라즈베리파이를 사용한
자율주행 RC카

서 경 대 학 교
전 자 공 학 과
김 나 연

라즈베리파이를 사용한
자율주행 RC카

지도교수 조 근 호

이 논문을 졸업논문으로 제출함

2021년 12월 24일

서 경 대 학 교
전 자 공 학 과
김 나 연

김나연의 졸업논문을 인준함

심사위원 _____ 인

심사위원 _____ 인

심사위원 _____ 인

서경대학교

2021년 12월 24일

차 례

그 림 차 례

국 문 요 약

라즈베리파이를 사용한
자율주행 RC카

서 경 대 학 교
전 자 공 학 과
김 나 연

본 논문은 라즈베리파이를 기반으로 하여 자율주행을 할 수 있는 rc카에 관하여 연구하였다.

자율주행자동차는 운전자 또는 승객의 조작 없이 자동차 스스로 운행이 가능한 자동차를 말한다. 2010년대에는 딥러닝을 이용한 자율주행 기술 연구가 급진전되어 상용차에 제한적으로 탑재되고 있다. 2012년 발표된 IEEE의 보고서에 의하면, 2040년에는 전 세계 차량의 약 75%가 자율주행 자동차로 전환될 것으로 예상된다.

스스로 라인을 따라 주행할 수 있는 자율주행 RC카를 구현해보았다. 기울기를 이용한 자율주행 방법을 사용하였다. 라인의 기울기를 측정하기 위해서 openCV 기술을 이용하였다. RC카를 제어하기 위해 하드웨어로 라즈베리파이를 사용하였다. 주행을 하기위한 DC모터를 모터드라이버를 사용했다. RC카를 제어하기 위한 코드들은 Python을 이용하여 제작했다. 적당한 기울기 값에 방향전환을 하기위해 코드의 값을 바꿔서 코드를 시도해본다. 제작된 RC카는 직진주행 뿐만 아니라 기울기를 이용한 방향전환이 가능하다.

제 1 장 서 론

자율주행의 개념은 1960년대에 벤츠를 중심으로 제안되었고, 1970년대 중후반부터 초보적인 수준의 연구가 시작되었다. 2010년대에는 딥러닝을 이용한 자율주행 기술 연구가 급진전되어 상용차에 제한적으로 탑재되고 있다. 본격적으로 상용화가 된다면 전체 교통사고의 95%가량을 차지하는 운전자 부주의에 의한 교통사고와 보복운전을 줄일 수 있다고 기대된다. 또한, 인간 운전자를 완전히 대체하게 되면 교통정체의 감소를 가져오고 교통경찰과 자동차 보험이 필요 없어질 것이다.

2010년대 이후로 현재 벤츠, 아우디, 현대자동차, 포드, GM 등의 많은 자동차 기업뿐만 아니라 웨이모, 엔비디아 같은 IT 기업들이 개발 중인데, 그 중 웨이모의 자율주행차는 약 1000여 대의 차량으로 실제 도로 주행을 하여 주행거리가 3,200만km를 넘어섰다. 이에 반해 테슬라의 경우, 2020년 1월 자율주행 시스템으로 실도로 총주행거리는 48억km가 넘으며, 70여만 대의 자율주행 하드웨어 차량을 통해서 단, 하루에도 약 881만km의 데이터를 수집할 수 있어서, 데이터의 양이 얼마나 큰 지가 압도적인 차이를 만들어 내는 딥러닝 기반의 자율주행 개발에 있어서 유리한 위치에 있다. 여기에 애플도 장기 프로젝트 중 하나로 자율주행차 개발을 진행하고 있다.

가장 일반적으로 쓰이는 기술은 SLAM(Simultaneous localization and mapping)이라는 기법이다. 동시적 위치추정 및 지도 작성으로 번역되는데 로봇에 의한 지도 작성의 한 분야로서, 이름처럼 자신이 알지 못하는 주위 공간의 지도를 만들면서 동시에 자신이 그 지도상의 어디에 있는지를 추정하는 기법이다. 차량에 붙은 여러 개의 센서와 오프라인 지도를 결합해서 활용한다.

이렇게 꾸준히 발전되고 있는 자율주행을 직접 RC카로 구현했다. picam으로 실시간 영상을 받아 라인을 인식하고 주행을 하는 RC카를 제작한다.

제 2 장 개발 환경

2.1 부품 소개

RC카를 제작하기 위한 주요 부품들을 소개한다.

2.1.1 라즈베리파이

(그림1)

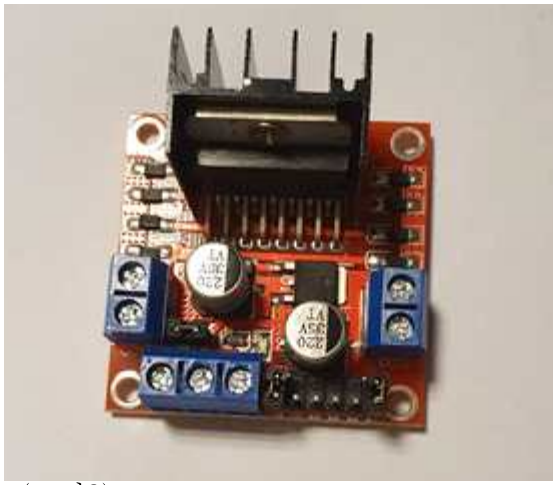
초소형 컴퓨터로 RC카 제작을 위한 하드웨어이다. 라즈베리파이로 명령을 설정하고 그 명령을 실행시키도록 한다.

2.1.2 DC모터

(그림2)

RC카에서 바퀴를 제어해줄 DC모터이다. 고정자로 영구자석을 사용하고, 회전자로 코일을 사용하여 구성한 것으로, 전기자에 흐르는 전류의 방향을 전환함으로써 자력의 반발, 흡인력으로 회전력을 생성시키는 모터이다.

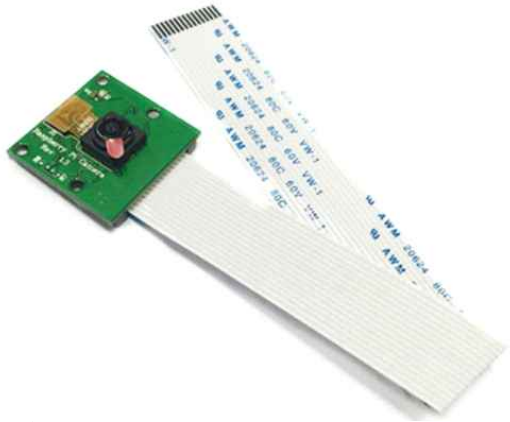
2.1.3 모터 드라이버



(그림3)

모터 드라이버로 DC모터를 제어해준다. DC모터를 라즈베리파이에 직접 연결하면 라즈베리파이의 단자가 파손될 수 있음으로 중간에서 중재 역할을 해준다.

2.1.4 Pi cam



(그림4)

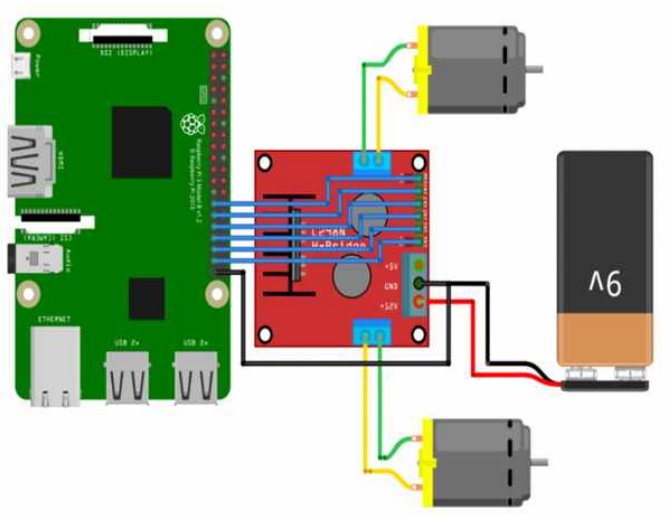
라즈베리파이에 연결할 수 있는 카메라 모듈이다.

2.2 개발 언어

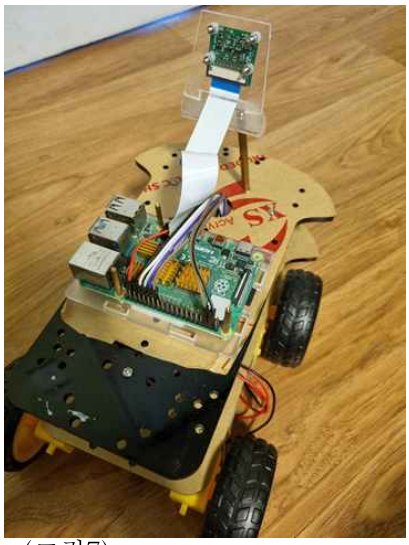
(그림5)

RC카를 제어하기 위해서 파이썬이라는 프로그래밍 언어를 사용한다. 파이썬은 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델을 가지고 있다.

2.3 RC카 설계



(그림 6)

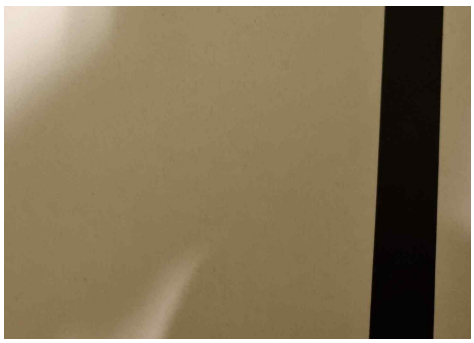


(그림7)

(그림6)은 RC카의 대략적인 설계도이며 (그림7) 실제로 제작된 RC카의 모습이다. 라즈베리파이와 DC모터 그리고 모터드라이버를 연결한 모습을 알 수 있다.

제 3 장 Open CV를 활용한 Pi cam 설정

3.1 라인 색상 추출



(그림8)

(그림9)

특정 색상 영역을 추출할 때 cv2.inRange 함수를 사용하는 것이 좋다.

```
lower_black = np.array([0, 0, 0])
upper_black = np.array([35, 35, 35])
white_mask = cv2.inRange(frame2, lower_black, upper_black)
```

이 함수를 사용하면 설정한 범위의 색상은 흰색으로 나머지는 검은색으로 바꿔준다. lower_black은 하한 값, upper_black은 상한 값으로 검은색의 범위를 설정해준다. 결과적으로 (그림8)이 (그림9)로 바뀐 것을 확인할 수 있다.

3.2 윤곽선 추출

```
img= cv2.Canny(img,200,400) #윤곽선 추출
```

이미지 상에서 밝기가 큰 폭으로 변하는 지점이 객체의 윤곽선이 된다. 그러므로 윤곽선은 픽셀의 밝기가 급격하게 변하는 부분으로 간주할 수 있다. 윤곽선을 찾기 위해 미분과 기울기 연산을 수행하며, 이미지 상에서 픽셀의 밝기 변화율이 높은 경계선을 찾는다. cv2.Canny 함수로 입력 이미지에 대한 윤곽선을 추출할 수 있다. 하위 임계값 200, 상위 임계값 400으로 설정해서 검출을 진행한다. 픽셀이 상위 임계값보다 큰 기울기를 가지면 픽셀을 윤곽선으로 간주하고, 하위 임계값보다 낮은 경우 윤곽선으로 고려하지 않는다.

3.3 ROI 설정 (관심 영역 설정)

```
img = img[160:240,0:320]
```

img[높이(행), 너비(열)]로 관심 영역을 설정한다. 이미지의 아래 부분만이 필요해 160부터 240까지로 슬라이싱을 해주었다. 기울기를 측정할 때 너무 멀리 있는 기울기까지 측정되면 RC카가 미리 움직일 수도 있어서 ROI 설정을 해주었다.

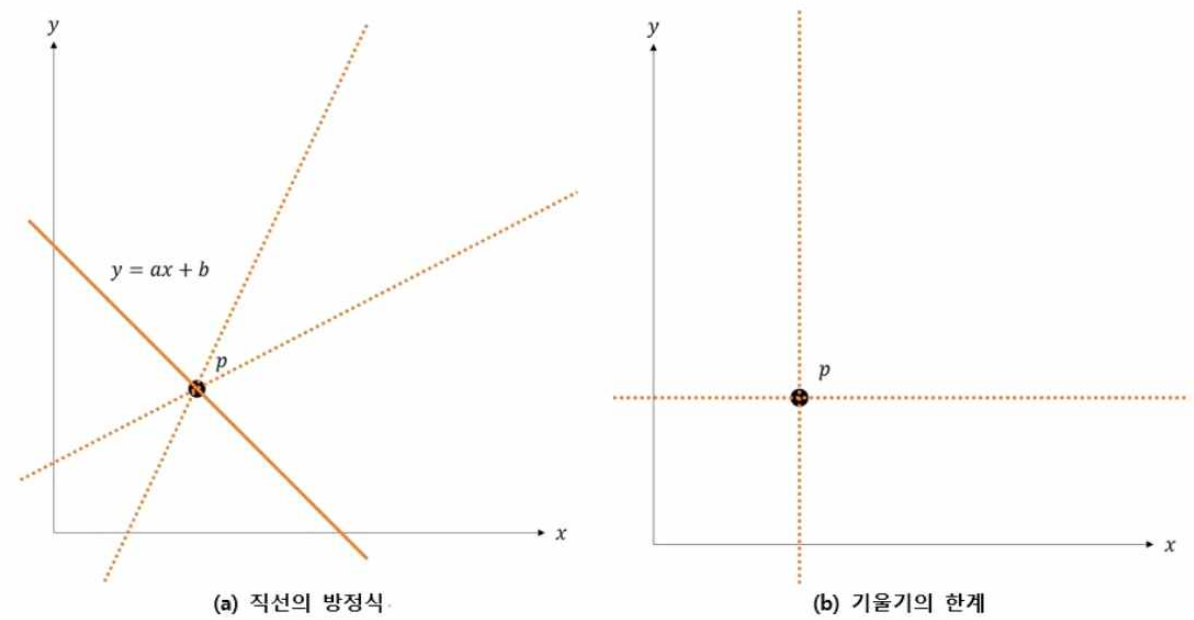
3.4 직선 검출

```
lines = cv2.HoughLinesP(img, 1, np.pi / 180, 20)

if(lines is not None):
for line in lines:
for x1, y1, x2, y2 in line:
cv2.line(frame2, (x1, y1+160), (x2, y2+160), (0, 0, 255), 2)
```

직선 검출 알고리즘은 허프 변환을 활용해 직선을 검출한다. 허프 변환은 이미지에서 직선을 찾는 가장 보편적인 알고리즘이다. 이미지에서 선과 같은 단순한 형태를 빠르게 검출할 수 있으며, 직선을 찾아 이미지나 영상을 보정하거나 복원한다. 허프 선 변환은 이미지 내의 어떤 점이라도 선 집합의 일부일 수 있다는 가정 하에 직선의 방정식을 이용해 직선을 검출한다.

직선 검출은 직선의 방정식을 활용해 y = ax + b를 극좌표(ρ, Θ)의 점으로 변환해서 사용한다. 극좌표 방정식으로 변환한다면 p = xsinΘ + ycosΘ이 되어, 직선과 원점의 거리(ρ)와 직선과 x축이 이루는 각도(Θ)를 구할 수 있다.

3.4.1 표준 허프 변환



(그림10)

표준 허프 변환은 입력 이미지(x, y 평면) 내의 점 p를 지나는 직선의 방정식을 구한다. 한 점을 통과하는 직선의 방정식을 구하면 기울기 a와 절편 b를 구할 수 있다. 점 p에 대해 직선의 방정식을 수식으로 표현하면 (그림10)의 그림 (a)와 같이 y = ax + b 로 표현할 수 있다. 모든 점에 대해 모든 직선의 방정식을 구한다면 평면상에서 점들의 궤적이 생성되며, 동일한 궤적 위의 점은 직선으로 볼 수 있다. 하지만, 한 점을 지나는 모든 직선의 방정식을 표현한다면 (그림10)의 (b)와 같이 기울기 a는 음의 무한대(-∞)에서 양의 무한대(∞)의 범위

를 갖는다.

또한 수평인 영역에서 기울기 a 는 0의 값을 갖는다. 기울기와 절편을 사용해 모든 직선의 방정식을 표현하는 것은 좋은 방식이 아니므로, 삼각함수를 활용해 각 선을 극좌표(ρ, Θ)의 점으로 변환해서 나타낸다.

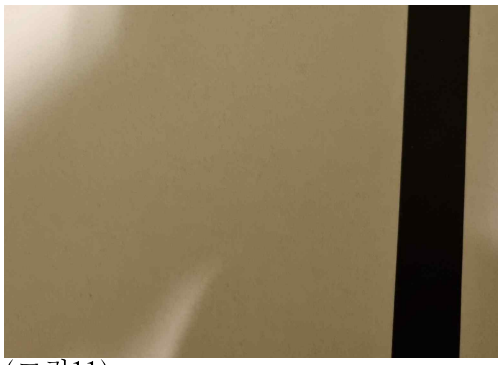
3.4.2 점진성 확률적 허프 변환

점진성 확률적 허프 변환은 또 다른 허프 변환 함수를 사용해 직선을 검출한다. 앞선 알고리즘은 모든 점에 대해 직선의 방정식을 세워 계산하기 때문에 비교적 많은 시간이 소모된다. 기본적으로 점진성 확률적 허프 변환 알고리즘은 앞선 알고리즘을 최적화한 방식이다. 모든 점을 대상으로 직선의 방정식을 세우는 것이 아닌, 임의의 점 일부만 누적해서 계산한다. 일부의 점만 사용하기 때문에 확률적이다. 그러므로, 정확도가 높은 입력 이미지에 대해 검출에 드는 시간이 대폭 줄어든다. 또한 이 알고리즘은 시작점과 끝점을 반환하므로 더 간편하게 활용할 수 있다.

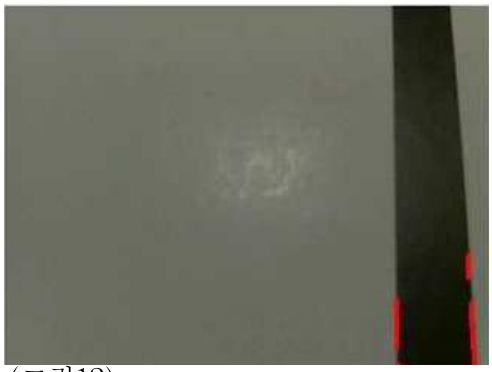
```
lines = cv2.HoughLinesP(img, 1, np.pi / 180, 20)
```

cv2.HoughLinesP(검출 이미지, 거리, 각도, 임계값, 최소 선 길이, 최대 선 간격)를 이용하여 직선 검출을 진행한다. 검출 이미지, 거리, 각도, 임계값은 앞선 허프 변환 알고리즘 함수와 동일한 의미를 갖는다. 최소 선 길이는 검출된 직선이 가져야 하는 최소한의 선 길이를 의미한다. 이 값보다 낮은 경우 직선으로 간주하지 않는다. 최대 선 간격은 검출된 직선들 사이의 최대 허용 간격을 의미한다. 이 값보다 간격이 좁은 경우 직선으로 간주하지 않는다. 위 코드를 보면 거리는 1, 각도는 np.pi/180 그리고 임계값을 20으로 설정한 것을 알 수 있다.

3.5 직선 그리기



(그림11)

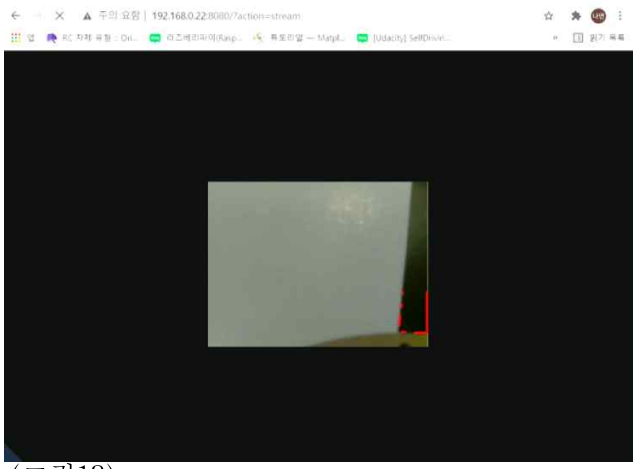


(그림12)

```
cv2.line(frame2, (x1, y1+160), (x2, y2+160), (0, 0, 255), 2)
```

직선 그리기 함수(cv2.line)로 입력 이미지에 직선을 그릴 수 있다. dst = cv2.line(src, pt1, pt2, color, thickness, lineType, shift)는 입력 이미지(src)에 시작 좌표(pt1)부터 도착 좌표(pt2)를 지나는 특정 색상(color)과 두께(thickness)의 직선을 그린다. 추가로 선형 타입(lineType), 비트 시프트(shift)를 설정할 수 있다. 설정된 입력값으로 그려진 직선이 포함된 출력 이미지(dst)을 생성한다. 위 코드를 보면 (0, 0, 255)는 빨간색 두께는 2로 설정해준 것을 알 수 있다. 결과적으로 (그림12)처럼 라인에 빨간 선이 나타난다.

3.6 mjpg 스트리밍



(그림13)

라인 인식을 하여 frame2에 직선을 그려주었는데 이제 이것을 외부에서 편하게 볼 수 있도록 설정해주었다. 외부에서 스트리밍을 볼 수 있도록 하는 것이 mjpg 스트리밍이다.

```
cv2.imwrite('/home/pi/project/pic.jpg', frame2)
```

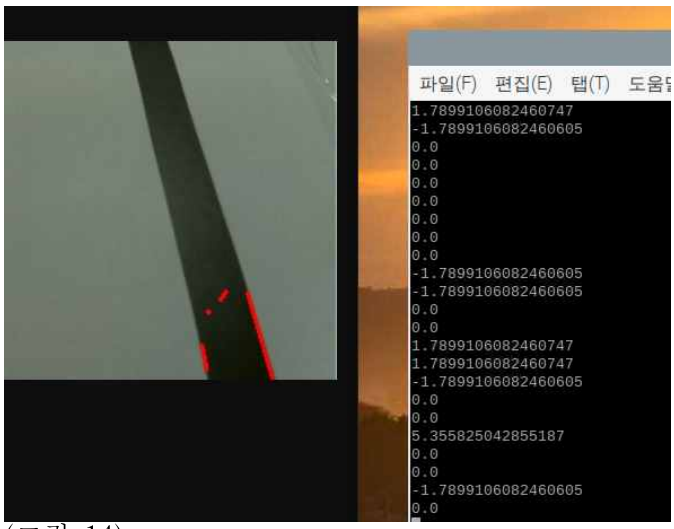
이때 frame2를 pic.jpg로 저장해준다. 그럼 계속해서 pic.jpg가 갱신이 되면서 (그림13)처럼 웹사이트에서 영상으로 나타나게 된다.

```
dx = int((x_end - x_start) * 0.23)
dy = int(max_y - min_y)
direction = math.atan2(dy, dx) / math.pi * 180 - 90
```

제 4 장 기울기를 이용한 모터 제어

4.1 기울기 측정

```
x_start = int(poly(max_y))
x_end = int(poly(min_y))
dx = int((x_end - x_start) * 0.23)
dy = int(max_y - min_y)
direction = math.atan2(dy, dx) / math.pi * 180 - 90
print(direction)
```



(그림 14)

라인을 추적하기 위해서 기울기를 구해 기울기 값에 따른 rc카의 방향을 판단한다. y 좌표로 x 좌표의 시작과 끝을 구한다음 x의 증가량과 y의 증가량을 구한다. 그리고 dx, dy 값을 이용해 수학적인 공식을 적용시켜 direction을 구한다. print(direction)을 통해 측정되고 있는 기울기 값들을 볼 수 있다.

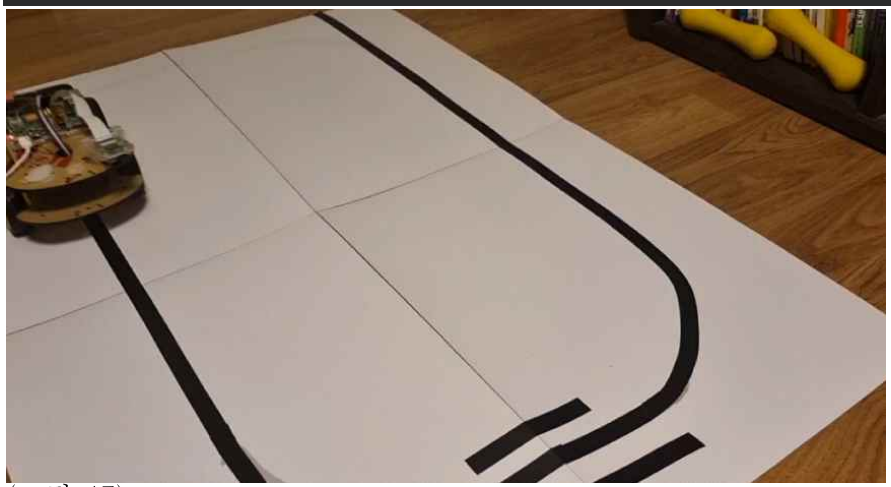
4.2 모터 제어

```
def back(speed):
leftPWM.ChangeDutyCycle(speed)
rightPWM.ChangeDutyCycle(0)
frontPWM.ChangeDutyCycle(speed)
backPWM.ChangeDutyCycle(0)


def go(speed):
leftPWM.ChangeDutyCycle(0)
rightPWM.ChangeDutyCycle(speed)
frontPWM.ChangeDutyCycle(0)
backPWM.ChangeDutyCycle(speed)


def stop():
leftPWM.ChangeDutyCycle(0)
rightPWM.ChangeDutyCycle(0)
frontPWM.ChangeDutyCycle(0)
backPWM.ChangeDutyCycle(0)


def right(speed):
leftPWM.ChangeDutyCycle(50)
rightPWM.ChangeDutyCycle(0)
frontPWM.ChangeDutyCycle(0)
backPWM.ChangeDutyCycle(speed)


def left(speed):

leftPWM.ChangeDutyCycle(0)
rightPWM.ChangeDutyCycle(speed)
frontPWM.ChangeDutyCycle(50)
backPWM.ChangeDutyCycle(0)
```

모터를 제어하기 위한 함수를 설정해준 코드이다. 진진, 후진, 정지, 우회전, 좌회전에 대해 각각 설정해주었다. 여기서 pwm.ChangeDutyCycle([duty cycle]) 함수를 사용하는데, 이 함수를 사용하면0% ~ 100% 사이의 [duty cycle] 값으로 바꿀 수 있다.

4.3 RC카 주행

```
count = 0
else:
count += 1
if count > 15:
stop()
else:
if(direction < -6):
right(sp)
elif(direction > 6):
left(sp)
else:
go(spgo)
```



(그림 15)

위에서 만든 모터 코드를 이용해서 RC카 주행을 한다. count값을 설정해주고 1씩 더해준다. count 값이 15보다 커지면 자동으로 RC카가 멈추게 된다. 다음으로 기울기 값을 이용한 코드이다. 기울기가 -6보다 작으면 우회전, 6보다 크면 좌회전을 하게 해주었고 나머지는 직진 주행을 하게 된다. 이 코드를 이용하여 주행을 실행시킨 모습이 (그림15)이다.

제 5 장 결 론

본 논문은 라즈베리파이를 기반으로 제작되었고 OpenCv를 사용하여 picam을 통해 자율주행 RC카를 구현하였다.

OpenCv를 사용하여 picam을 통해 보이는 화면을 파이썬으로 코드를 짜서 구현할 수 있었다. 차선인식을 위해서 회색조 변환을 하여 흑백으로 만들고, 윤곽선을 추출하여 사진을 단순화 시킨다. 단순화 시킨 영상 속에서 직선을 찾아 기울기를 구해주었다.

RC카를 주행시키기 위해 모터 드라이버를 이용하여 DC모터들을 제어해준다. DC모터의 방향을 설정해주기 위해서 파이썬으로 함수를 설정해주었고, 이 함수는 기울기를 이용한 주행을 구현해주었다.

마지막 과정에서 어느 정도의 기울기 값에 방향 전환을 해주어야 하는지가 어려웠다. 마찬가지로 카메라의 위치 조정과 기울기 값 여러 변수들이 있어서 모든 것을 고려하면서 조정하기가 힘들었다.

아쉬운 점은 DC모터가 방향 전환이 되는 모터가 아니라서 RC카 본체를 왼쪽 바퀴와 오른쪽 바퀴의 방향과 속도 차이를 두어 억지로 방향 전환을 하게 만들었는데 방향 전환이 되는 모터를 사용했으면 좋았을 것이라는 아쉬운 점이 있다. 또한 라인의 색상을 인식하여 주행하는 것이기 때문에 라인에 빛이 비춰져서 색이 달라지면 인식을 못할 때도 있는 것이 아쉬웠다.

향후 연구 방향으로는 움직임을 감지하여 피할 수 있게 하고 차선인식을 개선하여 많은 변수들에도 영향 받지 않는 자율주행을 계획한다. 또한 딥러닝을 통해 사물, 사람, 표지판 등을 학습시켜 전방에 무엇이 보이는지 인식할 수 있게 한다.