

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HCM

KHOA CÔNG NGHỆ CƠ KHÍ
BỘ MÔN CƠ ĐIỆN TỬ



ĐỒ ÁN TỐT NGHIỆP

**ROBOT TỰ HÀNH PHÁT HIỆN LÀN ĐƯỜNG
SỬ DỤNG KỸ THUẬT XỬ LÝ ẢNH**

Giảng viên hướng dẫn: **TS. NGUYỄN VIỄN QUỐC**

Sinh viên thực hiện:

TRẦN NGỌC DÂN 13097531

PHAN NGUYỄN NGỌC HIỀN 13088151

LÊ HOÀI THƯƠNG 13081051

Thông tin liên lạc:

ĐT: 01698 554 576

Email: pnnhsn20@gmail.com

TP. Hồ Chí Minh, ngày 29 tháng 11 năm 2018

Tp.HCM, ngày tháng năm 2018

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên SV 1: TRẦN NGỌC DÂN MSSV: 13097531 Lớp: DHC9B

Họ tên SV 2: PHAN NGUYỄN NGỌC HIỀN MSSV: 13088151 Lớp: DHC9B

Họ tên SV 3: LÊ HOÀI THƯƠNG MSSV: 13081051 Lớp: DHC9B

1. Tên đề tài: ROBOT TỰ HÀNH PHÁT HIỆN LANE SỬ DỤNG KỸ THUẬT
XỬ LÝ ẢNH

2. Nhiệm vụ (yêu cầu về nội dung và số liệu ban đầu):

Đồ án thực hiện việc phát hiện làn đường dùng kỹ thuật xử lý ảnh chạy trên nền
tản Linux với phiên bản Raspbian OS trên board nhúng Raspberry Pi 3.

3. Ngày giao nhiệm vụ ĐACN: ... tháng ... năm 2018

4. Ngày kiểm tra giữa kỳ (50%): .../.../2018

5. Ngày hoàn thành: .../.../2018

6. Giáo viên hướng dẫn:

1.....

Phần hướng dẫn:

.....

2.....

.....

3.....

.....

Ngày tháng năm 2018

CHỦ NHIỆM BỘ MÔN

GV HƯỚNG DẪN

TRƯỞNG KHOA

LỜI MỞ ĐẦU

Với niềm đam mê và thích thú đối với lĩnh vực xử lý ảnh và trí tuệ nhân tạo đồng thời muôn tìm hiểu và học hỏi cái mới. Quan trọng hơn hết là vấn đề giao thông ở Việt Nam còn chưa tốt, còn ùn tắc, tai nạn do nhiều lý do khác nhau trong đó có ý thức con người. Nhóm đã lựa chọn đề tài “robot tự hành phát hiện làn đường dùng kỹ thuật xử lý ảnh. Với mục đích muốn áp dụng ứng dụng xử lý ảnh và trí tuệ nhân tạo vào giao thông ở Việt Nam. Nhằm xây dựng một hệ thống giao thông mới hiện đại và an toàn.

Đề tài có gồm 13 chương, gồm:

- Chương 1 và 2 là giới thiệu tổng quan về đề tài
- Chương 3 nói về nguyên lý xử lý ảnh
- Chương 4 nêu lý thuyết về không gian màu
- Chương 5 trình bày khái niệm và ví dụ về threshoding trong xử lý ảnh
- Chương 6 hướng dẫn cách tính bán kính cong tại một điểm
- Chương 7 cách sử dụng mô hình hồi quy tuyến tính cho hàm bậc hai
- Chương 8 là biểu diễn dạng bird’eye cho hình ảnh
- Chương 9 là quá trình tìm Best Fit Line
- Chương 10 là xây dựng mô hình toán học và mô phỏng trên Simulink
- Chương 11 giới thiệu máy tính nhúng Raspberry, ngôn ngữ Python và thư viện xử lý ảnh OpenCV - Python
- Chương 12 là phần Hardware và bản vẽ lắp của robot
- Chương 13 là phần Software gồm thuật toán điều khiển và code

Hiện tại, phạm vi của đề tài chỉ dừng lại ở phần phát hiện làn đường, để hoàn thiện hơn ta cần áp dụng kiến thức Machine Learning và Deep Learning trong nhận dạng biển báo cũng như phát hiện vật cản. Nhưng sau đề tài này nhóm đã học được nhiều điều thú vị và có kiến thức để làm thực tiễn.

LỜI CẢM ƠN

Để hoàn thành công việc một cách dễ dàng và nhanh chóng nhất thì sự giúp đỡ của mọi người là điều vô cùng đáng quý và trân trọng. Trong thời gian từ khi bắt đầu bước chân vào giảng đường đại học cho đến xuyên suốt quá trình thực hiện đồ án, chúng em đã nhận được rất nhiều sự giúp đỡ và quan tâm của các thầy cô, bạn bè.

Chúng em xin cảm ơn các thầy cô trong Trường Đại Học Công Nghiệp Thành phố Hồ Chí Minh nói chung và các thầy cô trong Khoa Công nghệ Cơ Khí nói riêng đã truyền đạt kiến thức các môn học đại cương cũng như các môn cơ sở ngành và chuyên ngành giúp chúng em có cơ sở vững chắc để có thể hoàn thành đồ án tốt nghiệp.

Đặc biệt chúng em xin chân thành cảm ơn thầy Nguyễn Viễn Quốc đã hướng dẫn và giúp đỡ chúng em một cách tối đa để chúng em có thể hoàn thành được đồ án tốt nghiệp. Chúng em luôn nhận thức được rằng, nếu không có sự giúp đỡ tận tình của thầy thì sẽ rất khó khăn trong quá trình thực hiện đồ án.

Trong quá trình thực hiện đồ án này và quá trình viết báo cáo chúng em không thể tránh khỏi những sai sót, mặc dù vậy chúng em đã cố gắng hết sức để hoàn thành đồ án tốt nghiệp nên chúng em mong rằng thầy cô sẽ thông cảm và bỏ qua những sai sót đó, chúng em luôn luôn mong muốn nhận được những ý kiến đóng góp từ các thầy cô để bổ sung và nâng cao kiến thức của mình.

Chúng em xin chân thành cảm ơn!

Tp. Hồ Chí Minh, ngày 29 tháng 11 năm 2018

Nhóm sinh viên thực hiện

ROBOT TỰ HÀNH PHÁT HIỆN LÀN ĐƯỜNG DÙNG KỸ THUẬT XỬ LÝ ẢNH

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

MỤC LỤC

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

LỜI MỞ ĐẦU

LỜI CẢM ƠN

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

MỤC LỤC

DANH SÁCH CÁC BẢNG BIỂU, HÌNH VẼ VÀ SƠ ĐỒ

CHƯƠNG 1: GIỚI THIỆU VÀ ĐẶT VẤN ĐỀ	1
1.1 GIỚI THIỆU	1
1.2 GIỚI HẠN VÀ MỤC TIÊU ĐỀ TÀI	1
CHƯƠNG 2: TÔNG QUAN	2
CHƯƠNG 3: NGUYÊN LÝ XỬ LÝ ẢNH	3
3.1 GIỚI THIỆU	3
3.2 ĐỊNH NGHĨ MỘT ẢNH SỐ (DIGITAL IMAGE)	3
CHƯƠNG 4: LÝ THUYẾT VỀ KHÔNG GIAN MÀU	4
4.1 KHÁI NIỆM	4
4.2 MỘT SỐ KHÔNG GIAN MÀU	4
4.2.1 RGB	4
4.2.2 HSV, HSL	4
CHƯƠNG 5: NGƯỠNG CỦA HÌNH ẢNH – THRESHOLDING IMAGE	5
5.1 GIỚI THIỆU	5
5.2 THRESHOLDING TRONG OPENCV – PYTHON, [2]	5
CHƯƠNG 6: TÍNH BÁN KÍNH ĐƯỜNG CONG TẠI MỘT ĐIỂM	8
6.1 KHÁI NIỆM	8
6.2 VÍ DỤ	8
CHƯƠNG 7: POLYNOMIAL REGRESSION.....	10
7.1 LÝ THUYẾT	10
7.2 THỰC HIỆN	11
7.3 KẾT QUẢ	12
CHƯƠNG 8: BIRD'S EYE VIEW	13
8.1 GIỚI THIỆU	13
8.2 ẢNH BIRD'S EYE TRONG OPENCV - PYTHON	15
CHƯƠNG 9: TÌM VỊ TRÍ CỦA XE	17
9.1 NỘI DUNG	17
9.2 VẤN ĐỀ KHI TÌM ĐƯỜNG HỒI QUY	18
9.3 TÍNH GIÁ TRỊ OFFSET:	23
CHƯƠNG 10: MÔ HÌNH TOÁN HỌC CỦA HỆ THỐNG	24

10.1PHƯƠNG TRÌNH VI PHÂN	24
10.1.1 Môи quan hệ giữa θ và v của hai bánh.	24
10.1.2 Phương trình vi phân của động cơ DC.	26
10.1.3 Bộ điều khiển.	26
10.2MÔ PHỎNG TRÊN SIMULINK	27
10.3KẾT LUẬN	29
CHƯƠNG 11: RASPBERRY PI 3 AND OPENCV – PYTHON	30
11.1NGÔN NGỮ PYTHON, [3]	30
11.1.1 Giới thiệu	30
11.1.2 Đặc điểm	30
11.1.3 Các bản hiện thực	30
11.1.4 Trình thông dịch	31
11.1.5 Lệnh và cấu trúc điều khiển	31
11.1.6 Hệ thống kiểu dữ liệu	32
11.1.7 Module	32
11.1.8 Cú pháp	33
11.2CÀI ĐẶT HỆ ĐIỀU HÀNH RASBIAN CHO RASPBERRY PI 3B	36
11.3CÀI ĐẶT OPEN CV CHO RASPBERRY PI 3 MODEL B.	37
CHƯƠNG 12: HARDWARE	39
12.1GIỚI THIỆU	39
12.2SŌ ĐỒ KHỐI CỦA HỆ THỐNG	39
12.3SŌ ĐỒ NGUYÊN LÝ	39
12.4RASPBERRY PI 3 MODEL B.	40
12.4.1 Giới thiệu chung.	40
12.4.2 Thông số kỹ thuật	40
12.4.3 Ứng dụng	41
12.4.4 Ưu điểm.	41
12.4.5 Nhược điểm	41
12.5ADRUINO UNO BOARD	41
12.5.1 Khái niệm	41
12.5.2 Các loại Arduino thông dụng	42
12.5.3 Arduino Uno R3	42
12.6CÀU H (DRIVER L298D)	43
12.6.1 Nguyên lý cầu H	43
12.6.2 Điều khiển động cơ sử dụng L298D	44
12.7PIN LIPO	45

12.8 ĐỘNG CƠ DC 12V	46
12.9 PICAMERA MODULE	46
12.9.1 Giới thiệu	46
12.9.2 Thông số kỹ thuật	47
12.10 SERVO RC	47
12.10.1 Giới thiệu	47
12.10.2 Thông số kỹ thuật	47
12.11 CẢM BIẾN SIÊU ÂM SRF-05	47
12.11.1 Nguyên lý hoạt động	47
12.11.2 Cảm biến SRF-05	47
12.11.3 Thông số kỹ thuật của SRF - 05	48
12.11.4 Cách sử dụng SRF-05 và xác định khoảng cách bằng SRF-05	48
12.12 KẾT NỐI CÁC THIẾT BỊ	49
12.13 BẢN VẼ CHI TIẾT VÀ BẢN VẼ LẮP CỦA MÔ HÌNH	49
12.14 HÌNH ẢNH VỀ ROBOT CỦA NHÓM	63
CHƯƠNG 13: GIẢI THUẬT VÀ SOFTWARE	64
13.1 GIẢI THUẬT	64
13.2 KẾT QUẢ QUÁ TRÌNH XỬ LÝ ẢNH	65
13.2.1 Read Frame (frame size = (304,208)):	65
13.2.2 Filter Color:	66
13.2.3 Smoothing image (kernel = (11,11)):	66
13.2.4 Color warp image(warp size = (200, 80):	66
13.2.5 Thresholding image:	67
13.2.6 Histogram theo trục x:	67
13.2.7 Find best fit line:	67
13.3 CHƯƠNG TRÌNH	67
13.3.1 Chương trình Python trên Raspberry Pi 3	67
13.3.2 Chương trình C trên Arduino	77
TÀI LIỆU THAM KHẢO	82
PHỤ LỤC	83

DANH SÁCH CÁC BẢNG BIỂU, HÌNH VẼ VÀ SƠ ĐỒ

BẢNG

Bảng 5.1. Các loại thresholding thông dụng.....	5
Bảng 10.1. Các tham số của hệ thống	24
Bảng 12.1. Thông số kỹ thuật	42
Bảng 12.2. Tên chân sẽ kết nối với nhau	49

SƠ ĐỒ

Sơ đồ 10.1 Sơ đồ nguyên lý động cơ DC.....	26
Sơ đồ 10.2. Sơ đồ tả phương trình vi phân của động cơ DC	27
Sơ đồ 10.3. Quan hệ θ và v	27
Sơ đồ 10.4. Sơ đồ mô tả phương trình vi phân (1).....	27
Sơ đồ 10.5. Sơ đồ thực hiện nhiệm vụ tăng tốc cho một trong hai bánh xe theo điều kiện.....	28
Sơ đồ 10.6. Sơ đồ của hệ thống.....	28
Sơ đồ 12.1. Sơ đồ khối của hệ thống	39
Sơ đồ 13.1. Giải thuật xử lý ảnh	64
Sơ đồ 13.2. Giải thuật tìm best fit line	64
Sơ đồ 13.3. Giải thuật điều khiển robot	65

HÌNH

Hình 4.1. Không gian màu RGB	4
Hình 4.2. Không gian màu HSV	4
Hình 5.1. Ảnh gốc	6
Hình 5.2. Kết quả sử dụng các kiểu thresholding	7
Hình 6.1.Đường tròn tiếp xúc với đường cong tại một điểm.....	8
Hình 6.2. Tâm của đường tròn xấp xỉ	9
Hình 7.1. Kết quả tìm đường hồi quy bậc hai	12
Hình 8.1. Ảnh dạng Bird'eye (ảnh b)	13
Hình 8.2. Các góc vị trí của camera	13
Hình 8.3. Vị trí camera.....	14
Hình 8.4.Quá trình tạo ra một ảnh top – view (bird'eye).....	15
Hình 8.5. Perspective Transform	16
Hình 9.1. Ảnh gốc	17
Hình 9.2. Quá trình tìm best fit line	17
Hình 9.3. Kết quả tìm best fit line	18
Hình 9.4. Kết quả best fit line đúng	19
Hình 9.5. Các trường hợp ảnh tìm lane không đúng vị trí trên ảnh binary	19
Hình 9.6. Chỉ tìm thấy 1 lane hoặc không có lane nào	19
Hình 9.7 Giải thuật tìm lane None hoặc not None.....	20
Hình 9.8. Đoạn code trong chương trình con Polyfit()	20
Hình 9.9. Trường hợp tìm được hai lane lý tưởng	21

Hình 9.10. Vị trí và biên độ đỉnh của trường hợp hình 9.9.....	21
Hình 9.11. Trường hợp tìm lane sai vị trí	21
Hình 9.12. Vị trí và biên độ đỉnh của trường hợp hình 9.11	21
Hình 9.13. Thuật toán tìm vị trí lane đúng.....	22
Hình 9.14. Kết quả tìm best fit line không dùng thuật toán ở hình 9.13.....	22
Hình 9.15. Kết quả trên ảnh gốc ở trường hợp hình 9.14.	22
Hình 9.16. Kết quả tìm best fit line dùng thuật toán ở hình 9.13.....	22
Hình 9.17. Kết quả tìm trên ảnh gốc ở trường hợp hình 9.16.....	22
Hình 9.18. Biểu diễn giá trị offset.....	23
Hình 10.1. Phân tích hệ thống.....	25
Hình 10.2. Đáp ứng.....	28
Hình 12.1. Khối cảm biến	39
Hình 12.2. Khối điều khiển trung tâm	39
Hình 12.3. Khối nguồn.....	39
Hình 12.4. Khối servo	40
Hình 12.5. Khối động lực.....	40
Hình 12.6. Arduino UNO R3	42
Hình 12.7. Cầu H	43
Hình 12.8. Cầu H dùng BJT	43
Hình 12.9. Cầu H trong L298D.....	44
Hình 12.10. L298D Driver.....	44
Hình 12.11. Pin Lipo	45
Hình 12.12. Động cơ DC	46
Hình 12.13. PiCamera Module.....	47
Hình 12.14. Servo RC	47
Hình 12.15. SRF-05	48
Hình 12.16. Hình ảnh robot thực tế.....	63
Hình 12.17. Hình ảnh mô hình robot được thiết kế trên Solidworks.....	63
Hình 13.1. Orgirinal Frame.....	65
Hình 13.2. Filter Color	66
Hình 13.3. Smoothing Image	66
Hình 13.4. Color warp image.....	66
Hình 13.5. Thresholding Image	67
Hình 13.6. Histogram.....	67
Hình 13.7. Best fit line	67

CHƯƠNG 1: GIỚI THIỆU VÀ ĐẶT VẤN ĐỀ

1.1 GIỚI THIỆU

Hiện nay, phần cứng máy tính và các thiết bị có tốc độ tín toán, dung lượng và khả năng xử lý vượt bậc. Con người không còn xa lạ với khái niệm hình ảnh, nó đã trở thành tín năng không thể thiếu trong các thiết bị di động, máy tính dùng để lưu giữ những kỹ niệm, những chuyến đi với gia đình bạn bè. Nhưng muốn có một hình ảnh đẹp chất lượng không chỉ phụ thuộc vào thiết bị camera chuyên nghiệp, độ nét cao mà còn đòi hỏi quá trình xử lý bên trong đó làm cho hình ảnh chân thực và ít nhiễu hơn.

Quá trình tác động lên ảnh ban đầu qua nhiều quá trình xử lý ta cho ra một ảnh hoàn thiện thì ta đã dùng kỹ thuật gọi là xử lý ảnh. Nhưng xử lý ảnh cho mục đích đó chỉ là một ứng dụng nhỏ của kỹ thuật xử lý ảnh.

Kỹ thuật xử lý ảnh thuộc lĩnh vực thị giác máy tính (Computer Vision), dùng để phân tích, phát hiện nhận dạng và bám đối tượng qua những ảnh thu được từ camera, trong đó ứng dụng mà hiện nay đang được nghiên cứu và phát triển là công nghệ xe tự lái sử dụng kỹ thuật xử lý ảnh phát hiện làn đường, nhận dạng biển báo cũng như nhận dạng biển số xe, tính toán khoảng cách tới các đối tượng phía trước phương tiện với mục tiêu giảm tai nạn giao thông.

1.2 GIỚI HẠN VÀ MỤC TIÊU ĐỀ TÀI

Thấy được sự thú vị và giá trị của việc xử lý ảnh trong việc phát hiện làn đường, nhóm đã quyết định chọn đề tài xe tự hành này để thực hiện, học hỏi kỹ thuật mới này nhằm áp dụng nó vào thực tế. Tuy vấn đề áp dụng kỹ thuật này vào thực tế còn gặp phải nhiều vấn đề nhưng vấn đề chỉ là thời gian và sự đầu tư trí tuệ của con người. Đề tài của nhóm thực hiện với mục tiêu cho xe tự động chạy giữa làn đường với kỹ thuật tìm best fit line.

CHƯƠNG 2: TỔNG QUAN

Để tài xe tự hành dùng kỹ thuật xử lý ảnh được xây dựng dưới dạng một mô hình robot, điều hướng bằng việc điều chỉnh độ lệch vận tốc của hai bánh chính.

Robot xử dụng một camera kết nối với board nhúng Raspberry, board nhúng có nhiệm vụ xử lý hình ảnh với các giải thuật chính được đóng gói trong các modul, thư viện như numpy (các hàm hỗ trợ thao tác với mảng, ma trận, các toán hạng số học, đại số...), matplotlib(vẽ biểu đồ, hiển thị ảnh với các định dạng khác nhau...), opencv (thư viện có các hàm chuyên dụng cho việc xử lý ảnh....).

Để tài sử dụng giải thuật thuật Slide Windows để Extract Feature nhằm tìm vị trí hai line của làn đường, sau đó dùng giải thuật Polynomial Regression (hàm numpy.polyfit) để tìm hệ số phương trình hồi quy bậc hai, từ đó tìm vị trí giữa của làn đường và tính ra độ lệch của xe so với tâm làn đường để điều khiển xe chạy giữa làn. Sử dụng Arduino giao tiếp I2C để nhận giá trị lệch đó đưa vào bộ điều khiển PD controller để điều khiển vị trí xe theo đúng giá trị mong muốn.

Các lý thuyết và giải thuật được tham khảo chủ yếu trong tài liệu Image Processing Fundamentals, OpenCV-Python Turorial, Raspberry Pi Image Processing Programming và Programing in Python 3.

CHƯƠNG 3: NGUYÊN LÝ XỬ LÝ ẢNH

3.1 GIỚI THIỆU

Kỹ thuật số hiện đại tạo ra cho nó có thể thao tác tín hiệu đa chiều với hệ thống, từ mạch số đơn giản đến máy tính song song cao cấp. Mục đích của các thao tác này có thể chia thành các loại sau, [1]:

- Xử lý ảnh: ngõ vào là một hình ảnh → ngõ ra là hình ảnh
- Phân tích hình ảnh: ngõ vào là một hình ảnh → ngõ ra là các giá trị đo lường.
- Hiểu một hình ảnh: ngõ vào là một hình ảnh → ngõ ra là các mô tả cấp cao của ảnh đó.

Kỹ thuật xử lý ảnh thuộc lĩnh vực thị giác máy tính (computer vision). Computer vision là một lĩnh vực khoa học máy tính, cho phép máy tính nhìn thấy, nhận dạng và xử lý hình ảnh như những gì mà thị giác con người làm được.

Computer vision có mối liên hệ mật thiết với trí tuệ nhân tạo (Artificial Intelligence – AI) làm cho máy tính phải hiểu được những gì nó thấy và thực hiện phân tích hoặc hành động theo mục đích nào đó.

3.2 ĐỊNH NGHĨ MỘT ẢNH SỐ (DIGITAL IMAGE)

Hình ảnh được định nghĩa là một hàm hai chiều $f(x, y)$ hay $a(x, y)$, tại x và y là toạ độ không gian (spatial coordinate) và biên độ f (amplitude) tại mỗi cặp toạ độ (x, y) là mức xám (indensity) của ảnh tại điểm đó.

$$Image\ A = \begin{bmatrix} f(1,1) & f(1,2) & \dots & f(1,N) \\ f(2,1) & f(2,2) & \dots & f(2,N) \\ \dots & \dots & \dots & \dots \\ f(M,1) & f(M,2) & \dots & f(M,N) \end{bmatrix}$$

Khi x, y và giá trị của f đều là hữu hạn, rời rạt theo thời gian, thì hình ảnh đó ta gọi là ảnh số (digital image).

Amplitudes của một ảnh cho trước sẽ luôn có giá trị hoặc là số thực hoặc số nguyên. Một hình ảnh gồm các ảnh con gọi là region-of-interest (ROIs hay regions).

Gray image: thuật ngữ mức xám (gray level) thường dùng để chỉ cường độ xám của một ảnh đơn sắc. Color image: ảnh màu là hệ màu, ảnh màu gồm ba phần tử ảnh riêng biệt (red, green, blue).

CHƯƠNG 4: LÝ THUYẾT VỀ KHÔNG GIAN MÀU

4.1 KHÁI NIỆM

Không gian màu là một mô hình toán học dùng để mô tả các màu sắc trong thực tế được biểu diễn dưới dạng toán học. Có rất nhiều không gian màu khác nhau như RGB, HSV, HLS và CMYK ...

4.2 MỘT SỐ KHÔNG GIAN MÀU

4.2.1 RGB

Không gian màu RGB rất phổ biến trong đồ họa máy tính và các thiết bị kỹ thuật số khác.

Không gian màu RGB là sự kết hợp 3 màu cơ bản RED(R), GREEN(G) và BLUE(B) để mô tả tất cả các màu sắc khác.

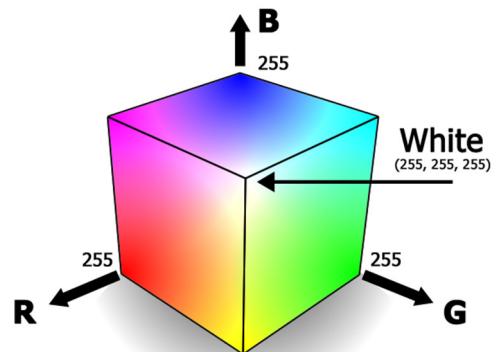
Một ảnh số 24bit, tức là có 8bit cho kênh R, 8bit cho kênh G và 8bit cho kênh B. Mỗi kênh có giá trị từ 0 đến 255.

4.2.2 HSV, HSL

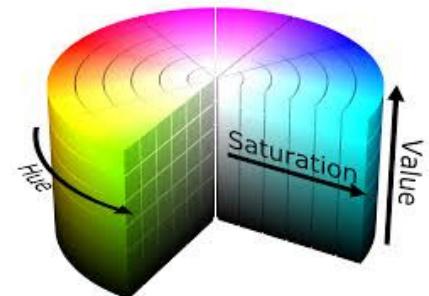
Cũng tương tự HSL, HSV là không gian màu được dùng nhiều để chỉnh sửa, phân tích và một phần của lĩnh vực thị giác máy.

Không gian màu HSV gồm có 3 kênh: Hue(H) – sắc độ, Saturation(S) – độ bão hòa và Value (V) – giá trị cường độ xám. Đối với HSL thì L là Lightness.

Không gian màu này được biểu diễn dưới dạng hình trụ hoặc hình nón.



Hình 4.1. Không gian màu RGB



Hình 4.2. Không gian màu HSV

CHƯƠNG 5: NGƯỠNG CỦA HÌNH ẢNH – THRESHOLDING IMAGE

5.1 GIỚI THIỆU

Gọi θ là một tham số chỉ *brightness threshold* (tạm hiểu là ngưỡng sáng). Tham số θ được chọn và áp dụng vào hình ảnh $a[m, n]$ như sau:

$$a[m, n] = \begin{cases} object = 1 & a[m, n] \geq \theta \\ background = 0 & a[m, n] < \theta \end{cases}$$

hoặc

$$a[m, n] = \begin{cases} object = 1 & a[m, n] < \theta \\ background = 0 & a[m, n] \geq \theta \end{cases}$$

Trong đó “object” hay “background” là hai output, vì tính lưỡng tính của chúng nên có thể biểu diễn như là một biến boolean “0” hoặc “1”.

Theo nguyên tắc, điều kiện kiểm tra có thể dựa trên một vài tính chất khác nhau (ví dụ dựa vào brigtness).

5.2 THRESHOLDING TRONG OPENCV – PYTHON, [2]

Nếu giá trị pixel lớn hơn giá trị threshold, thì tại pixel đó sẽ được gán giá trị 1 (ứng với màu trắng), ngược lại là 0 (ứng với màu đen). Hàm trong OpenCV – Python dùng để ngưỡng hóa ảnh là *cv2.threshold*.

Cú pháp:

```
res, thres = cv2.threshold(src, thresh, maxVal, type)
```

Trong đó:

- **src**: image input, hay array (đa kênh, 8bit kiểu float hoặc 32bit kiểu float).
- **thresh**: giá trị ngưỡng (tham số θ).
- **maxVal**: giá trị lớn nhất để dùng cho loại ngưỡng THRESH_BINARY và THRESH_BINARY_INV.
- **type**: kiểu ngưỡng, bảng bên dưới là các toán hạng của ngưỡng.

Bảng 5.1. Các loại thresholding thông dụng

THRESH_BINARY	$dst(x, y) = \begin{cases} maxVal & src(x, y) > thresh \\ 0 & \text{ngược lại} \end{cases}$
THRESH_BINARY_INV	$dst(x, y) = \begin{cases} maxVal & src(x, y) \leq thresh \\ 0 & \text{ngược lại} \end{cases}$

THRESH_TOZERO	$dst(x, y) = \begin{cases} src(x, y) & src(x, y) > thresh \\ 0 & \text{ngược lại} \end{cases}$
THRESH_TOZERO_INV	$dst(x, y) = \begin{cases} src(x, y) & src(x, y) \leq thresh \\ 0 & \text{ngược lại} \end{cases}$
THRESH_OTSU	Sử dụng giải thuật OTSU để chọn giá trị ngưỡng tối ưu

Code demo:

```
import cv2
import numpy as np

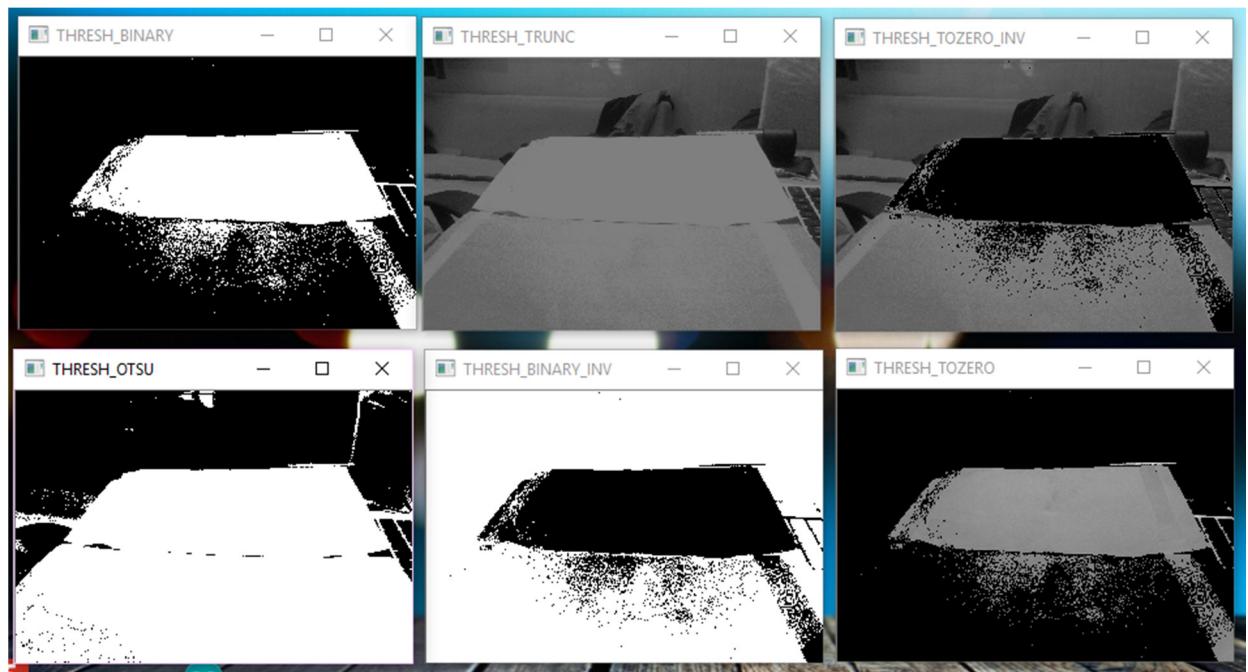
img = cv2.imread('G:/RoboData/Mechatronics Documents/Do An Tot
Nghiep/python_code/clouds.jpg', 0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
ret,thresh6 = cv2.threshold(img,127,255,cv2.THRESH_OTSU)

cv2.imshow('THRESH_BINARY', thresh1)
cv2.imshow('THRESH_BINARY_INV', thresh2)
cv2.imshow('THRESH_TRUNC', thresh3)
cv2.imshow('THRESH_TOZERO', thresh4)
cv2.imshow('THRESH_TOZERO_INV', thresh5)
cv2.imshow('THRESH_OTSU', thresh6)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kết quả:



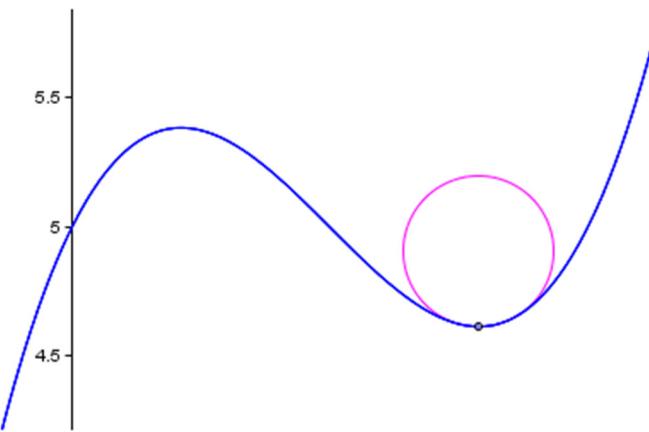
Hình 5.1. Ảnh gốc



Hình 5.2. Kết quả sử dụng các kiểu thresholding

CHƯƠNG 6: TÍNH BÁN KÍNH ĐƯỜNG CONG TẠI MỘT ĐIỂM

6.1 KHÁI NIỆM



Hình 6.1. Đường tròn tiếp xúc với đường cong tại một điểm

Bán kính cong của đường cong tại một điểm được định nghĩa là bán kính xấp xỉ của một đường tròn tiếp xúc với nó tại một điểm đang xét. Bán kính này thay đổi theo quỹ đạo của đường cong đó.

Công thức bán kính cong tại x đối với đường cong $y = f(x)$ là:

$$R = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}}{\left|\frac{d^2y}{dx^2}\right|} \quad (*)$$

6.2 VÍ DỤ

Ta có đường cong bậc 3: $y = 2x^3 - x + 3$, là một parabolic. Tìm bán kính đường cong tại điểm có toạ độ $(1,4)$?

Bước 1: Tìm đạo hàm cấp một của $f(x)$:

$$\frac{dy}{dx} = 6x^2 - 1 \Rightarrow \left(\frac{dy}{dx}\right)^2 = 36x^4 - 12x^2 + 1 \quad (1)$$

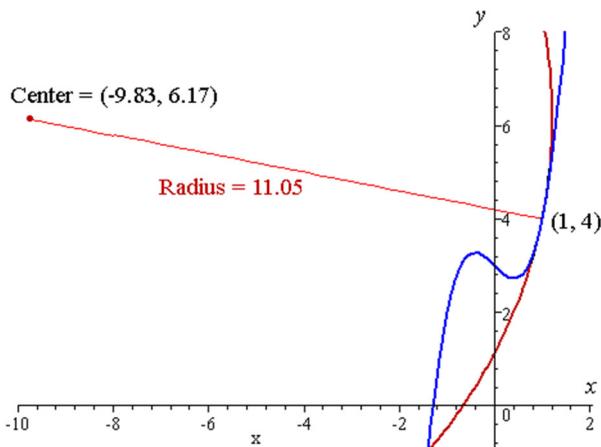
Bước 2: Tìm đạo hàm cấp hai của $f(x)$:

$$\frac{d^2y}{dx^2} = 12x \quad (2)$$

Bước 3: Thay (1) và (2) vào công thức (*), ta được:

$$R = \frac{\left[1 + 36x^4 - 12x^2 + 1\right]^{\frac{3}{2}}}{|12x|} = \frac{(36x^4 - 12x^2 + 2)^{\frac{3}{2}}}{|12x|}$$

Tại $x = 1$, thì $R = 11.047$



Hình 6.2. Tâm của đường tròn xấp xỉ

Vì $\frac{dy}{dx} = 12x$ nên độ dốc của tiếp tuyến tại $x = 1$ là:

$$\frac{dy}{dx} = 12 \times 1 = 12$$

Hai đường thẳng vuông góc có tích hệ số góc là -1 , do đó độ dốc của đường vuông góc với đường tiếp tuyến của đường cong tại $x = 1$ là $-1/12$.

Phương trình đường thẳng có dạng: $y - y_0 = m(x - x_0)$

Với $m = -1/12$ tại $x_0 = 1, y_0 = 4$, ta có:

$$y = -\frac{x}{12} + \frac{49}{12}$$

Bán kính của đường tròn xấp xỉ tại điểm $(1, 4)$ chính là khoảng cách từ tâm đường tròn đó đến điểm có toạ độ $(1, 4)$.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = R = 11.047$$

$$\text{Hay } \sqrt{(1 - x_1)^2 + \left[4 - \left(-\frac{x_1}{12} + \frac{49}{12}\right)\right]^2} = 11.047$$

Suy ra: $x_1 = -9.833, y_1 = 6.167$. Toạ độ tâm đường tròn đó là $(-9.83, 6.17)$.

CHƯƠNG 7: POLYNOMIAL REGRESSION

7.1 LÝ THUYẾT

Polynomial regression là một mô hình Regression (mô hình hồi quy), là mối quan hệ giữa các biến độc lập x và các biến phụ thuộc y được mô hình hóa ở dạng n bậc của x, [4].

Mặc dù Polynomial Regression Fits là một mô hình phi tuyến của dữ liệu, nhưng trong bài toán ước lượng thống kê, nó là tuyến tính. Phương trình $E(y | x)$ là tuyến tính theo tham số chưa biết mà nó được ước lượng từ dữ liệu. Vì vậy, Polynomial Regression được xem như trường hợp đặc biệt của Multiple Linear Regression.

Tổng quát, mô hình Polynomial Regression bậc n:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \varepsilon (*)$$

$$\hat{y}(x, \beta) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n = \sum_{j=0}^n \beta_j x^j$$

Trong đó: ε là một biến ngẫu nhiên không quan sát (phản tử sai lệch), trị trung bình là 0 và phương sai là σ^2 .

Loss Function:

$$E(w) = \frac{1}{2} \sum_{i=0}^p \{y(x_i, \beta) - y_i\}^2$$

Khi ta có m tập dữ liệu (x,y), ta có thể viết lại phương trình (*) dưới dạng ma trận như sau:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Như vậy, y là một vector m chiều, β là một vector $n + 1$ chiều.

Ta viết lại phương trình trên như sau:

$$\vec{y} = \vec{X}\vec{\beta} + \vec{\varepsilon}$$

Nhiệm vụ của chúng ta là tìm vector hệ số β sao cho loss function (error function, cost function) đạt nhỏ nhất.

Như vậy:

$$\hat{\vec{\beta}} = (\vec{X}^T \vec{X})^{-1} \vec{X}^T \vec{y}$$

Ví dụ sử dụng phương pháp khử Gaussian, ta tìm đường hồi quy bậc hai cho một tập dữ liệu có 3 điểm (1,-6), (2,2) và (4,12).

Ta xây dựng mô hình hồi quy bậc hai, ta có phương trình dạng như sau:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

Với 3 cặp tọa độ, ta sẽ có ba phương trình theo dạng trên, ta sẽ viết chúng dưới dạng ma trận.

$$\begin{bmatrix} 1 & 1 & 1^2 \\ 1 & 2 & 2^2 \\ 1 & 4 & 4^2 \end{bmatrix} \begin{bmatrix} \beta_2 \\ \beta_1 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} -6 \\ 2 \\ 12 \end{bmatrix}$$

Theo phương pháp khử Gaussian, ta có ma trận sau:

$$\begin{bmatrix} 1^2 & 1 & 1 & -6 \\ 2^2 & 2 & 1 & 2 \\ 4^2 & 4 & 1 & 12 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & -6 \\ -3 & -2 & 0 & 26 \\ -15 & -12 & 0 & 108 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & -6 \\ -3 & -2 & 0 & 26 \\ 3 & 0 & 0 & -48 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 1 & 1 & -6 \\ 0 & 2 & 0 & 26 \\ 1 & 0 & 0 & 108 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 11 \\ 1 & 0 & 0 & -16 \end{bmatrix}$$

Suy ra: $\begin{bmatrix} \beta_2 \\ \beta_1 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} -1 \\ 11 \\ -16 \end{bmatrix}$, như vậy ta có phương trình hồi quy là $y = -16 + 11x - x^2$

Tổng quát, nếu n là số bậc của phương trình hồi quy thì vector hệ số β sẽ có $n + 1$ phần tử.

7.2 THỰC HIỆN.

Cho tập dữ liệu ngẫu nhiên là tọa độ các điểm bất kỳ trên mặt phẳng xOy tập trung quanh một parabol chưa biết. Parabol đó là đường hồi quy bậc hai của tập dữ liệu. Hệ số của ma trận β là một ma trận Vandermonde.

Ví dụ với yêu cầu tìm đường hồi quy bậc hai cho một tập dữ liệu cho trước.

Code demo

```
import numpy as np
import matplotlib.pyplot as plt

def g(x):
    return x**3 + 5

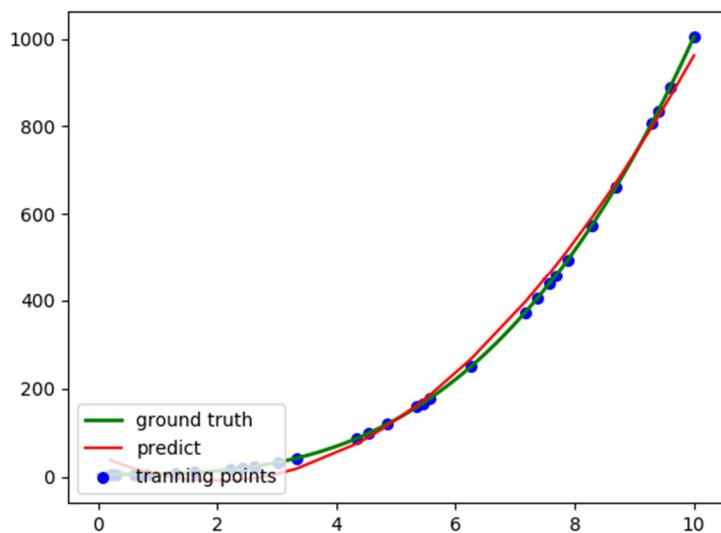
x_plot = np.linspace(0, 10, 100)
x = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
rng.shuffle(x)
# Initial set of data
x = np.sort(x[:30])
```

```

y = g(x)
# Finding best fit line
coeff = np.polyfit(x, y, 2)
f = np.poly1d(coeff)
# Plot
plt.plot(x_plot, g(x_plot), color='green', linewidth=2, label="ground truth")
plt.scatter(x, y, color='blue', s=30, marker='o', label="training points")
plt.plot(x, f(x), color='red', label='predict')
plt.legend(loc='lower left')
plt.show()

```

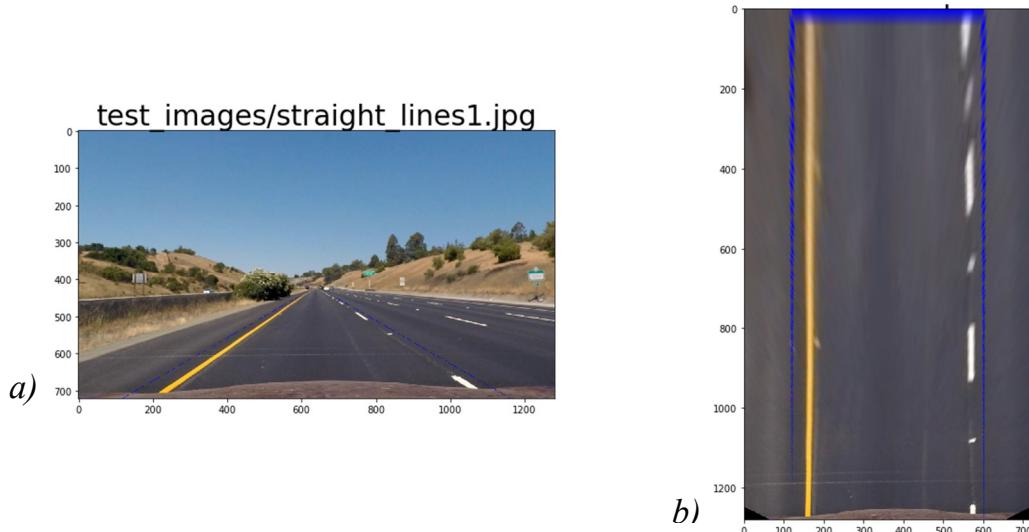
7.3 KẾT QUẢ



Hình 7.1. Kết quả tìm đường hồi quy bậc hai

CHƯƠNG 8: BIRD'S EYE VIEW

8.1 GIỚI THIỆU



Hình 8.1. Ảnh dạng Bird'eye (ảnh b)

Mục tiêu của mô hình biến đổi top-view là cho ta một hình ảnh top-view từ một ảnh chụp dạng phô cản.

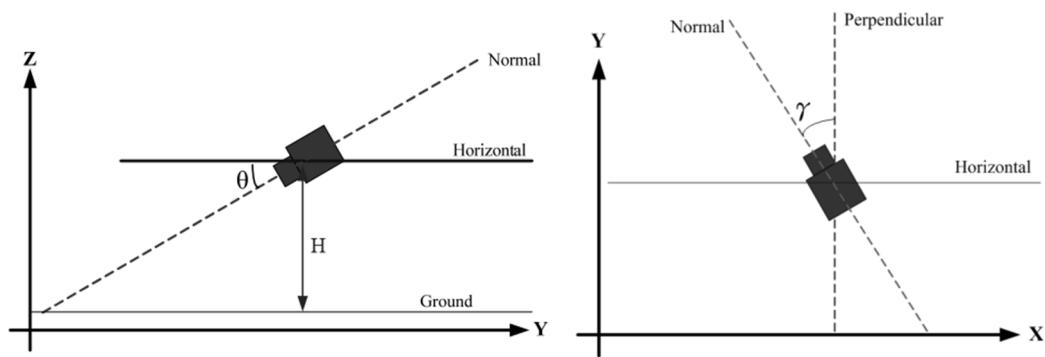
Phương pháp Homographic Mapping được dùng để biểu diễn mối quan hệ giữa hai khung nhìn khác nhau của thế giới thực, [5].

Ma trận thuần nhất H là ma trận không suy biến 3×3 có dạng:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Phương trình dạng thuần nhất có dạng: với $(x_1, y_1, z_1)^T$ là điểm ban đầu, $(x_2, y_2, z_2)^T$ là điểm đích.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$



Hình 8.2. Các góc vị trí của camera

Về mô hình camera, cấu trúc mô hình camera là lý thuyết và khái niệm cơ bản để phát triển ứng dụng thị giác máy tính. Trong ứng dụng này, hệ toạ độ 3D trong thế giới thực được giả thuyết theo hệ toạ độ Right Hand Cartesian. Trục Y được kéo dài về phái trước của xe, trục Z vuông góc với mặt đất.

Hình 8.2 là mô hình camera, có θ là góc nghiêng (tilt angle) của camera, H là độ cao của camera so với mặt đất, γ là góc xoay (pan angle) tạo bởi đường Perpendicular và Normal.

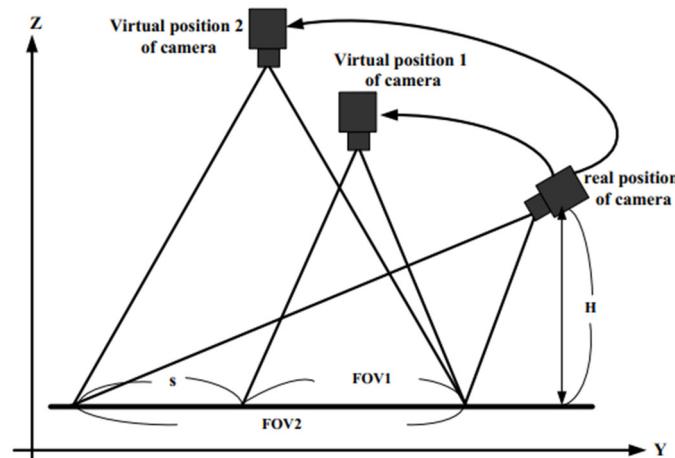
Khi $\gamma = 0$, đường chiếu của đường quang học camera trên mặt phẳng XY sẽ song song với trục Y. Trong trường hợp này, ảnh chụp bởi camera sẽ là hình chiếu phối cảnh của thế giới thực.

Khi $\gamma \neq 0$, ảnh chụp sẽ là ảnh lệch về phía phải hoặc trái của xe.

FOV (Field Of View) của camera được định nghĩa là một hình tháp. Đỉnh nằm tại tâm của mặt phẳng ảnh. Số lượng cảnh trong vùng không gian hình tháp này sẽ được chiếu lên mặt phẳng ảnh của camera phụ thuộc vào FOV và “block plane” phía trước camera.

Trong trường hợp ta đang khảo sát, “block plane” là mặt đất. Hình tháp cắt xén được gọi là một hình cùt chéo cục. Vì camera có thể ở vị trí có độ cao H khác hoặc θ khác. Hình chéo cục sẽ bị lệch phụ thuộc vào điều kiện H và θ .

Hình trên biểu diễn 3 vị trí của camera. “Block plane” được chiếu lên trục Y, ký hiệu là FOV. Vị trí thực của camera được bố trí trên xe ở độ cao H tính từ mặt đất và tầm nhìn là FOV. Vị trí ảo thứ nhất cao hơn độ cao H và tầm nhìn là FOV1. Vị trí ảo thứ hai cao hơn độ cao ở vị trí thứ 2 và tầm nhìn FOV2.



Hình 8.3. Vị trí camera

Như ta thấy tầm nhìn của camera ở vị trí thực và vị trí ảo thứ 2 có $FOV2 = FOV$. Yêu cầu cơ bản về thiết bị là cung cấp một ảnh rõ ràng để thấy được lane phía trước.

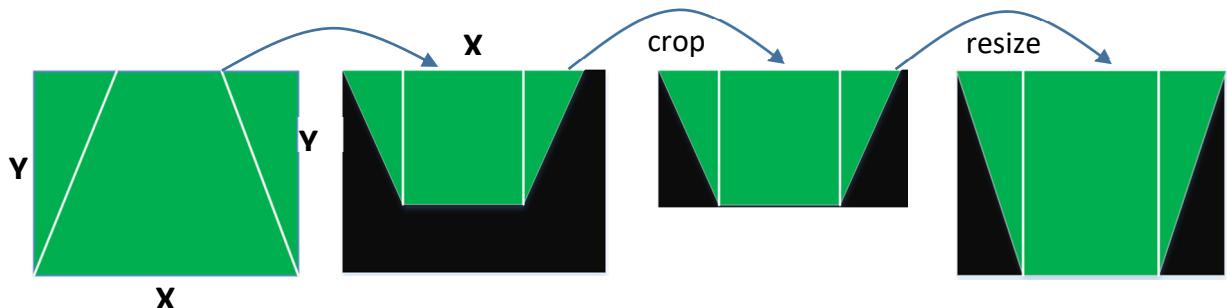
Mô hình biến đổi top-view biến đổi ảnh chụp tại vị trí thực của camera thành ảnh chụp của camera tại vị trí ảo thứ 2. Cả hai ảnh này có kích thước tương tự (vì có FOV giống nhau)

Bây giờ ta thực hiện mô hình biến đổi Top-view (TVTM), gọi (x, y) là toạ độ điểm tại ảnh gốc và (x^*, y^*) là toạ độ tại điểm mong muốn trên ảnh đích. H là khoảng cách từ camera đến mặt đất, f là tiêu cự của camera và θ là góc nghiêng của camera.

Phương trình dưới đây cho thấy giá trị x^* và y^* có thể nhỏ hơn hoặc bằng 0, chắc rằng giá trị x^* và y^* lớn hơn 0 và điều kiện giữ là tọa độ điểm trên ảnh gốc được ánh xạ lên tọa độ ảnh đích:

$$x^* = H \frac{x \sin \theta + f \cos \theta}{-y \cos \theta + f \sin \theta}; y^* = H \frac{y \sin \theta + f \cos \theta}{-y \cos \theta + f \sin \theta}$$

Hình dưới đây cho thấy quá trình mà TVTM biến đổi một ảnh gốc sang ảnh đích.



Hình 8.4. Quá trình tạo ra một ảnh top – view (bird's eye)

8.2 ẢNH BIRD'S EYE TRONG OPENCV - PYTHON

Biến đổi phối cảnh cần ma trận biến đổi 3×3 . Đường thẳng sẽ vẫn thẳng sau khi chuyển đổi. Để tìm ma trận 3×3 ta cần 4 điểm trên ảnh input và ứng với 4 điểm trên ảnh output. Trong số 4 điểm này, 3 trong số đó không nên thẳng hàng. Để tìm ma trận chuyển đổi ta dùng hàm `cv2.getPerspectiveTransform`. Sau đó sử dụng hàm `cv2.warpPerspective`.

Cú pháp:

```
M = cv2.getPerspectiveTransform(pts1, pts2)
dst = cv2.warpPerspective(img, M, dst_size)
```

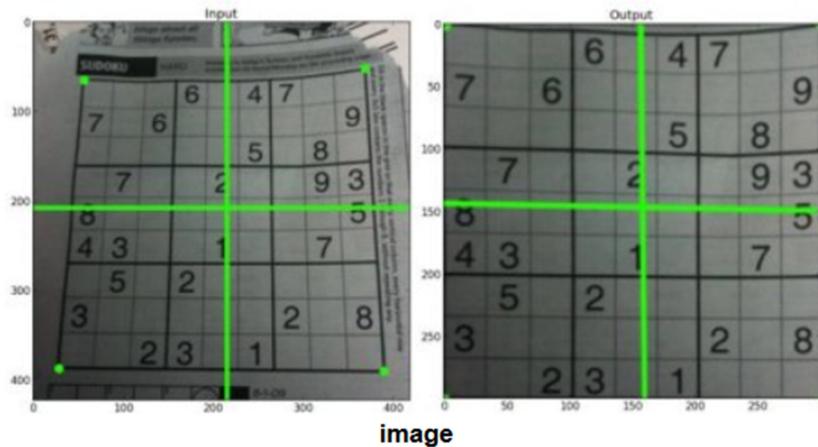
Trong đó:

- $pts1$ là tọa độ 4 điểm trên ảnh input.
- $pts2$ là tọa độ điểm trên ảnh output.
- dst là ảnh output.
- dst_size là kích thước của ảnh output.

Code demo:

```
img = cv2.imread('sudoku.png')
rows, cols, ch = img.shape
pts1 = np.float32([[56, 65], [368, 52], [28, 387], [389, 390]])
pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
M = cv2.getPerspectiveTransform(pts1, pts2)
dst = cv2.warpPerspective(img, M, (300, 300))
plt.subplot(121), plt.imshow(img), plt.title('Input')
plt.subplot(122), plt.imshow(dst), plt.title('Output')
plt.show()
```

Kết quả:



Hình 8.5. Perspective Transform

CHƯƠNG 9: TÌM VỊ TRÍ CỦA XE

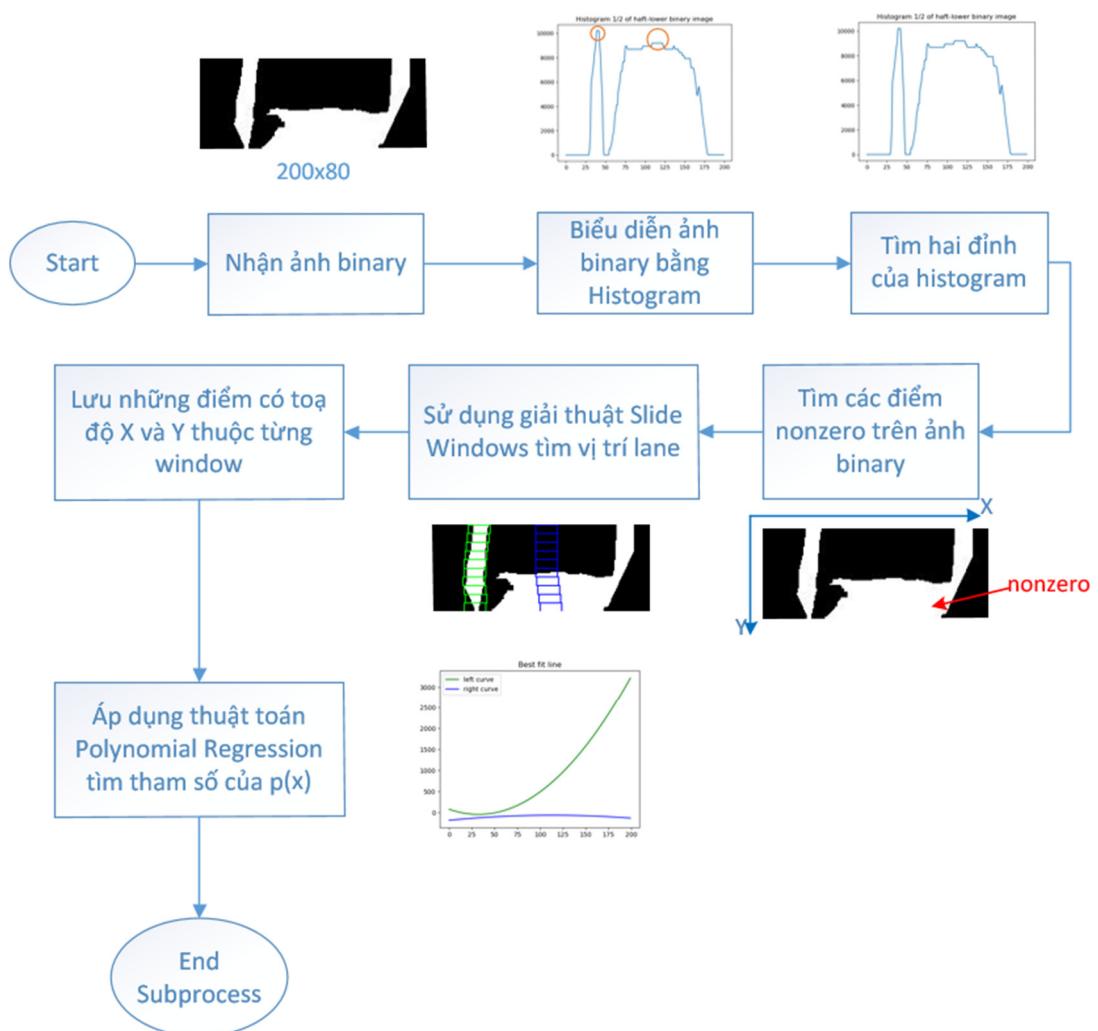
9.1 NỘI DUNG

Trong quá trình tìm best fit line, để tìm được vị trí chính xác của xe đang lệch về phía nào phụ thuộc rất nhiều vào quá trình preprocessing và quá trình tìm best fit line (ở đây sử dụng thuật toán slide windows và thuật toán hồi quy).

Quá trình này được thực hiện bởi hàm Polyfit(), gồm các quá trình cơ bản như sau:



Hình 9.1. Ảnh gốc



Hình 9.2. Quá trình tìm best fit line

Phương trình hồi quy trong trường hợp tìm best fit line là một đường cong bậc hai có dạng:

$$y = f(x) = ax^2 + bx + c$$

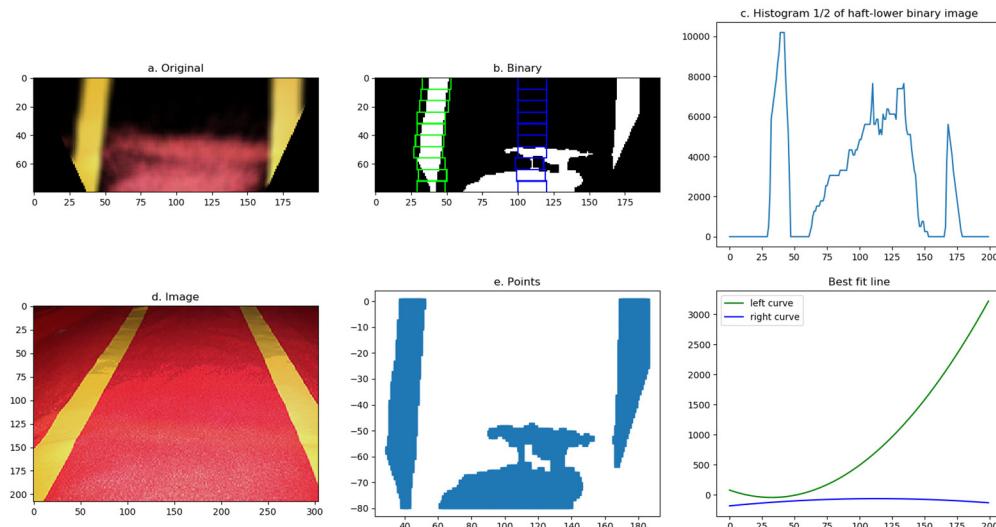
Nhiệm vụ của mô hình Polynominal regression (ở đây ta dùng hàm Polyfit()) là tìm tham số a, b và c của phương trình.

Hàm Polyfit() của numpy hỗ trợ việc tìm đường hồi quy đó, kết quả của hàm Polyfit() là một vector hàng $p = [p_0 \ p_1 \ p_2]$

Trong đó: $p_0 = a$, $p_1 = b$ và $p_2 = c$.

Việc tìm đường hồi quy tốt phụ thuộc rất nhiều vào quá trình Preprocessing, trong đó có quá trình xử lý ảnh binary.

Kết quả tìm best fit line như sau:



Hình 9.3. Kết quả tìm best fit line

9.2 VÂN ĐÈ KHI TÌM ĐƯỜNG HỒI QUY

Ta có hai vấn đề xảy ra:

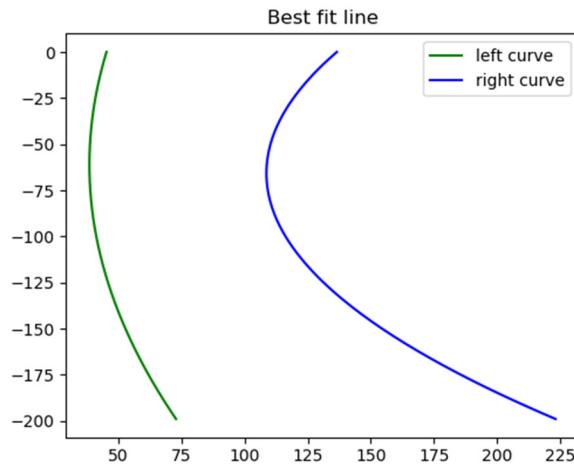
- (a) Đồ thị biểu diễn $f(x)$ không theo hướng của lane.
- (b) Các window bên phải đã không trượt trên phần lane thực tế mà trượt lên phần pixel không phải lane.

Ta đi giải quyết vấn đề (a), trong hệ toạ độ xOy, các hàm số là phương trình biểu diễn sự phục thuộc của biến y theo x.

Trong hàm $Polyfit(X, Y)$, X là biến độc lập và Y là biến phụ thuộc x. Thuật toán hồi quy dự đoán $\hat{y} = f(X)$, do đó kết quả biểu diễn trên hình “Best fit line” có vị trí trải dài theo phương ngang. Đó dạng đồ thị y theo x. Như vậy, đường hồi quy sai vì không fit theo lane.

Vậy bài toán hồi quy lúc này cần đổi x là biến phục thuộc y. Ta sẽ tìm đường hồi quy $\hat{x} = f(Y)$.

Kết quả như sau:



Hình 9.4. Kết quả best fit line đúng

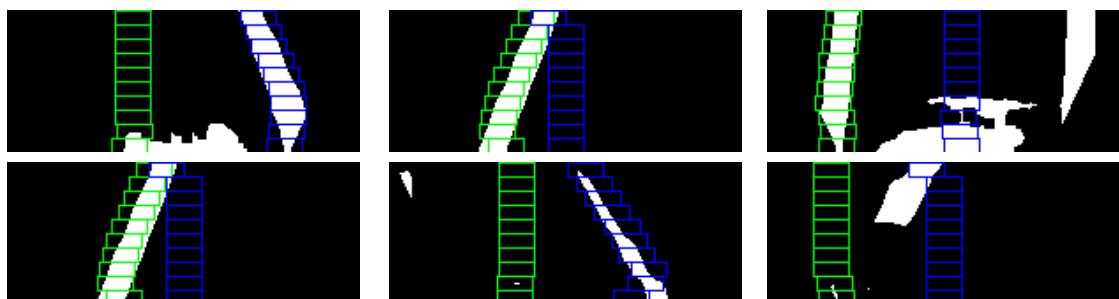
Hai đường cong cùng hướng với lane, hai đường cong nằm về đúng phía của chúng.

Như vậy ta đã xử lý xong vấn đề (a).

Đối với vấn đề (b), vì đỉnh histogram bên phải không phải của lane mà là của phần pixel ở giữa (không phải của lane), vì đỉnh của phần này lớn hơn đỉnh của lane phải nên được chọn làm vị trí của lane.

Xét lại ta có các trường hợp sau sẽ bị sai khi tìm best fit line hoặc sẽ bị lỗi (*TypeError: expected non-empty vector for x*) khi dùng hàm Polyfit() (do chỉ tìm thấy 1 trong hai lane hoặc không có lane nào).

Trường hợp 1: Tìm thấy 2 lane nhưng vị trí một trong hai lane không được tìm đúng hoặc tìm nhầm đối tượng không phải lane.



Hình 9.5. Các trường hợp ảnh tìm lane không đúng vị trí trên ảnh binary

Trường hợp 2: Chỉ tìm thấy 1 lane hoặc không có lane nào



Hình 9.6. Chỉ tìm thấy 1 lane hoặc không có lane nào

Ở đây ta tìm lane ở nửa dưới của ảnh binary, ta thấy phần nửa dưới ảnh bị mất các pixel của lane do đó, trường hợp 2 có thể làm chương trình báo lỗi trong quá trình tìm đường hồi quy bằng Polyfit() function và chương trình bị dừng lại.

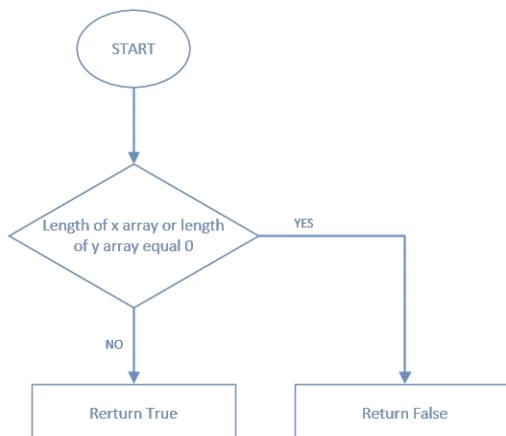
Nhận xét:

Ta cần ưu tiên giải quyết trường hợp 2 vì trường hợp này làm cho chương trình bị dừng lại.

Khi các pixel nonzero tồn tại thì sẽ tồn tại giá trị chỉ số x và y, nếu không có pixel nào nonzero(không có lane) thì sẽ không tồn tại x và y. Do đó, không thể thực hiện hàm Polyfit(), lỗi này sẽ làm chương trình dừng lại dù có tìm được lane trái hoặc lane phải.

Vì vậy, ta cần một hàm CheckLaneNone(), hàm này sẽ trả về True nếu có các pixel nonzero và trả về False nếu không tìm thấy các nonzero. Đồng thời khởi tạo hai biến lưu trữ chỉ số nonzero của các pixel được khởi tạo với giá trị ban đầu là [0, 0, 0].

Thuật toán của hàm CheckLaneNone() như hình bên dưới:



Hình 9.7 Giải thuật tìm lane None hoặc not None

Đoạn code như hình dưới đây, trong đó dòng 260 và 262 là điều kiện để thực hiện hàm Polyfit() khi hàm CheckLaneNone() trả về là True.

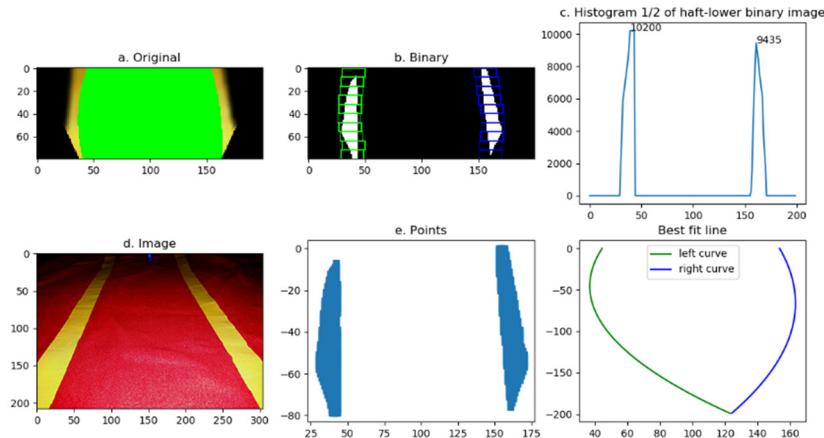
```

244     # Points Coordinate on Windows of binary image, same as dataset to predict parameters of curve #
245     x_left = X[left_indices]
246     y_left = Y[left_indices]
247     x_right = X[right_indices]
248     y_right = Y[right_indices]
249
250     #print("x_left = ", x_left, ", y_left = ", y_left)
251     #print("x_right = ", x_right, ", y_right = ", y_right)
252
253     check_left = CheckLaneNone(x_left, y_left)
254     check_right = CheckLaneNone(x_right, y_right)
255
256     y = np.linspace(0, hist.shape[0]-1, hist.shape[0])
257     # Apply regression algorithm for dataset #
258     p_left = np.array([0,0,0])
259     p_right = np.array([0,0,0])
260     if check_left == True:
261         p_left = np.polyfit(y_left, x_left, 2)
262     if check_right == True:
263         p_right = np.polyfit(y_right, x_right, 2)
264
265     # Square Function of lane is graph x = f(y) on xOy coordinate system #
266
267     fy_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
268     fy_right = p_right[0]*y**2 + p_right[1]*y + p_right[2]
  
```

Hình 9.8. Đoạn code trong chương trình con Polyfit()

Kết luận: Như vậy ta đã giải quyết được vấn đề phát sinh lỗi khi thực hiện hàm Polyfit() do không tìm thấy lane (không có các pixel nonzero bên trái hoặc phải hoặc cả hai đều không có).

Tiếp theo, ta sẽ giải quyết vấn đề ở trường hợp 1 (tìm thấy lane) nhưng không đúng vị trí. Xét hai lane lý tưởng như hình bên dưới:



Hình 9.9. Trường hợp tìm được hai lane lý tưởng

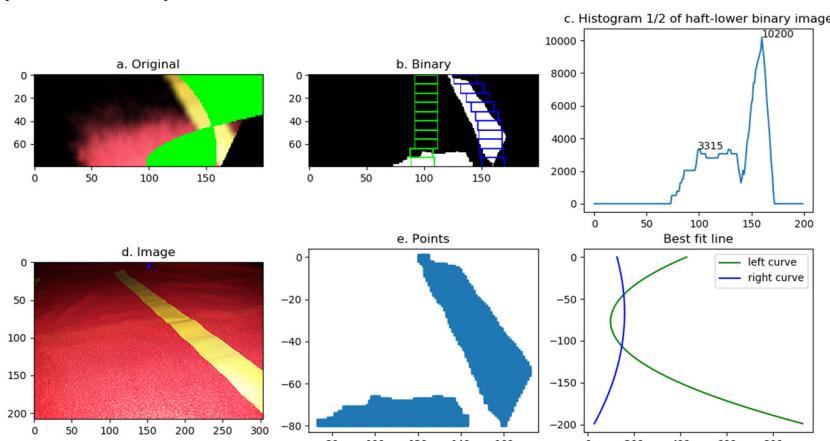
Kết quả tính số đỉnh, vị trí hai đỉnh như hình bên:

Khoảng cách hai lane trên ảnh bird'eye tính theo đơn vị pixel là: $161 - 39 = 122$ pixel.

Trường hợp ảnh trên là trường hợp lý tưởng, ta tìm được hai lane đúng. Tuy nhiên, trên thực tế còn nhiều làm cho lane được tìm sai vị trí. Như hình bên dưới:

Number of peak: 2
[29, 151]
[44, 171]
Left peak position: 39
Left peak value: 10200
Right peak position: 161
Right peak value: 9435

Hình 9.10. Vị trí và biên độ đỉnh của trường hợp hình 9.9



Hình 9.11. Trường hợp tìm lane sai vị trí

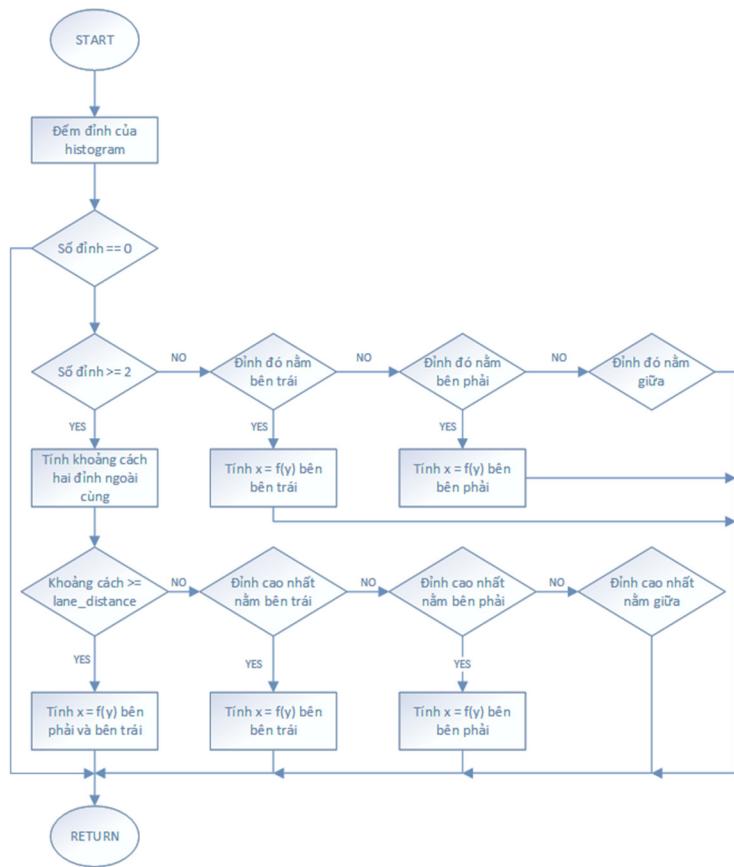
Kết quả tính số đỉnh và vị trí hai đỉnh như hình 9.12, như vậy khoảng cách hai đỉnh lúc này là

$$160 - 99 = 61 < 122.$$

Number of peak: 1
[73]
[172]
Left peak position: 99
Left peak value: 3315
Right peak position: 160
Right peak value: 10200

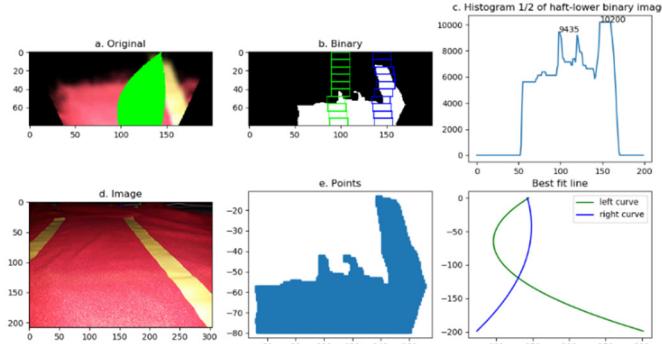
Hình 9.12. Vị trí và biên độ đỉnh của trường hợp hình 9.11

Thuật toán giải quyết vấn đề trên như sau:



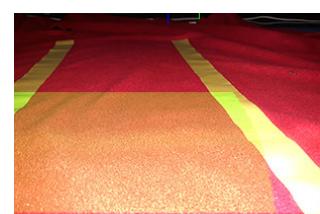
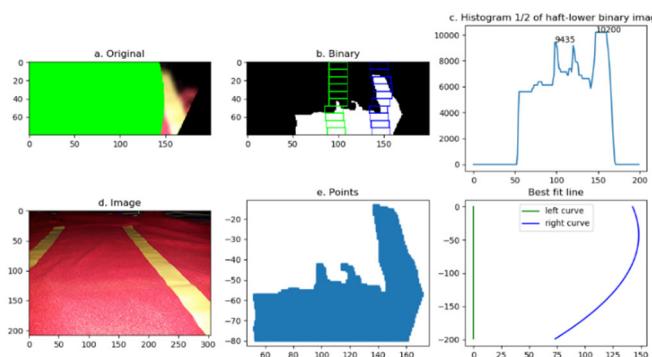
Hình 9.13. Thuật toán tìm vị trí lane đúng

Kết quả trước và sau khi dùng thuật toán trên:



Hình 9.15. Kết quả trên ảnh gốc ở trường hợp hình 9.14.

Hình 9.14. Kết quả tìm best fit line không dùng thuật toán ở hình 9.13.



Hình 9.17. Kết quả tìm trên ảnh gốc ở trường hợp hình 9.16.

Hình 9.16. Kết quả tìm best fit line dùng thuật toán ở hình 9.13

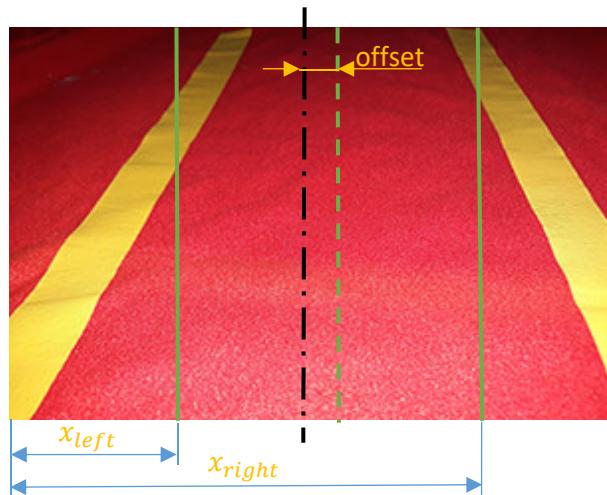
9.3 TÍNH GIÁ TRỊ OFFSET:

Giá trị offset lấy đường đối xứng của ảnh làm chuẩn, giá trị offset được tính bằng hiệu giữa đường đối xứng và vị trí tâm hai lane. Đường đối xứng màu đen là cố định và là vị trí setpoint của robot. Đường nét đứt là tâm hai lane hiện tại.

Độ lệch vị trí hai đường này chính là vị trí xe so với tâm hai lane.

$$\text{offset} = \text{centerline}_{\text{position}} - \frac{x_{\text{left}} + x_{\text{right}}}{2}$$

Trong đó, $x = f(y)$, tại $y = \text{warp_size.shape}[1] - 1 = 79$.



Hình 9.18. Biểu diễn giá trị offset

CHƯƠNG 10: MÔ HÌNH TOÁN HỌC CỦA HỆ THỐNG

Bảng 10.1. Các tham số của hệ thống

J	Momen quán tính của rotor	$kg \cdot m^2$
b	Hằng số ma sát nhót của motor	$N \cdot m \cdot s$
K_e	Hằng số lực điện động	$V/rad/s$
K_t	Hằng số momen xoắn	$N \cdot m/A$
K	Trong hệ đơn vị SI, $K = K_t = K_e$	
R	Điện trở cuộn dây	Ω
L	Điện cảm của cuộn dây	H
θ	Góc lệch của xe so với trục Ox	$^\circ$
φ	Góc quay của rotor	rad
U	Điện áp đặt vào phần ứng	V
i	Dòng điện trong phần ứng	A
l	Khoảng cách giữa hai bánh xe	m
$V_{R,L}$	Vận tốc tịnh tiến của bánh xe trái (V_L) hoặc phải (V_R)	m/s

10.1 PHƯƠNG TRÌNH VI PHÂN

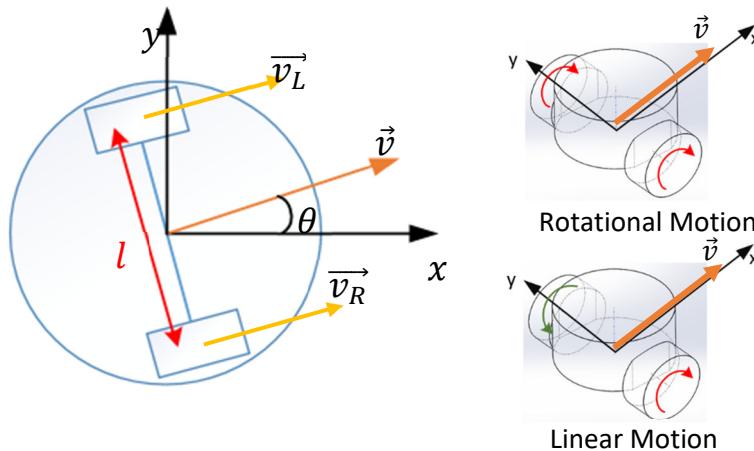
Mục đích xây dựng và mô phỏng mô hình toán học của hệ thống nhằm biết tính chất của hệ thống. Các tham số trong bảng 10.1 có thể được bỏ qua vì trên thực tế không tính được chính xác các tham số vì yêu cầu cần các thiết bị chuyên môn để thí nghiệm và xác định. Với mục đích đề tài ta có thể bỏ qua bước xác định các tham số này.

10.1.1 Mối quan hệ giữa θ và v của hai bánh.

Robot có hai chuyển động thẳng (linear motion) và chuyển động quay (rotational).

- Linear motion: hai bánh chuyển động ngược chiều với vận tốc và có cùng tốc độ.
- Rotation: hai bánh chuyển động ngược chiều với nhau và có cùng tốc độ, hoặc hai bánh chuyển động cùng chiều với vận tốc và khác tốc độ.

Khi xe chuyển động thẳng hay quay thì phụ thuộc vào chiều quay, tốc độ của mỗi bánh, bánh xe quay khi động cơ được đặt vào một điện áp U (Volt, V).



Hình 10.1. Phân tích hệ thống

Vận tốc góc của robot tỷ lệ thuận với vận tốc trung bình hai bánh và tỷ lệ nghịch với khoảng cách hai bánh xe.

Ta có phương trình vi phân sau:

$$\dot{\theta} = \frac{V_L - V_R}{2l} \quad (1)$$

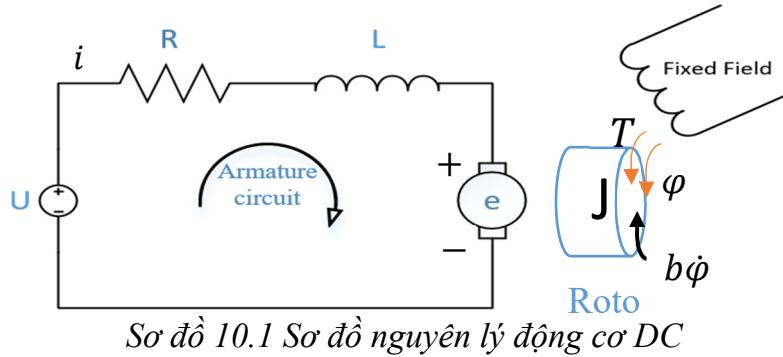
Tên “Angle” đặt cho block trong simulink đại diện cho góc θ , được hồi tiếp về hệ thống có thể mang dấu + hoặc -.

Góc θ phụ thuộc vào độ lệch vận tốc hai bánh xe như sau:

$\theta > 0$	$V_L - V_R < 0$
$\theta < 0$	$V_L - V_R > 0$
$\theta = 0$	$V_L - V_R == 0$

Với giá trị θ mong muốn là 0, thì ta cần dùng bộ điều khiển để hệ thống có ngõ ra như mong muốn.

10.1.2 Phương trình vi phân của động cơ DC.



Điện áp rơi trên cuộn dây:

$$U_L = L \frac{di}{dt}$$

Sức phản điện động trong mạch phần ứng: $e = K_e \dot{\varphi}$

Áp dụng định luật Kirchoff cho mạch phản ứng, ta có:

$$\frac{di}{dt} = \frac{1}{L} \left(-Ri + U - K \frac{d\varphi}{dt} \right) \quad (2)$$

Momen sinh ra từ rotor: $T = K_t i = Ki$

Áp dụng định luật Newton II trên rotor, ta có:

$$\frac{d^2\varphi}{dt^2} = \frac{1}{J} \left(Ki - b \frac{d\varphi}{dt} \right) \quad (3)$$

Vì R và L trong mạch rất nhỏ và không đáng kể nên cũng có thể bỏ qua R và L . Đồng thời J ở đây được quy đổi về trực động cơ, do đó nó còn phụ thuộc vào tải.

Như vậy: Để đơn giản ta có thể xem quá trình điều khiển động cơ gồm hai đại lượng điện áp U đặt vào động cơ và vận tốc động cơ V tỷ lệ với nhau qua hệ số nào đó (xét trường hợp tải không đổi).

10.1.3 Bộ điều khiển.

Tính chất của hệ thống: Khi sai lệch e càng + hoặc càng - thì ngõ ra θ sẽ không thể bám được, tức dao động càng lớn làm hệ thống mất ổn định. Những hệ thống như vậy thì trong hàm truyền của hệ thống thường có khâu tích phân rồi. Do đó ta chỉ cần khâu P và D để điều chỉnh ngõ ra. Vì vậy ta dùng bộ điều khiển PD.

Trong miền thời gian, ngõ ra $u(t)$ của bộ điều khiển có dạng:

$$u(t) = K_P e(t) + K_D \frac{de}{dt}$$

Để ngõ ra hệ thống là $\theta = y$ bám theo giá trị $r = 0$, ta cần có bộ điều khiển để điều khiển hai động cơ. Ngõ ra bộ điều khiển là giá trị u , cũng là giá trị ΔV được cộng vào một trong hai motor để sai lệch $e = r - y$ tiến về 0.

Việc cộng ΔV vào motor trái hay motor phải phụ thuộc vào dấu của θ hay dấu của e . Ngõ ra PD đóng vai trò là giá trị sai lệch chỉ được cộng vào một trong hai motor. Vì

ban đầu motor đã được đặt vào một điện áp ban đầu là V , do đó ta chỉ cần giá trị của u và không quan tâm đến dấu của u .

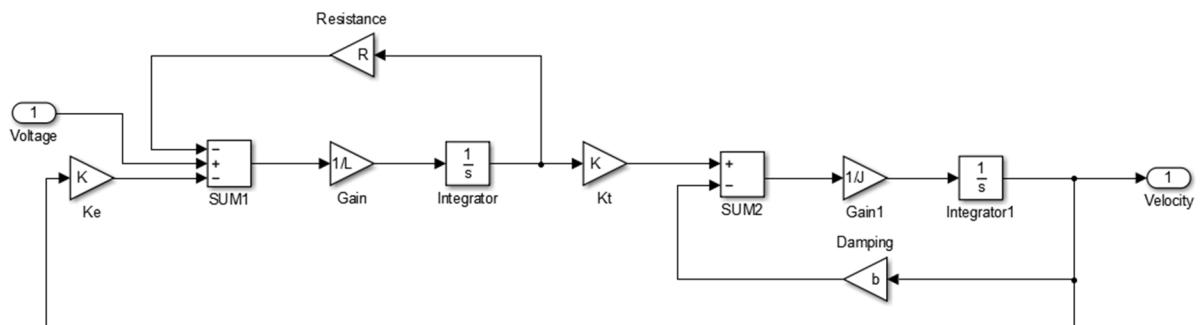
Dạng rời rạc của bộ điều khiển:

$$u(k) = u(k-1) + K_P e(k) + K_D \frac{e(k) - e(k-1)}{\Delta t}$$

Trên thực tế, tham số P và D được chọn theo thực nghiệm. Trước hết cho $D = 0$, P sẽ tăng dần đến khi robot bám được và còn dao động. Lúc này ta tăng dần D, khâu D sẽ làm giảm vọt lồ giữ cho xe bám tốt hơn mỗi khi xe có sự thay đổi độ lệch.

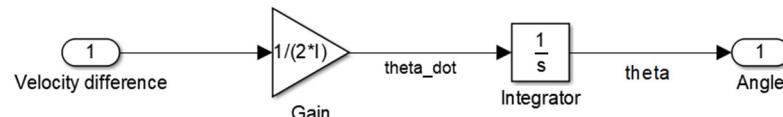
10.2 MÔ PHỎNG TRÊN SIMULINK

Sơ đồ khối điều khiển động cơ DC mô phỏng cho phương trình (2) và (3).



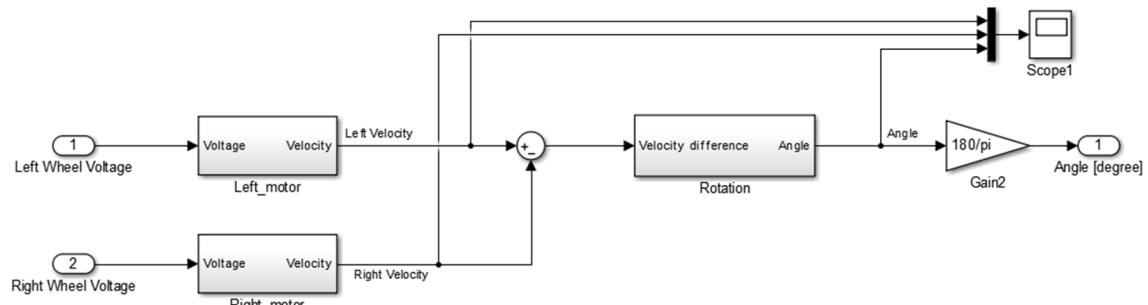
Sơ đồ 10.2. Sơ đồ tả phương trình vi phân của động cơ DC

Sơ đồ khối mô phỏng cho phương trình (1) với output là θ .



Sơ đồ 10.3. Quan hệ θ và v

Sơ đồ mô phỏng cho hệ thống với input là điện áp U đặt vào motor và output là góc lệch θ :



Sơ đồ 10.4. Sơ đồ tả phương trình vi phân (1)

Sơ đồ khối “Selection block” có nhiệm vụ lựa chọn việc tăng thêm giá trị ΔV cho động cơ bên trái/phải dựa vào dấu của e .

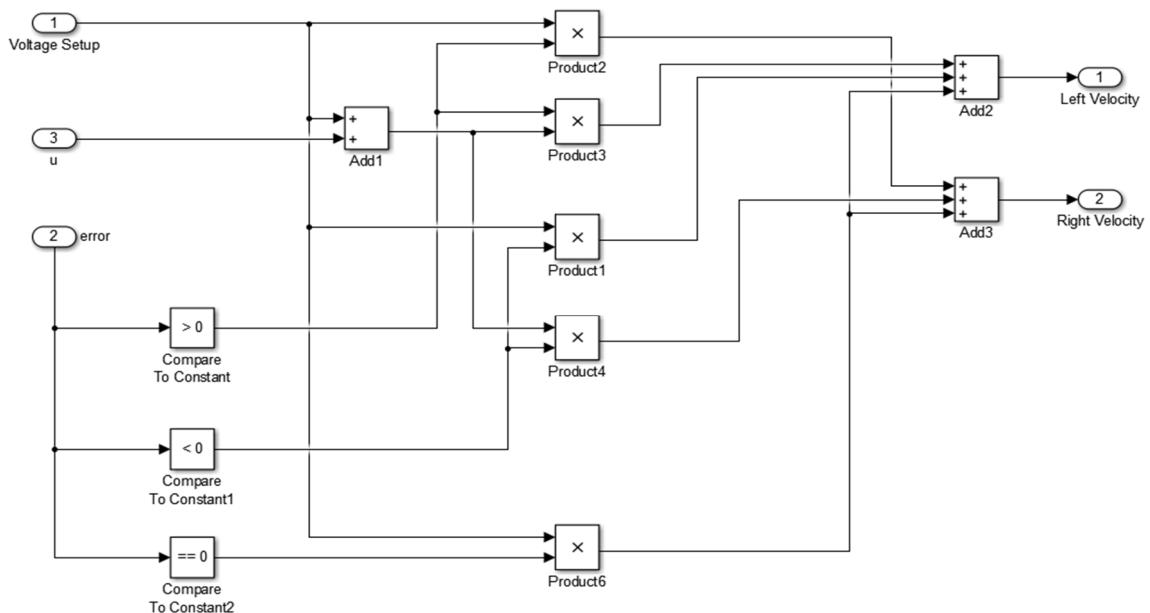
Thuật toán:

If $e < 0$ then $V_L = V_{setpoint} + \Delta V, V_R = V_{setpoint}$

If $e > 0$ then $V_L = V_{setpoint}, V_R = V_{setpoint} + \Delta V$

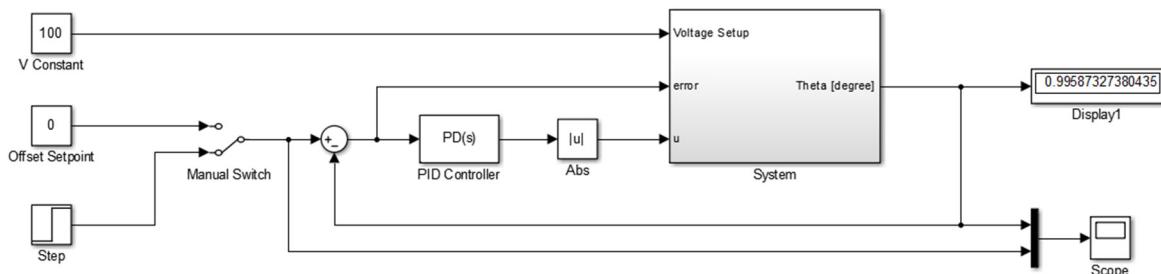
If $e == 0$ then $V_L = V_{setpoint}, V_R = V_{setpoint}$

Thiết kế trên simulink:



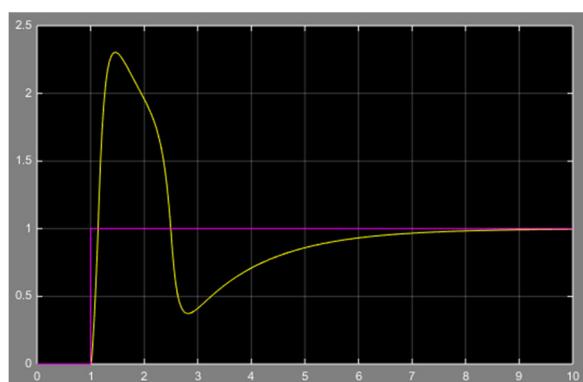
Sơ đồ 10.5. Sơ đồ thực hiện nhiệm vụ tăng tốc cho một trong hai bánh xe theo điều kiện.

Như vậy sơ đồ toàn hệ thống như sau:



Sơ đồ 10.6. Sơ đồ của hệ thống

Kết quả đáp ứng của hệ thống sử dụng PD controller:



Hình 10.2. Đáp ứng

10.3 KẾT LUẬN

Qua phương trình toán học ta thấy hệ thống là hàm truyền bậc 2. Ngõ vào là điện áp, ngõ ra hệ thống là vị trí của xe. Khi cấp điện áp cho động cơ, vị trí xe sẽ dần tiếng về vô cùng tức là xe lệch khỏi lane.

Bộ điều khiển PD controller được điều chỉnh theo thực nghiệm.

CHƯƠNG 11: RASPBERRY PI 3 AND OPENCV – PYTHON

11.1 NGÔN NGỮ PYTHON, [3]

11.1.1 Giới thiệu

Python là một ngôn ngữ lập trình thông dịch do Guido van Rossum tạo ra năm 1990. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động, do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

Theo đánh giá của Eric S. Raymond, Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu.

Ban đầu, Python được phát triển để chạy trên nền Unix. Nhưng rồi theo thời gian, nó đã chạy được mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix.

11.1.2 Đặc điểm

Dễ học, dễ đọc.

Python được thiết kế để trở thành một ngôn ngữ dễ học, mã nguồn dễ đọc, bố cục trực quan, dễ hiểu.

Từ khóa

Python tăng cường sử dụng từ khóa tiếng Anh, hạn chế các ký hiệu và cấu trúc cú pháp so với các ngôn ngữ khác.

Python là một ngôn ngữ phân biệt kiểu chữ HOA, chữ thường.

Như C/C++, các từ khóa của Python đều ở dạng chữ thường.

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def		finally	in	print

11.1.3 Các bản hiện thực

Python được viết từ những ngôn ngữ khác, tạo ra những bản hiện thực khác nhau. Bản hiện thực Python chính, còn gọi là CPython, được viết bằng C, và được phân phối kèm một thư viện chuẩn lớn được viết hỗn hợp bằng C và Python. CPython có thể chạy trên nhiều nền và khả chuyển trên nhiều nền khác. Dưới đây là các nền trên đó, CPython có thể chạy.

Các hệ điều hành họ Unix: AIX, Darwin, FreeBSD, Mac OS X, NetBSD, Linux, OpenBSD, Solaris,...

Các hệ điều hành dành cho máy desktop: Amiga, AROS, BeOS, Mac OS 9, Microsoft Windows, OS/2, RISC OS.

Các hệ thống nhúng và các hệ đặc biệt: GP2X, Máy ảo Java, Nokia 770 Internet Tablet, Palm OS, PlayStation 2, PlayStation Portable, Psion, QNX, Sharp Zaurus, Symbian OS, Windows CE/Pocket PC, Xbox/XBMC, VxWorks.

Các hệ máy tính lớn và các hệ khác: AS/400, OS/390, Plan 9 from Bell Labs, VMS, z/OS.

Ngoài CPython, còn có hai hiện thực Python khác: Jython cho môi trường Java và IronPython cho môi trường .NET và Mono.

11.1.4 Trình thông dịch

Python là một ngôn ngữ lập trình dạng thông dịch, do đó có ưu điểm tiết kiệm thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác.

Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix, tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ, hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm.

Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

11.1.5 Lệnh và cấu trúc điều khiển

Mỗi câu lệnh trong Python nằm trên một dòng mã nguồn. Ta không cần phải kết thúc câu lệnh bằng bất kì ký tự gì. Cũng như các ngôn ngữ khác, Python cũng có các cấu trúc điều khiển. Chúng bao gồm:

Cấu trúc rẽ nhánh: cấu trúc `if` (có thể sử dụng thêm `elif` hoặc `else`), dùng để thực thi có điều kiện một khối mã cụ thể.

Cấu trúc lặp, bao gồm:

- Lệnh `while`: chạy một khối mã cụ thể cho đến khi điều kiện lặp có giá trị `false`.
- Vòng lặp `for`: lặp qua từng phần tử của một dãy, mỗi phần tử sẽ được đưa vào biến cục bộ để sử dụng với khối mã trong vòng lặp.

Python cũng có từ khóa `class` dùng để khai báo lớp (sử dụng trong lập trình hướng đối tượng) và lệnh `def` dùng để định nghĩa hàm.

11.1.6 Hệ thống kiểu dữ liệu

Python sử dụng hệ thống kiểu *duck typing*, còn gọi là *latent typing* (tự động xác định kiểu). Có nghĩa là, Python không kiểm tra các ràng buộc về kiểu dữ liệu tại thời điểm dịch, mà là tại thời điểm thực thi. Khi thực thi, nếu một thao tác trên một đối tượng bị thất bại, thì có nghĩa là đối tượng đó không sử dụng một kiểu thích hợp.

Sử dụng Python, ta không cần phải khai báo biến. Biến được xem là đã khai báo nếu nó được gán một giá trị lần đầu tiên. Căn cứ vào mỗi lần gán, Python sẽ tự động xác định kiểu dữ liệu của biến. Python có một số kiểu dữ liệu thông dụng sau:

- int, long: số nguyên. Độ dài của kiểu số nguyên là tùy ý, chỉ bị giới hạn bởi bộ nhớ máy tính.
- float: số thực
- complex: số phức, chẳng hạn $5+4j$
- list: dãy trong đó các phần tử của nó có thể được thay đổi, chẳng hạn [8, 2, 'b', -1.5]. Kiểu dãy khác với kiểu mảng (array) thường gặp trong các ngôn ngữ lập trình ở chỗ các phần tử của dãy không nhất thiết có kiểu giống nhau. Ngoài ra phần tử của dãy còn có thể là một dãy khác.
- tuple: dãy trong đó các phần tử của nó không thể thay đổi.
- str: chuỗi ký tự. Từng ký tự trong chuỗi không thể thay đổi. Chuỗi ký tự được đặt trong dấu nháy đơn, hoặc nháy kép.
- dict: từ điển, còn gọi là "hashtable": là một cặp các dữ liệu được gắn theo kiểu {từ khóa: giá trị}, trong đó các từ khóa trong một từ điển nhất thiết phải khác nhau. Chẳng hạn {1: "Python", 2: "Pascal"}
- set: một tập không xếp theo thứ tự, ở đó, mỗi phần tử chỉ xuất hiện một lần.

Ngoài ra, Python còn có nhiều kiểu dữ liệu khác như sau:

- Kiểu số: 1234565396326 (số nguyên dài vô hạn) -86.12 7.84E-04 2j 3 + 8j
- Kiểu chuỗi (string) 'abcd' '1234' '3d5-d'
- Kiểu bộ (tuple): (1, 2.0, 3) (1,) ("Hello",1,())
- Kiểu danh sách (list): [4.8, -6] ['a','b']
- Kiểu từ điển (dictionary):

```
dict = {"Hanoi":"Vietnam", "Haiphong":"Vietnam", "Hochiminh":"Vietnam",
"Netherlands":"Amsterdam", "France":"Paris"}
```

11.1.7 Module

Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Nó cũng cung cấp sẵn một tập hợp các modul chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng.

11.1.8 Cú pháp

Khối lệnh:

Trong các ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp ký hiệu hoặc từ khóa. Ví dụ, trong C/C++, cặp ngoặc nhọn {} được dùng để bao bọc một khối lệnh. Python, trái lại, có một cách rất đặc biệt để tạo khối lệnh, đó là thụt các câu lệnh trong khối vào sâu hơn (về bên phải) so với các câu lệnh của khối lệnh cha chứa nó. Để biểu thị một khối rỗng Python có từ khoá "pass". Ví dụ, giả sử có đoạn mã sau trong C/C++:

```
#include <math.h>
//...
delta = b * b - 4 * a * c;
if (delta > 0)
{
    // Khối lệnh mới bắt đầu từ ký tự { đến }
    x1 = (- b + sqrt(delta)) / (2 * a);
    x2 = (- b - sqrt(delta)) / (2 * a);
    printf("Phuong trinh co hai nghiem phan biet:\n");
    printf("x1 = %f; x2 = %f", x1, x2);
}
```

Đoạn mã trên có thể được viết lại bằng Python như sau:

```
import math
#...
delta = b * b - 4 * a * c
if delta > 0:
    # Khối lệnh mới, thụt vào đầu dòng
    x1 = (- b + math.sqrt(delta)) / (2 * a)
    x2 = (- b - math.sqrt(delta)) / (2 * a)
    print "Phuong trinh co hai nghiem phan biet:"
    print "x1 = ", x1, "; ", "x2 = ", x2
```

Toán tử

```
+ - * /
// (chia làm tròn)
% (phần dư)
** (lũy thừa)
~ (not)  & (and)  | (or)   ^ (xor)
<< (left shift)
>> (right shift)
== (bằng)  <= (nhỏ hơn hoặc bằng)  >= (lớn hơn hoặc bằng)
```

```
!= (khác)
```

Python sử dụng ký pháp trung tố thường gấp trong các ngôn ngữ lập trình khác.

Chú thích

```
# Comments
"""
Comments
Comments
"""
```

Lệnh gán

```
#tên biến = biểu thức
x = 23.8
y = -x ** 2
z1 = z2 = x + y
loiChao = "Hello!"

i += 1    # tăng biến i thêm 1 đơn vị
```

In giá trị

```
print biểu thức
print (7 + 8) / 2.0
print (2 + 3j) * (4 - 6j)
```

Nội suy chuỗi (string interpolation)

```
print "Hello %s" %("world!")
print "i = %d" %i
print "a = %.2f and b = %.3f" %(a,b)
```

Cấu trúc rẽ nhánh

Dạng 1:

```
if biểu_thức_điều_kiện:
    # lệnh...
```

Dạng 2:

```
if biểu_thức_điều_kiện:
    # lệnh...
else:
    # lệnh...
```

Dạng 3:

```

if biểu_thức_điều_kiện_1:
    # lệnh... (được thực hiện nếu biểu_thức_điều_kiện_1 là đúng/true)
elif biểu_thức_điều_kiện_2:
    # lệnh... (được thực hiện nếu biểu_thức_điều_kiện_1 là sai/false,
    # và biểu_thức_điều_kiện_2 là đúng/true)
else:
    # lệnh... (được thực hiện nếu tất cả các biểu_thức_điều_kiện_1 là sai/false)

```

Cấu trúc lặp

```

while biểu_thức_đúng:
    # lệnh...
    for phần_tử in dãy:
        # lệnh...

L = ["Ha Noi", "Hai Phong", "TP Ho Chi Minh"]
for thanhPho in L:
    print thanhPho

for i in range(10):
    print i

```

Hàm

```

def tên_hàm (tham_biến_1, tham_biến_2, tham_biến_n):
    # lệnh...
    return giá_trị_hàm
def binhPhuong(x):
    return x*x

```

Hàm với tham số mặc định

```

def luyThua(x, n=2):
    """Lũy thừa với số mũ mặc định là 2"""
    return x**n

luyThua(3)    # 9
print luyThua(2,3) # 8

```

Lớp - class

```

if __name__ == "__main__":
    main()

class Tên_Lớp_1:
    #...

class Tên_Lớp_2(Tên_Lớp_1):

```

```

"""Lớp 2 kế thừa lớp 1"""
x = 3 # biến thành viên của lớp
#
def phương_thức(self, tham_biến):
    #...

# khởi tạo
a = Tên_Lớp_2()
print a.x
print a.phương_thức(m) # m là giá trị gán cho tham biến

```

Xử lý ngoại lệ

```

try:
    câu_lệnh
except Loại_Lỗi:
    thông_báo_lỗi

```

Tốc độ thực hiện

Là một ngôn ngữ thông dịch, Python có tốc độ thực hiện chậm hơn nhiều lần so với các ngôn ngữ biên dịch như Fortran, C, v.v... Trong số các ngôn ngữ thông dịch, Python được đánh giá nhanh hơn Ruby và Tcl, nhưng chậm hơn Lua.

11.2 CÀI ĐẶT HỆ ĐIỀU HÀNH RASBIAN CHO RASPBERRY PI 3B

Để cài đặt hệ điều hành Rasbian cho Raspberry Pi 3 model B ta cần:

- Phần mềm Win32 Disk Imager (WDI).
- Một thẻ nhớ MicroSD tối thiểu 4GB. Nhưng chúng ta nên sử dụng thẻ 8GB class 10 trở lên để có dung lượng đảm bảo và tốc độ tốt nhất.

Các bước cài đặt hệ điều hành cho Raspberry:

Bước 1:

Chèn thẻ MicroSD vào đầu đọc thẻ của máy tính và kiểm tra tên ổ được gán cho thẻ nhớ (ví dụ ổ H:), tránh nhầm ổ dẫn đến mất dữ liệu vì phần mềm sẽ format thẻ MicroSD.

Bước 2:

Mở phần mềm Win32DiskImager, phần mềm này chỉ cần download về rồi chạy mà không cần cài đặt.

Bước 3:

Lựa chọn file hệ điều hành đã tải về. (Lưu ý, hệ điều hành cần phải ở định dạng .img. Thông thường, hệ điều hành của RPI được nén dưới dạng .zip

hoặc .tar.gz,... Khi tải về chúng ta cần giải nén nó ra để có file hệ điều hành dạng .img). Sau đó lựa chọn ổ thẻ nhớ cần ghi.

Bước 4:

Cuối cùng bạn bấm Write và chờ đợi quá trình ghi hoàn tất.

11.3 CÀI ĐẶT OPEN CV CHO RASPBERRY PI 3 MODEL B.

OpenCV là một thư viện mã nguồn mở phục vụ cho việc nghiên cứu hay phát triển về thị giác máy tính. Tối ưu hóa và xử lý các ứng dụng trong thời gian thực. Giúp cho việc xây dựng các ứng dụng xử lý ảnh, thị giác máy tính,... một cách nhanh hơn. OpenCV có hơn 500 hàm khác nhau, được chia làm nhiều phần phục vụ các công việc như: xử lý hình ảnh y tế, an ninh, camera quan sát, nhận diện, robots,...

Trước khi cài đặt Open CV cho Raspberry Pi 3 model B thì chúng ta phải cài đặt hệ điều hành Raspbian trước, sau đó chúng ta thực hiện theo các bước sau đây:

Những thiết lập ban đầu

Đầu tiên cần phải cập nhật, nâng cấp các gói hiện có, tiếp theo là cập nhật phần mềm RaspberryPi.

```
sudo apt-get update  
sudo apt-get upgrade  
sudo rpi-update
```

Khởi động lại Raspberry Pi sau khi cập nhật phần mềm:

```
sudo reboot
```

Tiếp theo là cài đặt công cụ phát triển:

```
sudo apt-get install build-essential git cmake pkg-config
```

Cài đặt gói ảnh I/O cho phép chúng ta tải các định dạng tệp hình ảnh như JPEG, PNG, TIFF, v..v

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev  
sudo apt-get install libxvidcore-dev libx264-dev
```

Cài đặt thư viện phát triển GTK (công cụ tạo giao diện đồ họa) để chúng ta có thể biên soạn modun phục của OpenCV, cho phép chúng ta hiển thị hình ảnh trên màn hình và xây dựng các giao diện GUI đơn giản:

```
sudo apt-get install libgtk2.0-dev
```

Cuối cùng, chúng ta cần cài đặt các tập tin Python 2.7 và Python 3 để có thể biên dịch OpenCV + Python (ngôn ngữ biên dịch) mặc định.

```
sudo apt-get install python2.7-dev python3-dev
```

Lấy mã nguồn OpenCV:

```
cd ~  
wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip  
unzip opencv.zip
```

Để cài đặt đầy đủ OpenCV 3 (bao gồm các tính năng như SIFT (nhận dạng điểm đặc trưng) và SURF (nhận dạng đối tượng)) ta phải chắc chắn việc lấy kho lưu trữ của OpenCV.

Tiếp theo là thiết lập để xây dựng opencv:

```
cd ~/opencv-3.3.0/  
mkdir build  
cd build  
cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D INSTALL_C_EXAMPLE=ON \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \  
-D BUILD_EXAMPLES=ON ..
```

Biên dịch OpenCV:

```
make -j4
```

Comment “-j4” là viết tắt của số lõi CPU để sử dụng khi biên soạn OpenCV. Vì chúng ta đang sử dụng Raspberry Pi 3, chúng ta sẽ tận dụng tất cả bốn lõi của bộ vi xử lý để build nhanh hơn.

Giả sử OpenCV biên soạn không bị lỗi và cài đặt nó vào hệ thống bằng lệnh:

```
sudo make install  
sudo ldconfig
```

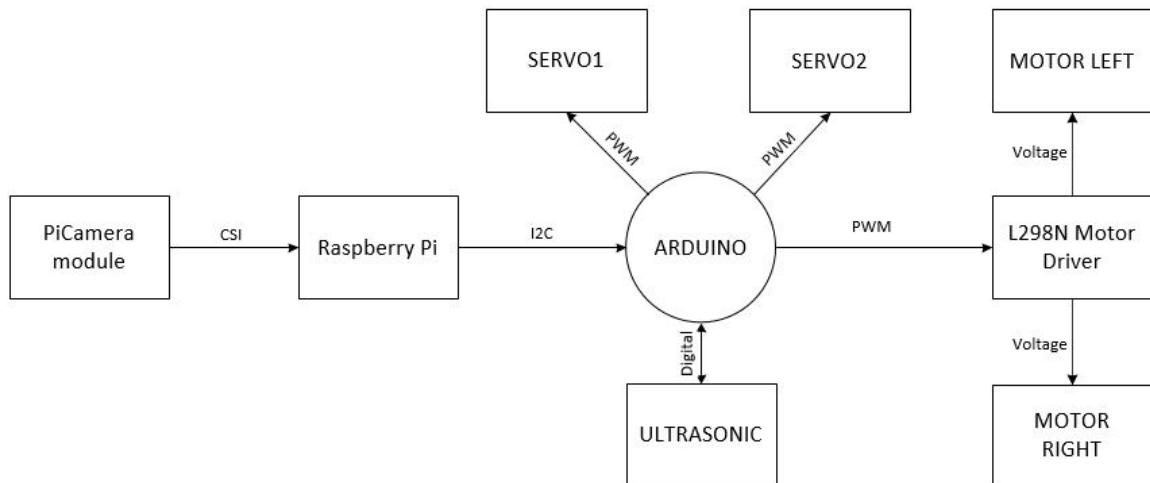
CHƯƠNG 12: HARDWARE

12.1 GIỚI THIỆU.

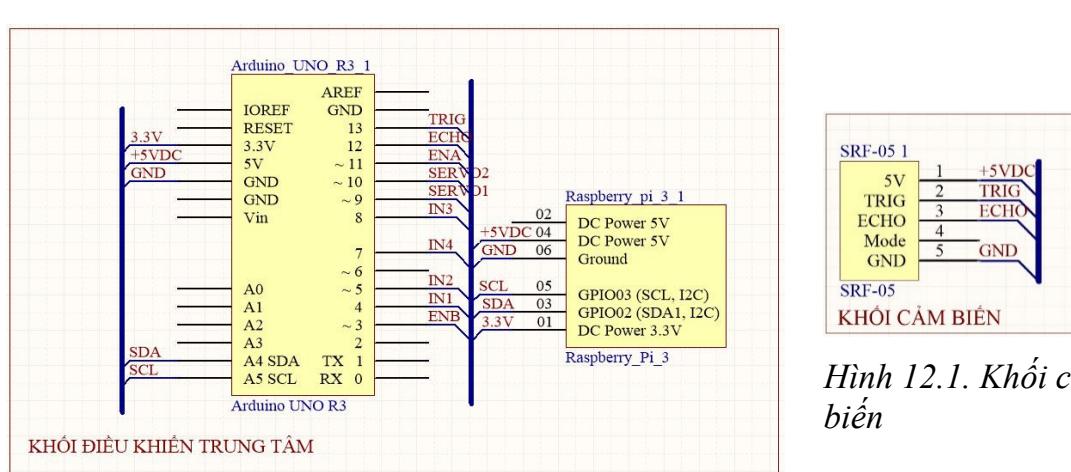
Kết hợp với máy tính nhúng Raspberry Pi và các thiết bị, driver như L298, DC motor, Pin LiPo, Servo motor và dây bus.

Robot được di chuyển nhờ điều khiển hai động cơ sử dụng cầu H. Hai Servo Motor dùng để điều chỉnh góc nhìn của camera.

12.2 SƠ ĐỒ KHỐI CỦA HỆ THỐNG

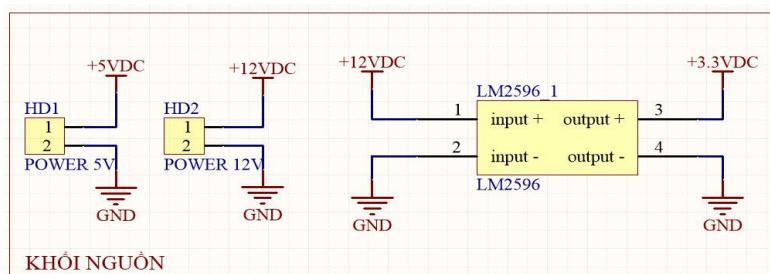


12.3 SƠ ĐỒ NGUYÊN LÝ

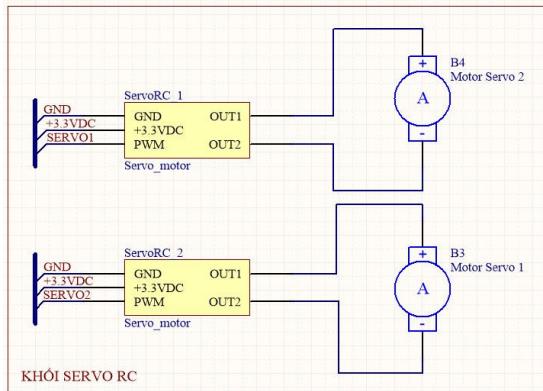


Hình 12.1. Khối cảm biến

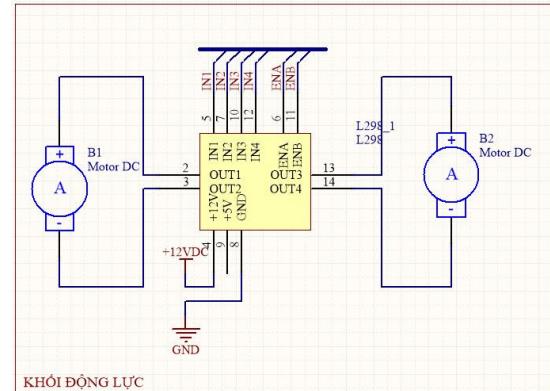
Hình 12.2. Khối điều khiển trung tâm



Hình 12.3. Khối nguồn



Hình 12.4. Khối servo



Hình 12.5. Khối động lực

12.4 RASPBERRY PI 3 MODEL B

12.4.1 Giới thiệu chung.

Raspberry Pi là máy tính nhúng giá khoảng 1.000.000 VNĐ kích cỡ như một cái thẻ ATM và chạy hệ điều hành Linux. Với mục tiêu chính của chương trình là giảng dạy máy tính cho trẻ em. Được phát triển bởi Raspberry Pi Foundation - là tổ chức phi lợi nhuận với tiêu chí xây dựng hệ thống mà nhiều người có thể sử dụng được trong những công việc tùy biến khác nhau.

Đặc tính của Raspberry Pi xây dựng xoay quanh bộ xử lý SoC Broadcom BCM2835 (là chip xử lý mobile mạnh mẽ có kích thước nhỏ hay được dùng trong điện thoại di động) bao gồm CPU, GPU, bộ xử lý âm thanh - video, và các tính năng khác...tất cả được tích hợp bên trong chip có điện năng thấp này.

12.4.2 Thông số kỹ thuật.

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE).
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL,...
Memory	1GB LPDDR2
Operating System	Boots từ thẻ nhớ, chạy phiên bản hệ điều hành Linux, Windows 10 IoT.
Dimensions	85 x 56 x 17mm.
Power	Micro USB socket 5 V1, 2.5A.

Connectors

Ethernet	10/100 BaseT Ethernet socket.
Video Output	HDMI (rev 1.3 & 1.4, Composite RCA (PAL and NTSC).

Audio Output	Audio Output 3.5mm jack, HDMI, USB 4 x USB 2.0 Connector.
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines.
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane.
Memory Card Slot	Push/pull Micro SDIO.

12.4.3 Ứng dụng.

- Điều khiển robot, máy in không dây từ xa, Airplay...
- Đầu coi phim HD giống như Android Box, hỗ trợ KODI đầy đủ.
- Máy chơi game cầm tay, console, game thùng. Chơi như máy điện tử băng ngày xưa, giả lập được nhiều hệ máy.
- Cắm máy tải Torrent 24/24.
- Dùng làm VPN cá nhân.
- Biến ổ cứng bình thường thành ổ cứng mạng (NAS).
- Làm camera an ninh, quan sát từ xa.
- Hiển thị thời tiết, hiển thị thông tin mạng nội bộ...
- Máy nghe nhạc, máy đọc sách.
- Làm thành một cái máy Terminal di động có màn hình, bàn phím, pin dự phòng để sử dụng mọi lúc mọi nơi, dò pass Wi-Fi...
- Làm thiết bị điều khiển Smart Home, điều khiển mọi thiết bị điện tử trong nhà.

12.4.4 Ưu điểm.

- Giá rẻ.
- Nhỏ gọn.
- Siêu tiết kiệm điện.
- GPU mạnh.
- Phục vụ cho nhiều mục đích.
- Khả năng hoạt động liên tục 24/7.

12.4.5 Nhược điểm

- CPU cấu hình thấp.
- Yêu cầu phải có kiến thức cơ bản về Linux, điện tử.

12.5 ADRUINO UNO BOARD

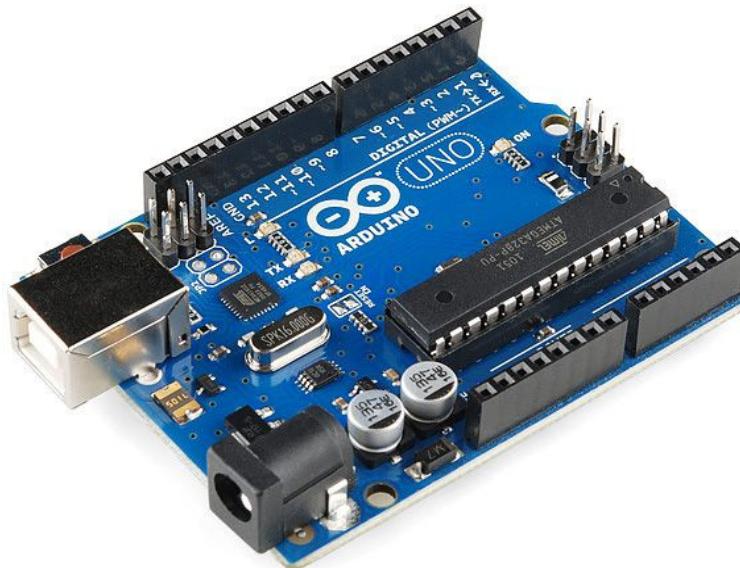
12.5.1 Khái niệm

Arduino là một board mạch vi xử lý, nhằm xây dựng các ứng dụng tương tác với nhau hoặc với môi trường được thuận lợi hơn.

Phần cứng bao gồm một board mạch nguồn mở được thiết kế trên nền tảng vi xử lý AVR atmel 8bit, hoặc ARM atmel 32 bit.

12.5.2 Các loại Arduino thông dụng

Các arduino thông dụng như Arduino Uno R3, Arduino Mega 2560, Arduino Leonardo, Arduino Due, Arduino Micro, Arduino Nano, Arduino Mini .v.v.....



Hình 12.6. Arduino UNO R3

12.5.3 Arduino Uno R3

Bảng 12.1. Thông số kỹ thuật

Vi điều khiển	Atmega328 họ 8bit
Điện áp hoạt động	5V DC (chỉ được cấp qua cổng USB)
Tần số hoạt động	16 MHz
Dòng tiêu thụ	Khoảng 30mA
Điện áp khuyên dùng	7 – 12V DC
Điện áp giới hạn	6 – 20V DC
Số chân Digital I/O	14 (6 chân PWM)
Số chân Analog	6 (độ phân giải 10bit)
Dòng tối đa trên mỗi chân I/O	30mA
Dòng ra tối đa (5V)	500mA
Dòng ra tối đa (3.3V)	50mA
Bộ nhớ flash	32KB (Atmega328) với 0.5KB dùng bởi bootloader
SRAM	2KB (Atmega328)
EFROM	1KB (Atmega328)

12.6 CẦU H (DRIVER L298D)

12.6.1 Nguyên lý cầu H

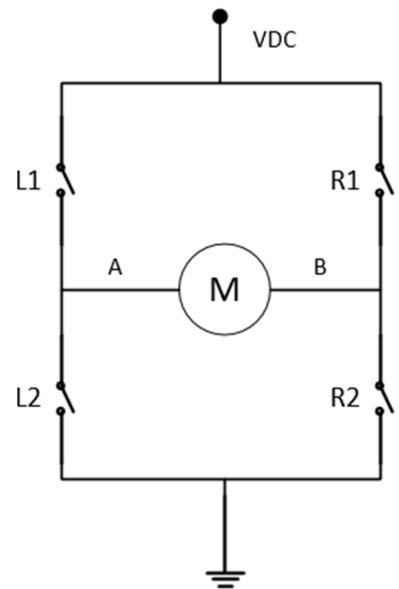
Với động cơ DC, khi ta nối một đầu dây của nó vào cực + của pin, đầu dây còn lại của động cơ nối vào cực – của pin thì động cơ sẽ quay, khi ta đảo hai dây đó thì động cơ sẽ quay với chiều ngược lại.

Dựa vào tính chất này, ta sẽ dùng một cầu H hoạt động dựa trên sự đóng ngắt các khoá L/R để điều khiển chiều của động cơ.

Khi hai khoá L1 và R2 đóng, khoá R1 và L2 mở thì áp áp xuống GND theo chiều từ A sang B qua động cơ. Động cơ quay theo chiều kim đồng hồ.

Khi hai khoá L1 và R2 mở, khoá R1 và L2 đóng thì áp áp xuống GND theo chiều từ B sang A qua động cơ. Động cơ quay ngược chiều kim đồng hồ.

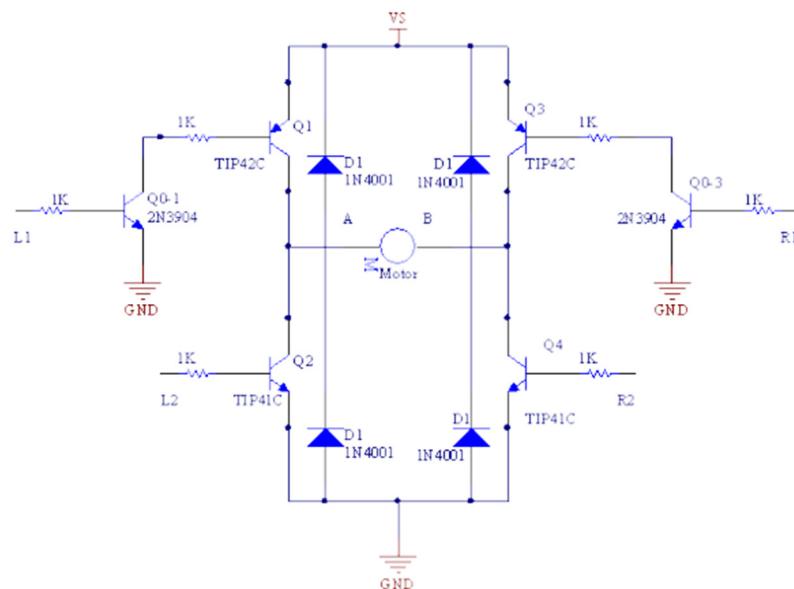
Và hiển nhiên không có trường hợp 4 khoá đóng cùng một lúc vì như vậy sẽ gây ra hiện tượng ngắn mạch làm hỏng hoặc có thể cháy nổ.



Hình 12.7. Cầu H

Để hoàn thiện việc điều khiển chiều quay và tốc độ của động cơ, người ta đã thiết kế cầu H với linh kiện BJT, relay hay MOSFET...

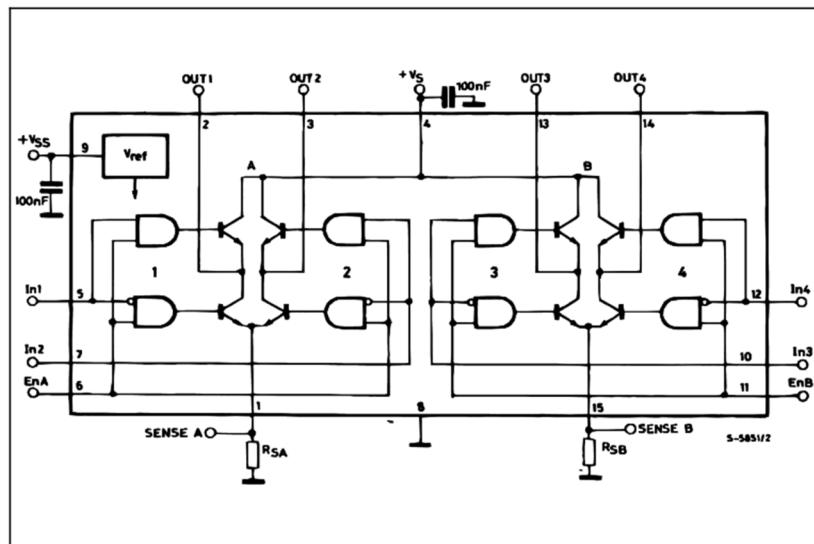
Hình 12.4 là Sơ đồ nguyên lý mạch cầu H dùng BJT.



Hình 12.8. Cầu H dùng BJT

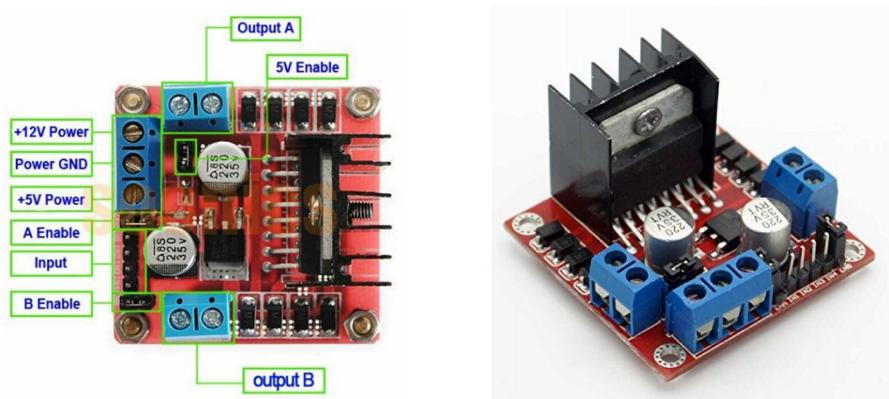
12.6.2 Điều khiển động cơ sử dụng L298D

Hình 10.3 là sơ đồ khối của L298D và breakout board của L298D



Hình 12.9. Cầu H trong L298D

Thông số kỹ thuật:



Hình 12.10. L298D Driver

Driver L298 tích hợp hai cầu H có điện áp điều khiển từ +5VDC đến +12VDC.

Dòng tối đa cho mỗi cầu H là 2A.

Dòng của tín hiệu điều khiển 0 đến 36mA (max là 40mA)

Công suất hao phí 20W ($T = 75^\circ C$)

Cách sử dụng:

- Chân 12V power và chân 5V power là 2 chân cấp nguồn trực tiếp đến động cơ. Ở chân 12V có thể cấp nguồn từ 9 đến 12V. Khi để jumper 5V bên cạnh như hình thì cổng 5V power sẽ là ngõ ra áp 5V.

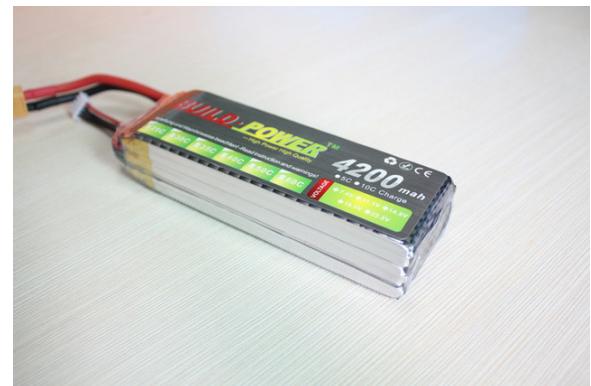
- Chân GND là chân GND của nguồn cấp cho động cơ, phải nối chung với GND của arduino để điều khiển.
- Hai jump A enable và B enable khi tháo ra, ta sẽ dùng một chân ở mỗi bên để cấp PWM, dùng điều khiển tốc độ động cơ.
- Chân IN1, IN2, IN3 và IN4 là các chân digital dùng để điều khiển chiều quay của động cơ, hoặc cũng dùng để điều khiển tốc độ. Hai jump ở A enable và B enable có thể không tháo ra vì không cần dùng tới khi dùng 4 chân digital để điều khiển tốc độ.
- Chân output A và output B được nối lần lượt vào hai động cơ. Khi điều khiển cho động cơ chạy nếu động cơ quay không đúng chiều quy ước ta có thể đảo hai dây của động cơ ở ngõ output.

Các chân INx được nối vào các chân digital của arduino, hai chân A enable và B enable được nối vào chân PWM của arduino.(với trường hợp đấu dây này ta sẽ tháo jump ở A enable và B enable).

12.7 PIN LIPO

Pin LiPo có tên đầy đủ là Lithium-Ion Polymer người ta rút gọn thành Lithium-Polymer để tránh nhầm lẫn với Li-Ion.

Pin LiPo không sử dụng chất điện phân dạng lỏng mà thay vào đó nó sử dụng chất điện phân dạng polymer khô, tương tự như một miếng phim nhựa mỏng. Miếng phim này được kẹp (thực sự là ghép lá) giữa cực dương và cực âm của pin cho phép trao đổi ion - do đó có tên là lithium polymer.



Hình 12.11. Pin Lipo

Phương pháp này cho phép Pin có thể làm rất mỏng với các hình dạng và kích thước của cell pin khác nhau.

Ưu điểm:

- Pin lipo có dòng điện rò rỉ thấp. Nếu so với lithium có cùng kích cỡ thì pin Lipo cho khả năng lưu trữ nhiều hơn.
- Trọng lượng nhẹ và khả năng phóng điện cao nên được áp dụng cho hầu hết các thiết bị số. Ngoài ra do cấu tạo là gel polymer nên có thể chịu va đập khá hơn pin Lithium-Ion, vậy nên cũng thường được áp dụng cho các thiết bị điều khiển radio như mô hình máy bay.
- Cũng giống như pin lithium ion chúng không bị hiệu ứng nhớ và ưu điểm nhất là chúng có thể tùy biến hình dáng và kích thước để phù hợp với các sản phẩm

khác nhau. Bởi thế, nó đang là chuẩn pin được nhiều nhà sản xuất điện thoại tin dùng.

Nhược điểm: Giá thành chi phí xuất xưởng cao.

12.8 ĐỘNG CƠ DC 12V

Động cơ được sử dụng cho đề tài là GA37 V1, có hộp giảm tốc với tỷ số truyền 1:33. Động cơ này phù hợp để làm xe dò line, xe chạy mè cung và xe hai bánh tự cân bằng...

Động cơ có encoder từ trường hai kênh AB và 11 xung/vòng. Số xung sau giảm tốc là 363xung/vòng.



Hình 12.12. Động cơ DC

Động cơ GA37 gồm có:

- Chân M1 và M2 là hai chân cấp nguồn tối đa 12VDC.
- Chân GND là chân mass của encoder
- Chân C1 (kênh A) và chân C2(kênh B) là hai chân output của encoder 3.3VDC
- Chân 3.3 là nguồn 3.3VDC cho encoder.

Thông số kỹ thuật của động cơ:

- Điện áp tối đa: 12VDC
- Dòng tối đa 3A
- Tốc độ động cơ: 10000rpm
- Tỷ số truyền 1:33
- Tốc độ sau giảm tốc: 303rpm
- Đường kính trực 6mm

Thông số kỹ thuật của encoder:

- Điện áp: 3.3VDC
- Encoder: đĩa từ có 11 xung, 2 kênh AB
- Số xung sau giảm tốc $11 \times 33 = 363$ ppr

12.9 PICAMERA MODULE

12.9.1 Giới thiệu

Raspberry Pi camera là module camera được chính Raspberry Pi Foundation thiết kế và đưa vào sản xuất đại trà từ tháng 5/2013.

Trước khi xuất hiện camera, điều duy nhất chúng ta có thể làm để thêm khả năng nhận biết hình ảnh, quay phim, chụp hình cho Raspberry Pi là sử dụng một webcam cắm vào cổng USB. Với các webcam Logitech tích hợp sẵn định dạng xuất jpeg sẽ giúp Raspberry xử lý nhanh hơn. Nhưng các webcam Logitech lại có giá thành khá cao, nhất là các webcam có độ phân giải lớn.

Raspberry Pi camera được tích hợp camera 5 Megapixel có độ nhạy sáng cao, có thể chụp tốt ở nhiều điều kiện ánh sáng khác nhau, cả trong nhà và ngoài trời. Điểm đặc biệt mà camera mang lại đó là chụp hình độ nét cao trong lúc quay phim.

Chúng ta không cần thêm cổng USB nào cho camera vì camera được gắn chắc chắn vào socket CSI. Điều này giúp hạn chế tình trạng nghẽn băng thông cho chip xử lý USB trên mạch Raspberry.

12.9.2 Thông số kỹ thuật.

Ống kính tiêu cự cố định.

Cảm biến độ phân giải 5 megapixel.

Hỗ trợ video:

$1080p\ 30\ fps, 720p\ 60\ fps$ và $640x480p\ 90\ fps$.

Kích thước: $25mm \times 23mm \times 9mm$.

Trọng lượng: 3g.



Hình 12.13. PiCamera Module

12.10 SERVO RC

12.10.1 Giới thiệu

Động cơ Servo RC 9G có kích thước nhỏ gọn phù hợp để làm các xe mô hình, máy bay mô hình...Động cơ Servo RC 9G có tốc độ đáp ứng nhanh, các bánh răng được làm bằng nhựa nên chú ý tải trọng khi nâng.

Servo này có driver điều khiển bên trong nên dễ dàng điều khiển góc quay bằng phương pháp PWM.

12.10.2 Thông số kỹ thuật

Điện áp hoạt động: $4.8 \div 5V\ DC$

Tốc độ: $0.12sec/60degrees\ (4.8V\ DC)$

Momen: $1.6kG.cm$

Kích thước: $21 \times 12 \times 22mm$

Trọng lượng: 9g



Hình 12.14. Servo RC

12.11 CẢM BIẾN SIÊU ÂM SRF-05

12.11.1 Nguyên lý hoạt động

Cảm biến siêu âm phát sóng âm tần số ngắn, tần số cao theo khoảng thời gian đều đặng. Chùm sóng âm sẽ phản hồi lại khi gặp vật cản (vật rắn, chất lỏng...). Dựa vào thời gian phát, thời gian phản hồi và vận tốc truyền ta sẽ tính được khoảng cách từ cảm biến đến đối tượng cần tính khoảng cách.

12.11.2 Cảm biến SRF-05

Cảm biến SRF-05 được sử dụng phổ biến trong các ứng dụng IoT, vì nó có giá thành thấp, chất lượng đảm bảo và dễ sử dụng.

Cảm biến SRF-05 là bản nâng cấp của SRF-04, cảm biến siêu âm này dùng để xác định khoảng cách trong bộ phận nhỏ. SRF-05 đo khoảng cách chính xác tốt và ổn định.



Hình 12.15. SRF-05

12.11.3 Thông số kỹ thuật của SRF - 05

- Điện áp định mức:	5V
- Dòng tiêu thụ:	4mA
- Tín hiệu ra:	Pulse (hight level – 5V, low level – 0)
- Tần số sóng âm	40kHz
- Chân TRIGGER	Tối thiểu $10\mu s$, chuẩn xung mức TTL
- Chân ECHO	Tín hiệu xung mức dương TTL
- Giới hạn đo:	$3cm \div 450cm$
- Độ chính xác:	0.5cm
- Kích thước:	$20x45x15mm$
- Góc cảm biến:	$< 15^\circ$
- Loại kết nối:	Digital

12.11.4 Cách sử dụng SRF-05 và xác định khoảng cách bằng SRF-05

Để đo khoảng cách, ta sẽ phát 1 xung tối thiểu $10\mu s$ vào chân TRIGGER. Sau đó cảm biến sẽ tạo một Pulse High ở chân ECHO cho đến khi nhận lại sóng phản hồi ở chân ECHO này và nó được kéo về 0.

Tốc độ âm thanh trong không khí là $340m/s$ (hằng số vật lý). Khi có được thời gian truyền đến vật thể và thời gian nhận lại tín hiệu sóng âm ta sẽ tính được khoảng cách.

Cấu hình chân của SRF-05:

- Vcc: Nguồn cấp cho cảm biến
- Trigger (chân input): kích hoạt quá trình phát sóng âm, tối thiểu $10\mu s$.
- Echo (chân output): bình thường là mức 0V, trong trạng thái kích hoạt sẽ là 5V cho đến khi cảm biến nhận lại sóng âm và được kéo về mức 0.
- GND: Chân mass
- OUT: chân này để hở

SRF-05 có hai chế độ hoạt động:

- **Mode 1:** Sử dụng hai chân TRIGGER và ECHO

Hoạt động SRF-04 và SRF-05 trong chế độ này là như nhau. Chân OUT để hở.

- **Mode 2:** Dùng một chân cho cả TRIGGER và ECHO

Chân OUT kéo xuống mass.

12.12 KẾT NỐI CÁC THIẾT BỊ.

Bảng 12.2. Tên chân sẽ kết nối với nhau

	Raspberry Pi	Arduino	L298D	Servo	SRF-05
UART	Sử dụng cáp USB				
PWM		11	ENA		
		3	ENB		
		9		RED (servo1)	
		10		RED (servo2)	
DIGITAL		4, 5, 8, 7	IN1, IN2, IN3, IN4		
		13, 12			Trig, Echo
		5V, GND			VCC, GND
I2C	3	4			
	5	5			
	1	3.3V			
	7	GND			GND

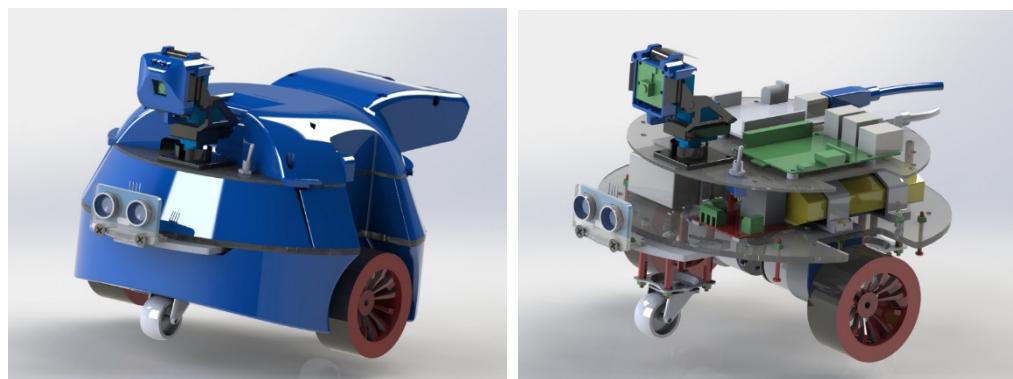
12.13 BẢN VẼ CHI TIẾT VÀ BẢN VẼ LẮP CỦA MÔ HÌNH

ROBOT TỰ HÀNH PHÁT HIỆN LÀN ĐƯỜNG DÙNG KỸ THUẬT XỬ LÝ ẢNH

12.14 HÌNH ẢNH VỀ ROBOT CỦA NHÓM



Hình 12.16. Hình ảnh robot thực tế

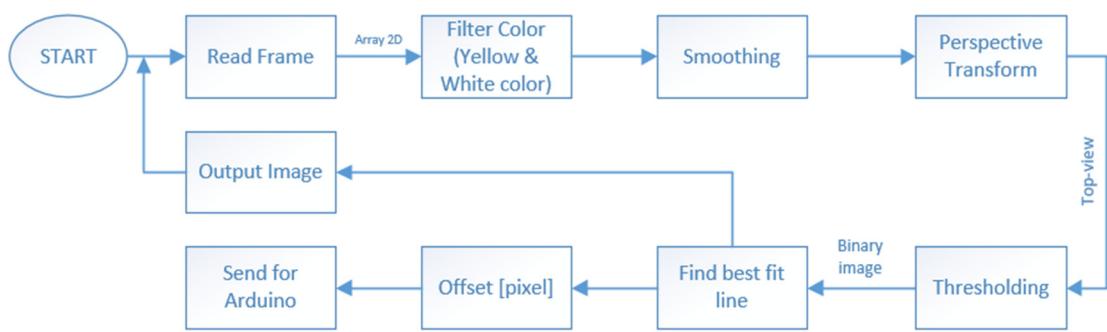


Hình 12.17. Hình ảnh mô hình robot được thiết kế trên Solidworks

CHƯƠNG 13: GIẢI THUẬT VÀ SOFTWARE

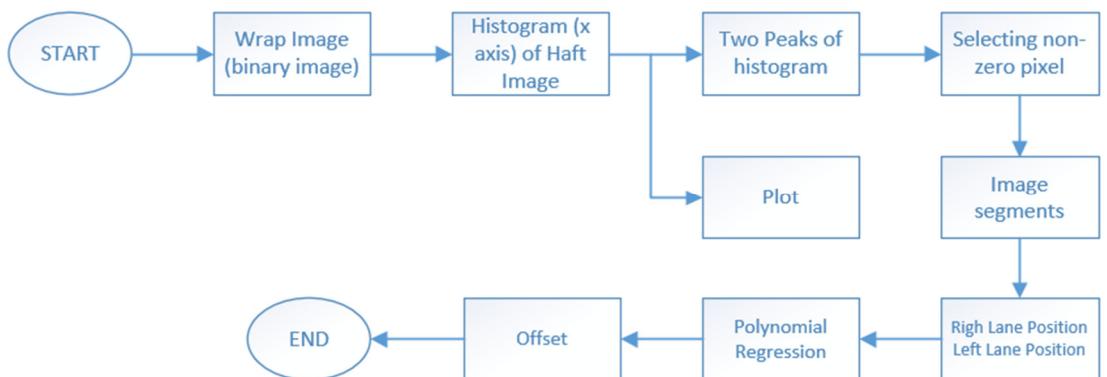
13.1 GIẢI THUẬT.

Giải thuật xử lý ảnh



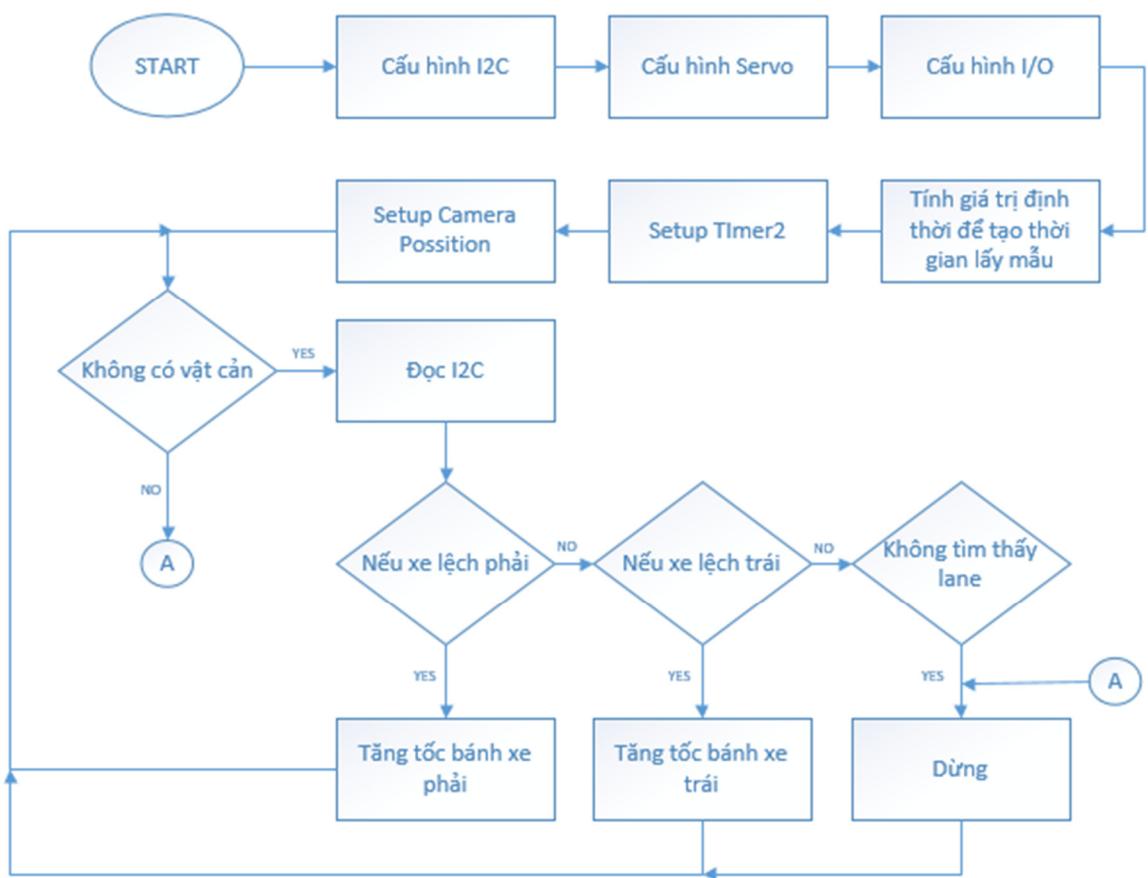
Sơ đồ 13.1. Giải thuật xử lý ảnh

Giải thuật tìm best fit line



Sơ đồ 13.2. Giải thuật tìm best fit line

Giải thuật điều khiển robot



Sơ đồ 13.3. Giải thuật điều khiển robot

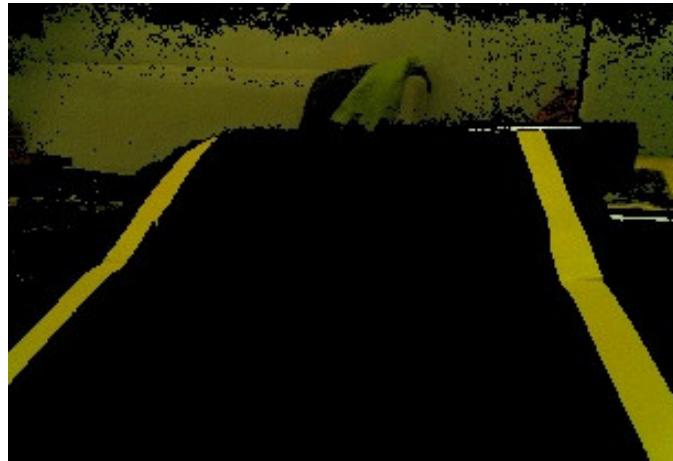
13.2 KẾT QUẢ QUÁ TRÌNH XỬ LÝ ẢNH

13.2.1 Read Frame (frame size = (304,208)):



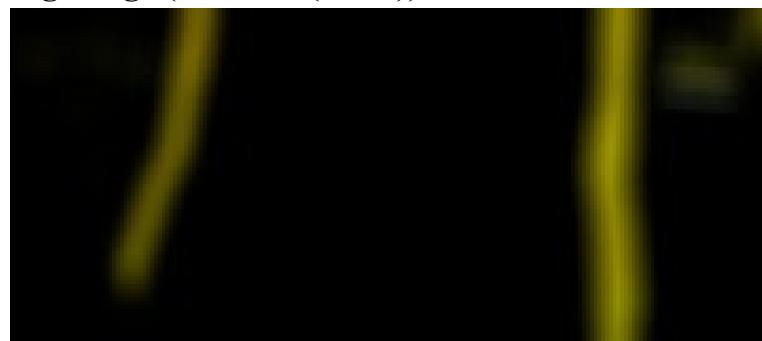
Hình 13.1. Orgirinal Frame

13.2.2 Filter Color:



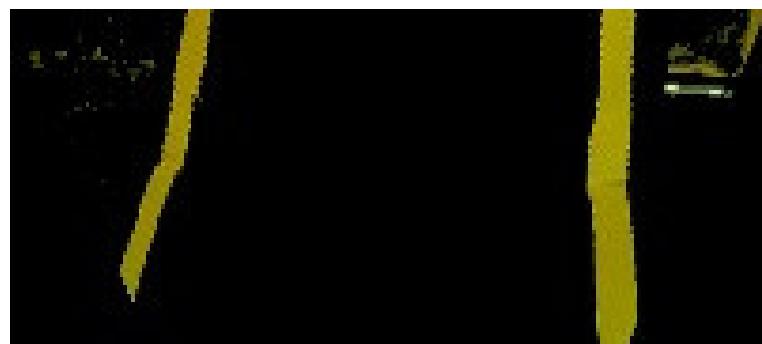
Hình 13.2. Filter Color

13.2.3 Smoothing image (kernel = (11,11)):



Hình 13.3. Smoothing Image

13.2.4 Color warp image(warp size = (200, 80)):



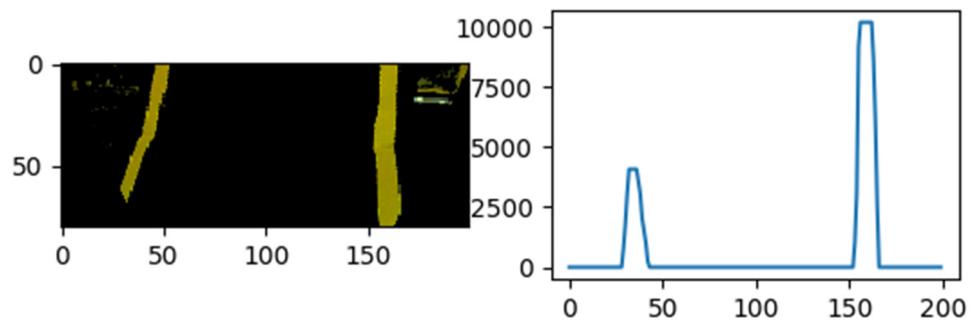
Hình 13.4. Color warp image

13.2.5 Thresholding image:



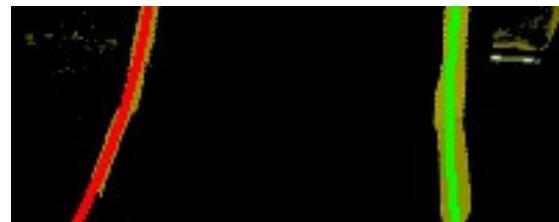
Hình 13.5. Thresholding Image

13.2.6 Histogram theo trục x:



Hình 13.6. Histogram

13.2.7 Find best fit line:



Hình 13.7. Best fit line

13.3 CHƯƠNG TRÌNH

13.3.1 Chương trình Python trên Raspberry Pi 3

a. Chương trình LaneLineDetection.py

```
# INDUSTRIAL UNIVERSITY OF HCM CITY      #
# INSTRUCTORS: TS. NGUYEN VIEN QUOC       #
# TEAM "Do an tot nghiep 2018"           #
# -----#
# Name program: LaneLineDetection.py      #
# -----#
import time
import warnings
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

```

from subfunctions import *
#import sys
#sys.path.append('/usr/local/lib/python3.5/site-packages')
#from picamera import PiCamera
#from picamera.array import PiRGBArray as array
#import smbus

img_size = (304, 208)

def Picamera():
    cap = PiCamera(resolution=img_size)
    time.sleep(2)
    rawCapture = array(cap)
    for frame in cap.capture_continuous(rawCapture, format='bgr',
use_video_port=True):
        img = frame.array
        img = Processing(img)
        cv2.imshow('Result', img)
        rawCapture.truncate(0)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()

def Image():
    img = cv2.imread('pictures\img8.jpg')
    img = Processing(img)
    cv2.imshow('Result', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main():
    # Picamera()
    Image()

if __name__=="__main__":
    main()

```

b. Chương trình subfunctions.py

```

# INDUSTRIAL UNIVERSITY OF HCM CITY          #
# INSTRUCTORS: TS. NGUYEN VIEN QUOC          #
# TEAM "Do an tot nghiep 2018"               #
# -----#
# Name program: subfunctions.py              #
# -----#


# import sys
# sys.path.append('/usr/local/lib/python3.5/site-packages')
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

# Image size #
img_size = (304, 208) # for PI
# img_size = (640, 480) # for Desktop
warp_size = (200, 80)
add_array = np.zeros([232, 200], dtype=np.uint8)

```

```

# Parameters #
thresh_for_binary = 140
# Distance between two lane actual #
lane_distance_m = 0.25      # [m]
# Distance between two lane on image #
lane_distance = 122          # [pixel]

# Create empty array to store indices of two lane #
left_indices = []
right_indices = []

# Zoom output image #
scale = 1.5

# Connect i2c with Arduino #
# Pin out #
# Pin 3(SDA)      -  Pin A4(SDA)
# Pin 5(SCL)      -  Pin A5(SCL)
# Pin 1(3.3V)     -  Pin 3.3V
# Pin 9(GND)      -  Pin GND
channel = 1
# Slaver Address, Arduino #
address = 0x08
#bus = smbus.SMBus(channel)

# Range to convert offset #
x = [-50, 50]  # on binary image 200x80
y = [0, 252]   # byte type
m = (y[1] - y[0])/(x[1]-x[0])

# This Subfunction use to call Preprocessing function and return final
# output image #
def Processing(img):
    erosion, birdeye = Preprocessing(img)
    polyfit_img = Polyfit(erosion, birdeye, img, plot=True)
    return polyfit_img

# This function is image pre-processing #
def Preprocessing(img):
    Filter = FilterColor(img)
    smoothing = Smoothing(Filter, kernel=11)
    birdeye = BirdEye(smoothing)
    binary_image = BinaryImage(birdeye, thresh_for_binary)
    binary_erosion = Morphological(binary_image, kernel=5)
    return binary_erosion, birdeye

# This function to filter yellow and white color #
def FilterColor(img):
    # Convert colorspace to filter color #
    conv = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)

    # Threshold for white lane #
    lower = np.uint8([0, 150, 0])
    upper = np.uint8([255, 255, 255])
    white_mask = cv2.inRange(conv, lower, upper)
    # Threshold for yellow lane #
    lower = np.uint8([10, 0, 150])
    upper = np.uint8([40, 255, 255])
    yellow_mask = cv2.inRange(conv, lower, upper)

```

```

#Create a mask #
mask = cv2.bitwise_or(white_mask, yellow_mask)
mask = cv2.bitwise_and(img, img, mask=mask)

return mask

# This function to filter noise use Gaussian Blur #
def Smoothing(img, kernel):
    # Filter noise using Gaussian Blur #
    return cv2.GaussianBlur(img, (kernel, kernel), 7)

# This function use to create region to warp #
def WarpPoint():
    rows = img_size[1]
    cols = img_size[0]
    src_points = np.float32([[cols*0.25, rows*0.375], [cols*0.1,
rows*0.95],
                                [cols*0.75, rows*0.375], [cols*0.9,
rows*0.95]])

    rows = warp_size[1]
    cols = warp_size[0]
    dst_points = np.float32([[cols*0.25, 0], [cols*0.25, rows],
[cols*0.75, 0], [cols*0.75, rows]])

    return src_points, dst_points

# This function use to create transform matrix from WarpPoint() #
def TransformMatrix(flag='invM'):
    src, dst = WarpPoint()
    if flag == 'invM':
        return cv2.getPerspectiveTransform(dst, src)
    if flag == 'M':
        return cv2.getPerspectiveTransform(src, dst)

# This function use to image top-view or bird'eye view #
def BirdEye(img):
    matrix = TransformMatrix(flag='M')
    return cv2.warpPerspective(img, matrix, warp_size,
flags=cv2.WARP_FILL_OUTLIERS+cv2.INTER_CUBIC)

# This function use to unwarp #
def Unwarp(img):
    matrix = TransformMatrix(flag='invM')
    return cv2.warpPerspective(img, matrix, img_size,
flags=cv2.WARP_FILL_OUTLIERS+cv2.INTER_CUBIC)

# This BinaryImage Subfunction used to convert image to binary image(only
two level gray) #
def BinaryImage(img, thresh=127):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, thresh, 255, cv2.THRESH_BINARY)
    return thresh

# This function to morphological for image #
def Morphological(gray_img, kernel):
    # Morphological a binary image #

```

```

kernel = np.ones((kernel, kernel), np.uint8)
return cv2.erode(gray_img, kernel, iterations=1)

# This function use to count number of peak #
def CountPeak(hist):
    count = 0
    pos1 = []
    pos2 = []
    dec = False
    inc = False

    for i in range(hist.shape[0]-1):
        if(hist[i] == 0):
            if(hist[i+1] == 0):
                continue
            if(hist[i+1] > 0):
                inc = True
                pos1.append(i)
        if(inc == True):
            count += 1
            inc = False

        if(hist[i] > 0):
            if(hist[i+1] > 0):
                continue
            if(hist[i+1] == 0):
                dec = True
                pos2.append(i+1)
    if(dec == True):
        dec = False

    #print("Number of peak: ", count)
    # print(pos1)
    # print(pos2)
    return count, pos1, pos2

# This function use to calculating offset value #
def CalculateOffset(hist, p_left, p_right, centerline_position):
    y = warp_size[1]-1
    x_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
    x_right = p_right[0]*y**2 + p_right[1]*y + p_right[2]

    point_offset = np.int((x_left + x_right)/2)
    offset = np.int(centerline_position - point_offset)
    return offset, point_offset

# This function use to check value in x and y is None or not None #
def CheckLaneNone(x, y):
    if len(x) == 0 or len(y) == 0:
        print('none lane')
        return False
    else:
        return True

# This function use to choose lane at true possition #
def ChooseLaneTrue(hist, p_left, p_right, y, left_peak, right_peak):
    # Counter peak #
    count_peak, pos1, pos2 = CountPeak(hist)

```

```

fy_left = np.zeros_like(y)
fy_right = np.ones(hist.shape[0])*hist.shape[0]
if count_peak == 0:
    return fy_left, fy_right
l_peak = np.argmax((hist[pos1[0]:pos2[0]-1]))+pos1[0]
r_peak = np.argmax((hist[pos1[-1]:pos2[-1]])) + pos1[-1]
# Finding lane position true #
if count_peak >= 2:
    distance = r_peak - l_peak
    if distance >= lane_distance:
        fy_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
        fy_right = p_right[0]*y**2 + p_right[1]*y + p_right[2]
    else:
        if hist[l_peak] > hist[r_peak] and l_peak <= 100:
            fy_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
            fy_right = np.ones(hist.shape[0])*hist.shape[0]
        if hist[l_peak] < hist[r_peak] and r_peak >= 100:
            fy_left = np.zeros_like(y)
            fy_right = p_right[0]*y**2 + p_right[1]*y + p_right[2]
        if hist[l_peak] == hist[r_peak]:
            if l_peak < 100:
                fy_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
                fy_right = np.ones(hist.shape[0])*hist.shape[0]
            if r_peak > 100:
                fy_left = np.zeros_like(y)
                fy_right = p_right[0]*y**2 + p_right[1]*y + \
p_right[2]
            if l_peak == 100 or r_peak == 100:
                fy_left = np.zeros_like(y)
                fy_right = np.ones(hist.shape[0])*hist.shape[0]
        if (count_peak > 0) and (count_peak < 2):
            if l_peak < 100:
                fy_left = p_left[0]*y**2 + p_left[1]*y + p_left[2]
                fy_right = np.ones(hist.shape[0])*hist.shape[0]
            if r_peak > 100:
                fy_left = np.zeros_like(y)
                fy_right = p_right[0]*y**2 + p_right[1]*y + p_right[2]
            if l_peak == 100 or r_peak == 100:
                fy_left = np.zeros_like(y)
                fy_right = np.ones(hist.shape[0])*hist.shape[0]
return fy_left, fy_right

# This function use to convert offset value to byte, from (49,149) to (0,
# 255) #
def ConvertByteRange(Input):
    return np.uint8(m*(Input - x[0]) + y[0])

# This Polyfit Subfunction used to find best fit line using Polynomial-
# Regresstion Algorithm #
def Polyfit(binary, color_warp, img, plot=False):
    # Region of image to calc histogram, haft-lower binary image #
    roi = binary[np.int(binary.shape[0]/2):, :]
    # Calculating for histogram #
    hist = np.sum(roi, axis=0)

    # Vertical centerline #
    centerline_position = np.int(hist.shape[0]/2)

    # Histogram Peaks of Haft-lower image #

```

```

left_peak = np.argmax(hist[:centerline_position])
right_peak = np.argmax(hist[centerline_position:]) + \
centerline_position

"""
print("Left peak position: ", left_peak)
print("Left peak value: ", hist[left_peak])
print("Right peak position: ", right_peak)
print("Right peak value: ", hist[right_peak])
"""

# Finding position of pixels which this pixels nonzero, dataset #
nonzero = binary.nonzero()
Y = np.array(nonzero[0])
X = np.array(nonzero[1])

# Number of Slide windows #
nWin = 10
# Size of Windows #
hWin = np.int(binary.shape[0]/nWin)      # Height of Windows #
wWin = 20                                # Width of Windows #
# Minimum Pixel on a window#
minPix = 20

# Initialize empty array to store indicates #
left_indices = []
right_indices = []
left_peak_current = left_peak
right_peak_current = right_peak

# Slide windows #
for i in range(nWin):
    y_upper = np.int(binary.shape[0] - i*hWin)
    y_lower = np.int(y_upper - hWin)

    left_upper = np.int(left_peak_current + wWin/2)
    left_lower = np.int(left_upper - wWin)

    right_upper = np.int(right_peak_current + wWin/2)
    right_lower = np.int(right_upper - wWin)

    # This function only return y indices #
    Y_left_indices = ((X >= left_lower) & (X < left_upper) &
                      (Y >= y_lower) & (Y < y_upper)).nonzero()[0]
    Y_right_indices = ((X >= right_lower) & (X < right_upper) &
                      (Y >= y_lower) & (Y < y_upper)).nonzero()[0]

    left_indices.append(Y_left_indices)
    right_indices.append(Y_right_indices)

    # Update peak of histogram of binary image #
    if len(Y_left_indices) > minPix:
        left_peak_current = np.mean(X[Y_left_indices])
    if len(Y_right_indices) > minPix:
        right_peak_current = np.mean(X[Y_right_indices])

if(len(binary.shape) == 2):
    binary = binary[:, :, np.newaxis]
    binary = cv2.merge((binary, binary, binary))

```

```

        cv2.rectangle(binary, (left_lower, y_lower),
                      (left_upper, y_upper), (0, 255, 0), 1)
        cv2.rectangle(binary, (right_lower, y_lower),
                      (right_upper, y_upper), (255, 0, 0), 1)

    if ((len(Y_left_indices) == 0) or (len(Y_right_indices) == 0)):
        continue

    left_indices = np.concatenate(left_indices)
    right_indices = np.concatenate(right_indices)

    # Points Coordinate on Windows of binary image, same as dataset to
#predict parameters of curve #
    x_left = X[left_indices]
    y_left = Y[left_indices]
    x_right = X[right_indices]
    y_right = Y[right_indices]

    # Print indices is nonzero #
    #print("x_left = ", x_left, ", y_left = ", y_left)
    #print("x_right = ", x_right, ", y_right = ", y_right)

    # Check Lane is None or not None #
    check_left = CheckLaneNone(x_left, y_left)
    check_right = CheckLaneNone(x_right, y_right)

    # Apply regression algorithm for dataset #
    # Initialize parameter of curvature #
    p_left = np.array([0, 0, 0])
    p_right = np.array([0, 0, 0])

    # Check condition #
    if check_left == True:
        p_left = np.polyfit(y_left, x_left, 2)
    if check_right == True:
        p_right = np.polyfit(y_right, x_right, 2)

    # Creat axis to plot #
    y = np.linspace(0, hist.shape[0]-1, hist.shape[0])

    fy_left, fy_right = ChooseLaneTrue(
        hist, p_left, p_right, y, left_peak, right_peak)
    # Calculating offset of car #
    offset, draw = CalculateOffset(hist, p_left, p_right,\ncenterline_position)
    #bus.write_byte(address, ConvertByteRange(offset))
    # Print offset value #
    print("Offset on binary: ", offset)
    print("Offset on image: ", np.int(offset*img_size[0]/warp_size[0]))

    # Concatenate points is nonzero on windows #
    points_left = np.array([np.transpose(np.vstack([fy_left, y]))])
    points_right = np.array([
        np.flipud(np.transpose(np.vstack([fy_right, y])))])
    points = np.hstack((points_left, points_right))

    # Fill region of two lane 3 #
    cv2.fillPoly(color_warp, np.int_(points), (0, 255, 0))
    # Drawing offset value #

```

```

cv2.line(img, (np.int(draw*img_size[0]/warp_size[0])), 0),\
        (np.int(draw*img_size[0]/warp_size[0]), 5), (0, 255, 0), 1)
# Drawing middle between of two lane #
cv2.line(img, (np.int(img_size[0]/2), 0),\
        (np.int(img_size[0]/2), 10), (255, 0, 0), 1)
# Unwarp image #
unwarp = Unwarp(color_warp)
# Final Output #
output = cv2.addWeighted(img, 1.0, unwarp, 0.3, 0.0)
output = cv2.resize(output, None, fx=scale, fy=scale)

# Plot #
if plot:
    fig = plt.figure()
    plot_img = cv2.cvtColor(binary, cv2.COLOR_BGR2RGB)
    fig.add_subplot(2, 3, 1)
    plt.imshow(cv2.cvtColor(color_warp, cv2.COLOR_BGR2RGB))
    plt.title('a. Original')

    fig.add_subplot(2, 3, 2)
    plt.imshow(plot_img)
    plt.title('b. Binary')

    fig.add_subplot(2, 3, 3)
    plt.plot(hist)
    plt.text(left_peak, hist[left_peak], str(hist[left_peak]))
    plt.text(right_peak, hist[right_peak], str(hist[right_peak]))
    plt.title('c. Histogram 1/2 of half-lower binary image')

    fig.add_subplot(2, 3, 4)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('d. Image')

    fig.add_subplot(2, 3, 5)
    plt.scatter(X, -Y, marker='s')
    plt.title('e. Points')

    fig.add_subplot(2, 3, 6)
    plt.plot(fy_left, -y, label='left curve', color='g')
    plt.plot(fy_right, -y, label='right curve', color='b')
    plt.title('Best fit line')
    plt.legend()

    plt.show()

"""
# Save binary image #
name = 'Result.png'
path = 'image_for_report'
cv2.imwrite(os.path.join(path, name), output)
"""

return output

# This function use to calculating radius of curvature #
def RadiusOfCurvature(y, p):
    y = warp_size[1]-1
    R = np.round(((1+(2*p[0]*y+p[1])**2)**2)**1.5)/np.absolute(2*p[0]), 2)
    return R

```

```

# This is function use to histogram equalization on single channel #
def HistogramBalance(img, show=False):
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(img_hsv)

    rows, cols = v.shape
    average = int(np.average(v))

    for row in range(rows):
        for col in range(cols):
            if v[row, col] > average:
                v[row, col] = average

    img_hsv = cv2.merge((h, s, v))
    img_bgr = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR)

    if show:
        cv2.imshow('Original', img)
        cv2.imshow('Equ', img_bgr)
        cv2.imshow('v channel', v)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return img_bgr

# Reference, linear regression with numpy #
# https://www.cs.toronto.edu/~frossard/post/linear_regression/ #

# This function is a example for a regression model, polynomial
# regression #
def PolynomialRegression(binary, plot=False):
    centerline = np.int(binary.shape[1]/2)
    print("Centerline position: ", centerline)

    # Dataset haft-left binary image #
    nonzero = binary[:, :np.int(binary.shape[1]/2)].nonzero()
    X = np.array(nonzero[1])
    Y = np.array(nonzero[0])

    print("Nonzero points: ", nonzero)
    print("X\n", X), print("Y\n", Y)

    p = np.polyfit(X, -Y, 2)
    print("Parameter of p(x): ", p)

    x = np.linspace(0, binary.shape[1]-1, binary.shape[1])
    y = p[0]*x**2 + p[1]*x + p[2]

    if plot:
        plt.subplot()
        plt.scatter(X, -Y, marker='s')
        plt.title('Dataset')
        plt.plot(x, y)
        plt.title('Regression Curvature')
        plt.show()

# The end #

```

13.3.2 Chương trình C trên Arduino.

```

// INDUSTRIAL UNIVERSITY OF HCM CITY          //
// INSTRUCTORS: TS. NGUYEN VIEN QUOC          //
// TEAM "Do an tot nghiep 2018"              //
// -----                                     //
// include the librarys //                    //
#include <Servo.h>
#include <Wire.h>
// Definations //
#define slave_addr 0x08
#define baudrate 9600
#define IN1 4
#define IN2 5
#define IN3 8
#define IN4 7
#define ENA 11
#define ENB 3
#define servoPin1 9
#define servoPin2 10
#define TRIG 13
#define ECHO 12
// Declerations //
Servo Servo1;
Servo Servo2;
// Offset actual from Raspberry Pi //
volatile byte y = 255;
//Offset Setpoint [pixel].
float r = 0;
//Input, Output u, Error e //
float e = 0, u = 0, u0 = 0, e0 = 0;
//Sampling Time, [ms]
float dt = 10;
//Parameter of PD controller.
float Kp = 10.0, Kd = 0.2;
//Velocity desire, v0 = [0;255]
unsigned char v0 = 150;
unsigned char delta_v = 0;
//Position of camera
unsigned int pitch = 100, yaw = 85;
// Timer value //
unsigned char tcnt2;
// Range to convert offset //
float x1 = 0.0;
float x2 = 252.0;
float y1 = -50;
float y2 = 50;
float m = (y2 - y1) / (x2 - x1);
// Velocity of sound in air //
float haft_sonar_velocity = 29.0 / 2; // [cm/s]
// Proportional with 100*dt = 1000ms = 1s #
float count = 100;
// Distance from car to obstruction //
float distance = 0;

// This function use to configure for L298 Driver //
void configIO()

```

```

{
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
}
// This function use to configure for Servo control pins //
void configServo()
{
    Servo1.attach(servoPin1);
    Servo2.attach(servoPin2);
}
// This function use to configure for I2C interface //
void configI2C()
{
    Wire.begin(slave_addr);
    Wire.onReceive(read_i2c);
}
// This function to setup //
void setup()
{
    configIO();
    configServo();
    configI2C();

    tcnt2 = TimerValue();
    SetTimer2(tcnt2);

    CameraPos(pitch, yaw);
    delay(200);

    Serial.begin(baudrate);
}
// Main Program //
void loop()
{
    // Sampling to calculate distance //
    if (count == 0)
    {
        distance = Distance_cm();
    }
    // Check distance, if distance <= 2 cm is not operance #
    if (distance <= 2)
    {
        if (y != 255)
        {
            read_i2c();
            y1, y2 = tracking(y, e, u);
            Run(y1, y2);
        }
        else
            Stop();
    }
    else
        Stop();
}

```

```

}

// Interrupt Program //
ISR(TIMER2_OVF_vect)
{
    TIFR2 = 0x00;
    TCNT2 = tcnt2;
    PDcontroller();
    count++;
}
// This function is PD controller program //
void PDcontroller()
{
    e = r - (float)y;
    u = (byte)(u0 + Kp * e + Kd * (e - e0) / (dt * 10e-3));
    u0 = u;
    e0 = e;
}
// This function use to setup Timer 2 to create Sampling for PD
controller //
void SetTimer2(unsigned char tcnt2)
{
    TCCR2A = 0;
    TCCR2B = 0;
    TIMSK2 = 0;
    TIMSK2 = 0x01;
    TCCR2B = 0x03;
    TIFR2 = 0x00;
    TCNT2 = tcnt2;
}
// Subprograms for control to Robot.
void Direction(String backward)
{
    if (backward == "false")
    {
        digitalWrite(IN1, 1);
        digitalWrite(IN2, 0);
        digitalWrite(IN3, 1);
        digitalWrite(IN4, 0);
    }
    if (backward == "true")
    {
        digitalWrite(IN1, 0);
        digitalWrite(IN2, 1);
        digitalWrite(IN3, 0);
        digitalWrite(IN4, 1);
    }
}
void Stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}
void Run(byte vL, byte vR)
{
    vL, vR = tracking(v0, e, u);
    Direction("false");
    analogWrite(ENA, vL);
    analogWrite(ENB, vR);
}

```

```

}

// This function use to setup camera position //
void CameraPos(unsigned int pitch, unsigned int yaw)
{
    Servo1.write(yaw); //pin 10
    Servo2.write(pitch); //pin 9
}
// This function use to receive Offset value from Pi via I2C interface //
void read_i2c()
{
    while (Wire.available())
        y = Wire.read();
    y = convertOffset((float)(y));
}
float convertOffset(float value)
{
    value = (m * (value - x1) + y1);
    return value;
}
// This function use to calculate Timer value for TCNT2 register //
unsigned char TimerValue()
{
    float temp;
    temp = round(256 - dt * 1000 * 16 / 1024); //max 16ms
    return (unsigned char)temp;
}
// This function is importance, it use to tracking setpoint offset value
//
byte tracking(byte v, float e, byte u)
{
    byte a, y1, y2, m1, m2, m3, m4, m5;
    byte x1 = 0, x2 = 0, x3 = 0;

    if (e > 0)
    {
        x1 = 1;
        x2 = 0;
        x3 = 0;
    }
    if (e < 0)
    {
        x1 = 0;
        x2 = 1;
        x3 = 0;
    }
    if (e == 0)
    {
        x1 = 0;
        x2 = 0;
        x3 = 1;
    }

    a = u + v;
    m1 = v * x1;
    m2 = a * x1;
    m3 = v * x2;
    m4 = a * x2;
    m5 = v * x3;
}

```

```
y1 = m2 + m3 + m5;
y2 = m1 + m4 + m5;

    return y1, y2;
}
float Distance_cm()
{
    unsigned long duration;

    digitalWrite(TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG, LOW);

    duration = pulseIn(ECHO, HIGH);

    distance_cm = (float)(duration / haft_sonar_velocity);
    return distance_cm;
}
// The end //
```

TÀI LIỆU THAM KHẢO

- [1] Ian T. Young, Jan J. Gerbrands và Lucas J. van Vliet (2007), Fundamentals of Image Processing. Delft University of Technology.
- [2] Alexander Mordvintsev & Abid K (2017), OpenCV-Python Tutorials Documentation.
- [3] Mark Summerfield, Programming in Python 3(2001). Addison-Wesley.

PHỤ LỤC**Phụ lục 1: Các tag dành cho hàm cv::cvtColor để chuyển đổi không gian màu**

cv.COLOR_BGRA2BGR
cv.COLOR_RGBA2RGB
cv.COLOR_BGR2RGB
cv.COLOR_RGB2BGR
cv.COLOR_BGR2GRAY
cv.COLOR_RGB2GRAY
cv.COLOR_GRAY2BGR
cv.COLOR_GRAY2RGB
cv.COLOR_BGR2HSV
cv.COLOR_RGB2HSV
cv.COLOR_BGR2HLS
cv.COLOR_RGB2HLS
cv.COLOR_HSV2BGR
cv.COLOR_HSV2RGB

Phụ lục 2: Các hàm OpenCV – Python dùng trong đồ án.

1	Chuyển đổi không gian màu: cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
2	Chọn những điểm ảnh trong conv nằm trong giới hạn (lower, upper): cv2.inRange(conv, lower, upper)
3	Phép OR hai mảng: cv2.bitwise_or(white_mask, yellow_mask)
4	Phép AND hai mảng và mask: cv2.bitwise_and(img, img, mask=mask)
5	Khử nhiễu: cv2.GaussianBlur(img, (kernel, kernel), 7)
6	Tính ma trận chuyển đổi: cv2.getPerspectiveTransform(dst, src)
7	Chuyển đổi ảnh đơn kênh sang ảnh nhị phân cv2.threshold(gray, thresh, 255, cv2.THRESH_BINARY)
8	Phép biến đổi hình thái làm co ảnh. cv2.erode(gray_img, kernel, iterations=1)
9	Tô một vùng ảnh với các điểm cho trước: cv2.fillPoly(color_warp, np.int_(points)), (0, 255, 0))
10	Hàm này hỗ trợ vẽ đường thẳng đi qua hai điểm: cv2.line(img, (np.int(draw*img_size[0]/warp_size[0])), 0), (np.int(draw*img_size[0]/warp_size[0])), 5), (0, 255, 0), 1)
11	Hoà trộn ảnh theo tỷ lệ: cv2.addWeighted(img, 1.0, unwarp, 0.3, 0.0)
12	Tách kênh màu: cv2.split(img_hsv)

13	Gộp kênh màu: cv2.merge((h, s, v))
14	Hiển thị ảnh: cv2.imshow('Original', img)
15	Chờ khi nhấn một phím bất kỳ sẽ thực hiện lệnh sau nó: cv2.waitKey(0)
16	Tắt tất cả windows đang mở từ OpenCV: cv2.destroyAllWindows()

--HẾT--