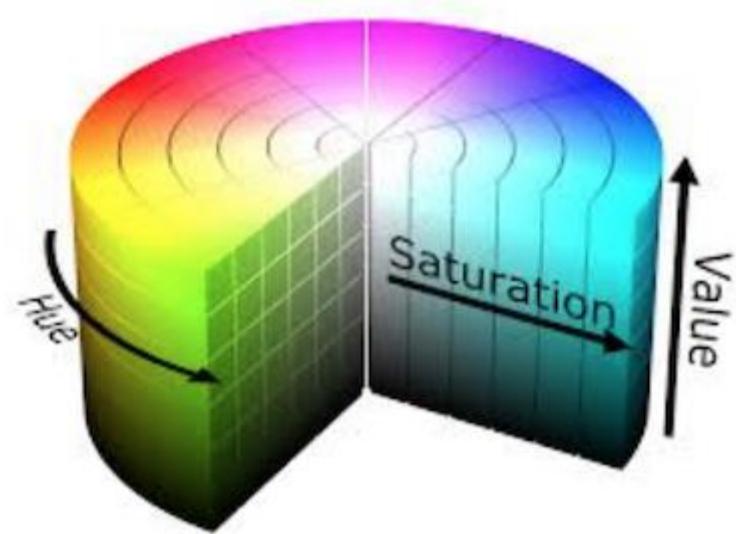
Ta rút ra những đặc điểm từ hệ màu LAB:

- Không gian màu gần gũi và khá giống với cách nhận diện của con người
- Khi hiển thị và chụp ảnh sẽ độc lập với thiết bị
- Biến đổi được qua hệ màu RGB qua một phương trình biến đổi phức tạp

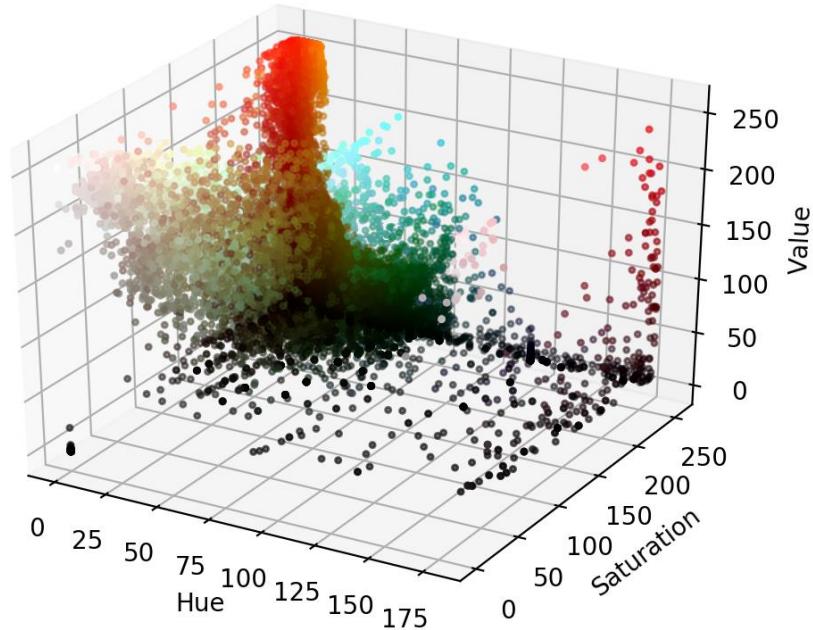
Hệ màu HSV: Có 3 thành phần: H – Hue (Bước sóng cực đại), S – Saturation (Độ tinh khiết, sắc thái màu sắc), V – Value (Cường độ). Chỉ sử dụng 1 kênh duy nhất để biểu diễn màu là Hue.



Hình 2.19. Phân biệt giữa 3 kênh H, S, V



Hình 2.20. Vòng tròn hệ màu HSV



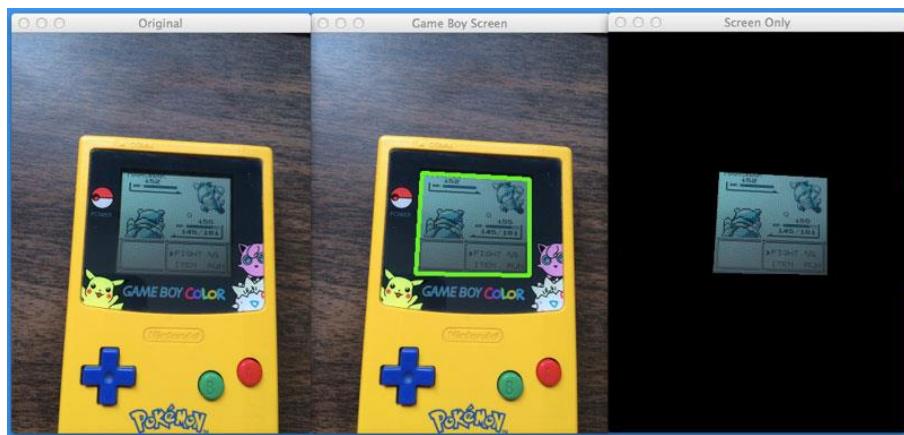
Hình 2.21. Không gian 3 chiều của không gian màu HSV

2.6.2 Lọc màu đường và biển báo

2.6.3 Thuật toán tìm góc lệch sử dụng contour

2.6.3.1 Crop image

- Dùng để lọc nhiễu: bỏ những khúc phía trên, có xen lẫn bầu trời, cây cối, ...
- Dùng để tìm những feature khác nhau: khi lấy phần gần với xe nhất và nhỏ thì sẽ tạo ra những feature khác biệt nhau để dễ dàng tính toán hơn



Hình 2.22. Crop ảnh [14]

2.6.3.2 Tìm contour

- Chuyển ảnh xám
- Làm mờ ảnh để làm mượt những đường viền
- Chuyển về ảnh nhị phân bằng thuật toán threshold
- Tìm contour



Hình 2.23. Ví dụ về contour trong Opencv [16]

- Tính toán góc bẻ lái bằng hàm arctan

2.6.4 Thuật toán tìm góc lệch sử dụng sliding window

2.6.4.1 Cân chỉnh camera [17]

Những chiếc camera giá rẻ trên thị trường sẽ cho ra những hình ảnh bị méo mó, biến dạng. Có 2 loại biến dạng chính: radial distortion và tangential distortion

Với radial distortion, những đường vẽ thẳng nhưng trên hình thực thì bị cong



Hình 2.24. Ví dụ về radial distortion

Để giải quyết vấn đề này, ta biến đổi qua công thức:

$$x_{corrected} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$y_{corrected} = y(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

Công thức 2.1. Cân chỉnh camera với radial distortion

Đối với tangential ta có công thức biến đổi sau:

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Công thức 2.2. Cân chỉnh camera với tangential

Ta cần tìm 5 biến (hệ số biến dạng):

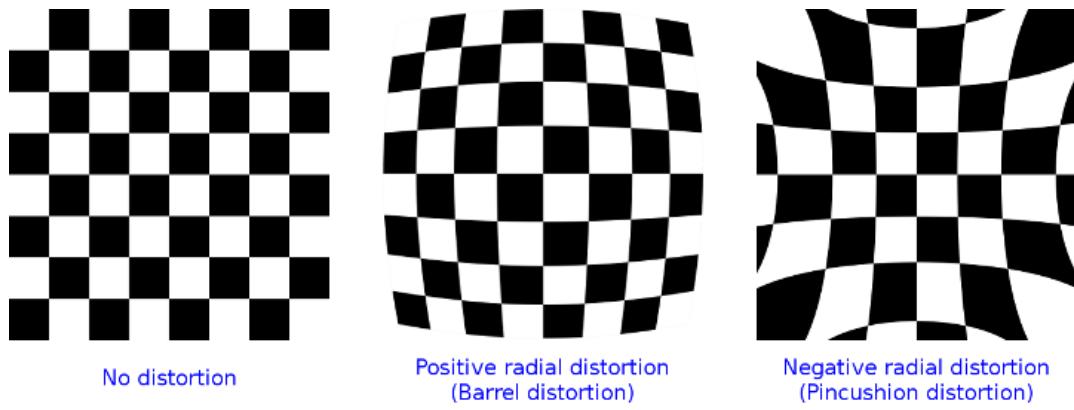
$$Distortion coefficients = (k_1 \ k_2 \ p_1 \ p_2 \ k_3)$$

Công thức 2.3. Hệ số biến dạng

Ngoài ra những biến ở các công thức trên được tìm và lưu sử dụng cho từng camera là 1 ma trận 3x3:

$$camera matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Công thức 2.4. Ma trận camera 3x3



Hình 2.25. Cân chỉnh frame ảnh



Hình 2.26. Kết quả sau khi cân chỉnh

2.6.4.2 Bé thăng frame ảnh

- Hàm getPerspectiveTransform: [18] hàm tính ra ma trận 3×3 của một ma trận biến đổi

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \text{map_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Với $\text{dst}(i) = (x'_i, y'_i)$, $\text{src}(i) = (x_i, y_i)$, $i = 0, 1, 2, 3$

Công thức 2.5. Hàm getPerspectiveTransform

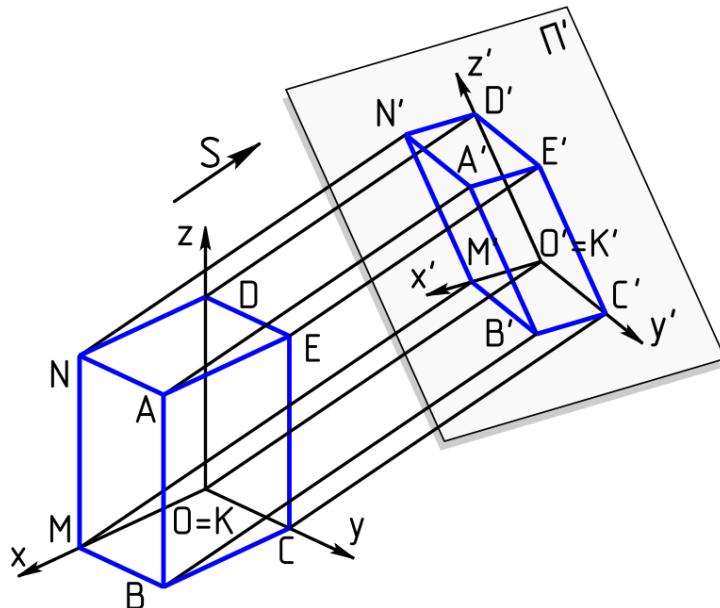
- Hàm warpPerspective: hàm áp dụng biến đổi perspective lên ảnh lấy được từ hàm getPerspectiveTransform.

Công thức sử dụng để biến đổi:

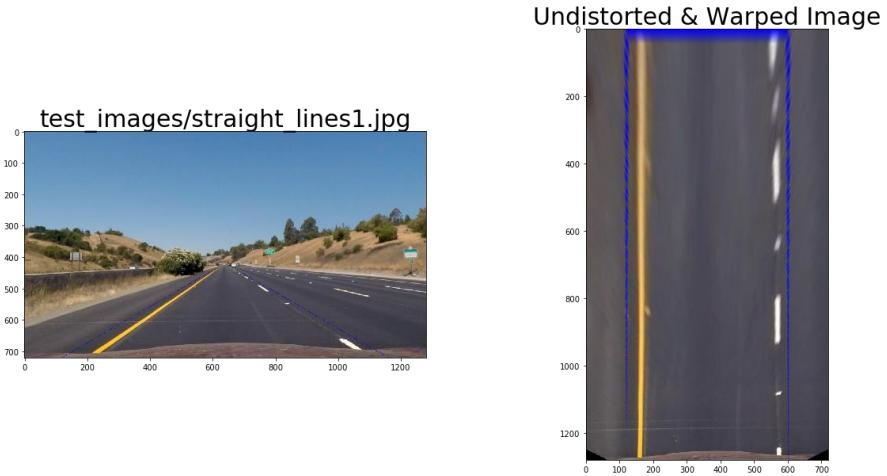
$$dst(x, y) = \text{src} \left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}} \right)$$

Công thức 2.6. Hàm warpPerspective

- Sử dụng 2 hàm này, ta được một bức ảnh 2 lines đường đã bẻ thẳng, còn được gọi là birdview
- Birdview là tầm nhìn vật thể từ trên cao nhìn xuống như cách nhìn của loài chim, thường được dùng để vẽ bản thiết kế, thi công, bản đồ.



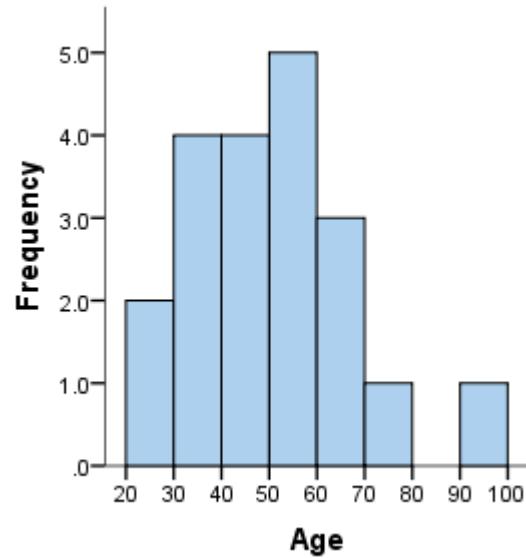
Hình 2.27. Mô hình về birdview



Hình 2.28. Birdview với lane đường

2.6.4.3 Biểu đồ những điểm thuộc làn đường

- Histogram là biểu đồ biểu diễn phân bố các điểm thỏa 1 yêu cầu với 1 lượng dữ liệu liên tục theo thời gian

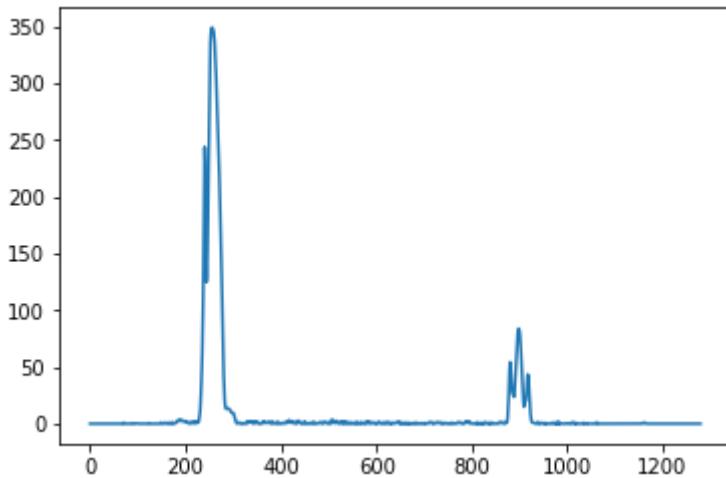


Hình 2.29. Ví dụ về histogram [19]

- Biểu đồ phân bố tần số (Biểu đồ phân bố mật độ, biểu đồ cột) dùng để đo tần số xuất hiện của một vấn đề nào đó, cho ta thấy rõ hình ảnh sự thay đổi, biến

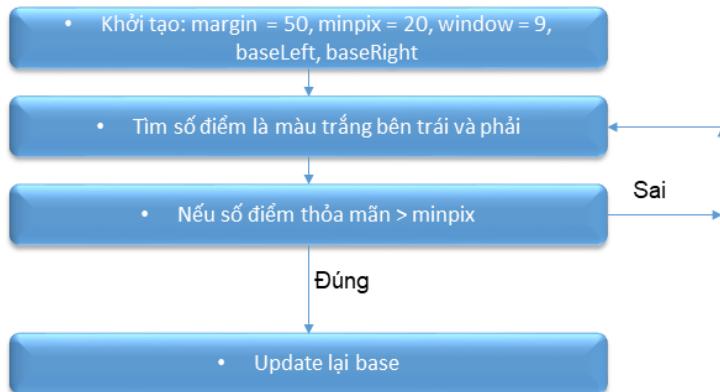
động của một tập dữ liệu. Đây là một khái niệm rất phổ biến, được sử dụng rộng rãi từ lĩnh vực kỹ thuật tới kinh tế.

- Biểu đồ phân bố tần số (Biểu đồ phân bố mật độ, biểu đồ cột) dùng để đo tần số xuất hiện của một vấn đề nào đó, cho ta thấy rõ hình ảnh sự thay đổi, biến động của một tập dữ liệu. Đây là một khái niệm rất phổ biến, được sử dụng rộng rãi từ lĩnh vực kỹ thuật tới kinh tế.
- Biểu đồ phân bố tần số (Biểu đồ phân bố mật độ, biểu đồ cột) dùng để đo tần số xuất hiện của một vấn đề nào đó, cho ta thấy rõ hình ảnh sự thay đổi, biến động của một tập dữ liệu. Đây là một khái niệm rất phổ biến, được sử dụng rộng rãi từ lĩnh vực kỹ thuật tới kinh tế.
- Trục tung (Oy) biểu diễn số lượng điểm ảnh (Pixel) của mức xám.
- Trục hoành (Ox) biểu diễn mức xám.
- Giá trị lớn nhất của trục hoành chính là số lượng điểm ảnh (Pixel) có trong một bức ảnh.
- Với ảnh màu như RGB thì có tới 3 biểu đồ Histogram thể hiện từng kênh màu.
- Một biểu đồ tốt à biểu đồ có số lượng điểm ảnh nhiều nhất ở vùng giữa (Độ sáng trung bình) và ít dần ra 2 vùng sáng tối (Ngọn núi).
- Dựa vào biểu đồ Histogram mà bạn có thể biết được hình ảnh sáng tối như thế nào.
- Áp dụng cho các xử lý ảnh cao cấp khác.
- Ảnh tối là ảnh có tập trung quá nhiều điểm ảnh bên vùng tối.
- Ảnh sáng là ảnh có tập trung quá nhiều điểm ảnh bên vùng sáng.
- Ảnh có độ tương phản cao là ảnh có điểm ảnh tập trung nhiều ở 2 vùng sáng và tối và tập trung ít ở vùng giữa.
- Ảnh có độ tương phản thấp là ảnh có điểm ảnh tập trung nhiều ở vùng giữa và tập trung rất ít ở vùng hai vùng sáng và tối.



Hình 2.30. Histogram của lane đường

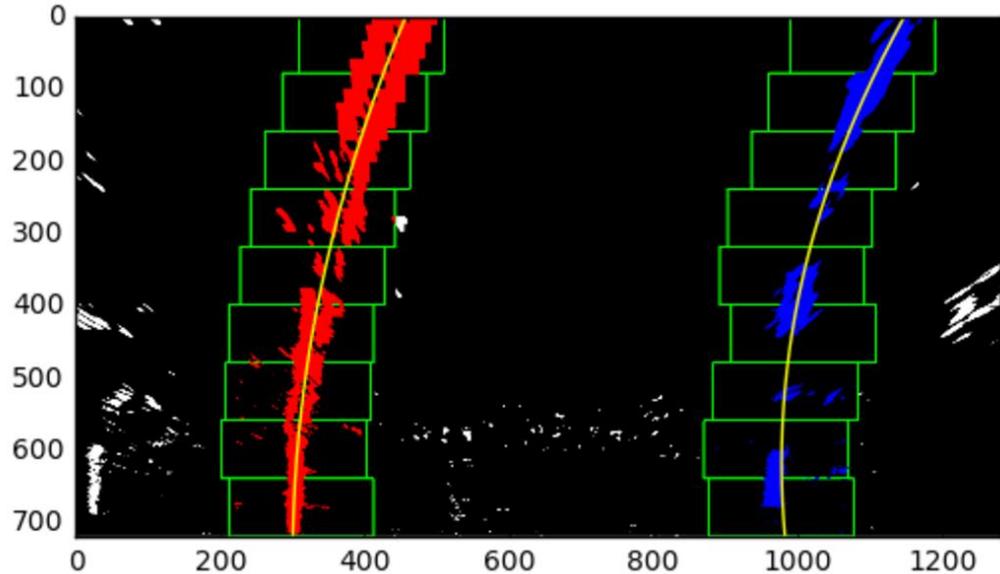
2.6.4.4 Chia khung làn đường (Sliding window)



Hình 2.31. Flowchart sliding window

- Chọn margin = 50: một window sẽ rộng 50 pixels
- Chọn 9 window
- Đầu tiên ta chọn điểm gốc của làn trái và làn phải bằng cách lấy max của histogram
- Đếm số điểm màu trắng (tương ứng với phần tử lane đường) với chiều rộng của khung bằng margin và tổng hợp vào một mảng.

- Nếu mật độ của những điểm màu trắng vượt mức minpix thì cập nhật lại điểm gốc
- Qua những khung đó, ta vẽ được đường cong



Hình 2.32. Sliding window

2.6.4.5 Phương trình đường cong

- Hàm numpy.polyfit: Tổng hợp 1 mảng điểm thành một đa thức với công thức: $p(x) = p[0] * x^{**deg} + \dots + p[deg]$ với góc tìm được từ những điểm (x, y) \Rightarrow Hàm trả về 1 vector hệ số p . Từ đó ta vẽ được đường cong qua phương trình

$$E = \sum_{j=0}^k |p(x_j) - y_j|^2$$

Công thức 2.7. Hàm numpy.polyfit

- Phân tích ta được thuật toán trên code như sau:

$$x[0]**n * p[0] + \dots + x[0] * p[n-1] + p[n] = y[0]$$

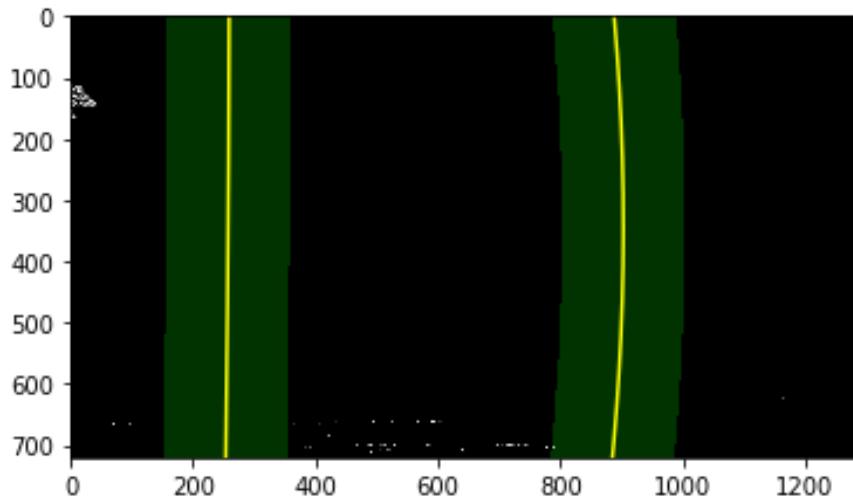
$$x[1]**n * p[0] + \dots + x[1] * p[n-1] + p[n] = y[1]$$

...

$$x[k]**n * p[0] + \dots + x[k] * p[n-1] + p[n] = y[k]$$

Công thức 2.8. Thuật toán hàm numpy.polyfit

- Sau khi tính 2 phương trình ta vẽ được qua hàm fill poly



Hình 2.33. Skip sliding window

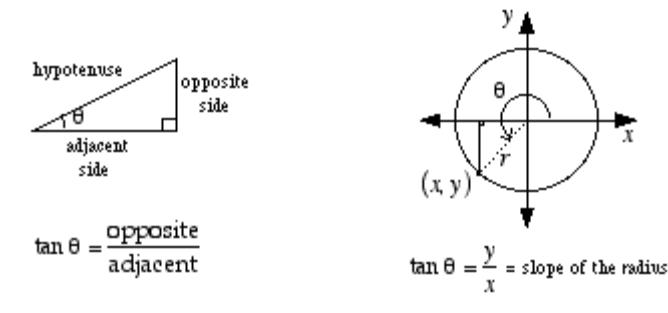
2.6.4.6 Tính toán dự đoán góc bẻ lái

Đo khoảng cách từ camera tới trung bình tâm của 2 bánh sau = 37cm

Đo khoảng cách từ camera tới điểm trên ảnh ở thực tế = 29cm

Tính khoảng cách từ điểm đích tới trung bình tâm của 2 bánh sau theo pixel = (29cm / 150) * (29cm + 37cm) = 341 pixel. 150 ở đây là mức pixel trên ảnh tính từ góc.

Tính góc lệch bằng hàm tan trong lượng giác



Hình 2.34. Tính góc

Cân chỉnh góc bằng một hệ số nhân với hệ số tìm được do thực nghiệm trên sa hình thực tế, ở đây chọn 1.5 để góc chính xác với sa hình thực tế.

2.6.4.7 Tính toán khoảng cách lệch so với 2 bên làn đường

- Sau khi tìm được 2 đường cong, ta tìm độ lệch (curvature) so với 2 làn đường qua công thức:

$$f(y) = Ay^2 + By + C \quad R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

Công thức 2.9. Công thức tính độ lệch (curvature)

```
ground_img_with_projection = birdsEye.project(ground_img, binary, left_fit_curve, right_fit_curve)
```



Hình 2.35. Kết quả tính độ lệch

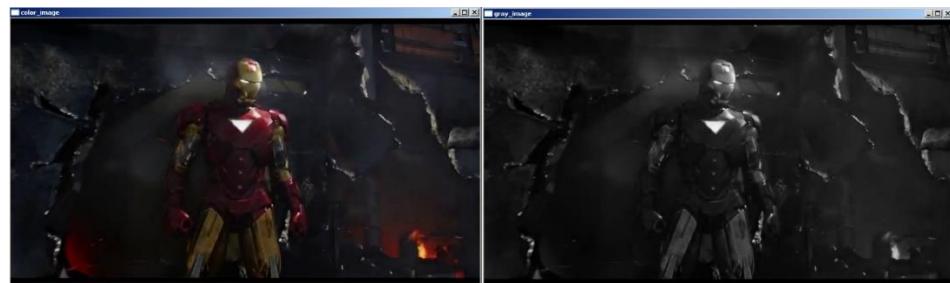
2.6.4.8 Lưu lịch sử cho những frame kế tiếp để dự đoán

- Mỗi lần xử lý 1 frame ảnh, so sánh curvation
- Nếu khác nhau → Lấy kết quả của frame mới để xử lý và cho xe chạy
- Nếu giống nhau → Lấy kết quả của frame cũ để xử lý và cho xe chạy
- Sau mỗi lần tính toán góc lệch, tâm điểm thành công thì lưu vào lịch sử.

2.6.5 Thuật toán tìm biến báo

2.6.5.1 Nhận diện biến báo

- Chuyển màu thành ảnh xám (RGB → GRAY):
 - Công thức: $Y = 0.2126R + 0.7152G + 0.0722B$
Công thức 2.10. Biến đổi ảnh RGB sang GRAY
 - Trong OpenCV dùng hàm: cvtColor(img, grayscale_img, CV_BGR2GRAY);



Hình 2.36. Chuyển ảnh màu thành xám

- Làm mờ ảnh bằng Gaussian
 - Sử dụng hàm cv2.GaussianBlur()
 - Đây là một dạng biến đổi Gaussian với công thức trong không gian 1 chiều:

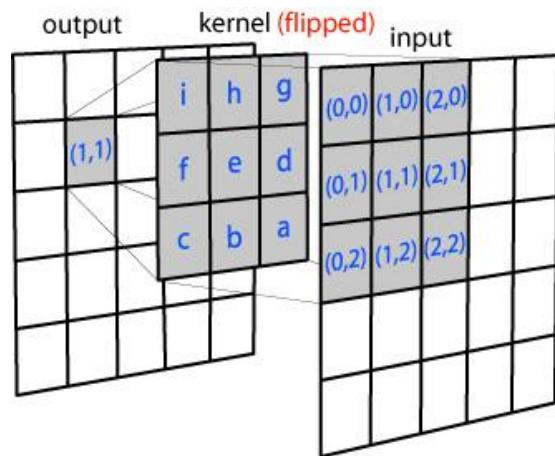
Công thức 2.11. Biến đổi Gaussian trong không gian 1 chiều

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

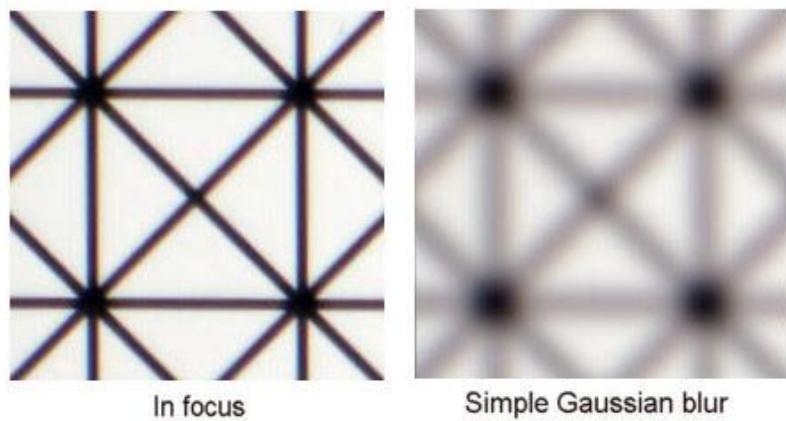
- Trong không gian 2 chiều:

Công thức 2.12. Biến đổi Gaussian trong không gian 2 chiều

- Để dùng Gaussian blur, ta cần một kernel [N x N] và mỗi pixel sẽ nhân với kernel này

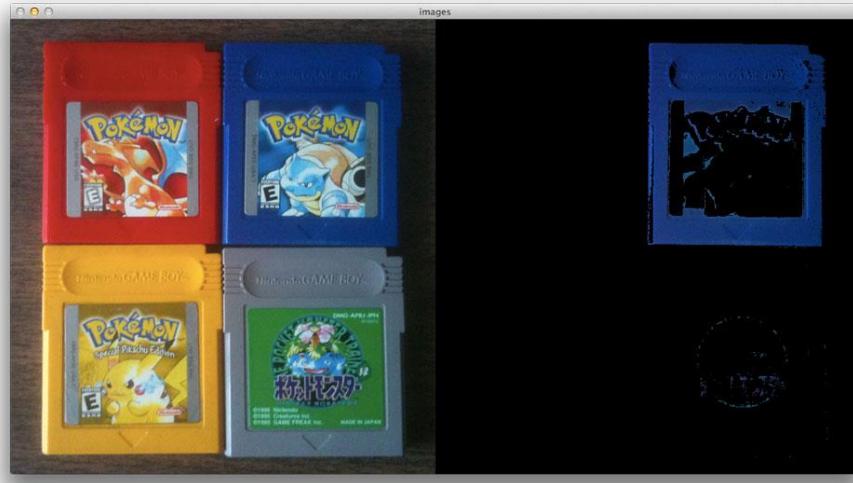


Hình 2.37. Ví dụ giải thích Gaussian blur bằng toán học



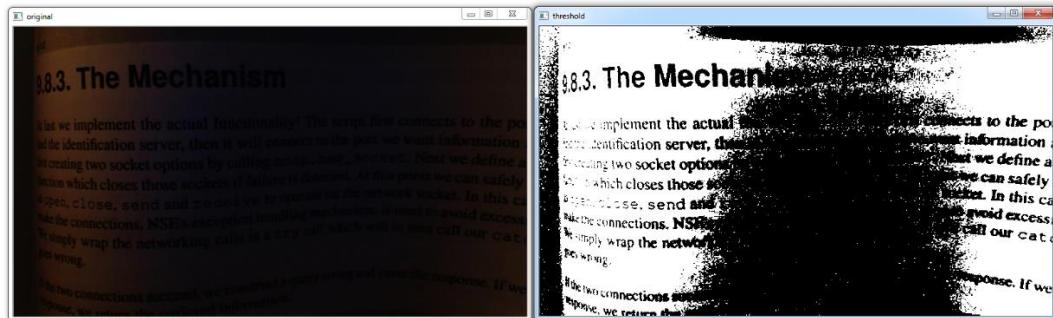
Hình 2.38. Gaussian blur

- Lọc màu xanh: Sử dụng hàm `inrange` của OpenCV trong không gian màu HSV



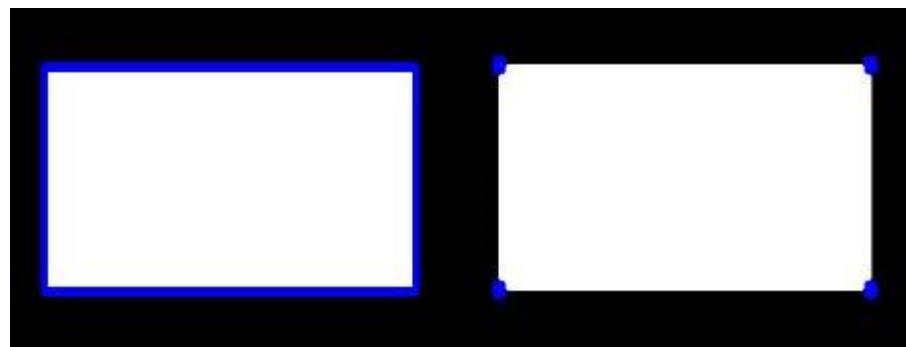
Hình 2.39. Lọc màu xanh

- Lọc trắng đen bằng ngưỡng → Đưa ảnh về dạng nhị phân (0 và 255): sử dụng hàm Threshold của OpenCV



Hình 2.40. Threshold [20]

- Tìm contours bằng hàm find contours:

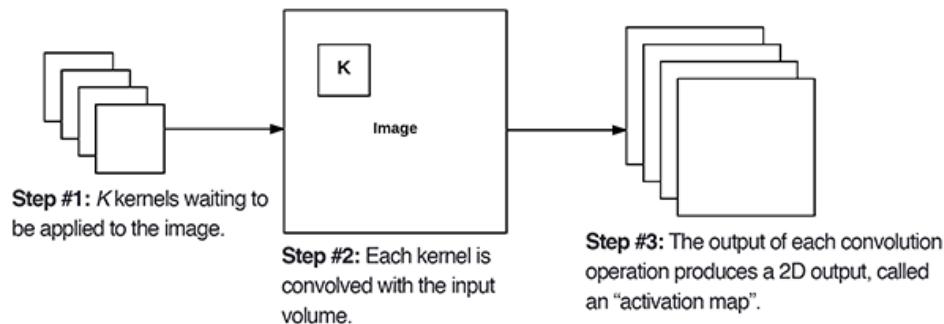


Hình 2.41. Contour

2.6.5.2 Những thành phần trong mạng neural

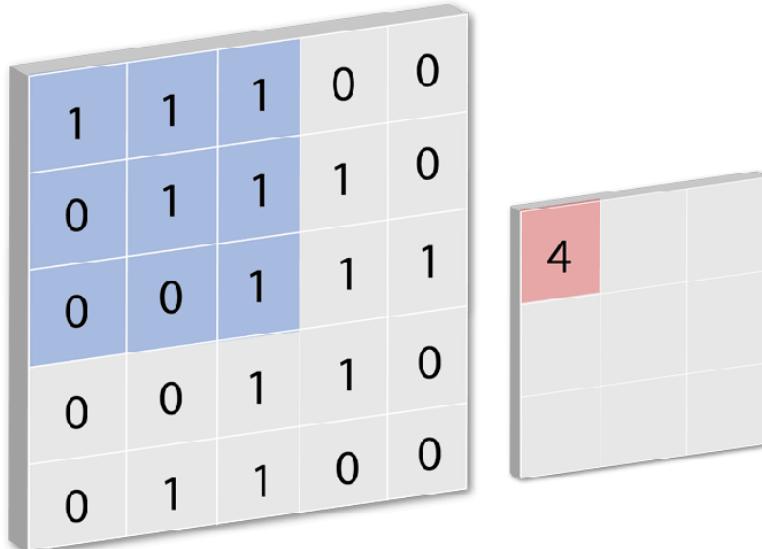
- Conv2D [21]

- o filters



Hình 2.42. Các bước tạo ra map 2D

- o Conv2D dùng để trích xuất những feature có cấp cao hơn bằng cách thay thế cho từng pixel với giá trị tổng hợp từ những pixel trong ma trận NxN cho trước.

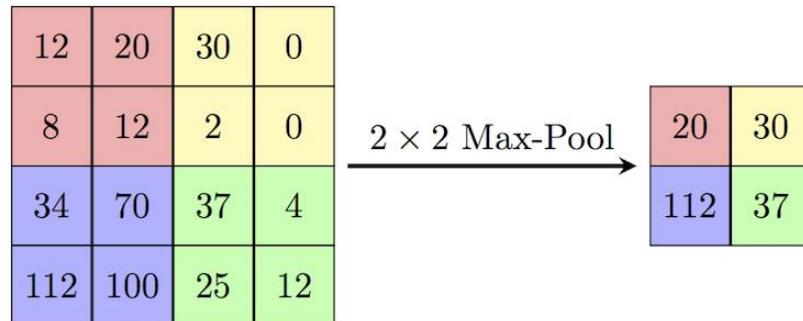


Hình 2.43. Ví dụ về Conv2d [22]

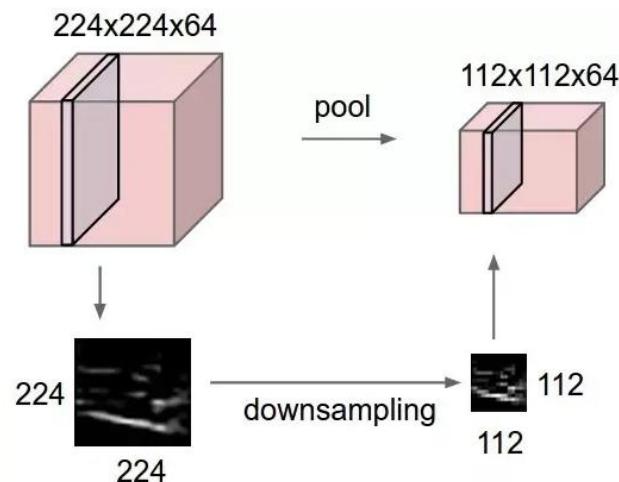
- MaxPooling

- o Quy trình hạ bậc, lấy 1 mẫu làm đại diện cho các phần tử đầu vào

- MaxPooling dùng để giảm thiểu phép tính bằng cách giảm số phần tử trên mảng và thu nhỏ mảng ban đầu



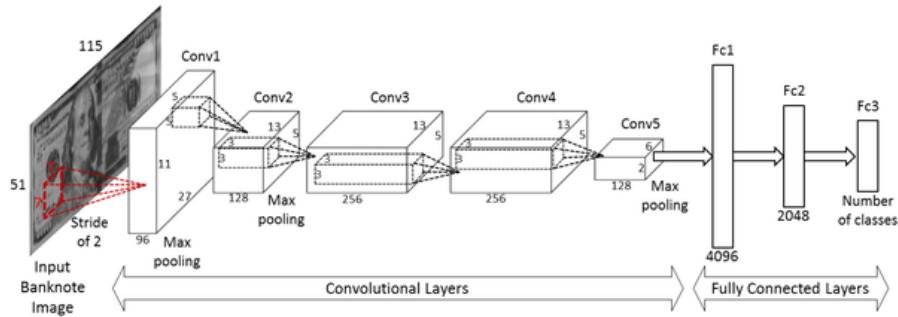
Hình 2.44. Ví dụ MaxPooling [18]



Hình 2.45. Ví dụ MaxPooling trong thực tế [23]

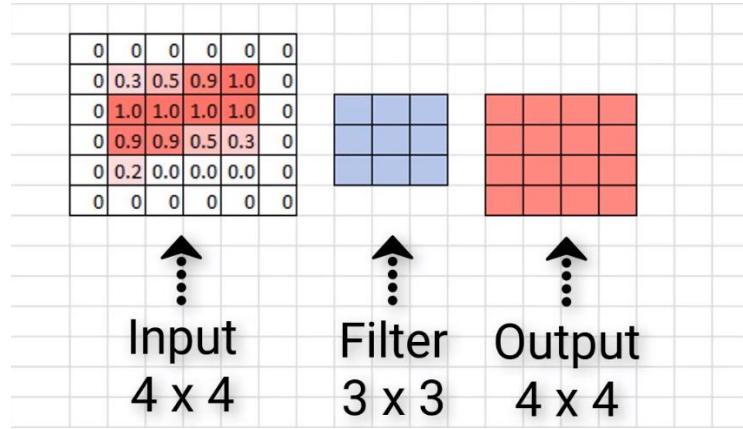
- Dropout
 - Dropout là kĩ thuật loại bỏ ngẫu nhiên những neuron trong mạng để giảm thiểu overfitting. Điều này có nghĩa là những output của neuron trước sẽ bỏ qua không sử dụng cũng như weight cũng sẽ không được update vào mạng hiện tại đang sử dụng dropout
- Flatten
 - Những layer fully connected được định nghĩa dưới dạng vector. Nhưng mạng lưới neural cho ra hàng loạt bộ lọc (filter) dưới hình dạng

lưới (grid). Flatten có nhiệm vụ biến đổi những bộ lọc này thành dạng vector để sử dụng tích chập và backpropagate (hồi tiếp).



Hình 2.46. Quá trình biến đổi của flatten [24]

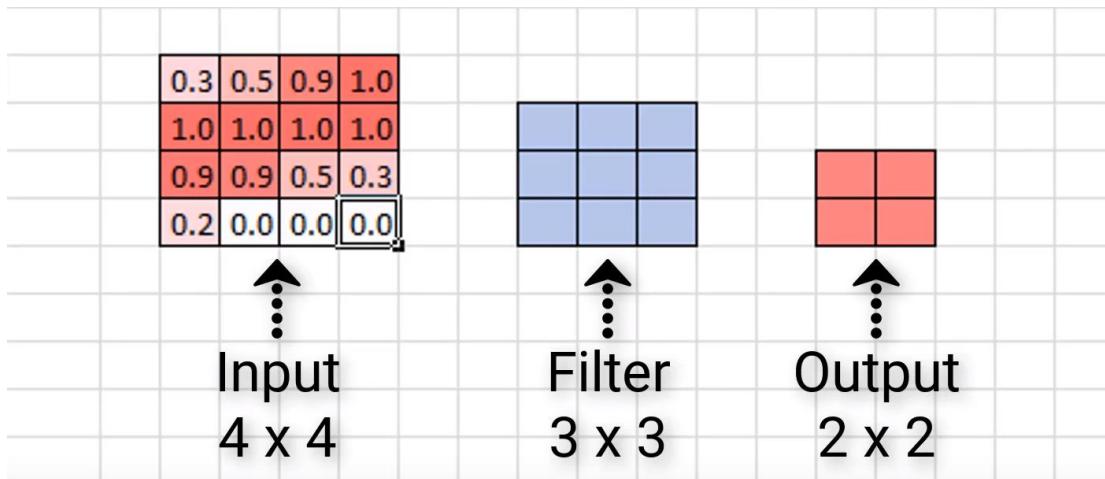
- Dense
 - o Lớp dense đại diện cho một ma trận phép nhân vector. Những giá trị của ma trận là những thông số dùng để huấn luyện và những những giá trị này cũng được update khi backpropagate
 - o $u^T W, W \in R^{n \times m}$
 - o Output sẽ cho ra một vector m chiều và lớp dense sẽ thay đổi số chiều của vector này. Theo cách nói toán học, nó sẽ áp dụng rotation (xoay), scaling (thay đổi kích thước), translation transform (biến đổi bằng cách dịch chuyển vị trí).
- Padding
 - o Zero padding
 - Zero padding là kỹ thuật cho phép giữ lại kích thước input (input size)
 - Zero padding xảy ra khi ta cho tất cả các phần tử viền của ma trận bằng 0



Hình 2.47. Ví dụ Zero padding [25]

- Valid padding

- Kích thước input sẽ bị co lại
- Với kích thước $n \times n$, ta sẽ giải quyết với 1 bộ lọc $f \times f$ và kết quả thu được $(n - f + 1) \times (n - f + 1)$
- Ví dụ với ma trận 4×4 : $(n - f + 1) = (4 - 3) + 1 = 2$



Hình 2.48. Ví dụ Valid Padding [25]

2.6.5.3 Huấn luyện và Dự đoán

- Compile:
 - Thành phần này có nhiệm vụ tạo thành một object Python, object này sẽ xây dựng mạng CNN

- Object này được xây dựng dựa trên tính toán đồ thị với một định dạng đúng với keras backend đang sử dụng.
- Complie có những thành phần như loss (hàm mất mát), optimizer (hàm tối ưu), ...
- Batch size [26]
 - Là 1 siêu tham số định nghĩa số lượng mẫu cần làm việc trước khi cập nhật những thông số trong model
 - Batch cũng có thể hiểu là vòng lặp for qua 1 hoặc nhiều mẫu và dự đoán. Cuối chuỗi hoạt động của batch, lớp dự đoán sẽ được so sánh với output mong muốn và tính toán ra sai số.
- Epoch
 - Là 1 siêu tham số định nghĩa số lần sẽ huấn luyện
 - 1 epoch bao gồm nhiều batch
- Verbose
 - Dùng để hiện thị quá trình huấn luyện
 - 0: chế độ im lặng, không hiện gì
 - 1: chế độ có thanh tiến độ
 - 2: chế độ chỉ hiện số epoch

2.7 YOLO v3 [27]

2.7.1 Giới thiệu

Đây là thuật toán nhận diện vật thể thứ 3 trong các thế hệ YOLO. Thuật toán này cải thiện độ chính xác và cải thiện khả năng nhận diện những vật thể nhỏ.

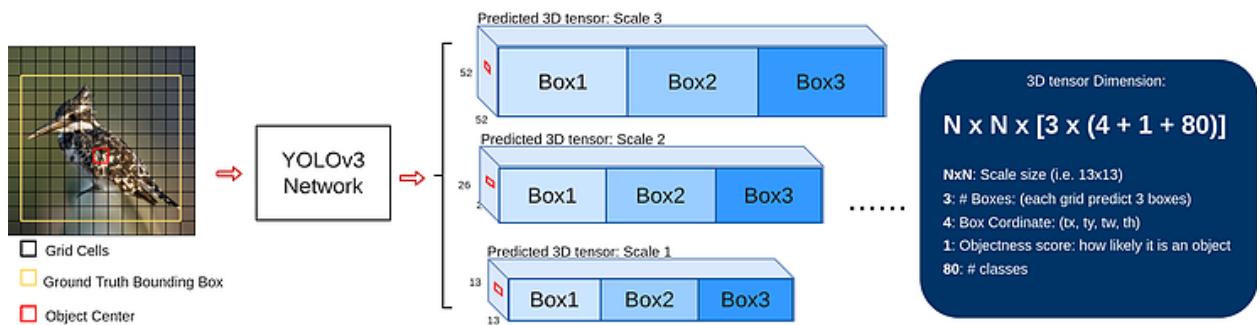
2.7.2 Thuật toán

Đầu tiên, mạng YOLO input vào hình ảnh dưới dạng 3D Tensors với 3 mức cân chỉnh(scale) khác nhau như hình dưới (bước 2). Những scale này được thiết kế để nhận diện vật thể với đa dạng kích thước.

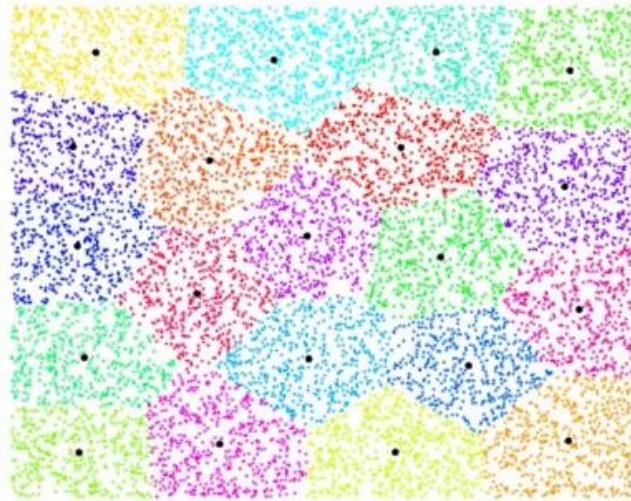
Ví dụ với scale 13x13, hình được phân ra thành ma trận 13x13 ô. Mỗi ô tương ứng với 1x1x255 voxel (điểm ảnh 3 chiều) trong 3D Tensors. 255 chính là $3 \times (4 + 1 + 80)$. Giá trị trong 3D Tensors bao gồm Tọa độ của khung bao quanh vật thể (bounding box), chỉ số đánh giá là vật thể (Objectness score), số của lớp phân loại (class confidence)

Thứ 2, nếu mà ô trung tâm của vùng tin tưởng (màu đỏ trong màu vàng), rơi vào vùng chắc chắn thì ô này sẽ được sử dụng để dự đoán khung bao của vật thể. Và Objectness score cho ô này là “1”, những ô khác là “0”. Mỗi ô sẽ có tương ứng với 3 vùng bao với 3 kích thước khác nhau. Thì nên quá trình training là để model này chọn được đúng vùng bao và tính toán chính xác tọa độ, độ bù (offset). Thì nên ô này sẽ chọn khung bao theo 1 luật là khung nào trong quá trình training được lặp lại nhiều nhất (overlap).

Cuối cùng, cách tìm ra giá trị cho 3 khung bao input: sử dụng K-mean clustering để tìm ra 9 vùng (clusters) trước khi train. 9 kích thước cho 9 cluster, 3 cho mỗi scale. Thông tin đầu vào này giúp cho mạng tính toán offset/coordinate chính xác hơn bởi vì nếu bằng trực giác hay những lựa chọn “đại” cho kích thước vùng bao sẽ làm cho mạng huấn luyện khó và lâu hơn.



Hình 2.49. Chuỗi xử lý của YOLO [27]

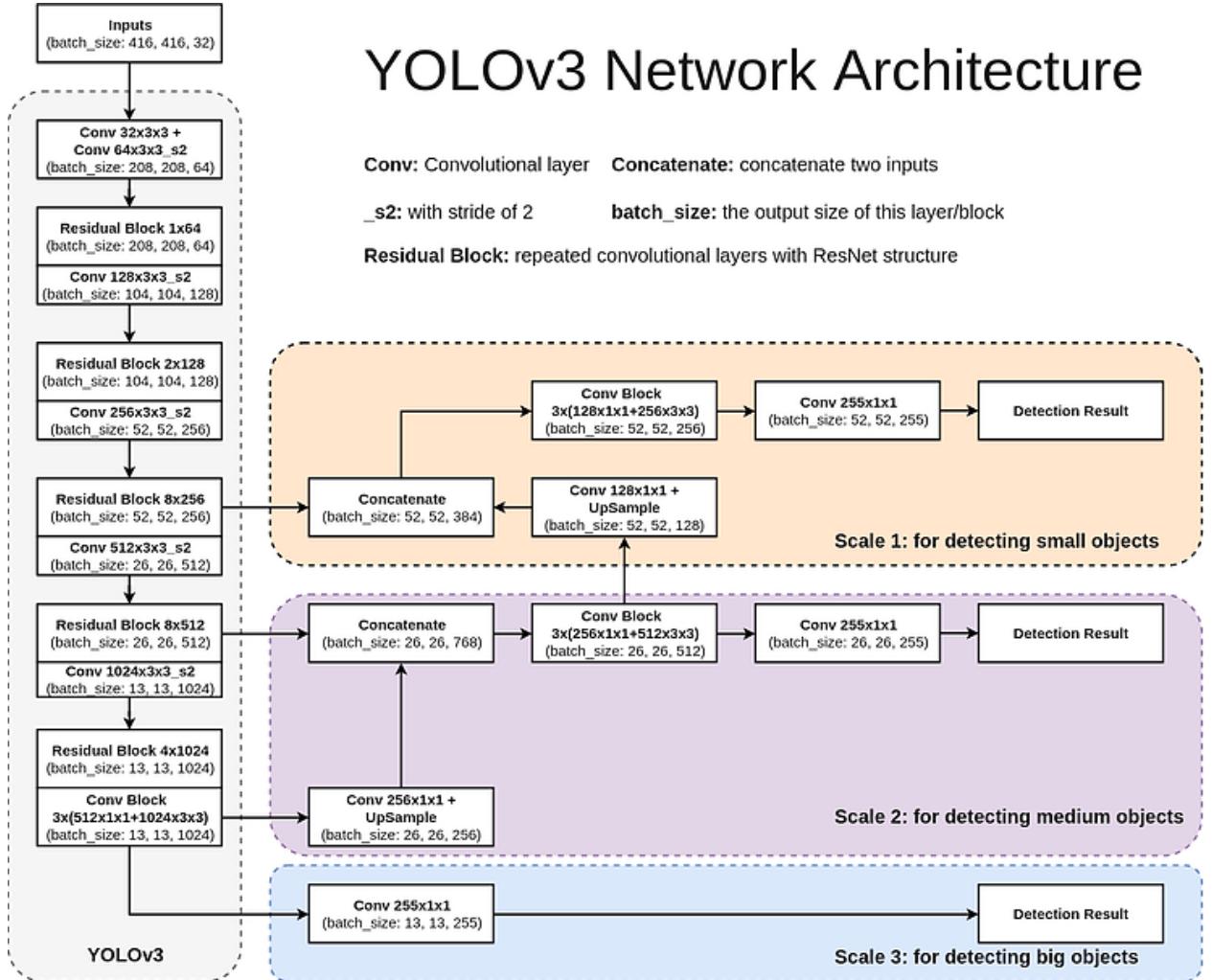


Hình 2.50. Ví dụ K-means clustering [28]

2.7.3 Kiến trúc mạng

Mạng YOLO bao gồm 75 convolutional layers, không sử dụng lớp kết nối đầy đủ (fully-connected layer). Kiến trúc này được xây dựng có khả năng xử lý với ảnh có kích thước khác nhau. Và cũng không sử dụng pooling layer. Có sử dụng 1 convolutional layer với stride 2 để giảm mẫu của feature map, bỏ qua đặc trưng bất biến kích thước. Ngoài ra, YOLO còn áp dụng ResNet-alike và FPN-alike để cải thiện độ chính xác.

YOLOv3 Network Architecture



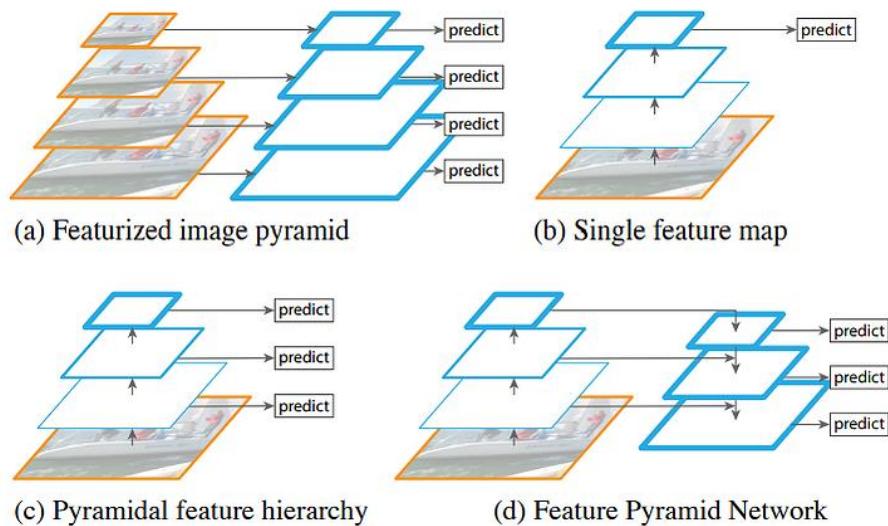
Hình 2.51. Kiến trúc mạng YOLO v3 [27]

2.7.4 Scales: Xử lý những kích thước khác nhau của vật thể

Vật thể có kích thước đa dạng, vì vậy mạng YOLO phải có khả năng đáp ứng được điều đó. Vì vậy bản đồ đặc tính (feature map) sẽ được tạo ra với nhiều kích thước từ lớn đến nhỏ. Mạng càng đi sâu thì feature map càng nhỏ lại làm khó khăn hơn trong việc nhận diện.

Tuy nhiên, những features không thích hợp tuyệt đối với những độ sâu khác nhau. Khi mà chiều sâu của mạng tăng lên, feature sẽ thay đổi từ low-level (edge, colors, 2D positions, ...) lên high-level (những thông tin có ngữ nghĩa: chó, mèo, xe, ...).

Trong YOLO v3, độ chính xác được cải thiện thông qua cấu trúc FPN-like



Hình 2.52. Minh họa quá trình huấn luyện đặc trưng với nhiều tỉ lệ khác nhau [27]

Giải thích:

- (a) Kiến trúc đơn giản nhất: Tạo ra hàng loạt ảnh dạng kim tự tháp và đưa từng tầng của kim tự tháp làm input cho một mạng riêng biệt cho từng tầng được thiết kế cho tỉ lệ của nó. Nhược điểm: xử lý chậm hơn vì cần xử lý mạng riêng biệt hoặc nhiều tiến trình riêng biệt.
- (b) Quá trình dự đoán kết thúc tại cuối feature map. Cấu trúc này không thể xử lý đa tỉ lệ.
- (c) Quá trình dự đoán hoàn thành tại những chiều sau nhau của kim tự tháp. Nó được mô phỏng theo SSD. Dự đoán dùng chính feature đã được huấn luyện cho riêng tầng đó (của kim tự tháp), còn những features khác ở những tầng khác sẽ không được sử dụng vào.
- (d) Giống với (c) nhưng những features ở những tầng khác cũng được áp dụng bằng cách nội suy mẫu của feature map tầng cao hơn và hợp nhất với feature map hiện tại. Nó tạo cho feature map hiện tại thấy được lớp “tương

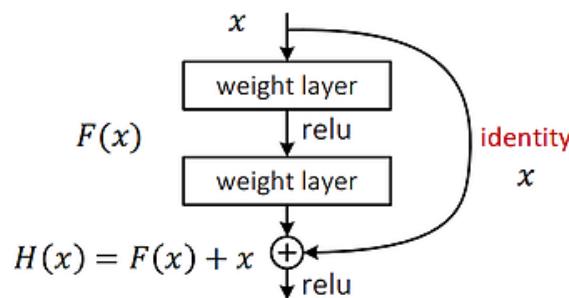
lai” và sử dụng cả 2 để dự đoán độ chính xác. Với kỹ thuật này, mạng bắt được thông tin của cả low-level và high-level tốt hơn.

2.7.5 Kiến trúc ResNet-alike: Cách tốt hơn tìm ra những feature riêng biệt

Kiến trúc ResNet-alike (Residual Blocks in the YOLOv3 Architecture Diagram). Về cơ bản, Residual Blocks bao gồm những convolutional layers và shortcut paths.

Deep Residual Learning

• Residual net



Hình 2.53. Ví dụ về Residual Blocks [27]

Đường cong biểu diễn cho shortcut path, nếu không có thì mạng sẽ trở thành CNN thuần, huấn luyện theo từng lớp. Những layer ở trong shortcut sẽ đưa thêm gì vào feature cũ để tạo ra feature tốt hơn. Feature phức tạp $H(x)$ thay vì được tạo theo đường thẳng, được thay bằng $H(x) = F(x) + x$ với x ở đây là x ở feature cũ và $F(x)$ là “supplement” hoặc “residual”. Việc này đơn giản hóa mạng để huấn luyện những feature ổn định, đặc biệt với một mạng cực sâu. Như vậy, thay vì huấn luyện duy nhất 1 feature phức tạp, việc huấn luyện được chuyển sang dạng bổ sung hoặc bù thêm vào feature cũ.

2.7.6 Không có lớp softmax: phân loại đa lớp

Lớp Softmax được thay bởi 1x1 convolutional layer với hàm logistic. Khi sử dụng Softmax, thì output chỉ chính xác là MỘT trong những lớp phân loại. Tuy nhiên trong 1 vài dataset, trường hợp nhãn có thể bị trùng nhau về mặt ngữ nghĩa (Nữ và Người),

softmax có thể sẽ không khái quát hóa được phân bố đa ta tối ưu. Thay vào đó hàm Logistic sẽ được dùng để xử lý với phân loại đa nhãn (multi-label).

2.8 Board Nvidia Jetson Tx2 và các ngoại vi

2.8.1 Thông số kỹ thuật: [29]



Hình 2.54. Board Nvidia Jetson Tx2

JETSON TX2 MODULE

- NVIDIA Pascal™ Architecture GPU
- 2 Denver 64-bit CPUs + Quad-Core A57 Complex
- 8 GB L128 bit DDR4 Memory
- 32 GB eMMC 5.1 Flash Storage
- Connectivity to 802.11ac Wi-Fi and Bluetooth-Enabled Devices
- 10/100/1000BASE-T Ethernet

I/O

- USB 3.0 Type A
- USB 2.0 Micro AB (supports recovery and host mode)
- HDMI
- M.2 Key E

- PCI-E x4
- Gigabit Ethernet
- Full-Size SD
- SATA Data and Power
- GPIOs, I2C, I2S, SPI, CAN*
- TTL UART with flow control
- Display Expansion Header*
- Camera Expansion Header*
 - *I/O expansion headers: refer to product documentation for header specification.

POWER OPTIONS

External 19V AC Adapter

JETSON CAMERA MODULE

5 MP Fixed Focus MIPI CSI Camera

BUTTONS

- Power On/Off
- Reset
- Force Recovery
- User-Defined

2.8.2 Camera Orbec Astra

Thông số kỹ thuật:

Kích thước: 165 x 30 x 40 mm

Cân nặng: 0.3 kg

Khoảng cách: 0.6 – 8.0m (Optimal 0.6 – 5.0m)

Kích thước ảnh chiều sâu:

- 640*480 (VGA) @ 30FPS
- 320*240 (QVGA) @ 30FPS

- 160*120 (QQVGA) @ 30FPS

RGB Image Size

- 1280*960 @ 7FPS
- 640*480 @ 30FPS
- 320*240 @ 30FPS

Field of View

- 60° horiz x 49.5° vert. (73° diagonal)

Data Interface

USB 2.0

Microphones: 2

Operating Systems

- Windows 7/8/10, Linux, Android

Power

USB 2.0

Software

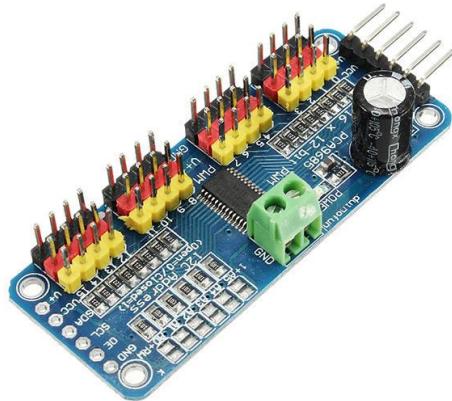
- Astra SDK or OpenNI 2 or 3rd Party SDK



Hình 2.55. Camera Orbbec Astra [30]

2.8.3 Driver Controller shield

2.8.3.1 PCA 9685



Hình 2.56. PCA9685 [31]

Mạch điều khiển 16 Chanel PWM PCA9685 được sử dụng để có thể xuất ra đồng thời 16 xung PWM từ 16 cổng khác nhau thông qua giao tiếp I2C sử dụng IC PCA9685, giúp bạn có thể điều khiển đồng thời 16 RC Servo hoặc Dimmer 16 thiết bị đồng thời, ...

Mạch điều khiển 16 Chanel PWM PCA9685 có cấu trúc phần cứng phần cứng đơn giản cũng như bộ thư viện có sẵn trên Arduino nên rất dễ dàng sử dụng và kết nối.

Mạch điều khiển 16 Chanel PWM PCA9685 có chất lượng phần cứng, gia công tốt, độ bền cao, phù hợp cho các nhu cầu cần điều khiển nhiều xung PWM như Robot cánh tay máy, Robot nhẹn, ...

Thông số kỹ thuật:

- Mạch điều khiển 16 Chanel PWM PCA9685
- IC chính: PCA9685
- Điện áp sử dụng: 2.3 ~ 5.5VDC.
- Số kênh PWM: 16 kênh, tần số: 40~1000Hz
- Độ phân giải PWM: 12bit.
- Giao tiếp: I2C (chấp nhận mức Logic TTL 3 ~ 5VDC)

- Kích thước: 62.5mm x 25.4mm x 3mm

2.8.3.2 Pinout board Nvidia trên shield

Jetson TX2 J21 Header

Sysfs GPIO	Connector Label	Pin	Pin	Connector Label	Sysfs GPIO
	<u>3.3 VDC</u> <u>Power</u>	1	2	<u>5.0 VDC</u> <u>Power</u>	
	<u>SDA1</u> <u>General I2C Data 3.3.V,</u> <u>I2C Bus 1</u>	3	4	5.0 VDC Power	
	<u>SCL1</u> <u>General I2C Clock 3.3.V,</u> <u>I2C Bus 1</u>	5	6	GND	
gpio396	GPIO_GCLK <i>Audio Master Clock</i> (1.8/3.3.V)	7	8	TXD0 <i>UART #0 Transmit</i>	
	GND	9	10	RXD0 <i>UART #0 Receive</i>	
gpio466	GPIO_GEN0 <i>UART #0 Request to Send</i>	11	12	GPIO_GEN1 <i>Audio I2S #0 Clock</i>	gpio392
gpio397	GPIO_GEN2 <i>Audio Code Interrupt</i>	13	14	GND	
gpio255	GPIO_GEN3 <i>From GPIO Expander</i> (P17)	15	16	GPIO_GEN4 <i>Unused</i>	gpio296
	<u>3.3 VDC</u> <u>Power</u>	17	18	GPIO_GEN5 <i>Modem Wake AP GPIO</i>	gpio481
gpio429	SPI_MOSI <i>SPI #1 Master Out/Slave</i> <i>In</i>	19	20	GND	

gpio428	SPI1_MISO <i>SPI #1 Master In/Slave Out</i>	21	22	GPIO_GEN6 <i>From GPIO Epander (P16)</i>	gpio254
gpio427	SPI_SCLK <i>SPI #1 Shift Clock</i>	23	24	SPI_CE0_N <i>SPI Chip Select #0</i>	gpio430
	GND	25	26	SPI_CE1_N <i>SPI #1 Chip Select #1</i>	
	ID_SD <i>General I2C #1 Data (3.3V), I2C Bus 0</i>	27	28	ID_SC <i>General I2C #1 Clock (3.3V), I2C Bus 0</i>	
gpio398	GPIO5 <i>Audio Reset (1.8/3.3V)</i>	29	30	GND	
gpio298	GPIO6 <i>Motion Interrupt (3.3V)</i>	31	32	GPIO12 <i>Unused</i>	gpio297
gpio389	GPIO13 <i>AP Wake Bt GPIO</i>	33	34	GND	
gpio395	GPIO19 <i>AUDIO I2S #0 Left/Right Clock</i>	35	36	GPIO16 <i>UART #0 Clear to Send</i>	gpio467
gpio388	GPIO26 <i>(3.3V)</i>	37	38	GPIO20 <i>Audio I2S #0 Data in</i>	gpio394
	GND	39	40	GPIO21 <i>Audio I2S #0 Data in</i>	gpio393

Bảng 2.1. Bảng đầu ra chân của board Nvidia Jetson

- Những chân sử dụng sẽ được bôi đỏ, in nghiêng và gạch dưới:
 - SDA1 SCL1: I2C1
 - 3V3, 5V, GND: Nguồn
 - GPIO393
 - GPIO394
 - GPIO297
 - GPIO388 ➔ Trở kéo xuống

- GPIO298 → Trở kéo xuông
- GPIO467 → Trở kéo xuông
- GPIO255 → Trở kéo xuông

2.8.4 Thư viện

2.8.4.1 I2CDEV

Thư viện hỗ trợ giao tiếp I2C cho những chân SCL SDA của board.

2.8.4.2 SMBUS2

Thư viện hỗ trợ đầy đủ struct của I2C và những hàm điều khiển I2C

- Get i2c capabilities (I2C_FUNCS)
- read_byte
- write_byte
- read_byte_data
- write_byte_data
- read_word_data
- write_word_data
- read_i2c_block_data
- write_i2c_block_data
- write_quick
- process_call
- read_block_data
- write_block_data
- block_process_call
- i2c_rdwr - *combined write/read transactions with repeated start*

Hoạt động với Python 2.7 và 3.*

2.8.4.3 Tensorflow

Framework sử dụng cho machine learning của Google

Tensorflow là một thư viện mã nguồn mở cung cấp khả năng xử lý tính toán số học dựa trên biểu đồ mô tả sự thay đổi của dữ liệu. Tensor được sử dụng khi bạn cần giải quyết các bài toán supervised learning.

Khái niệm cơ bản trong Tensorflow [32]

Khi thực hành với Tensorflow, sẽ có rất nhiều khái niệm phức tạp. Tuy nhiên chỉ ở mức cơ bản, chúng ta sẽ đi đến khái niệm quan trọng nhất trong Tensorflow là **Tensor**. Node

Vì Tensorflow mô tả lại dòng chảy của dữ liệu thông qua graph nên mỗi một điểm giao cắt trong graph thì được gọi là Node. Tại sao điều này quan trọng thì là vì các Node chính là điểm đại diện cho việc thay đổi của dữ liệu nên việc lưu trữ lại tham chiếu của các Node này là rất quan trọng.

Tensor

Như trong bài viết trước mình có đề cập, để giải được các bài toán Machine Learning, cần phải làm cho máy tính có thể hiểu được dữ liệu của tập nguồn và dữ liệu của tập đích. Tensorflow cung cấp một loại dữ liệu mới được gọi là *Tensor*. Trong thế giới của Tensorflow, mọi kiểu dữ liệu đều được quy về một mối được gọi là Tensor hay trong Tensorflow, tất cả các loại dữ liệu đều là Tensor. Vậy nên có thể hiểu được phần nào cái tên Tensorflow là một thư viện mô tả, điều chỉnh dòng chảy của các Tensor.

Tensor là một kiểu dữ liệu dạng mảng có nhiều chiều được mô tả dạng `Tensor = [[[1,1,1],[178,62,74]],[[45,2,2],[19,0,17]],[[7,5,2],[0,11,4]],[[8,13,5],[1,6,7]]]`. Mảng nhiều chiều này được đính kèm thêm một vài thuộc tính tham chiếu khác. Các thuộc tính của Tensor được mô tả trong tài liệu bao gồm:

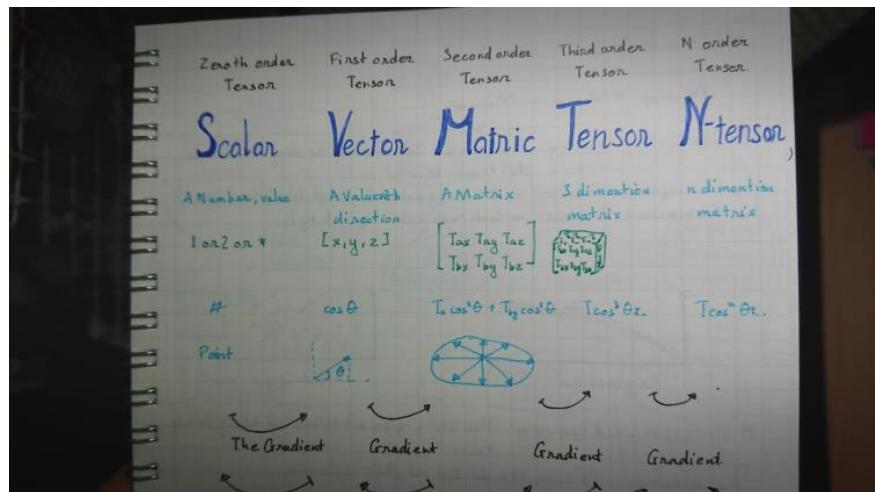
- **device**: Tên của thiết bị mà Tensor hiện tại sẽ được xuất bản. Có thể None.
- **graph**: Đồ thị chứa Tensor hiện tại.
- **name**: Tên của Tensor hiện tại.
- **shape**: Trả về `TensorShape` mô tả lại Shape của Tensor hiện tại.

- **op:** ^{Toán tử / Phép toán}
Operation được sử dụng để xuất bản Tensor hiện tại.
- **dtype:** ^{Phần tử} Kiểu của các *elements* trong Tensor hiện tại.

Rank

Rank là bậc hay độ sâu của một Tensor. Ví dụ như `Tensor = [1]` sẽ có rank là 1, `Tensor = [[[1,1,1],[178,62,74]]]` sẽ có rank bằng 3, `Tensor = [[1,1,1],[178,62,74]]` sẽ có rank bằng 2. Cách nhanh nhất để xác định rank của một Tensor là đếm số lần mở ngoặc vuông cho đến giá trị khác vuông đầu tiên. Việc phân rank này khá quan trọng vì nó đồng thời cũng giúp phân loại dữ liệu của Tensor. Khi ở cách rank đặc biệt cụ thể, Tensor có những tên gọi riêng như sau:

- **Scalar:** Khi Tensor có rank bằng 0, Tensor đại diện cho một số hoặc một chuỗi cụ thể. Ví dụ: `scalar=123`.
- **Vector:** Vector là một Tensor rank 1. Trong python thì Vector là một *list* hay *mảng* *một chiều* chứa các số. Ví dụ: `list=[123,345]`.
- **Matrix:** Đây là một Tensor rank 2 hay mảng hai chiều theo khái niệm của Python. Ví dụ: `matrix=[[1,2],[2,1]]`.
- **N-Tensor:** Khi rank của Tensor tăng lên lớn hơn 2, chúng được gọi chung là N-Tensor.



Hình 2.57. Hình minh họa cho Tensorflow [32]

2.8.4.4 Keras

Keras được coi là một thư viện ‘high-level’ với phần ‘low-level’ (còn được gọi là *backend*) có thể là TensorFlow, CNTK, hoặc Theano. Keras có cú pháp đơn giản hơn TensorFlow rất nhiều. Với mục đích giới thiệu về các mô hình nhiều hơn là các sử dụng các thư viện deep learning, tôi sẽ chọn Keras với TensorFlow là ‘backend’.

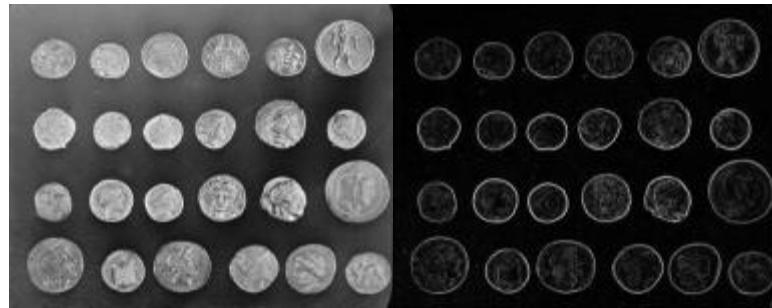
- Keras ưu tiên trải nghiệm của người lập trình
- Keras đã được sử dụng rộng rãi trong doanh nghiệp và cộng đồng nghiên cứu
- Keras giúp dễ dàng biến các thiết kế thành sản phẩm
- Keras hỗ trợ huấn luyện trên nhiều GPU phân tán
- Keras hỗ trợ đa backend engines và không giới hạn bạn vào một hệ sinh thái

Amazon hiện cũng đang làm việc để phát triển MXNet backend cho Keras. Mô hình Keras có thể được huấn luyện trên một số nền tảng phần cứng khác nhau ngoài CPU:

- NVIDIA GPU
- Google TPUs, thông qua TensorFlow backend và Google Cloud
- Các OpenCL GPU, chẳng hạn như các sản phẩm từ AMD, thông qua PlaidML Keras backend.

2.8.4.5 Scikit-image [33]

Bộ sưu tập những thuật toán xử lý ảnh thường dùng cho tiền huấn luyện, tiền xử lý, ...



Hình 2.58. Minh họa về tìm cạnh trong scikit-image

2.8.4.6 Openni2

Openni là framework mã nguồn mở dùng cho những ứng dụng để truy cập những thiết bị tương tác tự nhiên (natural interaction device).

APIs hỗ trợ:

- Voice and voice command recognition
- Hand gestures
- Body Motion Tracking

2.9 Lập trình đa luồng

2.9.1 Luồng (thread). Khác nhau giữa thread và process (tiến trình)

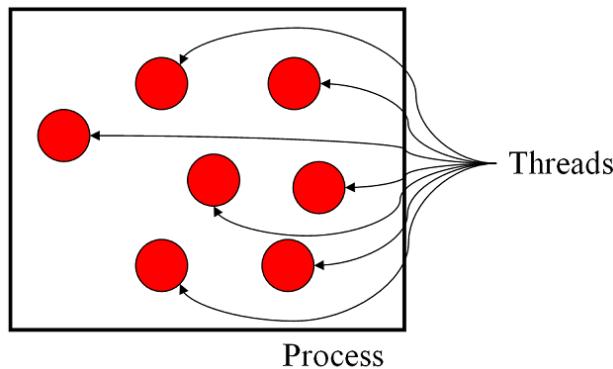
Thread là một đơn vị cơ bản trong CPU. Một luồng sẽ chia sẻ với các luồng khác trong cùng process về thông tin data, các dữ liệu của mình. Việc tạo ra thread giúp cho các chương trình có thể chạy được nhiều công việc cùng một lúc.

Process là quá trình hoạt động của một ứng dụng. Tiến trình (process) chứa đựng thông tin tài nguyên, trạng thái thực hiện của chương trình.

Thread là một bước điều hành bên trong một process. Luồng (thread) là một khối các câu lệnh (instructions) độc lập trong một tiến trình và có thể được lập lịch bởi hệ điều hành. Hay nói một cách đơn giản, Thread là các hàm hay thủ tục chạy độc lập đối với

chương trình chính. Một process dĩ nhiên có thể chứa nhiều thread bên trong nó. Điểm quan trọng nhất cần chú ý là một thread có thể làm bất cứ nhiệm vụ gì một process có thể làm.

Một điểm khác biệt nữa đó là nhiều thread nằm trong cùng một process dùng một không gian bộ nhớ giống nhau, trong khi process thì không. Điều này cho phép các thread đọc và viết cùng một kiểu cấu trúc và dữ liệu, giao tiếp dễ dàng giữa các thread với nhau. Giao thức giữa các process, hay còn gọi là IPC (inter-process communication) thì tương đối phức tạp bởi các dữ liệu có tính tập trung sâu hơn. Ngoài các tài nguyên riêng của mình (các biến cục bộ trong hàm), các luồng chia sẻ tài nguyên chung của tiến trình. Việc thay đổi tài nguyên chung (ví dụ, đóng file, gán giá trị mới cho biến) từ một thread sẽ được nhìn thấy bởi tất cả các thread khác. Vì vậy, lập trình viên cần phải thực hiện đồng bộ việc truy cập tài nguyên chung giữa các luồng.



Hình 2.59. Sự khác nhau giữa thread và process [34]

2.9.2 Đa luồng (Multithreading)

Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.

Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng. Mỗi một Thread đều có

vòng đời chung là bắt đầu, chạy và kết thúc. Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy – được gọi là yielding. Một ví dụ đơn giản chỉ sử dụng một thread, có truyền tham số. Để chạy thread, ta dùng method start(). Target sẽ là function myThread, là nhiệm vụ mà thread phải hoàn thành.

[31]

2.9.3 Đồng bộ hóa các thread trong python

Trong lập trình đa luồng, các threads chia sẻ chung tài nguyên của tiến trình, vì vậy có những thời điểm nhiều luồng sẽ đồng thời thay đổi dữ liệu chung. Do đó, ta cần những cơ chế để đảm bảo rằng, tại một thời điểm chỉ có duy nhất một luồng được phép truy cập vào dữ liệu chung, nếu các luồng khác muốn truy cập vào đoạn dữ liệu này thì cần phải đợi cho thread trước đó hoàn thành công việc của mình.

Python cung cấp threading Module, mà bao gồm một kỹ thuật locking cho phép bạn đồng bộ hóa các Thread một cách dễ dàng. Một lock mới được tạo bởi gọi phương thức Lock().

Phương thức acquire(blocking) của đối tượng lock mới này được sử dụng để ép các Thread chạy một cách đồng bộ. Tham số blocking tùy ý cho bạn khả năng điều khiển để xem một Thread có cần đợi để đạt được lock hay không.

Nếu tham số blocking được thiết lập là 0, tức là Thread ngay lập tức trả về một giá trị 0 nếu không thu được lock và trả về giá trị 1 nếu thu được lock. Nếu blocking được thiết lập là 1, thì Thread cần đợi cho đến khi lock được giải phóng.

Phương thức release() của đối tượng lock được sử dụng để giải phóng lock khi nó không cần nữa.

2.9.4 Ứng dụng đa luồng

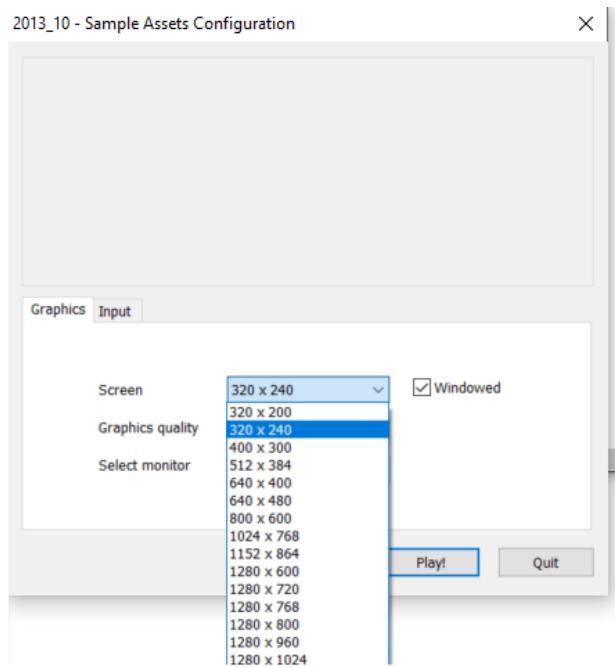
Đa luồng có rất công dụng vô cùng hữu ích thích hợp cho những tác vụ chạy ngầm không cần quan tâm chính xác thời gian hoàn thành, đặc điểm lớn nhất của nó vẫn là chạy song song nhiều luồng cùng 1 lúc, có thể kể đến tác dụng hữu hiệu nhất trong các ứng dụng

web, và dự án của bản thân là ghi log. Ví dụ dự án có tác vụ gồm nhiều ưu đãi, người dùng muốn lưu ưu đãi đó cho dùng lần sau thì cần ghi log 2 sự kiện gồm view log và save log

CHƯƠNG 3. XÂY DỰNG MÔ HÌNH VÀ GIẢI PHÁP

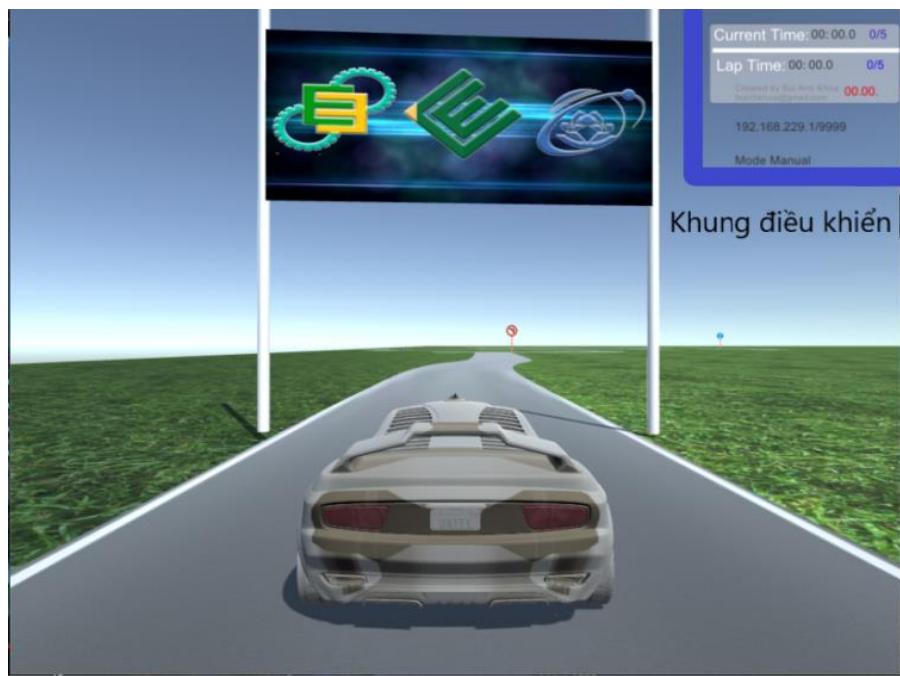
3.1 Chương trình mô phỏng Unity

3.1.1 Giao diện



Hình 3.1. Giao diện lựa chọn độ phân giải trong Unity

Mỗi chương trình Unity sau khi xuất (Dùng bản free của nhà sản xuất) sẽ cho ra giao diện cho phép người dùng chọn chất lượng hình ảnh và cài đặt những phím cho game.



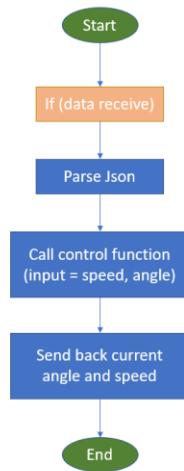
Hình 3.2. Giao diện hoạt động

Tại khung điều khiển bao gồm:

- Thời gian hiện hành
- Checkpoint đã hoàn thành / Checkpoint tổng
- Địa chỉ IP của server
- Mode hiện tại (Manual mode / Auto Pilot mode)

3.1.2 Flowchart

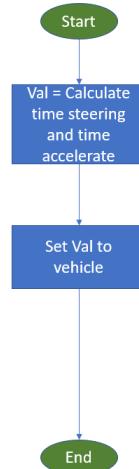
Server



Hình 3.3. Flowchart xử lý của server Tcp/Ip

- Flowchart (hình 3.3) mô tả quá trình nhận và xử lý dữ liệu của server sử dụng bất đồng bộ.
- Mỗi khi có data, server sẽ tính toán để điều khiển unity và feedback lại cho client

Control function



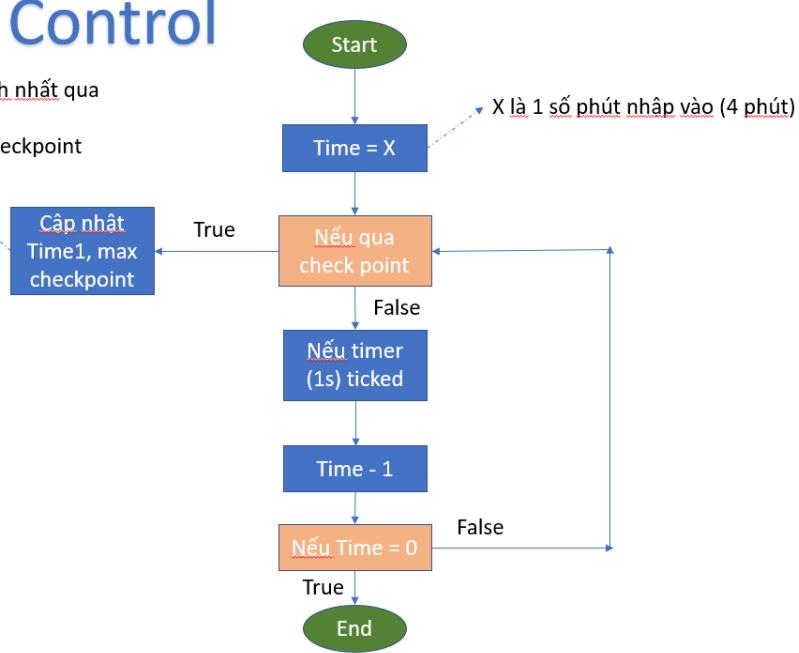
Hình 3.4. Flowchart xử lý tốc độ và góc bẻ lái

- Flowchart (hình 3.4) mô tả quá trình xử lý của class control trong unity

- Tại đây dữ liệu góc bẻ lái và tốc độ sẽ được tính toán và truyền cho class control để điều khiển xe

Lap-time Control

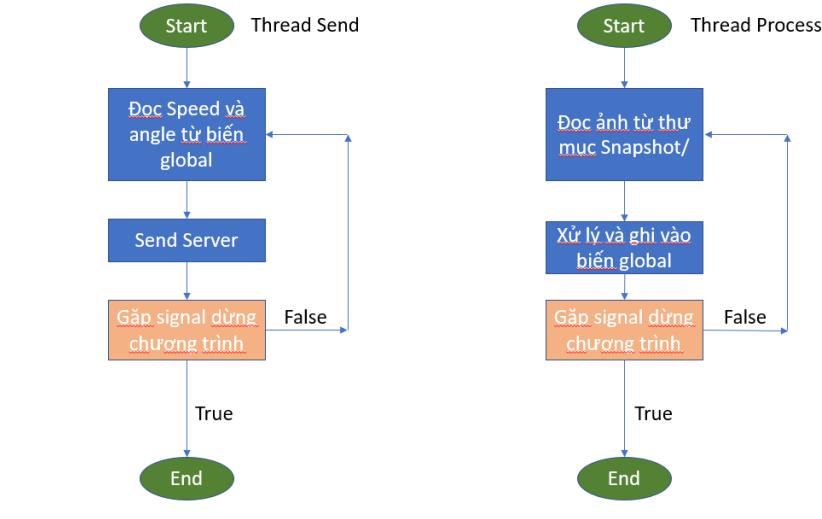
Time1 là thời gian nhanh nhất qua check point
 Max checkpoint là số checkpoint nhiều nhất vượt qua



Hình 3.5. Flowchart xử lý LapTime

- Flowchart (hình 3.5) mô tả quá trình xử lý giờ phút giây, số checkpoint đã vượt qua cũng như số lượng checkpoint và thời gian hoàn thành nhanh nhất

Client Control



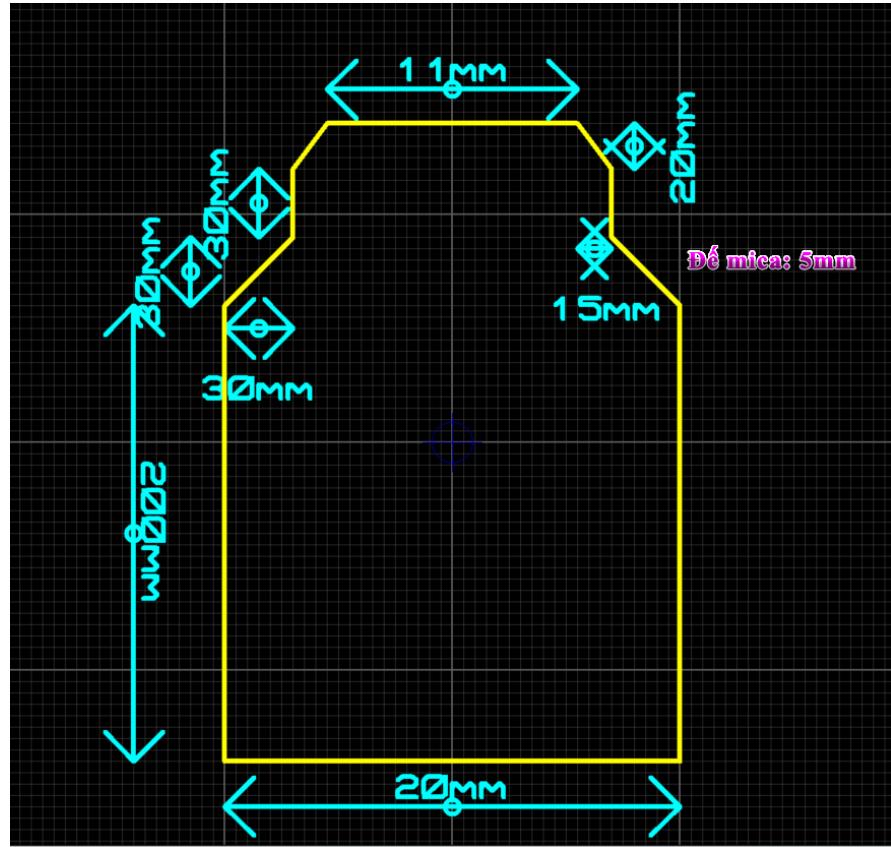
Hình 3.6. Flowchart giao tiếp giữa client và server

- Flowchart (hình 3.6) mô tả chương trình client với ứng dụng đa luồng
- Luồng truyền/nhận đảm nhận nhiệm vụ gửi data tới server với vòng lặp chu kì cho trước và nhận lại feedback
- Luồng xử lý đảm nhận những hàm tính toán, module tính toán, xử lý ảnh để tính ra góc và tốc độ và gửi cho luồng truyền nhận tiếp tục công việc

3.2 Mô hình xe RC

3.2.1 Tổng thể mô hình xe

- Thiết kế để mica để đặt board, đi dây điện và các ngoại vi:
 - Kích thước: 310 x 200 x 5mm
 - Màu sắc: đen
 - Hình thức sản xuất: Sử dụng máy CNC để cắt



Hình 3.7. Bản vẽ sơ lược về đế

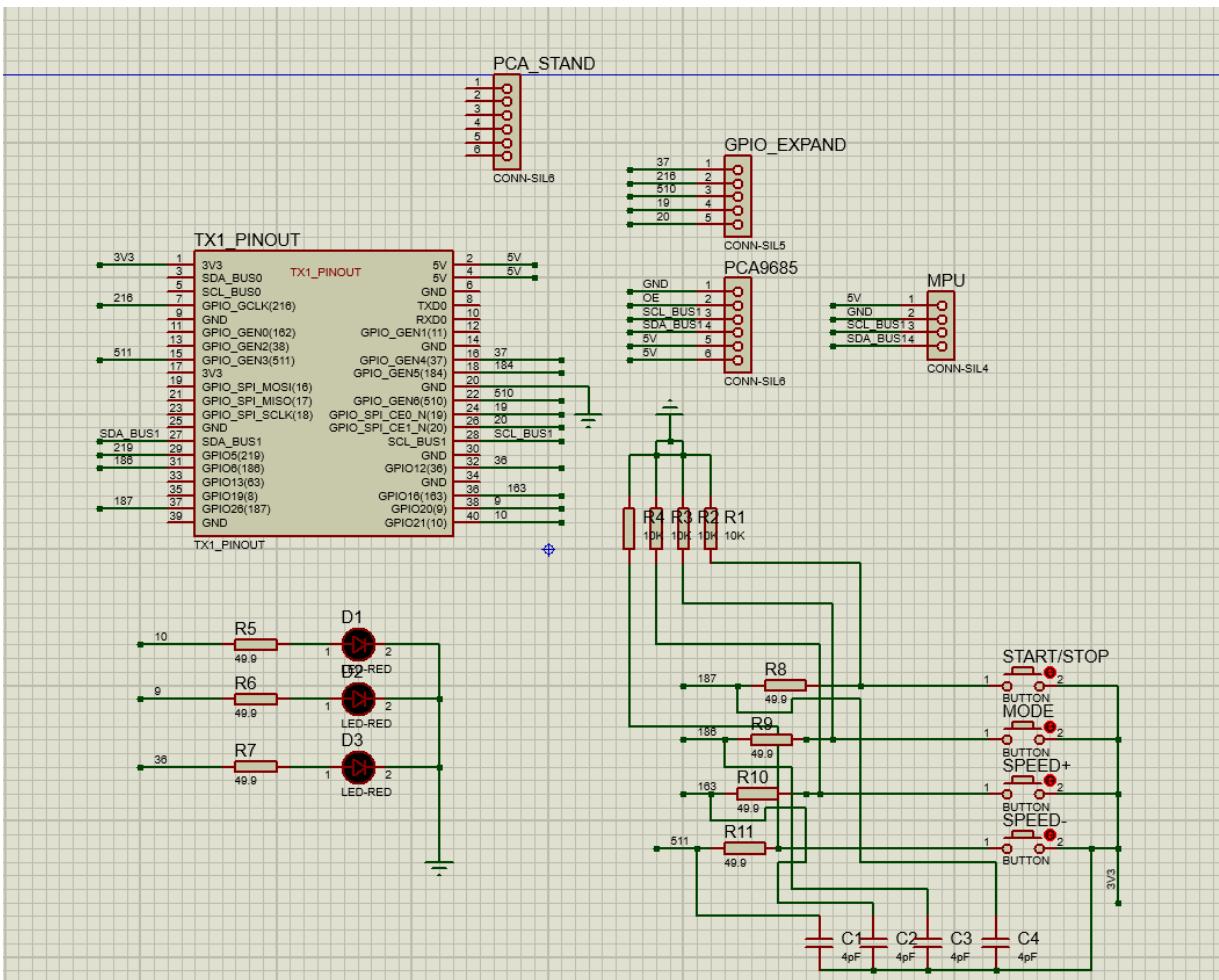
- Thiết kế lại lò xo:
 - o Kích thước: dài 60mm, đường kính 16mm, độ dày 2.2mm
 - o Hình thức sản xuất: Đi mua tại cửa hàng vật liệu sắt thép
- Thiết kế đế cho camera:
 - o Kích thước: 400 x 70 x 50mm
 - o Độ dày: 5mm
 - o Hình thức sản xuất: In 3D

3.2.2 Camera

Sử dụng camera Orbbec Astra

3.2.3 Shield

Schematic:



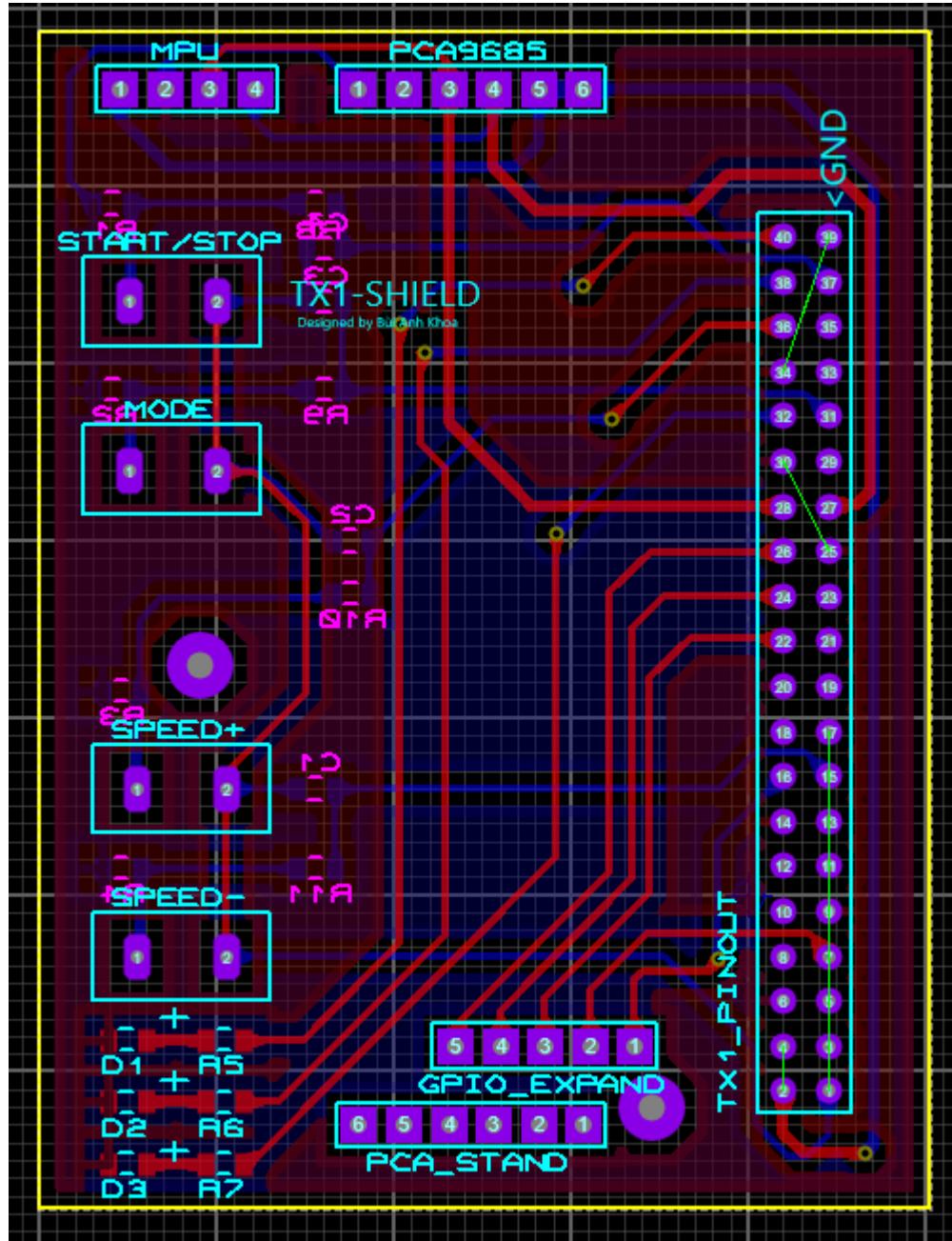
Hình 3.8. Schematic của shield Jetson TX

- Sử dụng trở kéo và tụ để lọc nhiễu cho nút bấm

Bao gồm:

- 1 board PCA9685
- 3 led 0805
- Trở kéo với 4 nút nhấn

PCB:



Hình 3.9. PCB của shield Jetson TX

- Đường dây 25 th
- Kích thước: 15x15 cm vừa với vị trí trống của board Nvidia
- Phủ đồng với chung GND

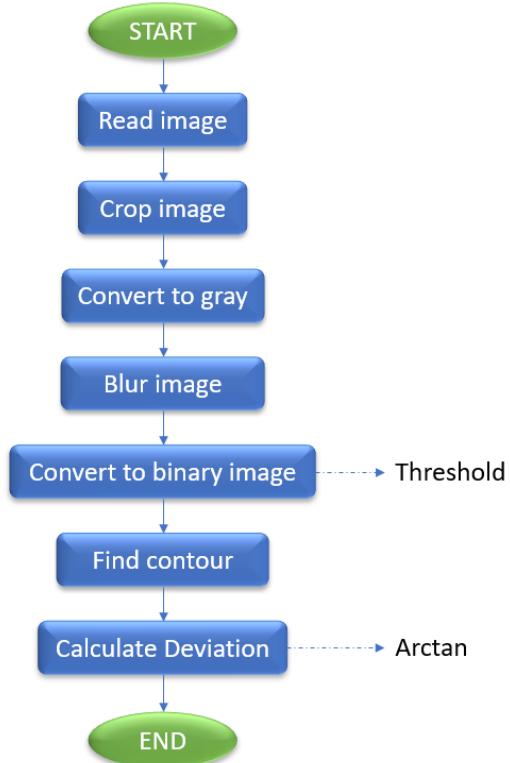
Thành phần:

- Dạng xếp tầng kết nối bằng rào cái vuông

3.3 Chương trình điều khiển

3.3.1 Nhận diện làn đường bằng opencv

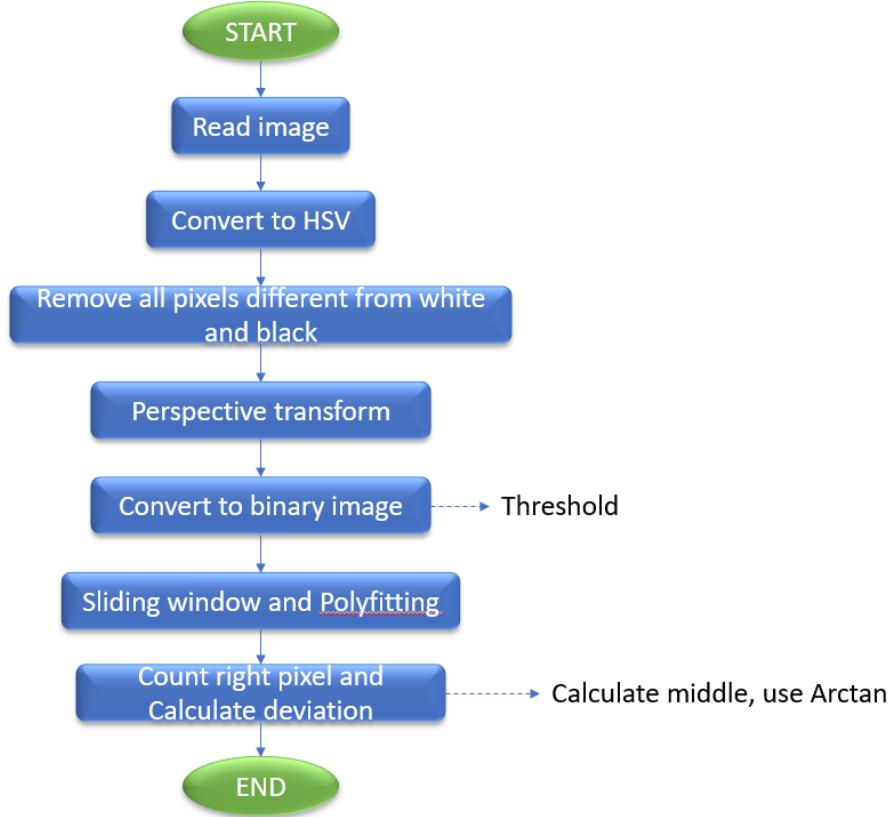
3.3.1.1 Contour



Hình 3.10. Flowchart xử lý sử dụng contour

- Những bước xử lý áp dụng những kỹ thuật căn bản của opencv.
- Áp dụng hình học lượng giác để tính toán góc lệch/ đánh lái

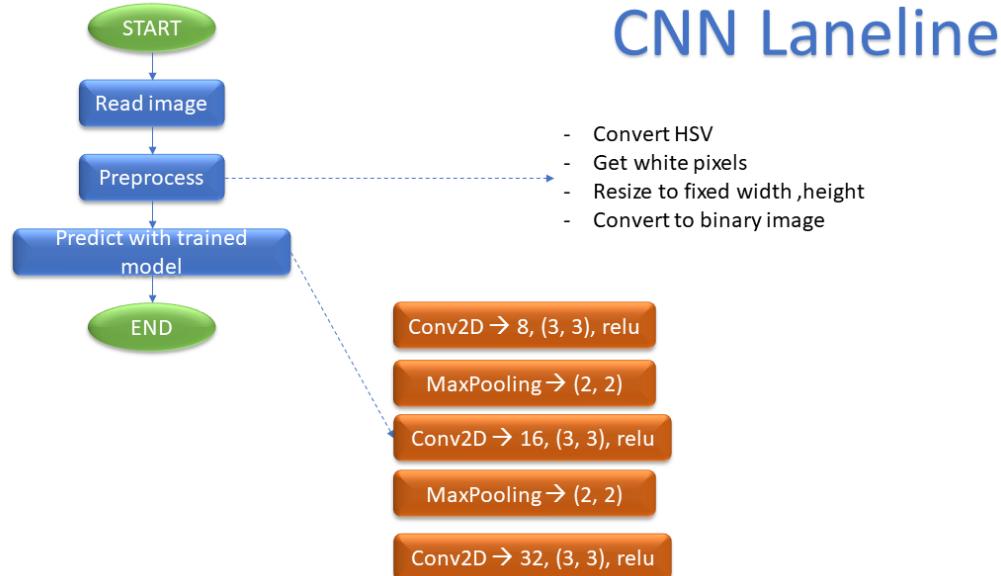
3.3.1.2 Sliding window



Hình 3.11. Flowchart xử lý sử dụng Sliding window

- Áp dụng kỹ thuật chia cửa sổ
- Qua mỗi cửa sổ, vẽ được 1 đường thẳng đi qua tâm, khi nối các đường thẳng của những cửa sổ, ta được một đường cong, từ đó tìm ra được phương trình đường cong xe sẽ chạy
- Áp dụng hình học lượng giác để tính góc lệch/ đánh lái

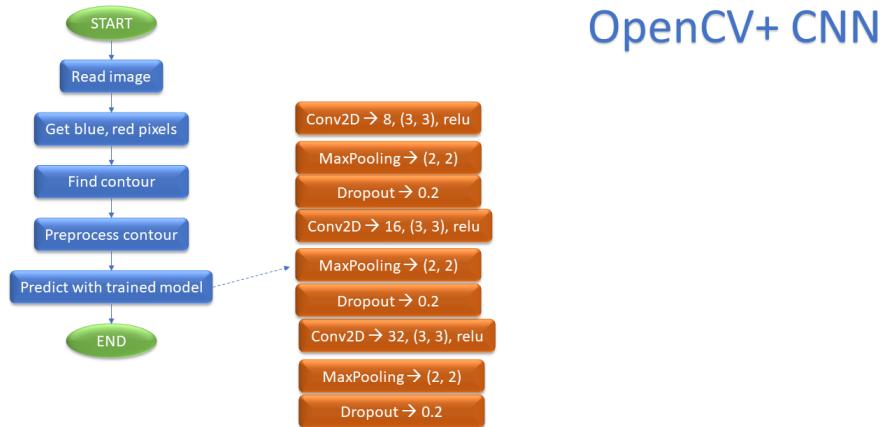
3.3.2 Nhận diện làn đường bằng Deep learning (CNN)



Hình 3.12. Flowchart xử lý sử dụng CNN

- Áp dụng mạng neural với 2 loại: Conv2D và MaxPooling
- Trước khi đưa vào mạng, đưa qua một hệ thống tiền xử lý để lọc ra những feature đặc trưng tốt nhất

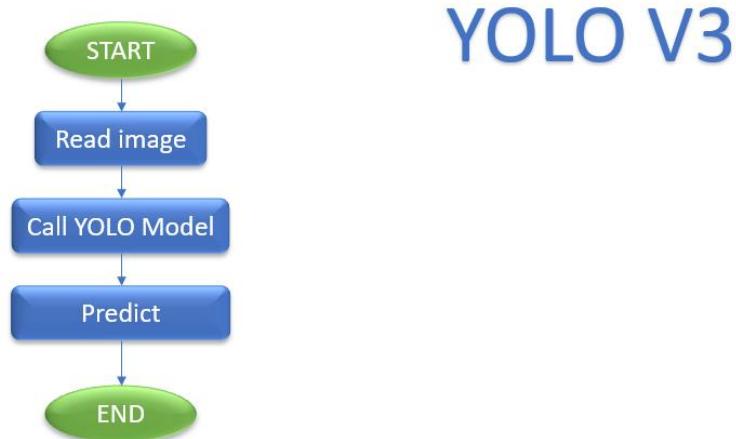
3.3.3 Phát hiện và nhận diện biển báo bằng OpenCV và CNN [32]



Hình 3.13. Flowchart xử lý sử dụng OpenCV và CNN

- Áp dụng mạng neural với 3 loại: Conv2D, MaxPooling và Dropout
- Trước khi sử dụng mạng, cần phân tích khu vực biển báo bằng cách sử dụng OpenCV với những thuật toán căn bản.

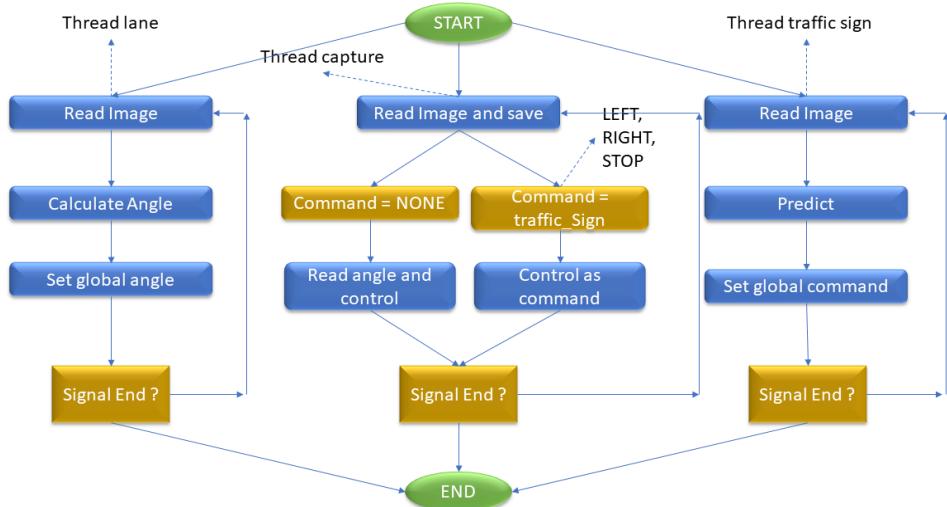
3.3.4 Phát hiện và nhận diện biển báo bằng YOLO v3



Hình 3.14. Flowchar xử lý sử dụng YOLO v3

- Sử dụng mạng neural có sẵn của YOLO v3 cho vật thể

3.3.5 Chương trình điều khiển đa luồng



Hình 3.15. Flow chart xử lý sử dụng đa luồng

- Sử dụng kĩ thuật chia luồng (3 luồng):
- Luồng đọc ảnh
- Luồng xử lý góc và tốc độ
- Luồng xử lý biến báo

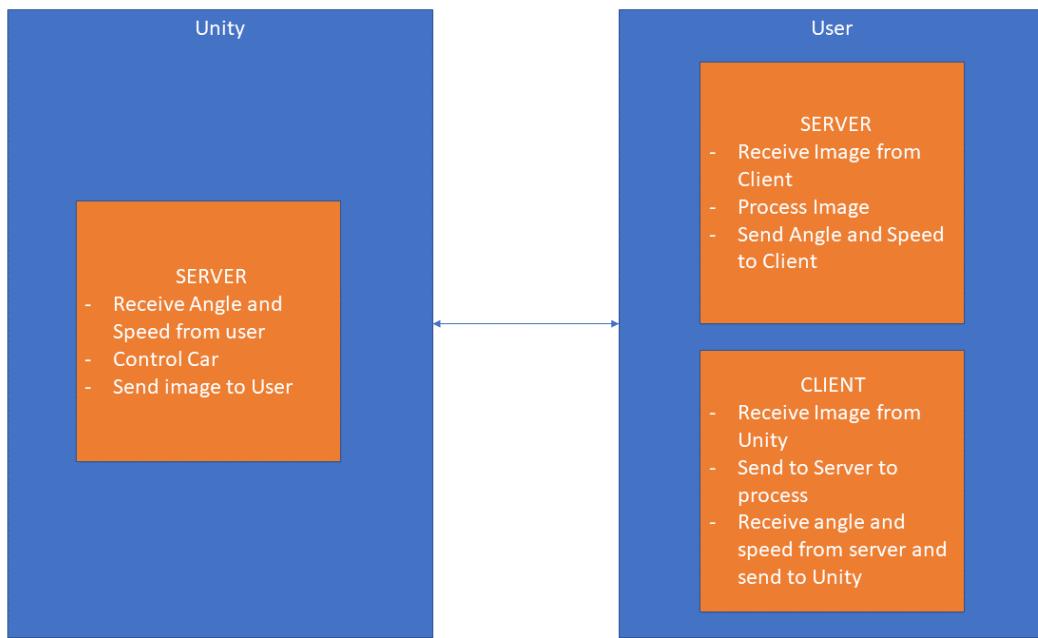
CHƯƠNG 4. KẾT QUẢ

4.1 Chương trình mô phỏng Unity

- Cấu hình máy sử dụng:
 - o Intel Core i7 9750H
 - o Gforce GTX 1050 Ti 4GB
 - o Ram 16Gb bus 2666
 - o SSD 128Gb
- Kết quả:
 - o Sa hình: Bao gồm đường thẳng, đường cong nhỏ/lớn (góc 15°, 30°, 90°). Ngoài ra còn có đường dốc. Bao gồm màu cỏ bên ngoài, lòng đường màu đen và 2 vạch kẻ màu trắng, mô phỏng theo đường thật bên ngoài. Đường có ngã 3, ngã tư
 - o Biển báo: Bao gồm 3 loại: Cấm rẽ trái, cấm rẽ phải và hiệu lệnh đi thẳng. Màu sắc được lấy từ màu của biển báo thật. Tuy nhiên hình ảnh thu được sẽ rõ hơn so với thực tế và chiều cao của biển phù hợp với camera mô phỏng, khác so với thực tế.
 - o Góc nhìn camera: Góc nhìn camera được đặt phù hợp với nhiều thuật toán (contour, sliding window, CNN, YOLO). Bên cạnh đó camera được đặt để cho có thể thu được đầy đủ hình ảnh của biển báo.
 - o Ánh sáng: Ánh sáng được mô phỏng có bóng cây như thực tế, tuy nhiên chỉ mô phỏng vào buổi sáng, Không có mô phỏng với ánh đèn vào buổi tối
- Do phần mềm liên tục render hình ảnh ra file cho nên sẽ tiêu tốn nhiều tài nguyên của máy. Với window 10, phần mềm bị đứng khi CPU không kịp xử lý nhiều tác vụ (Trường hợp này xảy ra 1/20, 20 lần chạy thì bị 1 lần). Với Linux máy ảo, khi mở phần mềm mà tối đen có nghĩa là phần mềm bị crash,

cần bật lại (Tần suất 1/10, 10 lần chạy thì bị 1 lần). Qua tìm hiểu, vấn đề này xảy ra phần lớn do tài nguyên của máy, máy mạnh hơn thì tần suất sẽ giảm thiểu.

4.1.1 Mô hình chương trình



Hình 4.1. Mô hình chương trình điều khiển Unity

- Unity đảm nhận nhiệm vụ chạy hình ảnh thực tế
- User sẽ đảm nhận nhiệm vụ xử lý (phần này dành cho người dùng tự do viết chương trình xử lý)

4.2 Liên kết giữa Python/C++ (Client) và C# (Unity Server)

- Quy tắc Json
 - o { "speed":0,"angle":0,"request":NONE}
 - o Speed và angle kiểu dữ liệu số nguyên (int)
 - o Request bao gồm:
 - NONE: không cần trả về
 - SPEED: trả về tốc độ

- ANGLE: trả về góc bẻ lái
- C++ Client:
 - Cần thay đổi đường dẫn file ảnh thời gian thực
 - Cần thư viện Cmake
 - Khi code thay đổi cần make lại
 - Hoạt động ổn định trong môi trường Linux (Ubuntu 18.04)
 - Cần cấp quyền cho file thực thi bằng lệnh chmod
- Python Client:
 - Hoạt động ổn định với Python 3.7.1
 - Cần thay đổi đường dẫn file ảnh thời gian thực
 - Hoạt động ổn định trong môi trường Window 10 và Linux (Ubuntu 18.04)
 - Cần chạy sudo nếu bị đòi quyền truy cập ảnh trên linux
- IP và Port
 - IP sẽ được lấy tự động theo family address
 - Port mặc định: 9999

4.3 Unity Server

- Class Checkpoint manager
 - Quản lý checkpoint nhiều nhất đạt được
 - Kết quả checkpoint cuối cùng sẽ được dùng từ class này
- Class LapTimeManager
 - Quản lý thời gian chạy còn lại
 - Quản lý thời gian chạy ngắn nhất
 - Kết quả thời gian cuối cùng sẽ được dùng từ class này
- Class Player
 - Render ảnh từ camera và lưu vào thư mục snapshots

- Class Server Controller
 - o Quản lý giao tiếp giữa server và client
 - o Nhận lệnh điều khiển từ client và gọi class Car Controller thực hiện
- Class Car Controller
 - o Quản lý những thuật toán điều khiển vật lý của xe
 - o Điều khiển xe với 2 thông số gia tốc và góc đánh lái

4.4 Client

- Sử dụng thư viện socket và thư viện threading cho ngôn ngữ python
- Sử dụng thư viện socket.h và thread cho ngôn ngữ C++



Hình 4.2. Chương trình điều khiển sử dụng YOLOv3 và phần mềm mô phỏng Unity



Hình 4.3. Chương trình sử dụng OpenCV và CNN

```
speed now is : {0} 0
(30, 320, 3)
(30, 320)
417.5
6155.5
2379.0
count= 1
X = 194.0
hieu so 34.0
goc lech = 25.542485129802884
```

Hình 4.4. Số liệu khi xử lí sử dụng OpenCV và CNN

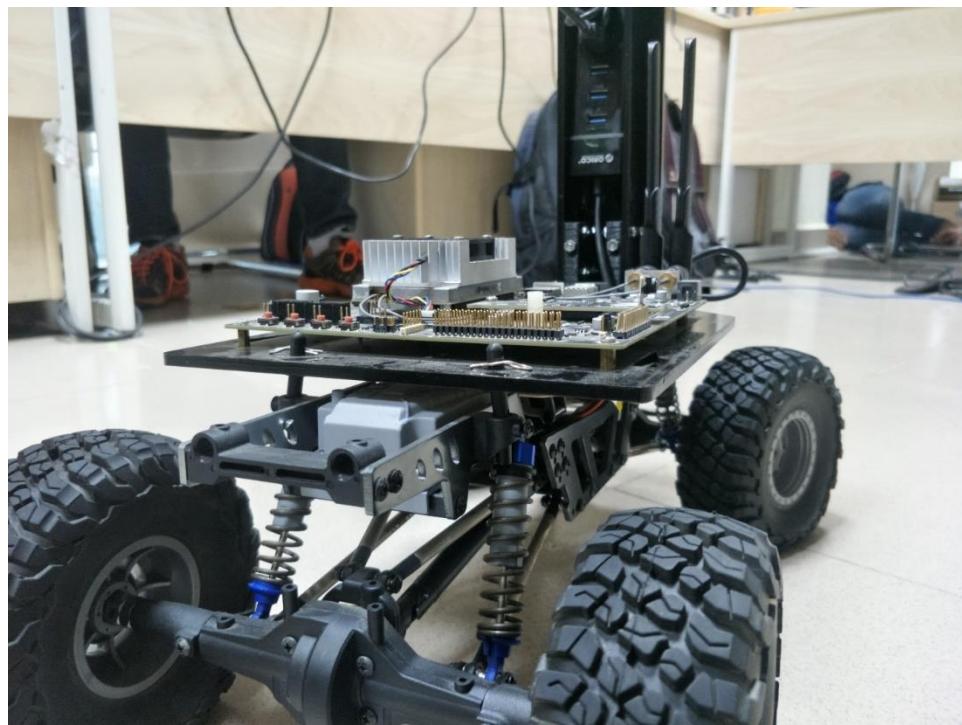
4.5 Mô hình xe RC

4.5.1 Tổng thể mô hình xe



Hình 4.5. Mô hình xe nhìn từ trên xuống

- Đế board, nắp hộp được cắt CNC với mica
- Đế camera được in 3D



Hình 4.6. Hệ thống lò xo

- Lò xo được mua tại những cửa hàng vật liệu sắt

4.5.2 Camera



Hình 4.7. Camera Orbbee Astra 3D

- Lựa chọn camera Orbbee Astra 3D vì:
 - Có góc nhìn tầm trung: $60^{\circ}\text{H} \times 49.5^{\circ}\text{V} \times 73^{\circ}\text{D}$
 - Giá cả trong khoảng giữa của thị trường
 - Sử dụng OpenNI

- Đạt được 30fps với độ phân giải 640 x 480

4.5.3 Driver shield



Hình 4.8. Driver shield

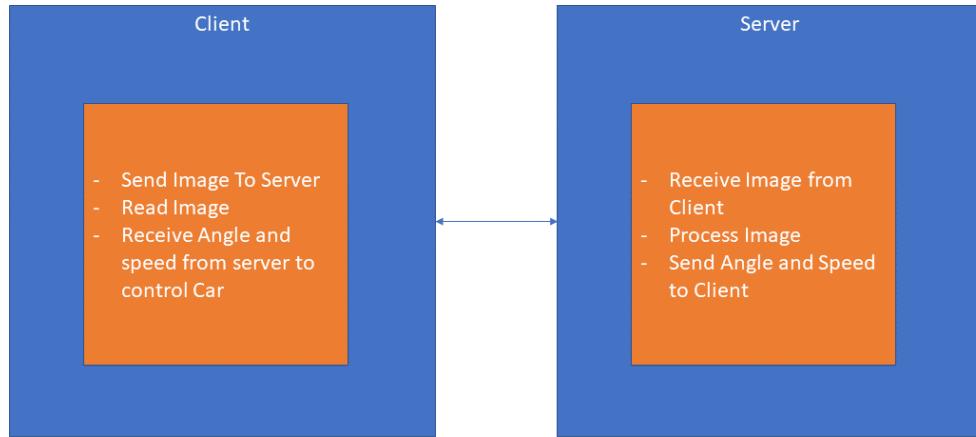
- Quá trình làm:
 - Thiết kế Schematic dựa trên những chân vật lý của Jetson TX
 - Layout PCB 2 mặt
 - Gửi đặt mạch tại Hà Nội
 - Tìm và test linh kiện
 - Hàn, hoàn thiện và kiểm thử bằng đo thông và đo nguồn

4.5.4 Kết quả

- Tốc độ:
 - Tốc độ nhanh nhất 22km/h
 - Tốc độ chậm nhất 10km/h với góc 30 và 0km/h với trường hợp đang xử lý biển báo

- Vật cản gồ ghề:
 - o Với vật cản cao 8cm được đặt dưới tâm thảm đường làm cho đường gồ ghề, nhấp nhô, xe xử lý được độ nhiễu ảnh khi camera bị rung
- Vật cản gây nhiễu:
 - o Đặt một cái đĩa nhựa mỏng màu trắng vào trong đường, xe giải quyết được, đi qua cái đĩa và vẫn đi giữa đường
- Chương trình điều khiển:
 - o Điều khiển từ xa thông qua phần mềm remote của ubuntu (ubuntu to ubuntu)
 - o Sử dụng wifi làm sóng trung gian để kết nối
 - o Chương trình điều khiển sử dụng nút nhấn vật lý để khởi động/Dừng xe
 - o Hoàn toàn sử dụng lệnh trên terminal
- Điện năng tiêu thụ:
 - o Đối với board Jetson
 - 2000mAH, 19V => 38mWh
 - Với 2 viên pin lipo 7200mAH 12V và 4800mAH 7.4V, xài được khoảng 6 giờ
 - o Đối với động cơ ESC
 - 7.4V, 40A => 296W
 - Với 1 viên pin lipo 4800mAH 7.4V xài được khoảng 25 phút
- Độ chính xác trên đường:
 - o Xe đi vào chính giữa đường với đường cong từ 0° - 15°
 - o Khi đường cong lớn hơn 15° xe có xu hướng ra sát vạch kẻ đường và điều chỉnh dần dần về chính giữa đường
- Độ ổn định:
 - o Xác định và đo đạc bằng stress test.

4.5.5 Mô hình chương trình điều khiển



Hình 4.9. Mô hình chương trình điều khiển xe RC Car

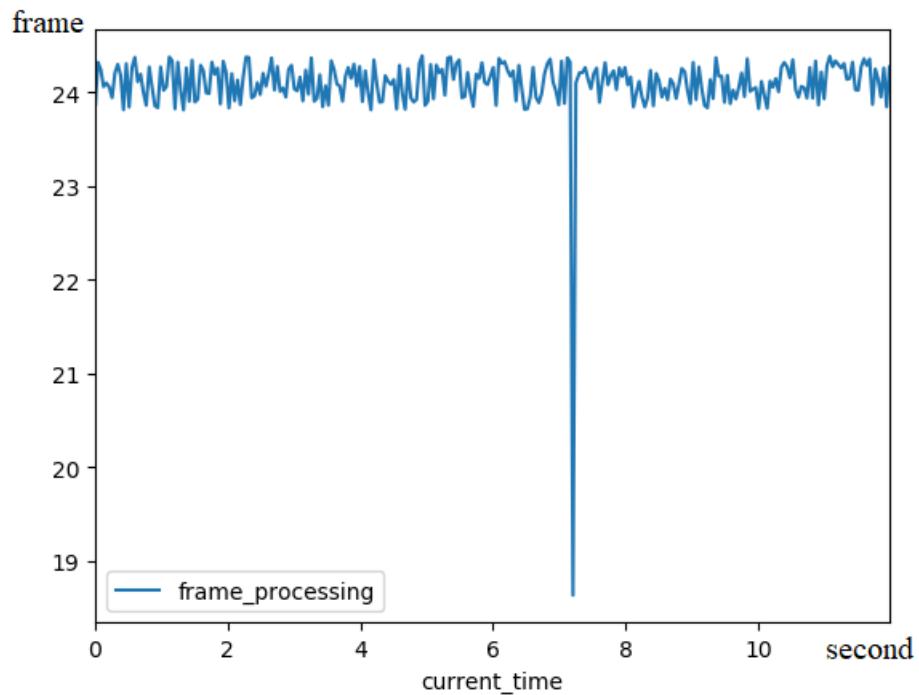
- Mô hình quản lý tài nguyên và xử lý dành cho xe RC Car
- Server là một bộ máy mạnh để xử lý
- Client đảm nhận nhiệm vụ điều khiển xe realtime

4.6 Chương trình điều khiển xe không sử dụng multithreading

4.6.1 Opencv và CNN

- Với OpenCV:
 - o Nhận diện, hoạt động tốt với 4 vùng màu: trắng, đỏ, xanh lá, xanh dương
 - o Có chế độ lưu lại lịch sử để bám góc cong lớn khi xe mất 1 bên vạch kẻ đường
- Với CNN:
 - o Sử dụng 11 layer: 6 layers Conv2D, 3 layers MaxPooling2D, 2 layers Dense

- Những activation sử dụng: relu, softmax
- Sử dụng tiền xử lý hình ảnh để làm tối ưu các đặc trưng, gộp phần tăng độ chính xác
- Độ chính xác CNN: 97%
- Số lượng mẫu huấn luyện: 1000 mẫu
- Số lượng mẫu thử: 300 mẫu
- Biểu đồ thời gian xử lí:



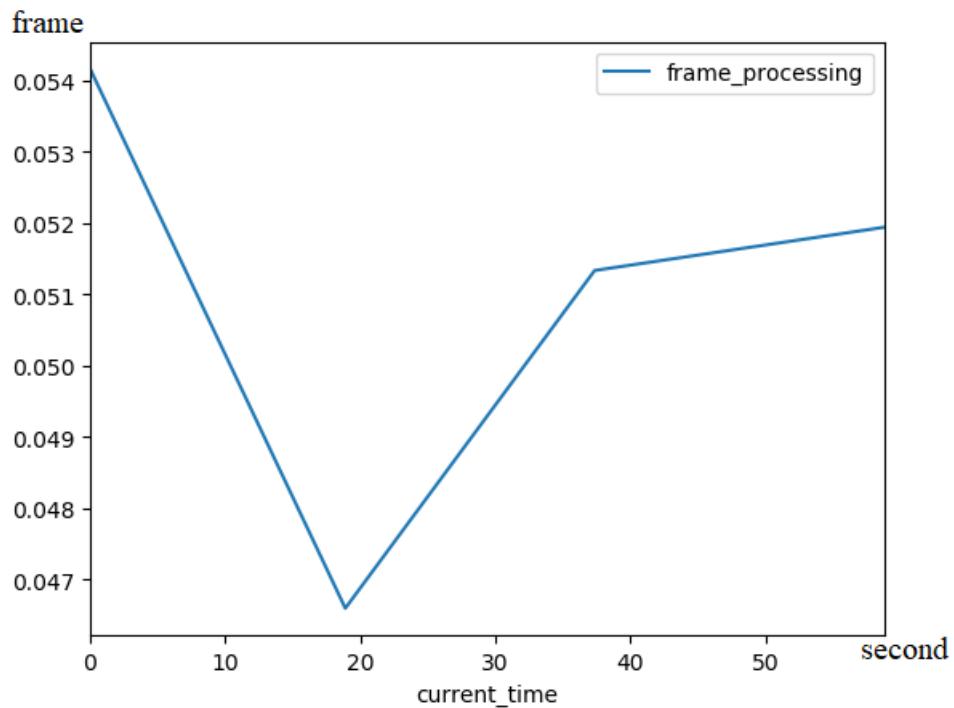
Hình 4.10. Biểu đồ thời gian xử lí với OpenCV và CNN

- Trong khoảng 6-8s có 1 lần chương trình xử lý biển báo nên tốc độ giảm xuống dưới 19fps

4.6.2 YOLO v3

- Độ chính xác:
 - Góc lệch của xe: 98%
 - Biển báo: 97%

- Số lượng mẫu huấn luyện (Bao gồm những hình chỉ có đường và những hình vừa có đường vừa có biển báo):
 - Biển cấm rẽ trái: 80
 - Biển cấm rẽ phải: 80
 - Biển đi thẳng: 90
- Số lượng mẫu thử:
 - Chỉ có đường: 30
 - Đường và biển cấm rẽ trái: 20
 - Đường và biển cấm rẽ phải: 20
 - Đường và biển đi thẳng: 10
- Biểu đồ thời gian xử lí:

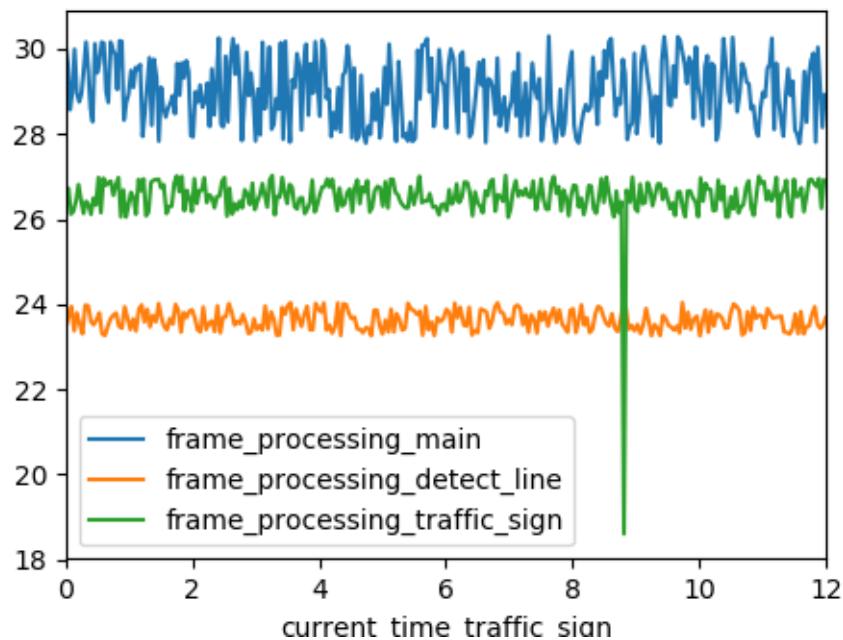


Hình 4.11. Biểu đồ thời gian xử lí với YOLO v3 (đơn vị fps)

4.7 Chương trình điều khiển xe sử dụng multithreading

4.7.1 OpenCV và CNN

- Tương tự với non-multithreading
- Độ chính xác CNN: 97%
- Số lượng mẫu huấn luyện: 1000 mẫu
- Số lượng mẫu thử: 300 mẫu
- Biểu đồ thời gian xử lý:



Hình 4.12. Biểu đồ thời gian xử lí với OpenCV và CNN (Multithreading)

- Khoảng thời gian 8-10s chương trình xử lý biển báo, tốc độ xử lý thread biển báo giảm xuống dưới 19fps
- Nhưng tổng thể chương trình vẫn đáp ứng realtime

4.8 So sánh giữa OpenCV + CNN và YOLOv3

- Cả 2 thuật toán đều cho độ chính xác > 95%
- OpenCV + CNN tốc độ xử lý 24-28 fps

- YOLOv3 tốc độ xử lý 0.047-0.054 fps
- Nguyên nhân YOLOv3 rất chậm so với OpenCV + CNN:
 - o Sử dụng model đầy đủ của YOLO → Liên kết mạng rất lớn → Performance giảm
 - o YOLO cùng lúc cho ra 2 objects là đường và biển vì vậy không chia luồng chạy riêng biệt 2 objects được → Multithreading không đáp ứng được

4.9 So sánh Tiny YOLO v3 và Fully-Connected YOLO v3

	Tiny YOLO V3	Fully-Connected YOLO V3
Accuracy	97%	76%
fps	~ 0.047-0.054 fps	~ 19-21 fps

Bảng 4.1. Bảng kết quả và tốc độ xử lý giữa Tiny Yolo và Full Yolo

➔ Tuy nhiên với yêu cầu là độ chính xác nhất có thể nên Tiny YOLO không được đưa vào sử dụng thực tế

4.10 So sánh chương trình điều khiển non-multithreading và multithreading

	Non-multithreading	Multithreading
Accuracy	97%	97%
fps	~ 19-24 fps	~ 24-28 fps

Bảng 4.2. Bảng kết quả độ chính xác và tốc độ xử lý (fps) một luồng và đa luồng

➔ Với chương trình sử dụng multithreading, tốc độ xử lý đảm bảo được mức độ real-time để kịp thời xử lý những tình huống tức thời.

4.11 Stress test trên xe RC

- Pin đầy, xe chạy ổn định hơn 7 vòng, với mỗi vòng có chu vi: 35m
- Đặt vật cản cao 8cm, chương trình xử lý được nhiều và điều chỉnh lại hướng đi ổn định

- Đặt những vật có màu trắng vào giữa đường để gây nhiễu, chương trình xử lý được nhiễu này, không chạy ra ngoài hay lạng.
- Chạy thử nghiệm chương trình với thời gian lớn hơn 1h không xảy ra crash hay treo.
- Tổng thời gian thử nghiệm xe: 58 giờ, mỗi ngày thử nghiệm từ 4-6 giờ

CHƯƠNG 5. KẾT LUẬN

5.1 Kết luận

Chương trình cơ bản hoàn thành. Xe, shield driver, Board Jetson Nvidia đã hoạt động bình thường, quá trình mô phỏng đã trọn vẹn.

Nhóm đã tiến hành thử nghiệm stress test trên xe (chạy thời gian dài, chạy quãng đường xa, chướng ngại vật), xe đã vượt qua được những bài test này. Với việc triển khai và thực hiện đa luồng vào chương trình, khả năng xử lý thời gian thực tăng lên đáng kể, đây là mục tiêu lớn của nhóm.

Như vậy điểm mạnh của đề tài là mức độ ổn định của chương trình với hệ thống đa luồng. Ngoài ra quá trình xử lý đa luồng thời gian thực được áp dụng đã làm gia tăng đáng kể hiệu năng của board Nvidia Jetson TX2.

Tuy nhiên với giới hạn thời gian và kinh phí, nhóm chưa thể thực hiện trên những xe lớn và môi trường outdoor. Bên cạnh đó, thuật toán nhận diện sử dụng deep learning vẫn còn hạn chế so với những thuật toán đã được công bố trên thế giới. Với GPU hạn hẹp, nhóm vẫn chưa thể kiểm chứng được tốc độ xử lý của thuật toán YOLO v3.

Với những chức năng cơ bản, mô hình xe như nhóm đã trình bày, ước tính hệ thống khoảng 30.000.000 VNĐ, số tiền tương đối ít so với những hệ thống bên ngoài.

5.2 Kiến nghị

Với thời gian tìm hiểu và phát triển đề tài trong khoảng 5 tháng, chắc chắn sẽ vẫn còn những trường hợp lỗi chưa được phát hiện. Do đó, đề tài cần có thêm nhiều thời gian để tìm hiểu và kiểm thử.

Đề tài cũng hướng đến việc ứng dụng lên những loại xe cỡ lớn (xe golf, xe điện, ...). Áp dụng thuật toán YOLO vào GPU mạnh hơn, và xử lý mượt mà trong môi trường outdoor cũng là một mục tiêu lớn.

TÀI LIỆU THAM KHẢO

- [1] autopro, "autopro," [Online]. Available: <https://autopro.com.vn/quoc-te/xe-tu-hanh-cua-google-co-lien-quan-den-11-vu-tai-nan-20150514113910771.chn>. [Accessed 2019].
- [2] zing, "zing," [Online]. Available: <https://news.zing.vn/apple-thu-nghiem-xe-tu-hanh-nhieu-hon-ca-tesla-uber-post842879.html>. [Accessed 2019].
- [3] "techcrunch," 2019. [Online]. Available: <https://techcrunch.com/2016/04/26/google-uber-lyft-join-automakers-in-self-driving-car-lobby/>. [Accessed 2019].
- [4] "theverge," 2019. [Online]. Available: <https://www.theverge.com/2017/9/20/16341478/tesla-amd-chip-self-driving-car>. [Accessed 2019].
- [5] "Unity Introduction," [Online]. Available: <https://viblo.asia/p/gioi-thieu-ve-unity-engine-game-engine-pho-bien-nhat-hien-nay-V3m5WBj8lO7>. [Accessed 2019].
- [6] T. X. Chu, "viblo asia," [Online]. Available: <https://viblo.asia/p/gioi-thieu-ve-unity-engine-game-engine-pho-bien-nhat-hien-nay-V3m5WBj8lO7>. [Accessed 2019].
- [7] "socket," [Online]. Available: <https://blog.tinohost.com/socket-la-gi/>. [Accessed 2019].
- [8] "vietrc," RGT Racing, 2019. [Online]. Available: <http://www.vietrc.com/rgt-rock-hammer>. [Accessed 2019].
- [9] "danhgiaxe," [Online]. Available: <https://www.danhgiaxe.com/he-thong-dan-dong-4-banh-toan-thoi-gian-doi-xung-cua-subaru-7780>. [Accessed 2019].

- [10 "myrcsaigon," [Online]. Available: <http://myrcsaigon.com/can-nam-ro-ve-esc-becca-va-ubec/>. [Accessed 2019].
- [11 "xomrc," [Online]. Available: <http://xomrc.com/kinh-nghiem-mua-hang/dong-co-khong-choi-than-la-gi-bai-viet-nay-se-giup-ban-hieu-ro-hon-ve-don-co-khong-choi-than/>. [Accessed 2019].
- [12 "solarstore," [Online]. Available: <https://solarstore.vn/khai-niem-dong-co-servo-la-gi/>. [Accessed 2019].
- [13 "bkaero," [Online]. Available: <https://bkaero.vn/nhung-kien-thuc-can-biet-ve-pink-lipo/>. [Accessed 2019].
- [14 "learnopencv," [Online]. Available: <https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/>. [Accessed 2019].
- [15 "Crop image," [Online]. Available: <https://www.pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/>. [Accessed 2019].
- [16 "contour_opencv," [Online]. Available: <https://medium.com/@ricardo.zuccolo/self-driving-cars-opencv-and-svm-machine-learning-with-scikit-learn-for-vehicle-detection-on-the-bf88860e055a>. [Accessed 2019].
- [17 "opencv-python-tutroals," [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html. [Accessed 2019].
- [18 "opencv," [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html
- .

- [19 "laerd," [Online]. Available: <https://statistics.laerd.com/statistical-guides/understanding-histograms.php>. [Accessed 2019].
- [20 "pythonprogramming," [Online]. Available: <https://pythonprogramming.net/thresholding-image-analysis-python-opencv-tutorial/>. [Accessed 2019].
- [21 "pyimagesearch-conv2d," [Online]. Available: <https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/>.
- [22 "jessicayung," [Online]. Available: <https://www.jessicayung.com/explaining-tensorflow-code-for-a-convolutional-neural-network/>. [Accessed 2019].
- [23 "MaxPooling," [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_-_Pooling. [Accessed 2019].
- [24 "quora flatten," [Online]. Available: <https://www.quora.com/What-is-the-meaning-of-flattening-step-in-a-convolutional-neural-network>. [Accessed 2019].
- [25 "deeplizard zero padding," [Online]. Available: http://deeplizard.com/learn/video/qSTv_m-KFk0. [Accessed 2019].
- [26 "difference-between-a-batch-and-an-epoch," [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>. [Accessed 2019].
- [27 "YOLO V3," [Online]. Available: <https://www.cyberailab.com/home/a-closer-look-at-yolov3>. [Accessed 2019].
- [28 "K-means Clustering," [Online]. Available: <http://www.vlfeat.org/overview/kmeans.html>. [Accessed 2019].

- [29 "Nvidia-jetson-tx2," [Online]. Available: <https://developer.nvidia.com/embedded/jetson-tx2-developer-kit>. [Accessed 2019].]
- [30 "Orbbec Astra," [Online]. Available: <https://orbbec3d.com/>. [Accessed 2019].]
- [31 "PCA9685," [Online]. Available: https://sea.banggood.com/PCA9685-16-Channel-12-bit-PWM-Servo-Motor-Driver-I2C-Module-For-Arduino-Robot-p-1170343.html?cur_warehouse=CN. [Accessed 2019].]
- [32 "Tensorflow Introduction," [Online]. Available: <https://kipalog.com/posts/Bat-dau-voi-Machine-Learning-thong-qua-Tensorflow--Phan-I-2>. [Accessed 2019].]
- [33 "Scikit-image," [Online]. Available: <https://scikit-image.org/>. [Accessed 2019].]
- [34 n. c. thanh, "viblo asia," [Online]. Available: <https://viblo.asia/p/da-luong-trong-python-multithreading-WAyK8MO6ZxX>. [Accessed 2019].]
- [35 "traffic sign," [Online]. Available: <https://chsasank.github.io/keras-tutorial.html>. [Accessed 2019].]
- [36 "opencv," 2019. [Online]. Available: <https://opencv.org/>.]
- [37 "xe tu hanh," fpt, 2019. [Online]. Available: <https://vnexpress.net/so-hoa/hanh-trinh-dau-tien-tren-xe-tu-hanh-cua-fpt-3664552.html>. [Accessed 2019].]
- [38 "vicotech," 2019. [Online]. Available: <https://vicotech.com.vn/he-thong-xe-tu-hanh-agv>. [Accessed 2019].]
- [39 vlfeat, "vlfeat," [Online]. Available: <https://www.vlfeat.org/overview/kmeans.html>. [Accessed 2019].]

PHỤ LỤC

Resize:	cv2.resize(image, (320, 160))
Crop image	frame = frame[130:160, 0:320]
Get white pixels	lower = np.uint8([83, 0, 0])
*	upper = np.uint8([180, 255, 255])
*	white_mask = cv2.inRange(hsv, lower,
*	upper)
*	result = cv2.bitwise_and(frame, frame,
*	mask = white_mask)
Threshold	ret, thresh = cv2.threshold(gray ,140
*	,255, cv2.THRESH_BINARY_INV)
Contour	ima, contours, hier =
*	cv2.findContours(thresh.copy(),
*	cv2.RETR_TREE,
*	cv2.CHAIN_APPROX_SIMPLE)
Dilate	cv2.dilate(roi, (3, 3))

