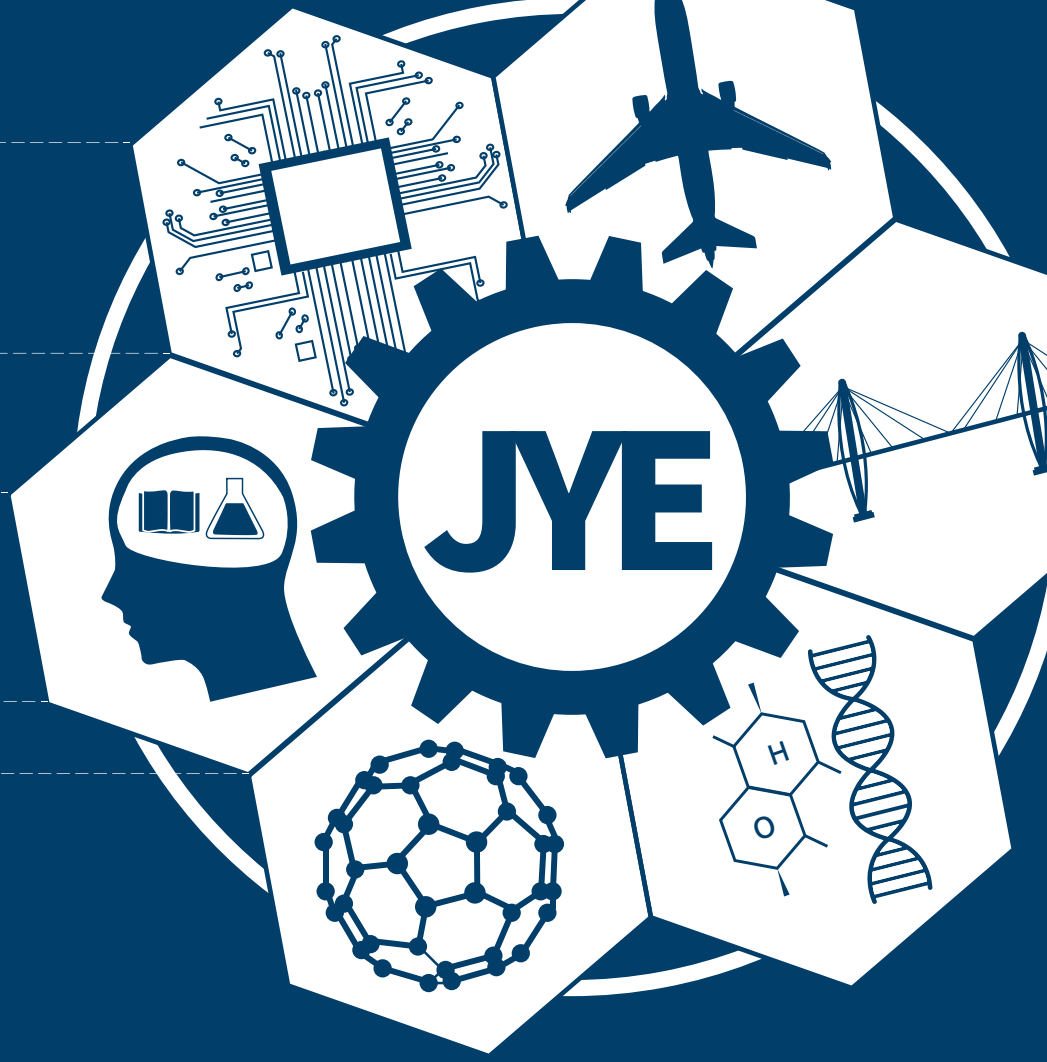


Journal of Youth Engineering

청년공학

제 5 집
—
2021년

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

NAEK 한국공학한림원

청년공학

Journal of Youth Engineering

제5집

2021. 08

NAEK 한국공학한림원
The National Academy of Engineering of Korea

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

차례

발간사

이제 교육 현장은 2022년 교육과정 개정을 앞두고 있습니다. 지난 2015년 개정 교육과정을 통해 융합 교육과 프로젝트 학습이 활성화되었고, 교내 활동 중심의 학생부종합전형이 자리 잡았습니다. 제4차 산업혁명의 물결을 타고 시작되는 2022년 개정 교육과정에서는 수월성을 추구하는 교과목(수학, 국제, 과학계열의 전문교과목, 과제 연구 등)과 프로젝트 활동이 더욱 확산할 것입니다.

이러한 사회적 흐름에 발맞추어 전국의 많은 고등학생이 의욕적으로 탐구를 하면서 탐구 보고서(논문)를 쓰고 있지만 대부분 교내 연구 발표 대회 자료집 발표 수준에 머물고 있습니다. 이는 교수와 전문가 중심의 학술지에서 고등학생들의 논문을 심사하거나 게재해주는 경우가 없기 때문입니다. 최근에는 대입 공정성 강화 정책에 따라 국내외 전문 학술지에 게재된 논문을 대입 실적으로 제출할 수 없게 되었습니다. 이 때문에 탁월한 연구 실적까지 사장되고 있는 안타까운 일이 벌어지고 있습니다.

한국공학한림원은 공학자(엔지니어)를 꿈꾸는 고등학생들이 공학 논문을 발표할 수 있는 학술지를 만들어 교육 현장의 연구 활동을 장려하고, 학술지에 투고하는 과정을 미리 체험할 수 있도록 2014년부터 <청년공학>을 펴내고 있었습니다. <청년공학>은 공정한 원칙과 절차에 따라 전문 학술지와 동일한 수준의 논문 심사 절차를 따르고 있습니다. 제3집부터는 한국공학한림원과 현대모비스가 주관하는 청소년 공학 리더 프로그램의 성과물을 논문으로 출판하고 있습니다. <청년공학>에 게재된 논문의 저자들은 서울대학교와 카이스트 등 대부분 상위권 공과대학에 진학하여 엔지니어의 꿈을 키우고 있습니다. 이번에 출판하는 <청년공학> 제5집 역시, 엔지니어를 꿈꾸는 고등학생들의 참신한 탐구 성과물을 담았습니다. 투고된 19편의 논문 중에서 5편은 게재 불가 판정을 받았으며, 최우수 논문 3편, 우수 논문 5편, 포스터 논문 6편을 선정하였습니다.

앞으로 <청년공학>은 발행 횟수를 점진적으로 늘려 일선 고등학교에 공학 연구 생태계를 조성하는 데 이바지하고자 합니다. <청년공학>이 국내 최초의 주니어 학술 저널로 발전할 수 있도록 일선 고등학교 현장 교사와 학생들의 많은 관심과 성원 부탁드립니다.

2021년 8월

한국공학한림원 회장

권 오 경

1

최우수 논문

1

Journal of Youth Engineering

01 주행 알고리즘 작성 및 Unreal Engine을 통한 시뮬레이션 구현

하나고등학교
박수경, 권능주, 김도현, 소현우, 심규민, 안민, 장민정, 정진용

주행 알고리즘 작성 및 Unreal Engine을 통한 시뮬레이션 구현

Coding Self-Driving Algorithms and Embodying Simulation through Unreal Engine

박수경, 권능주, 김도현, 소현우, 심규민, 안민, 장민정, 정진용*
Sukyoung Park*, Neungjoo Kwon, Dohyun Kim, Hyeonwoo So, Gyumin Shim, Min An, Minjeong Jang, Jinyong Jung

요약

본 연구는 자율 주행의 구동 원리를 차선 선택, 라이더 처리, 조향부 설정, 신호등 처리와 같은 기능으로 크게 나누어 분석하였으며, 이를 바탕으로 주행 알고리즘을 작성하였다. 곡선 트랙 주행 도중 차선을 인식하지 못하는 문제가 발생했을 때 트랙별로 발생한 횟수와 잘못 인식한 조향점의 위치를 분석하여 최적의 상수 값을 도출해내는 과정을 거쳤다. 더 나아가 Unreal Engine을 활용한 가상의 환경을 구현하고 직접적인 시뮬레이션을 진행하였다.

Keywords: 자율 주행 자동차, 주행 알고리즘, Unreal Engine

I. 서 론

본 연구팀은 소형 자율 주행 자동차 모델을 기반으로 하드웨어의 기본적인 구동 원리와 알고리즘 구성에 관한 연구를 진행하였다. 자율 주행 과정을 크게 차선 선택, 라이더 처리, 조향값 설정, 신호등 처리와 같은 핵심적인 기능들을 중심으로 구분하여 각각에 관한 분석 과정을 거쳤으며, 주행 알고리즘을 작성하여 여러 차례의 시뮬레이션을 진행하였다. 여러 시행착오를 거쳤으나 그중에서도 특히 곡선 트랙 주행 도중 차선을 제대로 인식하지 못하는 문제점에 집중하였고, Unreal Engine을 사용해 가상환경을 제작하여 문제 해결 방안을 도출해내고자 하였다.

II. 주행 알고리즘 분석

2.1. 신호등 처리

신호등을 인식할 때 자동차에 설치된 카메라에서 받아들이는 이미지에서 open CV에 내장된 HoughCircles 함수를 통해 원을 검출하고, 각각의 원을 HSV로 색 공간 변환을 하여 색상을 인식하게 된다. 이렇게 얻은 정보를 바탕으로 원이 인식되면, 지속해서 인식되는 프레임 수만큼 카운트한 후, 임계치를 넘어서면 자동차의 움직임을 바꾸는 방법을 사용하였다. 붉은 원이 일정 프레임 이상 검출되어 적색신호임을 인식하면 멈추고, 다시 녹색 원이 검출될 때 출발하는 방법을 택하였고, 이때 사용하는 검출 임계치를 달리하여 멈추는 거리를 조절하였다.

```
red = cv2.HoughCircles(r_range, cv2.HOUGH_GRADIE
green = cv2.HoughCircles(g_range, cv2.HOUGH_GRAD
```

그림 1. 원 검출 코드

```
def detect(self, img, aux_img):
    img = img[:self.cutoff_h]
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    r1_low = np.array([0, 100, 190])
    r1_high = np.array([30, 255, 255])
    r1 = cv2.inRange(hsv, r1_low, r1_high)
    r2_low = np.array([160, 100, 190])
    r2_high = np.array([179, 255, 255])
    r2 = cv2.inRange(hsv, r2_low, r2_high)
    r_range = cv2.bitwise_or(r1, r2)

    g_low = np.array([40, 100, 160])
    g_high = np.array([100, 255, 255])
    g_range = cv2.inRange(hsv, g_low, g_high)

    kernel = np.ones((5, 5), np.uint8)
    r_range = cv2.dilate(r_range, kernel)
    r_range = cv2.bitwise_and(v, v, mask=r_range)
    kernel = np.ones((3, 3), np.uint8)
    g_range = cv2.dilate(g_range, kernel)
    g_range = cv2.bitwise_and(v, v, mask=g_range)
```

그림 2. 색상 검출

* 박수경 (하나고등학교, psk8387@naver.com)
권능주 (하나고등학교, aurora_jjang@naver.com)
김도현 (하나고등학교, dohyun4316@naver.com)
소현우 (하나고등학교, shu7185@naver.com)
심규민 (하나고등학교, solargm04@naver.com)
안 민 (하나고등학교, anmin6749@naver.com)
장민정 (하나고등학교, minjrv0801@naver.com)
정진용 (하나고등학교, jyjeong04@naver.com)

```
# 신호등 처리
reds, greens = frontObject # reds : n*3의 크기
# reds: numpy array([[x1,y1,반지름], [x2,y2,반지름], ...])
if reds: # 빨간불이면
    self.vars.redCnt += 1
else:
    self.vars.redCnt = 0
if greens:
    self.vars.greenCnt += 1
else:
    self.vars.greenCnt = 0

if self.vars.redCnt >= 6:
    self.vars.greenCnt = 0
    self.vars.stop = True
    return 0,0
if self.vars.greenCnt >= 2:
    self.vars.redCnt = 0
    self.vars.stop = False

if self.vars.stop:
    return self.vars.steer, 0
```

그림 3. 신호등 처리

2.2. 라이다 처리

장애물 인식을 위한 라이다 처리에 있어서는 차체에 전면, 후면 라이다 총 2개의 라이다가 설치되어 있고, 각각의 라이다가 최대 2m 거리까지 측정할 수 있는데, 현재 차체 크기와 도로가 50cm X 50cm라는 점을 고려하여, 전후방 20cm 내에 물체가 있을 경우 정지하도록 하였다.

Ⅲ. 차선 선택

3.1 차선을 어디까지 볼 것인가

카메라를 통해 자동차가 인식하는 화면은 가로가 640px, 세로가 480px인 화면이다. 이 화면에서 보이는 차선을 픽셀 단위로 분석하여 자율 주행 자동차가 주행을 하게 된다.

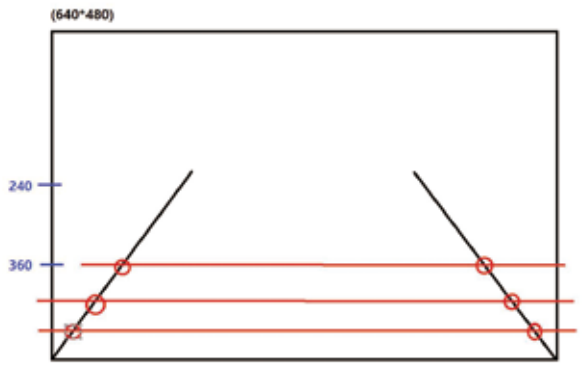


그림 4. 카메라를 통해 인식하는 화면

첫 번째로 차선을 어디까지 보아야 할지 범위를 설정하였다. 차선을 볼 수 있는 범위는 0부터 240까지이고 그 중간값은 240이다. 240까지 차선을 보겠다는 것은 480부터 240까지 차선을 보겠다는 것을 의미한다. 차선을 완전히 파악할 수 있도록 값을 0으로 하고 실행한 경우 차선이 없는 곳까지도 차선 분석을 시도하여 매우 비효율적이었다. 그래서 중간값인 240 값으로 자율 주행 자동차를 주행해본 결과, 카메라를 통해 보았을 때 굳이 240까지 하지 않아도 차선을 인식하는 데 있어 문제가 없었다. 그래서 그 값을 400으로 설정하고 주행해본 결과, 범위가 너무 작아 다음 차선을 보고 차가 대응하는 데 어려움이 있었다. 그래서 240과 480의 중간 지점인 360지점을 차선 인식 범위 값으로 설정하였다.

3.2 추세선 그리기

자율 주행 자동차가 카메라를 통해 받아오는 이미지를 분석하여 추세선을 그리기 위해서는 차선에 따라 점을 찍어야 한다. 따라서 차는 화면에 x축과 평행한 가상의 선을 긋고 차선과 그 가상의 선이 만나는 교점에 점을 찍는다. 본 연구팀에서 사용한 코드에서는 이 점을 양쪽에 각각 7개씩 찍도록 코딩되어 있다. 그리고 그 점을 지나는 가장 근사한 선을 긋고 그 선의 적절한 y 값을 찾아 점을 찍어 나타낸 것이 파란색 추세선이다.

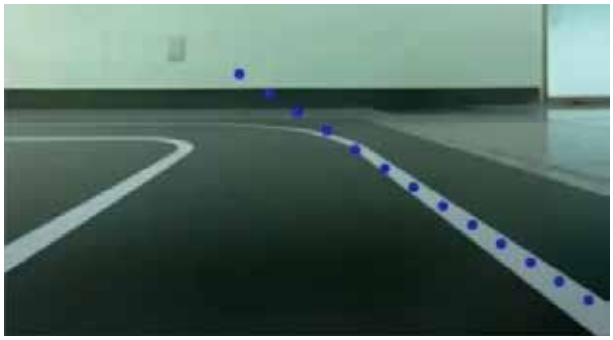


그림 5. 추세선

그림과 같이 추세선을 하나만 그리는 이유는 주선을 찾아서 그 주선의 추세선만 나타내기 때문이다. 여기서 주선은 가장 밑에서부터 먼저 보이는 선을 의미한다.

3.3 차가 중앙에 위치하도록 하기

차가 중앙에 위치하여 주행하게 하려면 x축에 평행한 가상의 선을 그어 추세선과 실제 차선이 얼마나 어긋나 있는지 파악해야 한다. 이때 이 가상의 선을 y축 어느 좌표에서 그을 것인지가 우리가 두 번째로 설정해야 하는 값이다. 이 값을 360으로 잡았을 때의 화면을 그림으로 나타내었다.

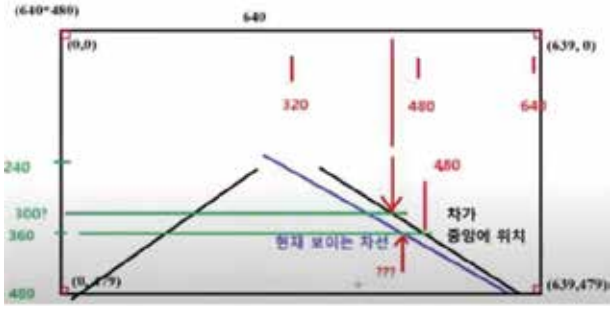


그림 6. 가상의 선이 y=360에 있을 때

가상의 선과 추세선이 만나는 점의 x 좌표의 값이 가상의 선과 실제 차선과 만나는 점의 좌표의 x값보다 작으면 차가 오른쪽으로 기울어 있다는 뜻이다. 따라서 차는 왼쪽으로 가야 한다. 예시로 들었던 360이라는 값이 너무 멀면 차가 너무 빠르게 판단하고 너무 가까우면 차가 너무 느리게 판단하게 된다. 이 값은 차의 속도에 의해서도 판단하는 속도가 달라질 수 있어 속도와도 관련이 있는 값이다.

우리는 차의 속도를 70으로 설정하고 주행 결과 360이라는 값이 가장 적당하다고 판단하였다. 이 값을 대입한 line(360)은 360에서 x축에 평행한 선을 그었을 때 현재 차선과 만나는 x 좌표의 값이고 이 값이 center_x값보다 크면 오른쪽 차선이고 작으면 왼쪽 차선이라는 것을 판단할 수 있다. 그리고 e의 값을 구하기 위해 차를 중앙에 놓고 중앙 위치 차선의 x 좌표 값을 출력하였더니 130이 나왔다. 결론적으로 e= line(360) - center_x -130이라는 식을 도출할 수 있었다.

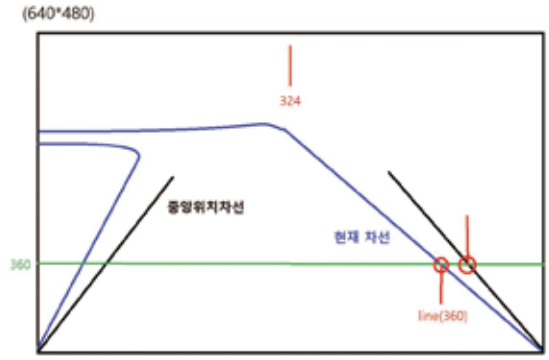


그림 7. e = line(360) - center_x - 130

3.4 Steer 값 조정하기

실제로 차가 중앙에 위치하기 위해 크게 좌회전을 해야 한다고 가정했을 때, 픽셀 차이는 300 이상이 나오게 된다. 하지만 조향은 자율 주행 자동차의 아커만 구조를 이루는 랙 앤 피니언과 양옆에 고정된 나사들로 인해 최소회전반경 270mm가 한계이다. 즉, 픽셀 차이가 조향 차이보다 훨씬 크게 설정이 되어야 한다. 따라서 steer 값의 e의 계수가 1이 넘지 않도록 0.5로 설정하였고 주행하면서 상수를 여러 번 바꾸며 시도한 결과 상수의 최적값은 -15임을 구하였다.

3.5 no line found 개선방안

기존의 코드와는 다르게, 대회 당시 사용했던 알고리즘에서는 조향 부분에서 크게 두 가지 차이점이 존재한다. 수많은 시뮬레이션과 실험을 통해, 자율 주행 자동차가 여러 가지 곡선 구간에서 필연적으로 no line found가 뜬다는 사실을 파악하였으며, 대회의 코스를 활용하여 최종적으로 시뮬레이션을 돌려보며 자율 주행 자동차가 no line found가 뜬 시점으로부터 몇 초간 상수 steer 값을 유지하면 안정적인 주행이 가능한지 테스트를 해보았다. No line found가 뜬 시점이 곡선 구간이 아닌 직선 구간에서도 일어나 갑작스러운 라인 아웃이 발생한 예도 있었는데, 이를 보완하기 위해 no line found가 몇 회 이상 일어나는지 곡선, 직선, S자 각각 실험을 통해 적절한 값을 찾아내었으며, 이 값을 대입하여 no line found가 특정 횟수 이상 일어나면 상수 steer를 유지하도록 조정했다. 다음 문제점은 no line found가 발생하면 그것이 좌회전 시 차선 이탈인지 우회전 때 차선 이탈인지 판별하여 두 가지 경우에서 각각 다른 상수 steer 값을 부여해야 한다는 점인데, 이를 판별하는 방법은 생각보다 간단하였다. No line found가 일어나기 직전에 직진보다 우측 조향점을 잡았는지, 좌측 조향점을 잡았는지 판단하여, 직진 steer 값보다 크다면 우측 steer, 상수가 작다면 좌측 steer 상수를 부여하여 자율 주행 자동차가 주행하도록 하였다.

Ⅳ. Unreal Engine 환경 구현

4.1. 개발 동기

앞서 자율 주행 알고리즘을 개발하는 과정에서 발생한 주요 문제점으로 차선을 올바르게 인식하지 못하는 문제를 해결하기 위하여 속도, 카메라 위치, 화각 등의 요인들을 변화시켜 실험을 진행하고자 하였다. 그러나 차량을 이용하여 시뮬레이션을 여러 번 반복하기에는 시공간적 제약이 있었으며, 현재 보유하고 있는 자율 주행 자동차를 개조하는 것은 현실적으로 어려울 것으로 판단하였다. 이러한 제약을 고려하여 Unreal 게임엔진을 활용한 시뮬레이션을 진행하기로 실험 계획을 수립하였다.

4.2. 사전 작업

본격적인 제작에 들어가기에 앞서 먼저 주행 코스의 타일들을 3D로 구현하기 위해 실제 타일의 치수를 측정하고, 이를 포토샵으로 구현하여 이미지로 변환한 뒤, 가상환경에 삽입하는 과정을 거쳤다. 연구에서 이용했던 코스의 검은색 타일은 1개당 가로와 세로가 각각 50X50(cmXcm)인 정사각형 모양이고, 가로의 왼쪽에서부터, 세로의 하단에서부터 각각 12cm, 36cm가 되는 지점에 폭이 2cm로 일정한 흰색 라인을 그려 제작하였다.

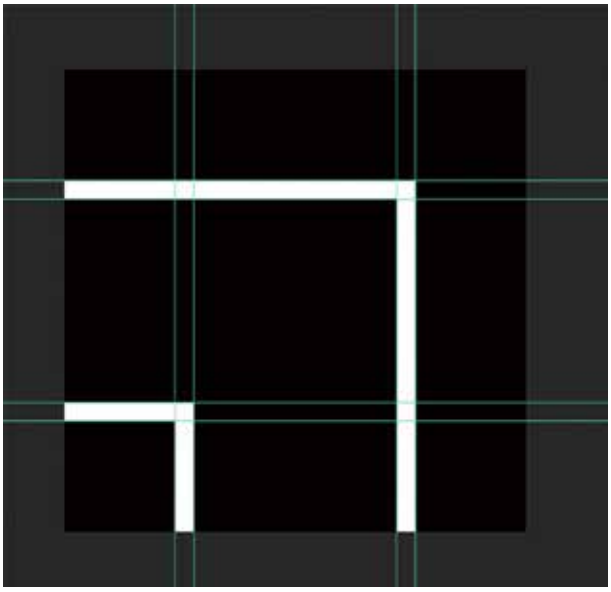


그림 8. 코스 3D 구현

Ⅴ. 제작 과정

다음 작업으로 해당 시뮬레이션을 Unreal Engine에서 구현하기 위해 다양한 방법을 시도하였고, 그중 가장 유의미한 결과가 도출되었던 방법을 토대로 Unreal Engine 환경을 구축하였다.

5.1 첫 번째 시도

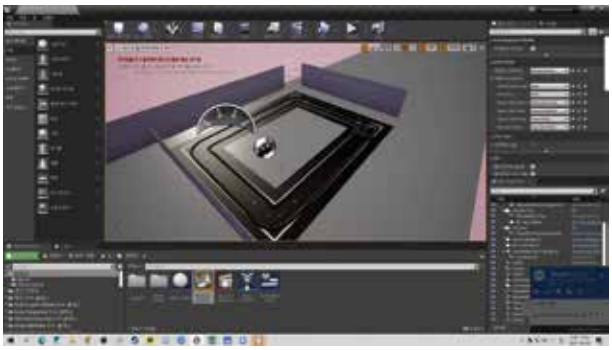
가장 이상적이었던 방법으로, 가상환경 내에서 실제의 것과 같은 차량을 제작하고 각 조향 부, 엔진, 바퀴 등이 내부적으로 연결된 알고리즘을 통해 실제 구동 과정에서 사용하는 코드를 추가하였을 때 바로 작동할 수 있는 상황을 가정하였다. 그러나 이를 완벽하게 구현하려면 실제 자동차 내부에서 바퀴가 정상적으로 굴러가는 데 필요한 부품들을 제작할 필요가 있었으며, 따로 작성해둔 Python 알고리즘을 가져온다고 해도 내부 라이브러리와의 상호작용이 완벽하게 일어날 것이라는 보장이 없었기에 차선책을 선택하기로 하였다.

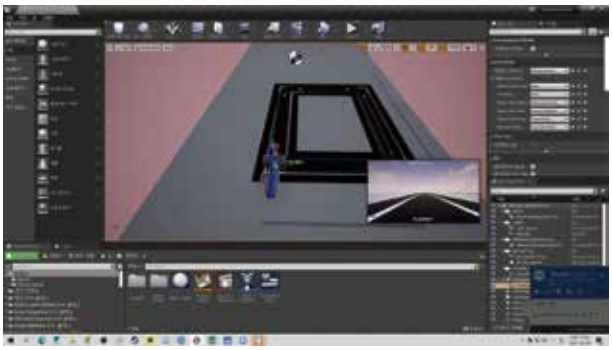
5.2 두 번째 시도

Unreal Engine의 맵 속에서 어떤 한 직육면체에 개체를 넣고 그 직경을 조절한 것을 자율 주행 자동차라고 했을 때, Unreal Engine 기본 모듈 중 하나인 내비게이션 모듈을 개체에 적용해 해당 개체가 맵에 있는 구조물과 상호적으로 영향을 주고받을 수 있도록 하였다. 이것을 활용해 코스 외에 나머지 공간에는 투명한 벽을 세우고 출발점과 도착점을 x, y, z 좌표로 지정하였다. 직교 좌표계 상에서 움직임이 가능한 경로에 대하여 최단 거리로 주행하도록 경로를 설정하는 방법이었으며, 추가로 내비게이션 모듈 설정 창에서 개체의 이동 속도를 조절하는 기능을 사용하여 시뮬레이션을 진행하려는 시도였다. 그러나 구동을 하는 과정에서 차가 코스의 정중앙에 오지 않아 다른 방법을 사용하게 되었다.

5.3 세 번째 시도

결과적으로 실제 연구에서 사용한 방법은 영화 촬영 시 사용하는 카메라 릭 기법을 활용한 구성이다. Unreal Engine 상에서 만들어 둔 코스 위에 레일을 설치하고 이 위를 카메라가 지나가는 방법을 사용하여, 문제점으로 제기되었던 레일 내 중앙에 맞춰 전진하지 못하는 문제를 해결하였다. 코스를 주행하면서 받아오는 이미지는 실제 자율 주행 자동차에서 주행 시 가져오는 이미지처럼 130도의 화각을 가지도록 하였고, 필요하다면 추후에 조정할 수 있도록 설정하였다. 이렇게 받아온 이미지를 바탕으로 자율 주행 자동차 컨트롤러 내에서 이미지를 돌려보았으며, 실제 Python 코드상에서 차선을 어떻게 인식하고 어느 방향으로 이동하려고 하며, 어떤 선을 잡는지 확인할 수 있었다. 실제 연구에 사용한 자율 주행 자동차의 카메라에서 얻어온 이미지의 경우 광각카메라로 인해 왜곡된 이미지이기 때문에 컨트롤러 내에서 왜곡 보정을 하는데, 가상환경에서 구현한 자율 주행 자동차에서 출력한 이미지의 경우 왜곡이 보정되어 나오기 때문에 이중 보정이 일어나 오히려 왜곡이 발생하였다. 이를 해결하기 위해 자율 주행 자동차 컨트롤러의 왜곡 보정 코드를 주석처리 하여 완전한 이미지가 보이도록 하였다.





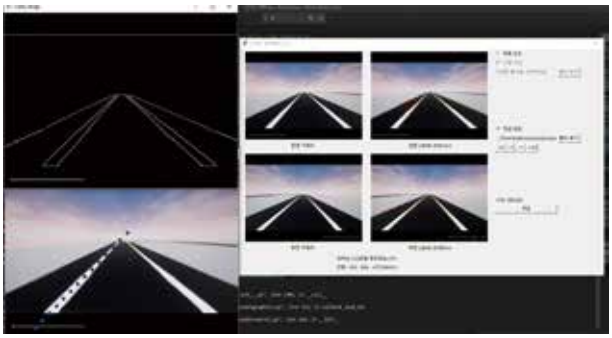


그림 9, 10, 11. Unreal Engine 구현

VI. 결론 및 제언

6.1. 결론

본 연구에서는 신호등 처리, 라이더 처리, 차선 선택, 그리고 Steer 값 설정 부분으로 나누어 분석하고 이에 해당하는 코드 값들을 계산해 주행 알고리즘을 작성하였다. 또한 실행하는 동안 곡선 주행 과정에서 필연적으로 No Line Found가 뜨는 문제를 확인하였고, 이를 해결하기 위해 No Line Found가 곡선, 직선, s자 코스에서 몇 번 발생하는지 실험을 통해 적절한 값을 찾아내었으며, 이 값을 대입하여 No Line Found가 특정 횟수 이상 발생했을 때 상수 steer 값을 유지할 수 있도록 하였다. 또한 탈선 방향을 고려해 No Line Found가 뜨기 직전에 어느 조향 점을 잡았는지에 따라 다른 상수 steer 값을 부여하였다. 그리고 No Line Found가 뜨지 않는 것을 Unreal Engine 환경을 통해 실험하였다.

6.2. 제언

6.2.1 Unreal Engine에서의 한계 1

자율 주행이라 하면, 자동차가 도로를 주행하며 스스로 차선의 위치를 확인하고 중앙을 맞추어야 하는데, 연구에서 사용한 방법은 반드시 중앙을 지날 수 있도록 레일을 고정하여 속도 및 화각, 카메라의 위치 등만 조절할 수 있도록 한 것이기 때문에, 자율 주행 자동차 컨트롤러에서 재생하였을 때 중앙을 맞추고, 방향을 트는 것에서 중요한 역할을 하는 center_x의 값을 확인하는 데 어려움을 겪었다.

그림 12. Unreal Engine 구현 문제점1

6.2.2 Unreal Engine에서의 한계 2

실제 세계에서 자율 주행 자동차를 주행하였을 때 저장하는 이미지에는 저장 당시의 시간, 전/후면 라이다, 속도 등이 이미지의 이름으로써 명시되어 이미지를 불러올 때 컨트롤러에 반영되는 것이다. 반면, 가상환경인 언리얼 엔진에서 자율 주행 자동차를 움직이고 이미지를 받아올 때는 속도와 라이다를 임의로 설정할 수 있으며 실제 주행 경우와 다르게 통신 지연의 문제점이 존재하지 않아 실제보다 더 많은 양의 정보를 받을 수 있어 정확도가 높다. 이렇게 Unreal Engine에서의 실험을 통해 제작한 코드는 실험을 통해 라이다를 사용할 수 없고 통신 지연이라는 변수를 고려하지 못하기 때문에 실전에 바로 적용하기 어렵다는 한계가 존재한다.

그림 13. 임의로 설정한 속도와 전·후면 라이다값

참고문헌

1) 김평원, 정형식, 홍범진. (2019). 함께 만드는 인공지능 자주차. 인천: 인천대학교출판부.

02 안정적인 곡선 주행을 위한 차선 인식 알고리즘 개선 방안 고찰

동산고등학교
김태호, 김용진, 박민제, 박영준, 이기훈, 임수홍

안정적인 곡선 주행을 위한 차선 인식 알고리즘 개선 방안 고찰

A Study on the Improvement of Lane Recognition Algorithm for Stable Curve Driving

김 태 호, 김 용 진, 박 민 제, 박 영 준, 이 기 훈, 임 수 홍*
Tae-ho Kim, Yong-Jin Kim, Min-jea Park, Young-jun Park, Gi-hun Lee, Su-hong Lim

요약

기존의 자율 주행 자동차 알고리즘에서는 차선의 곡률이 심한 S자 코스나, U턴 구간에서 차선 인식 능력이 떨어지는 경향을 보였다. 본 연구에서 기존 차선 인식 알고리즘의 개선안을 제안하여, 곡선 커브 구간에서의 주행 능력을 향상하고자 한다. 새로운 알고리즘은 기존 알고리즘에서 부족했던 점이나 단점들을 보완하고 개선해나가는 방향으로 제시를 하였다. 여러 가지 개선안들의 결과를 도출하여, 그 결과들을 기존의 알고리즘과 비교하여 향상된 차선 인식 알고리즘에 따른 곡선 주행의 개선 여부를 알아내고자 하는 것이 이 연구의 중심 소재이다. 차선 인식 알고리즘을 제외한 나머지 요소들은 동일화시켜 결과에 최대한 영향을 주지 않도록 하였고, 연구 과정은 자율 주행 알고리즘을 통해 시뮬레이션이 진행되었다, 개선된 알고리즘을 통해 더욱 안정적인 곡선 주행이 가능했다.

Keywords: 자율 주행 자동차, 차선의 곡률, 차선 인식, 곡선 주행

I. 서 론

자율 주행 자동차는 자신의 위치와 도로 위의 많은 요소를 예측하고 계산함으로써, 운전자 없이, 스스로 주행하는 자동차이다. 중심 요소에서 차선 인식은 가장 기본적이면서, 중요한 능력이다. 기존 자율 주행 자동차 모델은 원본 카메라에서 관심 영역(Region of Interest, ROI)을 설정하여, 해당하는 영역의 차선을 인식하여 검출한 뒤, 그레이 변환 후, 명도의 급격한 변화량을 분석하여, 캐니 엣지 이미지(Canny Edge Image)로 변환하여 차선을 인식한다. 인식한 차선을 토대로 자율 주행 자동차가 스스로 주행한다.

기존의 자율 주행 자동차 모델이 차선을 인식하여 곡선 주행을 할 때, 불안정한 모습과 하드웨어의 영향을 크게 받는 모습을 관찰해냈다. 본 연구에서는 기존의 차선 인식 과정에서 곡선 구간 주행 시, 생기는 문제점을 개선·보완하여, 세 가지의 새로운 알고리즘들을 제안하였다. 각자 다른 방식으로 개선된 차선 인식 알고리즘들은 기존 알고리즘과의 곡선 주행의 안정성을 분석함으로써, 기존보다 더 안정적이고, 이상적인 곡선 주행을 가능하게 한다. 시뮬레이션 된 결과의 오차를 줄이고, 적정 값을 찾기 위해 충분히 반복과정을 거쳐 알고리즘을 완성하게 하였다.

II. 본론

2.1. 속도와 조향의 관계 이용

속도 값을 조향 값에 대한 2차 함수로 표현하여, 직선 주로에서는 최고 속도로 주행하되, 차선의 굴곡이 심해질수록 속도가 점점 느려지도록 알고리즘을 구현했다. 우리의 자율 주행 자동차 모델에 적절한 최저·최고 속도를 각각 45, 90으로 주행하도록 하였는데, 최저·최고 속도 값은 차량 모델에 따라 달라질 수 있어 다른 차량 모델에 알고리즘을 적용할 때 유의해야 한다. 최저 속도에 최초로 도달하는 조향 값의 크기에 따라 [그림 1]의 기울기가 달라지는데, 일반적으로 곡선 코스를 주행하는 동안 출력되는 조향 값의 범위는 ±40º ~ ±60º이었다. 그래서 이 범위를 최초로 도달하는 조향 값의 범위로 정하여 시뮬레이션을 진행하였다. 시뮬레이션을 진행한 주행 코스는 [그림 2]와 같다.

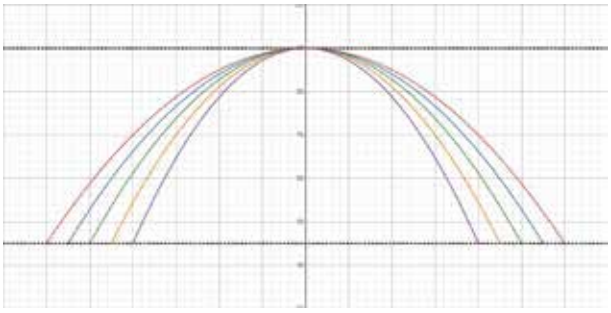


그림 1. 속도 값 정의

* 김태호 (동산고등학교, taeho0319@naver.com)
김용진 (동산고등학교, parkjunghee658@gmail.com)
박민제 (동산고등학교, alswp6676@naver.com)
박영준 (동산고등학교, junizzang123@naver.com)
이기훈 (동산고등학교, lkh040910@naver.com)
임수홍 (동산고등학교, dlatnghd0406@naver.com)

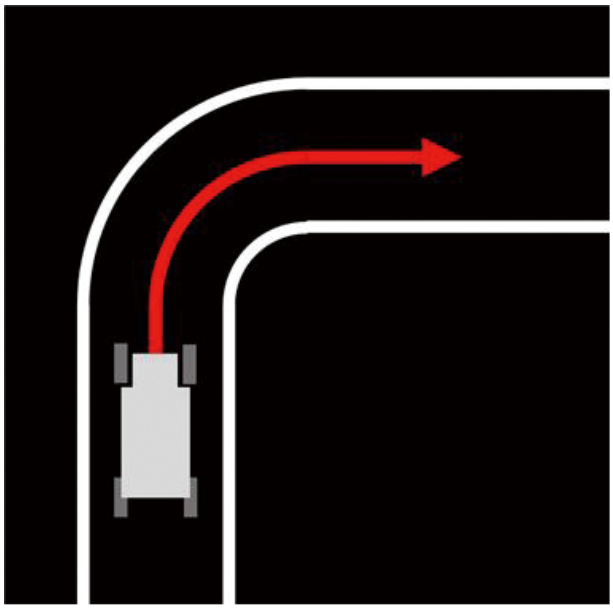


그림 2. 시뮬레이션 코스

[그림 2]의 트랙으로 시뮬레이션을 돌린 결과, 곡률 반경에 대한 속도 값을 이용하여 그래프[그림 3]와 같이 나타냈다. 범례는 최저 속도에 최초로 도달하는 조향 값을 의미한다. 그래프의 결과를 보면, 곡선 주행이 끝난 뒤에 모든 경우가 정상적으로 최고 속도에 가까워졌지만, 곡선 트랙을 진입할 때, ±40º ~ ±60º 범위에서 감속한 경우는 ±40º, ±45º 일 때, 두 가지 경우였다. 그중에서도 ±45º의 경우가 ±40º의와 비교해서 속도 값의 변화가 완만했기 때문에 비교적 안정적으로 곡선 주행을 했다. 따라서 시뮬레이션의 결과와 그래프를 분석해보았을 때, 최저 속도에 최초로 도달하는 조향 값을 ±45º로 했을 때, 가장 안정적인 곡선 주행을 했다.

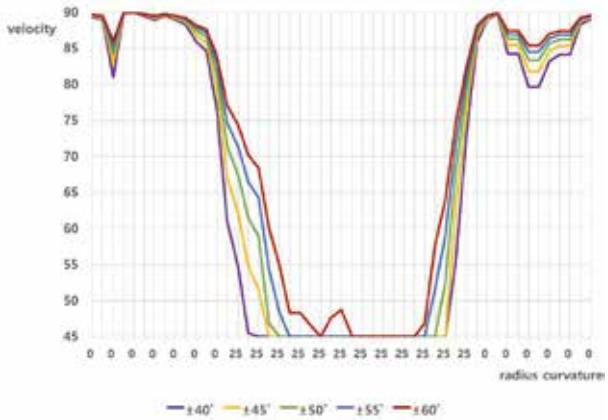


그림 3. 곡선 주행 거리에 따른 속도 값 변화 그래프

최저 속도 값에 도달하는 조향 값의 크기에 따른 곡선 주행에서의 속도 값 변화([그림 2])를 분석하여, 기울기의 변화가 곡선 주행에 알맞은 경우를 반영한 코드는 [그림 4]와 같다.

```
velocity = -45/(45**2) * (steer**2) + 90
if velocity < 45:
    velocity = 45
```

그림 4. 결과를 반영한 코드

2.2. 관심 영역의 재구성

기존 알고리즘에서의 추세선을 설정하는 관심 영역은 고정되어있었다. 그런데 [그림 5]와 같이 곡선 주행에서 관심 영역에 대한 추세선을 인식할 때, 실제 차선이랑 오차가 생긴다. 또한 [그림 6]과 같이 곡선 주행 후, 곡선 차선과 직선 차선이 만나는 지점을 추세선으로 인식할 때, 추세선의 위치가 급격하게 변화하여 뒷바퀴가 차선에 걸려버리거나, 조향 값이 갑자기 증가하여 차선을 이탈할 수 있었다.

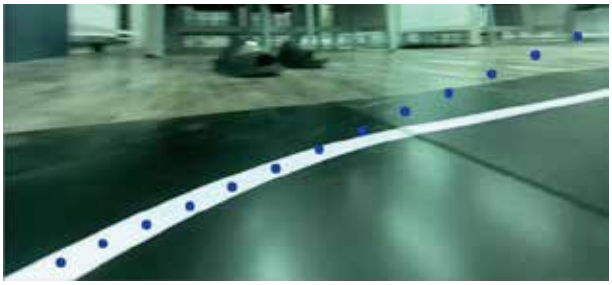


그림 5. 추세선이 급격하게 변화하기 전

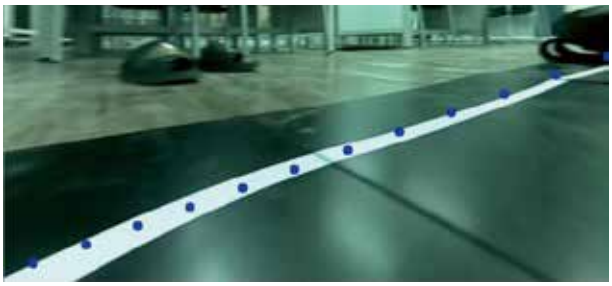


그림 6. 추세선이 급격하게 변화한 후

즉 차량이 곡선 구간을 이동할 때, 실제 차선에 대한 오차 범위를 따라서 주행한다는 것이다. 이를 방지하기 위해서 직선 주로일 때와 곡선 주로일 때, 추세선을 잡아주는 관심 영역을 다르게 재구성해주어, 곡선 주행 시, 생기는 오차를 최소화하여, 추세선의 변화를 부드럽게 바꿔주는 알고리즘을 고안해보았다.

```
for i in range(len(frontLines)):
    if self.vars.steer < -45 or self.vars.steer > 45: # 곡선 주행 시 관심영역 재조정
        print('관심영역 재조정')
        if frontLines[i][0, 1] < 200 or frontLines[i][0, 1] > 420:
            continue
        x = frontLines[i][:, 0]
        y = frontLines[i][:, 1]
        coefficient = np.polyfit(y, x, 1)
        line = np.poly1d(coefficient)
        if line(400) < center_x:
            line = 'left', i
            break
        else:
            line = 'right', i
            break
    else:
        if frontLines[i][0, 1] < 360: # 기본의 관심영역 적용
            continue
        x = frontLines[i][:, 0]
        y = frontLines[i][:, 1]
        coefficient = np.polyfit(y, x, 1)
        line = np.poly1d(coefficient)
        if line(400) < center_x:
            line = 'left', i
            break
        else:
            line = 'right', i
            break
```

그림 7. 관심 영역 재구성 알고리즘 코드

python의 If문을 이용함으로써, 직선 구간에서 주행 시, 기존의 관심 영역을 따르지만, 조향 값이 일정 값 이상이 될 때, 관심 영역을 기존보다 높고 넓게 잡아주어 추세선 위치를 교정함으로써, 자율 주행 자동차 모델이 곡선 주행 시, 실제 차선에 인접한 추세선에 따라 주행할 수 있도록 해주었다. 이 내용을 토대로 [그림 7]과 같이 코드를 작성하였다. 하지만, 관심 영역의 위치나 범위의 크기에 따라 추세선이 변화하는 정도가 다르므로, 적절한 값을 찾아주어야 한다. 그래서 곡선 주행에서의 추세선과 실제 차선이 일치하는 정도를 파악하여 적정 값을 찾아주었다.

표 1. 관심 영역의 범위의 크기에 관한 결과

재조정된 관심 영역	결과
기존보다 작을 때 (x < 120)	관심 영역의 크기가 작아, 추세선과 실제 차선과의 차이가 컸다.
기존과 동일할 때 (x = 120)	관심 영역의 크기가 적당하지만, 추세선을 제대로 잡을만한 정보가 부족했다.
기존보다 클 때 (x > 120)	관심 영역이 기존보다 커서, 추세선을 잡을 정보가 충분하였다.

[표 1]의 결과를 수용하여 여러 번의 시뮬레이션 결과에 따라, X좌표 280에서 420까지의 범위를 곡선 주행에서의 관심 영역으로 설정했을 때, 곡선 코스의 추세선이 실제 차선에 가장 근접되게 교정될 수 있었다.

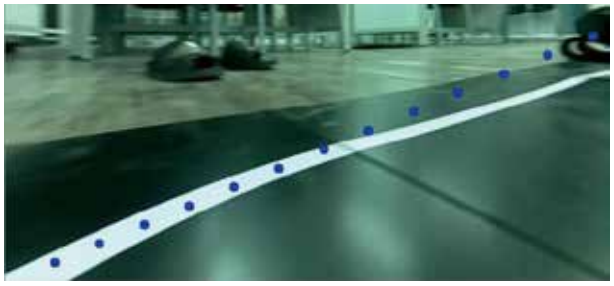


그림 8. 추세선 교정 전

관심 영역이 고정되어있는 기존의 알고리즘을 적용했을 때 나타난 추세선의 모습이다.

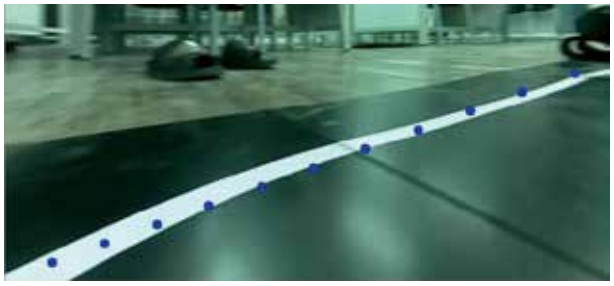


그림 9. 추세선 교정 후

[그림 9]는 [그림 8]과 똑같은 차선이지만, 관심 영역이 곡선 주행일 때, 재구성되어 실제 차선과의 오차가 줄어든 모습이다.

2.3. 추세선 기울기 이용

기존 알고리즘에서 추세선이 차량 모델을 기준으로 왼쪽 차선인지, 오른쪽 차선인지 구분하는 방법은 [그림 10]과 같이 추세선의 특정 X값이 $Center_X$값을 기준으로 더 작다면 왼쪽 차선, 더 크다면 오른쪽 차선으로 인식하는 구조이다.

```
if line(400) < center_x:
    line = 'left', i
    break
else:
    line = 'right', i
    break
```

그림 10. 기존 차선 방향 인식 알고리즘

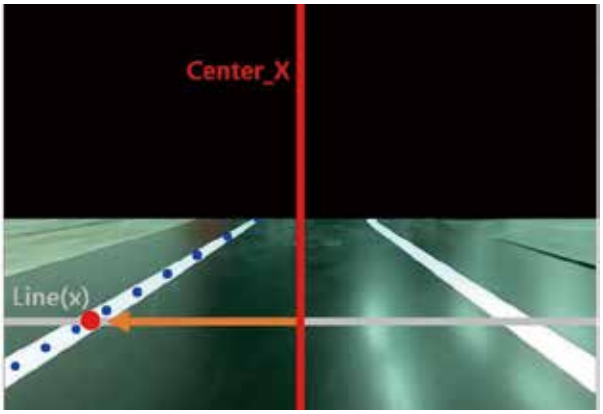


그림 11. 기존 알고리즘의 차선 방향 인식

하지만, [그림 12]와 같이 곡선 주행상황에서 회전하고 난 뒤, 직선 주로로 복귀하는 과정에서 기울어진 차선 하나로만 인식하게 되는데, 차선의 위치가 중앙 쪽에 걸쳐있어, 실제 오른쪽 차선을 차량이 왼쪽 차선으로 오인할 수 있는 상황이 발생할 수 있었다.

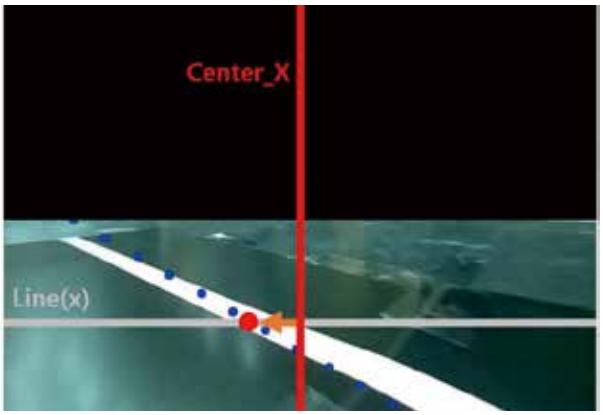


그림 12. 오른쪽 차선을 왼쪽 차선으로 인식한 상황

기존 알고리즘의 원리에 따라서 차선 방향을 오인한 예시[그림 12]이다. 이러한 문제점을 해결하기 위해, 차선 방향 인식 원리를 다르게 재구성한 알고리즘을 [그림 13]과 같이 구현하였다.

```
if line(480)-line(479) < 0:
    line = 'left', i
    break
else:
    line = 'right', i
    break
```

그림 13. 차선 인식 개선 알고리즘

추세선의 기울기가 차선 방향에 따라 음수값 또는 양수 값으로 나뉜다는 점을 이용하여, 추세선의 기울기가 음수 값이라면 왼쪽 차선, 양수 값이라면 오른쪽 차선으로 인식할 수 있게끔 구축하였다. 개선된 알고리즘을 이용하여 차선이 카메라의 중앙 쪽으로 위치하여도 추세선의 기울기는 차선의 위치랑 관계가 없으므로, 차선이 애매한 위치에 존재하더라도 차선 이탈이 발생하는 일을 방지할 수 있었다. [그림 14]와 [그림 15]는 [그림 12]에 대한 정보이다.

```
왼쪽 차선 선택
line(400) 316.8167509053118
center_x= 324.46466 e= 267.3520902607805
steer : 80
velocity : 45
```

그림 14. 차선 방향 인식이 개선되기 전

[그림 14]는 오른쪽 차선임에도 불구하고, 왼쪽으로 인식하는 오류가 확인되었다.

```
오른쪽 차선 선택
line(400) 316.8167509053118
center_x= 324.46466 e= -252.64790973921947
steer : -50
velocity : 45

1.728 x - 374.4
1.7281668290521566
```

그림 15. 차선 방향 인식이 개선된 후

[그림 15]는 마지막 줄의 추세선 함수식의 기울기를 반영하여 오른쪽 차선으로 정상 인식하였다.

Ⅲ. 시뮬레이션 결과

각각의 다른 방식으로 차선 인식 능력이 개선된 3가지 알고리즘의 곡선 주행 능력을 기존의 자율 주행 알고리즘과 비교하였다.

3.1. 결과 분석

속도 값, 관심 영역, 추세선. 이 3가지 방향으로 개선된 알고리즘들은 곡선 주행 시뮬레이션의 결과가 기존보다 더욱 향상된 성향을 띄었다. 3가지의 알고리즘을 잘 접목하여, 적용한다면, 다방면으로 강화된 차선 인식 알고리즘을 도출해낼 수 있을 것이다.

3.2. 결과에 대한 오차

시간 관계상, 실제 자율 주행 자동차 모델이 아닌 자율 주행 알고리즘을 통해 시뮬레이션 된 결괏값을 분석한 결과이기 때문에, 실제 차량 주행의 결과랑 다르게 도출될 수 있다. 이런 가능성을 줄이기 위해, 여러 번 시뮬레이션을 실행해보았으며, 여러 가지 변수를 대입했다. 따라서 본 연구 결과를 무조건 수용하기는 어려우나, 심화한 연구를 통해 논문의 주제를 발전시켜나간다면, 자율 주행 알고리즘에 적용할 가치가 충분하다고 생각한다.

Ⅳ. 결론

이 연구에서 고찰된 차선 인식 알고리즘은 시뮬레이션 과정을 통해 향상된 결과치를 보여주고 있다. 첫 번째 시뮬레이션 알고리즘은 차선 굴곡에 따른 조향 값 변화를 감지하여, 속도 값에 영향을 주어, 커브 주행에서 감속되어 차선 정보 인식률을 높여 안정적인 곡선 주행이 이루어졌다. 두 번째 시뮬레이션 알고리즘은 직선 주행에서의 관심 영역과 곡선 주행에서의 관심 영역을 구분화하여, 기존 알고리즘에 있었던 실제 차선 추세선과의 오차를 최소화할 수 있었다. 그 결과 차선 인식 정보의 오차율을 줄여 안정적인 곡선 주행이 가능했다. 마지막 시뮬레이션 알고리즘은 곡선 주행 시, 때때로 발생하는 차선 방향 오인 문제를 해결하기 위해, 차선 방향 인식 원리를 다른 방식으로 발전시켜 올바른 차선 인식이 가능하게 되었다. 따라서 곡선 주행 중 차량이 차선을 완전히 이탈할 가능성이 줄어들게 되었고, 안정적인 곡선 주행이 가능하였다. 종합적으로 이 3가지의 알고리즘 시뮬레이션 결과치는 실질적 모델링의 주행 결과와 다를 수는 있어도, 전반적으로 향상된 곡선 주행이 가능하여진다는 결괏값에 도달하게 되었다.

참고문헌

[1] 손행선, 이선영, 민경원, 서정진, “IMP기반 곡선 차선 검출기 하드웨어 구조 설계 및 구현”, 한국정보전자통신기술학회 논문지 제10권 제4호 pp. 304-310, 2017.

[2] 오현찬, 이웅희 정지원, 정석우, 유하람, 홍준, 정찬영, 도종용, 심현철, “곡선 근사 방식을 이용한 차선 추정 기법”, 한국자동차공학회 pp.1296-1299, 2016.

[3] 김건정, 김상동, 허수정, 이종훈, 박용완, Advanced Automotive Sensor Technologies: RADAR and LIDAR vol.25, no.3, SK telecom, pp. 383-405, 2015.

[4] David Douglas, Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), pp. 112–122, 1973

03 원근법이 고려된 차선폭을 이용한 차량의 주행 방향 결정 알고리즘

하나고등학교 김마린, 방수현, 임정환, 장현욱

원근법이 고려된 차선폭을 이용한 차량의 주행 방향 결정 알고리즘

using the distance between lanes in consideration of perspective, for determining the direction to drive a vehicle

김 마 린, 방 수 현, 임 정 환, 장 현 욱*
Ma-rin Kim, Su-hyeon Bang, Jeong-hwan Lim, Hyoun-wook Chang

요약

자율 주행 알고리즘의 기본 원리는 차선을 인식하여 차체의 중앙이 인식된 차선들로부터 떨어진 정도, 회전한 정도를 기준으로 운행을 제어한다. 그러나 차선을 인식하는 과정에서 의도치 않은 선을 차선으로 인식하는 오류가 가능하다. 본 연구는 기본 원리의 토대가 되는 차선 인식에 대해 카메라를 통해 인식된 여러 차선 중, 중앙에 가장 가까운 차선을 선택한 후 직선 차선의 기울기 변화를 비교하여 유효한 차선을 선택하는 방식을 제안한다. 이 과정에서 카메라를 통해 인식된 환경은 원근법에 의해 실제와 차이를 보이기 때문에 카메라의 위치에 따라 이를 보정해주어야 한다. 이는 차량별로 고유한 값을 지니며 이에 대해 고찰한다. 위 알고리즘은 차선들이 복잡하게 얽힌 환경에서 장점이 있다.

Keywords: 자율 주행 자동차, 차선 인식, 주행 알고리즘, 기울기 변화, Diff

I. 서 론

자율 주행 자동차의 기본적인 주행 원리는 카메라와 라이다를 통해 인식된 이미지를 통해 차선을 인식하고 이에 따라 미리 작성된 알고리즘에 의해 차량의 운행을 제어한다. 이때 작성되는 알고리즘의 유형은 매우 다양한데, 각각의 장단점이 존재한다. 알고리즘을 수행하기 위해 가장 먼저 이루어져야 하는 것은 카메라를 통해 주변 환경의 이미지를 받아들이는 것이다. 하지만 카메라는 원근법에 따라 실제와는 다른, 왜곡된 이미지를 가져온다. 따라서 실제 차선의 모습과도 미세한 차이가 존재하는 이미지를 가져오는데, 이를 알고리즘에 적용하는 과정에서 보정은 필수적이다. 따라서 본 연구에서는 보정을 위해 필요한 차량의 유형별 고유한 값에 대해 고찰하고 차선으로 인식될 수 있는 다양한 환경 속에서 유효한 차선을 계산해내기 위한 알고리즘을 제작하여 실험을 위한 차에 적용하였고, 시험 도로 주행을 완료하였다.

II. 주행 알고리즘

2.1 차선 인식 장애 해결 알고리즘

2.1.1 장애 제거 알고리즘의 필요성

실제 도로에서 실제 차량을 주행하는 과정에서는 많은 변수들을 만나게 된다. 생각지도 못한 요철이 있을 수도 있으며, 음영을 가진 방지턱, 각종 쓰레기나 장애물들로 인한 차선 인식에 어려움 또는 심각한 장애를 겪을 수 있다. 본 연구의 주행 알고리즘에 따르면 차선을 정상적으로 인식하는 것을 전제로 하는 것을 알 수 있다. 그러므로 위에서 설명한 여러 가지 요소들로 인한 차선 인식에 관한 변수들을 제거하는 과정(알고리즘)이 필수적으로 필요하다.

그림 1. 요철이 존재하는 시험 도로

2.1.2 Weird 함수의 정의

앞선 문단에서 앞, 뒤 카메라로 인식되는 이미지를 처리 및 적용하는 방법을 설명하였다. 또한 본 연구의 주행 알고리즘은 오로지 앞 카메라를 통한 상황 판단이 이루어지기에, frontLines 배열의 값(앞 카메라를 통해 인식된 차선 정보)을 분석하여 위에서 설명한 차선 인식에 관한 변수를 제거하는 과정을 거친다. 즉 인식되는 각 차선에 대하여 인접한 점들의 기울기(좌표 간의 차를 이용)를 비교하여 분석한 뒤, 그 편차가 특정 임곗값보다 크다면 앞서서 언급한 장애 상황이라고 판단할 수 있을 것이다. 반대로 특정 임곗값보다 작다면 정상 상황이라고 인식한 뒤 다음 과정을 이어간다.

* 김마린 (하나고등학교, kimmarin0420@naver.com)
방수현 (하나고등학교, rogella8989@gmail.com)
임정환 (하나고등학교, ldihkn3@gmail.com)
장현욱 (하나고등학교, joseph.hw.chang@gmail.com)

```
def weird(arr):
    l = []
    l2 = []
    for i in range(len(arr)-1):
        l.append((arr[i+1][1] - arr[i][1])/(arr[i+1][0] - arr[i][0]))

    for i in range(len(l)-1):
        l2.append(l[i+1] - l[i])


    if max(l2) - min(l2) > 8:
        return True
    else:
        return False
```

그림 2. Weird 함수

2.1.3. frontLines의 전처리

윗 문단에서 frontLines의 값을 처리한 방식을 이용하여 알고리즘을 작성할 수 있다. 즉 인식된 차선의 개수만큼 반복하여 각 차선의 y값(인식된 이미지의 수직 성분)중 가장 작은 값들을 저장하는 temp를 지정하고 이를 오름차순 처리한 arr 배열을 입력받는다.

2.1.4. 배열을 통한 상황 판단

입력받는 배열은 2차원 배열로써, 인식된 차선의 각 점의 좌표를 담고 있다고 볼 수 있다. 이를 이용하여 arr의 각 인자들에 대하여 인접한 인자들 간의 기울기를 비교한다면, 인식되는 차선의 기울기를 알 수 있다. 두 배열(l, l2)를 정의한 뒤, 각각의 배열에 frontLines[idx]의 인접한 두 점 사이의 기울기와 인접한 기울기들 사이의 차이를 각각 저장한다. l2배열에 위치한 데이터 중 가장 큰 값과 가장 작은 값을 구한 뒤, 그 사이의 차이를 구하면 편차를 구할 수 있다.

2.1.5. 임곗값 설정

임계 값의 설정에 따라 차선 인식 장애 판단 알고리즘의 성능이 크게 좌우되기에 적절한 임곗값을 지정하는 것이 매우 중요하다. 본 연구에서는 실험적으로 최적의 임곗값을 찾을 수 있었다. 제공된 시뮬레이터를 통해 각각 7개의 차선 데이터(좌진입 직선, 좌진입 좌회전, 우진입 좌회전, 좌진입 우회전, 우진입 우회전, LR, S자)에 대하여 불량차선 인식을 할 수 있었다. 각각의 임계값에 대하여 weird 함수에서True와 False가 반환되는 비율을 비교하여 True의 비율이 가장 높은 값을 선정하였다. 그 결과 정수 8이 가장 적합한 임계값으로 선정되었다.

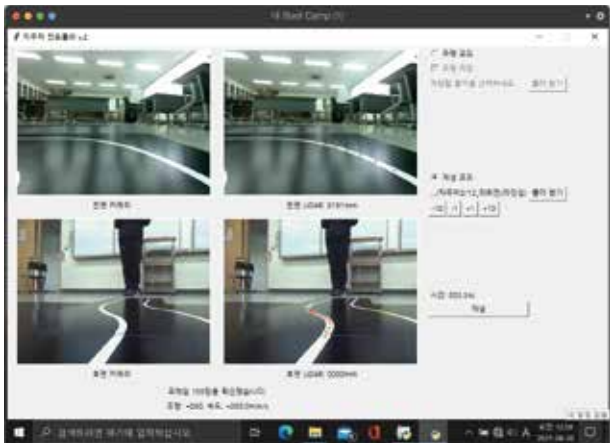


그림 3. 시뮬레이터를 이용한 불량차선 인식 과정

2.1.6. Weird 함수의 반환 값

전 문단에서 구한 임계값을 기준으로 불량차선을 판단한다. 편차가 임계값보다 크다면 불량차선으로 인식할 수 있기에, 이 상황에서는 True를, 다른 상황에서는 False를 반환한다.

2.2. 주행 방향 결정 알고리즘

2.2.1 주행 방향 결정 알고리즘의 목적

주행 방향 결정 알고리즘의 목적은 주변의 도로를 인식해 도로에서 벗어나지 않도록 주행 방향을 결정하는 것이다. 가장 기본적인 직선도로에서조차 차량의 주행 방향이 잘못될 경우 도로에서 벗어날 수 있기에 주행 방향을 결정하는 알고리즘은 필수적으로 필요하다. 직선, 곡선, 수직 도로 등 실전에서 볼 수 있는 수많은 도로에서 차선을 침범하지 않고 올바른 방향으로 갈 수 있는 알고리즘은 제작하는 것이 주행 방향 결정 알고리즘의 목적이다.

2.2.2 우회전/좌회전 결정

주행 방향을 결정하기 위한 첫 단계는 차량이 우회전할 것인지, 좌회전을 할 것인지 결정하는 것이다. 그러기 위해 카메라에 보이는 화면의 중앙과 화면의 중앙에 가장 가까운 차선의 거리를 비교한다.

카메라에 보이는 화면은 가로 640픽셀이다. 즉, 화면의 중앙은 x좌표가 320인 픽셀들의 모임일 것이다. 이 정보를 이용해 320에서 frontLines[idx][0][0] (가장 가까운 차선의 가장 차량과 가까운 점의 x좌표)를 뺀 값이 양수라면 오른쪽 차선이, 음수라면 왼쪽 차선이 차량에 더 가깝다는 것을 알 수 있게 된다. 차선을 따라 올바른 방향으로 주행하기 위해 전자의 경우에는 좌회전을 해야 하고, 후자의 경우에는 우회전을 해야 한다. 아래 그림은 이를 적용한 예시다.

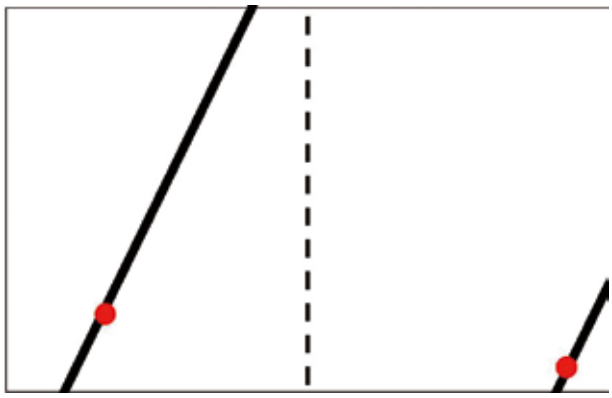


그림 4. 카메라 화면의 예시

위 그림에서는 왼쪽 차선의 점이 오른쪽 차선의 점보다 중앙에 더 가깝기에 frontLines[idx][0][0]는 왼쪽 점의 x좌표를 의미한다. 또한 왼쪽 좌표의 점이 중앙보다 -x방향에 자리 잡고 있기에 우회전을 해야 한다.

```
print("line : ", idx)
if 320 - frontLines[idx][0][0]<0:
    stat = "r"
else:
    stat = "l"
```

그림 5. 우회전/좌회전 결정 알고리즘 코드

이를 구현하는 코드 자체는 위 그림과 같이 매우 간단하다. 앞서 설명한 논리를 따라 우회전을 해야 한다면 'stat' 변수에 'r'을, 좌회전해야 한다면 'l'을 저장한다.

2.2.3 회전 정도 결정

차량이 우회전할 것인지, 좌회전을 할 것인지를 결정한 뒤에는 각 방향에 따라 바퀴를 어느 정도로 회전시켜야 하는지 되는지 결정해야 한다.

이와 관련된 변수는 steer 변수로, steer의 절댓값이 클수록 바퀴가 더 많이 회전하는 것이다. 그리고 steer의 부호는 우회전과 좌회전을 결정하는데, 이는 앞서 구한 stat 변수에 따라 결정하게 될 것이다.

steer의 크기를 구하기 위한 알고리즘을 간단하게 설명하면 양쪽 차선의 실제 중앙점을 예측한 뒤 그 중앙점을 향하도록 steer의 절댓값을 정하는 것이다.

```
dif = {398: 507.337, 389: 416.25083, 348: 357.23248, 332: 288.36295, 320: 262.63855,
p1 = frontLines[idx][0]

if stat == "r":
    mid = p1[0] - ((dif[int(p1[1])])/2)
elif stat == "l":
    mid = p1[0] + ((dif[int(p1[1])])/2)

steer = (mid-320)*0.8
```

그림 6. steer 결정 코드

dif에는 실험적으로 화면에 나온 두 차선의 차에 가장 가까운 점의 거리를 측정한 결과를 입력한 것이다. 원근법 때문에 선이 멀리 있을수록 차선의 거리가 달라진다. dif에 a:b 꼴의 것들이 여러 개 있는 것을 볼 수 있는데, a가 y좌표, b가 차선 사이의 거리를 의미한다. p1은 인식된 점 중 중앙에 가장 가까운 점의 2차원 좌표 (x, y)를 의미한다.

이 코드대로라면 중앙에 가장 가까운 점의 y좌표를 정수로 반올림한 다음에 그에 해당하는 두 점 사이의 거리의 실험값을 갖고 온다. 그리고 이를 2로 나누어 그 중앙값, 즉 도로의 중앙과 차선의 거리를 구한다. 마지막으로 이를 stat이 r이냐 l이냐에 따라 중앙값에서 빼거나 더한다. 이를 통해 현재 도로의 중앙 x픽셀값 (mid)를 예측하는 것이다.

도로의 중앙을 예측한 뒤에는 mid에서 320을 빼고 보정값을 곱한다. 이 역시 실험적으로 나온 값으로 이 보정을 해줌으로써 필요한 만큼 차량이 꺾인다는 것을 확인했다. 실험적으로 정한 보정값은 0.8로 결정하였다.

우회전/좌회전 결정 알고리즘과는 달리 회전하는 정도를 결정하는 알고리즘은 짜기 위해서는 실험적인 데이터를 많이 사용해야 했다. 우회전/좌회전과 같은 경우에는 정확한 수치없이 대략적인 값만 알아도 판단이 가능했지만 회전의 정도는 상대적으로 정확한 수치를 필요로 한다. 이 때문에 실험적으로 구한 상수들을 이용하는 것이다.

2.2.4 steer에 따른 이동속도

```
if abs(steer) > 40:
    velocity = 40

if steer > 95:
    steer = 95
elif steer <-95:
    steer = -95

if abs(self.vars.prev_steer - steer)>40:
    steer = self.vars.prev_steer
    print("#######steer error########")

print("prev steer : ", self.vars.prev_steer)
self.vars.prev_steer = steer

print("steer : ",steer)
print("velocity : ",velocity)

return steer, velocity
```

그림 7. 이동속도 결정 코드

안전성을 고려하여 얼마나 큰 각도로 도냐, 즉 steer의 절댓값이 얼마나 크냐에 따라 이동속도를 다르게 설정하였다. 많이 회전하는 상태에서 너무 빠르게 이동할 경우 다시 방향을 꺾기 전에 차선을 이탈하거나 너무 극단적으로 방향을 계속 바꿔야 할 수 있다. 이는 차량의 하드웨어에 무리를 가할 수 있기에 지양해야 한다.

따라서 steer의 절댓값이 40보다 큰 경우에는 속도를 40으로 고정한다. 또한 steer의 절댓값이 95를 넘지 못하게 설정하였다. 마지막으로 이전 steer 값과 현재 계산된 steer 값이 차이가 너무 많이 날 경우 이전 steer 값을 현재의 steer로 사용되게 했다.

2.3. error 처리

2.3.1 무(無) 차선 인식 error 처리

앞 문단에서 각 차선에서 인식하는 수직 방향의 값들을 크기별로 정리한 배열을 만들었다(temp). 그리고 이를 y값을 가장 작은 값에서부터 큰 값까지 오름차순으로 정렬한 배열을 새로 생성하였다.(arr). 차선이 인식되지 않으면 이 배열이 들어오지 않는다. 즉, arr의 성분의 개수가 0개일 때는 차선 인식 자체에 error가 발생한 경우이다.

그림 8. 차선이 인식되지 않는 경우

이때 별도로 차량의 행동을 정의할 필요가 있다. 따라서 먼저 error임을 표현하기 위하여, 컨트롤러에 차선 없음을 출력하여 사용자가 알 수 있다. 차체의 행동 모드에 따라 기존 counter 값이 0일 때 특정 값으로 변환한다. counter 값이 특정한 값보다 작다면 이전 행동 모드에 따라 자율 주행차의 행동을 정의한다. 차선이 인식되지 않은 상황에서도 차량이 운행을 지속하기 위함이다. 따라서 steer 값을 유지하고 counter를 줄여나가면서 출력한다. 좌우 모두 동일하게 같은 방식을 따라 차체의 행동을 제어한다. counter가 특정 값 이상이라면, 다시 차선을 인식할 시간을 확보하기 위해 저속으로 후진한다.

2.3.2 weird에 따른 error 판단

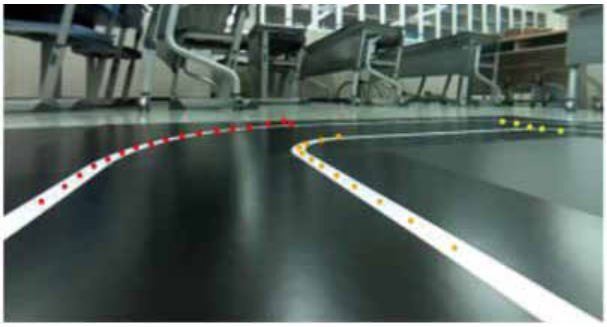


그림 9. weird 함수에서 정의한 오류

또한, 다른 error 상황을 처리하기 위해 함수 weird를 정의하였다. weird 함수에서 판단하는 장애 상황은 장애물이나 다른 요철에 의해 실제 차선과 다른 점들이 인식되어 이를 이은 선들이 차선으로 인식된 경우를 판단한다. 위의 사진에서 나타난 바와 같이, 주행에서 의미 있는 차선만이 인식되는 것이 아닌 다른 요인에 의해 의도하지 않은 점, 차선이 인식되어 운행에 지장을 줄 수 있다. 위의 사진에서 만약 빨간색, 주황색 점이 인식되지 않고 노란색 점들만 인식되었다면, 차량은 이 프로그램에서 노란색 점을 이은 선을 차선으로 인식하고 그에 맞추어서 운행을 제어한다. 이러한 error가 발생하는 것을 처리하기 위해 weird 함수를 작성하였다. weird라는 단어에서 알 수 있듯, weird에서 차선 인식에 장애 상황이 발생하였다면 true가 반환된다. 따라서 이 경우에는 이전의 steer 값을 유지한 채 속도를 특정값으로 설정하여 움직인다.

2.3.3 steer값 차이에 의한 error

이전에 설정된 steer값과 새로 계산해낸 steer값의 차이가 크다면 이는 하드웨어에 무리가 갈 수 있다. 따라서 이들의 차이의 절댓값이 특정값 이상이라면 이를 'steer error'라고 판단하여 이전의 steer값을 유지하여 운행하도록 하였다.

2.3.4 인지하지 못한 error 발생

무(無) 차선 인식 error, weird에 따른 error 등을 제외한 error가 발생할 수 있다. 인지하지 못한 에러가 발생할 때를 대비하여 counter에 따른 설정을 달리하였다. counter가 특정값 이하라면 이전의 행동 모드를 유지하여 속도를 계속 이어나간다. 반대로 counter가 특정 값 이상일 때 후진하여 error 발생한 것을 인지하고, 수정하는 시간을 확보한다. 확보된 시간을 통해 차체가 차선을 다시 인식하여 문제를 파악하고 기존 프로세스를 다시 실행하여 차체의 행동을 제어하도록 설정하였다.

Ⅲ. 결론 및 제언

3.1 결론

본 연구에서는 차량의 카메라의 위치에 따른 차선의 기울기 값을 이용하여 자율 주행 알고리즘을 제작하였다. 특히 불량차선을 인식하며 분류하는 전처리 과정을 통해 본 연구의 자율 주행 알고리즘의 성능 및 정확도를 크게 향상시킬 수 있었다. 또한 제작한 자율 주행 알고리즘을 차량에 적용하여 7개의 코스(좌진입 직선, 좌진입 좌회전, 우진입 좌회전, 좌진입 우회전, 우진입 우회전, LR, S자)를 주행 완료하였으며 그에 따른 적합도를 검증할 수 있었다.

3.2. 제언

3.2.1 코너 진입의 한계

Formula 1, DTM(독일 투어링카 마스터즈), Wrc(World Rally Championship)과 같은 최상급의 레이싱 분야에서는 차량에 따른 서킷의 레코드 라인이 존재한다. 레코드 라인이란 차량의 접지 한계, 브레이킹 한계 및 기어비의 관계를 고려하여 코너에 진입하는 각도, 미션의 단수, 차량의 속도를 표시한 레이스 라인이라고 할 수 있다. 본 연구의 차량은 위에서 나열한 차량들의 성능을 가지고 있지 않으며, 일반적인 차량의 성능에도 미치지 못하지만, 공통점은 존재한다. 코너 진입에 따른 차량의 위치선정은 동일하다. 즉 우측 코너라면 코너에 진입하기 전, 차량을 최대한으로 차선의 왼쪽에 위치시켜 안정적인 스티어링 및 효율성을 만들어낼 수 있다. 즉 본 연구에 적용한다면, 차선을 침범하지 않을 수 있는 최적의 방법이다. 따라서 차선 침범과 같은 오작동을 제거하기 위해서는 이러한 레이싱 레코드 라인에 대한 추가적인 연구가 필요하다.

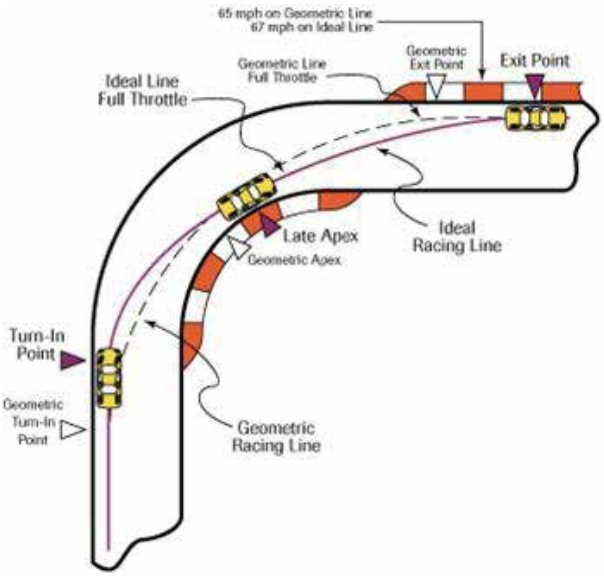


그림 10. Formula 1의 레코드 라인

3.2.2 후진 프로세스의 한계

본 연구의 자율 주행 알고리즘은 파악된 상황 'L(좌진입)' 또는 상황 'R(우진입)'이 아닌 경우에는 후진을 통해 전체 프로세스를 다시 반복한다. 즉 'L' 또는 'R' 상황이 아니면 조건 없는 후진을 하게 되는데, 이러한 상황에서는 차선을 침범할 가능성이 매우 커진다. 따라서 'L' 또는 'R' 상황이 아니라면, 뒷 카메라를 통한 차선 인식 과정 및 코너에 적합한 위치(레이싱 레코드 라인)를 고려한 알고리즘 제작이 필요하다.

3.2.3 weird 함수의 오류 가능성

차선 인식 장애 해결 알고리즘(Lane Recognition Fault Resolution Algorithm) 단락에서 언급한 Weird 함수는 인식된 차선의 기울기의 편차를 고려하여 불량차선을 판단한다. 하지만 차량의 이미지 센서가 인식하는 프레임은 항상 일정하지만, 이 상황에서 차량의 속도가 증가한다면, 정확한 불량차선 판단이 이루어질 수 없게 된다. 따라서 차량의 속도 증가에 따른 불량차선 인식 프로세스에 관한 추가적인 연구가 필요하다.

3.2.4 각 차량에 대한 고유 Dif 데이터 필요

본 연구에서 제시한 자율 주행 알고리즘은 차량 카메라의 위치에 따른 차선의 기울기 값을 이용한다. 즉 지면을 기준으로 차량의 카메라의 높이가 매우 중요한 변수라고 할 수 있다. 위에서 소개한 바와 같이 di에는 직선 주행을 한다고 가정할 때 컨트롤러의 화면에 표현된 두 차선의 차에 가장 가까운 점의 거리를 측정한 결과들이다. 예를 들어, 화면상에 있는 점의 y픽셀이 398이라면, 동일한 y픽셀을 가진 다른 직선 차선과의 실제 거리는 507.337만큼의 차이가 발생한 것이다. 이를 알고리즘이 선택한 y픽셀의 좌표(398)의 x픽셀 좌표에서 실제 거리의 절반을 빼거나 더하여 실제 도로의 x방향 중앙값을 계산한다. 이렇듯 현재 도로의 x방향 중앙 픽셀값을 유추하는 정보에 이용되므로 알고리즘에서 판단 기준이 되는 중요한 정보이다. 이는 원근법에 따라서 차선의 거리가 다르게 측정될 수 있으므로 차량 유형이나 하드웨어의 위치에 따라 달라질 수 있다.

본 연구에서 진행한 자율 주행 자동차의 모델의 전체적인 제원은 전고 약 130mm, 전장 약 225mm, 전폭, 약 100mm이다. 축간 길이 약 125mm, 앞 차륜 거리 약 70mm, 뒤 차륜 거리 약 90mm이다. 위 모델에서 카메라의 높이는 약 75mm이고, 라이다는 이보다 조금 아래에 있는 약 73mm에 위치한다. 하지만 실제 승용차의 전고는 평균적으로 1,500mm, 중형 SUV 기준 전고는 평균 1,750mm이다. 전방 카메라를 뷰 미러에 장착한다고 하였을 때, 바닥으로부터 카메라의 높이는 각각 대략 1,300mm, 1,500mm라고 할 수 있다. 또한, 연구에서 사용된 자율 주행차가 인식할 수 있는 라이다의 길이는 약 2M 정도에 불과하지만, 실제 주행상황에 맞게 사용되려면 라이다의 인식 범위는 승용차 기준으로 최대 200M, 카메라의 인식 범위는 최대 250M이다. 이 두 유형의 차량은 전고의 차이가 카메라의 실제 거릿값의 차이에 큰 영향을 끼친다고 보기는 어렵다. 하지만 또 다른 유형의 차량인 버스의 전고는 약 3,555mm이다. 이는 앞에서 본 승용차나 중형 SUV의 경우와 대략 2배 정도 차이가 나는 값으로, 이 높이에서 측정한 라이다의 화면상 픽셀 차이와 실제 거리상의 차이는 유효할 정도로 차이가 난다고 할 수 있다. 또한 오토바이도 전고가 약 1,100mm로 다른 차량들에 비해서 상당히 낮다. 이러한 경우에 대한 카메라로 측정한 화면에서의 두 차선 간 픽셀값의 차이와 실제 거리 값의 차이가 심해져 알고리즘의 판단에서 사용될 도로의 중간값이 차량의 유형별로 차이가 상당히 벌어질 수 있다. 앞서 소개한 바와 같이 차량의 유형에 따른 전고의 차이, 카메라와 라이다를 위치시키는 높이의 차이에 따라 Dif 값에 차이가 존재할 수 있다. 이는 공정 과정의 단일화로 각각의 차량 유형에 따라 발생할 수 있는 오차를 줄일 수 있으나 다른 차량에는 다른 알고리즘, 다른 고유한 값들이 들어가기에 이에 대한 물리적인, 소프트웨어적인 추가적 해결책은 필요한 것으로 보인다. 또한 라이다의 측정 오차로 생각될 수 있는 요소 중 하나로 초기 공정 과정에서 발생하는 오차가 발생할 수 있다. 라이다가 차량의 좌표축과 기울어진 정도에 따라서 앞에서 소개한 dif 값이 달라질 수 있는 만큼 공정의 과정에서도 이는 고려되어야 할 사항이다. 공정 과정의 오류의 경우 이에 대한 오차를 좌표축 변환으로 해결하는 방안을 연구한 선행 연구가 있다.

참고문헌

[1] P.Lemerle, O.H ppner, J.Rebelle, "Dynamic stability of forklift trucks in cornering situations: parametrical analysis using a driving simulator", Vehicle System Dynamics, Vol.49, No. 10, pp. 1673-1693, 2011.

[2] Kihun Lee, Geunho Lee, Dohsik Kim, Gyoungmo Kang, Sangho Shin, "Simulation and Anaysis using Dynamic System Modeling for Forklift Truck Automatic Transmission", KSAE06-S0143, pp. 897-902, 2006

[3] 김평원,정형식,홍범진, 함께 만드는 인공 지능 자주차인천대학교 출판부, 2019.

[4] Dugoff, H.,Fancher, P.S. and Segal, L., "Tyre performance characteristics affecting vehicle response to steering and braking control inputs" Final report, Contract CST-460, Office of Vehicle System Research, US National Bureau of Standards, 1969.

[5] 조해준, 이재천, 곽성우. (2020). 자율 주행 자동차용 3D 라이다의 초기 장착 오차 보정. 한국지능시스템학회 논문지, 30(6), 417-423.

우수 논문

Journal of Youth Engineering

04 딥러닝을 활용한 도로 종류 파악

선덕고등학교 정영찬

딥러닝을 활용한 도로 종류 파악

Identifying Road Type by Using Deep-Learning

정 영 찬* Jeong Yeong Chan

요약

프로세서의 발달과 자율 주행 자동차의 보급으로 자율 주행 자동차의 안전에 대한 요구가 증가하였다. 이에 따라 자동차의 자율 주행능력을 완벽에 가까운 수준까지 올리기 위해 다양한 알고리즘이 제안되는데, 그중 하나는 딥러닝을 이용해 차선을 인식하거나 GPS를 보완하는 것 등이다. 본 논문은 기존 자율 주행 자동차 코드로부터 산출된 주행이 옳은 방향인지를 검증하기 위한 교차 검증용 딥러닝 알고리즘을 제안한다. 도로 종류를 파악할 수 있도록 자율 주행 자동차가 인지하는 도로 종류를 모아 각각의 데이터 세트를 모았으며, 오차를 줄이기 위해 Canny Image를 데이터 세트로 설정하였다. 그리고 구글 Teachable Machine을 통해 데이터를 학습시켰으며, 이 학습된 딥러닝을 통해 파이썬으로 구현할 수 있었다. 구현하는 과정에서 발생한 다양한 문제점들이 보완될 필요성이 있으므로 그 부문을 제안한다.

Keywords: 자율 주행 자동차, 딥러닝, 케라스

I. 서 론

자율 주행 기능이 포함된 차량이 시중에 속속 등장하면서 자율 주행 자동차의 보급이 빠르게 진행되고 있다. 특히 자율 주행 자동차와 전기차의 결합으로 그 보급 속도가 증가하고 있는데, 산업연구원 보고서에 따르면 한국의 경우 자동차 수요가 2009년 146만 1,865대에서 2017년 179만 8,796대로 평균 성장률이 25.3%에 육박한다.[1] 거기에 리싱크X 보고서에 따르면 2030년 전체 자동차 가운데 60%를, 전체 주행 거리의 95%를 자율 주행 자동차가 차지할 것이라는 전망하기도 했다.[2] 이와 같은 증가 추세로 자율 주행 자동차에 대한 대중들의 인식이 높아지면서 자연스럽게 안전 문제와 윤리 문제에 관심이 집중되었다.

이에 따라 자율 주행 자동차에서 가장 중요한 기술인 '자율 주행'의 안전을 제대로 확보하기 위해 정부는 145억 원을 투입하여 차선 인식, 자율 주행 기술 개발에 총력을 기울이고 있다.[3] 이와 관련하여 다양한 방안을 활용한 차선 인식 방안들이 제시되었는데, 그중 딥러닝을 활용한 방안이 있다. 기존 연구에 따르면 차량의 정확한 위치 파악을 위해 자동차에 탑재된 센서와 딥러닝을 융합거나[4] 네비게이션에 추가 정보를 제공하기 위한 목적으로 딥러닝과 차선 인식을 활용하거나[5] 차선변경을 위한 매끄러운 경로를 제공하기 위해 딥러닝을 활용하거나[6] 딥러닝을 사용하여 차선 자체를 인식하는 방법을 제시하기도 한다.[7]

그러나 본 연구는 자율 주행 자동차가 도로 위에서 주행할 때 Canny Image만을 활용한 차선 인식으로 결정된 차선이 실제 눈에 보이는 방향과 맞는지 교차검증을 위한 목적으로 딥러닝을 이용하고자 한다. 즉, 보조적 용도로 사용가능한 딥러닝 교차검증 시스템을 제작해보고자 한다. 그리고 이를 통해 소형 프로세서를 통해서도 매끄럽고 빠른 연산이 가능한 알고리즘을 위해 관련 방안을 제안한다.

딥러닝이란 여러 비선형 변환기법의 조합을 통해 높은 수준의 추상화를 시도하는 기계학습 알고리즘의 집합이다.[8] 이 중 심층 신경망(Deep Neural Network, DNN)은 입력층과 출력층 사이에 여러 은닉층들로 이루어진 인공신경망(Artificial Neural Network, ANN)이다.[8-9]

이때 입력층으로 들어온 특정 이미지의 내용은 은닉층으로 가면서 특정 함수를 거친다. 그 함수는 다음과 같다.

$$a^{(1)} = \sigma(Wa^{(0)} + b) \quad (1)$$

이때 $a^{(1)}$는 다음 층의 뉴런을 의미한다. $\sigma(x)$ 함수는 로지스틱함수, 혹은 시그모이드 함수(Sigmoid Function)라고 불린다.

그리고 시그모이드 함수 내에 합성되어있는 함수는 가장 간단한 형태의 1차 회귀식으로 다음과 같다.

$$H(x) = Wx + b \quad (2)$$

그리고 이러한 딥러닝 모델을 더 학습시키기 위해서는 비용함수라는 피드백이 필요하다. 식(2)에 존재하는 값 W와 b를 결정할 수 있는 함수를 정해야 한다. 데이터 분포에 따라 정확한 회귀선을 그려야 하므로 각각의 데이터와 회귀선 간 거리가 가장 짧은 선으로 회귀선을 결정해야 한다. 이때 이 두 요소 사이의 거리를 재는 함수를 비용함수(Cost Function)이라고 한다.

$$H(x) - y \quad (3)$$

* 정영찬 (선덕고등학교, ingabbang@gmail.com)

각각의 요소 하나만을 생각할 때 식(3)으로 따질 수 있다. 식 (3)은 제 1사분면에만 존재하는 데이터에 한하여 양수의 거릿값을 표현한다. 각각의 요소들의 y좌표 값과 함수 값 사이의 거리가 가장 최소가 되는 부분을 정해야 한다. 따라서 비용함수는 다음과 같이 정의된다.

$$Cost = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)}) - y^{(i)}\right)^2 \quad (4)$$

이때 거리를 계산해야 하기 때문에 거리는 제곱한다.

다음과 같은 비용함수를 통해 그 비용함수의 거리가 가장 적은 구간을 계산해내야 하므로 다음과 같이 편미분과 값 대입을 통해 계산한다.

$$\begin{aligned} &repeat\ until\ convergence\,(\text{수렴})\{ \\ &\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}Cost(\theta_0, \theta_1) \\ &(\text{f or } j = 0 \text{ and } j = 1) \\ &\} \end{aligned}$$

이때 $H(x)$에서의 W와 b를 각각 θ_1, θ_0 으로 나타내었다. 이러한 과정을 '경사 하강법(Gradient Descent)'라고 하며, 이와 같은 과정을 통해 비용함수의 거리가 가장 적은 구간을 계산한다. 기울기가 양수면 왼쪽, 음수면 오른쪽으로 이동하여 최종적으로 지역 최솟값(Local Optima)을 계산할 수 있다. 그리고 이와 같은 과정을 통해 α값으로 학습률(Learning Rate)을 계산할 수 있다.

지금까지의 과정은 출력을 모르는 상황에서 입력만 가지고 그 추세를 계산하는 것이었다면, 이번 논문은 입력(도로 사진)과 출력(도로 형태)이 모두 결정되어있으므로 이 데이터를 활용하여 학습시킬 수 있다. 이러한 과정을 위한 것이 역전파 알고리즘이다. 입력한 정보로 산출된 출력층의 결과물과 이미 알고 있는 정답과 비교한다. 이 과정에서 다른 값들에 각각의 차이(bias)를 두어, 즉 역으로 경사 하강법을 실시하여 다시 출력층에서 입력층으로 되돌아가는 연산을 거친다. 그리고 이 과정을 입력을 바꾸어가며 반복한다. 이런 과정을 역전파 알고리즘이라고 한다.

이렇게 모인 데이터들을 정립해 평균적인 차이값을 계산하면 되지만 연산이 너무 오래 걸린다. 이런 문제를 해결하기 위해 미니 배치(Mini-Batch)방식을 활용한다. 각각의 데이터를 몇 개의 미니 배치로 묶어, 컴퓨터가 각 미니 배치마다 역전파 알고리즘을 실시한다. 그리고 그 과정에서 발생한 평균 차이값을 계산한다. 이 과정을 미니 배치 수만큼 반복하여 최종 평균 차이값을 계산한다. 이 과정을 미니 배치 확률적 경사 하강법(Mini-Batch Stochastic Gradient Descent)이라고 불린다.

Ⅲ. 학습

도로 형태 데이터를 모으기 위해 다음과 같이 도로 종류를 나누었다.

표 1. 도로 종류 구별

		
직진	곡선 좌회전	곡선 우회전
		
직각 좌회전	직각 우회전	

그리고 이 이미지들은 그대로 활용될 수 없다. 오차율이 높아질 우려가 있으므로 다음과 같은 예로 모든 데이터를 원근 왜곡법을 이용하여 실제 도로로부터 위에서 바라보는 형태의 이미지(Bird-view Image)를 산출하고, 그 이미지를 Canny Image화 시켜 오차율을 줄인다. 그 과정은 [그림 1], [그림 2]와 같다.

Canny Image화 시킨 이미지들을 각 항목에 따라 데이터 세트로 제작한다. 각 데이터는 딥러닝이 구별하기 힘든 도로 종류일수록 더 많은 데이터 세트를 준비한다. 각각의 데이터 수는 다음 [표 2]를 참고하면 된다.

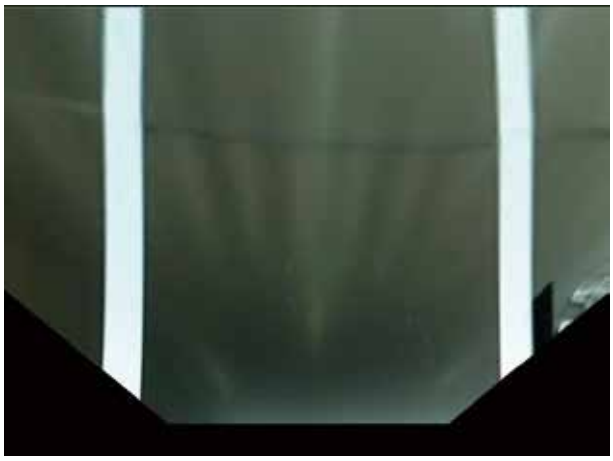


그림 1. 원근 왜곡된 도로 사진
Figure 1. Perspective Distorted Road Photographs

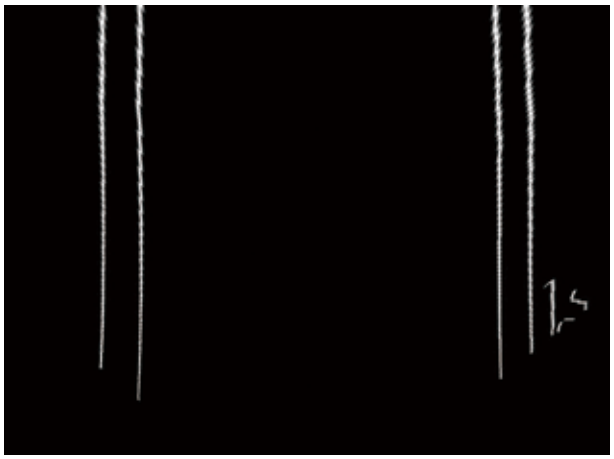


그림 2. Canny Image로 변형
Figure 2. Transform to Canny Image

표 2. 도로 데이터 수

도로 종류	데이터 수 (단위 : 개)
곡선 좌회전	27
곡선 우회전	29
직각 좌회전	24
직각 우회전	21
직진	14

이와 같은 데이터 세트를 구글에서 제공하는 Teachable Machin에 적용한다. Teachable Machine에 각각 'Right Angle Left', 'Right Angle Right', 'Curve Left', 'Curve Right', 'Straight'로 항목을 구분 짓고 데이터 세트를 적용해 학습시킨다. 이때 Epoch는 5000, Batch는 16, Learning Rate는 0.001로 설정하였다.

학습된 모델을 파이썬 코딩으로 적용한 후 예측 결과를 확인하였다. 테스트 이미지를 대입하고 각각의 항목으로 얼마나 추정되는지를 % 단위로 출력하도록 하였다.

Ⅳ. 결론 및 제언

학습시킨 모델로 이미지를 분석한 결과 다음과 같이 도출되었다.

도로 종류	항목	예측값(%)
	Right Angle Left	50
	Right Angle Right	42
	Curve Left	3
	Curve Right	1
	Straight	2
	Right Angle Left	0
	Right Angle Right	0
	Curve Left	99
	Curve Right	0
	Straight	0

표 3. 결과
Table 3. The Result

도로 종류를 구별하고 이 데이터를 구글 티쳐블 머신(Teachable Machine)을 이용하여 케라스 파일로 만들었다. 그리고 이 딥러닝 내용을 파이썬 코딩으로 구현하여 도로 종류를 구별할 수 있었다.

그러나 이 과정에서 다양한 오차가 발생하였으며, 특히 차선이 한 개만 인식되는 경우 곡선이 보임에도 불구하고 직선으로 인식하거나, 좌회전 도로임에도 우회전으로 인식하는 문제가 발생하였다. 특히 Canny Image로 데이터를 학습시켰음에도 불구하고 Canny Image로 데이터를 입력했을 때보다 변형 전 이미지를 입력했을 때 결과값이 더 잘 도출되었다. 이러한 내용을 보완하기 위해 딥러닝의 학습 수를 증가시키거나 데이터 수를 증가시키는 방안으로 딥러닝의 정확도를 증가시킬 필요가 있다.

또한 Canny Image와 원근 왜곡법을 거쳐 발생하는 화질 저하가 딥러닝의 학습률을 떨어뜨려 구분 짓지 못하게 하는 경우도 있다. 따라서 이러한 방안을 해결하기 위해 하드웨어의 개선도 제안된다.

또한 연산속도가 다소 지체되는 부분이 있어 알고리즘이 효율적으로 동작하도록 기존 자율 주행 자동차 코드를 수정해야 할 필요성이 있다. 딥러닝을 운행하기 위해 자율 주행 자동차로부터 들어오는 이미지 하나하나를 분석해야 하므로 주행 시 지체되는 부분이 있다. 이러한 부분을 수정해야 할 것을 제안한다.

참고문헌

[1] 이항구, 윤자열. 전기동력·자율 주행 자동차산업의 현황 및 전망. 산업연구원. 2018.8. 보고서번호:2018-332

[2] James Arbib, Tony Seba. Rethinking Transportation 2020-2030;The Disruption of Transportation and the Collapse of the Internal-Combustion Vehicle and Oil Industries. ReThinkX. 2017.5.

[3] "전기차·자율차 핵심기술 R&D 사업에 올해 279억원 지원". 연합뉴스. 2021.01.27. https://url.kr/658m1r. URL.2021.05.08. 접속.

[4] 유주희, 서재규, 최경택, 정호기. 센서 융합 기반 정밀측위를 위한 딥러닝 기반 차선 끝점 검출. 한국자동차공학회 추계학술대회, pp. 716-717, 2020.11.

[5] 전우태, 김태욱, 최대호, 김종찬. 모바일 내비게이션을 위한 딥러닝 기반 주행 차선 식별. 한국자동차공학회 추계학술대회, pp. 740-741, 2018.11.

[6] 유세욱, 서승우. 딥러닝 기반의 경로 생성을 이용한 차선변경의 개선방안. 대한전자공학회 학술대회. pp.632-633. 2019.11.

[7] 이동현, 이경민, 최병호, 인치호. 딥러닝을 이용한 적응적 차선 검출 알고리즘. 대한전자공학회 학술대회. pp.1571-1572. 2018.6.

[8] Y. Bengio, A. Courville, and P. Vincent., "Representation Learning: A Review and New Perspectives," IEEE Trans. PAMI, special issue Learning Deep Architectures, 2013

[9] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview" http://arxiv.org/abs/1404.7828, 2014

05 다중 제어기를 통한 자율 주행 차량의 승차감 개선

하나고등학교 이정호

다중 제어기를 통한 자율 주행 차량의 승차감 개선

Ride Quality Improvement via Multiple Controls on Steering

이 정 로* Lee, Jeong Ro

요약

본 연구의 목적은 저사양 환경에서 PID를 주 제어로 삼고 보조 제어기를 추가하여 원만한 자율 주행 알고리즘을 개발하는 것이다. 이를 위해 3축 가속도 측정장치를 이용해 Jerk를 평가하였으며 5개의 3종 타일을 이용하여 양방향 시범 운행을 하였다. 저사양 환경에서 구동할 수 있는 간단한 제어기를 설계하고 최소한의 자원만을 사용했다. 시스템에 대한 이해가 없는 PID 제어기를 포함한 만큼 여러 차례의 시험 운행을 거쳐 안정적인 승차감을 확보할 수 있었다.

Keywords: 다중 제어, 자율 주행, 승차감, 가가속도, 튜닝, 저사양

I. 서 론

저사양 환경에서의 자율 주행은 간단한 제어 알고리즘을 따라야만 성립할 수 있다. 본 연구는 주행의 효율 혹은 소요 시간 등에 초점을 맞추기보다 완만한 주행을 통해 안정감 있는 승차감을 얻고자 한다. 이를 위해 저사양 모델 차량의 승차감 평가 기준을 정하고 시험 운행을 통해 제어기 튜닝 과정을 거칠 것이다. 원활한 승차감 평가를 위해 일정한 속력으로 주행할 것이며 조향은 다중 제어기를 거칠 것이다. 본 연구는 평면 도로와 신호등, 차단기 등의 장애물을 상정하는 대신 조향에 초점을 맞추었다.

II. 3축 Jerk 승차감 평가

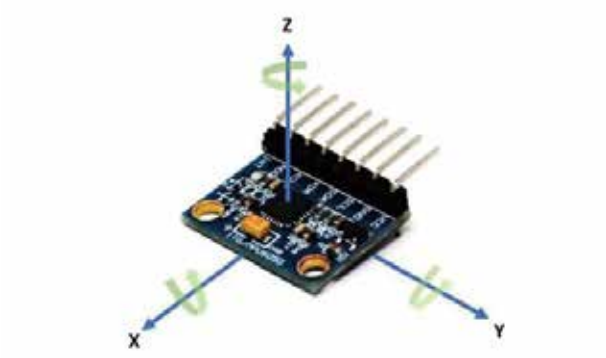


그림 1. MPU-6050 칩

차량 승차감 평가에 있어 가장 중요한 기준 중 하나는 가속도와 Jerk(가가속도)이다. 이에 따라 모델 차량의 승차감 또한 MPU-6050 칩을 이용한 3축 Jerk로 평가할 것이다.

$$j = \frac{da}{dt} = \frac{d^2v}{dt^2} = \frac{d^3r}{dt^3} \qquad (1)$$

관찰에서 사용된 MPU-6050 칩은 XYZ 축에 대한 가속도를 반환하며 이를 미분하여 Jerk를 얻을 수 있다.

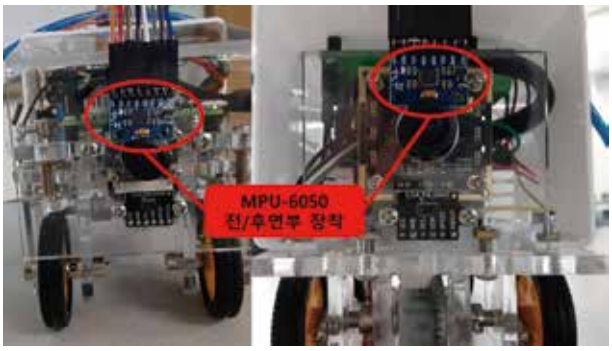


그림 2. MPU-6050 장착 후 모델 차량의 전면부와 후면부

III. Proportional-Integral-Differential Controller

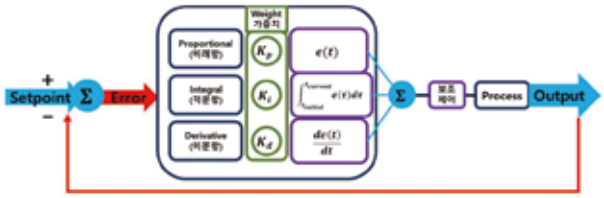


그림 3. 보조 제어기 간략화된 조향 제어 순서도

모델 차량에 사용된 임베디드 보드의 저사양 환경에 따라 PID(Proportional-Integral-Differential) 제어방식[1]을 택하였다. 다른 제어에 비해 비교적 적은 연산량에도 불구하고 직관성과 보편성 덕분에 이미 많은 디지털/아날로그 기기에 적용되어 있다. PID 제어는 시스템에 대한 이해 없이도 이용할 수 있으므로 수작업 공정에 따른 불균일 하드웨어, 배터리 잔여 전력에 따른 모터 출력 변화 등의 모델 차량 문제에 유동적으로 대응한다.

* 이정로 (하나고등학교, jrlego0331@gmail.com)

3.1. Ziegler-Nichols Tuning

	K_p	K_i	K_d	T_i	T_d
P	$0.5K_u$				
PI	$0.45K_u$	$0.54K_u / T_u$		$0.8\,T_u$	
PD	$0.8K_u$		$0.10K_u T_u$		$0.125\,T_u$
PID	$0.6K_u$	$1.2K_u / T_u$	$0.075K_u T_u$	$0.5\,T_u$	$0.125\,T_u$

표 1. Ziegler-Nichols Tuning Chart

PID 제어에 있어 적절한 Gain(이득 값)은 직진-직진 코스 시험 운행 데이터를 Ziegler-Nichols Tuning[2] 차트에 대입하여 얻어내었다. 승차감을 위한 튜닝인 만큼 직진 코스에서의 안정적인 주행을 목표로 한다.

3.2. Saturation

Saturation[3]은 Error의 누적으로 인해 출력에 대한 적분항의 영향력이 극대화되어 새로운 Error에 대한 비례항의 영향이 줄어드는 현상이다. 일반적인 Integral Anti-Windup Method인 Clamping은 타일에 따라 목푯값이 변하는 자율 주행에는 적용할 수 없다. 따라서 적분 interval을 단위시간마다 갱신한다.

3.3. 적분항 초기화

$$K_i \int_{t_{initial}}^{t_{current}} e(\tau)d\tau \quad (2)$$

주행 과정에서 적분항의 $t_{initial}$ 값을 유지하면 Saturation이 반복되므로 이미지의 frame 수를 기록하여 단위시간마다 $t_{initial} = t_{current}$ 로 갱신한다.

4. PID 이외의 제어

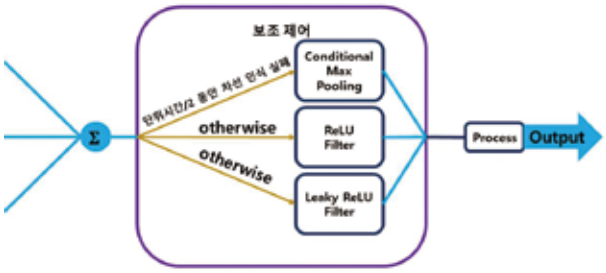


그림4. 보조 제어기 순서도, 이외 간략화됨

PID 제어기는 다양한 상황에 유동적으로 대처할 수 있지만 차선 인식을 전제해야만 유의미한 결과를 내놓고 하드웨어의 실제 능력을 벗어나는 $output$을 필터링할 수 없다. 따라서 PID 제어기를 거친 값을 가공하는 보조 제어기 또한 필요하다.

4.1 Conditional Max Pooling

모델 차량의 전면부 사각지대와 좁은 최소회전반경은 급격한 90도 타일을 주행할 때 치명적으로 작용한다. 차량은 해당 타일을 주행할 때 추적하던 차선이 갑자기 사각지대 안으로 사라지며 조향각 설정에 어려움을 겪게 된다. 단위시간 이상 임계범위 이내의 차선이 인식되지 않았을 때 기존 경향성을 유지하는 급격한 조향을 이행할 수 있어야 한다. 이때 단위시간은 적분항 초기화 단위시간의 절반과 같다.

4.2 활성화 함수

만약 $\Delta\ output$이 조향정밀도보다 작을 때 주행에 의미 있는 변화를 줄 수 없다. 또한, 미분항이 조정하지 못한 비정상 변화에 대응하는 안전장치로 $\Delta\ output$ 최대 임계 또한 필요하다. 이렇게 PID 제어를 거친 값을 특정 임계를 기준 삼아 변형시키는 기능이 인공신경망의 활성화 함수와 유사한 역할을 한다는 점에서 착안, 대표적 활성화 함수를 적용하였다.

4.2.1 ReLU

$$f(x) = \begin{cases} \min(-T_{\min}, x), & \text{if } x < 0 \\ \max(T_{\min}, x), & \text{if } x \geq 0 \end{cases} \quad (3)$$

최소임계는 ReLU[4] 활성화 함수를 통해 구현한다. $\Delta\ output$의 절댓값을 측정하여 조향정밀도 이하의 변화를 무시하도록 한다. 모델 차량의 조향정밀도가 1.8° 이므로 $T_{\min}$=1.8° 이다.

4.2.2 Leaky ReLU

$$f(x) = \begin{cases} \begin{cases} 0.1x, & \text{if } x < -T_{\max} \\ x, & otherwise \end{cases}, & \text{if } x < 0 \\ \begin{cases} 0.1x, & \text{if } x > T_{\max} \\ x, & otherwise \end{cases}, & \text{if } x \geq 0 \end{cases} \quad (4)$$

ReLU 함수는 *Dying ReLU Problem*을 가지기에 경향성을 유지해야 하는 최대 임계에 적용할 수 없다. 따라서 임계 이상의 값이 나왔을 때 10%만큼만을 반영하는 Leaky ReLU[5] 함수를 사용한다. 이를 통해 최대조향각 이상의 $output$을 방지하면서 $output$의 경향성을 다음 연산에 반영시킬 수 있다.

V. 결론 및 제언

5.1. 결론

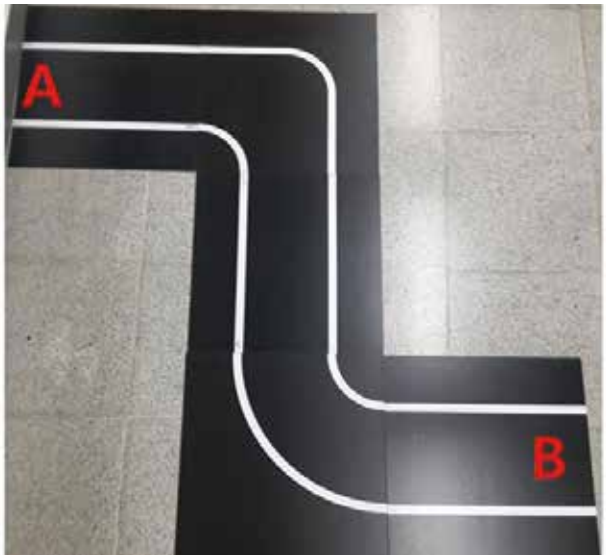


그림 5. 시험주행 도로, 두 개의 출발지 표시

본 연구에서는 그림 1과 같은 타일에서 A->B와 B->A 모두 주행하며 MPU-6050 칩을 통한 Jerk를 추적하였다. 주 제어인 PID 제어의 적분항 Saturation 문제를 해결하기 위해 주기적인 초기화를 적용했다. 또한, 시스템에 대한 이해도가 떨어지는 주 제어의 한계를 극복하기 위해 보조 제어로 Max Pooling, ReLU, Leaky ReLU를 적용하였다. 3축 Jerk 기록을 통해 다중 제어의 완만한 주행을 확인했고 편안한 승차감을 제공할 수 있게 되었다. PID-3중 필터 방식의 제어가 저사양 환경에 있어 적절한 승차감과 안정적인 주행을 제공함을 검증하였다.

5.2. 제언

5.2.1. 단순화된 승차감 평가

실제 차량의 승차감 평가는 Jerk만을 이용하지 않고 훨씬 더 다양한 기준[6]을 가진다. 게다가 본 모델 차량의 Jerk 계산은 좌석이나 몸무게에 따른 하중 변화 등을 고려하지 않고 단순히 전면부 3축 Jerk와 후면부 3축 Jerk 계산으로 이루어졌다. 주행에 초점을 맞춘 모델 차량이 가지는 한계 이외에도 다양한 기준을 적용해 다각적인 평가를 제공해야 한다.

5.2.2. 주 제어기 튜닝

주 제어기인 PID 제어기의 K_p, K_i, K_d의 튜닝은 표에 대입하는 방식인 Ziegler-Nichols Tuning을 이용한다. Ziegler-Nichols Tuning은 간단하고 쉽지만 여러 차례의 시험 운전이 필요하고 Overshooting 등의 치명적 문제를 인식하기까지 많은 시간이 소요된다. 운행할 때마다 하드웨어가 조금씩 바뀌는 모델 차량의 특성상 최종 튜닝 값이 미래의 주행에도 유효하리라 장담할 수 없다.

5.2.3. 제한적 도로 환경

모델 차량의 주행은 평면 타일 3종류에서만 이루어졌고 차단기와 신호등을 고려하지 않았다. 만약 차단기와 신호등이 포함된다면 제어 위계 설정에 있어 많은 변화가 있어야 할 것이고 평면 타일 또한 차선 인식을 단순화시킨다. 따라서 복합적 환경에서 유동적으로 제어할 수 있는 기술에 관한 연구가 필요하다.

참고문헌

[1] Jinghua Zhong ."PID Controller Tuning: A Short Tutorial", 2015-04-21

[2] Ziegler J.G & Nichols N. B.. "Optimum settings for automatic controllers". (1942). ASME. 64: 759–768.

[3] © 2020 The MathWorks Inc. "Design and implement PID controllers", 2018-5-22

[4] Brownlee, Jason . "A Gentle Introduction to the Rectified Linear Unit (ReLU)". (8 January 2019). Machine Learning Mastery. Retrieved 8 April 2021.

[5] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng (2014). Rectifier Nonlinearities Improve Neural Network Acoustic Models.

[6] www.ridetech.com. "What is Ride Quality?". Retrieved 10 April 2018.

06 자율 주행 자동차 안정적 주행 능력 알고리즘 구현

한양대학교 사범대학 부속 고등학교
강규리, 기영욱, 김민경, 성시연, 조서연

자율 주행 자동차 안정적 주행 능력 알고리즘 구현

Autonomous Driving Car Stable Driving Capability Algorithm Design

강 규 리, 기 영 욱, 김 민 경, 성 시 연, 조 서 연*
Gyuri Kang, Youngwook Ki, Minkyoung Kim, Siyeon Sung, Seoyeon Cho

요약

기존 자율 주행 알고리즘은 추세선을 하나만 잡아 주행을 하는 방식으로 진행되었다. 본 연구는 자율 주행 자동차의 안정적인 직선 주행을 위한 알고리즘을 구현하는 데 중점을 두었다. 이에 대한 해결책으로는 직선 주행 능력 개선을 위한 추세선 개수 변경 알고리즘을 구상했다. 기존과는 다르게 frontLines가 2 이상일 경우 추세선을 2개 잡도록 하는 방식의 알고리즘을 통해 자주차가 오차 없이 정확히 가운데 위치하고 있을 때의 x좌표를 구하기 어렵다는 문제점을 보완한다. 이 방법이 임의의 트랙에서 항상 성립하지는 않지만 이 방법을 통해 더 정확한 직선 주행을 할 수 있을 것이다.

Keywords: 자율 주행, 차선 인식, 추세선, 직진 주행, 후진, 곡선 주행

I. 서 론

기존 코드는 자동차와 상대적으로 더 가까운 쪽 차선 한 개만 이용했다. 주행 중에 특정 y좌표 값에 대응하는 실제 차선의 x좌표 값과 자주차가 정 가운데에 있을 때의 x좌표 값의 차이 'e'를 구했고, 조향 값(steer)을 e에 대한 일차식으로 정의했다.

그러나 자주차가 오차 없이 정확히 가운데에 있을 때의 x좌표를 구하기 어렵다는 단점이 있다. 그래서 차선이 두 개 이상 인식되는 경우에는 주 차선을 두 개를 이용하는 방식을 고안해보았다. len(frontLines)>=2일 경우에 차량과 가장 가까운 차선과 두 번째로 먼 차선의 특정 y좌표에 대응되는 x좌표를 구한 뒤 평균을 내어 middle로 정의한다. 그리고 모듈의 내장함수를 통해 구할 수 있는 화면상 정중앙의 x좌표 값(center_x)과 middle의 차를 구해 e에 저장하고, steer를 e에 대한 일차식으로 정의했다. 차선이 오직 한 개 인식되는 경우, 즉 len(frontLines)=1인 경우에는 기존 코드의 방식을 따르도록 했다.

* 강규리 (한대부고, riverfall1027@gmail.com)
기영욱 (한대부고, 0.wook03@gmail.com)
김민경 (한대부고, minkyoung2550@gmail.com)
성시연 (한대부고, siyeon5081@naver.com)
조서연 (한대부고, winniecho1202@naver.com)

II.주행 알고리즘

2.1. 변수 설정

```
class Planning(BasePlanning):
    def __init__(self, graphics):
        super().__init__(graphics)
        self.vars.redCnt = 0
        self.vars.greenCnt = 0
        self.vars.stop = True
        self.vars.steer = 0
        self.vars.velocity = 0
```

그림 1. 생성자를 이용한 변수 설정

속도(velocity), 조향 값(steer) 등 여러 변수를 다루기 위해서 생성자(__init__)을 이용해 초기 변수를 설정해준다. 클래스에서 인자를 사용하기 위해서는 첫 번째 인자를 self로 사용하는 것이 원칙이므로 각 변수 앞에 self를 붙여준다. 따라서 위 코드의 인자는 순서대로 빨강불 카운트 변수 설정, 녹색불 카운트 변수 설정, 차가 처음에 정지해 있음을 정의한 변수 설정, 조향 값은 0인 상태로 변수 설정, 속도가 0인 상태로 변수 설정을 의미한다.

```
def process(self, t, frontImage, rearImage, frontLidar, rearLidar):
```

그림 2. 생성자를 이용한 변수 설정

앞에서 설정한 변수를 사용하기 위해 자주차의 센서 정보(이미지, 라이다)를 바탕으로 조향과 속도를 결정하는 함수입니다. 각 변수의 뜻은 t: 주행 시점으로부터의 시간, frontImage: 전면 카메라 캘리(보정)된 이미지 (640X480), rearImage: 후면 카메라 캘리(보정)된 이미지, frontLidar: 전면 거리 센서 (mm), 0은 오류를 의미하며 최대거리는 2m, rearLidar: 후면 거리 센서 (mm).

```
frontLines, frontObject = self.processFront(frontImage)
rearLines= self.processRear(rearImage)
```

그림 3. 자주차의 센서 정보를 변수로 한 함수 설정

frontLines는 [[x1, y1],[x2, y2], …], [[x1,y1], [x2,y2], …]와 같이 좌표로 설정되며 차선을 인식하는 중요한 변수로 작용한다. frontlines[0]이 가장 왼쪽 차선으로 빨강(0), 주황(1), 노랑(2), 초록(3), 파랑(4), 남색(5), 보라(6)로 화면에 표시된다.

```
reds, greens = frontObject
```

그림 4. 신호등 인식 변수 설정

신호등을 인식할 변수는 frontObject의 변수로 취급되며 reds : n*3의 크기, numpy array의 형태로 ([[x1,y1,반지름], [x2,y2,반지름], …])으로 표시할 수 있다.

```
canny = self.canny(frontImage)
self.imshow('Canny Image', canny)
```

그림 5. 캐니 이미지 시각화

캐니 이미지(카메라가 인식한 물체를 선으로 표시한 이미지)를 화면에 나타날 수 있게 한 코드이다.

```
steer=self.vars.steer

x = 320
y = 479
while y >= 0:
    if canny[y, x] > 0:
        break
    y -= 1
```

그림 6. 조향값의 변수 설정과 이미지 좌표 분석

steer를 조정할 변수를 코드가 찾지 못했을 때 이전 변수를 사용하여 반복할 수 있게끔 이전 변수를 기억할 수 있게 하는 코드이다. x와 y는 frontImage에서 사용하는 변수로써 이미지의 크기로 변수를 설정해주고 조향 값을 설정해주기 위해 y값을 분석할 수 있도록 한다.

```
if 0 < frontLidar < 300 :
    velocity = 0

else:
    velocity = 30
```

그림 7. 전방 장애물 인식 코드

자주차의 앞에 있는 장애물을 인식하기 위한 frontLidar에 대한 코드이다. 300은 300mm를 의미하며 30cm 안에 장애물이 있을 경우 속도를 멈추게 한다. 그렇지 않을 경우 자주차의 속도는 일정하게 유지된다. 코드가 계속 반복되므로 앞에 장애물이 있는지 자주차가 계속 인식하며 주행한다.

```
center_x = mtx[0, 2]

middle=center_x
frontLines.sort(key=lambda x:x[0, 1], reverse=True)
rearLines.sort(key=lambda x:x[0, 1],reverse=True)
```

그림 8. 중간 지점 변수 설정과 내림차수 정렬 코드

frontLines를 x[0,1]을 기준으로 내림차순으로 정렬해주는 코드이다. 자주차가 차선을 인식하기 위해서는 가장 밑 좌표부터 인식하기 때문에 frontLines를 내림차순으로 정렬해준다. 화면상에서도 가장 밑에 점부터 표시하게 된다.

```
laneImage = frontImage.copy()
rearlaneImage = rearImage.copy()
```

그림 9. 추세선의 시각화

전면 카메라에 잡힌 차선 수가 한 개 이상이면 frontLines의 첫 번째 요소(점들의 집합)의 x좌표를 x_1에 y좌표를 y_1에 저장하고 정의역이 x_1, 치역이 y_1인 일차식과 정의역이 y_1이고 치역이 x_1인 일차식을 세운 뒤, 후자를 line_1로 정의하고 이를 전면 카메라 화면에 시각화하였다.

```
if len(frontLines)>=1:
    if frontLines[0][0, 1] > 380:

        x_1 = frontLines[0][:, 0]
        y_1 = frontLines[0][:, 1]
        coefficient_1 = np.polyfit(x_1, y_1, 1)
        coefficient = np.polyfit(y_1, x_1, 1)
        line_1 = np.poly1d(coefficient)

        for i in range(300, 480, 1):
            cv2.circle(laneImage, ((int(line_1(i)), i), 5, (255, 0, 0), -1)
```

그림 10. 차선이 한 개 이상일 때의 함수와 그에 따른 좌표

전면 카메라에 인식된 차선 수가 1개 이상일 때의 코드이다. 차선이 y=380보다 아래에 위치해 있을 때 코드가 실행되며 coefficient가 polyfit함수로 작용해 앞서 설정한 일차식에 따라서 자주차가 인식하고 있는 차선의 좌표를 알고 시각화할 수 있다. line은 ay+b의 일차식으로 표현되며 가장 밑에 있는 차선의 추세식을 의미한다.

2.2. 차량이 전진하는 상황에서의 주행 알고리즘 설계

차량이 전진하는 상황에 대해 직선과 곡선주행을 따로 구별하지 않고 하나의 알고리즘으로 전부 처리할 수 있도록 코드를 작성했다. 다만 카메라에 잡히는 차선이 없는 경우, 한 개인 경우, 두 개인 경우를 나누어 조향 값을 구하는 방식을 조금씩 달리하였다.

먼저, 코드에 사용되는 두 변수를 초기화시켰다. middle은 차선이 두 개 인식되는 경우 두 차선의 가운데 지점의 x좌표가 저장될 변수이며, 오류발생을 막기 위해 화면상의 가운데지점으로 초기화해두었다. n은 차선의 추세식에 대입하는 상수를 하나로 통일하고, 이를 한 번에 수정하기 편리하도록 설정해놓은 변수이다.

```
middle=center_x
n=300
```

그림11. 주행 알고리즘에 사용되는 변수 초기화

전면 카메라에 인식된 차선 수가 한 개 이상일 경우, 첫 번째 차선의 추세선을 구하는 코드이다. frontLines의 첫 번째 요소의 x좌표를 x_1에, y좌표를 y_1에 저장한 후 정의역이 y_1이고 치역이 x_1인 일차식을 세워 line_1으로 정의하고 이를 전면 카메라 화면에 시각화시켰다.

```
if len(frontLines)>=1:
    if frontLines[0][0, 1] > 380:

        x_1 = frontLines[0][:, 0]
        y_1 = frontLines[0][:, 1]
        coefficient_1 = np.polyfit(x_1, y_1, 1)
        coefficient = np.polyfit(y_1, x_1, 1)
        line_1 = np.poly1d(coefficient)

        for i in range(300, 480, 1):
            cv2.circle(laneImage, (int(line_1(i)), i), 5, (255, 0, 0), -1)
```

그림 12. 전면 카메라에 인식된 차선이 1개 이상인 경우

전면 카메라에 잡힌 차선 수가 두 개 이상일 경우, 두 번째 차선의 추세선을 구하는 코드이다. frontLines의 두 번째 요소의 x좌표들을 x_2에, y좌표를 y_2에 저장한 후, 정의역이 y_2이고 치역이 x_2인 일차식을 세워 line_2로 정의한다. middle은 y좌표가 n이라는 특정 값일 때의 첫 번째 차선의 x좌표와 두 번째 차선의 x좌표의 평균값이 저장된다. 이를 이용하여 e를 middle과 화면상의 가운데 지점의 x좌표의 차로 정의하고, 조향값 steer를 e에 대한 일차식으로 정의하였다. 그리고 line_2를 전면 카메라 화면에 시각화시켰다.

```
if len(frontLines)>=2:
    if frontLines[1][0, 1] > 380:
        x_2 = frontLines[1][:, 0]
        y_2 = frontLines[1][:, 1]
        coefficient_2 = np.polyfit(x_2, y_2, 1)
        coefficient = np.polyfit(y_2, x_2, 1)
        line_2 = np.poly1d(coefficient)
        middle=(line_1(200)+line_2(200))/2

        e=middle-center_x
        steer = 0.5*e

        self.vars.steer = steer


        for i in range(300, 480, 1):
            cv2.circle(laneImage, (int(line_2(i)), i), 5, (255, 0, 0),  1)
```

그림 13. 전면 카메라에 인식된 차선이 2개 이상인 경우

전면 카메라에 잡힌 차선 수가 오직 하나인 경우에는 기존 코드에서 추세선 1개를 잡아 주행하던 방식을 거의 그대로 적용했다. 다만, 기존 코드를 실행시켜 보았을 때 i가 0인 경우에 대해서만 for문이 실행되고, 'break'에 의해 곧바로 중단되어버리는 것을 확인할 수 있었기 때문에 for문을 사용하는 것이 불필요하다고 판단했다. 따라서 if문만을 이용하여 기존 코드에 비해 간략하게 코드를 작성해보았다.

```
if len(frontLines)==1:
    x_1 = frontLines[0][:, 0]
    y_1 = frontLines[0][:, 1]
    coefficient_1 = np.polyfit(x_1, y_1, 1)
    coefficient = np.polyfit(y_1, x_1, 1)
    line_1 = np.poly1d(coefficient)


    if line_1(400)<center_x:
        direction='left'
        e=line_1(400)-36
    else:
        direction='right'
        e=line_1(400)-center_x-218

    steer = 0.5 * e
    self.vars.steer = steer
```

그림 14. 전면 카메라에 1개의 차선만이 인식된 경우

자율 주행 자동차가 운행 도중 이탈하거나 시스템 자체의 문제로 인해 차선이 잡히지 않을 때를 고려해 인식된 차선이 없는 경우에는 이전의 조향 값과 속도를 그대로 유지하도록 하였다.

```
velocity=30

if len(frontLines)==0:
    print('No line found')
    return self.vars.steer, self.vars.velocity
```

그림 16. 전면 카메라에 인식된 차선이 없을 경우

2.3.후진 주행 알고리즘 사전 작업

기존 코드에서 활용되지 않았던 후면 카메라 정보를 이용하여 후진 주행을 구현할 수 있도록 사전 작업을 해두었다. 차량이 전진하는 상황에서의 코드와 동일한 방식으로 진행된다. 후면 카메라에 인식된 차선이 1개 이상인 경우에 첫 번째 차선의 추세선을 구해 후면 카메라 화면에 시각화시키고, 차선이 2개 이상인 경우에는 두 번째 차선의 추세선까지 구하여 시각화시켰다. 또한 차량이 전진하는 상황에서 조향 값을 구하는데 사용되었던 픽셀 값 e와 같은 역할을 하는 변수로서 d를 정의하였다. 본 연구에서는 주행 알고리즘에 후진 주행을 활용하지 않았지만, 후진 주행을 추가하게 된다면 d를 이용하여 조향 값을 구할 수 있을 것이다.

```
if len(rearLines)>=1:
    if rearlines[0][0, 1] > 380:
        r_x_1=rearLines[0][:, 0]
        r_y_1 = rearLines[0][:, 1]
        #r_coefficient_1 = np.polyfit(r_x_1, r_y_1, 1)
        r_coefficient = np.polyfit(r_y_1, r_x_1, 1)
        rearline_1 = np.poly1d(r_coefficient)
        if len(rearLines)==1:
            if rearline_1(n)<center_x:
                d=rearline_1(n)-36
            else:
                d=rearline_1(n)-center_x-218

        for i in range(300,480,1):
            cv2.circle(rearlaneImage, (int(rearline_1(i)), i), 5, (255, 0, 0), -1)

    if len(rearLines)>=2:
        if rearlines[1][0, 1] > 380:
            r_x_2=rearLines[0][:,0]
            r_y_2 = rearlines[0][:, 1]
            #r_coefficient_2 = np.polyfit(r_x_2, r_y_2, 1)
            r_coefficient = np.polyfit(r_y_2, r_x_2, 1)
            rearline_2 = np.poly1d(r_coefficient)
            rear_middle=(rearline_1(n)+rearline_2(n))/2
            d = rear_middle-center_x

            for i in range(300, 480, 1):
                cv2.circle(rearlaneImage, (int(rearline_2(i)), i), 5, (255, 0, 0), -1)
```

그림 17. 후진 주행 사전 작업

III. 결론 및 제언

본 연구에서는 2개의 추세선을 이용한 차선 인식, 후면 차선 인식 알고리즘을 제안하였다. 이는 보다 안정적인 직진 주행을 위해 제안되었으며 제안된 알고리즘은 자주차에 적용되어 시험을 위해 제작된 모형 도로에서의 주행을 완료하여 알고리즘의 유효성을 검증하였다.

3.1. 곡선 주행의 한계

현재 코딩된 자주차는 부드러운 곡선만을 주행할 수 있다. 급격한 커브를 주행할 시 추세선 인식이 안 되는 경우가 발생할 수 있어 차선을 벗어날 가능성이 매우 커진다. 따라서 급격한 커브에 대비한 코드 연구가 필요하다.

3.2. 차선 인식

차선의 중간 값을 찾기 위해서는 정확한 픽셀을 아는 것이 중요하다. 다만, 이 픽셀 값을 정확하게 알기 어렵고 주행 중에도 계속해서 변수가 변할 수 있다. 실제 자율 주행차는 평면 위의 차선뿐만 아니라 다양한 차선을 주행하기 때문에 차선 인식에 관한 더 정밀한 연구가 필요하다.

참고문헌

[1] 김평원, 정형식, 홍범진, 함께 만드는 인공지능 자주차, 인천대학교출판부, 2019.

[2] Shuang Wu, Jie Tang, Liyun Li, Shaoshan Liu, Jean-Luc Gaudiot, “Creating Autonomous Vehicle Systems”, 2017

[3] 문주현, 이상규, “자율 주행 교육용 모형 자동차의 인식 및 주행 능력 개선”, 청년공학, 제4집, pp. 7-13, 2020.

07 자주차 직각 회전 구간 알고리즘에 대한 연구

선덕고등학교 김건우

자주차 직각 회전 구간 알고리즘에 대한 연구

Algorithm for Right Angle Rotation

김 건 우* Kim Geon Woo

요약

현재 자율 주행 자동차는 상용화가 되었지만, 고도 자동화 이상의 자율 주행 자동차는 아직 많은 연구가 필요하다. 특히, 회전 구간에서의 알고리즘에 관한 연구가 중요한데 회전 구간의 경우 고려해야 할 요소가 많기 때문이다. 이 연구를 통해서 회전 가능 각도의 제한, 회전 공간 등의 주행 시 발생할 수 있는 문제 상황을 파악해보고 그 원인을 분석하고자 한다. 이를 해결하기 위해 보조선을 이용해서 자율 주행 자동차의 회전 구간 주행 과정에서 상황을 판단하고 상황에 따라 조향값을 산출해내는 방법을 고안해보고자 한다. 또한, 후진 등 알고리즘과 후면 카메라 등의 하드웨어를 통해서 기존의 한계점을 보완한 새로운 직선 회전 알고리즘을 제안한다.

Keywords: 자율 주행 자동차, 직각 회전 구간, 보조선, 후진, 후면 카메라

I. 서 론

현재 가장 상용화된 4차 산업 혁명의 기술은 자율 주행 자동차이다. 하지만 아직 운전자가 시스템에 개입하지 않는 고도 자동화 단계 이상의 자동차는 아직 개발 중이며 앞으로 많은 연구가 필요하다. 특히 회전 구간에서의 알고리즘에 대한 개발이 중요하다. 2016년 교통사고 관련 보고서를 보면 우, 좌회전 상황에서 직진 상황보다 사고 발생률이 더 높게 나타나는 등 실제 인간의 주행에서도 어려운 것이 회전 구간이기 때문이다. 그렇다면 회전 구간에서의 주행이 어려운 이유는 무엇일까? 그것은 회전 구간의 경우 직진 구간과 다르게 회전각이나 자동차의 진입 속도 등 다양한 변수를 고려해야 하기 때문이다. 그렇기에 자율 주행 알고리즘 역시 여러 가지 상황에 대응할 수 있도록 제작되어야 한다. 그래서 이 연구를 통해 한림공학원에서 제공한 자율 주행 자동차 주행 예제 소스 파일이 가진 회전 구간에서의 문제점을 분석하고 다양한 회전 구간을 통과할 수 있는 효율적인 회전 알고리즘을 제작하고자 한다.

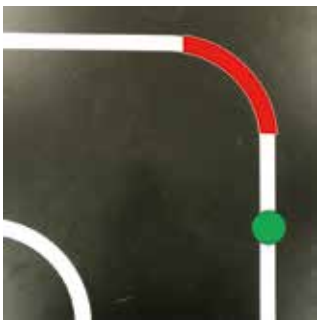
II. 본론

2.1. 문제 인식

자율 주행 자동차 예제 소스 코드에서 회전할 때 바깥쪽 차선을 기본차선으로 인식한다. 그리고 특정한 가로선을 그리고 여기서 생기는 접점을 지나면서 기본차선에 접하는 추세선의 기울기에 따라 자동차의 조향 값을 정하고 그에 맞춰서 회전한다.

일반적인 회전 구간의 경우 기울기가 구간의 처음부터 생기기 때문에 문제없이 회전이 가능하다. 그러나 90도 이상의 회전 구간(이하 직각 회전 구간)에서는 구간 시작 부분이 직선이기 때문에 구간 중간, 곡선이 생길 때부터 회전하게 된다.

[그림 1] 일반 곡선 구간에서의 회전 시작점



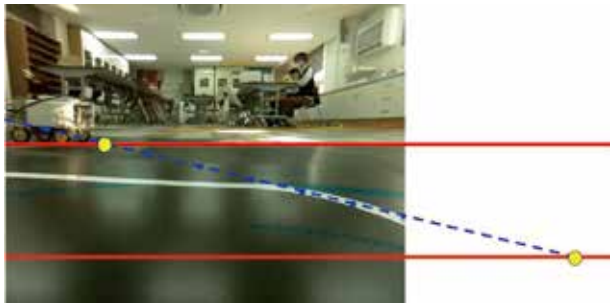
[그림 2] 직각 곡선 구간에서의 회전 시작점

* 김건우 (선덕고등학교, 4947x@naver.com)

여기서 발생하는 문제점은 회전 시작이 늦어지고 회전각과 회전을 위한 공간이 작아지는 것이다. 이렇게 되면 추세선의 기울기 변화량이 줄어들어서 통과할 만큼의 회전 각도를 설정할 수 없어진다. 또한, 회전 과정에서 차선이 카메라 시야 범위 밖으로 벗어나면서 차량이 차선을 잡지 못하고 이탈하게 된다.

2.2. 보조선을 이용한 알고리즘

위 문제를 해결하기 위해서 자율 주행 자동차가 일반 회전 구간과 직선 회전 구간을 판단하고 직선 회전 구간일 경우 조향값에 상수를 더해줘서 더 큰 각도를 회전할 수 있도록 한다. 회전 상황을 판단하기 위해서 기존의 가로선보다 위에 새로운 가로선을 추가하고 추세선과의 교점의 x좌표를 구한다. 기존 가로선의 추세선과의 교점과 새로 구한 교점 사이의 x좌표 차이를 이용해서 상황을 판단한다. 회전 구간을 지날수록 추세선의 기울기의 절댓값이 작아지게 되는데 이 과정에서 직각 회전 구간의 경우 차체가 정면을 향한 상태이기 때문에 차체가 회전 상태인 일반 회전 구간보다 x좌표 차이가 커지게 된다.



[그림 3] 직각 곡선 구간에서의 보조선

[그림 4] 일반 곡선 구간에서의 보조선

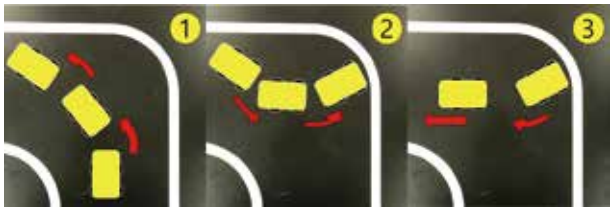
```
else:
    # follow left
    line1 = frontLines[line1[1]]
    x = line1[:, 0]
    y = line1[:, 1]
    coefficient = np.polyfit(y, x, 1)
    line1 = np.poly1d(coefficient)
    if line1(380) - line1(240) < -20 :
        e = line1(380) - center_x + 380
    else:
        e = line1(380) - center_x + 317
    print('frontleftline(240)', line1(240))
    print('bbaegi', line1(380)-line1(240))
    print('center_x=', center_x, 'e=', e)
    for i in range(200, 480, 20):
        cv2.circle(laneImage1, (int(line1(i)), i), 5, (255, 0, 0), -1)
    # cv2.circle(frontImage, (int(f(400)), 400), 5, (255, 0, 0), -1)
    cv2.imshow('Front Lane Image', laneImage1)
    cv2.imshow('Rear Lane Image', laneImage2)
```

[그림 5] 좌회전 시 보조선 알고리즘

반복적인 주행을 통해서 각각의 상황의 x좌표 차이의 사잇값을 구하고 이를 기준으로 구간을 판단한다. 여기서 직선 회전일 경우 회전각에 보정값을 더해줘서 더 크게 회전시킨다.

2.3. 후진 알고리즘

자율 주행 자동차 키트는 회전 가능 각도가 -80도에서 80도로 정해져 있어서 직각 회전 구간처럼 좁은 공간에서 큰 회전을 요구하는 경우 어려움이 있다. 이를 해결하기 위해서 T자형 주차에서 사용되는 후진을 이용한다. 만약 절댓값 80도 이상의 회전값이 산출될 경우 방향은 기존 조향 방향의 반대로 설정하고 조향값은 70으로 고정한다. 그리고 velocity 값을 음수로 설정하여 기존 움직임과 반대로 작동하게 해서 회전을 위한 공간과 회전 각도를 확보한다.



[그림 6] 후진 알고리즘의 움직임

```
if steer > 80  :
    steer = -70
    velocity = -20

elif steer < -80 :
    steer = 70
    velocity = -20

else :
    velocity = 40
```

[그림 7] 후진 알고리즘

2.4. 후면 카메라를 이용한 알고리즘

보조선 알고리즘을 이용해서 회전하는 과정에서도 순간적으로 차선이 카메라 시야 밖으로 벗어나면서 차량이 차선을 이탈하는 경우가 발생한다. 이에 대한 해결책으로 후면 카메라를 사용한다. 예제 소스의 전면 카메라 코드를 바탕으로 후면 카메라 코드를 작성한다. 이때 후면과 전면에서 보이는 기본차선의 방향은 반대가 되기 때문에 후면 카메라에서 차량의 위치를 판단하는 코드는 전면 카메라와 반대로 설정한다.

그리고 전면 카메라에 선이 잡히지 않으면 후면 카메라를 이용해서 차량 뒤쪽 차선을 통해서 주행 환경을 판단하고 계속 회전을 이어가도록 한다.

```
for k in range(len(rearLines)):
    if rearLines[k][0, 1] < 310:
        continue
    x = rearLines[k][:, 0]
    y = rearLines[k][:, 1]
    coefficient = np.polyfit(y, x, 1)
    line2 = np.poly1d(coefficient)
    if line2(380) > center_x:
        line2 = 'left', k
        break
    else:
        line2 = 'right', k
        break
# print('line', line)
laneImage2 = rearImage.copy()
```

[그림 8] 후면 카메라 사용

```
if line1 is None:  # 앞line 이 없으면
    if line2 is None:  # 뒷line 이 없으면
        # No line
        print('No line found')
        return self.vars.steer, self.vars.velocity

    elif line2[0] == 'right':   # line이 우차선이면
        line2 = rearLines[line2[1]]
        x = line2[:, 0]
        y = line2[:, 1]
        coefficient = np.polyfit(y, x, 1)
        line2 = np.poly1d(coefficient)
        e = line2(380) - center_x - 220
        print('rearrightline(380)', line2(380))
        print('center_x=', center_x, 'e=', e)
        for k in range(200, 480, 20):
            cv2.circle(laneImage2, (int(line2(k)), k), 5, (255, 0, 0), -1)
        cv2.imshow('Rear Lane Image', laneImage2)

    else:  #line이 좌차선이면
        line2 = rearLines[line2[1]]
        x = line2[:, 0]
        y = line2[:, 1]
        coefficient = np.polyfit(y, x, 1)
        line2 = np.poly1d(coefficient)
        e = line2(380) - center_x + 317
        print('rearleftline(380)', line2(380))
        print('center_x=', center_x, 'e=', e)
        for k in range(200, 480, 20):
            cv2.circle(laneImage2, (int(line2(k)), k), 5, (255, 0, 0), -1)
        cv2.imshow('Rear Lane Image', laneImage2)
```

[그림 9] 후면 카메라를 통한 주행 알고리즘

Ⅲ. 결론

3.1. 결론

이 연구를 통해서 일반 회전 구간과 직각 회전 구간을 보조선을 이용해 구별하고 더 큰 회전을 위한 보정값을 더해서 구간을 통과할 수 있게 하였다. 또한, 회전 구간의 크기와 회전 가능 범위라는 한계점을 후진 알고리즘을 이용해서 해결할 수 있었다. 후면 카메라를 이용해서 차선을 인식하지 못하는 상황에서 주행을 계속할 수 있도록 하였다.

회전 구간에서 발생하는 다양한 문제 상황을 확인하고 이에 대한 해결책을 제시하는 등 회전 알고리즘에 대한 새로운 의견을 제시했다는 의의가 있다.

3.2. 한계 및 제언

보조선을 이용한 알고리즘에서는 회전 구간 시작 부분에서의 진입을 어떻게 하냐에 따라서 직진을 하는 등 오류가 발생하는 경우가 발생했다. 이를 해결하기 위해서 정확한 기준값을 찾을 필요가 있다.

후진 알고리즘의 경우 실제 자동차에도 적용되는 구조적 문제에 대해서 생각해볼 수 있었다는 의의가 있지만, 실제 주행 환경에서 회전 시 후진할 수 없기 때문에 현실적인 적용은 불가능하다는 한계점이 있다.

또한, 후진 알고리즘과 후면 카메라를 동시에 이용했을 때 카메라에 선이 잡혔다가 안 잡히는 순간에 회전각이 80도 이상일 경우 두 코드가 충돌해서 직진과 후진이 반복되는 오류가 발생했다. 이는 후면 카메라의 코딩에서 기본차선을 설정할 때 좌측부터 탐색하는 코딩을 수정하지 못해서 발생한 것이라고 판단되며 이를 해결하기 위해 우측부터 탐색하는 코딩을 제작해서 실험해 볼 필요가 있다.

참고문헌

[1] 배광수, 최진욱, 안휘근, 『교차로 신호위반사고 요인분석 및 교통안전 개선방안 수립 연구, 도로교통공단 교통과학연구원(2016), 53쪽.

[2] 허준호, 이선봉. (2016). 카메라 위치를 반영한 후진 주차 가이드라인 생성 연구. 한국산학기술학회 논문지, 17(3), 591-598.

08 차량 주행상황에 따른 안정적 조향 값 산출을 위한 유동적 시스템 및 역함수 기반 알고리즘

하나고등학교
김준수, 김윤동, 이용욱, 장진영, 강민지, 강승연, 방유빈, 안형준, 황정우

차량 주행상황에 따른 안정적 조향 값 산출을 위한 유동적 시스템 및 역함수 기반 알고리즘

Adaptable system and inverse function-based algorithm for calculating stable steering value according to vehicle driving conditions

김준수, 김윤동, 이용욱, 장진영, 강민지, 강승언, 방유빈, 안형준, 황정우*
Junsu Kim*, Yoondong Kim, Yonguk Lee, Jinyoung Jang, Minji Kang, Seungeon Kang, Yubin Bang, Hyungjun An, Jeongwoo Hwang

요약

기존의 자율 주행 차량의 알고리즘 시스템은 하나의 정교한 알고리즘이 모든 주행상황을 커버할 수 있도록 하는 방식이었다. 하지만 이러한 방식은 주행상황에 따라 돌발 상황이 나오거나, 적절하지 못한 조향이 나오는 상황에 대한 대처가 쉽지 않은 단점이 있다. 본 연구에서는 이러한 단점을 보완하여 주행상황을 차량이 구분하고 이에 따라 상황별로 최적화된 알고리즘을 적용하여 주행하는 시스템을 고안한다. 이러한 시스템에서 집약된 알고리즘에서는 어레이 데이터 처리와 변수 설정 방법 등을 통해 유용한 데이터를 추출하는 방법을 활용하는 방식을 고안하였다. 또한 각 주행상황별로 적용되는 알고리즘에서 좌표계와 역삼각함수 활용 기법을 활용하여 상황에 맞는 데이터 처리 방법을 선택할 수 있는 유동적 주행 알고리즘 시스템을 고안하여 상황에 최적화된 차량의 조향 값이 산출될 수 있도록 설계하였다.

Keywords: 자율 주행 자동차, 유동적 주행 시스템, 데이터 선별, 추세선, 역함수

I. 서 론

4차 산업혁명은 그 어느 때보다 우리의 삶을 광범위한 영역에서 빠르게 바꿔놓고 있다. 그중에서도, 기존의 자동차 기술에 인공지능과 사물 인터넷을 접목한 자율 주행 자동차는 4차 산업 혁명의 핵심 기술로 주목받고 있다. 테슬라의 CEO 일론 머스크는 올해 말까지 레벨 5의 완전 자율 주행 기술을 완성하겠다고 발표하였으며, 혼다자동차는 올해 3월경부터 세계 최초로 레벨 3 자율 주행 자동차를 대량으로 생산하고 있다. 이렇듯 자율 주행 자동차에 관한 연구는 세계적으로 계속되고 있다. 이러한 연구를 하는 과정에서 이미지 데이터를 처리하는 방법에 개선이 필요하다고 판단하였다. 기존에 사용하던 이미지 데이터는 범위가 좁고 상대적으로 정보량이 적은 대신 처리가 쉬웠지만 새로운 이미지 데이터는 범위가 넓어진 만큼 불필요한 정보가 포함된다는 문제점을 가지고 있었다. 향상된 이미지 데이터를 효과적으로 활용하기 위해서는 데이터의 정제가 필요했고 본 연구에서는 어레이 데이터 처리와 변수 설정 방법 등을 통해 유용한 데이터를 추출하는 방법을 고안해내었다. 이미지 처리 방법 이외에도 알고리즘 자체에서도 문제점을 찾을 수 있었다. 기존에 사용하던 알고리즘은 한 가지 데이터 처리 방법을 직진, 좌회전, 우회전 상황에 모두 적용하여 주행하고자 하였다. 이 방식은 데이터 처리 방법을 하나로 고정하여 비교적 간단하게 이해하고 수정할 수 있다는 장점은 있지만, 직진과 좌우 회전 상황은 서로 다른 데이터 처리와 상황 판단이 필요하기에 안정한 주행이 불가능하다는 문제점을 가지고 있어 상황에 대해 다소 경직되어있는 알고리즘이라고 할 수 있다. 따라서 본 연구에서는 좌표계와 역삼각함수 활용 기법을 활용하여 상황에 맞는 데이터 처리 방법을 선택할 수 있는 유동적 주행 알고리즘 시스템을 고안해내었다. 이 알고리즘에서는 이미지 데이터를 토대로 차의 주행 상태를 설정하여 상태에 맞는 알고리즘을 활용할 수 있도록 설계하였다.

* 김준수 (하나고등학교, junsu01041526634@gmail.com)
김윤동 (하나고등학교, marchello.kim@gmail.com)
강민지 (하나고등학교, sally5180@naver.com)
이용욱 (하나고등학교, dldyddnr4285@gmail.com)
장진영 (하나고등학교, whdmsql2014@naver.com)
강승언 (하나고등학교, chm6206@naver.com)
방유빈 (하나고등학교, bubin040612@naver.com)
안형준 (하나고등학교, ahyungjun0913@naver.com)
황정우 (하나고등학교, timothy20040503@gmail.com)

II. 이미지 데이터 선별 시스템

2.1. 어레이 데이터 처리

자율 주행차의 카메라에서 들어오는 정보는 frontImage에 numpy.ndarray의 class로 이미지 데이터 처리되어 저장된다. 자율 주행차에서 들어오는 이미지 데이터는 480*640 크기로 되어있으며 각각의 픽셀을 [y, x, c], (x, y)의 좌표의 color(0~255)로 표현한다. 이 값을 numpy.array를 사용해서 행렬의 형태로 frontImage에 정보를 저장한다.

2.2. 데이터 처리

frontImage에 저장된 데이터를 사용하기에는 불필요한 정보들이 많다. 자율 주행차 알고리즘은 차선과 관련된 이미지 데이터만을 필요로 하므로 frontImage에서 차선 정보를 추출해야 한다. 그림1과 같이 트랙은 검은색을 바탕으로 흰색으로 차선이 그려져 있다.

```
<class 'numpy.ndarray'>
[[[203 223 199]
  [202 222 199]
  [205 226 203]
  ...
  [119 133  96]
  [119 133  97]
  [119 133  97]]
```

그림 1. 어레이 데이터

그림 2. 카메라 이미지

그림2가 이를 이용해서 frontLines 데이터를 추출할 것이다. opencv의 canny image를 사용해 색의 변화가 급격한 부분을 표시해주면 그림1과 같은 형태가 나온다.

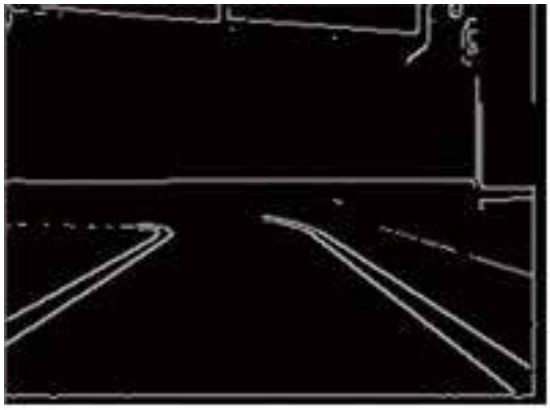


그림 3. Canny Image

3. 데이터 변환 및 선별

canny image에 하얀 점의 좌표를 다시 array를 사용해 데이터를 변환해주고 데이터들을 frontLines에 list 형태로 저장한다. 저장된 데이터는 아래의 모습과 같다.

```
<class 'list'>
[array([[ 42.409416, 369.1334  ],
        [ 78.00247 , 348.16974 ],
        [104.2554  , 332.5585  ],
        [125.27841 , 320.4819  ],
        [141.1727  , 310.86154 ],
        [153.94905 , 303.01727 ],
        [164.28882 , 296.49878 ],
        [173.13095 , 290.99615 ],
        [180.85747 , 286.2891  ],
        [186.42883 , 282.21667 ],
        [190.6973  , 278.6587  ],
        [187.56165 , 275.52344 ],
        [182.93068 , 272.73978 ],
        [177.54802 , 270.25174 ],
        [168.51544 , 268.01462 ],
        [155.49426 , 265.99225 ]], dtype=float32), array([[545.3697 ,
        398.77164],
```

그림 4. 저장된 어레이 데이터

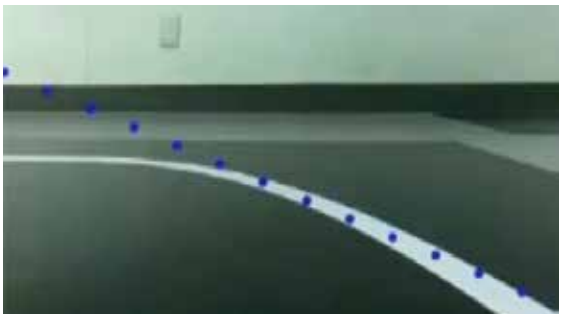


그림 5. 이미지상에 표현된 점

그림 4와 같은 데이터는 이미지상에서 가상의 x, y축을 기준의 좌표평면에서 그림 5과 같이 점으로 표시된다. 본 연구에서는 이처럼 수집된 데이터를 두 단계의 과정을 통해 선별하도록 하는 알고리즘을 설계하였다. 첫 번째 과정은 인식된 여러 개의 차선은 좌 차선과 우 차선으로 분류하는 것이다. 중앙 좌표를 기준으로 frontline 어레이 데이터에서 가장 상단의 점 좌표가 중앙보다 왼쪽에 위치하면 좌 차선, 오른쪽에 위치하면 우 차선으로 분류하도록 설계하였다. 이처럼 분류된 차선들은 다시 차선 데이터별 길이를 비교하여 가장 데이터 수가 많은, 즉 인식된 점의 수가 가장 많은 차선을 선별하도록 하였다. 이와 관련된 내용은 III. 유동적 주행 시스템에서 자세히 다루도록 하겠다.

III. 유동적 주행 시스템

차량을 의도한 대로 주행하도록 하는 알고리즘의 구현을 위해서는 알고리즘의 구동 방식을 표현한 플로우 차트의 설계가 필수적이다. 본 연구에서 의도한 주행 시스템은 차량이 주행상황을 직진, 좌회전, 우회전 및 차선 인식실패라는 주행상황을 분류하여 판단한 수, 그 상황에 최적화된 알고리즘을 알맞게 적용할 수 있도록 하는 시스템이다.

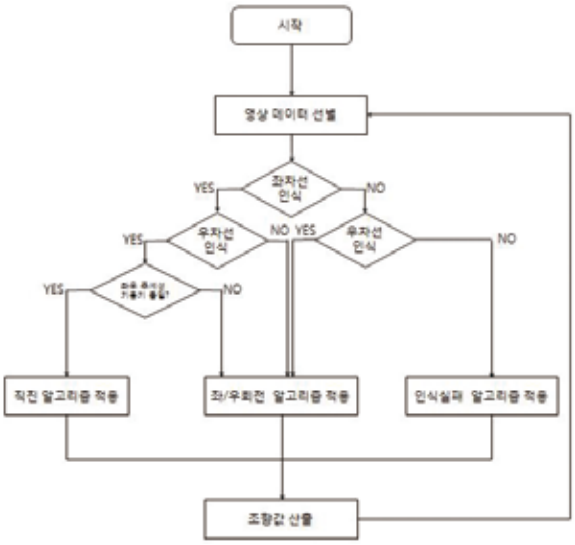


그림 6. 주행 플로우 차트

따라서 플로우 차트로 표현한 본 연구에서의 알고리즘 설계는 그림6과 같다. 전체적인 틀은 차량이 받아들인 카메라 프레임 데이터를 하나씩 연속하여 처리하는, 즉 하나의 Loop를 설계하였다. 주행 상황별로 최적화된 알고리즘을 각 상황에 적용하여 조향 값을 산출하고, 다시 데이터로 회귀하는 Loop이다. 실제 코드에서 활용한 변수는 [표 1]과 같다.

표 1. Parameter setting을 위해 활용한 변수

변수	정의
t	자율 주행차의 주행 시작으로부터의 시간 (초)
frontImage	전면 카메라 이미지(480*640*3)
[y, x, c]	(x, y) 좌표의 color (0~255) 정보
rearImage	후면 카메라 이미지
frontLidar	전면 거리 센서(mm) 0은 오류를 의미함
rearLidar	후면 거리 센서(mm) 0은 오류를 의미함
frontLines	[[x1, y1],[x2,y2]….], [[x1, y1],[x2,y2]….] (y1 > y2 > y3 > ...) frontLines[0]이 가장 안쪽 차선, 빨(0), 주(1), ... , 보(6)
Llinelist	왼쪽 인식선 선별 리스트
Rlinelist	오른쪽 인식선 선별 리스트
Lline	선별된 왼쪽 인식선
Rline	선별된 오른쪽 인식선
L	최종 선별 왼쪽 인식선 번호
R	최종 선별 오른쪽 인식선 번호
K	조향각 조정 상수
Status	자동차 주행 상태
LDiff	왼쪽 차선 추세선
RDiff	오른쪽 차선 추세선
TanA	왼쪽 추세선 기울기
TanB	오른쪽 추세선 기울기
ThetaA	왼쪽 추세선 각도
ThetaB	오른쪽 추세선 각도
Coeff	steer 조절 상수

3.1 Driving Situation: 주행 상황 판단

자율 주행차는 전면 카메라와 후면 카메라를 이용하여 이미지 처리를 진행한다. 우리는 전면 카메라의 이미지 처리를 중심으로 하였다. 전면 카메라 이미지는 640*480 이미지를 캘리브레이션 한 것으로 (x, y) 좌표의 색 (0~255) 정보를 포함하면 480*640*3 [y, x, c]로 생각할 수 있다. 자율 주행차 실험 시에 나타날 수 있는 주행상황은 기본적인 좌/우 차선의 판단, 직선 주행, 좌/우회전의 판단이 있다. 전면 카메라의 이미지를 통해 Llinelist, Rlinelist를 설정할 수 있고, 이는 주행 상황 판단에 사용된다. 카메라 인식선 선별 리스트이므로 각각 리스트의 값에 따라 기준을 설정해서 특정 조건을 만족시키면 특정한 주행상황으로 판단하도록 하는 것이다.

```
if len(Llinelist) == 0 and len(Rlinelist) == 0:
    Status = "인식실패"
if len(Llinelist) == 0 and len(Rlinelist) != 0:
    Status = "좌회전"
if len(Llinelist) != 0 and len(Rlinelist) == 0:
    Status = "우회전"
```

그림 7. 주행상황 분류 코딩

3.1.1. Llinelist, Rlinelist가 모두 0

Llinelist와 Rlinelist가 모두 0이면 자율 주행차는 판단의 근거가 없으므로 주행상황을 인식실패로 정한다. 인식실패 상황은 트랙이 없거나 식별되지 않는 상황뿐만 아니라, 회전하거나 카메라 화각에서 벗어날 때도 인식실패 상황이 되므로 여타 알고리즘을 통해 이전 명령을 지속하도록 처리하게 된다.

3.1.2. Llinelist가 0 Rlinelist가 0이 아님

Llinelist의 수가 0이고 Rlinelist의 수가 0이 아니면 좌회전으로 판단하는 것이다. 이러한 기준으로 세운 이유는 다음과 같은데, 자율 주행차의 카메라는 광각이 아니므로 회전 시에 회전 중심이 가까운 차선이 카메라에 인식이 안 되는 경우가 많다. 간혹 회전 중 모두 인식이 되는 경우가 있지만, 인식 여부에 따른 단순한 주행 상황 판단을 하는 것이 아니라 회전 중 인식이 안 되면 부정확한 주행을 초래하므로 좌회전 상태로 판단하여 다른 알고리즘에서 이용하여 회전 주행에 도움이 되도록 한다.

4.1.3. Llinelist가 0이 아님, Rlinelist가 0

이번에는 반대의 경우이다. 우회전으로 판단하게 된다. 이렇게 정해진 좌회전 또는 우회전 상태는 상태에 따라 알고리즘을 통해 차량이 실제로 좌회전 또는 우회전하도록 한다. 또한, Llinelist, Rlinelist만으로 우회전과 좌회전의 상태를 정하지는 않는다. 후술할 직진 주행 알고리즘에 대한 설명에서 다른 방식으로 우회전과 좌회전 상황을 인식하며 차의 주행성을 증가시킨다.

3.2 직진 상황으로 판정 시 알고리즘

차량이 직진 상황으로 판단되면 인식되는 좌, 우 차선이 모두 직선이라는 가정을 바탕으로 알고리즘이 실행되게 설계하였다. 직선 주행에서 중앙을 유지하는 알고리즘은 차량이 중앙으로부터 떨어진 거리를 통해 조정하는 방법과 기울어진 각도를 통해 조정하는 방법으로 분류될 수 있는데, 본 알고리즘에서는 차량의 각도를 활용해 중앙을 유지하는 알고리즘을 직선 주행 알고리즘에 적용하였다.

직진 상황으로 인식된 후, 가장 먼저 알고리즘은 데이터를 재선별한다. 차선 인식 시스템에서는 한 프레임의 이미지에서 인식되는 차선을 모두 데이터로 넘겨주게 되는데, 이때 차선이 아닌 이미지도 일단 인식하여 데이터가 산출되기 때문에 이에 대한 선별이 필요하다.

```
for i in Llinelist:
    if len(frontLines[i]) >= len(Lline):
        Lline = frontLines[i]
        del L[0]
        L.append(i)
    else:
        Lline = Lline
```

그림 8. 차선 선별 코딩

즉 앞서 선별한 좌 차선 데이터 어레이 중에서 길이가 가장 긴 것, 즉 차선의 점이 가장 많이 인식된 리스트를 선발하는 과정이다. 이를 우 차선도 같이 진행하여 최종적으로 활용할 좌, 우 차선 데이터를 각각 하나씩 산출하게 된다.

차선 데이터는 차량의 카메라에서 인식한 점이 리스트로 표현되어서 모인 어레이 형태의 데이터이다. 따라서 이를 polifit 함수를 통해 이 점들의 추세선을 도출하여 차량의 조향 값이 결정되게 하였다. polifit 함수를 통해 얻는 데이터는 좌표상에서 추세선의 기울기와 절편값이다.

```
TanA = -LDiff[0]
TanB = -RDiff[0]
CetaA = np.arctan(TanA)
CetaB = np.arctan(TanB)
Coeff = -K*np.rad2deg((CetaA+CetaB)/2)
```

그림 9. 직진 조향 산출 코딩

직진 주행에서는 기울기 값을 활용하여 조향각을 산출하도록 설계하였다. polifit 함수에서 산출한 기울기 값은 역삼각함수를 통해 y축으로부터 기울어진 각도를 산출한다. 이를 앞서 선별한 좌, 우 차선에 적용하여 나온 값을 각각 LDiff, RDiff로 정의하였다. 그다음 과정으로 단위 변환 함수를 활용하여 라디안 값으로 산출된 값을 각도로 변환시킨다. 이 과정을 통해 좌, 우 차선이 각각 중앙축으로부터 기울어진 각도 데이터를 얻게 된다. 이때 기울어진 각도가 같은 경우 직진 상황보다 좌/우회전 상황에서의 알고리즘 적용이 더 적절하므로 다시 status를 좌/우회전 상황으로 변환시켜준다. 두 기울어진 각도가 다를 경우, 한 각도는 -값이고 다른 한 각도는 +값일 것이다. 이 두 수치를 평균을 내어 산출하면, 차량이 조향해야 할 최적의 각도가 나오고, 이를 조향 값으로 설정하여 명령하도록 하였다. 이때 실제 산출된 조향 값은 명령 값에 넘어가기 전 처리 과정이 필요한데, 이때 활용한 변수가 K와 Coeff이다. 두 조정 상수를 실제 주행을 실행시키며 조정하였고, 적절한 상숫값이 조정 완료되었을 때 실주행에서도 안정적으로 직선 트랙의 중앙을 유지함을 확인할 수 있었다.

3.3 좌/우회전 상황으로 판정된 경우

우선 조건문을 활용해 현재 주행상태가 우회전인지 좌회전인지를 판단한다. 현재 주행상태가 "좌회전"이라면 Rlinelist, "우회전"이라면 Llinelist를 가져와서 For문을 실행한다.

좌회전의 경우 For문에서는 frontLines[i]와 Rline의 길이를 비교하여, 만약 frontLines[i]의 길이가 더 길다면 이를 Rline에 대입한다. 그리고 최종 선별된 오른쪽 인식 선을 나타내는 리스트인 R에 기존에 저장되어 있던 값을 지우고 현재 frontLines 번호를 추가한다. 이를 i를 모든 i에 대해 실행하면, 최종적으로 선별되는 l값이 R에 저장될 것이다.

이후 np.pyplot 함수를 이용하여 인식된 차선의 추세선을 계산한다. 이후 이 값이 차선이 기울어진 각도의 Tan 값이라는 사실을 이용하여 Theta 값을 구하는데, 이때 역삼각함수를 사용한다. 그리고 앞에서 정의했던 방향각 조정 상수를 Theta에 곱해서 회전각 조절 상수인 Coeff를 도출한다. 이때 만약 Rline 리스트의 min 값이 324 미만이라면, Coeff를 15로 나누어 작게 만든다.

현재 주행 상태가 우회전이면 역시 R과 L만 바꾸고 같은 방식으로 상수를 계산한다.

하지만 이러한 방법으로 조향각을 조정하면 steer 값이 커지는 경우가 발생한다. 하지만 자율 주행차는 steer 값이 -100에서 100 사이로 전달되어야 하며, -60에서 60 사이가 무리 없이 운행할 수 있는 범위이다. 따라서 우리는 steer 값이 60을 넘거나 -60보다 작아질 경우, 60 또는 -60으로 설정하도록 설계하였다.

3.4 차선 인식실패 상황으로 판정된 경우

유의미한 숫자의 점들이 확보되지 않은 데이터가 나와 좌/우 차선 모두 인식을 실패한 경우, 이전 명령을 활용하여 대처하도록 하였다. 차선 인식이 실패된 경우는 주로 매우 드물게 예기치 못한 한 프레임의 이미지에서 일어나는 경우였기에, 이전 명령을 자시 시행시킴을 통한 대처가 실주행에서도 안정적으로 적용됨을 확인할 수 있었다.

Ⅳ. 결론 및 제언

이로써 우리가 고안한 알고리즘을 요약하자면, 추세선 해석, 차량 회전각 해석, 역삼각함수 활용, 좌/우회전 상황 처리, 회전각 조정, 차선 인식실패 상황, 이전명령활용 알고리즘으로 요약할 수 있다. 우리는 위 알고리즘을 통하여, 기존 알고리즘과 비교해 더욱 유연한 정보 처리, 상황 판단 및 대처를 구현하였다고 생각한다. 또, 역삼각함수를 알고리즘상에 활용하는 새로운 접근을 시도한 것에 우리의 알고리즘의 의의를 찾을 수 있다. 하지만, 여전히 한계점이 존재한다. 그 한계점은 이전 시스템보다 정보량이 많아 비교적 정확한 주행이 가능해졌지만, 불필요한 정보가 포함되어있기 때문에 정보를 정제할 방안에 대한 고민이 필요하다. 또한, 직진과 좌우 회전 상황은 서로 다른 데이터 처리와 상황 판단을 하도록 설계하였기에 좌회전·우회전 코스를 잘 통과하는 때도 있지만, 간혹 완전히 반대 방향으로 방향을 틀어버리는 문제점을 시뮬레이션 과정 중 발견하였다. 마지막으로, 차선 데이터는 차량의 카메라에서 인식한 점들이 리스트로 표현되어 조향이 산출되게 되는데, 다른 점들까지 카메라가 인식하여 리스트에 다른 값들이 포함됨으로써 조향 산출에 오류가 나는 부분을 확인할 수 있었다. 즉, 좌회전· 우회전 시 작동과정에 있어 '일관성'을 확보하는 것과 차선 인식률을 높이는 것이 아직 남은 과제이다. 이러한 점들을 개선한다면 충분히 우리의 알고리즘이 실생활에 충분히 적용이 가능할 것으로 보인다.

참고문헌

[1] 김평원, 정형식, 홍범진, 함께 만드는 인공지능 자주차인천대학교출판부, 2019.

[1] 김평원, 정형식, 홍범진. “함께 만드는 인공지능 자주차.” 인천대학교 출판부. 2019.

[2] M. Martinez, A. Roitberg, D. Koester, R. Stiefelhagen, B. Schauerte, "Using Technology Developed for Autonomous Cars to Help Navigate Blind People", IEEE International Conference on Computer Vision Workshops (ICCVW), 1424-1432, 2017.

[3] C. Chandni, V. Variyar, K. Guruvayurappan, "Vision based closed loop pid controller design and implementation for autonomous car", International Conference on Advances in Computing Communications and Informatics (ICACCI), 1928-1933, 2017.

[4] M. Bashiri, C. Fleming, "A platoon-based intersection management system for autonomous vehicles", IEEE Intelligent Vehicles Symposium (IV), 667-672, 2017.

[5] P. Falcone, F. Borrelli, J. Asgari, H. Tseng, D. Hrovat, "A model predictive control approach for combined braking and steering in autonomous vehicles", Mediterranean Conference on Control & Automation, 1-6.

[6] D. Petrich et al., "Map-based long term motion prediction for vehicles in traffic environments", in Proc. IEEE Conf. Intell. Transp. Syst., 2166-2172, 2013.

[7] C. Guo, J. Meguro, Y. Kojima, T. Naito, "A multimodal ADAS system for unmarked urban scenarios based on road context understanding", IEEE Trans. Intell Transp. Syst., 16(4), 1690-1704, Aug. 2015.

3

포스터 논문

3

Journal of Youth Engineering

01 급변하는 차선의 기울기 값의 안정화와 빠른 속도로 직각 코스를 탈출하는 주행 알고리즘

Self-Driving Algorithm Stabilizing Sudden Changes in the Value of the Lane Slope and Escaping the Right-Angled Course at High Speed

백승주 (통진고등학교)

1. 서론

자율 주행 자동차란 운전자 또는 승객의 조작 없이 자동차가 스스로 운행이 가능한 자동차를 말한다. 이는 라이더 센서, 카메라, 레이더, 컴퓨터 등 다양한 장비들이 복잡하게 연결되어 문제 상황을 감지하고, 감지한 정보를 바탕으로 차가 놓인 상황을 인식하며, 각각의 상황별로 정리된 코딩이 자동차를 올바르게 주행시키기 때문에 가능하다. 현재 자율 주행 자동차는 전 세계 기업들이 앞다투어 개발하고 있는 기술이다. 최근 출시된 현대의 아이** 5에는 자율 주행 기능이 탑재되어 있으며, 테**는 모든 기종에 자율 주행 기능이 들어가 있다. 자율 주행기술은 미래의 자동차 시장과 더불어 연결된 무수한 산업들을 좌지우지할 중요한 요인이기에 전 세계로부터 주목받고 있다.

필자는 자율 주행 자동차 알고리즘 연구를 진행하며, 자율 주행 자동차가 90° 각도의 코너를 도는 상황에서 차선을 밟거나 이탈하는 등의 문제점들이 많이 발생한다는 사실을 알게 되었다. 따라서 코너를 안정적이며, 빠르게 도는 방법에 초점을 맞추고 탐구를 진행했다.

2. 추세선 선정과 기울기

1. 차선 선택

연구에 사용한 모형 자율 주행 자동차는 차선이 있는 곳에서만 주행할 수 있다.

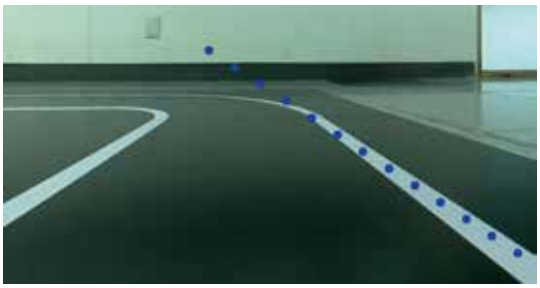


그림 1. 추세선이 인식된 모습

[그림 1]은 자율 주행 자동차 노트북과 연결하면 뜨는 화면이다. 자율 주행 자동차는 카메라를 통해 보이는 두 가지 차선 중 한 차선을 메인 차선으로 설정하며, 그 차선의 기울기 값을 구해 저장하며, 그 기울기 값을 바탕으로 차의 조향값을 조절하며 차선 중앙을 유지하며 직진한다.

간단하게 왼쪽 차선의 추세선을 구했을 때 추세선의 기울기가 음수라면 자동차의 앞 부분이 왼쪽으로 쏠려 는 상황이므로 오른쪽으로 갈 수 있는 조향값을 코딩한다.

2. 수시로 튀는 추세선 값의 안정화 방법

이러한 방법으로 차선을 인식한 후 차동차를 주행시키면 아래의 경우처럼 차선이 아닌 곳을 차선으로 인식하는 경우가 있다.

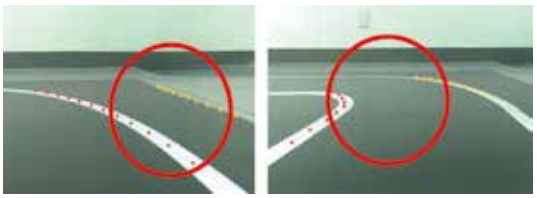


그림 2. 차선이 아닌 곳을 차선으로 인식하는 장면

특히 곡선 구간에서 이러한 현상이 심해지는데, 더욱 정교하게 조향값을 조절해야 하는 곡선 구간에서 크게 요동치는 추세선 기울기 값은 안정적인 주행에 치명적이다. 따라서 계속해서 변화하는 추세선의 기울기 값을 저장해 더하고, 평균값을 내는 코드를 통해 큰 값으로 튀는 추세선의 기울기를 무시하고 주행해 더욱 안정적인 주행이 가능했다.

Previouse라는 변수를 지정해 바로 직전의 e(차선의 기울기 값)을 불러오도록 설정한다. 그 후 자율 주행 자동차의 카메라가 본 화면을 저장하여 주는 프로그래밍을 통해 저장한 사진의 제목에 시간과 e값, 조향값(steer)을 나오게 하여 그 장면에 어떤 e 값과 steer 값이 나오는지를 확인하여 값을 조정하여준다. 이러한 방식으로 중간에 갑자기 값이 튀는 구간을 전부 확인하여 그 값이 튀는 구간에서의 평균값이 180정도 보다 크면 -10을 적용하였다.

```
if self.vars.previouse is None:
    pass
elif abs(self.vars.previouse - e) > 180:
    e= self.vars.previouse - 10
```

그림 3. 평균 기울기를 구하는 코드(일부)

2. 차선(추세선)을 구하지 못하는 경우

급격한 코너를 돌다 보면 잠깐 추세선의 기울기를 구하지 못하는 상황이 생긴다. 그럼 차선의 기울기를 바탕으로 움직이게 설계된 코딩에 오류가 발생하게 되고 차가 더 이상 진행하지 못하게 된다. 이 문제를 해결하기 위해 global 함수를 사용했다.

```
if line is None:
    global e
    if e < 0:
        s = - 110
        v = 200
    elif e > 0:
        s = 110
        v = 200
    print("velocity : ", v, "rapid_steer : ", s)

    steer = s
```

3. 결론

탐구를 진행하며, 속도를 느리게 하면, 안정적으로 코너를 통과할 수 있게 되었다. 그러나, 속도를 빠르게 하면 선을 밟거나, 코스를 이탈하는 일이 자주 발생했다. 따라서 이를 해결할 수 있는 방법에 대해서 생각해보았고 다음과 같이 알고리즘을 수정하였다.

일반적으로 선을 인식하여야 하는 곡선은 원래 다니던 주행 속도로 하고 너무 많이 꺾지않았다. 하지만 직각과 같은 선이 갑자기 사라지는 경우에는 회전을 할 때 급격하게 회전을 시켜줘야 하므로 steer 값을 일반적인 회전보다 값을 더 주어 차선을 벗어나지 않고 회전을 할 수 있도록 만들었다.

본 연구를 통해 한국공학한림원이 제공한 자율 주행 자동차 키트의 난주행 구간을 성공적으로 해결할 수 있었다. 특히 후진 주행상황을 설정하기 위해 수십 번의 알고리즘 수정을 거치며 정확한 후진 주행상황을 도출해낼 수 있었다. 하지만 연구 과정에서 한 가지 문제점을 발견하기도 하였다. 바로 앞서 기술한 외부 요인에 의한 차선 검출 오류 문제이다. 이 부분은 후진 주행 알고리즘을 추가하였음에도 작은 작동 문제들을 일으켜 추후 하드웨어적 보완 등의 연구를 통해 개선할 필요성이 있다.

02 신호등 코딩과 픽셀값 조정을 통한 곡선 주행

Curved Driving with Traffic Light Coding and Pixel Value Adjustment

양호진, 이희찬, 윤민혁, 박준혁 (통진고등학교)

1. 서론

본 연구에서 다루는 주제는 신호등을 인식하는 기본 알고리즘과 곡선 주행 시 속도를 제어하는 것에 관한 알고리즘이다. 기본으로 설정한 직선 주행 코딩과 신호등 인식과 곡선 주행의 차이를 분석하여 변수로 설정한 후 여러 번 실험을 통해 수정을 거듭한 후 최적의 코딩을 완성하였다.

2. 추세선 선정과 기울기

1. 신호등 코딩

모형 자율 주행 자동차에서 신호등 인식은 기본적으로 색상 인식 알고리즘을 기반으로 한다. 카메라에서 신호등의 색상을 분석한 후, 빨간 등일 경우 빨간색 원으로 표시하면서 빨간 등 카운트를 시작한다. 녹색 등을 인식하는 순간부터는 초록색 원이 나오면서 초록색 카운트를 시작한다. 하지만 자율 주행 자동차가 움직이면서 실시간으로 영상을 처리하는 과정에서 주변 붉은색과 녹색 물체를 빨간 신호등과 녹색 신호등으로 잘못 인식하는 경우가 발생한다.

이를 해결하기 위해서는 빨간 등의 카운트를 1로 설정하여 1보다 같거나 크면 초록색 등의 카운트를 [0]으로 설정하는 코딩을 추가하였다. 이렇게 하면 자율 주행 자동차가 운동하면서 카메라에서 신호등의 빨간 색등을 인식하여 그 값이 1보다 같거나 크면 녹색 등 카운트를 [0]으로 변환하여 확실하게 빨간 등임을 인식하도록 했다. 정지 상태에서는 녹색 등 카운트를 [0]으로 전환했기 때문에 녹색 등으로 점멸하는 순간 잘못 인식할 확률을 낮출 수 있다.

```
if not self.vars.stop:
    if reds:
        self.vars.redCnt += 1
    else:
        self.vars.redCnt = 0
    if self.vars.redCnt >= 1:
        self.vars.greenCnt = 0
        self.vars.stop = True

if self.vars.stop:
    if self.vars.greenCnt >= [0]:
        self.vars.redCnt = 0

if 0 < self.vars.redCnt < 500:
    self.vars.stop = True

if 0 < self.vars.greenCnt < 500:
    self.vars.stop = False


if self.vars.stop:
    print('Light stop!!!', 'Red Cnt',self.vars.redCnt, 'Green Cnt=',self.vars.greenCnt )
    return 0, 0
```

그림 1. 신호등 코딩

2. e값 조정을 통한 곡선 주행

e값은 중앙에서 벗어난 픽셀 값으로, S자, U자, 그리고 직각으로 이루어진 차선을 주행하기 위해서는 e값의 범위를 잘 조정하면서 판단해야 한다. 전방 카메라가 차선을 인식하면서 미리 지정해둔 중앙에 해당할 때의 x값과 함숫값의 그래프를 가상으로 그리면서 추세선이 나오고, 이 추세선 그래프에서 400 부근의 x값과 중앙 부분의 x값을 빼고 반복 실험 주행으로 얻은 값인 320을 빼주어서 e값을 구하고 이 e값의 범위를 조정하여 주행하는 전략이다.

본 연구팀이 사용한 추세선의 함숫값은 [그림 2]와 같다

```
line = frontLines[line[1]]
x = line[:, 0]
y = line[:, 1]
coefficient = np.polyfit(y, x, 1)
line = np.poly1d(coefficient)
e = line(400) - center_x - 320
print('line(400)', line(400))
print('center_x=', center_x, 'e=', e)
```

그림 2. 곡선 주행 코딩

차선 인식을 할 수 없는 직각 코스의 경우에는 최근 인식했던 차선 위치의 e값을 적용하여 넘어가는 방식을 채택하였고, 이 때 역시 반복 실험 주행으로 얻었던 평균 e값의 범위를 steer값과 velocity의 값을 일반 주행보다는 빠르게 설정하였다. 이처럼 e값의 범위를 조정 해주는 이유는 곡선과 직선에서 감지되는 e값의 차이를 활용하여 자율 주행 자동차에 steer 값을 변화시켜 곡선과 직선 주행을 원활하게 만들기 위해서이다. e값의 범위에 따라 velocity와 steer 값을 바꾸는 코드는 [그림 3]과 같다.

```
if -160 < e < -110:
    velocity = 60
    steer = -80


elif -120 < e < -90:
    velocity = 60
    steer = -70

elif 80 < e < 200:
    velocity = 60
    steer = 70


else:
    velocity = 60
    steer = 1/20 * e - 20
```

그림 3. velocity와 steer 값을 바꾸는 코드

3. 결론

신호등 코딩은 자기가 생각해서 이러한 상황일 때는 어떻게 할까? 라는 식으로 생각해보면 정말 다양한 방법들을 생각할 수 있다,

결국 알고리즘에는 정답이 없으므로 얼마나 많은 고민과 시행착오를 했느냐에 따라 다른 결과가 나올 수밖에 없다. 본 연구팀은 e값의 범위를 조절하여 회전해야 할 때 회전할 수 있는 각을 만들기 위해 반대쪽 차선으로 붙어 회전하는 각도가 나오도록 유도하였고, 이를 통해 차선 인식이 안 되는 직각 코스에서도 수월하게 넘어갈 수 있었다.

03 자율 주행 자동차의 안정적인 주행을 위한 급격한 곡률 변화 구간에서의 알고리즘 제안 및 클로소이드 곡선을 활용한 주행 경로 설계

Proposal of algorithms in the fast-paced curvature interval for stable driving of autonomous vehicles and design of driving paths utilizing clothoid curves

강예인, 이시현, 정한나 (미림여자고등학교)

1. 서론

본 탐구에서는 급격한 곡률 변화 구간을 원활히 돌 수 있게 하는 알고리즘을 설계하였다. 급격한 곡률 변화 구간과 같은 특수한 상황의 주행에서 클로소이드 곡선을 적용해 자율 주행 자동차 경로를 수학적으로 예측하고, python으로 자율 주행 코드를 작성하여 자율 주행 자동차의 안정적인 주행을 구현할 수 있는 주행 알고리즘 설계를 제안한다.

2. 차선 인식 알고리즘

1. 추세선 생성 (임의의 차선을 직선으로 변환시키는 과정)

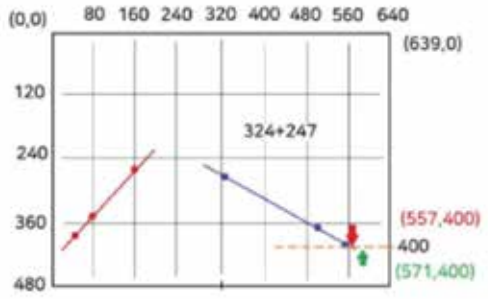
곡선으로 되어 있는 실제 차선을 직선으로 변환시키기 위해서 차선의 점들을 x좌표와 y좌표로 numpy array 값으로 받는다. x, y 좌표의 array를 사용하여 polyfit 함수로 차선에 가장 가까운 추세선을 찾는다. 이때 polyfit 함수를 통해 기울기와 절편으로 이루어진 list가 생성되고 이를 사용하여 poly1d 함수를 사용하여 x = ay+b 라는 일차함수를 만들어 추세선을 생성한다. 이때 생성한 추세선 식을 우리는 line이라고 부른다.

2. 우차선과 좌차선 구별

기본적으로 구별하는 데 있어서는 line(350)을 사용한다. 즉, y축의 350 값에 대응하는 x좌표의 값을 사용하는 것이다. 이때 x좌표의 값이 center_x보다 작으면 좌차선, 크면 우차선으로 인식한다. (center_x = mtx[0,2]로 이미지의 가운데, x=320 부근의 값을 가진다.)

3. e 값에 따른 조향 설정

먼저, line이 우차선일 경우와 좌차선일 경우로 분류하여 e 값을 구한다. 자율 주행차가 양 차선의 중심에 위치해서 주행해야 하므로 위의 그림에서 x좌표의 중심(center_x)과 추세선과의 차이를 계산해 steer 값을 조절하는 방식을 적용한다. 본 연구팀에선,s 추세선과의 차이를 구할 때 line(350)을 사용했다. 먼저 차량을 차선의 중심에 위치시킨 후 출력한 line(350) 값에서 center_x를 뺀다. '실제로 주행을 하면서 나오는 line(350) 값에서 center_x를 뺀 값'을 A라고 하고 '직선 차선의 중앙에 놓았을 때의 line(350) 값'을 B라고 한 다음, A에서 B를 뺀다. 따라서 차량이 얼마나 중앙으로부터 떨어졌는지 그 e값을 계산할 수 있으며 e 값을 기준으로 steer 값이 변화된다.



3. 클로소이드 곡선을 적용한 자율 주행차 운동의 수학적 예측

자율 주행차가 직각인 회전 코스에서 line이 생성되지 않을 경우 어떻게 회전할 수 있는지 추가적인 알고리즘을 고안하였다. line이 없으면, self.vars.steer 값을 기준으로 세 가지 조건을 나눈다. steer 값을 조정하여 반복적으로 자주차 컨트롤러 프로그램을 돌려본 결과, frontlines를 인식하지 못하기 직전의 steer 값이 18(우회전) 이상일 경우 steer값을 80까지 등속도로 증가시켜 고정하였고, steer값이 - 18(좌회전)이하 일 경우에는 steer 값을 -80까지 감소하여 고정시켜 회전하는 것이 적합하다고 판단하였다. 따라서 frontlines가 잡히지 않는 동안만 steer을 고정시켜 직각 코스를 회전하고 다시 잡히는 순간 원래의 알고리즘으로 돌아가도록 하였다.

이러한 문제를 해결하기 위해서 사용한 것이 바로 클로소이드 곡선의 원리이다. 직선부와 원형부 사이에 있는 곡률의 불연속에서 자동차의 원활한 주행을 위해서는 완화곡선이 필요하다. 완화 곡선을 사용하게 되면 주행 시에 원심력의 증감을 적절히 조절할 수 있으므로 보편적으로 활용된다. 이러한 완화곡선의 종류로는 클로소이드, 3차 포물선 등이 있는데, 클로소이드는 자동차의 완화 주행 궤적과 가장 비슷하여 대표적인 완화곡선으로써 도로 설계에서 많이 사용된다는 점에서 클로소이드 곡선을 채택했다. 클로소이드는 곡선 길이에 비례하여 곡률이 증가/증감하는 성질을 가진 곡선으로, 차량이 등속 주행 시 등각속도로 핸들을 회전할 수 있다. 실제 도로에서는 일반적으로 클로소이드 곡선을 사용하여 도로의 완화곡선을 설치한다.

4. 장애물 인식

LiDAR 센서는 레이저를 이용하기 때문에 직진성이 굉장히 높아서 매우 정확한 거리를 감지할 수 있다는 특징이 있다. 또한, 단일센서로 되어있기 때문에, 전면 거리 측정에 능하고, 다른 여러 가지 빛이 존재해도 크게 오류가 생기지 않는다. 반면, 레이저가 빛이기 때문에 비나 안개와 같은 입자들이 있게 되면 빛의 굴절에 의해 직진성이 낮아져 센서로 정확히 돌아오지 못하는 경우도 있다. 또한, 물체의 반사율에 따라 측정 오류가 날 수 있으며, 반사율이 작은 물체는 측정이 안 될 수 있다.

5. 결론

본 연구에서는 다양한 주행 코스에서 차선의 e 값, steer 값, 추세선 등을 계산하여 주행 알고리즘을 제안했다. 특히 클로소이드 곡선을 활용한 완화곡선으로 더욱 안정적인 주행을 위한 주행 경로를 수학적으로 분석하였다. 이러한 사항을 알고리즘 설계에 적용하여 직접 자율 주행 자동차의 주행에 구현함으로써 그 유효성을 검증하였다.

04 차량의 이상적인 조향 방식을 이용한 아커만 조향 구조의 회전각 조정 방법 설계

Designing an rotation angle adjustment method for the Ackerman steering geometry using the vehicle's ideal steering method

김민서 (선덕고등학교)

1. 서론

자율 주행 기술, 그 중에서도 현재 운용하고 있는 차량에 적용할 수 있는 기술이 등장한 이후로, 자율 주행에 관한 관심이 꾸준히 증가해왔다. 자율 주행 자동차에 필요한 기능을 몇 가지만 꼽아 보면 직진 주행, 곡선 주행, 주차, 차선 선택과 변경, 신호등과 장애물 인식이 있다. 이들 모두가 카메라 영상 처리와 조향 · 속도 제어를 연결하는 데 쓰이는 기술이지만, 교차로, 회전 구간, 다차선 고속 도로 등 고난도의 주행 차선이 많이 있는 우리나라의 도로 환경에서 회전은 꽤 중요하다. 직선 도로는 양옆에 차선이 있어 차가 도로의 어디에 위치하는지 정확히 알 수 있다. 그러나 회전 구간의 경우 차량이 이동해야 할 각도와 앞바퀴의 각도가 일치하지 않는 경우가 많다. 따라서 카메라로 촬영한 영상을 처리할 때 자율 주행차량이 회전해야 할 상황인지 인식하는 것은 필수적이라고 할 수 있다.

그런데 회전 상황을 살펴보면 그 안에 여러 가지 경우가 있다. 회전해야 할 각도가 적어서 직진 주행 알고리즘으로 주행한다고 해도 괜찮은 경우, 회전해야 할 각도가 커서 회전 알고리즘을 사용해 회전해야 하는 경우, 회전각이 아주 커서 속도를 조절할 필요가 있는 경우 등이 그것이다. 따라서 이러한 경우 중에서 하나를 선택하여 주행하기 위해서는 도로에 따른 차량의 이상적인 회전 각도와 실제 앞바퀴의 회전 각도와 이동 경로에 관한 함수를 작성하는 과정이 필요하고 할 수 있다.

2. 이론적 배경

1. 차선 인식과 주 차선의 추세선

이미지에서 차선이 특정한 위치의 가로선과 만나는 점에서 추세선을 긋고, 그것의 기울기를 이용해 조향각을 변화시켰다. 추세선 결정에 영향을 주는 가로선의 위치는 차량의 이상적인 조향에 따라 최대한 카메라와 가깝게 위치하도록 설정했다.

2. 차량의 이상적인 조향 방식

[그림 1]와 같은 형태의 설계가 미끄러짐이 없는 차량의 이상적인 조향 방식이다. 사진의 자동차 모델을 기준으로 왼쪽 바퀴의 조향각을 α, 오른쪽 바퀴의 조향각을 β라고 가정했을 때 항상 $\alpha < \beta$이며, 실제 차량의 이동 경로는 동심원의 중심 O에서 차체의 방향 l에 내린 수선의 발 H를 반지름으로 하는 원의 일부이다. 이 설계가 실제 주행에서 쓰인 것은 90도 이상의 급격한 회전 구간이었기 때문에 동심원의 중심이 뒷바퀴와 같은 선 위에 있어도 회전하는 데에 지장이 없었다.

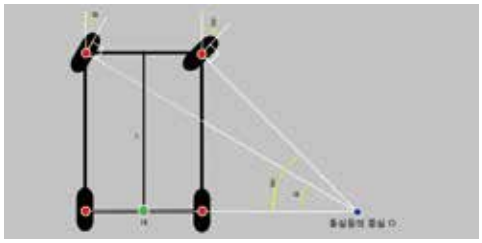


그림 1. 차량의 이상적인 조향 방식.

3. 알고리즘과 조향의 연결

[그림 2]는 직접 측정한 차량의 제원과 설계도를 참조하여, 차체와 이전에 구한 동심원의 중심을 좌표평면 위에 나타낸 것이다. f는 차량의 중심과 동심원의 중심을 이은 직선이다. g는 f와 직교하면서 네 바퀴의 주심을 지나는 직선으로, 차량의 순간 이동 경로를 보여준다. [그림 2]에서 각 EFC를 θ라고 하면, $\tan\theta = \frac{6.25}{c}$이므로 직선 f의 기울기는 $-\frac{6.25}{c}$가 된다. 또한 직선 f와 직교하는 직선 g의 기울기는 $\frac{c}{6.25}$가 된다.

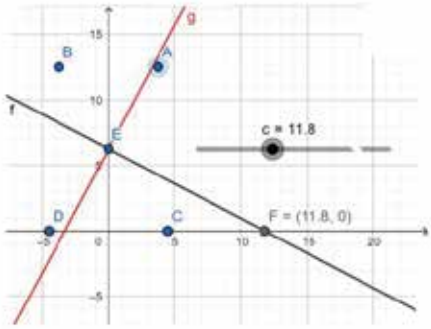


그림 2. 차량의 회전각 계산.
(A ~ C는 4개의 바퀴 위치이고, E는 네 바퀴의 중심이며, F는 동심원의 중심이다.

4. 결론

차량의 이상적인 조향과 차선의 추세선 기술을 활용하여 모형 자율 주행 자동차의 주행 경로를 계획하고, 특히 차량의 회전을 좌표평면 위에 올려놓고 분석하여 직각 회전 구간에서 원의 접선 방향으로 이동할 때 필요한 차량의 조향각을 계산했다. 차량의 무게중심을 기준으로 회전 각도를 계산하여 90도 직각 회전에서 큰 영향력을 발휘했다.

특히 기존의 직진 알고리즘을 이용한 회전에서는 그 각도가 충분하지 않아서 바깥쪽 차선을 밟으면서 회전했고, 그 결과 회전하여 안쪽으로 들어오는 중, 바깥쪽 차선을 안쪽 차선으로 인식하는 한계점이 있었다. 이 설계를 코드에 적용했을 때, 직각 코너에서 이전보다 훨씬 수월하게 회전할 수 있었다.

그러나 이 계산이 항상 옳지는 않았다. 타원이 아닌 반지름이 언제나 일정한 원을 기준으로 회전각을 설계했기 때문에 이 설계를 적용한 실제 주행에서는 90도보다 더 작은 각도로 회전해야 할 때도 원 궤도를 따라 회전하는 오류가 종종 발생했다. 설계 과정에서 회전의 중심을 뒷바퀴와 같은 선 위에 두어서 회전이 잘 된 것이 적은 각도로 회전할 때는 오히려 부정적인 효과를 가져왔다. 이 설계에서 개선하고 싶은 점이 있다면 이것이다. 회전 방향이 정해진 후에는 그 방향 차선을 주 차선으로 설정하고, 회전하며 얻은 정보를 토대로 이심률을 변화시켜서 원 궤도만을 이용하지 않을 수 있다면 더 좋은 회전을 할 수 있을 것이라고 생각한다.

05 오픈 소스를 활용한 교육용 자율 주행 자동차 시스템 개발

Development of Educational Self-driving Vehicle System Using Open Source

이정민, 백용주, 김유빈, 최현서, 최준혁, 배승현 (인천하늘고등학교)

1. 서론

본 연구는 오픈 소스를 활용하여 중고등학생 수준에서 실제 자동차와 유사한 원리로 운영되는 모형 자동차를 이용하여 자율 주행 자동차의 알고리즘을 연구할 수 있는 로봇 시스템을 구축하였다. 고등학교 수학 개념을 기반으로 한 알고리즘을 실제 주행을 통해 확인하였다.

2. 실선과 점선을 구분하는 알고리즘

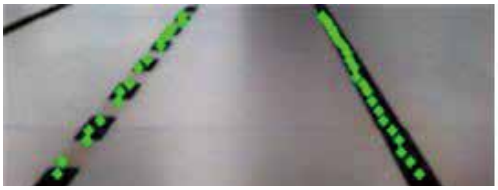


그림 1. 표준편차가 큰 차선 인식(좌측), 표준편차가 작은 차선 인식(우측)

본 연구팀에서는 점선 차선과 실선 차선을 구분하는 방법으로 표준편차를 이용하는 방법을 제안하였다. 이 알고리즘에서는 좌측 선과 우측 선 각각 점 간의 거리 간격을 리스트로 만든 후, 그 간격의 표준편차를 차선별로 각각 구하여 표준편차가 비정상적으로 작다면 어느 부분이 점선인지 판별해낼 수 있다. 방향벡터를 이용한 주행 시스템에 있어 점선은 주행을 불안정하게 하는 방해 요소로 작용하기에 이 알고리즘을 통해 양쪽 선 중 직선과 점선을 판별한다. 이후 평면도로 바꾼 Canny 이미지로부터 직선으로 판별된 선에 대해서만 좌표 정보를 얻어내고 방향벡터를 이용한 주행 시스템에 적용하였다. 수학 교과와 관련된 의사 코드는 다음과 같다.

두 점의 거리를 구하는 함수
function getDistance($p_{1,x}$, $p_{1,y}$, $p_{2,x}$, $p_{2,y}$):
Input: 두 점의 x, y 좌표 ($p_{1,x}$, $p_{1,y}$, $p_{2,x}$, $p_{2,y}$)
Output: 두 점 사이의 거리

$$\text{return } \sqrt{(p_{1,x}-p_{2,x})^2+(p_{1,y}-p_{2,y})^2}$$

차선의 x, y점들의 리스트를 매개변수로 받아 점의 거리로 표준편차를 구하는 함수
function getStandardDeviation(P):
Input: 한 차선 위의 x, y 좌표 리스트
Output: 점들 사이 거리의 표준 표차
gap: 리스트
for I : = 0 to length(P) - 2:
gap[i] = getDistance(P[i].x, P[i+1].x, P[i+1].y)
return gap의 표준편차

코드 1. 점들의 간격으로 표준편차를 구하는 의사 코드

3. 방향 벡터를 이용한 주행 시스템

양쪽 차선의 기울기를 비교하는 데에는 절대적인 위치가 중요하지 않기에 방향 벡터만을 고려해 기울기를 구해볼 수 있다. 평균 방향 벡터, 즉 모든 점들을 합한 벡터를 p라 할 때, p는 다음과 같은 좌표를 가진다.

$$\vec{p}=\left(\sum_{k=1}^{n}P_{kx},\ \sum_{k=1}^{n}P_{ky}\right)$$

수식 1. 평균 방향벡터의 수식

위 수식에서 P_{kx}는 k번째 위치벡터의 x좌표, P_{kx}는 k번째 위치벡터의 y좌표이며, n은 위치벡터의 개수이다. 이 시스템의 주행 알고리즘에서는 벡터의 크기가 의미를 지니지 않기에 위치벡터의 크기를 무시하였지만 좌측 노선과 우측 노선으로부터 입력되는 점들의 개수가 일치하지 않는 경우 문제가 발생할 수 있다. 우측 또는 좌측 노선에 더 많은 점이 존재한다면, 방향 벡터에 영향을 미쳐 단순히 점이 더 많이 감지된 선에 기우는 일이 발생할 수 있다. 이 문제를 해결하기 위해 좌측 노선으로부터 들어오는 점들과 우측 노선으로부터 들어오는 방향 벡터의 평균을 각 노선에 인식된 점의 개수로 나누어야 한다. 이러한 위치벡터를 u라고 할 때, 이를 수식으로 나타내면 (수식 2)와 같이 된다.

$$\vec{u}=\left(\frac{\sum_{k=1}^{n}L_{kx}}{n}+\frac{\sum_{k=1}^{m}R_{kx}}{m},\ \frac{\sum_{k=1}^{n}L_{ky}}{n}+\frac{\sum_{k=1}^{m}R_{ky}}{m}\right)$$

수식 2. 양측 선으로부터 들어오는 정보량 차이를 고려한 방향 벡터값

위 수식에서 L_{kx}는 좌측 선의 k번째 점의 x좌표, R_{ky}는 우측 선의 k번째 점의 y좌표를 나타낸다. n과 m은 각각 좌측 선에서 검출된 점의 개수와 우측 선에서 검출된 점의 개수를 의미한다. 점들의 방향 벡터를 이용하여 자동차의 방향을 결정하는 의사 코드는 다음과 같다.
실제로 적용할 때는 점이 감지되지 않는 경우도 있으므로
0으로 나누어 일어나는 에러를 막기 위해 미소값을 더하였음.
function sumPoints(h = 10^{-8}):

$$\text{return } \frac{\dfrac{\sum_{k=1}^{n}L_{kx}}{n+h}+\dfrac{\sum_{k=1}^{m}R_{kx}}{m+h}}{\dfrac{\sum_{k=1}^{n}L_{ky}}{n+h}+\dfrac{\sum_{k=1}^{m}R_{ky}}{m+h}}$$

코드 2. 점들의 방향벡터를 이용하여 자동차의 방향을 결정하는 의사코드

3. 결론

본 연구는 오픈 소스를 기반으로 자율주행 자동차 시스템을 구성해 코딩 교육용 자율주행 프로젝트의 가능성을 확인하였다. 특히 고등학교 수학 개념을 바탕으로 한 알고리즘을 적용하여 수학 교육의 연장선 상에서 코딩에 접근할 수 있도록 했다는 데에 의의가 있다. 다만 노이즈로 인해 Canny Edge Detection을 이용한 차선 검출이 원활하지 않았다. 이는 물리 교과의 조도 센서 도입을 통해 해결 가능할 것으로 보이며, 이처럼 수학 외 교과 융합으로의 확장을 시도할 수 있다.

06 자율 주행 자동차의 안정적 L자형 곡선 주행을 위한 속도 조절 알고리즘 연구

A Study of Speed Regulation Algorithms Stable L-Shape Curve Driving in self-driving cars

김유진, 변선하, 오민경 (미림여고)

1. 서론

자율 주행 자동차를 횡 방향으로 제어하는 알고리즘에서 경로점 기반 항법을 이용할 때에는 수시로 경로점이 전환되면서 발생하는 불연속적인 제어 명령 때문에 차량이 불안전한 상태가 되기 쉽다. 연구의 목적은 자율 주행 자동차가 곡선 주행 시 속도를 조절하는 것과 관련된 알고리즘을 제안하는 것이다.

2. if 구문 이용 알고리즘

L자형 도로 주행 알고리즘은 line값을 찾지 못하면 바로 전 조향값으로 되돌려 이용한다. 이렇게 되면 line을 인식하지 못하고 그대로 직진할 확률이 커진다. 따라서 바로 전의 값을 사용하도록 하는 것이 아니라 정해진 값을 입력하여 line을 찾도록 하는 방식이 안정적이다.

파이썬의 if 조건문은 조건(참/거짓 - 불 값)에 따라 코드를 실행하는 구문이다. steer값이 20보다 크면 velocity값을 70으로, steer값이 30보다 크면 velocity값을 60으로, steer값이 40보다 크면 velocity값을 65으로, steer값이 50보다 크면 velocity값을 50으로, steer값이 60보다 크면 velocity값을 45로 설정함으로써 steer값이 커짐에 따라 velocity가 줄어든다. 곡선 주행을 할 때 if 구문에 따라 속도가 줄게 되는 원리이다.

```
steer = 0.43 * e + 3

velocity = 80
# e=0일때 가운데, e>0일때 우회전, e<0일때 좌회전


if steer > 20 :
    velocity = 70

    if steer > 30 :
        velocity = 60

        if steer > 40 :
            velocity = 65

            if steer > 50 :
                velocity = 50

                if steer > 60 :
                    velocity = 45
```

그림 1. if 구문 이용 알고리즘

3. 절댓값 기호 이용 알고리즘

절댓값 구문 사용을 위해 math 함수를 불러와야 한다. e의 값에 따라 유동적으로 달라지는 속도를 만들기 위해 abs() 함수를 사용한다. e값이 0에 가까워질수록 주행 구간은 직진에 가깝고 e값이 커질수록 곡산 구간을 주행하고 있는 것이다. e값이 커지면 커질수록 속도는 낮아지고, e값이 0에 가까워질수록 속도가 커진다.

```
import math
```

```
steer = 0.43 * e + 3

velocity = 80 - 0.25 * abs(e)
# e=0일때 가운데, e>0일때 우회전, e<0일때 좌회전
```

그림 2. 절댓값 기호 이용 알고리즘

효율적인 운행을 위한 알고리즘을 짜는 것도 중요하지만 안전성을 위한 알고리즘도 배제하지 않아야 한다. 곡선 주행에서는 차선이 인식되는 범위 밖에서 안정적으로 주행할 수 있도록 하는 속도 조절 알고리즘은 if 구문보다 절댓값을 사용한 알고리즘이 더 효율적이다.

속도의 변화를 불연속적으로 나타내는 if 구문을 사용한 알고리즘보다 절댓값을 사용한 알고리즘이 연속적이고 안정적인 속도 변화를 나타낼 수 있다. 또한 절댓값 구문의 길이가 if 구문의 길이보다 상대적으로 짧기 때문에 경제적이고, 처리 속도가 빠르다. if 구문을 사용할 때, steer 값이 구문에 제시한 값으로 한정되기 때문에, steer 값이 기하급수적으로 커지거나 작아질 때 속도가 유동적으로 바뀌기 어렵다.

4. 결론

if절을 사용한 구문은 연속적이지 못하다는 한계가 있고, 절댓값 기호를 사용한 구문은 다소 기계에 무리가 갈 수 있다는 한계가 있다. 본 연구를 통해 절댓값 기호를 이용하는 방식이 더 안정적임을 밝혔지만 절댓값 기호 구문 역시 한계가 있다. 이를 보완하기 위해서는 너무 작은 값에서 움직여 기계의 무리가 야기되는 것을 최소화하기 위해 일정 크기를 넘어야 알고리즘이 실행될 수 있도록 하는 등의 개선이 필요한다.

4

특별 기고

수

Journal of Youth Engineering

특별 기고

01 규칙 기반 자율 주행 알고리즘 코드의 구조화

서울대학교 컴퓨터공학부 안국현

※ 이 논문은 2019~2020년 청소년 공학 리더(통진고등학교) 출신 학생이 작성한 것입니다.

규칙 기반 자율 주행 알고리즘 코드의 구조화

Structuring the code of the rule-based autonomous driving algorithm

안 중 원* Joongwon Ahn

요약

본 연구에서는 자율 주행 알고리즘이 구현된 코드를 가독성 및 유지 보수성 측면에서 재조명하여 분석하였다. 먼저 유한 상태 기계의 관점에서 선행 연구에서 제시한 알고리즘을 재해석한 후, 함수 분리, 상수 명명 등 일반적인 규칙 기반 알고리즘을 구현할 때 코드를 구조화할 수 있는 방법을 제안하였다. 필자가 제안한 방법은 2019년 자율 주행 알고리즘 경진대회에서 결승 과제를 해결하는 데 사용되었으며, 코드 가독성 및 유지 보수성을 높였음을 입증하였다.

KeyWords: 구조화, 규칙 기반 인공지능

I. 서 론

1.1. 연구 목적 및 개요

자율 주행 알고리즘 개발은 단순히 아이디어를 코드로 구현하는 것으로 끝나지 않는다. 구현된 프로그램을 테스트하고 오류의 원인을 알아내 수정하는 과정을 끊임없이 반복해야 한다. 특히, 사람이 모든 규칙을 명시하는 규칙 기반 알고리즘은 테스트와 수정을 반복하면서 다양한 예외 상황에 대처하기 위해 규칙이 복잡해지며, 그에 따라 코드도 점점 복잡해진다. 즉 시간이 흐르면서 알고리즘 자체는 정교해지지만 알고리즘이 개선되는 속도는 느려진다. 특히 직진, 방향 전환, 정지 등 여러 가지 주행 기능이 코드 상에서 서로 의존성이 커지면 한 가지 기능을 수정했을 때 다른 기능에서도 오류가 발생하고, 간단한 기능을 추가하는 일도 어려워진다.

따라서 정교한 자율 주행 알고리즘을 구현하기 위해서는 코드의 가독성과 유지 보수성을 높이는 것도 무시할 수 없는 과제이다. 본 연구에서는 필자가 제안한 제안된 규칙 기반 자율 주행 알고리즘이([1]), 실제 코드로 구현될 때 어떻게 가독성을 높이고 유지보수를 용이하게 했는지 고찰했다. 또한 대회 당시에 돌발 미션 수행 과정을 살펴봄으로써 실제로 코드 수정이 편리했음을 확인했다.

1.2. 알고리즘 개요

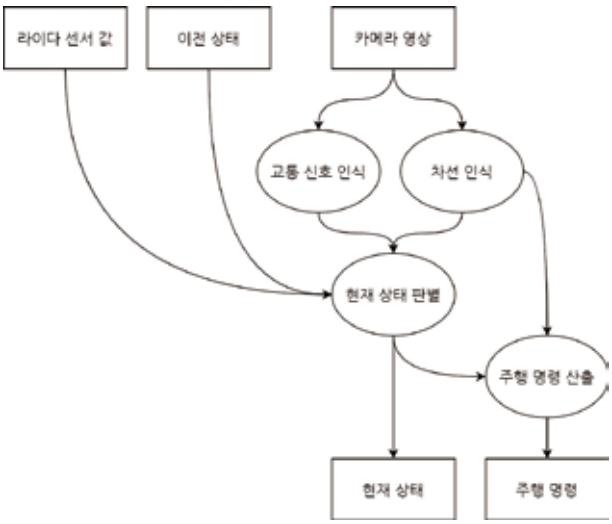


그림 1. 알고리즘의 전체적인 데이터흐름도

[1]에서 자율 주행 알고리즘은 기본적으로 카메라 영상과 센서의 값을 전달받아 주행 속도와 조향각을 반환하는 함수로 구현되었다. 이때 이전의 주행상황을 토대로 현재의 주행상황을 판단할 수 있도록 이전 상태를 매개변수로 받아 현재 상태를 반환하게 하였다. 이러한 구조는 일종의 유한상태기계로 볼 수 있다. 입력으로 들어온 데이터는 그림 1과 같이 가공된다.

상태들 간의 전환 관계는 그림 2와 같다. 이전 상태에 따라 현재 가질 수 있는 상태가 달라지며, 현재 상태에 따라 주행 명령을 산출하는 알고리즘이 달라진다. 상태가 전환되는 조건은 [1]에 상세하게 공개하였다.

* 안중원 (서울대 컴퓨터공학부, winterdawnlight123@gmail.com
통진고 졸업생(현대 모비스 & NAEK 청소년 공학 리더)

Ⅱ. 코드 구조화

2.1. SRP에 의한 함수 분리

SRP(single-responsibilityprinciple, 단일 책임 원칙)는 객체 지향 프로그래밍의 5가지 원칙 중 하나로, 하나의 클래스는 한 가지 기능만을 책임져야 한다는 원칙이다. 이 원칙은 클래스를 사용하지 않는 절차 지향 프로그래밍에서도 함수에 대해 적용할 수 있다. 즉, 한 함수는 한 가지 기능만을 책임진다. 자율 주행 알고리즘에는 교통 신호 인식, 전방 장애물 인식, 출발 및 정지 판단, 주행상태 판단, 주행 속도 및 조향각 판단 등 수많은 과정이 포함된다. 모든 과정을 한 함수에 구현하기보다 각 과정을 담당하는 함수를 따로따로 만드는 편이 가독성이 높아지고 오류가 났을 때 수정할 부분을 더 쉽게 찾을 수 있게 되어 유지보수가 편리해진다. [1]에서 구현된 코드에는 위에서 열거한 기능뿐 아니라 주행 상태 판단에 필요한 각각의 산술적 계산과 각 주행 상태에서의 주행 속도 및 조향각 판단을 수행하는 함수를 각각 만들어 총 21개의 함수가 정의됐으며, 함수들 사이에 명확한 계층 구조를 갖춰 함수 간 독립성을 유지하게 하였다.

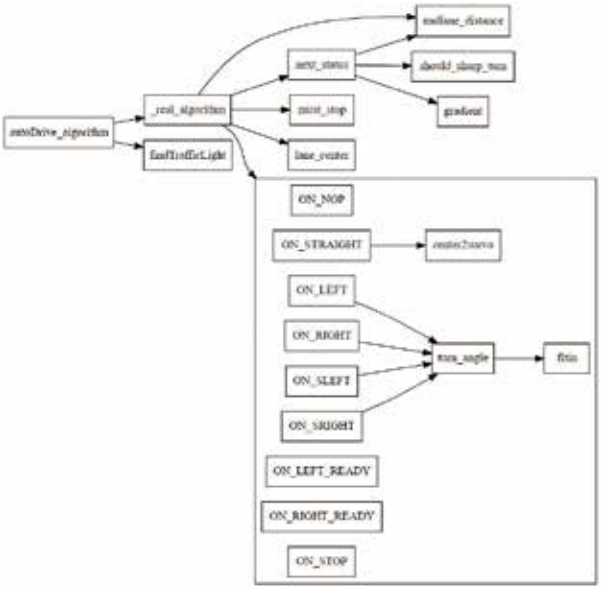


그림 2. 알고리즘 구현 코드의 함수 호출 트리

2.2. 상수 명명을 통한 하드코딩 배제

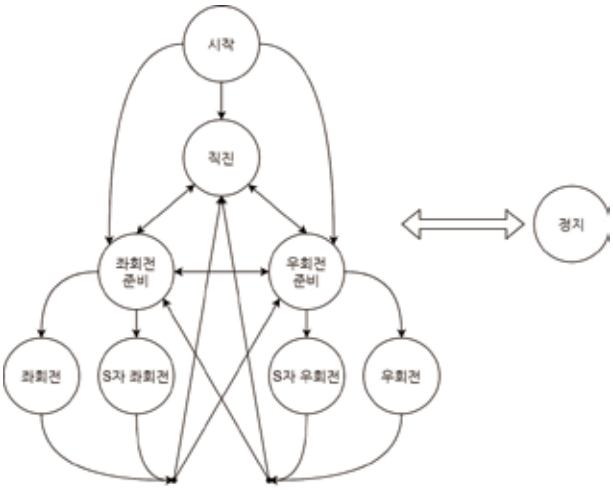


그림 3. 알고리즘을 유한상태기계로 해석한 상태도

[1]에 사용된 모든 상수는 하드코딩(hardcoding)되지 않고 코드의 도입부에 파이썬 변수로 선언되었다. 모형 차량이 주행하는 환경에 따라 알고리즘에 사용되는 상수들의 값은 수정될 필요가 있는데, 모든 상수에 이름을 붙여 한데 모아놓으면 각 오류 상황에 따라 어떤 상수를 수정해야 할지 빠르고 정확하게 판단할 수 있다. 또한 알고리즘에 직접적인 영향을 미치는 상수뿐 아니라 차량과의 통신에 사용되는 특별한 값까지 모두 상수로 정의하여 코드의 가독성을 향상시켰다.

표 1. 코드에 사용된 상수의 개수

분류	개수
디버깅 플래그	2
주행 및 타이밍	15
오차 보정	2
통신 규약 관련	5

Ⅲ. 코드 수정 과정

2019년 청소년 공학 리더 자율 주행차 경진대회 결승전에서 돌발 과제가 주어졌을 때, [1]의 알고리즘에서 수정해야할 부분을 찾는 것은 명확했다.

주차 상황을 나타내는 상태를 정의하고, 이 상황에서의 주행 동작을 정의한다. 직진 상황에서 주차 상황으로 넘어가는 조건을 코드에 추가한다.

다른 주행 상황에서의 조건이나 동작은 완전히 독립되어있어 주차 동작에 영향을 받지 않았고, 함수 내에서 수정할 부분을 찾는 것도 용이하였다. 주차 상황의 동작은 직진 상황의 동작과 비슷하게 특정한 기준선을 따라가도록 하되 일정 시간이 지나면 정지하도록 정의했다. 직진 상황에서 주차 상황으로 넘어가는 조건은 좌우 차선과 정면에 차선이 동시에 인식될 때로 하였다. 상태 전환 조건과 동작 구현 자체는 단순했다는 점을 고려할 때, 코드 구조화가 돌발 과제 성공의 비결이었다고 볼 수 있다.

Ⅳ. 결론 및 제언

4.1. 결론

자율 주행 알고리즘에 관한 기존의 다양한 전략들은 계속해서 발전하고 있다([3]~[5]). [1]에서 제안한 알고리즘을 구현할 때 코드를 효과적으로 구조화할 수 있었던 방법은 SRP를 엄격히 적용한 함수 분리와 하드코딩을 배제하는 상수 명명이었다. 이러한 코드 구조화 방법은 자율 주행 알고리즘뿐 아니라 비디오 게임, 챗봇 등 다른 분야에 사용되는 모든 규칙 기반 알고리즘에도 적용할 수 있다. 이를 통해 코드의 가독성과 유지 보수성을 높여 알고리즘 개선을 코드에 빠르게 반영할 수 있으며 오류에 대한 코드 수정도 신속하고 정확하게 이루어질 수 있다. 실제로 2019년 청소년 공학 리더 자율 주행차

경진대회 결승전에서 주어진 시간 내에 새로운 기능을 추가하는 돌발 미션을 해결하는 데 성공함으로써 코드 구조화가 유지 보수성을 크게 향상시켰음을 입증하였다. 이처럼 코딩은 남에게 보여주기 위해 짜는 것이 아니지만 문제를 해결하기 위해 코딩을 수정할 경우, 신속하게 수정해야 할 모듈을 찾아갈 수 있도록 코딩을 하나의 텍스트로 간주하고 구조화하는 전략이 필요한 것이다.

4.2. 제언

규칙 기반 알고리즘은 본 연구에서 제시했듯 논리적 기능에 따라 코드를 여러 부분으로 나누어 놓음으로써 이후 유지 보수성을 크게 높일 수 있지만, 최근 부상하고 있는 딥러닝(심층 학습)의 알고리즘은 사람이 내부 논리 구조를 완전히 이해하기 어려워([2]) 본 연구에서 제시한 방법을 적용해도 쉽게 기능을 추가하거나 개선하기 어려웠다. 따라서 규칙 기반이 아닌 다른 형태의 알고리즘을 사용하는 경우 학습 데이터 추가와 같은 다른 방향의 접근이 필요할 것이다.

참고문헌

[1] 안중원 외, "연속적 되먹임 체계를 통한 직선 및 곡선 차선 자율주행과 다각형 추출을 통한 신호등 인식 알고리즘", 청년공학, 제4집, pp.27-30, 2020. 9.

[2] S. Grigorescu외 3인, "A survey of deep learning techniques for autonomous driving", Journal of field robotics, Vol.37 (3), pp.362-386, 2020. 4.

[3] 김평원, 정형식, 홍범진. 함께 만드는 인공지능 자주차, 인천대학교 출판부. 2019.

[4] Satoshi Suzuki and others, Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, pp. 32–46, 1985.

[5] John Canny, A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, pp. 679–698, 1986.

특별 기고

02 2021년형 3세대 자주차의 특징

카이스트 최진혁

※ 이 논문은 2018~2019년 인천하늘고등학교 청소년 공학 리더 출신 학생이 작성한 것입니다.

2021년형 3세대 자주차의 특징

Upgrade a 2020 Autonomous Vehicle to a 2021 Model

최 진 혁* Choi jin-hyeok

요약

김평원 교수팀은 교육용 자주차를 만들어 지속적으로 개량해왔다. 1세대 자주차는 미국의 로봇 키트를 활용한 것으로 교육용 키트로 개발하는 데 실패하였고, 2세대 자주차는 제1회, 제2회 자주차 경진대회에 사용되었으나 하드웨어에 결함이 많아 단종되었다. 이 연구에서는 2020년 개발한 3세대 자주차가 우수한 성능에도 불구하고 장시간 사용하면서 발생할 수밖에 없는 문제의 원인을 진단하고, 교육 현장에서 안정적으로 활용할 수 있도록 업그레이드한 2021년형 자주차를 개발하였다. 앞으로 자주차는 개량을 거듭하여 4세대 자주차로 발전할 예정이다.

KeyWords: 자율 주행 자동차, PCB 보드, CNC 밀링

I. 서 론

자주차는 많은 시행착오를 겪으면서 발전해왔다. 2017년 인천대학교 임베디드시스템공학과 팀에서 개발한 1세대 자주차는 미국의 로봇 키트를 활용하는 수준으로 교육용 키트로 발전시키지 못하고 실패하였다.

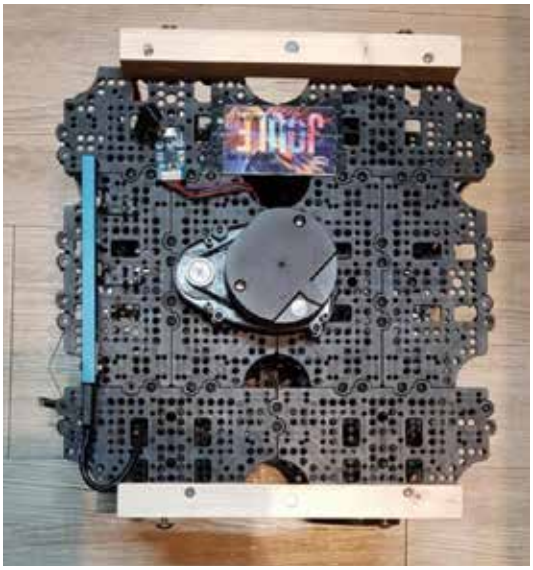


그림 1. 2017년형 자주차(1세대 자주차)

교육용 키트로 자체 개발한 2세대 자주차는 2018년~2019년 동안 청소년 공학 리더 프로그램에 사용되었다(그림 2). 하지만 통신 및 하드웨어의 잦은 고장으로 문제가 누적되어, 3세대 자주차로 대체되었다. 2020년 개발된 3세대 자주차는 현대모비스와 한국공학한림원이 지원하는 청소년 공학 리더 출신 서울대학교 재학생들이 개발했다는 점에서 의미가 있다. 필자 역시 청소년 공학 리더 출신으로 김평원 교수팀에 참여하여 3세대 자주차를 업그레이드하는 데 기여하였다. 이 논문은 3세대 자주차를 업그레이드한 사항을 공개한 것이다.

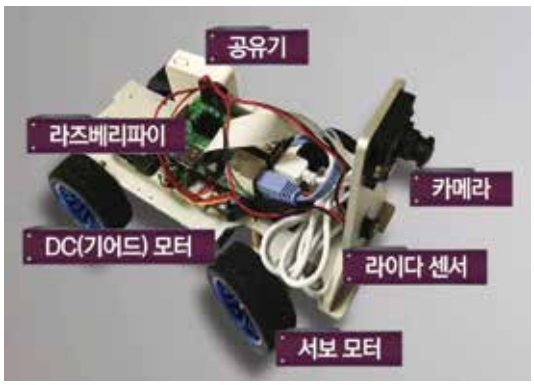


그림 2. 2018년형 자주차(2세대 자주차)

그림 3. 2020년형 자주차(3세대 자주차)

* 최진혁 (카이스트, tammy0502@naver.com
인천하늘고 졸업생(현대모비스 & NAEK 청소년 공학 리더)

Ⅱ. 2020년형 자주차의 문제점 분석

2.1. Regulator 설계 문제

그림 3. 2020년형 자주차(3세대 자주차)

- Input voltage: 4.5-28V
- Output Voltage: 0.8-20V (adjustable)
- Output current: rated current 3A (MAX).
- Switching Frequency: 1MHz
- Output Ripple:less than 30mV
- Efficiency:96%(max)
- Operating temperature: Industrial grade (-40 C to +85 C)
- Module Properties: Non-isolated step-down module (buck)
- Size:22*17*4mm (size of a coin)
- Weight: 2g

그림 4. Picture of the regulator and specification

Switching regulator 의 특성상, 기본적으로 많은 열은 발생하지 않는다. 하지만 Maximum current가 3A라는 특징 때문에 요구되는 부팅 전류가 2A가 넘은 RPI 4B+를 구동시킬 경우 많은 열이 발생하지만 기존 회로에서는 방열판을 사용하지 않았다. 또한 module의 maximum operating temperature가 섭씨 85도로 상대적으로 낮다. 또한 가변 저항으로 OUTPUT 단의 전압을 조절하기 때문에 INPUT 전압에 따라서 일정하게 OUTPUT 유지가 힘들어 battery level에 따라 output 전압이 달라질 수 있다.

2.2. PCB 보드 디자인 문제

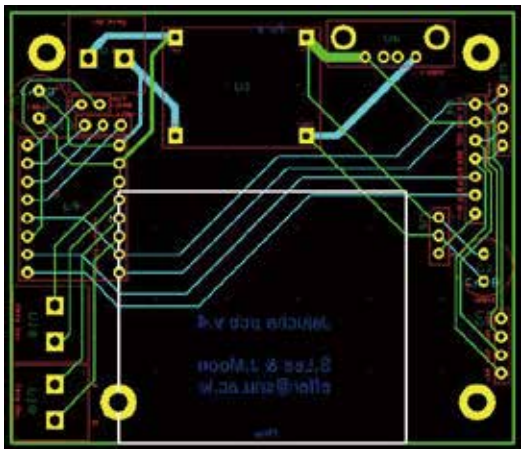


그림 5. Picture of the prior printed circuit board

PCB를 디자인할 때는 지나갈 전류에 따라 line width를 지정해 주어야 한다. Line의 width의 두께를 정하는 공식은 다음과 같다.

The trace width is calculated as follows:
First, the Area is calculated:
Area[mils^2]=(Current[Amps]/(k*(Temp_Rise[deg. C])^b))^(1/c)
Then, the Width is calculated:
Width[mils] = Area[mils^2]/(Thickness[oz]*1.378[mils/oz])
For IPC-2221 internal layers: k = 0.024, b = 0.44, c = 0.725
For IPC-2221 external layers: k = 0.048, b = 0.44, c = 0.725

공식에 따라 계산하게 되면 stall torque가 0.7A인 MG90를 안정적으로 구동하기 위해서는 0.5mm 이상의 line width가 요구되지만 기존 설계는 0.3mm를 사용하였다 또한 booting current가 2A가 넘는 RPI를 구동시키기 위해서는 2mm 이상의 line width가 요구되지만 기존 설계에서는 1mm를 사용하였다.

또한 [그림 5]를 보면 DCDC regulator의 output 전류를 RPI와 servo가 동시에 사용한다. 두 기기 모두 peak current를 사용한다고 생각했을 때 0.7A(servo)+2.0A(RPI)= 2.7A 정도를 사용한다고 할 수 있다. DC-DC regulator가 3A가 maximum이기 때문에 많은 량의 열이 발생하여 regulator가 견디지 못하고 녹아버리는 상황이 발생할 수도 있다.

Ⅲ. 2021년형 자주차의 PCB 개선 사항

3.1. Regulator의 개선

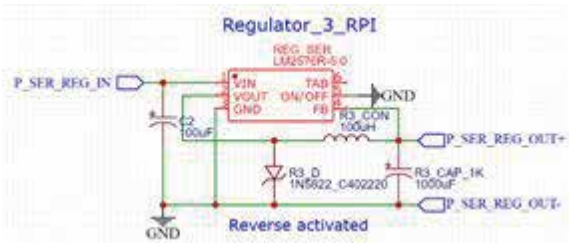


그림 6. Picture of the recently designed regulator

HTC사의 LM2576R-5.0 모델을 사용하였다 maximum 40v 까지 input을 줄 수 있는 모델로 섭씨 150도의 내열성을 가지고 있다. 또한 INPUT = 5V-40V, OUTPUT = 5.0V로 일정하기 때문에 추가적으로 voltage를 조정해 줄 필요가 없다. 또한 방열판을 추가적으로 달아주었다.

3.2. PCB 보드의 개선

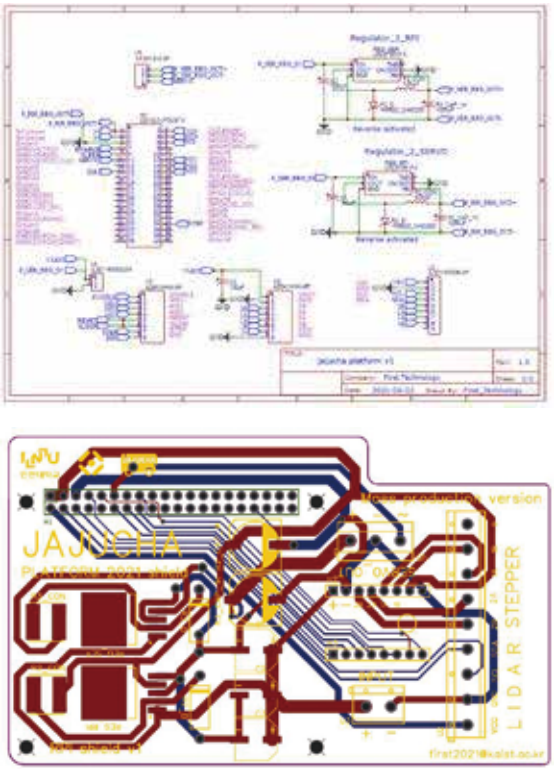


그림 7. 8. Picture of the recently designed PCB

우선 DC-DC regulator의 load를 최소화하기 위해서 SERVO, RPI에 각각 1개씩 전류를 공급하는 DC-DC regulator를 할당하여 설계하였다. 또한 최대 3A까지 사용하는 RPI에 맞게 line width 2mm로 할당하여 PCB에서 발생할 수 있는 열과 저항을 최소화하였다.

3.3. 연결선의 분리 방지

그림 9. Picture of the additional modification parts

2020년형 자주차를 2021년형으로 개조하면서 가장 초점을 맞추었던 부분은 hardware abstraction을 통한 조립 process의 simplification이었다. 기존에는 점퍼선이 사용되었는데 pin header에 점퍼선을 연결하면 선이 분리될 수 있는 확률이 매우 크다. 따라서 [그림 9]의 2*20 Female pin header를 사용하여 RPI의 GPIO 핀들이 PCB 보드와 직접 연결될 수 있도록 하였다.

3.4. 접촉 불량 방지

또한 [그림 10]에서 환인 할 수 있듯이 필수적으로 선이 연결되어야 하는 부분(INPUT, SERVO_OUT, LIDAR, STEPPER)에는 terminal block을 사용하여 screw를 이용하여 cable을 고정할 수 있도록 하여 주행 중에 발생할 수 있는 접촉 불량 문제를 대폭 개선하였다. 또한 PCB board에서 RPI로 전류를 직접적으로 공급하기 때문에 부가적인 USB on board 모듈을 사용할 필요가 없었다.

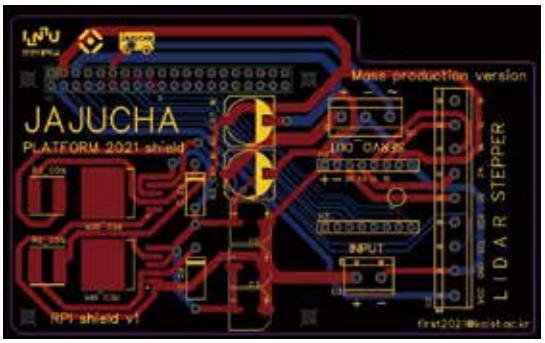


그림 10. Picture of the additional modification parts

3.5. PCB 보드 제작 방법

ALLPCB라는 업체에 GERBER(회로 기판), BOM(부품 리스트), PICK&PLACE(부품 위치) 파일을 제공한 후 TURNKEY(제조자가 부품을 조달하는)방식으로 제작하였다.

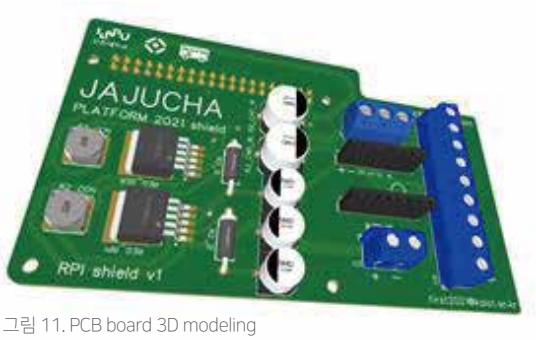


그림 11. PCB board 3D modeling

Ⅳ. 2021년형 자주차의 하드웨어 문제점

4.1. 프레임 공차 발생

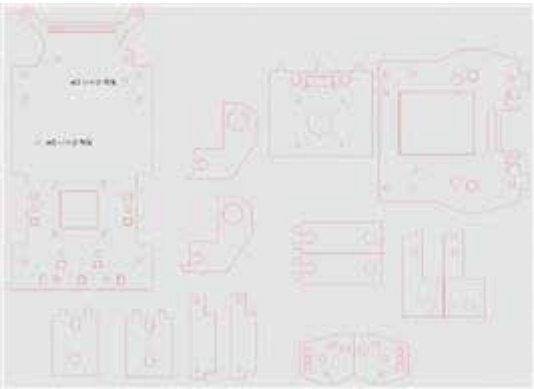


그림 12. Frame svg file

교육용 프로젝트인 만큼 학생들이 자동차의 설계를 바탕으로 문제점을 파악하고 수정할 수 있도록 하는 것이 중요하지만 기존 설계에는 기본적으로 완전한 모델이 제공되지 않았고, [그림 12]와 같이 SKETCHUP 설계를 svg로 변환한 것이다. 이 때문에 원이 표현되지 않는 sketchup의 특성상 가공을 진행할 경우 공차가 발생할 수밖에 없다. 또 laser cutting 방식을 사용하여 frame 가공을 진행하였기 때문에 심각한 공차가 발생하는 것을 피할 수 없다.

4.2. 기어와 모터 사이의 유격 발생

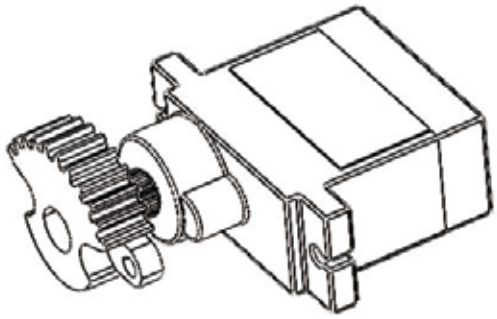


그림 13. prior pinion fixture (2020년형 자주차)

2020년형 자주차는 pinion 기어를 바로 servo motor에 고정하는 방식이었다. 하지만 PLA 소재로 강성이 약하고 프린팅 시 수축이 발생하는 FDM 방식의 3D printer의 특성상 내구성이 약하다는 한계점을 가지고 있다. 따라서 이러한 방식으로 제작할 때 추후에 기어와 모터 사이의 유격이 발생할 가능성이 매우 높다.

4.3. 하드웨어 매뉴얼 부재

교육용 프로젝트인 만들 학생들이 자동차를 분해, 조립할 수 있도록 정확한 매뉴얼을 제공해야 한다. 하지만 기존 프로젝트에서는 매뉴얼을 제공하지 않았기 때문에 학생들이 자동차의 고장이 발생할 경우 원인을 파악하기 어려웠다는 한계가 있다.

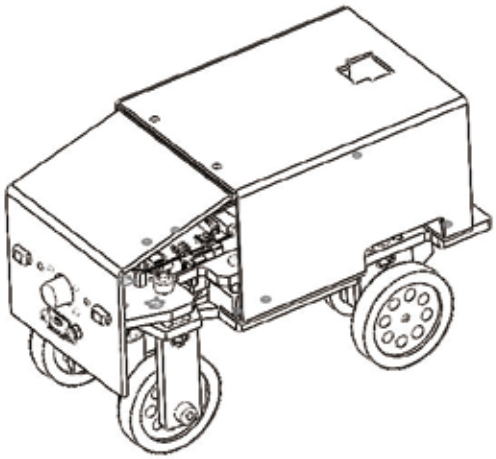


그림 14. AJUCHA Solidworks modeling

V. 2021년형 자주차의 하드웨어 개선 사항

5.1. 레이저 커팅에서 CNC 밀링으로 변경

Solidworks를 사용하여 자동차를 처음부터 재설계를 진행하고 Assembly 작업을 진행하여 자동차를 기계적으로 Simulation 할 수 있고 추후에 ROS를 바탕으로 simulation할 수 있도록 URDF 논리에 맞게 mate를 구성하였다.

그림 15. frame CAM

기본적인 설계가 끝난 후에는 제작 방법을 laser cutting에서 CNC Milling으로 변경하였다. 본 연구자가 보유하고 있는 2.5D CNC machine으로 가공할 수 있도록 Fusion 360 manufacturing 기능을 사용하여 tool path를 생성한 후 가공을 진행하였다.

그림 16. JAJUCHA frame CNC milling

5T 백색 아크릴의 CNC milling 작업을 진행하였다. 직경 2mm 엔드밀을 feed 1000, 15,000rpm으로 2.5mm씩 절입하며 진행하였다. 아크릴에서 많은 열이 발생하면 아크릴이 녹아 엔드밀에 눌어붙을 우려가 있었기 때문에 수용성 절삭유를 공급하였다.

5.2. 기어와 모터 사이의 유격 방지

기본적으로 MG90s 서보 모터에는 servo arm이 포함되어 있다. Pinion gear를 서보 모터에 고정하는 방식을 직접 고정하는 방식에서 arm 모터에 고정한 후 pinion 기어를 arm에 고정하는 방식으로 하여 arm에서 발생하는 backlash를 최소화하였다.

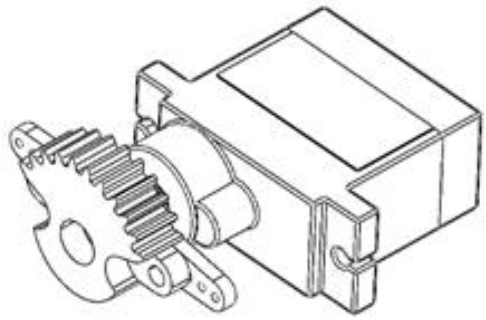


그림 17. New pinion fixture

5.3. 상세한 하드웨어 매뉴얼 제공

교육용 프로젝트인 만큼 기존의 완제품을 distribution 하는 방식에서 벗어나 부품만 있으면 설명서를 바탕으로 조립할 수 있도록 체계적으로 설명서를 제작하였다. 설명서는 base plate 조립, PCB plate 조립, wiring, 세 파트로 나누어 구성하여 누구나 자동차를 처음부터 조립할 수 있고 개조 혹은 부품 수리를 원하는 경우 부품만 있다면 얼마든지 실행할 수 있도록 플랫폼을 설계하였다.

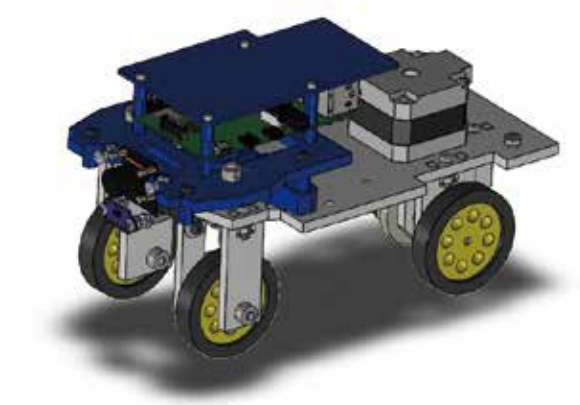


그림 18. 개선된 부분(청색)

VI. 2021년형 자주차의 소프트웨어 개선 사항

기본적으로 자주차의 main software는 기존 것을 수정하는 것보다는 개선된 하드웨어에 맞게 새로 개발하는 것이 더 쉽다. 하지만 이미 2020년형 자주차의 소프트웨에 익숙한 학생들이 있으므로, 기존의 소프트웨어를 그대로 사용하되 동작을 기본적으로 테스트할 수 있는 Middleware를 PYQT를 제작하여 보완하였다(그림 19).

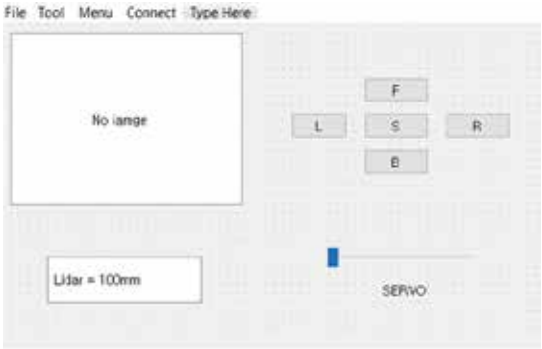


그림 19. New JAJUCHA tester

참고문헌

[1] 김평원, 정형식, 홍범진, 함께 만드는 인공지능 자주차인천대학교출판부, 2019.

[2] R. Wallace, A. Stentz, C. E. Thorpe, H. Maravec, W. Whittaker, T. Kanade, "First results in robot road-following" in IJCAI, Citeseer, 1089-1095, 1985.

[3] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt et al., "Towards fully autonomous driving: Systems and algorithms", in Proc. IEEE Intelligent Vehicles Symposium 2011, 163-168, 2011.

[4] M. P. Philipsen, M. B. Jensen, A. Mogelmose, T. B. Moeslund, M. M. Trivedi, "Traffic light detection: A learning algorithm and evaluations on challenging dataset", in Proc. 18th IEEE International Conference on Intelligent Transportation Systems. 2341-2345, 2015.

[5] E. Guizzo, How Google's Self-Driving Car Works, October 2011.

[6] P. Joshi, Artificial Intelligence with Python, Packt Publishing, 2017.

[7] A. Mukhtar, L. Xia, T.B. Tang, "Vehicle detection techniques for collision avoidance systems: A review", IEEE Trans. Intelligent Transportation Systems, vol. 16, no. 5, Oct. 2015.

[8] X. T. Nguyen, K.-T. Nguyen, H.-J. Lee, H. Kim, "ROI-based LiDAR sampling algorithm in on-road environment for autonomous driving," vol. 7, IEEE Access, pp. 90243-90253, July 2019.

[9] W. Chu, Y. Liu, C. Shen, D. Cai, X.-S. Hua, "Multi-task vehicle detection with region-of-interest voting," IEEE Trans. Image Processing, vol. 27, no. 1, pp. 432-441, Jan. 2018.

2021년 <청년공학> 제5집 발간 개요

목적

·이공계를 희망하는 청소년들이 수월성을 함양할 수 있도록 연구 성과물을 발표할 수 있는 장을 만듦

·이공계를 희망하는 고등학생들의 신선한 아이디어 돋보이는 주제와 연구 결과물 공유

·논문 작성과 발표를 통해 수월성 교육을 경험할 수 있는 계기를 제공

필요성

·창의인재 교육을 중시한 융합형 교육과정, 프로젝트 학습, 수행평가 등의 교육 방법론이 대두되면서 탐구활동을 통한 수준 높은 실험 및 연구보고서가 만들어지고 있으나 단위학교 발표에 그쳐 널리 알려지지 않고 사장되고 있음. 고등학생이 발표할 수 있는 전문 학술지는 없음.

·국내 외 저널에 미성년자가 논문을 게재한 경우 대입에 활용할 수 없도록 조치했기 때문에 논문을 대체할 포스터 발표 형태의 결과물을 만들고 있어 이를 공유하고 선의의 경쟁을 할 수 있는 발표의 장이 필요함.

1. 주제

※ 논문 : 자율 주행 자동차의 주행 능력과 관련된 방법 제안

- 예시 1 : 효과적인 장애물 회피 제어를 위한 라이더 센터 데이터의 처리 방법 연구
- 예시 2 : 안정적인 곡선 주행을 위한 실시간 데이터 보정 방법 연구

※ 포스터 논문 : 심사 결과 논문이 아닌 보고서 수준으로 평가 받은 경우

2. 자격

※ 논문 투고 자격

2020년 청소년 공학 리더 자주차 경진 대회 본선 참여한 학교의 학생들

① 저자 표기 : 소수의 인원이 논문을 쓰고, 팀원 전체의 이름을 올리는 행위는 심사 과정에서 적발하여 게재 불가. 실제 논문을 쓰고 기여한 학생만 저자 표기를 해야 함.

② 중복 투고 : 개인 논문 투고와 동시와 팀원 일부와 단체 논문을 투고하는 것을 허용함.

3. 출판 일정

① 투고 마감 : 2021년 6월 30일까지, 청소년 공학 리더 담당 교사에게 제출
② 논문 심사 : 2021년 7월 31일까지 논문 심사 및 PDF 출판 완료
③ 출판 : 2021년 8월

4. 투고 논문 작성법

1. 논문 원고의 본문 중에 사용되는 영어는 소문자를 사용하는 것을 원칙으로 한다(단 고유명사, 약자는 제외). 문장의 처음이 영어단어로 시작되는 경우에는 첫 글자를 대문자로 한다.

2. 논문 원고의 초록(한글 요약문)은 200-400자를 기준으로 한다.

3. 원고작성은 한글맞춤법 표준안에 준하여 작성하고, 내용은 장과 절로 구분하여 다음의 번호체계를 따른다.
   1.
   1.1
   1.1.1
   1.2
   2.

4. 원고작성은 논문의 한글제목, 영문제목, (한문)저자명, 영문 저자명, 초록(한글), Key Words, 본문, 참고문헌 순서로 작성한다.

5. 그림과 표는 그림 1, 그림 2, 표 1, 표 2 등으로 표시하고, 그림의 제목은 그림 밑에, 표의 제목은 표 위에 기입한다.

6. 인용된 참고문헌은 원고의 끝에 기재하며, 인용 번호를 본문의 인용 장소에 반드시 기입하고, 인용 순서대로 다음과 같이 표시한다.
   가. 단행본의 경우 : 저자명, 책명, 출판사, 인용페이지, 출판년도.
      例 [1] : 홍길동, 전기기기공학, 동명사, pp. 186-195, 1990.
      例 [2] : C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley pp.145-188, 1981.
   나. 논문지의 경우 : 저자명, 논문제목, 논문지명, 권, 호, 페이지, 출판년월
      例 [1] : 홍길동, 김유신, "2상8극형 HB형 리니어 펄스모터의 자속분포와 정특성 해석", 대한전기학회논문지, 제42권, 9호, pp. 9-18, 1993. 9.
      例 [2] : T. Larrabee, "Test pattern generation using boolean satis fiability", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 3, no. 11, pp. 4-15, January 1992.
      例 [3] : 이순신외 4인, "3상 전압형 PMW 컨버터의 특성개선에 관한 연구", 대한전기학회 하계학술대회 논문집 (B), pp. 830-832, 1993. 7.

7. 원고의 모든 단위는 SI단위를 원칙으로 한다.

8. 공정한 심사를 위해 아래와 같은 요령으로 최종 파일을 작성하여 제출하되, 최종 편집과 교정은 한국공학한림원에서 투고 논문과는 다른 출판 양식으로 진행한다.
   ① 원고 내 모든 항은 양쪽 혼합으로 정렬시키며, 이 원칙은 표 제목, 그림 제목, 문단 등에도 적용된다. (단, 제목, 저자이름, 소제목(장)은 가운데 정렬이다)
   ② 문단이 끝날 때에는 Enter Key를 사용하여 줄 바꿈을 하며, 문단이 끝나는 곳 이외에는 "Enter Key"를 사용하지 않는다. 소제목(장), 절 다음은 본문 글자 크기로 위아래를 한 칸씩 띄운다.
   ③ 그림이나 도표의 가로길이는 80mm로 맞추고, 이보다 크거나 작을 경우 확대 혹은 축소를 하여야 한다. 이때 (확대 혹은 축소 후) 글자의 크기는 가독성을 고려하여 최소 2mm 이상 3mm가 넘지 않도록 한다.
   ④ 도표를 그릴 때에는 표 그리기로 하고 선 그리기로 하지 않는다.
   ⑤ 논문의 기본 편집 형식은 2단 조판이지만, 매 논문 첫 페이지는 한글 제목부터 Key Words까지는 1단 편집, 논문 내용부터 2단 편집형식이다.

9. 장 제목은 9 point, 서체는 중고딕, 진하게로 하고 위아래를 한 줄씩 띄운다.

10. 절 제목은 8.5 point, 서체는 중고딕, 진하게로 한다. 위아래를 한 줄씩 띄운다.

11. 본문 내용은 8.5point, 서체는 신명조, 줄 간격은 150%로 한다.

12. 수식 편집 과정은 다음과 같다.

① 수식은 8.5point, 신명조로 한다.

② 수식을 작성하고 나서 위, 아래 한 줄씩 여백을 준다.

③ 본문 속의 수식은 수식이 있는 경우와 없는 경우가 줄 간격이 틀리기 때문에 수식이 있는 경우 아래 줄은 따로 줄 간격을 조절해야 한다. 즉, 수식 아래 줄은 100~130%으로 줄 간격을 조정해야만 줄 간격이 보기 좋게 된다.

④ 수식 다음에는 번호를 차례대로 매긴다.

13. 참고문헌 제목은 글자 크기는 9point, 서체는 중고딕, 줄 간격은 150%, 가운데 정렬이고 참고문헌은 2칸씩 띄운다. 다음에 한 줄을 띄우고 내용을 기입한다. 참고문헌 내용은 신명조 8.5point, 줄 간격은 150%, 왼쪽 정렬, 왼쪽 여백 4ch, 내어 쓰기 3ch로 한다.

청년공학 제5집

인쇄일 2021년 8월 6일
발행일 2021년 8월 10일
발행인 권오경
발행처 한국공학한림원
TEL 02-6009-4000
FAX 02-6009-4010
E-MAIL naek@naek.or.kr
ISBN 2383-885X

이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

논문 심사위원장 : Prof. Marco Anisetti (University of Milan)
논문 심사위원 : 대외비

NAEK 한국공학한림원

서울시 강남구 테헤란로 305 한국기술센터 15층 한국공학한림원
www.naek.or.kr

95
9 772383 885000
ISSN 2383-885X