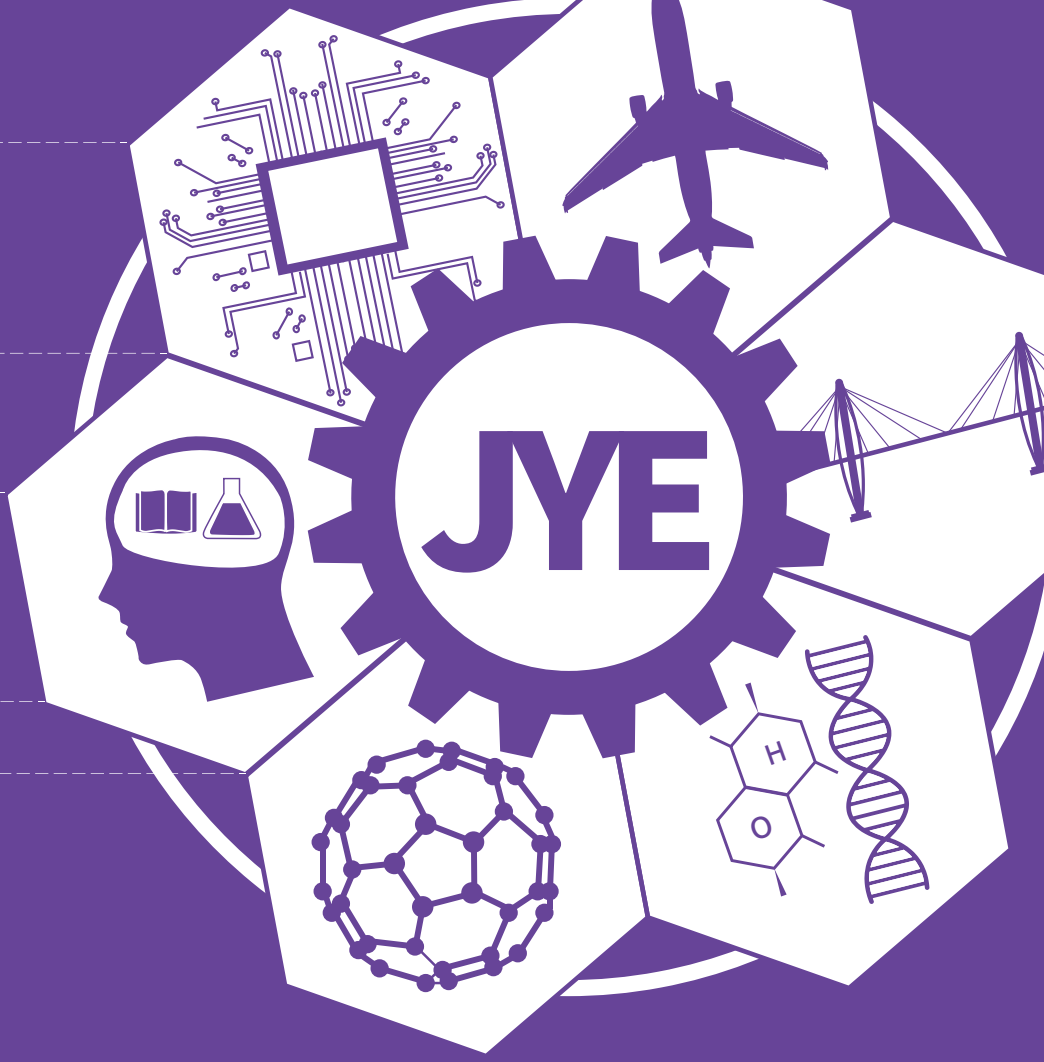


Journal of Youth Engineering

청년공학

제 6 집

2022년

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

NAEK 한국공학한림원

청년공학

Journal of Youth Engineering

제6집

2022

NAEK 한국공학한림원
The National Academy of Engineering of Korea

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

차례

발간사

지난 2015년 개정 교육과정을 통해 융합 교육과 프로젝트 학습이 활성화되었고, 교내 활동 중심의 학생부종합전형이 자리 잡았습니다. 4차 산업혁명의 물결을 타고 시작되는 2022년 개정 교육과정에서는 수월성을 추구하는 교과목(수학, 국제, 과학계열의 전문교과목, 과제 연구 등)과 프로젝트 활동이 더욱 확산되었습니다.

이러한 사회적 흐름에 발맞추어 전국의 많은 고등학생이 의욕적으로 탐구를 하면서 보고서(논문)를 쓰고 있지만 대부분 교내 연구 발표 대회 자료집 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 엔지니어를 꿈꾸는 고등학생들이 공학 논문을 발표할 수 있는 학술지를 만들어 교육현장의 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 대부분 우수한 공과대학에 진학하여 엔지니어의 꿈을 키우고 있습니다. 이번에 출판하는 <청년공학> 제6집 역시, 엔지니어를 꿈꾸는 고등학생들의 참신한 탐구 성과물을 담았습니다.

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.

2022년 8월

한국공학한림원 회장

권오경

1

대상 논문

Journal of Youth Engineering

01 Canny Image와 CNN, 최소제곱법을 이용한 자주차의 주행 및 보조 알고리즘 연구

하나고등학교
정윤재, 김민준, 이준석

Canny Image와 CNN, 최소제곱법을 이용한 자주차의 주행 및 보조 알고리즘 연구

Study of Jajucha Driving Algorithm using Canny Image, CNN, and LSM

정윤재*, 김민준, 이준석
Yoonjae Joung*, Minjun Kim, Junseok Lee

요약

본 논문에서는 Canny image 좌표분석과 CNN을 이용해 도로 상황을 분석하고 모터를 제어하는 자율주행 알고리즘과 장애물에 대응하거나 자율주행의 정확도를 높이는 보조 알고리즘을 제시한다. Canny image 좌표분석에서는 canny image와 grid를 통해 도출된 좌표들을 이용해 도로 상황을 분석하며, CNN에서는 Adam optimzer를 통해 도로 상황을 분석한다. 모터 제어는 도로 상황 분석의 결괏값에 따라 속도 제어와 조향 제어로 나뉘어 이루어지며 장애물 대응 알고리즘은 Lidar를 이용한다. 경로 추적과 turtle을 이용한 지도 생성 알고리즘을 통해 반복되는 주행의 효율성을 높이며, 근사 함수를 통한 주행 보조 알고리즘을 이용해 기울기, 함숫값, 곡률을 계산함으로써 주행의 정확도를 높인다.

Keywords: canny image, CNN(Convolutional Neural Network), Lidar, 경로 추적, 최소제곱법, 지도 생성, 옌센 부등식

I. 서 론

자율주행자동차의 구동을 위해서는 안전하고 효율적인 알고리즘 개발이 필수적이다. 따라서 본 논문에서는 canny image 좌표분석 기반 주행 알고리즘과 CNN을 이용한 도로 상황 판단 알고리즘, 그리고 이를 보완하는 형태의 보조 알고리즘을 새로이 제안한다.

본론은 다음의 5개 장으로 이루어져 있으며, (1)주행 상황 분석, (2)모터 제어, (3)장애물 인식, (4)상황 기록 및 mapping, (5)주행 보조 방안 제시 자세하게는 다음과 같다. 자주차는 주행을 위해 먼저 위치한 도로의 상황을 분석하고, 분석한 상황에 맞게 모터를 제어해 구동한다. 이때 본 논문에서는 (1)도로 상황 분석을 위해 canny image와 grid에서 도출된 좌표를 분석하는 방법과 CNN(Convolutional Neural Network)을 이용하는 방법 두 가지를 제시하고 이를 기반으로 자주차의 (2)모터를 제어하는 방법을 소개한다. 더 나아가, (3)차단기와 같은 장애물을 인식하는 예외 상황에서의 주행 알고리즘, 효율적인 반복 주행을 위해 주행이 끝난 후 실시되는 (4)주행 추적 및 지도 생성 알고리즘을 소개하고자 한다. 마지막으로는 canny image에서 도출된 좌표들을 최소제곱법을 이용하여 도로의 개형을 근사시킨 함수를 통해 자주차의 (5)주행 상황 분석 및 모터 제어 보조 방법을 제안하고자 한다.

II. 주행상황 (도로 종류) 분석

1. 주행 상황(도로)의 분류

주행상황은 차선 종류에 따라 5가지(dead_end, t-turn, left, right, straight)로 기본 분류를 한 후 left와 right의 경우 곡선 곡률에 따라 left90, leftnorm, right90, rightnorm으로 세부 분류를 한다. 이후 주행 상황 분석 알고리즘들에 사용되는 분류별 도로 조각 모양은 다음과 같다.

그림 1. dead_end

그림 2. t-turn

그림 3. left90

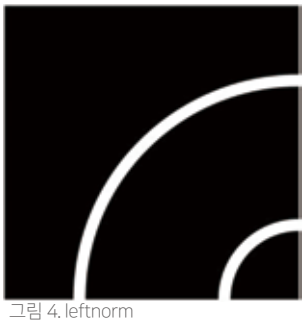


그림 4. leftnorm

* 정윤재 (하나고등학교, andrew05.joung@gmail.com)
김민준 (하나고등학교, mywhitewolf35@gmail.com)
이준석 (하나고등학교, seoki3937@gmail.com)

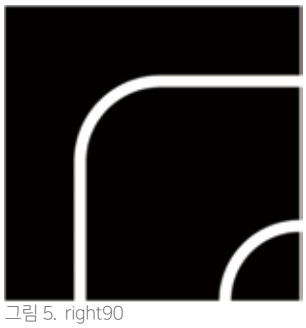


그림 5. right90

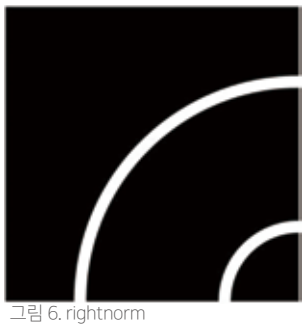


그림 6. rightnorm

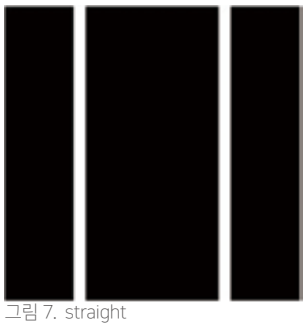


그림 7. straight

2. Canny Image 좌표분석 방식

Canny Image 좌표분석은 왜곡 처리를 마친 자주차로부터 수신한 이미지의 왜곡을 해소한 다음 Canny를 이용해 테두리들을 흰색, 배경을 검은색으로 변환한 canny image를 사용한다. Canny Image에서 이미지의 하단부(절반 아래)에 가로 7개, 세로 3개 선으로 이루어진 grid를 그리고, 그 grid와 차선의 테두리가 만나는 점들의 좌푯값을 구한다. grid의 세로선과 차선이 만나는 점들을 왼쪽에서부터 V[0] ~ V[6]으로 지정해 이미지 하단에서부터의 픽셀 거릿값을 저장한다. grid의 가로선과 차선이 만나는 점들은 이미지 중심 기준 좌측에서 만나는 경우 L[0] ~ L[2], 우측에서 만나는 경우 R[0] ~ R[2]로 지정하고 이미지 중앙에서 왼쪽 또는 오른쪽으로의 픽셀 거릿값을 저장한다. 이때 V[0] ~ V[6]의 값들이 특정 조건을 만족하는지 분석하여 주행 상황을 결정하는 방식이다. 분석의 용이성을 위해 V[0]~V[2]를 VL, V[4]~V[6]을 VR로 지칭한다. 그리고 V[0]~V[2] 중 최대, 최소를 각각 VLMax, VLMin이라고, V[4]~V[6] 중 최대, 최소를 각각 VRMax, VRMin이라고 지칭한다. 또한, VLavr와 VRavr는 아래와 같이 정의한다.

$$VLavr = \frac{V[0]+V[1]+V[2]}{3}$$

$$VRavr = \frac{V[4]+V[5]+V[6]}{3}$$

2.1 dead_end 분류

dead_end 상황의 경우 다음과 같은 사진이 수신된다.


그림 8. dead_end 수신 사진

이 상황에서는 R[2], L[2] 값이 감지되지 않으며, $VLavr$ - V[3], $VRavr$ - V[3] 이 매우 작다. 즉, 이 두 가지 조건을 만족하면 case는 dead_end로 정의된다.

2.2 t-turn 분류

t-turn case에서는 옌센 부등식을 활용한다.

옌센 부등식은 열린 구간 (a, b)의 함수 $f:(a,b)\to R$ 이고 실수 $x_1, \ldots, x_n \in (a,b)$이며 음이 아닌 실수 $p_1, \ldots, p_n \in [0,1]$ $(p_1+\ldots+p_n=1)$가 주어졌다고 하자.

이때 옌센 부등식에 의하면 오목 함수는 아래의 수식을 만족한다.

$$f(p_1x_1+p_2x_2+\ldots+p_nx_n) \le p_1f(x_1)+p_2f(x_2)+\ldots+p_nf(x_n)$$

[그림 2]와 같이 우회전이 포함된 t-turn 구간은 옌센 부등식에서 $p_i=\frac{1}{3}$로 다음의 식을 만족해야 한다.

$$\frac{V[4]+V[5]+V[6]}{3} \ge V[5]$$

하지만 위 조건은 straight일 때도 만족할 가능성이 존재하기 때문에 두 경우를 구분해줄 필요가 있다. 직진 상황에서는 도로의 기울기가 매우 가파른 직선이므로 V[4]와 V[6]의 차이가 매우 크고 t-turn에서는 이와 반대로 V[4]와 V[6] 차이가 작다는 점을 이용하여 V[4] - V[6]의 값을 두 가지 상황을 구분하는 요소로 사용한다.

따라서 위 옌센 부등식을 이용한 조건을 만족하고 V[4] - V[6]의 절댓값이 15보다 작을 때 t-turn 상황임을 알 수 있다. 본 논문에서는 [그림 2]와 같이 t-turn이 우회전인 경우만 고려하였으므로 이때의 case를 t_right으로 정의하였다.

2.3 straight 분류

직진 상황은 dead_end와 t-turn이 아니고 VLMin이 VRMax보다 작거나 같고 VLMax이 VRMin보다 크거나 같을 때이다. 위 조건을 만족할 때 case를 straight로 정의한다.

또, right나 left의 상황에서 R[2] 또는 L[2] 값이 감지되며 그 값이 큰 경우, 이를 straight 상황에서 자주차가 좌 또는 우로 조금 틀어진 경우로 인지하고 case를 straight로 정의한다.

2.4 left, right 분류

left 상황의 경우 다음과 같은 사진이 수신된다.

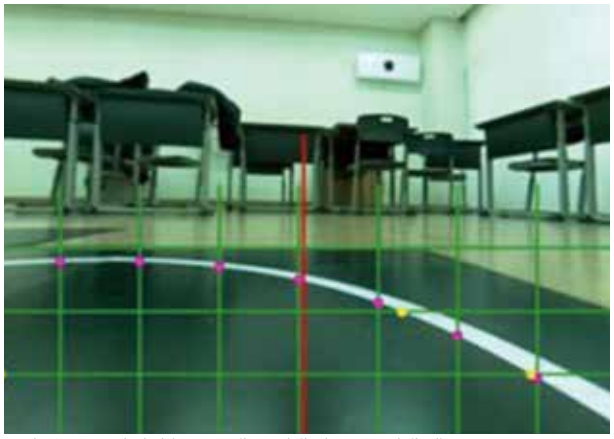


그림 9. left 수신 사진 (grid: 녹색, V: 적색 점, L,R: 노란색 점)

이 상황에서는 VL의 모든 값이 VR보다 크다. 따라서 VLMin이 VRMax보다 크다면 case를 left로 정의한다.

right 상황은 left 상황과 반대이므로 VRMin이 VLMax보다 큰 경우 case를 right으로 정의한다.

2.5 curve 정의 (90, norm)

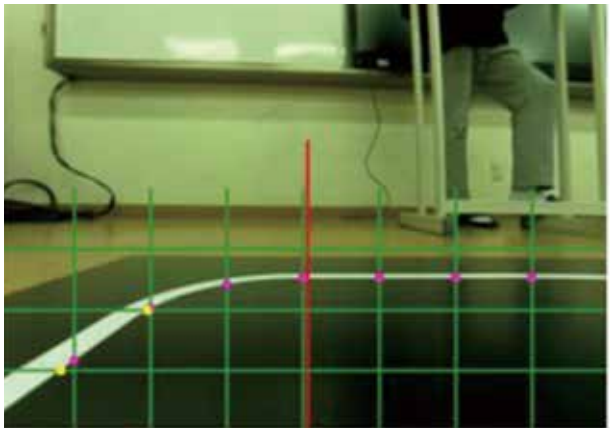


그림 10. 직각 커브 구간 수신 사진

case가 right 또는 left일 때 [그림 3], [그림 4] 또는 [그림 5], [그림 6]과 같이 각각 2가지 곡률의 도로가 존재한다. 직각 커브(90)[그림 10]과 일반 커브(norm)[그림 9]는 V[2] - V[4]의 값의 크기로 구별한다. 일반 커브에서는 이 값이 크게 나타나고, 직각 커브에서는 이 값이 작게 나타날 것이다. 이를 이용해 left90, leftnorm, right90, rightnorm의 상황을 추가로 정의하고, 이전의 left, right는 이용하지 않는다.

3. CNN(Convolutional Neural Network) 이용 방식

3.1 원리 소개

3.1.1 Perceptron 원리

인공신경망에서 단일 Perceptron은 여러 입력값(input 값)에 대해 각각의 input 값에 적절한 가중치(weight)를 부여하여 합한다. 계산된 합을 어떠한 활성화 함수 σ(activation function)에 대입하여 Perceptron의 output 값을 얻는다. 활성화 함수는 인공신경망 학습 용도에 적합한 함수를 사용해야 하고, 비선형적이어야 한다. 따라서 input 벡터를 X, weight 벡터를 W라고 하면 Perceptron의 output은 다음과 같이 계산된다. [1]

$$\hat{y} = \sigma(b + \sum_{i=1}^{n} x_i w_i) = \sigma(b + X^T W)$$

3.1.2 CNN 원리

그러나 사진의 픽셀들 데이터를 input 값으로 받아 처리하는 인공신경망은 이미지의 공간적 구조를 포함하여 전달해야 한다. 따라서 기존의 fully connected neural network를 사용하면 정확도가 높지 않다. 이러한 문제점을 보완하여 이미지 픽셀값들의 공간적 정보도 인공신경망에 같이 전달해 줄 수 있는 모델이 CNN (Convolutional Neural Network) 모델이다. [2] CNN 모델에서도 fully connected 모델이 존재하지만, fully connected 모델에서 데이터를 처리하기 전에 Convolution Layer와 Pooling 단계를 통해 이미지의 공간적 정보도 같이 저장한다. Convolution Layer는 특정 filter를 통해 입력받은 이미지의 특징적인 부분을 부각하는 역할을 한다. 입력받은 input 이미지의 특정 부분을 filter의 아다마르 곱 (element-wise product) 이후 feature map에 표시한다.

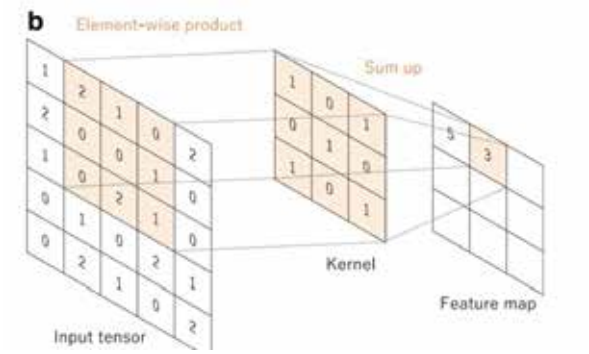


그림 11. Convolutional Layer [3]

이 과정을 수학적으로는 다음과 같이 기술할 수 있다.

$$y_{ab} = \sigma\left(b + \sum_{i=0}^{k_i} \sum_{j=0}^{k_j} w_{ij} x_{(a+i)(b+j)}\right)$$

Pooling 작업은 이미지의 크기를 줄여주는 과정이다. 이미지의 크기를 줄이면 데이터 손실이 나기 마련인데, 이러한 부분을 보완하기 위해 Average, Max, Mixed, Lp, Stochastic, Spatial Pyramid 등의 Pooling 기법이 존재한다. 본 논문에서는 Max Pooling을 사용하였다.

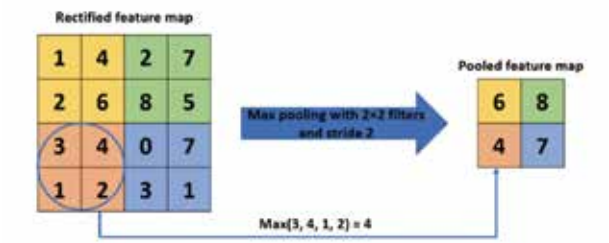


그림 12. Max Pooling [4]

이렇게 Convolution Layer와 Pooling 과정을 통해 크기는 줄이고, 공간적 정보를 저장한 픽셀 데이터값들은 일렬로 정렬되어 일반적인 Fully Connected Neural Network의 구조를 갖춘다. CNN의 마지막 node는 인공신경망 모델로 얻고자 하는 결과물에 따라 다르게 설정한다.

3.2 주행 상황 판단을 위한 CNN 네트워크 설계

Canny Image 좌표분석 방식에서 자주차로부터 수신한 이미지를 canny image로 변환하고, case를 정의하는 작업을 거쳐 데이터를 모은 다음, CNN을 통해 학습시켰다. 이때 도로 상황만을 분석하기 위해 이미지의 하단부를 ROI (Region of Interest)로 지정하여 320 × 90(픽셀)의 이미지로 변환하였다.

그림 13. CNN 모델 학습에 사용된 데이터 예시, (320 × 90 pixels)

위 그림과 같이 canny image로 변환된 데이터를 모으고, case마다 label을 지정해 분류 및 저장하였다.

표2. CNN 모델 학습에 사용된 데이터와 개수, 총 5,323개의 데이터를 사용하였다.

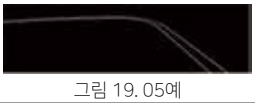
Label	데이터 예시	상황 분류	데이터 개수 (개)
00	 그림 14. 00예	dead_end_right	185
01	 그림 15. 01예	straight	1,905
02	 그림 16. 02예	rightnorm	1,338
03	 그림 17. 03예	leftnorm	725
04	 그림 18. 04예	right90	453
05	 그림 19. 05예	left90	329
06	 그림 20. 06예	t_right	388
합계			5,323

표3. CNN 모델 학습을 위해 train_set과 test_set을 0.2 비율로 나누어준 결과

train_test_split : 0.2	총 데이터 : 5,323개	train_set	4,258개
		test_set	1,065개

위의 데이터를 학습시킬 CNN 모델은 Python Tensorflow를 사용하여 Google Colaboratory에서 구현했다. CNN 모델의 코드는 다음과 같다.

CNN 모델의 구조는 다음과 같다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 88, 318, 8)        80
_________________________________________________________________
max_pooling2d (MaxPooling2D  (None, 44, 159, 8)        0
)
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 42, 157, 16)       1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling  (None, 21, 78, 16)        0
2D)
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 19, 76, 32)        4640
_________________________________________________________________
flatten (Flatten)            (None, 46208)             0
_________________________________________________________________
dense (Dense)                (None, 64)                2957376
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 455
=================================================================
Total params: 2,963,719
Trainable params: 2,963,719
Non-trainable params: 0
_________________________________________________________________
None
```

그림 21. CNN 모델의 메인 코드 요약.

Conv2D Layer(Convolutional Layer)가 3개, MaxPool Layer가 2개 존재한다. 모든 Conv2D Layer의 활성화 함수는 ReLU를 따른다. CNN 모델은 마지막에 두 개의 Fully Connected Neural Network를 통해 output을 출력한다.

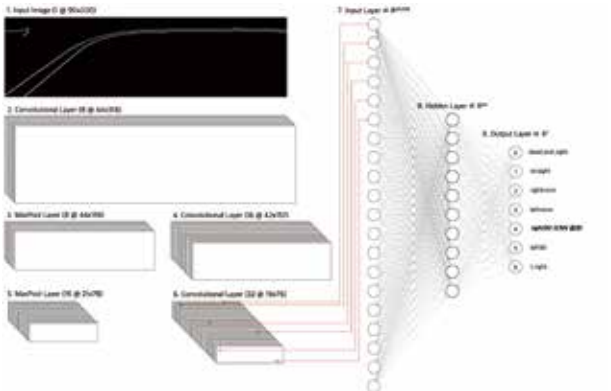


그림 22. [그림 21]를 시각화한 CNN 구조도. right90 이미지를 수신한 경우이다.

또한 CNN 모델에서 사용한 Convolution Layer의 활성화 함수 “ReLU”와 마지막 Dense Layer에서 사용한 활성화 함수 “softmax”는 다음과 같다.

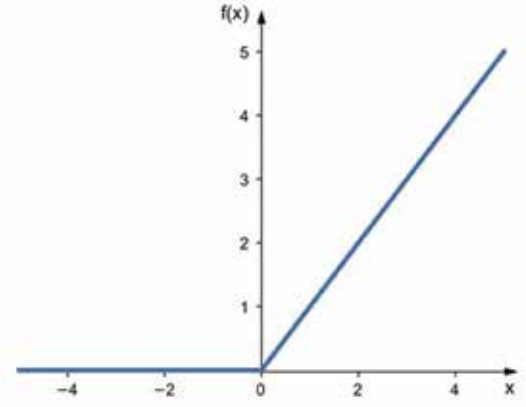


그림 23. ReLU 함수

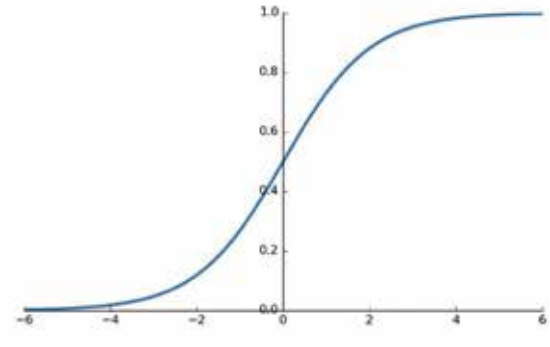


그림 24. Softmax 함수

위의 CNN 모델은 어떤 사진을 입력값으로 받고, 그 사진의 데이터를 Conv2D Layer와 Pooling Layer, Dense Layer를 거치게 한다. 마지막 7개의 node는 해당 사진의 종류가 무엇인지를 분류하는 기능을 한다. 즉, 7개의 node는 각각 하나의 label에 대응된다. 다음은 임의의 사진 하나에 대해서 CNN 모델을 실행한 결과이다.

```
array([[0.1412761 , 0.14438805, 0.13874161, 0.14487654, 0.14496654,
        0.14367197, 0.14207919]], dtype=float32)
```

그림 25. CNN 모델 실행 결과

위의 그림과 같이 CNN 모델은 7개의 수를 return 한다. 각각의 값은 표2에 해당하는 label에 대해서 CNN 모델이 예측한 확률을 뜻한다. 이때, 위의 7개 수를 합한 값은 항상 1이 나온다. 그리고 가장 확률값이 크게 나온 label을 CNN 모델의 최종 예측 결과로 정의하여 출력한다.

3.3 CNN 모델의 학습

인공신경망에서 신경망은 자신이 예측한 결괏값과 실제 결괏값을 비교하고, 손실함수를 통해 예측한 값과 실제값의 차이를 구한다. 따라서 인공신경망의 목적은 손실함수를 최소화하여 신경망의 예측값과 실제값이 거의 일치하도록 만드는 것이다. 이때 대부분의 인공신경망에서는 다음과 같은 과정을 통해 손실함수를 최소화한다.

본 논문에서는 CNN 모델의 손실함수로 다중 분류 모델 중 가장 효과적인 Sparse Categorical Cross-entropy를 사용했다.

$$Loss = -\frac{1}{N}\sum_{j=1}\sum_{i=1} t_{ij}\log(y_{ij})$$

CNN 모델은 위의 손실함수를 최소화해야 한다. 이때 모델을 update 시키기 위한 optimizer를 사용한다. Optimizer의 가장 대표적인 예인 gradient descent는 조작 가능한 가중치 변수들에 대해서 연쇄법칙을 이용해 가중치 변수들의 변화로 인한 손실함수의 변화를 계산하는 방식이다. 이러한 gradient descent는 다음의 수식으로 나타낼 수 있다. [5][6]

$$\theta_{t+1} = \theta_t - \alpha\nabla L(\theta_t) \quad (\theta : parameter,\ L : loss\ function)$$

위 식에서 는 learning rate로 한번 model을 update 할 때 변화시키는 정도를 의미한다. 적합한 optimizer 함수를 찾기 위해 [표 4]에서와 같이 4개의 optimizer의 learning rate를 각각 10^{-3}과 10^{-4}로 2가지로 하여 총 8개 모델의 정확도를 측정하였다. 다음 표는 CNN model을 훈련하여 얻은 정확도를 비교한 것이다. 모든 model은 Google Colaboratory의 GPU로 학습시켰다. [7]

표4. 8개의 서로 다른 optimizer의 성능 비교

Optimizer (learning rate)	Train accuracy (epoch = 5)	Test accuracy
SGD (10^{-3})	0.5908	0.6784
Adam (10^{-3})	**0.9923**	**0.9957**
Adagrad (10^{-3})	**0.9083**	**0.9166**
Adadelta (10^{-3})	0.4552	0.4512
SGD (10^{-4})	0.3752	0.3701
Adam (10^{-4})	**0.9784**	**0.9861**
Adagrad (10^{-4})	0.4913	0.4935
Adadelta (10^{-4})	0.3729	0.4033

위의 표를 통해서 가장 효과적인 optimizer 모델이 Adam (learning rate = 10^{-3}, 10^{-4})과 Adagrad(learning rate = 10^{-3})임을 알 수 있다. 특히 Adam은 learning rate = 10^{-4}일 때도 매우 높은 정확도를 기록했기 때문에 본 연구에서는 optimizer 함수를 Adam으로 설정하고 learning rate는 10^{-3}으로 설정하였다.

다음은 CNN model의 학습 코드이다.

```
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

그림 26. CNN model 컴파일 과정, optimizer는 Adam, learning_rate=10^{-3}, 손실함수는 sparse_categorical_crossentropy 함수를 사용하였다.

```
BATCH_SIZE = 50
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EPOCHS)
```

그림 27. CNN model 학습을 위해 EPOCH = 5로 설정했다.

학습 과정은 다음과 같이 진행되었다.

그림 28. training set에 대한 학습 과정

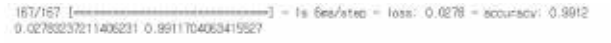


그림 29. test set 학습 과정

test set에서 모델을 테스트한 결과 정확도는 99% 정도로 overfitting 되지 않았음을 알 수 있다.

CNN 모델이 결과를 예측하는 모습을 시각화하면 다음과 같다.

그림 30. Adagrad optimizer(learning rate: 10-3)의 예측 결과 (파란색은 올바르게 예측한 경우, 빨간색은 잘못 예측한 경우이며 프레임마다의 캡션은 순서대로 예측 label, 예측 label일 확률, (실제 label)이다.)

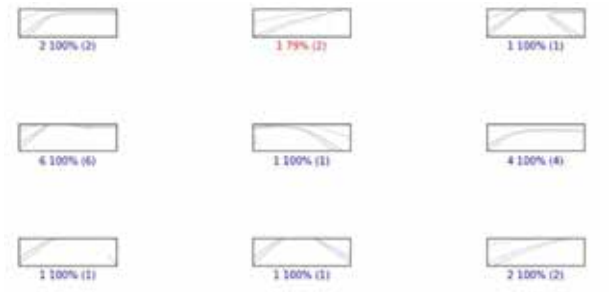


그림 31. Adam optimizer(learning rate: 10-3)의 예측 결과 (파란색은 올바르게 예측한 경우, 빨간색은 잘못 예측한 경우이며 프레임마다의 캡션은 순서대로 예측 label, 예측 label일 확률, (실제 label)이다.)

첫 번째 사진은 Adagrad optimizer, 두 번째 사진은 본 연구에서 사용한 Adam optimizer이다. Adam optimizer가 매우 높은 정확도를 보이고 있음을 알 수 있다. 학습된 모델을 저장하여 다음과 같이 자주차 main_grid.py에서 실행하여 주행상황을 분류할 수 있다.

```
def process(self, t, frontImage, rearImage, frontLidar, rearLidar):
    start1 = time.time()
    canny = self.canny(frontImage)
    scaleX = 0.5
    scaleY = 0.5
    cropCanny = canny[300:480,0:640]
    img = cv2.resize(cropCanny, None, fx=scaleX, fy=scaleY, interpolation = cv2.INTER_LINEAR)
    cv2.imshow("cropped", img)
    image = (np.expand_dims(img, axis=-1)/255.).astype(np.float32)
    image = np.expand_dims(image, axis=0)

    predict = self.model.predict([[image]])
    result = np.argmax(predict)
```

그림 32. CNN 모델을 저장하여 자주차 파일에서 실행시킨 코드

따라서 위 코드를 사용하여 주행상황을 분석하면 다음과 같은 결과가 나온다.

그림 33. CNN에 전달한 canny image 사진

```
--------------------
CNN Result - |time: 0.0829372
  0 dead_end_right :    4.3424887e-05
  1 straight :          0.00017777276
  2 rightnorm :         1.8666775e-06
  3 leftnorm :          1.7859442e-09
  4 right90 :           0.9997688
  5 left90 :            7.978848e-06
  6 t_right :           1.2971132e-07
  Prediction :  right90

Algorithm Result - |time: 0.0110035
  Prediction : right90
--------------------
```

그림 34. CNN 모델이 제시한 결과와 기존의 알고리즘이 제시한 결과이다. CNN 모델은 입력받은 사진이 각각의 상황일 확률을 같이 출력한다. (일반적으로 CNN 모델의 예측 시간이 더 길다.)

III. 모터 제어

1. 주행 변수 설정

본 알고리즘은 프레임마다 메서드가 한 번씩 실행되고, 기존 변수들은 초기화된다. 따라서 속도나 후진 및 커브 탈출을 정의해주는 변수의 경우 인스턴스 변수를 사용해 새로운 메서드를 실행할 때 초기화되지 않도록 하였다.

자주차의 모터 제어는 크게 속도 제어와 조향 제어로 나뉠 수 있다. 프레임마다 정의된 속도로 메인 모터가 움직이고, 정의된 조향 값으로 서브 모터가 움직여 자주차가 주행하는 방식이다.

2. 속도 제어

자율주행 상황에서 목표 지점에 도달하는 데 걸리는 시간을 최소화하기 위해서는 기본적으로 자주차의 속도를 높이면 된다. 하지만 커브 또는 특수한 상황에서 자주차가 최대 속도를 유지하면 사고가 발생할 가능성이 크다. 따라서 본 주행에서는 가변 알고리즘을 적용하여 상황별 자주차의 속도를 달리하였다.

자주차의 속도는 인스턴스 변수 self.vars.velocity로 정의하고 프레임마다 변수 velocity로 불러와 사용하였다.

2.1 가변 속도 범위

속도를 제어할 때는 사용자가 지정한 최댓값과 최솟값 범위 내에서 가변하였다.

2.2 case가 straight인 경우

직선 도로에서는 가속하기 위해 다음 과정을 수행한다.

정면 도로 경계선과의 거리를 의미하는 V[3]이 감지되지 않거나 거리가 충분히 멀 때, 자주차의 속도를 일정하게 증가한다. 반대로 경계선이 인식되고 그 거리가 작을 때 자주차의 속도를 일정하게 줄인다.

2.3 case가 straight가 아닌 경우

직선 도로가 아닌 경우, 즉 커브 상황에서 직선 도로와 마찬가지로 자주차가 빠른 속도의 주행 또는 가속 주행을 한다면 목적지 도달 시간을 감소할 수 있다. 하지만 실제 상황에서는 컴퓨터와의 통신 지연으로 인해 자주차의 실시간 이미지 송신과 컴퓨터의 수신 사이에 시간 차이가 발생한다. 따라서 커브 탈출을 정상적으로 수행하기 어렵다. 그러므로 이 경우에는 자주차의 속도를 일정하게 줄인다.

2.4 가변 속도 순서도

위 가변 속도 알고리즘을 순서도로 나타내면 다음과 같다.

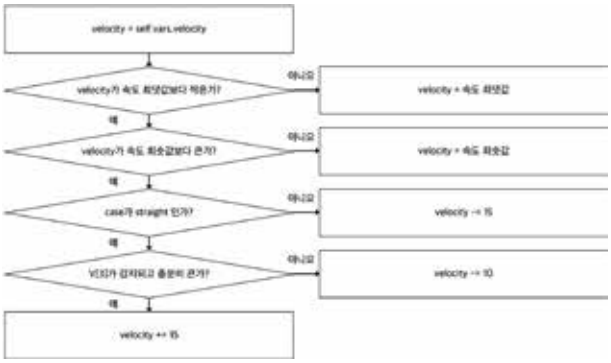


그림 35. 가변 속도 순서도

3. 조향 제어

조향 값은 self.vars.steer를 통해 저장하고 프레임마다 변수 e를 통해 정의하였다.

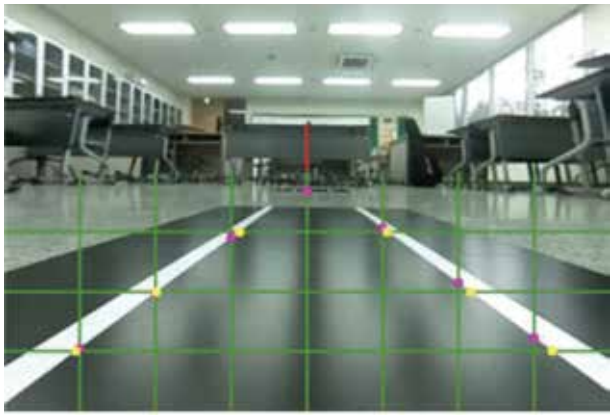


그림 36. straight에서 자주차 수신 이미지 (grid 적용), 이미지 좌측에서 차선과 grid의 가로선이 만나는 점은 L, 우측에서 만나는 점은 R이고, 위에서부터 순서대로 0, 1, 2이다.

조향할 때는 [그림 36]에서 노란색 점인 L[0]~L[2], R[0]~R[2]를 이용해 제어한다.

3.1 straight 조향 제어

자주차가 straight 차선의 중앙에 위치할 때, 즉 e가 0이어야 할 때, R[2]는 250이 측정된다. R[2]가 이보다 크다면 자주차가 차선의 왼쪽에 치우쳐져 있다는 뜻이므로 e > 0이어야 하고, 반대로 작다면 자주차가 차선 오른쪽에 치우쳐져 있다는 뜻이므로 e < 0이어야 한다. 따라서 이때는 e를 e = R[2] - 250으로 정의한다.

또, case가 right이거나 left이더라도 커브를 거의 탈출했다면 straight 상황에서 차량이 조금 오른쪽 또는 왼쪽으로 치우쳐져 있는 상황과 같다. 이때 R[2] 또는 L[2]가 감지되기 시작하고 그 값이 적당히 크므로(R[2] > 100 또는 L[2] > 100) straight로 case를 변경해 straight처럼 조향을 제어한다.

3.2 right 조향 제어

3.2.1 rightnorm

rightnorm 상황에서는 R[2]가 감지되지 않는다. 대신 L[2]가 감지되므로 이를 이용해 회전할 수 있다. straight와 비슷하게 차량이 도로 중앙에 위치할 때의 L[2] 값을 기준으로 L[2] 값이 이보다 더 크면 왼쪽으로, 더 작으면 오른쪽으로 주행해야 하므로 e를 e = 260 − L[2]로 정의한다.

3.2.2 right90

right90은 rightnorm과 같게 주행했을 경우 갑자기 정면에 차선이 생겨 회전할 수 없는 상황이 발생한다. 따라서 자주차가 정면 경계선에 도달하기 전에 미리 회전하고, 거의 탈출했을 때 straight 조향 제어를 해줄 수 있다. 이때는 e = L[2] - 70으로 e를 정의한다.

3.3 left 조향 제어

left 상황에서의 조향 제어는 right와 그 방법이 같지만, 방향만 반대로 하여 e를 정의하였다.

3.4 dead_end, t-turn 조향 제어

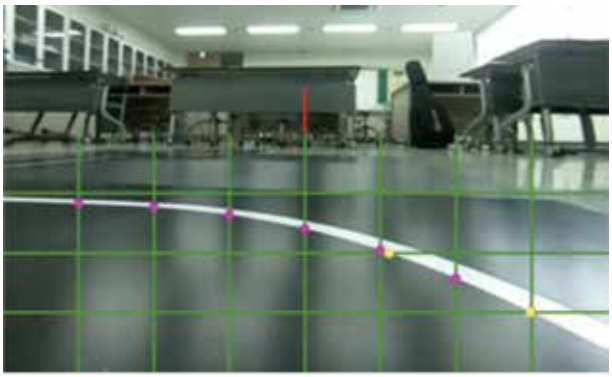


그림 37. leftnorm 자주차 수신 이미지

위 그림과 같이 앞서 서술한 case 들은 회전 시 조향 정의에 사용할 수 있는 좌, 우 또는 앞의 경계선이 존재한다. 하지만, dead_end 또는 t-turn의 최소 하나의 경계선이 존재하지 않아 조향 정의에 어려움이 있다. 따라서 이때에는 회전이 종료될 때까지 인스턴스 변수인 self.curve_loop을 사용해 회전 루프를 실행한다.

먼저 dead_end_left 또는 t_left의 경우 self.curve_loop을 1로, dead_end_right 또는 t_right의 경우 2로 지정한다. 그리고 self.curve_loop이 1이면 e를 -200으로, 2이면 +145로 정의해 주행한다. 그러다 self.curve_loop이 1이면 case가 다시 straight가 되고 V[3]과 L[2]가 감지될 때, 2이면 case가 다시 straight가 되고 V[3]과 R[2]가 감지될 때, self.curve_loop을 0으로 정의하여 루프를 탈출한다.

이를 순서도로 나타내면 다음과 같다.

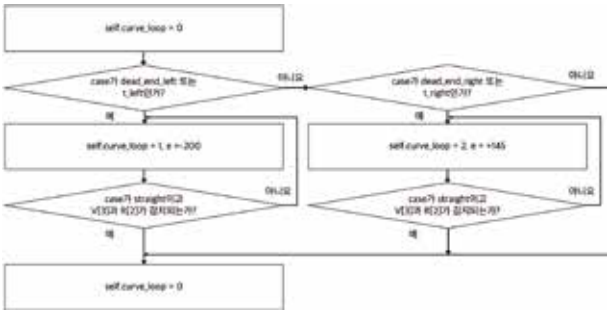


그림 38. dead_end / t-turn 상황에서의 회전 루프 순서도

IV. 장애물 인식 (특수 case)

1. 차단기 알고리즘

앞서 서술한 상황에 포함되지 않는 특수한 상황에서는 특별한 알고리즘이 필요하다. 예를 들어 장애물이 인식되는 경우를 들 수 있다. 이때는 Lidar 센서를 사용해 차단기를 인식하고, 주변의 양쪽 차선에 대한 정보를 바탕으로 주행상황을 판단해 모터를 제어한다. 본 논문에서는 차단기가 있는 상황에서 왼쪽 또는 오른쪽에 경로가 존재하는 경우를 차단기 회전, 경로가 없는 경우를 차단기 정지로 구분하여 설명한다. 차단기가 있는 상황에서 양쪽 경로가 모두 존재하는 경우는 dead_end와 같다. 즉, 자주차가 어떤 경로를 선택할 것인지 미리 사용자가 입력해주거나 임의(무작위)로 결정해야 하는 상황이 생기므로 본 논문에서는 고려하지 않았다.

1.1 차단기 감지

자주차와 차단기 사이의 거리에 대한 Lidar 측정값이 충분히 작을 때, 이 차단기가 유의미한 거리에 감지되었다고 분류한다.

1.1.1 차단기 회전

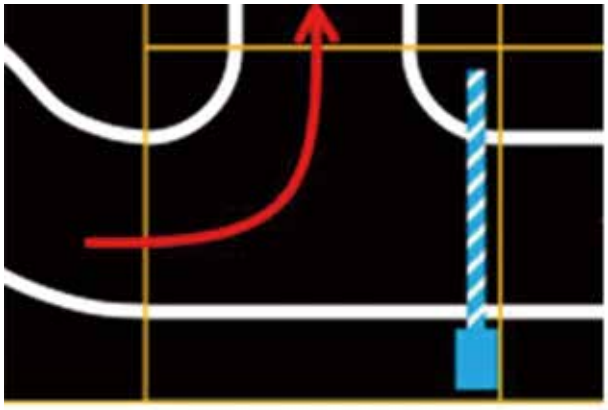


그림 39. 차단기 회전 상황에서 자주차의 진행 경로

위 그림과 같이 왼쪽에 경로가 존재하면 L[2]가 감지되지 않는다. 이때는 case를 barrier_left로 정의한다.

반대로 오른쪽 경로가 존재하면 R[2]가 존재하지 않는다. 이때는 case를 barrier_right로 정의한다.

1.1.2 차단기 정지

차단기가 감지되었으나 차단기 회전 상황이 아닌 경우(다른 경로가 존재하지 않는 경우) case를 barrier_wait로 정의한다.

1.1.3 차단기 감지 순서도

차단기 회전 및 정지 case 분류 과정을 순서도로 정리하면 다음과 같다.

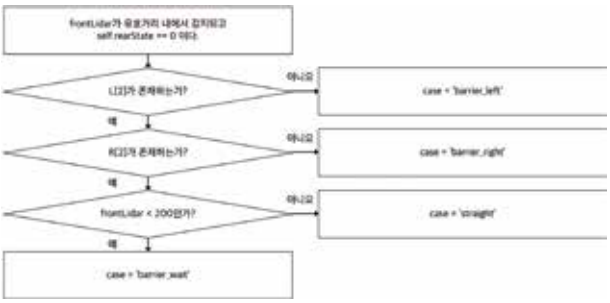


그림 40. 차단기 감지 case 순서도

1.2 차단기 모터 제어

case가 barrier_left 또는 barrier_right인 경우, 경로 진입이 완료될 때까지 곡선 주행을 수행해야 하며, barrier_wait인 경우, 정지해 기다려야 한다. 이때 모터 제어 방식은 dead_end 또는 t-turn의 상황과 유사하며, 다음과 같다.

1.2.1 차단기 회전 모터 제어

자주차 제원상 Lidar는 자주차 정면 물체와의 거리만을 측정할 수 있다. 즉, [그림 39]와 같이 곡선 주행 이후 차단기가 있는 상황에서는 Lidar를 통해 차단기를 감지했을 때 충분한 회전 거리를 확보하기 어렵다.

따라서 이때에는 후진을 통해 먼저 회전 거리를 확보한다. 먼저 case가 barrier_left의 경우 다음과 같다. 차단기 회전 상황이 시작되면 값이 0이던 인스턴스 변수 self.rearState을 3으로 변경하고, 회전하고자 하는 방향과 반대 방향으로 (시계 방향 회전을 하고자 하는 경우 반시계 방향으로) 조향을 설정해 후진한다. 차단기와의 거리(Lidar 측정값)가 충분히 멀어지면 self.rearState을 1로 지정한다. self.rearState가 1이 되면 회전을 시작한다. 그러다 다음 조건을 만족하면 회전을 멈추고 self.rearState을 0으로 지정해 차단기 회전 상황을 탈출하여 다시 일반적인 주행 상황 분석을 시작한다. case가 straight가 되어야 하고 L[2]가 감지되어야 하며, 회전 도중 잘못된 차선이 인식되어 위 조건을 만족하는 것을 방지하기 위해 V[3]이 VLMax보다 커야 한다.

case가 barrier_right인 상황의 경우 barrer_left와 같지만, 회전 상황이 시작될 때 self.rearState를 4로 변경해 후진하고, 차단기와의 거리가 충분히 멀어졌을 때 self.rearState를 2로 설정하여 회전을 시작한다는 점에서 다르다. 또, 직선 도로에 진입해 case가 straight이고 R[2]가 감지되며, V[3]이 VRMax보다

커져 회전 상황을 탈출해야 하는 경우 self.rearState를 0으로 지정한다.

이를 순서도로 나타내면 다음과 같다.

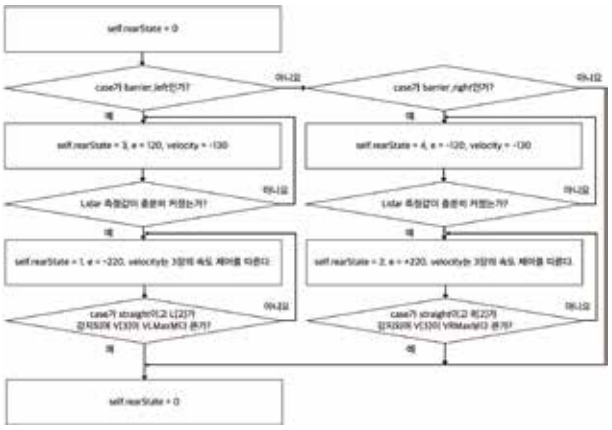


그림 41. 차단기 회전 모터 제어 순서도

1.2.2 차단기 정지 모터 제어

차단기 정지 상황에서는 3장에서의 속도 제어를 사용한다. case 문자열을 '_'로 구분했을 때 wait로 끝나면, 속도 제어 부분에서 추가로 velocity = 0을 실시한다. 예를 들어 dead_end_right은 case문자열을 '_'로 구분했을 때 right로 끝나고, barrier_wait은 wait으로 끝난다.

V. 지도 생성

1. 자주차 지도 생성의 필요성

본 논문에서 소개한 알고리즘의 경우 자주차가 미래 도로의 상황 및 모습을 알지 못하는 상태에서 현재 센서값과 알고리즘, 또는 CNN 예측값을 통해 주행하는 방식이다. 하지만 실제 자율주행자동차는 주행에 있어 사용자의 GPS와 서버상 데이터를 비교하여 현재 및 미래의 도로 정보를 예측하여 목적지를 향해 주행할 수 있다. 어떤 경로가 제공되면, 자율주행자동차는 센서값이 아닌 주어진 경로를 따라 주행하는 것을 우선으로 하며 센서 측정값과 알고리즘을 통해서는 예외 상황에 대처하는 방식이다.

이와 같은 방식의 주행을 수행하기 위해서는 도로 상황에 대한 데이터가 있어야 하고, 각 도로 상황별로 자주차의 주행 방법이 사전 정의되어 있어야 한다. 데이터 획득을 위해 자주차는 센서와 알고리즘에 기반한 주행을 함과 동시에 도로 상황을 데이터베이스에 저장할 수 있다.

2. 자주차 경로 추적

자주차는 프레임마다 case를 정의하고 이에 맞게 모터를 제어한다. case가 달라질 때까지 속도가 100 이상인 모든 프레임의 case를 tracking 리스트에 저장하고, tracking 리스트의 길이를 한 개의 도로 조각을 지날 때 평균적으로 저장되는 case들의 수로 나누어 중복되는 도로 조각의 개수를 구한 다음 mapping 리스트에 case를 개수만큼 더한다.

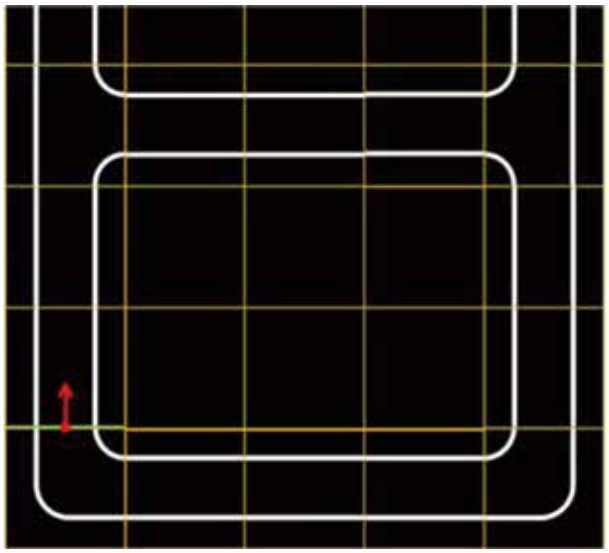


그림 42. 경로 추적 예시 트랙, 빨간색 화살표가 출발 지점이다.

이렇게 하면 [그림 42]와 같은 트랙을 주행한 다음 자주차는 [straight, straight, t_right, straight, straight, straight, dead_end_right, straight, straight, right90, straight, straight, straight, right90]과 같은 mapping 리스트를 얻을 수 있다.

3. 자주차 경로 시각화 (지도 생성)

mapping 리스트가 길어질 경우를 대비해 사용자 편의를 위해 추적 및 사전 정의된 경로의 모습을 시각화하여 보여주는 것도 매우 중요하다. 이는 turtle 모듈을 사용하여 수행하였는데, 자주차의 경로를 따라 좌표를 설정하고 도로 중심에서 turtle의 펜 모양을 도로 조각 모양으로 설정한다. 예를 들어 straight는 50 이동하고 도로를 그리고, right90은 50 이동하고 도로를 그린 다음 오른쪽으로 90도 회전한다.

이와 같은 방식을 통해 경로 추적이 종료된 시점, [그림 42]에서 추적된 mapping 리스트를 시각화하는 과정 및 결과는 다음과 같다.

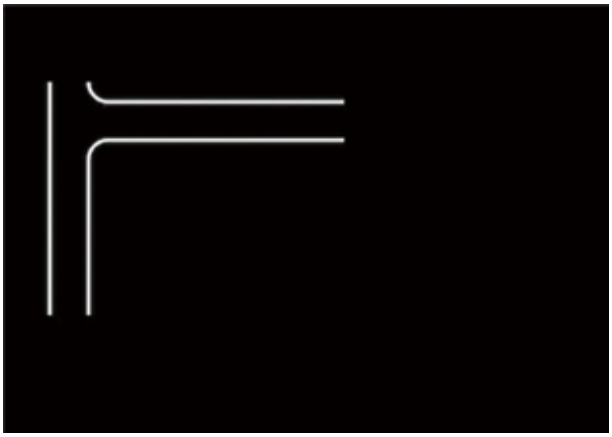


그림 43. 지도 생성 과정 1

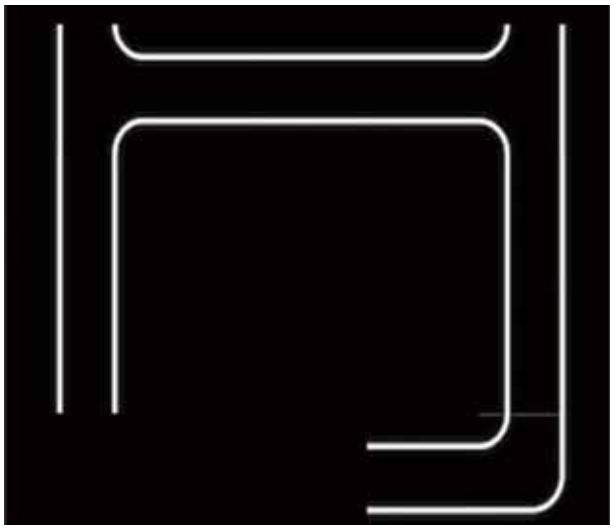


그림 44. 지도 생성 과정 2

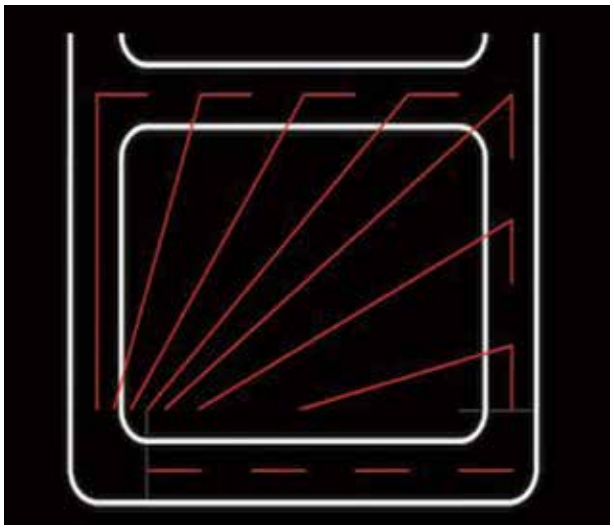


그림 45. 지도 생성 결과 (turtle pen 경로 표시)

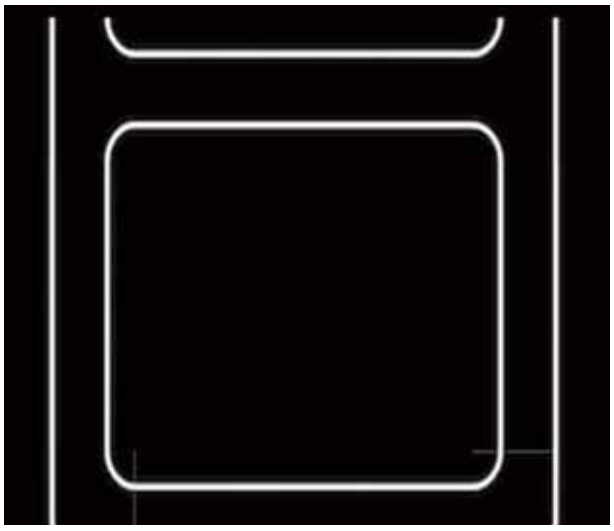


그림 46. 지도 생성 결과 (turtle pen up)

VI. 수학적 해석을 통한 자율주행 알고리즘 보조

1. 보조 알고리즘의 필요성

하나의 알고리즘에만 기반하여 자율주행자동차가 주행하면 case를 잘못 정의하거나 고려하지 못한 예외 상황이 발생하여 주행에 오류가 발생할 가능성이 크다. 따라서 여러 개의 보조 알고리즘이 존재하면 도로 이탈률을 줄이는 등 더욱 안전한 주행을 할 수 있을 것이다.

본 논문에서는 V값들을 최소제곱법을 이용해 하나의 함수로 근사함으로써 일부 case에서의 주행을 보조하는 방법을 제안하고자 한다.

2. V값의 확장

차선과 grid의 세로선이 만나는 점들의 수가 많을수록 정밀하고 경향성을 가지는 함수를 근사할 수 있다. 따라서 본 논문에서는 기존의 사용하던 V값 들을 K로 확장하여, 근사의 정확성을 높이고자 하였다. grid의 세로선의 수를 늘리는 것이다. 이때 세로선의 수는 12p + 7 (p는 자연수)로 일반화하여 정의하였다. 다만, 본 논문에서는 p = 8인 상황으로 연산을 수행한 결과만을 서술하였다.

또, 기존의 V값들을 이용한 주행을 위해서 V는 K를 7개로 등간격 배분하여 아래의 꼴로 K값과 대응을 시켜주었다.

V=[K[0], K[2p+1], K[4p+2], K[8p+4], K[10p+5], K[12p+6]]

3. 근사에 사용된 이론 (최소제곱법)

$i \in N_0\,(0 \le i \le 12p+7)$에 대해 모든 K[i]값을 최소제곱법을 사용하여 numpy.polyfit로 근사하였다.

근사 함수 $p(x)$에 대한 다음의 식에서 E가 최소가 되게 하는 함수 $p(x)$를 찾고자 하는 것이다. 이때 주행상황에 따라서 함수 $p(x)$는 여러 형태의 함수가 될 수 있다.

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

n차 다항식 $p_n(x)$는 아래와 같다.

$$p_n(x) = \sum_{k=0}^{n} a_k x^k = a_0 + a_1 x + \ldots + a_n x^n$$

E는 자주차에서 아래의 식으로 나타낼 수 있다.

$$E = \sum_{i=0}^{12p+6} y_i - p_n(x_i)^2 = \sum_{i=0}^{12p+6} (y_i - \sum_{k=0}^{n} a_k x_i^k)^2$$

따라서 이 E라는 함수는 $a_0, a_1, \ldots, a_n$에 의존하는, 즉 n+1개의 변수에 의존하는 함수임을 알 수 있다. 이때 이 함수가 최솟값은 존재하므로 $\frac{\partial}{\partial a_j} E(a_0, \ldots, a_j, \ldots, a_n) = 0$을 만족해야 한다. 따라서 이를 만족하는 식에 대하여 계산을 해주면 아래의 행렬식으로 E의 최솟값을 만족하는 $a_0, \ldots, a_j, \ldots, a_n$을 찾을 수 있다. [8]

$$\begin{pmatrix} \Sigma_{i=0}^{n} x_i^0 & \cdots & \Sigma_{i=0}^{n} x_i^j & \cdots & \Sigma_{i=0}^{n} x_i^m \\ \Sigma_{i=0}^{n} x_i^1 & \cdots & \Sigma_{i=0}^{n} x_i^{j+1} & \cdots & \Sigma_{i=0}^{n} x_i^{m+1} \\ \Sigma_{i=0}^{n} x_i^2 & \cdots & \Sigma_{i=0}^{n} x_i^{j+2} & \cdots & \Sigma_{i=0}^{n} x_i^{m+2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \Sigma_{i=0}^{n} x_i^{m-1} & \cdots & \Sigma_{i=0}^{n} x_i^{j+m-1} & \cdots & \Sigma_{i=0}^{n} x_i^{2m-1} \\ \Sigma_{i=0}^{n} x_i^m & \cdots & \Sigma_{i=0}^{n} x_i^{j+m} & \cdots & \Sigma_{i=0}^{n} x_i^{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ \cdots \\ a_j \\ \cdots \\ a_m \end{pmatrix} = \begin{pmatrix} \Sigma_{i=0}^{n} y_i \\ \cdots \\ \Sigma_{i=0}^{n} y_i x_i^2 \\ \cdots \\ \Sigma_{i=0}^{n} y_i x_i^m \end{pmatrix}$$

4. 주행의 수학적 해석

직진 상황에서는 선형 근사를, 그 외의 상황에 대해서는 np(=numpy).polyfit으로 근사 가능한 최고 차수의 함수인 4차 함수로 근사를 진행하였다.

4.1 straight case에서의 주행 보조

직진 주행상황에는 K값들에 대해서 $y = ax + b$ 꼴의 선형 근사를 진행한다. K값을 좌, 우 둘로 나누어 각각 근사 함수를 생성하면, 두 근사 함수의 기울기의 절댓값의 차이와 $x = 6p + 3$ 에서 근사 함수의 함숫값의 차이를 이용하여 차체가 정중앙에 있도록 하는 알고리즘을 구상해볼 수 있다.

4.1.1 자주차가 도로의 중앙에 있는 경우

다음은 자주차가 직진 차선의 정중앙에 위치할 때 수신되는 이미지와 그 근사 함수를 시각화한 것이다.

그림 47. 자주차가 도로 정중앙에 있는 경우

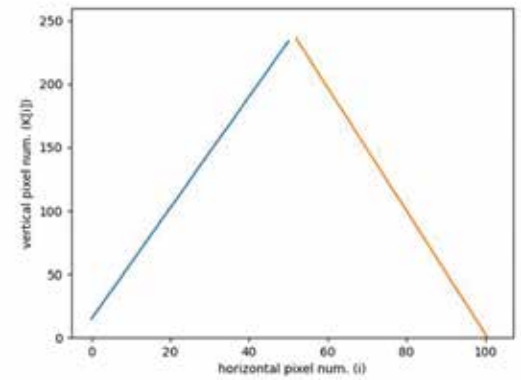


그림 48. [그림 47]의 선형 근사 함수 그래프

[그림 48]에서 파란색과 주황색으로 나타난 그래프의 기울기는 각각 4.37710407, -4.85466063로, 자주차는 두 기울기의 절댓값의 차가 약 0.5일 때 차량이 정중앙에 있다고 판단할 수 있다.

또, $x = 6p + 3$ 에서 좌, 우 근사 함수의 함숫값은 각각 238.02039215686273, 240.78980392156865로 두 값의 차이가 5 이하일 때 차량이 정중앙에 있다고 정의할 수 있다.

4.1.2 자주차가 직선 도로 왼쪽에 치우쳐져 있는 경우

다음은 직진 상황에서 자주차가 왼쪽 차선에 치우친 상황에서의 수신 이미지와 그 근사 함수의 그래프이다.

그림 49. 자주차가 왼쪽으로 치우쳐져 있는 경우

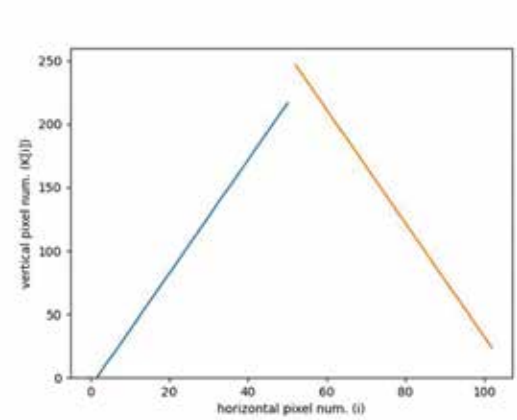


그림 50. [그림 49]의 선형 근사 함수 그래프

[그림 50]에서 파란색과 주황색으로 나타난 그래프의 기울기는 각각 4.47384615, -4.47438914이다. 두 기울기의 절댓값의 차이를 구하면 0.00054299로, 그 값이 매우 작음을 알 수 있다.

또, $x = 6p + 3$ 에서 좌, 우 근사 함수의 함숫값은 각각 220.88862745098047, 251.51058823529397로 두 값의 차이가 커 차량이 정중앙에 있지 않음을 알 수 있다.

4.2 right90, left90에서의 주행 보조

앞서 right90과 left90 주행상황은 조향 값에 명확한 기준이 될 수 있는 V값이 존재하지 않아 e = L[2] - 70과 같이 실험적 데이터에 기반한 조건식을 도출하여 조향 값을 설정하였다. 따라서 자주차의 속력 및 제원에 따라 차선 이탈을 유발할 가능성이 있는데, 이를 해결하기 위해 근사 함수를 사용할 수 있다.

곡선 주행상황에서는 K값들에 대해서 $y = ax^4 + bx^3 + cx^2 + dx + e$ 꼴의 근사를 진행했다. 다음은 right 90 상황일 때 수신되는 이미지에 K값 그리드를 적용한 사진과 주행에 따른 14개 프레임에서의 근사 함수 그래프이다.

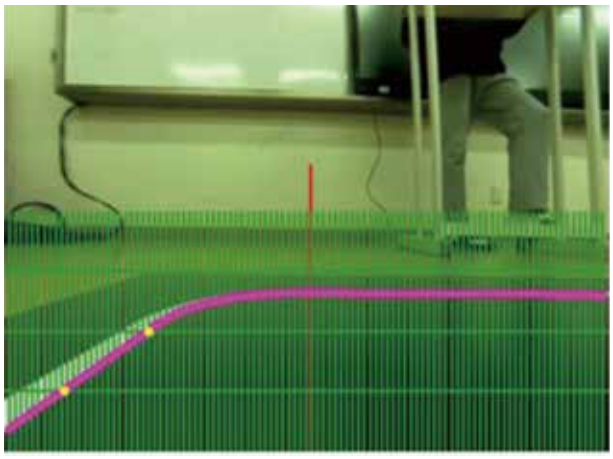


그림 51. right 90일 때 K값 적용 이미지 (분홍색 점들이 K[]이다.)

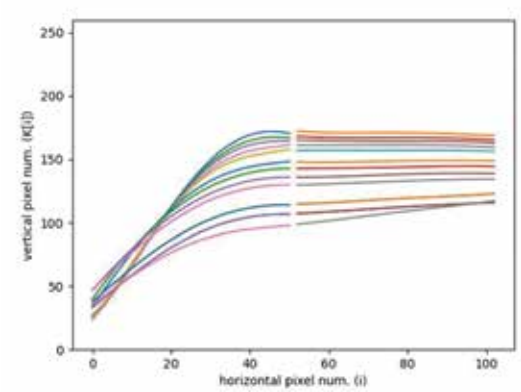


그림 52. right 90 주행 시 [그림 51]를 포함한 14개 프레임에서의 근사 함수 그래프

이때는 곡률의 개념을 도입해 상황을 수학적으로 해석할 수 있다. 곡률은 곡선 위의 점 P에서의 접촉원의 반지름 r의 역수로 정의된다. 이때 곡률 반지름 r은 다음과 같이 구할 수 있다. [9][10]

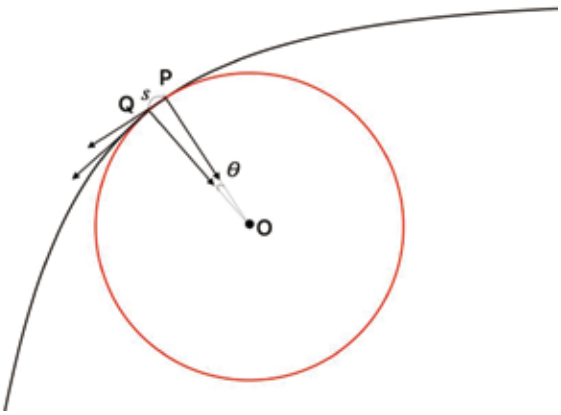


그림 53. 함수의 곡률을 구하는 방법

위 그림에서와 같이 어떤 함수에서 한 점 P가 존재하고 이에 대한 접촉원 O가 있을 때, 그 접촉원과 함수가 만나는 다른 한 점 Q가 있다고 하자. 이때, 그 원과 두 점 P, Q에서의 접선 벡터와 각 벡터에 수직인 법선 벡터가 이루는 각을 θ라고 하면, $\theta \to 0$ 이므로 호 PQ와 곡선 PQ가 같다고 근사할 수 있고, 부채꼴의 중심각도 θ라고 할 수 있다. 호 PQ의 길이를 s라고 하면 $s = r\theta$ 이고 다음을 만족한다.

$$r = \left| \lim_{\theta \to 0} \frac{s}{\theta} \right| = \left| \frac{ds}{d\theta} \right|$$

이를 x와 y에 대한 식으로 나타내면 아래와 같다.

$$\frac{ds}{d\theta} = \frac{ds}{dt}\frac{dt}{d\theta} = \sqrt{x'^2 + y'^2} \cdot \frac{x'^2 + y'^2}{y''x' - x''y'}$$

이때 본 논문에서 근사한 함수의 경우에는 매개변수 t를 사용하지 않는 y = f(x)형태의 함수이므로 곡률 k(x)를 아래와 같이 표현할 수 있다.

$$k(x) = \left| \frac{f''(x)}{((f'(x))^2 + 1)^{\frac{3}{2}}} \right|$$

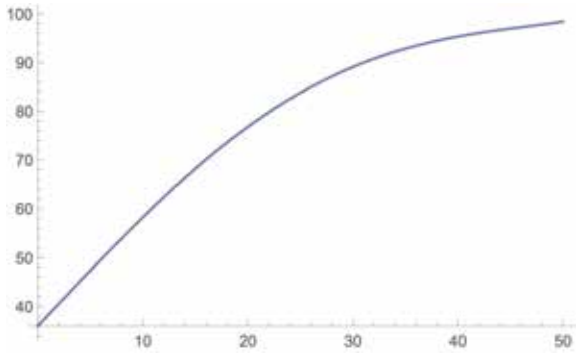


그림 54. right90 상황에서 x=0부터 x=50까지 f(x)

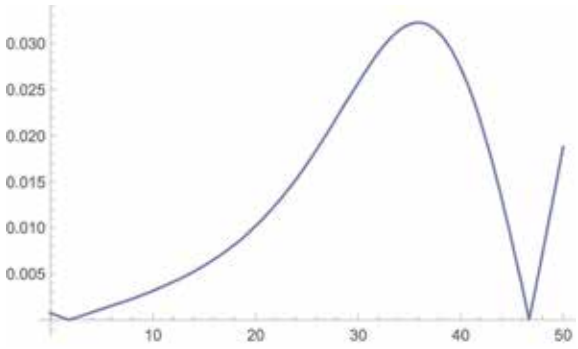


그림 55. [그림 54]에서의 곡률 계산

따라서, 위의 식을 근사한 4차 함수에 적용하면 각 점에 대한 곡률 반지름을 구할 수 있으며, 더 나아가 근사 함수의 곡률의 최댓값을 구할 수 있다. 예를 들어, right90 상황에서 f(x)의 그래프는 [그림 54]와 같이 구해지며 이때의 곡률은 [그림 55]로 구해진다. 또, 회전이 진행됨에 따라 좌측 근사 함수 그래프의 최대 곡률이 감소해야 하므로 e는 최대 곡률을 포함하는 식으로 다음과 같이 정의할 수 있다. e = k × {(현재 프레임의 좌측 근사 함수 그래프의 최대 곡률) - (회전이 거의 끝났을 때 계산되어야 하는 근사 함수 그래프의 최대 곡률)} (k는 상수)

VII. 결론

본 논문에서는 canny image 좌표분석과 CNN, 그리고 최소제곱법을 이용한 주행 및 보조 알고리즘에 대해서 제시하였다.

Canny Image 좌표분석 방식은 엔센 부등식을 활용해 t-turn 상황을 정의하고, 기본적인 case 분류에 있어 L, R이 아닌 V 값을 사용하였다는 점에서 의의가 있으며, CNN을 이용한 주행 상황 분석은 Adam optimzer를 통해 상황 판단 정확도를 99%까지 향상할 수 있음에 의의가 있다. 이렇게 판단한 상황에 따라 모터를 제어하고, 예외 상황(차단기)을 처리하는 알고리즘은 [그림 42]와 [그림 56]과 같은 시험 도로에서의 주행을 빠르고 정확하게 수행함으로써 유효함을 확인하였다.

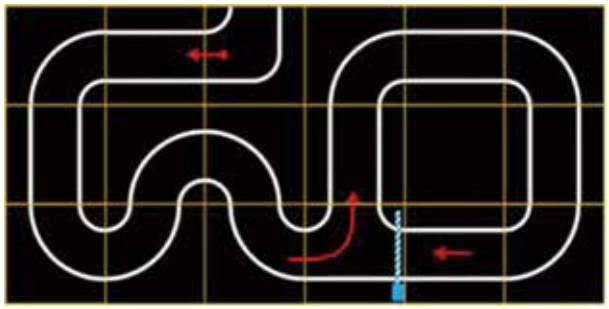


그림 56. 시험 도로 (빨간색 원이 출발 지점이며, 처음 차단기를 마주하였을 때(가운데 화살표)는 왼쪽으로, 두 번째 차단기를 마주하였을 때(오른쪽 화살표)는 차단기가 올라갈 때까지 정지하여 대기한다.)

경로 추적과 지도 생성을 통해 반복적인 주행에서의 효율성을 높일 수 있을 것으로 생각되며, 최소제곱법을 이용한 근사 함수와 같은 수학적 해석을 통해 주행의 정확도를 높일 수 있을 것이다.

참고문헌

[1] Geron Aurelien, Hands-On Machine Learning with Scikit-Learn and tensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 283~289, 2019

[2] Geron Aurelien, Hands-On Machine Learning with Scikit-Learn and tensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 445~483, 2019

[3] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629, 2018.

[4] Gholamalinezhad, Hossein, and Hossein Khosravi. "Pooling methods in deep neural networks, a review." arXiv preprint arXiv:2009.07485, 2020.

[5] Geron Aurelien, Hands-On Machine Learning with Scikit-Learn and tensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 118~127, 2019

[6] 장철원, 선형대수와 통계학으로 배우는 머신러닝 with 파이썬, 비제이퍼블릭, pp. 106-118, 2021

[7] Geron Aurelien, Hands-On Machine Learning with Scikit-Learn and tensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 351~370, 2019.

[8] 김창효, 수치해법과 전산 프로그래밍, 교학사, 1988.01.01.

[9] Goldman, Ron (2005). "Curvature formulas for implicit curves and surfaces". Computer Aided Geometric Design. 22 (7): 632–658. CiteSeerX 10.1.1.413.3008. doi:10.1016/j.cagd.2005.06.005.

[10] Coolidge, Julian L. (June 1952). "The Unsatisfactory Story of Curvature". American Mathematical Monthly. 59 (6): 375–379. doi:10.2307/2306807. JSTOR 2306807.

청년공학

최우수 논문

Journal of Youth Engineering

02 다양한 차선에 따른 자율 주행 알고리즘과 최적의 하드웨어 설정 연구

인천하늘고등학교
김우진, 성승민, 이호진

다양한 차선에 따른 자율 주행 알고리즘과 최적의 하드웨어 설정 연구

Robust algorithm for autonomous driving on various lanes and solutions for optimal driving condition

김우진, 성승민, 이호진
Woojin Kim, Seungmin Seong, Hojin Lee

요약

자율 주행차의 기본 조향 설정, 오 좌표 무시 알고리즘을 통해 차량 구동 환경을 최적화하였으며 중앙 보정, 커브 주행, 점선 주행 알고리즘을 작성하였다. 특히 점선 차선 주행 시 바깥 차선이 인식되는 문제가 발생하여 바깥 차선과의 거리를 인식하여 비어있는 구간을 감지하였고 실제로 잘 작동하였다. 또한 하드웨어에 대해서도 Canny역치와 딥러닝을 통해 차량의 사소한 문제까지 모두 잡았다.

Keywords: 자율 주행, 점선 인식, 커브 주행, 중앙보정, Canny 역치

I. 서 론

차선유지는 자율주행단계 1단계에서부터 도입되는 기능으로, 자율주행에 근본이 되는 기능이자 사람들이 가장 빠르게 자동화하고 싶어하는 일이다. 이상적인 실선 직선도로 상에서는 좌우 차선 좌표쌍의 평균값을 이용해 구할 수 있다. 하지만 실제 도로 상에서는 커브, 점선 등 다양한 상황이 있기에, 평균값만으로는 차선을 따라가기 힘들다. 이 문제를 해결하기 위해, 딥러닝을 활용해 이미지 자체를 input으로 활용해 중앙 좌표를 얻거나, 강화학습을 바로 사용하여 조향값을 얻을 수도 있다. 하지만 두 방식 모두 많은 양의 데이터와 학습을 필요로하며, 주행시에도 큰 컴퓨팅 파워가 필요하다. 이에 본 연구에서는 rule based model을 활용하되, canny detection 결과를 처리하여 만들었다. 차선 유지를 기본으로, 곡선 주행과 점선 주행의 다양한 차로 환경이 구현된 트랙 주행을 통해 이런 필터링이 필연적임을 확인하였고 이 전반의 과정을 뒷받침해주는 Hardware도 주행을 위한 최적의 상태로 해줄 수 있도록 여러 방안을 고안하였다.

II. 주행 알고리즘

1. 주행을 위한 기본 설정

1.1 조향의 기본값

주행 후에 전원을 끄면 모터가 돌아간 상태에서 종료되어버리는데, 이때 인위적으로 바퀴를 회전시키려하면 steer = 0이 가리키는 방향이 틀어진다. 때문에 자동차를 주행시키기 전 시범 주행을 통해 틀어진 정도를 측정하여 기준 조향으로 설정해야 한다.

하지만 기본조향이 바뀌면 설정해두었던 조향값의 모두를 달라진 조향에 따라 업데이트해야 해 불편함이 있었다. 이를 개선하기 위해 본 연구팀은

```
standard_steer = 11
steer = standard_steer  #기본 조향
```

그림 1. 기본조향 코드

위 사진처럼 변수를 따로만들어 기준 조향을 설정해주었다. 위 사진에서 11은 시범 주행을 통해 얻은 자동차의 현재 기준 조향이며, 아래는 직진을 위한 코드의 예시이다.

이처럼 이후부터는 steer 값을 조정할 때 위에서 설정한 변수를 수정하는 것 만으로 코드 전반의 조향 정보를 업데이트할 수 있다.

1.2 좌표설정 바꾸기

제공된 자율주행차 코드은 카메라로 받아온 이미지를 7x3의 격자로 나누는 것으로 설정되어있다. 하지만 연습을 거듭하면서 적은 좌표로는 차선의 변화에 즉각적이면서도 부드러운 대응을 할 수 없다고 판단하였다. 본 연구팀은 이미지의 격자를 늘리는 방법을 생각하였다. 좌표를 늘릴수록 잘못 인식된 좌표 하나가 전체적인 주행에 미칠 수 있는 영향이 적고 움직임이 부드러워진다는 장점이 있지만 그만큼 데이터를 관리하는 것이 어려워질 수 있고, 컴퓨터의 연산량이 많아져 딜레이가 생길 수 있어 실험을 거쳐 격자를 가로 41줄, 세로 21줄로 변형하였다.

```
V, L, R = self.gridFront(frontImage, cols=7, rows=3)
                        ↓
V, L, R = self.gridFront(frontImage, cols=41, rows=21)
```

그림 2. 격자 변경

김우진 kwoojin05@gmail.com
성승민 seongseungmin5470@gmail.com
이호진 hojin1566@naver.com

2. 필요없는 점 무시하기 코드

2.1 코드의 필요

canny 인식은 차선뿐만 아니라 주변의 사물에 대해서도 매우 민감하게 반응하기 때문에 자율주행 자동차가 주행하는 환경에 따라 불필요한 좌표가 감지될 수 있다. 잘못 인식된 좌표가 주행 알고리즘에 개입하는 것을 막기 위해 본 연구팀은 차선과의 거리, 즉 V[]의 정보를 활용하여 도로 영역 밖에서 인식된 L, R의 좌표를 무시할 수 있는 코드를 개발하였다.

2.2 코드 알고리즘

Algorithm. 무시코드

```
for 매개변수 in 0부터 20까지
    t <-- 도로 영역내에서 인식될 수 있는 L,R인덱스의 최댓값
    = V의 최댓값의 L, R 인덱스
    if t보다 매개변수가 크다면
        R[매개변수] = "무시"
        L[매개변수] = "무시"
```

V의 최댓값이 차선 영역내의 가장 먼 거리이므로 이보다 멀리 떨어져있는 L, R의 좌표는 차선 밖의 점임을 확인할 수 있다. 때문에 V의 최댓값은 L, R의 좌표계로 변환하여 이 값보다 큰 인덱스를 가진 L, R좌표들을 무시하도록 하였다.

V[]는 가로의 순서에 따라 세로의 값이 대응되는 데이터지만 R[], L[]은 세로의 순서에 따라 가로의 값이 대응되는 데이터이기에, 두 값의 세로 성분 크기를 비교하려면 각각의 값을 하나의 좌표계로 통일시켜주어야 한다. 이 때문에 실제 코드를 작성할 때는 V값을 11로 나누어줌으로써 t를 구할 수 있다.

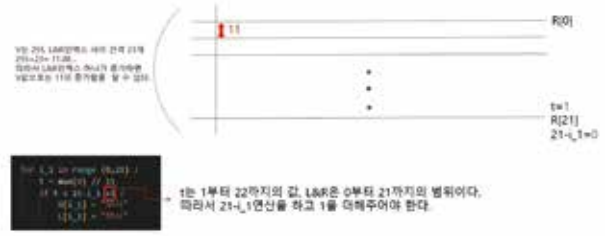


그림 3. 오좌표 무시 코드의 설명

2.3 실행 결과

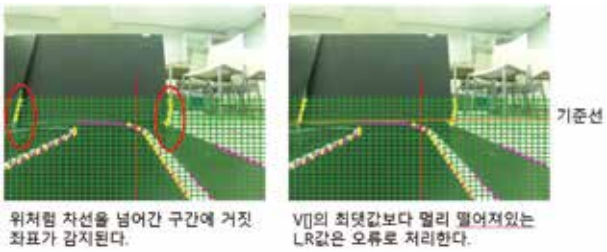


그림 4. 무시코드 실행결과

위 사진처럼 차선 밖에 있는 점의 무시가 원활히 이루어짐을 확인할 수 있었다. 하지만 다양한 상황에서 작동시켜 보면서 한계점을 발견하였는데 바로 아래 사진처럼 왼쪽과 오른쪽의 정보가 다른 경우에, 한쪽의 정보가 다른 쪽 좌표의 처리에 영향을 미칠 수 있다는 것이다.

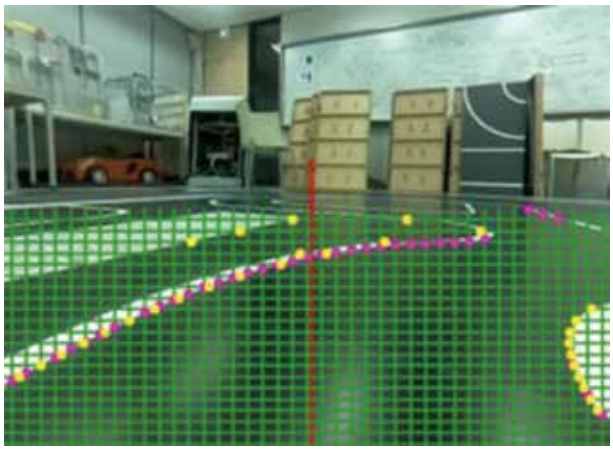


그림 5. 수정 전 무시코드의 문제점

특별히 이 경우에는 오른쪽의 최댓값이 매우 크게 감지되는 바람에 왼편에 생긴 오 좌표들이 무시되지 못하고 있다. 이를 해결할 방법으로 왼쪽(L[])과 오른쪽 (R[])의 정보를 '구분'하여 각각 처리할 방법을 고안하였다. 바로 중심인 V[20]을 기준으로 V[]를 20개씩 나누어 새로운 list를 만드는 것이다.

그러면 오른쪽, 왼쪽에 대한 V의 최댓값을 각각 얻어내는 것이 가능해진다. 이를 바탕으로 보완한 알고리즘은 아래와 같다.

Algorithm. 보완된 무시코드

```
for 매개변수 in 0부터 20까지
  t_L = 왼쪽에 있는 V값의 집합 중 가장 큰 값 // 11
  t_R = 오른쪽에 있는 V값의 집합중 가장 큰 값 // 11
  if t_L보다 매개변수가 크다면
    L[매개변수] = "무시"
  if t_R보다 매개변수가 크다면
    R[매개변수] = "무시"
```

3. 중앙보정코드

3.1 중앙보정코드의 필요

중앙보정은 자동차가 한쪽의 차선으로 치우치지 않고 도로의 중앙에서 안정적으로 위치를 잡을 수 있도록 해주는 코드이다. 곡선 구간에 최적화하여 코드를 설계하더라도 어떻게 진입하는지에 따라 주행 결과가 다르게 나타날 수 있기 때문에 안정적인 주행과 코드에서 의도한 결과가 이상적으로 잘 드러나기 위해서는 자동차가 중앙을 찾아 스스로 위치를 조정하는 코드가 필요하다고 느꼈다.

3.2 첫 번째 시도

Algorithm. 중앙보정 코드_첫 번째

```
if 중앙의 R 값이 L 값보다 클 때: 우회전
if 중앙의 L 값이 R 값보다 클 때: 좌회전
```

*회전의 조향값 크기는 | L-R |로 하였다.

처음에는 중앙보정을 위와 같이 하나의 R,L에 대해서 지엽적으로 설계하였다. 하지만 시행착오를 겪으면서 하나의 L, R 데이터에 의존하여 알고리즘을 설계했을 때, 차선의 흐름과 인식된 R, L의 좌표가 불일치하는 문제가 특수한 경우에서 발생하였다.

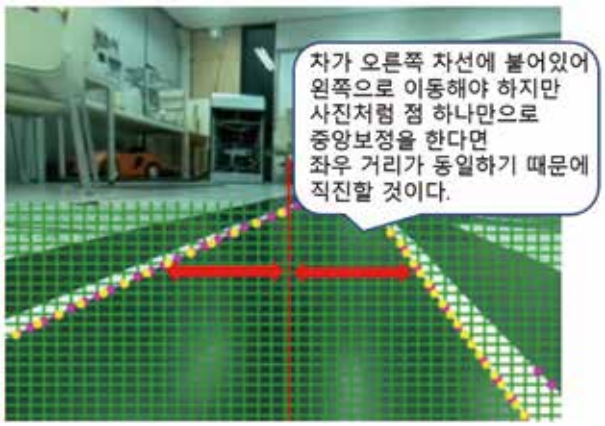


그림 6. 수정 전 중앙보정 코드의 문제점

3.3 두 번째 시도

첫 번째 시도에서 드러난 한계를 해결하기 위해 우리가 사용한 방법은 모든 L, R값의 평균을 이용하는 것이다.

Algorithm. 중앙보정 코드_두 번째

```
total = 0
for 매개변수 in 0부터 20까지
        center = L[매개변수] - R[매개변수]
        if center == 0 <--차가 중앙에 있는 경우이다.
                total +=0
        elif center < 0 <--차가 왼쪽에 치우쳐있는 경우이다.
                total += lcenterl <--우회전을 위해 절댓값을 취해준다.
        if center > 0 <--차가 오른쪽에 치우쳐 있는 경우이다.
                total += - lcenterl <--좌회전을 위해서

average = total // 21
steer = average // 5 + standard_steer <--조향의 변화폭을 부드럽게 만들기 위해 5를 평균값에 5를 추가로 나눠주었다.
```

*변수 total은 $\sum_{n=0}^{20}(R[n]-L[n])$와 같다.

더 많은 데이터를 활용하기 때문에 오류를 최소화할 수 있으며 휙휙 바퀴를 꺾는 것이 아니라 섬세하게 값을 조정하여 차선의 중심으로 부드럽게 돌아갈 수 있다는 장점이 있었다. 이에 더해 커브 코드에 진입하기 전에 미리 값을 조금씩 조정하여 약간의 회전을 주는 '완화곡선' 역할을 할 수 있었다.

4. 점선 차선 주행

4.1 점선 주행 코드의 필요

앞서 우리는 주행의 안정성을 꾀하기 위해 L,R의 값을 평균내어 차선의 중앙을 지키도록 하는 '중앙보정' 알고리즘에 대해 설명하였다. 하지만 이는 실선이 아닌 점선인 차선에서는 오히려 주행의 불안을 야기하였다. 우리는 점선으로 차선이 변화할 때 생기는 처리의 오류를 줄이기 위한 알고리즘을 추가적으로 설계할 필요를 느꼈다.

4.2 코드 알고리즘

그림 7. 점선 주행 코드

위 코드를 1, 2, 3으로 나누어 설명하겠다.

1.

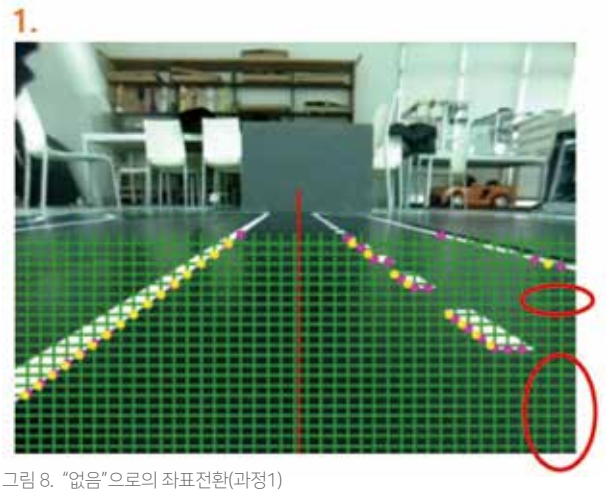


그림 8. "없음"으로의 좌표전환(과정1)

점선의 비어있는 부분은 좌표가 잡히지 않기 때문에 항상 최댓값으로 존재한다. 따라서 이를 무시할 수 있도록 해준다.

2.

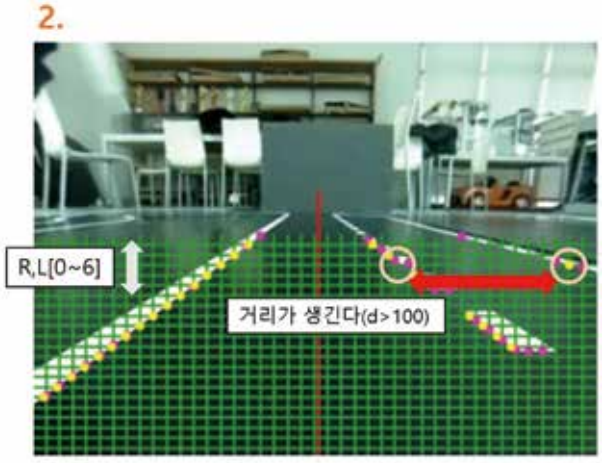


그림 9. "없음"으로의 좌표전환(과정2)

차로부터 멀어질수록 카메라가 인식하는 도로의 폭이 좁아지기 때문에 그림처럼 점선의 '비어있는 구간'임에도 바깥 차선이 인식되어 카메라 앵글 안에서 좌표가 잡히는 상황이 생긴다. 이 상황에서는 R의 값이 최댓값보다 작기 때문에 1의 과정을 따를 수 없다.

하지만 여전히 위아래 좌표와 큰 거리가 생기므로 위와 같이 멀리 떨어져 있는 좌표(R[0~6])에 한해서 최댓값 인식이 아닌 특정값 이상의 거리 인식으로 '비어있는 구간'을 감지하기로 하였다.

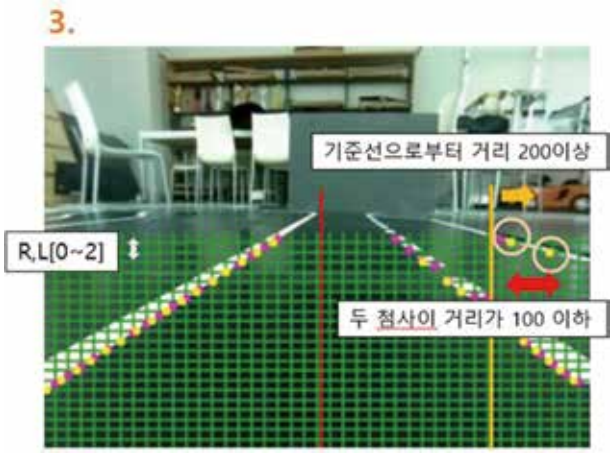


그림 10. "없음"으로의 좌표전환(과정3)

테스트를 거듭하면서 순간적인 모멘트에서 다음처럼 비어있는 구간이 연속되어 '2.' 과정에서 의도한 '거리'가 감지되지 않는다는 것을 발견하였다. 따라서 R[0~2]의 좁은 구간에 한하여 위의 '1.', '2.' 과정에서 걸러지지 않았으나 R 값(기준선으로부터의 거리)이 200 이상인 좌표들을 추가로 '비어있는 부분'으로 인식하도록 하였다. *다른 주행상황의 개입을 방지하기 위하여 V[20]==255(앞에 벽이 없는 직선 도로)를 추가 조건으로 덧붙였다.

마지막으로 "없음" 비어있는 구간의 개수를 세어repeat이라는 변수에 이를 저장한다.

세 가지 경우를 나눠 비어있는 부분의 R값을 "없음"이라는 문자로 바꾸었다. 다음으로는 이 정보를 활용하여 비어있는 구간에 적절한 가상의 좌표를 대입해주는 과정을 거칠 것이다.

점선 코드는 얻어진 L,R 데이터를 재가공하는 과정이기 때문에 점선이 아닌 곳에서 작동이 된다면 주행이 오히려 더 불안해질 수 있다. 따라서 코드에 앞서 확실히 점선으로 인식되었을 때만 점선 코드가 작동될 수 있는 조건문이 필요하다.

```
#점선코드 진입 조건
if repeat >= 3:
    if L[5] != "없음" and L[6] != "없음":
```

그림 11. 점선 코드 진입 조건

repeat =1, 2는 일반적인 주행상황에서도 발견될 수 있기 때문에 점선으로 인식하는 기준을 repeat = 3으로 설정하였다.

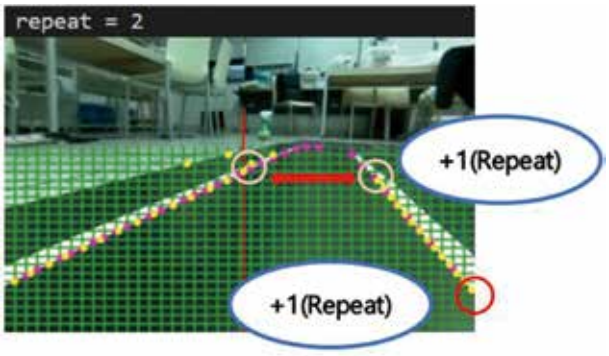


그림 12. repeat이 잘못 감지되는 case

다음으로 점선의 공백을 감지하는 코드이다. 점선의 비어있는 구간이 시작되는 점을 기준으로 비어있는 부분이 연속된 좌표에서 얼마만큼 이어지는지 세고 이를 리스트로 저장한다.

그림 13. 점선 코드(공백 카운트)

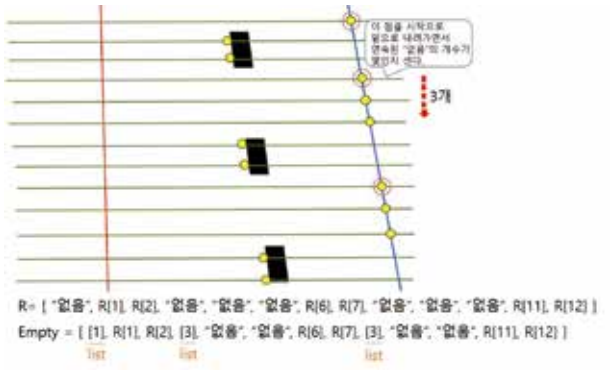


그림 14. 빈 구간의 폭을 리스트 정보로 변환하는 과정

위 그림과 같이 새로운 list를 만들어 비어있는 구간에 대한 정보를 담았으니 기존의 "없음"을 지워준다.

```
num =empty.count("없음")
for delete in range(num):
    empty.remove("없음")
```

Empty = [[1], R[1], R[2], [3], R[6], R[7], [3], R[11], R[12]]

그림 15. '없음' 지우기

이렇게 list가 깔끔해진다. 자동차가 중앙에서 정면을 바라보고 있는 상황이므로 왼쪽의 차선과 오른쪽의 차선이 대칭을 이룬다. 따라서 왼쪽 차선의 기울기(V 인덱스 하나가 증가할 때 증가하는 L값)의 절댓값과 오른쪽 차선의 기울기의 절댓값이 같다고 할 수 있다. 왼쪽 차선의 기울기를 sample_L로 정의하고 양쪽 차선의 기울기 합이 30임(양쪽이 대칭일 때 각각 15이기 때문이다)을 이용해 오른쪽의 기울기를 gap으로 정의한다. 비어있는 좌표의 위아래의 R, L값에 gap을 더하거나 뺌으로써 가상의 좌표를 만들어줄 수 있다. 아래의 그림을 통해 과정을 자세히 알아보자.

```
# # (3) 공백이 생긴 부분에 유의미한 좌표를 채워준다
for i_5 in range(20+1):
    if type(empty[i_5]) == list:
        a = empty[i_5][0]
        sample_L = (L[6] - L[5])
        gap = 30 - sample_L
        if i_5 == 0:
            after = empty[i_5 + 1]
            del empty[i_5]
            for p in range(1,a+1):
                if after - gap*p <0:
                    empty.insert(i_5,"없음")
                else:
                    empty.insert(i_5, after - gap*p)

        else:
            before = empty[i_5 - 1]
            del empty[i_5]
            for m in range(0, a):
                try:
                    if before + gap*(m+1) <0:
                        empty.insert(i_5 + m , "없음")
                    else:
                        empty.insert(i_5 + m , before + gap*(m+1))

                except:
                    print(before)

R = empty
```

그림 16. 점선 코드(좌표 채우기)

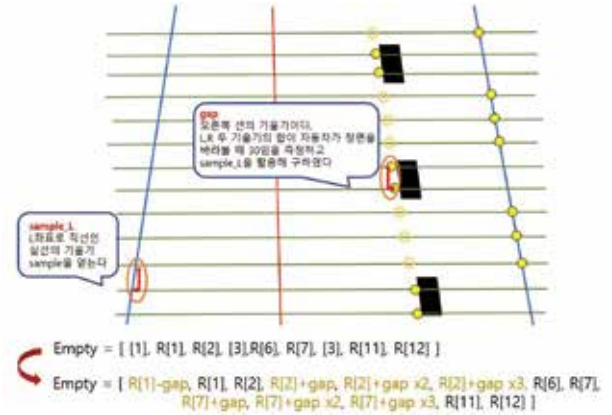


그림 17. 좌표 생성과정

4.3. 실행 결과

점선인 차선이 존재하는 도로에서도 차의 조향이 흔들리지 않고 안정적으로 주행하는 것을 확인할 수 있었다.

5. 커브 주행

5.1 코드 알고리즘

Algorithm. 커브 주행

if 왼쪽 차선의 기울기가 달라짐:
Idx_shift_L = 차선의 기울기 변화가 처음으로 감지되는 점
slope_L = Idx_shift_L과 Idx_shift_L로부터 3번째 앞에 있는 점 사이의 기울기
angle_L = slope_L의 벡터와 자동차의 이동방향과 수직한 벡터가 이루는 각도
if 오른쪽에 빈 공간이 있고, Idx_shift_L이 유의미하게 정의되었다면: <- 커브코드의 진입조건
조향 = 90-angle_L <- 회전해야 하는 각도

if 오른쪽 차선의 기울기가 달라짐:
Idx_shift_R = 차선의 기울기 변화가 처음으로 감지되는 점
slope_R = Idx_shift_R과 Idx_shift_R로부터 3번째 앞에 있는 점 사이의 기울기
angle_R = slope_R의 벡터와 자동차의 이동방향과 수직한 벡터가 이루는 각도
if 왼쪽에 빈 공간이 있고, Idx_shift_R이 유의미하게 정의되었다면: <- 커브코드의 진입조건
조향 = - (90-angle_R) <- 좌회전이라 조향이 음수값

```
while  V[Idx_shift_L] > V[Idx_L] + 11 * (Idx_shift_L - Idx_L) -30 and Idx_shift_L < 19:
    Idx_shift_L += 1
if L.count("없음") < R.count("없음") and V[Idx_shift_L] > V[Idx_L]:
    slope_L = (V[Idx_shift_L + 3] - V[Idx_shift_L]) / 3
    angle_L= np.arctan(slope_L/6)*180//3.14
    steer = int (90-angle_L) +standard_steer

while  V[40-Idx_shift_R] > V[40-Idx_R] + 11 * (Idx_shift_R - Idx_R) -30 and Idx_shift_R < 19:
    Idx_shift_R += 1
if L.count("없음") > R.count("없음") and V[40-Idx_shift_R] > V[40-Idx_R]:
    slope_R = (V[37-Idx_shift_R] - V[40-Idx_shift_R]) / 3
    angle_R = np.arctan(slope_R/6)*180//3.14
    steer =  int (angle_R - 90) +standard_steer
```

그림 18. 커브 주행 코드

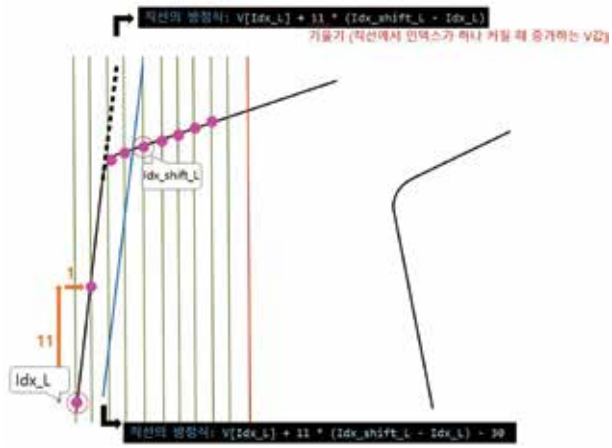


그림 19. 기울기가 변하는 지점(Idx_shift_)을 찾는 과정

자동차가 직진을 하고 있을 때, V의 인덱스가 하나 증가할 때 V의 값은 11 증가한다. 왼쪽, 오른쪽 각 방향에서 V값이 최소가 되는 점을 기준으로 하여 직진 방향의 직선의 방정식을 모델링할 수 있다. 우회전 하는 상황을 생각해보자. 현재 자동차는 직진을 하고 있고 이 왼편의 직선을 연장한다면 직선의 방정식은 “V[Idx_L] + 11* (Idx_shift_L- Idx_L)”가 될 것이다. 이 직선을 -30만큼 세로성분으로 평행이동 시켜보자. 만약 차선이 계속 직선으로 뻗어있다면 이 평행이동한 직선과 차선은 서로 평행하여 만나지 않을 것이다. 하지만 차선이 중간에 꺾인다면 평행이동과 차선은 한 교점에서 만날 것이다. 이 교점의 유무로 차선의 기울기가 달라짐을 감지하고 교점 바로 다음의 점을 Idx_shift_L 로 설정하였다.

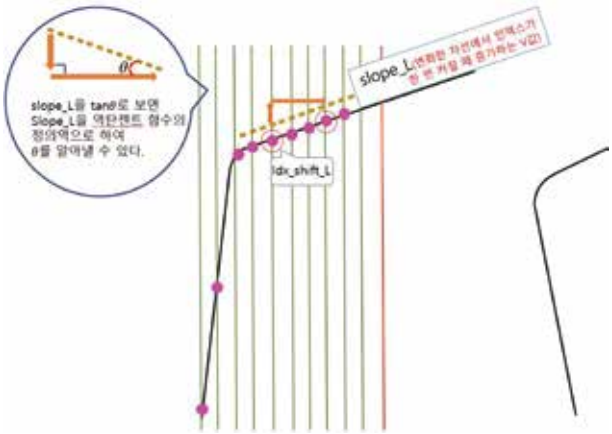


그림 20. 차선의 꺾인 기울기를 구하는 과정

이전 과정에서 설정한 Idx_shift_L을 기준으로 그 앞에 3번째 점까지의 기울기를 구하고 이를 slope_L로 정의한다. 기울기를 탄젠트의 개념으로 생각해 slope_L을 역탄젠트의 함수의 정의역으로 하면 치역으로 slope_L의 벡터와 자동차의 이동방향과 수직한 벡터가 이루는 각도를 얻을 수 있다.

그림 21. 각도 계산의 유의점

V의 인덱스와 값이 1:6으로 대응하므로 각을 계산할 때 slope_L에서 6을 나누어주어야 한다. 이후 180//3.14를 곱해준 것은 라디안을 의 단위로 변환하기 위함이다. 계산된 값을 angle_L로 정의한다.

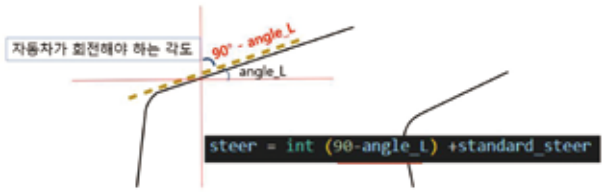
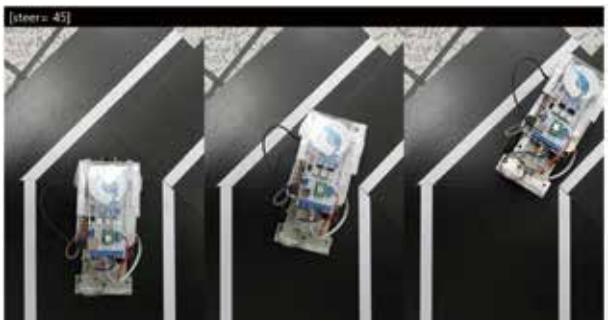


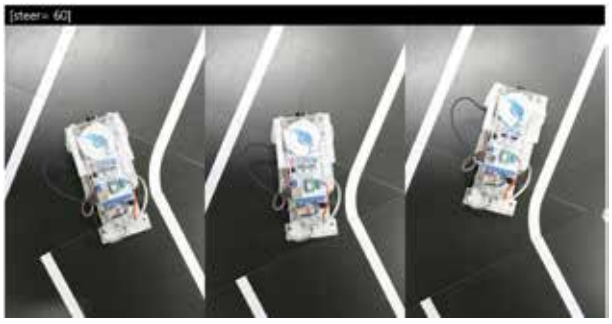
그림 22. 조향 설정

자동차가 실제로 회전해야 하는 각도는 90-angle_L이므로 이를 조향값의 크기로 설정한다.

5.2 회전각과 조향의 수치의 관련성

위에서 설명한 코드 알고리즘은 회전해야 하는 각도와 회전을 위해 설정해야 하는 조향값이 정확히 일치함을 전제로 하고 있다. 우리는 알고리즘을 설계하기 전에 실험을 통해 이를 발견하였다. 자료를 첨부하여 이해를 돕고자 한다.





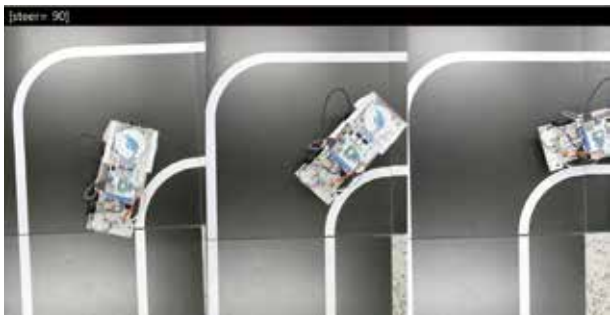


그림 23. 실제 각도와 steer값의 관계

6. 한계

이상적인 상황을 바탕으로 작성한 알고리즘이기 때문에 오류로 예상치 못한 점이 인식되면 처리가 불안정해진다는 단점이 있다. 이를 해결하기 위해서 도로 밖의 점은 무시할 수 있는 코드를 만들었지만 도로 내부에서 생기는 문제는 해결하지 못하였다. 구동 환경의 조도, 도로의 이물질 등 오류를 야기할 수 있는 변인들을 다 통제하여 항상 이상적인 데이터가 들어오게 하면 되지만, 완벽히 통제하기에 한계가 있어 도로 내부에서 오류가 생기더라도 안정적으로 주행을 이어나갈 방법에 대한 탐구가 필요하다.

III. 하드웨어 최적화

1. 캐니 역치

앞선 도로 내부의 문제를 해결하기 위해 하드웨어적 방안들을 고안하였다.

1.1 캐니 역치의 설명

Canny Edge Detection은 이미지의 윤곽을 찾아내는 알고리즘이다. 현재 cv2 라이브러리에서 cv2.Canny()로 사용할 수 있다. 이 함수의 필수 매개변수들은 다음과 같다.

cv.Canny(image, threshold1, threshold2)
image : 변환할 이미지
threshold1 : 첫 번째 역치
threshold2 : 두 번째 역치

threshold2에 적힌 역치 값을 넘으면 선으로 인식하게 되며, threshold1부터 2까지의 값들 중 윤곽에 인접하고 있는 부분은 윤곽으로 인식하게 된다.

오츠의 이진화 알고리즘은 임계값을 임의로 정해 픽셀을 두 부류로 나누고 명암 분포를 구하는 작업을 반복한다. 모든 경우의 수 중에서 두 부류의 명암 분포가 가장 균일할 때의 임계값을 선택한다.

1.2 오츠 알고리즘 적용

본 연구팀은 주행시 명암 분포가 낮을 때와 높을 때의 역치를 계산하여 차선이 더 잘 인식되도록 하였다.

함수의 정의는 다음과 같다.

ret, out = cv2.threshold(img, threshold, value, type_flag)
img: 변환할 이미지
threshold: 스레시홀딩 임계값
value: 임계값 기준에 만족하는 픽셀에 적용할 값
type_flag: 스레시홀딩 적용 방법

cv2.threshold는 전역 thresholding을 해준다. 이때 전역 thresholding이란 어떤 임계값(역치)를 정한 뒤 픽셀 값이 임계값을 넘으면 255, 임계값을 넘지 않으면 0으로 지정하는 방식이다.

```
a, b, c = cv2.split(frontImage)

otsu_threshold, image_result = cv2.threshold(
a, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

그림 24. 오츠 알고리즘 적용 코드

이미지 매개변수로 blue값을 가진 단일 채널 이미지를 넣어주었다. 그리고 cv2.THRESH_OTSU를 마지막 파라미터로 전달하여 오츠 알고리즘이 적용된 적정값을 얻었다.

실제 대회에서는 어두운 상황에 81이 나오며, 밝은 상황에 144가 나오는 것을 확인할 수 있었다. 결과적으로 canny 안에 있는 threshold1을 81으로, threshold2을 144로 설정해 차로를 정확하게 탐지할 수 있었다.

2. 자주차 연결 불량 원인 파악

2.1 자주차의 통신상태

빈번하게 발생하는 자주차의 연결 불량을 파악하기 위해 자주차의 통신상태를 연구하였다. cmd의 명령어 tracert, ping을 통해 통신 속도와 상태를 볼 수 있었다.

tracert는 지정된 호스트에 도달할 때까지 통과하는 경로의 정보와 각 경로에서의 지연 시간을 추적하는 명령이다. 컴퓨터와 pi 간의 반응속도를 점검할 때 쓸 수 있었다.

ping 명령어는 IP 수준 연결을 확인하는 명령어로 응답시간과 패킷의 손실을 확인할 수 있다. 따라서 손실되는 데이터는 없는지 확인할 때 쓸 수 있었다.

```
C:₩Users₩gram_>tracert 192.168.0.10

최대 30홉 이상의 192.168.0.10(으)로 가는 경로 추적

  1    36 ms     2 ms     3 ms  192.168.0.10

추적을 완료했습니다.

C:₩Users₩gram_>ping 192.168.0.10

Ping 192.168.0.10 32바이트 데이터 사용:
192.168.0.10의 응답: 바이트=32 시간=11ms TTL=64
192.168.0.10의 응답: 바이트=32 시간=3ms TTL=64
192.168.0.10의 응답: 바이트=32 시간=3ms TTL=64
192.168.0.10의 응답: 바이트=32 시간=8ms TTL=64

192.168.0.10에 대한 Ping 통계:
    패킷: 보냄 = 4, 받음 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 3ms, 최대 = 11ms, 평균 = 6ms
```

그림 25. cmd에서 본 통신 속도

2.2 라즈베리파이의 처리상태

라즈비안 cmd의 명령어 htop은 시스템 실시간 모니터링으로 각 코어의 부하 상태와 프로그램의 상태, cpu 점유율 등을 보여준다. 이에 따라 하드웨어의 상태와 연산량을 실시간으로 확인해 프로그램에 반영할 수 있었다.

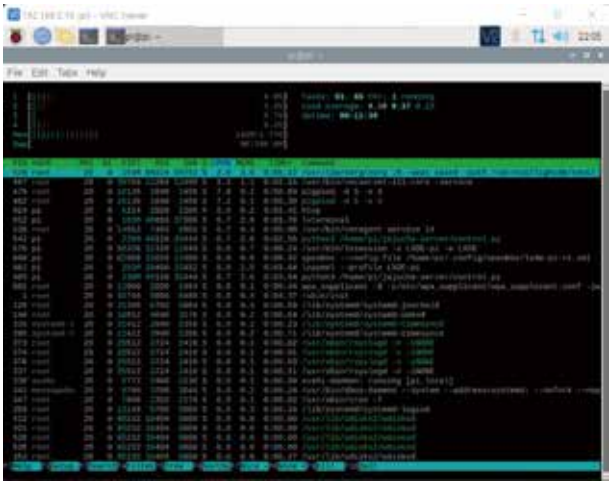


그림 26. 프로그램 상태 모니터링 화면

2.3 framerate 조정에 따른 통신상태

아래 사진은 자주차 내부의 streaming.py로, fps를 framerate=(숫자)를 통해 조정할 수 있다. 숫자를 늘릴수록 화면이 부드러워져 차의 방향 조절을 부드럽게 할 수 있지만, 그에 따라 연산량이 많아져 하드웨어가 연산속도를 받쳐주지 못하는 것을 확인할 수 있었다.

예시로 100을 넣어 테스트해본 결과 연산속도가 따라가지 못해 딜레이가 나타나며, 프로그램이 끝나도 이미지 처리를 하는 현상을 htop을 통해 확인할 수 있었다.

그림 27. streaming.py

3. 추가적인 운전 방안

본 연구팀은 보행자를 인식하고 멈출 수 있는 알고리즘을 구현하고자 사람의 얼굴을 인식하고 그 위치를 파악하는 프로그램을 cascade classifier란 머신러닝 기반 분류 알고리즘을 통해 만들어보려 하였다. 그 과정에서 OpenCV에 내장되어 있는 얼굴 인식 학습 데이터를 활용해 실시간 얼굴인식 프로그램을 만드는데 성공했다. 이 프로그램은 먼저 학습된 XML 데이터를 불러와 입력된 사람의 사진을 흑백사진으로 변환한다. 사람을 감지한 뒤 직사각형으로 표시해주었다. 이를 활용하면 자주차가 도로에서 보행자를 인식하고 멈추거나 다른 방향으로 피해갈 수 있는 알고리즘을 설계하는데 활용할 수 있다.

```
import cv2
faceCascade = cv2.CascadeClassifier('cv2\haarcascade_frontalface_default.xml')

def detect(gray, frame):
    faces = faceCascade.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=5,
    minSize=(100,100), flags=cv2.CASCADE_SCALE_IMAGE)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x,y), (x+w, y + h), (255,0,0), 2)
    return frame

video_capture = cv2.VideoCapture(0)

while True:
    _, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame)
    cv2.imshow('face_detected', canvas)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```

그림 28. 얼굴인식 프로그램

4. 서보모터의 각도로 인한 바퀴 각도 최댓값

4.1 서보모터

그림 29. 차량의 앞부분

자율 주행 자동차 주행 시 steer 값이 일정 정도를 넘어가면 넘어간 상태로 고정되어 원상태로 복구가 안되는 문제가 발생하였다. 위의 사진은 서보모터에 연결되어있는 기어의 모습이다.

서보모터에 연결된 피니언이 돌아가며 전륜의 방향을 조정하는 것인데, 차체의 구조상 래크의 길이가 짧아 일정 정도 이상 회전하면 맞물리지 못하게 된다. 또한 3D 출력물로 출력한 탓에, 견고하지 않아 맞물림이 풀어지는 문제도 발생하였다. 본 연구팀은 기어의 허용범위 내에서만 톱니가 회전할 수 있도록 알고리즘을 구성하였다.

4.2 실제 코드

다음은 조향의 의사코드이다.

If 조향 > 100 + 기본 조향
　　조향 = 100 + 기본 조향
If 조향 < -100 + 기본 조향
　　조향 = -100 + 기본 조향

최대값을 설정하여 그 이상으로 꺾일 수 없도록 하였다. 여기서 standard_steer는 차량이 정확히 직진할 때의 steer 값으로, '11' 정도에 머물렀다.

IV. 결론 및 제언

결론

본 연구를 통해 커브 도로, 점선인 차선이 존재하는 도로, 외부점의 간섭이 생기는 구간에서의 안정적 주행을 위한 알고리즘을 발전시켜나갈 수 있었다. 첫 번째로 실시간으로 차선의 기울기를 계산하고 이를 바탕으로 회전값을 조정하는 알고리즘은 커브가 일정한 각도로 휘어져 있지 않고 구불구불한 길로 이어져 있더라도 자동차가 상황에 적응할 수 있도록 긍정적인 발전을 주리라 기대된다. 두 번째로 중앙보정은 본래 의도했던 중앙으로의 위치보정의 목적을 달성함과 동시에 곡선 구간 진입시에 완화곡선의 역할을 하여 커브를 쉽게 돌 수 있도록 도와주었다. 차선이 점선일 때 이를 실선의 데이터로 보정하는 방법을 고안한 것은 점선이 있는 구간에서도 중앙보정의 장점을 활용할 수 있게 하였다는 의의가 있다. 도로 외부의 점을 무시하는 코드를 추가하여 canny의 오 인식으로 발생하는 불필요한 간섭이 큰 폭으로 줄어들었음을 확인하였다.

또한 주행 조건 최적화를 위해 오츠의 알고리즘을 이용한 캐니 역치 계산방식으로 조도차이에 의한 차선 인식 오류를 없앨 수 있었고 라즈베리파이 시스템 실시간 모니터링, 통신상태 확인을 통해 처리된 데이터가 안정적으로 전달되게 하였다. 이에 더해 OpenCV를 이용하여 머신러닝 기반 감지 프로그램을 만들어 연구의 확장을 유도하였다.

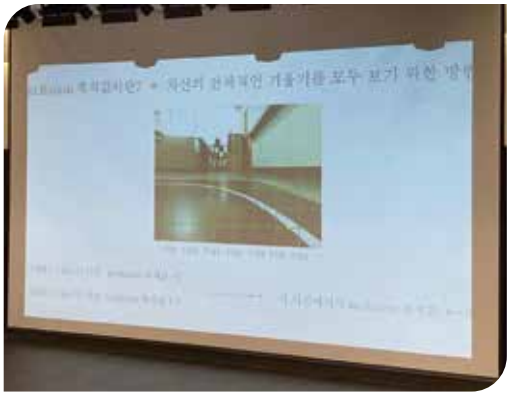
본 연구는 특히 일어날 수 있는 변인들을 통제할 수 있는 방법을 제시하였다는 것에 의의가 있다. 앞에서 소개한 두 가지의 기본값 설정방법 또한 자율주행차 알고리즘을 설계하는데 있어 다른 연구자들에게도 도움이 되리라 생각된다.

한계 및 제언

다양한 차선에서 각각을 위한 알고리즘을 따로따로 설계하였기 때문에 특수한 경우에 2개 이상의 알고리즘이 중첩되는 현상이 발견되었다. 본 연구에서는 이를 방지하기 위해 각각의 코드 앞에 특수한 진입 조건을 달아주어 간섭을 최소화할 수 있었지만 본 연구환경에 맞춰진 조건인 만큼 다른 환경에서 자동차를 구동시키면 간섭 여부를 장담할 수 없다. 따라서 각각의 알고리즘을 간섭없이 하나로 통합하는 방법에 대한 탐구가 이루어지면 좋을 것이다.

03 연속적인 곡선 차선에서 유동적이고 효율적인 주행을 위한 주행 알고리즘

선덕고등학교
정재훈



연속적인 곡선 차선에서 유동적이고 효율적인 주행을 위한 주행 알고리즘

Driving algorithms for fluid and efficient driving in continuous curved lanes

정재훈
Jae hoon Jung

요약

자율주행차에게 카메라를 통한 곡선 차선 인식과 이를 통한 주행은 매우 중요한 작업이다. 이 연구에서는 직각 차선과 유턴 차선 등 곡선 주행에 대해 연구하였고, 각각 비슷한 V[]값과 후진을 이용하여 직각 차선 인식 불가 문제와 최대 회전각 문제를 해결하였다. 또한 자율 주행 자동차가 직각 구간을 지날 때 효율적으로 움직일 수 있도록 하는 알고리즘을 연구하였고, 곡선 차선 인식 판단의 순서를 바꾸어보는 것을 제안한다.

Keywords: 자율주행자동차, 차선인식, 곡선 주행

I. 서 론

자율주행 자동차란 운전자가 차량을 운전하지 않아도 스스로 움직이는 자동차를 의미하며, 카메라를 사용하여 곡선 차선을 감지한다. 자주차를 사용하여 곡선 차선을 주행해본 결과 하나의 곡선 길을 지나는 것은 문제가 없었지만 여러 개의 곡선 코스가 이어져 있는 경우(유턴, 직각 + 곡선 등) 각각의 판단이 뒤섞여 길을 인식하지 못하는 경우가 많았고, 유턴시 최대 회전각으로 회전하였음에도 불구하고 회전각의 부족 문제로 코스를 이탈하는 경우가 발생하였다. 이 연구는 여러 개의 곡선 코스가 이어져 있을 경우 곡선차선 인식에 순서를 부여하여 효율적이고 유동적으로 곡선 차선을 통과할 수 있도록 하는 것을 목적으로 한다.

II. 문제점 인식

자주차의 곡선 차선은 크게 2가지, 직각 코스와 부드러운 곡선 코스로 나누어지고, 이 곡선 차선들에서 해결해야 할 문제점은 다음과 같다.

1. 직각 코스의 정확한 판단과 안정적으로 지나갈 수 있는 방안
2. 부드러운 코스를 이은 유턴 코스에서 회전각 부족으로 인한 코스 이탈
3. 곡선 코스 이후 바로 이어진 곡선코스에 대한 인식 문제

기본 차선보정을 위한 steer값은 +25를 하였고, V[0] ~ V[6]까지는 세로축 인식값을, R[]과 L[]은 각각 오른쪽 가로축, 왼쪽 가로축 인식값을 의미한다.

III. 연구방법

1. V[]값 경향성에 따른 기울기 인식

```
for i in range(0,6):

    if V[i] - V[i+1] > 10 and (V[i] and V[i+1]) <270:
        incliation += 1
    elif  V[i] - V[i+1] < -10 and (V[i] and V[i+1]) <270:
        incliation -= 1

print(incliation)


if -3 < incliation < 3 and V[3] >= 130:
    state = "변화없음"
    print("변화없음")
    if 325<= L[1] + R[1] <=335 and 155<= L[0] + R[0] <=165:
        print("직진")
    elif R[1] < L[1]:
        print("오른쪽 휨")
        steer = steer - 25
    elif R[1] > L[1]:
        print("왼쪽으로 휨")
        steer = steer + 25
```

그림 1. incliation 값 축적을 통한 기울기 인식

자주차의 V값 경향성을 판단하여 차선의 휘어짐과 현재의 상태를 파악하도록 설계하였다. 자주차는 n번째 V값과 (n+1)번째 V값의 차이를 판단하여 “incliation” 값을 축적시킨다. 이런 식으로 하면 전반적인 경향성이 파악되기 전까지는 incliation이 3미만으로 유지되도록 할 수 있기 때문에 (직선도로의 경우에는 축적되는 incliation값이 오른쪽 차선 인식과 왼쪽 차선 인식이 반대여서 서로 상쇄된다.) 특정 v값으로 인한 판단오류의 확률을 낮출 수 있다.

또한 두 v값 차의 절댓값이 10미만일 경우 미세한 차이로 오류를 일으킬 수 있어 이 경우에는 배제하고 incliation 값에 반영시키지 않았다.

정재훈 bonanja2004@gmail.com

표 1. V[n+1] - V[n]값에 따른 incliation값 변화

V[n+1] - V[n] > 10	축적값 + 1
V[n+1] - V[n] < -10	축적값 - 1
-10 < V[n+1] - V[n] < 10	축적값 변화없음

$$incliation = \sum_{n=1}^{6} f(n)$$

수식 1. f(n)이 V[n+1] - V[n]값에 따른 incliation의 변화량이라고 할 때 incliation 값의 표현

2. 우회전과 좌회전 판단

```
elif V[2] < V[3] < V[4] and incliation <= -2 and V[3] < 100:
    print("우회전")
    state = "우회전"
    steer = -15 * incliation + 25
    velocity = 50



elif incliation >= 2 and V[2] > V[3] > V[4] and V[3] < 100:
    print("좌회전")
    state = "좌회전"
    steer = -15 * incliation + 25
    velocity = 50
```

그림 2. incliation값을 통한 우회전, 좌회전 판단 기준

위의 경우와 달리 incliation의 절댓값이 특정값 이상이 되면 자주차는 incliation의 부호에 따라 우회전 또는 좌회전으로 인식하고 steer값을 바꾼다. steer값을 V[]에 따른 기울기 값으로 할 수도 있지만 오류가 발생하는 경우가 많았기 때문에 다음과 같이 간략화해서
+15 * incliation으로 설정하였다.

3. 직각 코스에 대한 판단

직각차선의 경우에는 V값의 대부분이 비슷한 값을 가지는 특징이 있다. incliation 값의 경우 오류를 최소화하고자 V값의 차 10미만은 incliation의 축적에서 배제하였는데, 이로 인하여 비슷한 값들이 모두 제대로 처리되지 않는 문제가 발생하였다. 이 값들이 연산되지 않으면서 주변의 오류값이 incliation값에 포함되었고, 직각차선를 안정적으로 분석하지 못했다. 이를 해결하기 위해서 기존의 우회전, 좌회전과는 다른 직각 회전 알고리즘을 설계하였다. 이 알고리즘에서는 직각 코스에서 v값의 차가 얼마나지 않는 것을 역으로 이용한다. 직각 코스에서 v값의 변화를 분석하는 도중 직각 코스에 정면으로 들어갈 경우 모든 v값이 한 줄의 격자무늬로 들어가는 것을 확인하였다. 모든 v값의 차의 절댓값이 20이하로 붙어있는 경우를 직각 회전구간으로 정의하였고, 오류값이 있을 경우를 대비하여 5개 이상의 점들이 모여있는 경우로 조건을 설정하였다. 또한 직각 구간으로 판단한 이후 v[3]값이 일정 값 이상이 되기 전까지 steer값을 일정하게 설정하였다. 만약 직각차선을 정면으로 들어가지 않을 경우 차선은 우회전 또는 좌회전으로 인식되며, 직각 차선의 진행방향과 다른 회전을 한 경우 뒤의 후진 알고리즘을 통해 해결하였다.이 알고리즘으로 시도한 결과 정면에서는 직각 구간으로 인식하였고, 안정적으로 구간을 통과하였다.

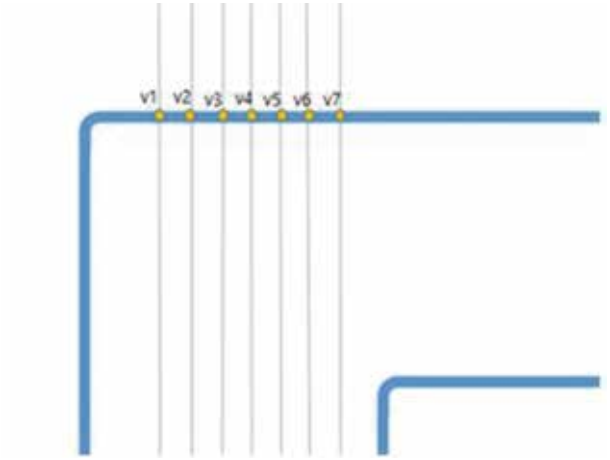


그림 3. 직각 차선에서 V값이 일정한 이유

```
b =0
global c

for i in range(0,6):
    if -20 <= V[i] - V[i+1] <= 20:
        b += 1

if b >= 5 and V[0] < V[6] and V[3] < 150:
    velocity = 40
    steer = 25 + 70
    print("직각 우회전")
    c =1

if b >= 5 and V[0] > V[6] and V[3] < 150:
    velocity = 60
    steer = 25 - 70
    print("직각 좌회전")
    c = 2

if c == 1:
    steer = 25 + 90
    velocity = 60
    if V[3] >= 200:
        c = 0

if c == 2:
    steer = 25 - 90
    velocity = 60
    if V[3] >= 200:
        c = 0
```

그림 4. 직각 차선 알고리즘

4. 유턴 코스의 회전각 문제

유턴 차선에서 일반적으로 회전할 경우 최대 회전각으로도 한 번에 회전할 수 없어 약 110도 정도 회전했을 때 차선이 카메라 인식 범위에서 벗어나면서 차선을 이탈한다.

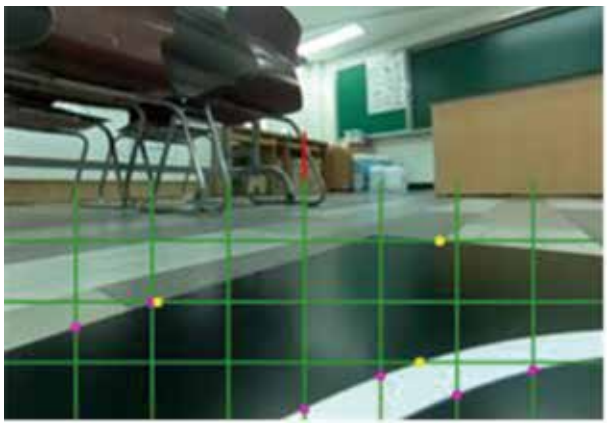


그림 5. 약 110도 부근에서 차선을 이탈한 경우

후진을 통해 카메라가 차선을 인지할 수 있는 각도를 확보하기 위해서 v[3] 값이 50 이하인 경우 후진 알고리즘을 추가하였다.

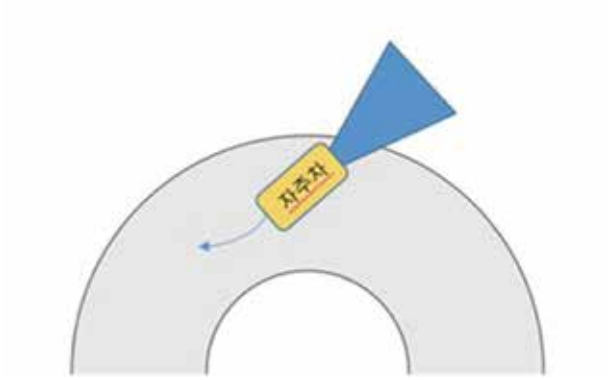


그림 6. 자주차의 시야문제와 후진 방향

후진 알고리즘에서는 더 효율적으로 방향을 틀기 위해서 오른쪽 후진과 왼쪽 후진을 구분하였다. 후진 코드를 통해 실행한 결과 유턴차선의 통과가 가능함을 확인하였다.

5. 효율적인 차선판단

직각 코스에서 실험 결과 기존의 알고리즘뿐만 아니라 후진을 이용한 알고리즘 또한 안정적인 주행이 가능하였다. 후진만을 이용한 알고리즘을 사용할 경우 기존의 알고리즘보다 더 안정된 주행이 가능하였지만 직각 차선에서 평균적으로 2번의 후진을 하였기 때문에 동선에 낭비가 발생하였다. 이를 통해 곡선 주행시 효율적으로 경로를 찾기 위해 기존의 알고리즘을 통합한 체계를 설정하였다.

최종적인 차선 판단 알고리즘은 [표 2]와 같다. 차선의 경향성 판단 후 V[3]값을 통해 카메라가 차선이탈 전인지를 판단한다. 이후 직각 차선으로 확정할 수 있는지를 판단하고 지각차선으로 확정지을 경우 밑에서 우회전, 좌회전으로 인식하는 것에 비해 더 효율적으로 구간을 통과할 수 있다. 만약 직각 차선으로 인지하지 못하더라도 후진과 우,좌회전 알고리즘을 통해 구간을 통과 가능하다.

표 2. 알고리즘의 간략화

IV. 연구결과

이 연구에서는 곡선 차선들에서 발생하는 문제점들을 파악하고 상황에 맞는 해결방안을 탐색하였고, 여러 곡선 차선의 판단에 있어 순서에 변화를 주어 자주차가

상황에 따라 유동적이고 대부분의 상황에 대처가능하도록 설계하였다. 또한 판단 순서에 따라 직각 차선과 같은 특수한 경우 최적의 경로로 이동하게 하여 효율적으로 이동할 수 있는 방법에 대해 연구하였고, 실제로 주행시간이 단축되는 것을 확인할 수 있었다.

V. 제언

1. 이어진 곡선 코스에 대한 문제점

실험도중 유턴 후 한 블록의 직선 뒤 바로 곡선 코스가 진행되었을 때 자주차가 차선을 이탈하는 사례가 발생하였다. 이 문제점은 차가 곡선 주행 후 직선코스에서 다시 정확한 경로를 잡지 못해 발생한 것으로, 곡선 코스 후 다시 직선코스 진입시 정확한 직선 경로로 이동하지 않아 오류가 발생한 것이다. 현재 알고리즘에서는 아래의 코드와 같이 유동적인 자주차의 움직임을

```
if 325<= L[1] + R[1] <=335 and 155<= L[0] + R[0] <=165:
    print("직진")
elif R[1] < L[1]:
    print("오른쪽 턴")
    steer = steer - 25
elif R[1] > L[1]:
    print("왼쪽으로 턴")
    steer = steer + 25
```

그림 7. 직진 차선 알고리즘

위해 L과 R값의 합을 범위로 지정한 것이 오류의 발생 원인으로 보여진다. 이를 해결하기 위해서는 직선차선에서 R[]과 L[]의 정밀한 조정을 통해 다음 곡선차선에 대한 대비를 하는 것이 필요하다.

2. 차선 인식의 한계

카메라를 통한 차선인식에서 사진과 같이 다른 두 차선이 나란히 있는 경우 우 차선이 동시에 인식되는 오류가 발생하였다. 이를 해결하기 위해서는 가장 가까운 차선을 기준으로 인식을 해야하지만 가까운 차선을 기준으로 하여도 V[]가 일정한 경향성을 보이는 것에는 문제가 있었다. 이로 인해 incliation값이 제대로 축적되지 않는 문제가 발생하였다. 이를 해결하기 위해서는 v[]값을 더 세분화하여서 많은 정보를 기준으로 판단하는 것이 필요하다.

참고문헌

신희석, 정성학, 김정하.(2022).자율주행 Tractor - Trailer 의 후진 직각주차 System 을 위한 경로 및 제어 Algorithm개발. 제어로봇시스템학회 논문지,28(2),109-118.

3

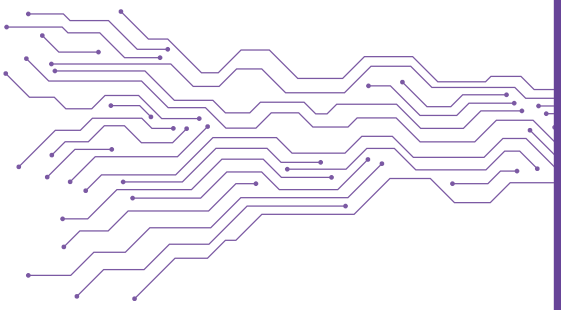
우수 논문

Journal of Youth Engineering

04 pure pursuit 기반의 주행 알고리즘과 전방 목표지점의 적합한 거리 설정에 대한 시뮬레이션 연구

하나고등학교

박민서, 박선우, 신정원, 심유민, 조정훈



pure pursuit 기반의 주행 알고리즘과 전방 목표지점의 적합한 거리 설정에 대한 시뮬레이션 연구

A Simulation Study on the Driving Algorithm Based on Pure Pursuit and the Appropriate Distance Setting of the Look-ahead Point

박민서*, 박선우, 신정원, 심규민, 조정훈
Minseo Park*, Sunwoo Park, Jungwon Shin, Gyumin Shim, Junghoon Cho

요약

경로 추종은 자율주행차의 주행 과정의 필수적인 요소이다. 특히 곡선 도로를 매끄럽게 주행하기 위해서는 정확한 경로 추종을 바탕으로 한 조향이 매우 중요하다. Pure pursuit은 기하학적 경로 추종 방법의 하나로, 이 방법을 사용하면 전방 목표지점을 선택하고 제어 입력을 실시간으로 계산할 수 있는데, 이는 주어진 경로가 매끄럽지 않거나 경유지를 거치는 경로를 지정할 때 유리하다. 본 연구는 Pure pursuit을 적용한 알고리즘을 개발하고 전방 목표지점까지의 거리를 조작변인으로 하여 시뮬레이션을 통해 테스트되었다. 실험 결과 차량이 이전의 방법을 사용하는 것보다 제안된 알고리즘을 사용할 때 원하는 경로를 더 정확하게 추종하였으며, 전방 목표지점까지의 거리가 1.0일 때 가장 오차가 적었다.

Keywords: 자율주행자동차, pure pursuit. 주행 알고리즘, 전방 목표지점, 시뮬레이션

I. 서 론

1. 자율주행 자동차

자율주행 자동차의 작동 원리는 기본적으로 '인지-판단-제어'의 3단계로 구성되어있다. 먼저 사용자는 목적지를 입력하고 경로를 결정한다.

인지단계에서는 카메라·라이더(LiDAR)·GPS·레이더(RADAR) 등차량의센서와 V2X(Vehicle-to-Everything) 시스템과의 통신을 통해 주변 환경에 대한 정보를 수집하고 인지한다. 여기서 V2X 시스템은 V2V (Vehicleto-Vehicle), V2I(Vehicle-to-Infrastructure), V2P(Vehicle-to-Pedestrian) 통신으로 구성된 지능형 교통 시스템이다. 이 단계에서는 운전에 필요한 능력인 도로의 환경 및 경로, 다른 차량의 차선 변경 및 끼어들기 등의 의도를 인지함으로써 사람의 눈과 같은 역할을 한다.

판단단계에서는 인지 단계에서 얻은 환경 정보를 이해하여 주행상황을 판단한다. 사람을 대신해 조향 및 가·감속을 결정해 차량이 스스로 경로에 따라 이동하도록 한다. 이때 장애물·교통신호 등을 반영하여 안전하고 적절한 주행 경로를 생성한다. 이처럼 인지 단계에서 얻은 정보를 바탕으로 주행전략을 수립하는 판단단계는 사람의 두뇌와 같은 역할을 한다.

마지막 제어단계에서는 판단단계에서 결정된 판단을 제어 시스템을 통해 실제 주행에 적용하고자 운전 시스템을 제어한다. 이는 실질적으로 차량의 속도 조절, 조향각/토크 조절, 방향 제어 등에 해당하므로 사람의 혈관이나 근육과 같은 역할을 한다.

연구를 진행하면서 사용한 자율주행 자동차의 경우 앞뒤에 모두 달린 카메라와 거리를 측정하는 센서 라이다를 통해 주변 상황의 정보를 인지한다. 얻은 정보는 작은 컴퓨터인 라즈베리파이와 알고리즘을 작동할 외부 컴퓨터와 연결하기 위한 공유기를 통해 알고리즘이 판단을 내린다. 마지막으로 알고리즘이 결정한 정보들을 바탕으로 차량의 핵심부품을 제어한다. 여기서 핵심부품으로는 앞바퀴의 방향을 바꾸는 서보모터, 자동차의 구동력을 결정하는 스텝모터, 곡선(curve) 도로를 주행할 때 안쪽 바퀴와 바깥쪽 바퀴가 다른 거리를 돌게 하여 미끄러지지 않게 도와주는 차동기어가 있다. 특히 정교한 속도제어가 가능한 스텝모터와 서보모터를 이용하여 주행을 제어하게 된다.

2. 조향 장치

2.1. 애커먼-장토식 조향 장치

조향장치는 운전식의 조향 핸들을 회전시키고 각 링크기구를 움직여 좌우 앞 바퀴의 방향을 바꾸어 차량의 주행 방향을 제어할 수 있도록 하는 장치이다. 이상적인 선회 방법은 차량의 각 바퀴의 궤적이 움직이는 방향이 모두 동심원이 되도록 하는 것이다. 그렇기 위해서는 내측 차륜이 외측 차륜의 조향각도보다 커야 함을 알 수 있다. 대부분의 차량 및 본 연구에서 사용한 자율주행차는 전자축 양쪽바퀴로 조향조작을 하는 전륜 조향 방식을 따른다.

애커만 방식은 양쪽의 바퀴가 방향전환을 하고자 하는 방향으로 평행하게 방향을 바꾼다. 하지만 이 방법대로 따르면 어느 한쪽의 바퀴가 미끄러지면서 회전되어 타이어가 심하게 손실 마모된다는 단점이 있다.

박민서 andymspark@gmail.com
박선우 psw7490@gmail.com
신정원 has_20099@hana.hs.kr
심규민 solar9179@gmail.com
조정훈 justin8021@kakao.com

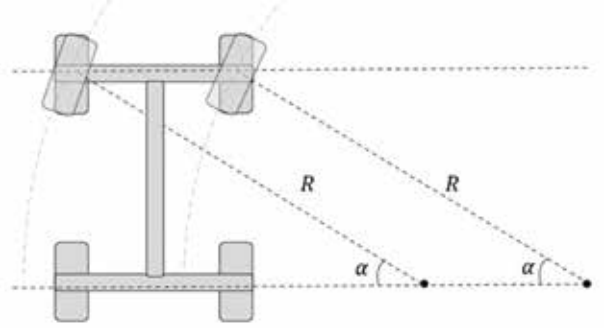


그림 1. 애커만 방식의 조향, α : 조향각, R : 주어진 조향 각으로 차축이 움직이는 원의 반경. 두 앞바퀴가 각각 같은 크기의 반지름을 가진 원 궤도를 같은 각속도로 따라가므로 외측 차륜의 마모가 발생한다.

이를 보강한 방법이 애커만-장토식인데, 양쪽의 앞바퀴 축의 중심선과 뒷바퀴 축의 중심선이 방향을 바꾸고자 하는 쪽의 1점에 교차하는 기구를 설치한다. 그 결과 조향 암과 타이 로드를 둘러싼 부분은 사다리꼴이며, 타이로드가 좌우로 움직이면 바퀴도 함께 움직이는데, 각 막대의 길이 및 각도에 따라 좌우 바퀴가 꺾이는 각도가 달라진다.

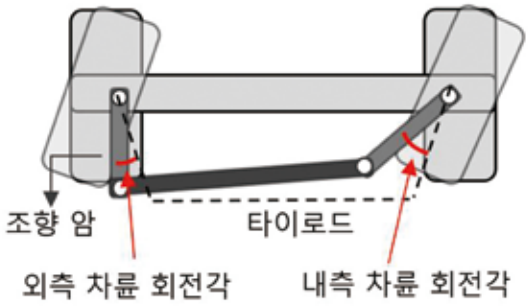


그림 2. 애커먼-장토식 조향 장치. 조향 방향에 따라 내측 차륜 회전각의 크기가 외측 차륜의 회전각보다 크다.

이를 적절하게 조절하면 자동차의 네 바퀴가 항상 동일한 회전축을 가지며 코너를 돌 수 있다. 이때 회전반경은 자동차가 선회할 때 앞바퀴의 외측이 그리는 최소 반경으로, 회전반경이 작을수록 작은 원으로 회전할 수 있다.

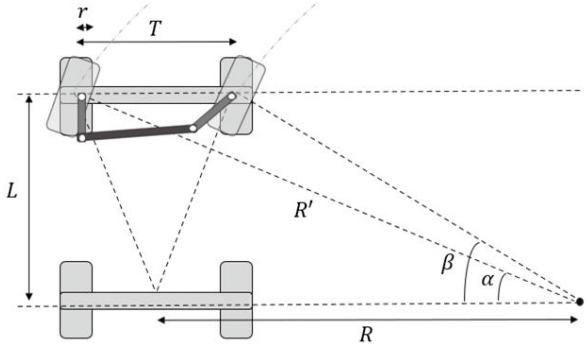


그림 3. 애커먼-장토식 조향 장치. α : 외측 차륜의 조향각, β : 내측 차륜의 조향각, R : 차축 중심선과 회전 중심점 사이 거리, R' : 최소 회전반경, L : 축거, T : 윤거, r : 킹핀 중심선에서 타이어 중심선까지 거리

$$\alpha = \tan^{-1}\left(\frac{L}{R+\frac{T}{2}}\right) \quad (1)$$

$$\beta = \tan^{-1}\left(\frac{L}{R-\frac{T}{2}}\right) \quad (2)$$

$$R' = \frac{L}{\sin\alpha} + r \quad (3)$$

여기서 α는 외축 차륜의 회전 중심각이며, β는 내측 차륜의 회전 중심각이다. 이때 최소 회전반경 R'는 외측 차륜의 회전 반지름과 같다. 회전반경이 작을수록 차량은 큰 곡률로 회전할 수 있다.

2.2. 랙 앤드 피니언형 조향기어 조향 장치

랙 앤드 피니언 조향기어 조향 장치는 조향 핸들을 움직이면 피니언 샤프트의 선회 운동이 랙 기어의 왕복 운동으로 변환되어 이것이 타이로드와 조향 암을 움직여서 조향 너클과 바퀴를 회전시킨다.

본 연구에서 사용한 자율주행차 모형은 다음과 같은 규격의 애커먼 장토식 조향 장치와 랙 앤드 피니언 조향 기어를 이용한다.

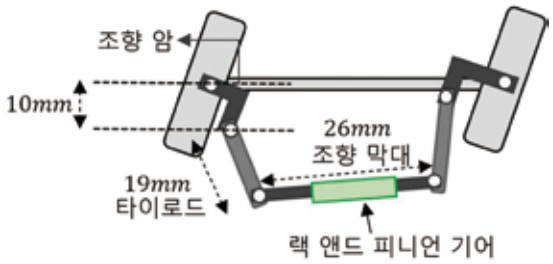


그림 4. 본 연구의 모델이 된 자율주행차 모형의 조향 장치

3. 기존 알고리즘의 문제점

앞서 소개한 자율주행 자동차의 작동 원리인 '인지-판단-제어'에 기반하여 기존 알고리즘을 살펴보면, 기존 알고리즘은 카메라, 라이더, 라즈베리파이로 이미지 정보를 수집하고 왜곡된 정보를 바로잡는다. (인지) 그리고 이미지 속 차선의 후보를 고르고 추세선이 될만한 차선을 판단한다. (판단) 마지막으로 추세선으로부터 얻은 steer값으로 서보모터의 가동범위를 제어한다. (제어) 이 알고리즘을 활용해 대회 준비 및 실험을 진행하면서, 추세선으로부터 steer값을 고를 때 문제점이 발생하였다. 추세선의 후보군 중 중앙에 가까운 추세선을 비교할 때 사용하는 상수값은 임의의 실험값으로 오차범위가 크고 상황에 따라 다른 값이 필요하다. 이로 인해 서보모터의 방향 제어 실패로 인한 차선 이탈이 빈번하게 일어났다. 이를 개선하고자 본 연구에서는 pure pursuit을 기반으로 한 알고리즘을 개발하고 시뮬레이션을 통해 가장 적합한 전방 목표지점까지의 설정 거리를 제안하고자 한다.

Ⅱ. 기존 알고리즘

1. 기본 원리

추세선 기법은 python에 있는 opencv라이브러리의 canny edge detection기능을 적극 활용한 것으로, 화면 내에 있는 차선 후보를 전부 탐지한다. 이 후보들은 실제 차선일수도 있고, 차선이 아닌 노이즈일수도 있다. 이 후보군 중 화면의 중앙에서 가장 가까운 선을 차선으로 선택해 추세선을 그리고, 그 추세선의 기울기를 취해 적절한 상수값을 곱해줘 steer값으로 한다. 이때 물리적으로 가능한 steer값은 -80~80도 이므로 steer값을 상한을 설정해 서보모터의 가동 범위 안으로 제한해준다.

2. 코드

본 연구에서 사용한 자주차의 알고리즘의 개략적인 메커니은 다음과 같다.

1. 카메라로 이미지를 촬용
2. 자주차의 라즈베리파이에서 이미지 왜곡 보정
3. 컴퓨터에 이미지를 전송
4. process함수에 이미지를 파라미터 형식으로 전송
   이미지를 이용해 velocity, steer값을 결정
5. 라즈베리파이에 피드백.

여기서 중점적으로 들여다봐야 할 부분은 바로 process함수이다. process함수에서 중요한 장면은 이미지에서 차선 후보를 추출하는 장면, 차선 후보 중 차선을 선정하는 장면 이렇게 두 부분으로 나누어 볼 수 있다.

2.1 이미지에서 차선 후보를 추출하는 장면

※코드는 문서 하단 참고
jajucha패키지의 processFront함수는 정면 이미지 사진을 정제하여 그 안의 추세선 후보를 리턴해 주는 기능을 가지고 있다. processFront함수를 들여다보면, findLines함수에서 그 선들을 찾고 있음을 알 수 있고, findLines함수에서는 opencv의 Canny Edge Detection기능을 이용해 이미지의 윤곽을 감지하고, 그 이미지의 특정 y좌표까지만 인식하도록 이미지를 자른 뒤 warpPerspective함수를 이용해 이미지를 위에서 내려다 본 것처럼 처리해 차선 후보를 추출한다.

2.2 차선 후보 중 추세선을 뽑아낼 차선을 선정하는 장면

frontLines에는 차선 후보가 모두 담겨 있다. 이 차선 후보들 각각의 x, y좌표값들을 이용해 np.polyfit과 np.poly1d를 이용해 1차함수 꼴의 추세선을 그리고, y=400에서의 x값을 구한다. 이를 통해 y=400에서 어떤 차선이 가장 중앙에 가까운지 비교하고, 가장 가까운 추세선을 우리가 관측해 steer값 피드백에 이용할 추세선으로 선정한다. 여기서 400은 임의의 실험값이다.

Ⅲ. pure pursuit 알고리즘

1. pure pursuit

기하학적 경로 추종 방법은 차량과 경로 간의 기하학적 관계를 이용하며, 그 종류로는 Stanley method와 pure pursuit 이 있다. 이 방법은 현재의 차량 위치와 주어진 경로 상의 적합한 목표지점을 선정해 이를 추종하기 위한 조향 각을 계산한다. Stanley method는 스티어링 각도를 계산해 전방 휠과 경로에서 가장 가까운 지점 사이의 방향의 가로 방향 거리와 각도 차이를 줄이는 방식이다. 이 방법은 고속 주행에 적합하며 코너를 잘 추종하지만, 전진 주행만을 위해 개발되었으며, 불연속 곡률에서 불안정해지는 경향이 있다. pure pursuit은 현재 차량의 위치로부터 일정 거리에 위치한 전방 목표지점을 설정하여 조향 각도를 계산한다. 이 점을 잘 선택하면 Stanley method에 비해 매끄럽지 않은 경로에서 더 안정적으로 주행할 수 있다. 또한 이 방법은 구현이 간단하므로 가장 널리 사용되는 방법이다. 본 연구에서는 기하학적 경로 추종 방법 중 pure pursuit 조향 방식을 사용한다.

pure pursuit에 대한 단순화된 식을 유도하기 위해 자전거 모델을 도입한다. 자전거 모델은 경로 추종을 위해 차량을 단순화시킨 모델로 자동차의 두 앞바퀴와 뒷바퀴를 각각 하나로 결합하여 자전거같이 주행하는 것으로 해석한다.

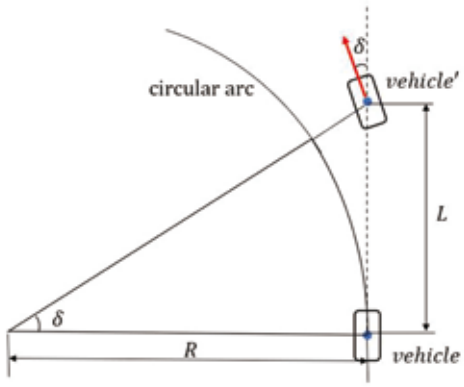


그림 5. δ : 조향각, L : 전륜 축과 후륜 축 사이의 거리, R : 주어진 조향 각으로 후륜 차축이 움직이는 원의 반경.

$$\tan\delta = \frac{L}{R} \qquad (4)$$

pure pursuit을 통한 조향각은 다음과 같이 유도된다.

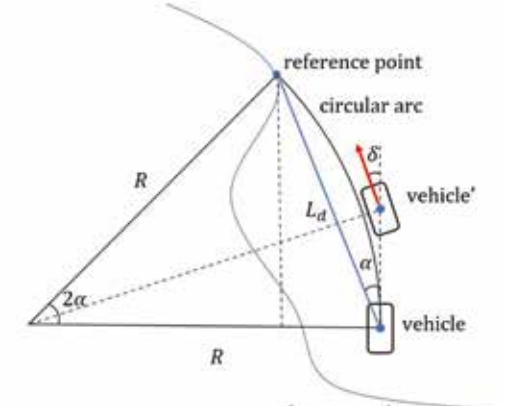


그림 6. δ : 조향각, L_d : 전방 목표지점까지 거리, α : 차량과 전방 목표지점 사이 각, R : 주어진 조향 각으로 후륜 차축이 움직이는 원의 반경

$$\frac{L_d}{\sin 2\alpha} = \frac{R}{\sin(\frac{\pi}{2} - \alpha)} \quad (5)$$

$$\frac{L_d}{\sin \alpha} = 2R \quad (6)$$

$$k = \frac{1}{R} = \frac{2\sin\alpha}{L_d} \quad (7)$$

이때 자전거 모델에 의해 식은 다음과 같이 정리된다.

$$\tan\delta = \frac{L}{R} = kL \quad (8)$$

$$\delta = \tan^{-1}(kL) = \tan^{-1}(\frac{2L\sin\alpha}{L_d}) \quad (9)$$

최종적으로 차량이 경로를 추정하기 위한 조향각은 ()의 식을 이용해 알고리즘에 적용한다.

2. 코드

pure pursuit알고리즘의 핵심은 gridFront함수이다. gridFront 함수는 화면을 3개의 가로선과 7개의 세로선으로 나누고, 각 가로선이 차선과 만나는 점의 좌표를 알 수 있게 해준다. 그리고 중앙 차선과 그 점까지의 거리를 알 수 있으므로 이를 이용해 error 값을 구할 수 있다.

우선 jajucha 패키지의 gridFront함수에 frontImage를 넘겨줘 필요한 정보를 받아온다. 중앙으로부터의 차선까지의 거리는 gridFront 내부에서 Canny 함수를 이용해 알아낸다.

다음은 단순한 계산이다. 우리가 목표로 하는 global point가 왼쪽에 있는지, 오른쪽에 있는지 판단하고, error값을 계산((R[1]-L[1])/2) 하여 steer값을 결정하는 수식에 대입해준다. 이때, arctan이 return 하는 값은 rad 단위이고, 자주차가 인식하는 steer값은 도 단위이므로, 57.3을 곱해 rad단위를 도 단위로 대응시켜준다.

Ⅳ. 시뮬레이션

1. 시뮬레이션 방법

실험을 위한 주행 경로는 꼭짓점이 둥근 형태의 경로를 사용하였다. 직선 경로와 곡선 경로를 번갈아 연결한 형태를 실험용 경로로 선택하였다.

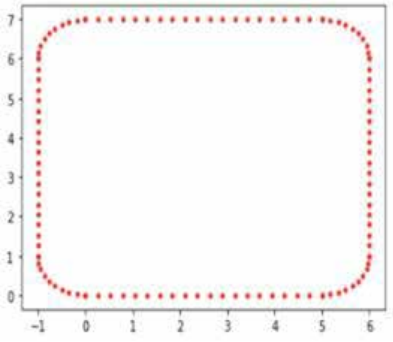


그림 7. 시뮬레이션을 위한 주행 경로

Numpy의 linspace를 이어 붙여 그림 1과 같은 경로를 구현하였다. 하나의 직선경로에 20개, 하나의 곡선경로에 10개의 원소를 주었고 크게 (-1,-1)부터 (7,7)까지를 꼭짓점으로 가지는 그림 1과 같은 끝이 둥근 정사각형 형태의 경로를 선택하였다.

총 120개의 linspace상의 점과 자율주행자동차의 시점 좌표 간 직선 거리를 계산하여 Ld값에 가장 가까운 점의 인덱스를 찾아 그 점을 초기 목표점으로 잡아 조향각을 계산한다. 이때 자율주행자동차의 좌표는 중심의 좌표에서 차의 길이 L의 절반을 이용하여 차의 뒤쪽을 계산하여 사용하였다. 조향각 계산에는 $\tan\theta = \frac{L}{R}$을 이용하였다. 목표점이 직선 경로상일 때는 자율주행자동차와 경로와 거리를 R값으로 사용하였고 곡선 경로상일 때는 곡선경로가 반지름 1인 사분원을 사용하여 만든 것을 고려하여 그 중심부터 자율주행자동차까지의 거리를 R값으로 사용하였다.

계산된 조향각 방향으로 각 프레임 당 나타내는 시간인 dt와 초기 속도를 곱한 값만큼 자율주행자동차를 이동시키고 다시 목표점을 찾는다. 이때 이전 프레임의 목표점을 기준으로 좌우로 가까운 인덱스부터 조사하여 알고리즘의 시간 복잡도를 낮추었다. 찾은 인덱스를 바탕으로 다시 조향각을 조정한 뒤 나아가는 작업을 반복적으로 진행하는 형태로 알고리즘을 구성하였다. 목표점이 경로상 마지막 점으로 설정되거나 그 인덱스를 넘어가면 알고리즘을 종료 시키고 matplotlib을 통해 그림2와 같이 자율주행자동차의 경로를 표현하였다.

이후 경로와 각 프레임별 자율주행자동차의 경로를 pandas를 통해 csv형태로 내보냈다. 시점 상 오차를 고려해 기본 시점 (-2,-0.5)에서 시점을 조금씩 변경하여 데이터를 생성했다.

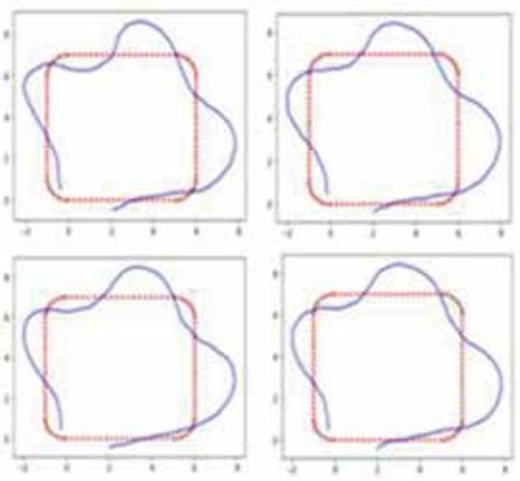


그림 8. 초기값에 따라 생성된 다양한 데이터

예측력 평가 방법으로는 NRMSE (Normalized Root Mean Square)를 사용하였다. 이는 모두 0에 가까울수록 예측한 값이 실제 값과 유사하다는 것을 의미하며, 아래와 같이 도출한다.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \qquad (10)$$

$$\text{NRMSE} = \frac{RMSE}{Max(y_i) - Min(y_i)} \qquad (11)$$

여기서 y_i는 i번째 실제 값을 의미하며 $\hat{y_i}$는 I번째 예측값을 의미한다.

2. 시뮬레이션 결과

pure pursuit에서 전방 목표지점까지의 거리를 조작변인으로 하여 설정된 트랙과 시뮬레이션 사이의 오차를 연산하였다.

각각의 전방 목표지점까지의 거리에 따른 주행 경로는 다음과 같다.

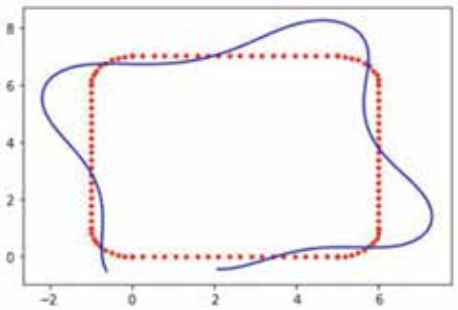


그림 9. (전방 목표지점까지 거리)=1.0일 때 주행경로

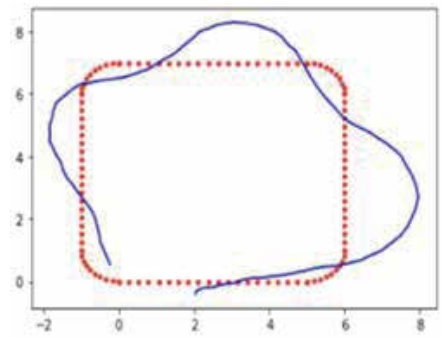


그림 10. (전방 목표지점까지 거리)=2.0일 때 주행경로

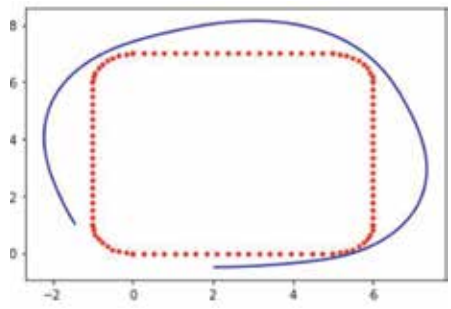


그림 11. (전방 목표지점까지 거리)=3.0일 때 주행경로

설정된 트랙과 시뮬레이션의 오차는 아래의 그래프로 나타내었다. 여기서 가로축을 t축, 세로축은 x좌표 값 또는 y좌표 값들이다.

이때 시뮬레이션은 전방 목표지점까지의 거리마다 각각 10번 거쳤으며, 그렇게 얻은 평균적인 x, y축 값들을 빨간 점으로 표시하였다. 10차례의 랜덤한 시뮬레이션을 통해 발생한 각 시각에서의 오차는 빨간 점을 지나는 각각의 막대들로 표시하였다. 또, 기준 트랙의 x, y축 좌표값은 파란 점들로 나타냈다. 그리고 t축을 근방으로 형성된 검은 막대들은 시뮬레이션 값과 트랙의 실제 값 사이의 오차를 나타내었다.

전방 목표지점까지의 거리를 1.0으로 설정했을 때 결과는 표 (1)~(4)와 같다.

표 1. t-x좌표 그래프, time step 0~60

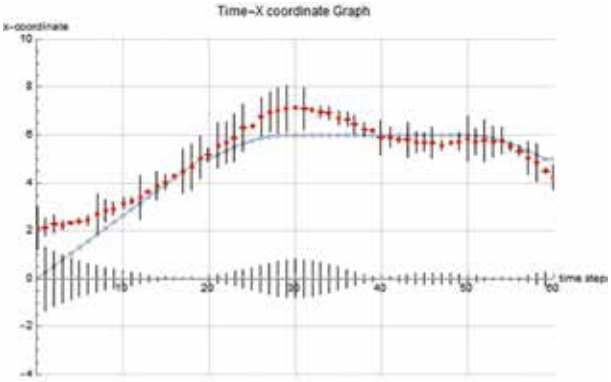


표 2. t-x좌표 그래프, time step 61~120

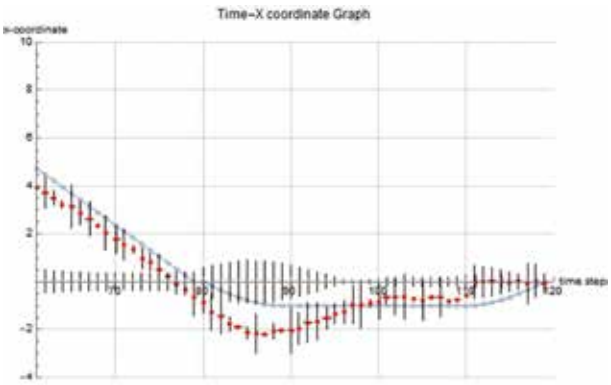


표 3. t-y좌표 그래프, time step 0~60

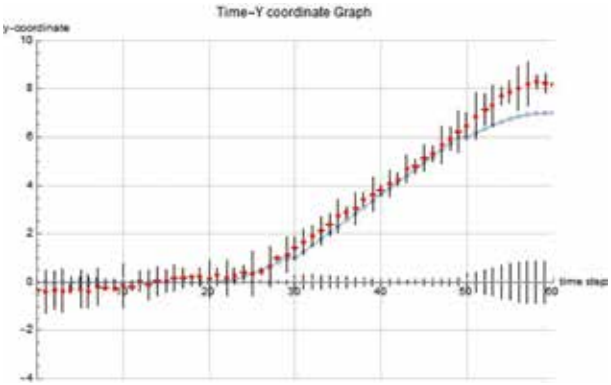
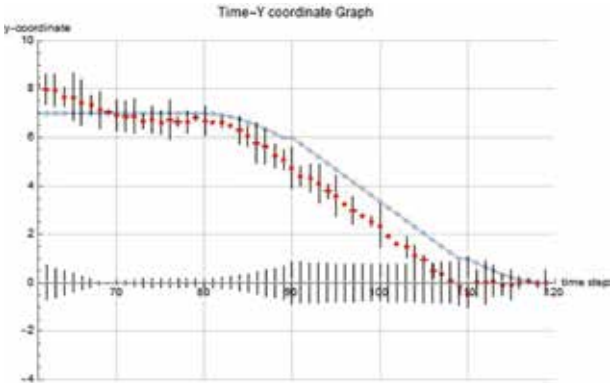


표 4. t-y좌표 그래프, time step 61~120



전방 목표지점까지의 거리를 2.0으로 설정했을 때 결과는 표(5)~(8)과 같다.

표 5. t-x좌표 그래프, time step 0~60

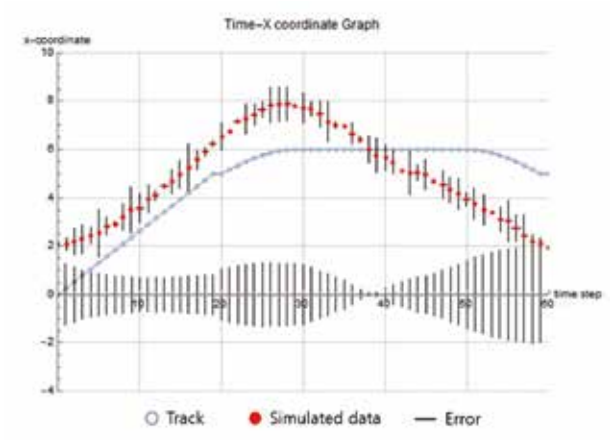


표 6. t-x좌표 그래프, time step 61~120

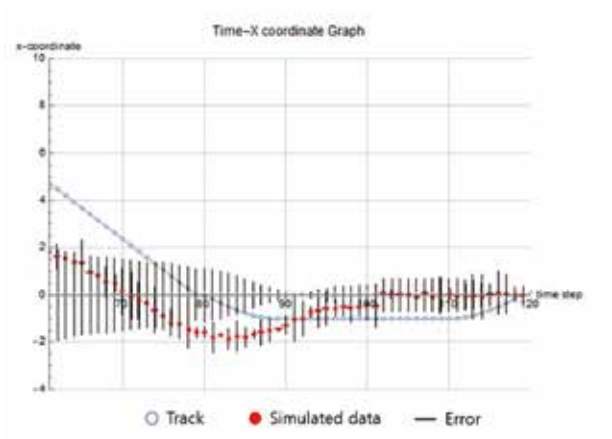


표 7. t-y좌표 그래프, time step 0~60

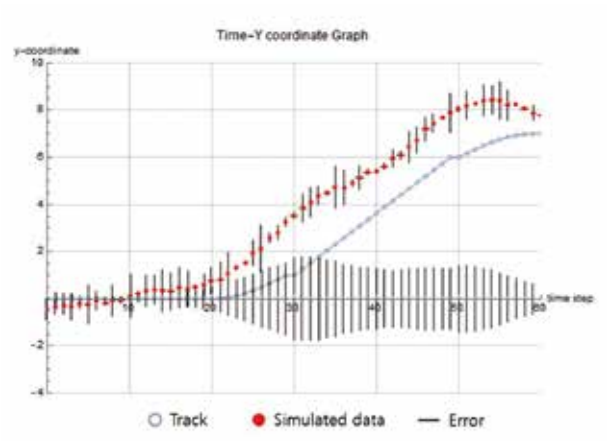
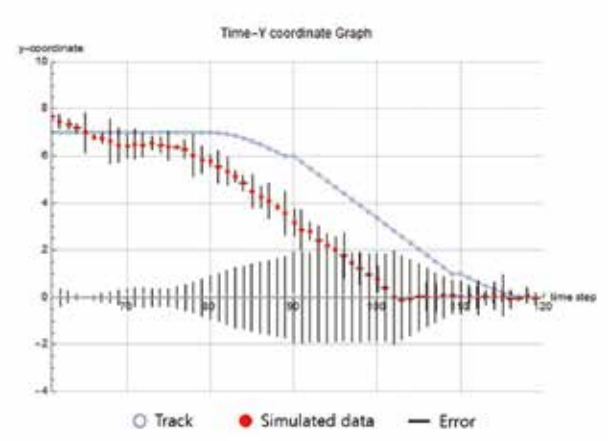


표 8. t-y좌표 그래프, time step 61~120



전방 목표지점까지의 거리를 3.0으로 설정했을 때 결과는 표(9)~(12)와 같다.

표 9. t-x좌표 그래프, time step 0~60

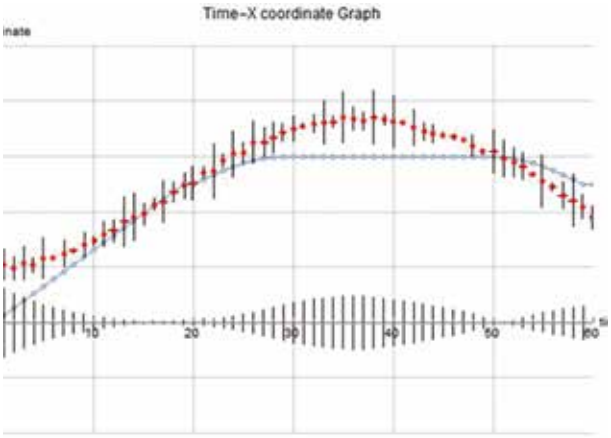


표 10. t-x좌표 그래프, time step 61~120

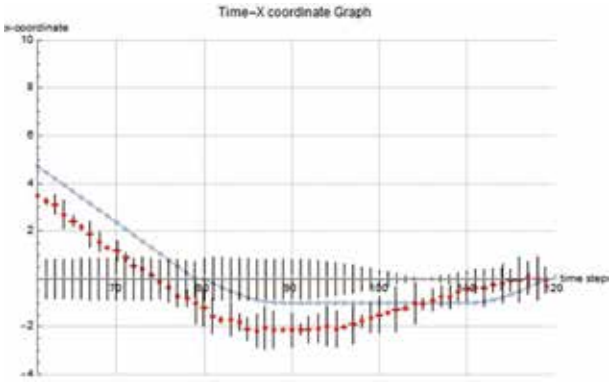


표 11. t-y좌표 그래프, time step 0~60

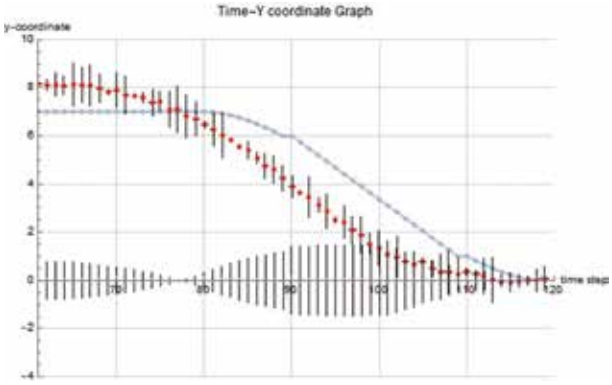
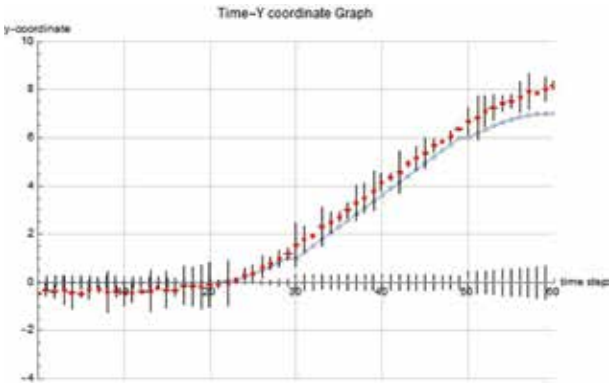


표 12. t-y좌표 그래프, time step 61~120



전방 목표지점이 1.0, 2.0, 3.0일 때의 그래프를 모두 비교해 보면 다음 표 (13)~(14)과 같다. 여기서 검은색은 1.0, 빨간색은 2.0, 파란색을 3.0일 때의 모습이다.

표 13. t-x좌표 그래프, time step 0~120

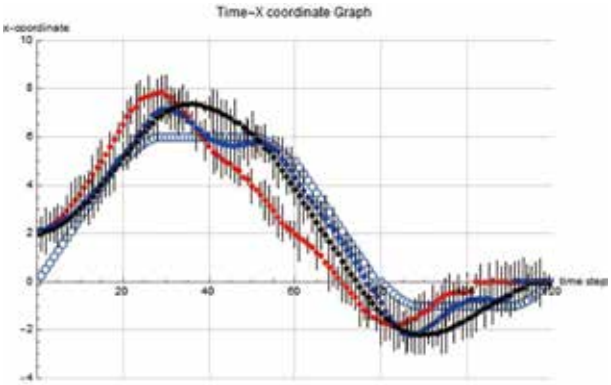
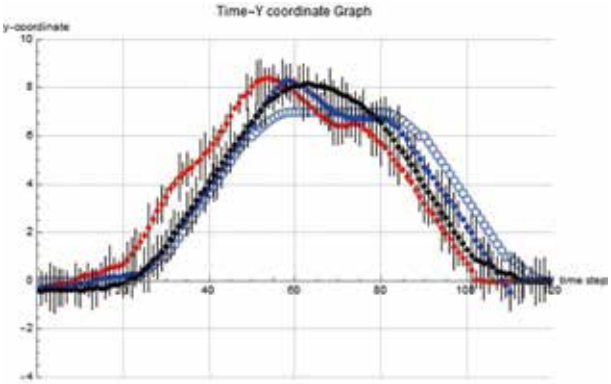


표 14. t-y좌표 그래프, time step 0~120



3. 데이터 해석

시뮬레이션 결과, 전방 목표지점까지의 거리에 따른 실험값과 트랙의 NRMSE 는 다음과 같았다.

표 15. 전방 목표지점까지의 거리에 따른 NRMSE

전방 목표지점까지 거리	NRMSE(%)
1.0	7.82
2.0	11.35
3.0	9.12

랜덤한 초기값을 기반으로 도출된 NRMSE의 평균은 다음과 같았다.

표 16. 랜덤한 초기값에 따른 NRMSE의 평균

전방 목표지점까지 거리	NRMSE(%)
1.0	2.16
2.0	2.54
3.0	2.63

Ⅳ. 결론 및 제언

1. 결론

전방 목표지점까지의 거리에 따른 실험값과 트랙의 NRMSE를 통해 우리가 설정한 차량의 속도에 따르면 전방 목표지점까지의 거리를 1.0으로 설정했을 때 매우 안정적인 주행(<10%)을 하고 있음을 알 수 있다. 또, 주행 경로 및 오차를 분석한 결과 차량이 트랙에서 완전히 벗어나거나 다른 길로 가는 경우는 없는 것으로 보인다.

랜덤한 초기값을 기반으로 도출된 NRMSE의 평균을 통해 본 연구에서 사용된 pure pursuit 기반의 알고리즘이 매우 안정적으로 추정하고 있음을 알 수 있으며, 초기조건에는 거의 영향을 받지 않는 것으로 보인다.

2. 제언

본 연구에서는 20개의 원소로 구성된 직선 구간과 10개의 원소로 구성된 곡선 구간으로 이루어진 트랙에서 시뮬레이션을 진행하였으며 실험값과 트랙의 오차분석을 진행했다. 그러나 시뮬레이션의 한계도 분명히 존재한다. 해당 알고리즘을 실제 자율주행 자동차에 적용하였을 때 다양한 요인들에 의하여 노이즈 및 추가적인 오차가 발생할 수 있기 때문이다. 따라서 아래와 같은 후속 연구가 추후에 필요하다.

1) 자율주행 자동차를 이용한 실험

시뮬레이션에서 설정했던 트랙을 구현하고, 실제 자율주행 자동차에 알고리즘을 적용하여 시뮬레이션과 비교, 분석해 본다.

2) 다양한 형태의 차선에서의 실험

시뮬레이션 상에서 제작한 트랙(직선, 곡선)에서만 적용해 보았기에 다른 형태의 차선에서는 적용하기 힘들다는 점이 있다. 차선 종류의 예시로는 점선 차선, 교차로, 오르막길 등이 있다. 점선 차선의 경우 우려되는 부분은, 현재 시뮬레이션을 통해 전방 목표지점까지의 거리(L_d)를 1.0으로 설정했을 때 매우 안정적으로 주행한다는 결과를 도출할 수 있었는데 주기적으로 끊기는 점선 차선의 상황에서도 주어진 트랙을 벗어나지 않고, 알고리즘에서 인식 오류가 뜨지 않는지 확인해보아야 한다. 또한 연구에서 사용한 자율주행 자동차 차량은 평평한 도로 위에서만 실험이 진행되었기에 오르막길, 내리막길에서도 개발한 알고리즘이 잘 작동되는지에 대한 연구도 필요하다.

3) 변화하는 곡선 구간의 곡률에 따른 실험

본 연구에서는 제작한 트랙을 통제 변인으로, 전방 목표지점까지의 거리(L_d)를 조작변인으로 설정하여 시뮬레이션을 진행해 보았다. 다만 실제 곡선 도로들에는 곡률이 다양하다. 대회용 트랙에서만 하더라도 커브A형, B형이 있고, 주위의 도로환경에 따라서 곡률이 범위 내에서 다양한 값을 가진다. 따라서 곡률을 값을 변화시키면서 차량이 주어진 경로를 추종하는지를 판단하고 이에 따른 오차값을 구하는 실험이 필요하다.

참고문헌

[1] 정석우, 심현철, “자율주행 자동차의 인공지능”, 기계저널 v.57 no.3, pp.42 – 45,2017

[2] 이현우. "자율주행차량의 안정적인 곡선 경로 추종을 위한 차량 무게 배분을 고려한 횡 방향 오차 보정 Pure Pursuit에 관한 연구." 국내석사학위논문 국민대학교 자동차산업대학원 자동차공학과, 2018.

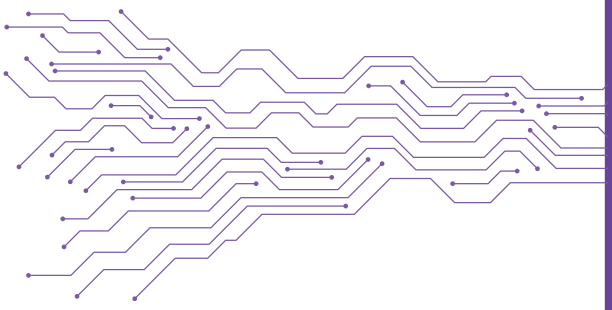
[3] 안준우 외 3인, “Accurate Path Tracking by Adjusting Look-Ahead Point in Pure Pursuit Method”. International Journal of Automotive Technology, vol. 22, no.1, pp. 119-129. 2021.

[4] 주형진, 이기범. “Pure-Pursuit 알고리즘의 피드백 제어 이득과 전방주시거리 설계에 따른 자율주행 자동차의 경로 추종 성능 변화”, 한국자동차공학회, vol. 29, no.9, pp. 839-846, 2021

[5] 김민정, 정대한, 김회경, “영상기반 차량인식 기법을 이용한 교통류 추정에 관한 연구”, 한국 ITS 학회, vol.18, no.6, pp. 110~123, 2019

05 자율주행 차선 인식 알고리즘 개선에 대한 고찰

하나고등학교
정진용, 권능주, 김도현, 김민재, 김준식



자율주행 차선 인식 알고리즘 개선에 대한 고찰

Improvement of Lane Recognition Algorithm in Self-Driving

정진용*, 권능주, 김도현, 김민재, 김준식
Jinyong Jung*, Neungjoo Kwon, Dohyun Kim, Minjae Kim, Junsik Kim

요약

본 연구에서는 기존의 자율주행 알고리즘을 통해 확인한 문제점을 해결하기 위한 다방면에서의 개선 작업을 진행하였다. 가우시안 블러와 ROI 및 허프 변환을 적용한 직선 검출 알고리즘, 기울기를 이용한 곡선 구간에서의 조향 함수 정의 등 차선 인식을 중심으로 한 알고리즘의 작성 및 이론적 증명 과정을 거쳤다. 더불어 허프 변환 스텝 증가값 등 수학적 최적화를 통해 파라미터 값을 도출하였다.

Keywords: 자율주행자동차, 차선 인식, 가우시안 블러, ROI, 허프 변환, 조향 함수, 최적화

I. 서 론

본 연구는 기존의 자율주행 알고리즘을 바탕으로 주행 과정에서 발생한 각종 시행착오를 이론적으로 분석하고자 하였다. 알고리즘 설계에 관여하는 여러 영역 중에서도 주행 안정성 측면에서 핵심이 되는 차선 인식에 초점이 맞추어져 있다.

기존 알고리즘은 카메라로 이미지를 받아오고, Canny edge detection을 활용해 차선의 위치를 인식하며, 차선 위 점의 좌표값을 받아 numpy 모듈의 poly1d 함수를 이용해 일차함수 형태의 추세선을 구하는 방법을 사용하였다. 이렇게 얻어 온 일차함수의 기울기에는 가중치를 주어 조향각을 조절하도록 했다. 또한, 운행 시 차량은 차선의 중앙에서 주행해야 하므로 사전 측정 데이터(차량이 차선 정중앙에 놓였을 때 나타나는 좌표들을 포함하는 일차함수 추세선의 기울기)와 비교하여 기울기를 일정한 값으로 유지할 수 있도록 조향을 조절하는 방식을 사용하였다.

그러나 카메라 인식 정보를 바탕으로 추세선을 도출하는 과정에서 사람의 움직임이나 빛 반사를 비롯한 외부 장애물의 영향을 받는다는 점을 확인할 수 있었다. 차선의 위치에 해당하는 점을 잘못 인식하여 눈에 보이는 것과는 다른 추세선을 검출하는 경우가 대다수였고, 곡선 타일을 통과할 때에는 추세선을 긋지 못 하여 경로를 이탈하기도하며, 더불어 출발 지점이 한쪽 차선으로 치우쳐 있을 때 도로의 중앙을 찾아가지 못 하여 주행이 불안정한 경우도 존재했다. 따라서 올바른 차선 인식을 통해 차량이 안정적으로 주행할 수 있는 알고리즘으로의 개선 작업이 필요하다고 판단하여 연구를 진행하게 되었다.

본 연구에서는 이미지의 노이즈 제거를 시작으로 장애물의 영향을 줄이기 위한 인식 범위의 설정, 직선과 곡선의 효과적인 검출과 더불어 조향 함수의 정의에 도달하기까지 차선 인식 과정 전반에 대한 고찰을 진행하였다. 특히 경험에 기반한 수치 탐색과 이론적으로 입증하는 과정을 거쳐 최적의 차선 인식 알고리즘을 구상 및 적용하였다.

II. 본론

1. 가우시안 블러링

1.1. 가우시안 블러링의 효과

가우시안 블러링(Gaussian Blurring)이란 이미지의 노이즈와 불필요한 그래디언트를 제거하기 위해 고안된, 가우시안 함수에 기반한 2D 컨볼루션 연산의 일종이다. Canny 에지 검출을 수행하기 전에 삽입하였으며, OpenCV의 GaussianBlur 함수는 src, ksize, sigmaX, sigmaY, borderSize 다섯 가지의 매개변수를 갖는다.

src는 블러링 대상이 되는 이미지 프레임 파일, ksize는 (width, height) 꼴의 커널 사이즈, sigmaX, sigmaY는 각각 x, y방향 표준편차를 의미한다. 즉, 2차원 가우시안 분포 함수를 근사해 2차원 필터 마스크 행렬을 생성하고, 커널 행렬의 크기와 표준편차를 매개변수로 취해 가중치를 부여하는 방식이다.

* 정진용 (하나고등학교, has_20168@hana.hs.kr)
권능주 (하나고등학교, has_20007@hana.hs.kr)
김도현 (하나고등학교, has_20020@hana.hs.kr)
김민재 (하나고등학교, has_20025@hana.hs.kr)
김준식 (하나고등학교, has_20044@hana.hs.kr)

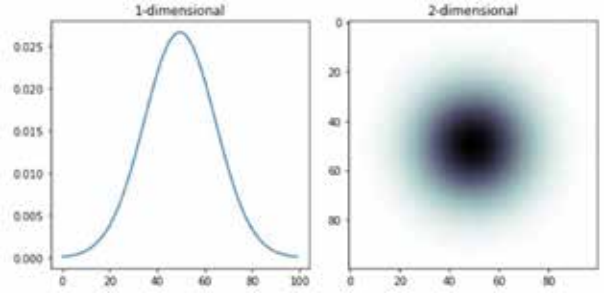


그림 1. 가우시안 커널 가중 방식 시각화

이러한 기본 개념을 바탕으로 주행 화면에 가우시안 필터링을 적용하면 다음 [그림 2]와 같다.

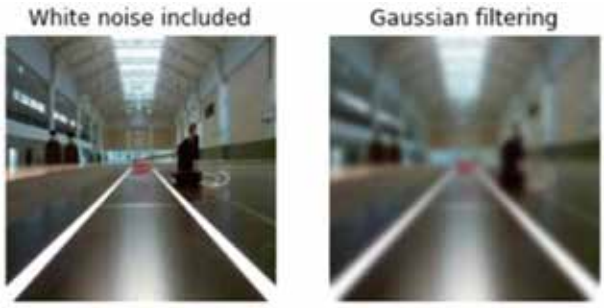


그림 2. 가우시안 블러링을 샘플 프레임에 적용한 결과

1.2. 가우시안 필터의 최적화

픽셀 수가 많으면 필터 크기가 커질수록 효율적일 수 있는데, 사용하는 파라미터 수가 증가하기 때문이다. 대상으로 하는 이미지에 따라 필터 크기를 달리할 필요성이 있다고 보아 차선 인식이 가능한 범위 내에서 연산량과 연산 시간을 최소화하는 최적의 표준편찻값을 이론적으로 도출해내어 알고리즘에 적용하는 것이 초기 목표였으나, 일반적으로는 최소한의 크기를 사용해도 연산 속도에 큰 차이가 없음을 확인하였다.

CNN 필터에서 대부분 3x3, 5x5 행렬을 사용한다는 점으로 미루어 보아 블러링 대상이 되는 이미지마다 픽셀 수에 따른 필터 크기의 최적화를 진행하지 않고 최소한의 크기를 채택하는 이유를 알 수 있었다. 연구에서 사용한 촬영 영상 프레임의 경우 3x3 커널로 충분했으나, 현실 환경을 대상으로 하면 이미지에 따라 사용하는 커널 수를 조절할 수 있도록 하이퍼 파라미터를 도입한 알고리즘을 설계하면 연산 효율을 높일 수 있을 것이라 기대한다.

더 나아가 절대적인 계산량을 최소화하여 함수 자체의 연산 속도를 증가시키는 방법으로는 선형 보간법을 활용할 수 있다. 선형 보간법이란 양 끝점이 주어질 때 그 사이에 있는 값을 추정하기 위해 직선거리에 따라 선형적으로 계산하는 방법인데, 두 개의 1차 가우시안 함수를 곱했을 때 2차 가우시안 함수를 결정할 수 있다는 가우시안 함수의 특성과 맞물려 효율을 높일 수 있다.

2. ROI 구간 설정

ROI(Region of Interest)는 이미지나 영상 내에서 관찰자가 관심을 보이는 부분을 뜻한다. 본 연구에서는 불필요한 영역에서의 차선 검출을 줄임으로써 연산 속도의 향상과 더불어 차선 인식의 정확성을 높이기 위해 전면 카메라에 ROI를 적용하였다.

단순 직진뿐만 아니라 좌회전, 우회전 모두 고려해야 하므로 해당 프레임에서의 차선이 ROI 영역 내에 포함되어야 함은 당연하다. 그리하여 최적의 ROI 값을 찾기 위해 사전에 촬영한 주행 영상 프레임을 토대로 영역을 설정하였으며, 방법은 다음 [표 1]과 같다.

표 1. 주행 영상 기반 ROI 영역 설정 과정 (ROI region setting process based on the driving image)

1단계	직진, 좌회전, 우회전 시 화면에 표시되는 프레임의 표본을 각각 000장씩 확보한다.
2단계	파이썬의 OpenCV와 HSV Extraction을 통해 각 프레임의 차선을 검출하고, 모든 프레임을 중첩하여 각 프레임의 60% 이상이 등장하는 픽셀을 포함하도록 n각형을 구상하며, 이 중 n이 최소가 되는 형태로 결정한다.

초당 20프레임의 주사율을 지니는 카메라를 사용할 때, 주행 1회당 평균 1,200장의 프레임을 출력한다. 이때, 각각의 상황별로 특징을 잘 드러낼 수 있는 이미지를 선별하여 표본으로 삼는다.

그림 3. 프레임 표본 중첩 및 n각형 도출 (Frame sample overlap and n-gonal derivation)

OpenCV와 Pillow, Numpy 라이브러리를 사용해 작성한 파이썬 코드는 다음 [그림 4]와 같다.

```python
def main():
    results = []
    img = cv2.imread('./sample/1.jpg')
    final = np.zeros_like(img)
    for i in range(0, 624, 1):
        path = './sample/'+str(i)+'.jpg'
        print(str(path))
        image = cv2.imread(path)
        hsvLower = np.array([0, 0, 250])
        hsvUpper = np.array([180, 10, 255])
        hsvResult = hsvExtraction(image, hsvLower, hsvUpper)
        results.append(hsvResult)
    for j in range(0,624,1):
        final = cv2.addWeighted(final,1,results[j],1,0)

    cv2.imshow('Line Detection', final)

    while True:
        #키 입력을 1ms 기다리고, key가 'q'면 break
        key = cv2.waitKey(1)&0xff
        if key == ord('q'):
            break

    cv2.destroyAllWindows()

def hsvExtraction(image, hsvLower, hsvUpper):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hsv_mask = cv2. inRange(hsv, hsvLower, hsvUpper)
    result = cv2. bitwise_and(image, image, mask=hsv_mask)
    return result
```

그림 4. 알고리즘의 구현

먼저 image를 불러온 후, 각각의 픽셀에서 hue(0~180°), saturation(0~10), value(250~255) 영역 이내의 색상이 포함된 픽셀만을 남겨두며, 나머지 픽셀은 모두 삭제한다. 이러한 방식으로 '0.jpg'부터 '624.jpg' 625개의 이미지에 대한 분석을 진행하고, 이를 addWeighted 함수를 통해 합성하여 하나의 이미지로 만든다. 이 이미지를 활용하여 영역 내부로 들어가는 ROI 경계선을 정하면 다음 [그림 5]와 같은 결과를 확인할 수 있다.

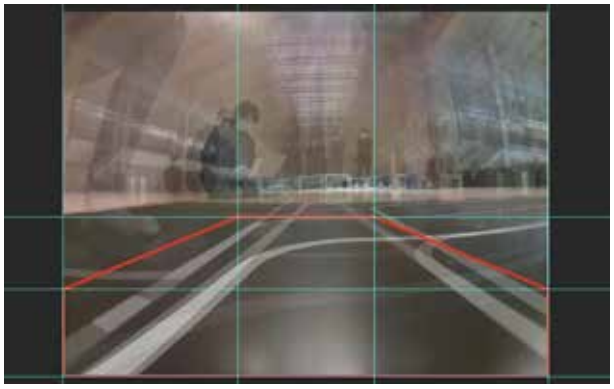


그림 5. 가장 간단한 n각형으로 구현한 ROI

위 코드의 실행 결과를 바탕으로 육각형을 이용하여 ROI 범위를 설정하였으며, 왼쪽 위 끝의 좌표를 (0, 0)으로 잡을 때 (230, 270), (410, 270), (639, 365), (639, 479), (0, 479), (0, 365)로 결정되었다.

3. 허프 변환 알고리즘

허프 변환(Hough Transform)이란 좌표축에서 x, y좌표값 대신 ρ(원점에서 직선까지의 거리)와 (x축으로부터 기울어진 각도)를 이용하여 직선을 나타내는 것으로, 차선의 기울기를 구하여 차량의 조향을 결정할 때 연산 효율성을 높이기 위해 채택하였다. 허프 변환은 HoughLines라는 함수명으로 OpenCV 모듈에 내장되어 있다.

기존 알고리즘처럼 Canny edge detection만을 이용해 기울기를 구할 경우, 추세선 정보를 통해 점을 구하고 이를 일차함수로 근사하는 과정이 필수적이었다. 그리고 이 과정에서 올바른 추세선을 구하지 못하는 경우가 다수 발생하여 본 연구팀은 허프 변환 알고리즘을 도입하여 직선을 검출하도록 하였다.

허프 변환은 image space 상에서 x, y로 표시되는 직선을 원점에서 직선까지의 거리를 나타내는 ρ와 x축으로부터의 각도를 나타내는 θ로 표현할 수 있는 $\rho - \theta$ parameter space 상에 나타나는 과정을 칭한다.

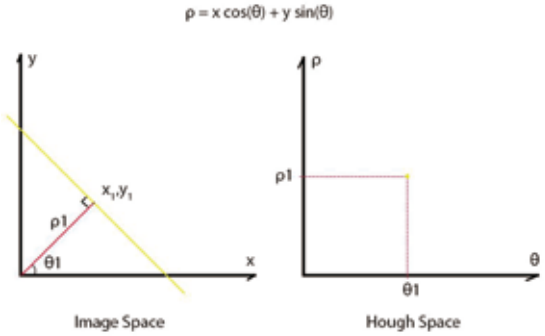


그림 6. 허프 변환 과정의 시각화

[그림 6]과 같이 Image Space 상의 직선을 Hough Space ($\rho - \theta$ parameter space)에서 하나의 점으로 표현되므로 기울기에 대한 정보를 이미 포함하고 있고, 컴퓨팅 자원을 아낄 수 있다는 장점이 있다. 따라서 본 연구팀은 직선 검출에 허프 변환 알고리즘을 사용하기로 하였다.

다만 OpenCV 라이브러리에 내장된 HoughLines 기본 함수에서는 입력된 이미지의 모든 픽셀을 대상으로 허프 변환을 수행하므로 효율성 측면에서 컴퓨팅 자원의 낭비가 발생한다고 볼 수 있다. 이에 따라 본 연구에서는 무작위로 픽셀을 선택하여 직선을 탐색하는 확률적 허프 변환(Probablistic Hough Transformation)을 사용하였다.

확률적 허프 변환은 HoughLinesP라는 함수를 사용하며, 매개변수로 주어진 조건에 해당하는 선분을 찾고 각 선분의 시작점과 끝점의 좌표를 반환한다. 각 함수의 사용법은 아래와 같다.

cv2.HoughLines(image, rho, theta, threshold)
cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]), min_line_length, max_line_gap)

매개변수의 차이에서 알 수 있듯, HoughLinesP 함수는 에지 검출을 통해 얻어낸 값을 반환해주어야 한다. 각 함수의 공통 매개변수인 rho, theta는 ρ값과 θ값을 얼마나 증가시키면서 조사할 것인지, 즉 스텝 증가 값을 묻는 것이며, threshold는 직선의 검출 기준을 나타내는 값으로 이를 넘어서면 직선으로 간주한다.

본 연구에서 사용한 HoughLinesP 함수의 매개변수 min_line_length는 얻고자 하는 선의 최소 길이로, 일정 이하로 짧은 선은 무시한다. max_line_gap은 선 위의 점들 간 최대 거리를 나타내는 것으로, max_line_gap보다 점 간 거리가 크다면 서로 다른 선으로 인식하도록 한다. 이 두 매개변수를 통해 기존 알고리즘과는 달리, 추가적인 판단 조건이 없는 명확한 검출이 가능하다. 직선 검출에 도달하는 과정을 요약하면 다음 [그림 7]과 같다.

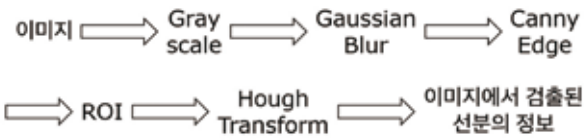


그림 7. 이미지로부터 직선 검출까지의 과정 (Process from image to straight line detection)

허프 변환을 적용한 파이썬 코드는 다음 [그림 8]과 같으며, 얻어낸 선분의 정보를 기존 이미지 위에 합성하여 차선을 시각적으로 확인할 수 있도록 실행한 결과는 [그림 8]로 나타낼 수 있다.

그림 8. 허프 변환 적용 코드

그림 9. 허프 변환 코드 실행 결과

결과적으로 커브 구간에서 불규칙적이었던 차선 검출 문제를 해결하였으며, 불필요한 장애물 인식이 감소하였다.

4. 허프 변환 스텝 증가값

4.1. 허프 변환 스텝 증가 값의 최적화 필요성

허프 변환 스텝 증가 값이 의미하는 바를 다음과 같이 정리할 수 있다. 첫째, 허프 변환 시 각도 θ를 일정 크기만큼 증가시키면서 직선을 탐색하게 되는데, 이때의 각도 증가 값을 말한다. 둘째, 허프 변환 시 원점으로부터 직선까지의 거리 ρ를 증가시키면서 직선을 탐색하게 되는데, 이때 거리의 크기 증가 값을 말한다.

이때 두 변수의 스텝 증가값에 따라 변환의 정확도나 계산 속도가 달라질 수 있는데 원활한 진행을 위해서는 최적화가 필요하다.

본 연구에서는 첫 번째의 각도 증가 최적화 값을 탐색하여 허프 변환을 수행했을 때의 정확도를 증가시킴과 동시에 연산량을 최소화할 수 있는 각도 증가 값을 찾는 것을 목표로 하였다.

4.2. 허프 변환 스텝의 수식화

허프 변환 스텝에서의 각도 θ를 수식화하기 위해 다음과 같은 방법을 사용하였다.

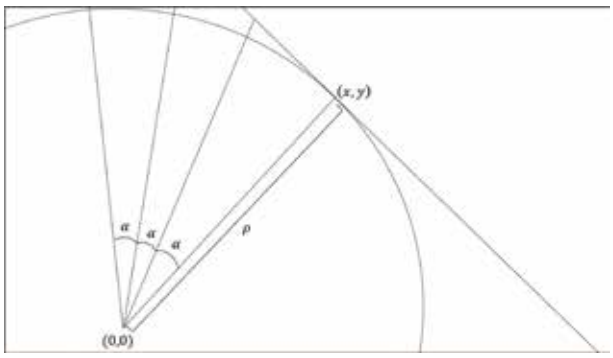


그림 10. 허프 변환 시의 각도 증가 값을 시각화한 모식도

위의 [그림 10]에서 볼 수 있듯, 허프 변환 시 각도를 α만큼 증가시키면서 직선을 탐색한다. (0, 0)에서 반지름이 ρ인 원을 생각할 수 있고, 이 원을 이용하여 원점으로부터의 길이가 ρ인 선분을 각도의 증가 값에 따라 여러 개 그을 수 있으며, 이것이 바로 일정한 각도의 증가 값을 따라 허프 변환이 진행되는 과정이다. 다만 이때 탐색하는 직선까지의 거리가 각도가 증가함에 따라 일정하게 ρ로 유지되는 것이 아닌, 거리가 증가하는 것임을 알 수 있다. 그러나 이는 자율주행 자동차가 차선을 탐색할 때, 차선이 아주 얇은 선이 아닌 일정한 두께를 가진 선이므로 값이 일정하더라도 허프 변환을 계속해서 이어갈 수 있다. 이러한 점에서 ρ를 고정한 채 α의 값을 증가시키면서 결과를 도출하는, 본 연구에서 채택한 방식이 타당함을 알 수 있다.

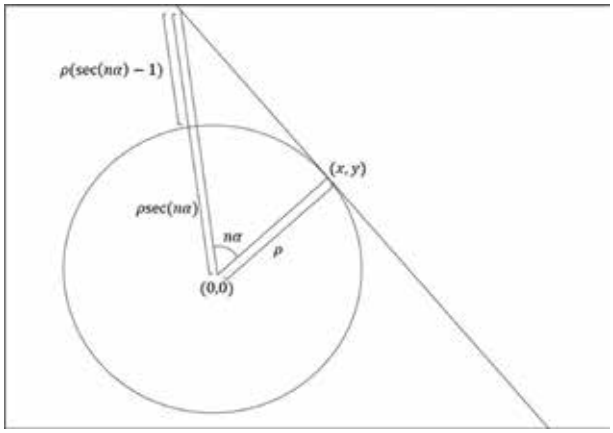


그림 11. 허프 변환 직선 탐색의 결과

다만 본 연구에서는 ρ가 일정하므로, [그림 11]과 같은 오차가 발생함을 알 수 있다. 원의 반지름의 길이가 이고 $n\alpha$이기 때문에 오차의 길이는 $(\cot(n)-1)$의 꼴로 나타남을 알 수 있다. 이때 일정 수치 이상의 오차가 발생하기 전까지 직선을 탐색한다고 가정하면, 해당 오차 전까지 몇 회에 걸쳐서 직선을 탐색했는지가 직선 탐색의 정확도와 귀결된다. 즉, 일정량의 오차에 도달하기 전까지 최대한 많은 탐색 횟수가 누적되어야 전체적인 직선 검출의 정확도가 높아진다는 것이다. 본 연구의 목적은 허프 변환에 따라 나타나는 위의 오차까지의 직선 탐색 누적 횟수가 최대가 되는 α의 증가 간격을 구하는 것이다.

고려해야 할 또 다른 사항으로 α의 값이 너무 작아질 경우 직선 탐색의 정확도 자체는 증가할 수 있으나, 계산의 횟수가 증가하여 오히려 효율성이 떨어질 수 있다. 그러므로 이 또한 유기적으로 고려할 필요가 있다. 계산의 편의를 위해 Wolfram Mathematica를 사용하였으며, 다음과 같은 순서로 계산을 진행하였다.

우선 각 시행에 따라 오차가 축적되는 것으로 판단하였으며, 위의 [그림 11]을 참고하면 각 시행에 따른 오차는 $(\sec n-1)$의 값으로 나타나므로 이를 더한 값을 전체 오차로 설정하고, 전체 오차의 최솟값을 찾는다. 이를 Wolfram Mathematica에 구현하기 위해 다음과 같은 식을 사용하였다.

$y=\sec x$ 함수에서 정의역이 $y=nx$의 바닥함수로 나타나는 $y=nx$함수의 차를 구해주게 되면 오차의 전체 합과 같고, 이 식을 x에 대해 적분하면 오차의 전체 합을 계산할 수 있다. 이때 n은 스텝 증가 값을 의미하는데, 즉 n이 클수록 전체 스텝 횟수가 줄어들며 동시에 계산 횟수도 줄어들게 되지만, 오차가 커질 경향성이 있다. 계산의 단순화와 실제 적용에서의 편의를 위해 n은 자연수로 설정했으며, Wolfram Mathematica의 이산 플롯(n의 값이 이산적으로 나타날 때의 그래프를 보여주는 것)을 이용해 계산할 수 있다. 식은 다음과 같다.

위의 식에서 x의 범위가 의미하는 것은 각도의 시작과 끝값인데, 이때 탐색이 각도가 0일 때부터 시작하므로 시작 각도가 0인 것은 당연하다. 실험에서 x의 값은 여러 종류의 값으로 설정하여 여러 번의 결과를 해석한다. 또한, 위의 식에서 적분값을 n으로 나눈 이유는 계산 횟수가 증가할수록 계산 속도가 느려질 가능성이 있기 때문이며, n 값이 클수록, 즉 전체 스텝 횟수가 작을수록 적분 값이 작도록 역수를 곱하였다.

4.3. 허프 변환 스텝 증가의 수치 대입 결과 및 해석

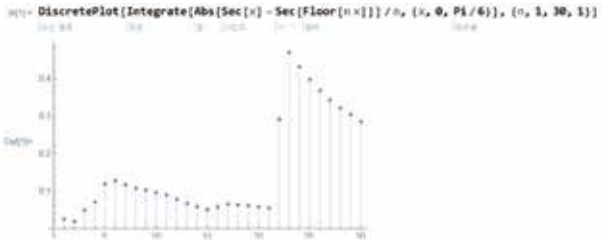


그림 12. $x=\frac{\pi}{6}$일 때의 계산 결과

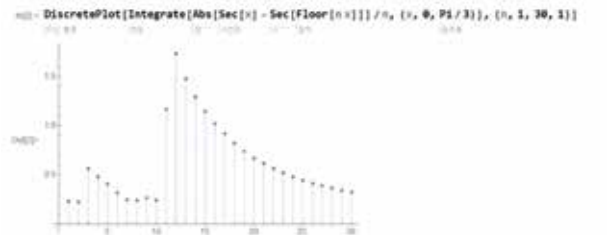


그림 13. $x=\frac{\pi}{3}$일 때의 계산 결과

두 차례의 연산에서 $x=\frac{\pi}{6}$인 상황에서 n이 2, 15, 21일 때 극솟값을 가지며, 절대적으로 작은 n의 값은 n이 1, 2, 3, 15일 때임을 [그림 12]를 통해 확인하였다. $x=\frac{\pi}{3}$일 때는 n이 2, 8, 10일 때 극솟값을 가지며, 절대적인 값이 작은 n은 1, 2, 8일 때라는 점을 [그림 13]을 통해 확인하였다. 차선의 두께나 카메라의 각도를 비롯한 자율주행 자동차의 주행 환경이 변수로 작용할 수 있는데, 개별 상황에서의 실험을 진행해야 하며 이때 n값을 사용할 수 있다. 즉, 스텝 증가 값이 1, 2, 3, 8, 10, 15, 21일 때를 실험을 통해 확인해야 한다는 뜻이다.

5. 조향 함수 정의

기존 조향 함수는 인식된 차선의 추세선을 직선으로 구한 뒤 이를 활용하여 조향각을 정하는 방식이었다. 그러나 앞서 기술한 바와 같이 추세선 대신 허프 변환 알고리즘을 통한 직선 검출 방식을 채택하였기에 조향 함수의 조정 또한 이루어져야 한다. 허프 변환으로 검출한 직선이 차선 자체를 나타내므로 따로 추세선을 구할 필요가 없으며, 변경한 조향 함수의 알고리즘은 다음과 같다.

우선 차선의 기울기를 구한다. 이는 3차원의 공간을 2차원 이미지로 바라볼 때 원근에 따라 차선이 소실점을 향하여 양쪽 차선의 기울기의 부호가 다른 점, 그리고 차선의 꺾인 정도가 기울기로 곧바로 나타난다는 점을 이용하기 위해서이다.

그림 14. 차량이 소실점을 향하는 모습 (Car heading towards the vanishing point)

왼쪽 위 끝의 픽셀 좌표가 (0, 0)이고, 오른쪽 아래로 갈수록 x, y좌표가 증가하므로 좌측 차선의 기울기는 음수, 우측 차선의 기울기는 양수이다. 이를 바탕으로 좌측과 우측의 차선을 구분하고 각 차선의 기울기를 구한다. 이때 허프 변환을 통해 하나의 직선만 검출되는 것이 아니므로, 검출된 직선들의 평균 기울기의 절댓값을 구하여 좌측, 우측을 나타내는 1행 2열의 리스트로 저장했다. 양 혹은 음, 한쪽의 기울기가 하나도 없으면 그에 해당하는 좌측 혹은 우측의 차선이 인식되지 않은 것으로 판단해 -1의 기울기 값을 반환하였다.

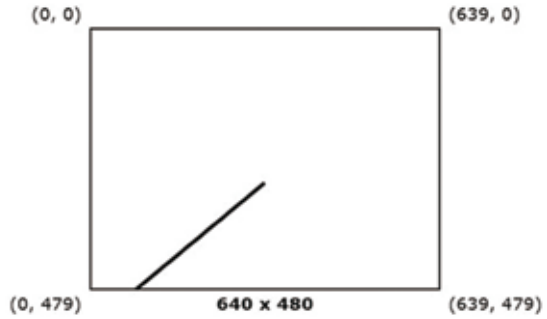


그림 15. 이미지의 x-y좌표

검출된 모든 직선을 기울기의 부호에 따라 구분한 후 단순히 평균을 내면 실제 차선이 아닌 다른 곳에서 검출된 직선까지 계산에 포함하므로, 이를 방지하기 위해 기울기의 절댓값의 임계를 설정하였다. 이미지의 바닥에 대해 수평인 차선이 검출되는 경우는 발생하지 않기에 0.2 이하의 기울기를 갖는 직선을 제외하는 방식을 택했다. 이것을 적절한 ROI 설정과 병행하여 다수의 경우에 차선만을 인식할 수 있었다. 간혹 차선 외의 물체가 직선으로 검출되는 경우에도 검출한 직선의 평균 기울기를 구하기 때문에 그 값이 조향에 미치는 영향을 줄일 수 있었다.

ROI를 바닥에 가깝도록 아주 낮게 설정했고, 카메라가 비추는 각이 낮아서 급격한 커브에서는 한쪽 차선만 인식되는 것을 확인할 수 있었다.

그림 16. 우측 90도 커브에서 한쪽 차선만 인식되는 모습 (Single lane detected from the right 90 degree curve)

반대로 한쪽 차선만 인식되는 경우는 급격한 커브뿐이라고 생각할 수 있기에, 두 가지 상황으로 나누어 조향각을 다르게 설정하였다.

(i) 양쪽 차선이 모두 인식되는 경우

기본적으로 차선을 따라 직진해야 하지만, 조향각을 0으로 고정할 경우 한쪽으로 틀어져 주행할 가능성이 매우 크다. 이를 해결하기 위해 'bias'라는 개념을 도입하였다. 양쪽 차선의 x좌표의 평균에서 이미지 중앙의 x좌표인 320을 뺀 것을 bias로 설정하고, 이를 조향각에 더하였다. 차량이 차선의 중앙으로 주행할 수 있도록 bias가 조향각을 보정하는 원리는 다음과 같다.

주행 중인 차량이 차선의 중앙보다 왼쪽에 있는 경우, 차선은 상대적으로 오른쪽으로 편향되어 진행한다. 따라서 이때 차선의 x좌표의 평균은 이미지의 중앙인 320보다 커져 bias가 양수가 된다. 이것이 조향각에 더해지면 차량은 설정된 조향각보다 더 오른쪽으로 틀어져 다시 차선의 중앙으로 주행할 수 있다. 반대의 경우도 원리가 같다.

(ii) 한쪽 차선만 인식되는 경우

급격한 곡선 주행 트랙을 지날 때 한쪽 차선만 인식되는 경우가 발생한다. 이 경우 직선 도로의 기울기인 0.7에서 많이 벗어날수록 조향의 정도가 기하급수적으로 커져야 차선을 벗어나지 않을 수 있었다. 따라서 조향 정도의 절댓값을 $|\alpha| = \nu^{|x-0.7|} - 1$로 설정했으며, x는 차선의 기울기, ν는 7 이상 15 이하의 상수이다. 차량이 급격한 커브를 돌아 다시 직선 차선이 카메라에 비추어질 때 그 기울기가 최대 1.3까지 증가했다가 다시 감소하는 현상이 나타났기에, 차선의 기울기가 0.7보다 커져도 조향각이 커지도록 설정하였다. 이후 수차례 시행착오를 거쳐 적절한 임곗값을 찾는 과정을 밟아 위의 수식을 결정했다.

Ⅲ. 결론

본 연구에서는 자율주행 자동차가 받아오는 이미지에 canny, 가우시안 블러, ROI를 적용하고, 허프 변환 중에서도 확률적 허프 변환을 적용함으로써 직선 검출 성능을 향상하였다. 얻어낸 결과값을 바탕으로 조향 함수 설계로의 적용이 가능했으며, 기존의 자율주행 알고리즘을 바탕으로 차선 인식을 중심으로 한 알고리즘 보완 과정을 거치며 크게 세 가지의 결론 및 제언 사항을 제시할 수 있었다.

우선 대회용 자율주행 환경에 대해 알고리즘을 설계한다는 전제에 기반해 연구를 진행했다는 점을 고려할 때, 설계한 알고리즘도 이에 한정되어 있다. 특히 조향 함수의 경우 해당 차량의 규격과 카메라 위치를 고려하여 급격한 커브가 발생할 때로 특화되어 있으므로, 일반적인 상황을 고려하여 차선 이탈을 방지하는 알고리즘을 정교하게 설계할 필요가 있다. ROI도 마찬가지로 차량의 카메라 위치가 변함에 따라 본 연구에서 거친 프로세스를 다시 수행하여 새롭게 구현해야 하는데, 하드웨어를 교체할 때마다 시험 주행을 진행하고, 자료를 수집하여 새로운 합성 이

미지를 제시해야 한다는 번거로움이 한계점으로 남아 있다. 더불어 1.2에서 언급한 바와 같이 가우시안 필터의 크기를 3x3 이상으로 확대할 필요가 없었는데, 현실 환경에 기반한 이미지 프레임을 필터링한다면 하이퍼 파라미터를 활용할 방안을 모색해야 할 것이다.

또한, 연산 속도의 향상에 영향을 미칠 만한 요인들을 일부 발견할 수 있었고, 대표적으로 편광 필터가 있다. 편광 필터는 표면의 반사광을 제거하는 필터로서 반사율이 높은 사물의 본질을 그대로 입력받을 수 있도록 설계되어 있다. 광선은 직각을 이루는 선 위에서 사방으로 진동하게 되는데, 편광 필터는 이 중 특정한 한 방향에서 오는 빛의 파장만을 받아들여 불필요한 빛이 렌즈 내부로 입사하는 것을 방지한다. 덕분에 물이나 유리 표면에서의 반사와 플레어를 획기적으로 줄일 수 있으며, 받아들이고자 하는 빛은 더욱 선명하게 입력받을 수 있다. 이를 활용하면 반사광이 상대적으로 많이 발생하는 주행 도로 위에서의 차선 인식률을 개선하고, 허프 변환의 연산 속도를 향상하는 하드웨어 측면에서의 개선점으로 작용할 것이다.

허프 변환의 경우 각도 스텝 증가 값의 최적화를 위해 두 가지 변수, 즉 허프 변환의 정확도와 연산 속도를 고려한 계산 실험을 수행하였다. 이때 스텝 증가 값은 n이 1, 2, 3, 8, 10, 15, 21일 때 유의미한 계산 결과가 도출되었으며 최적화 값의 후보가 됨을 알 수 있었다. 다만 정확한 n값에 도달하지는 못했는데, 개별 상황에서의 주행 환경이 n의 최적값에 매우 큰 영향을 미치기 때문이다. 이는 실험을 통해 도출한 n의 값일지언정 실제 주행 환경에 적용했을 때 차선의 두께를 비롯한 여러 변수를 고려해야 함을 의미한다. 이와 더불어 확률적 허프 변환 자체에서 사용한 매개변수 중 선분의 최소 및 최대 길이 또한 조정할 필요가 있으며, 개별 실험에서의 보정 값을 구하는 후속 연구를 제안할 수 있다.

참고문헌

[1] https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html - OpenCV 공식 문서

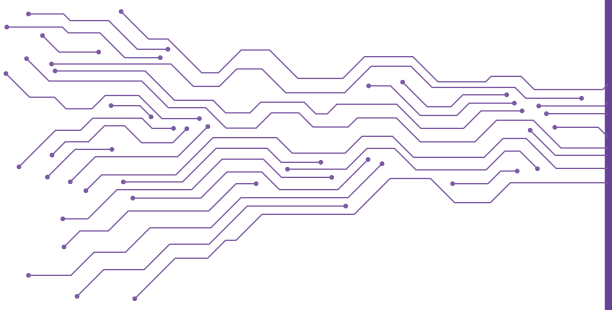
[2] 조재현, 엘뎅토야, 장영민, 조상복.(2013).보간법과 다중 ROI가 적용된 Hough Transform을 통한 향상된 차선 인식 알고리즘.대한전자공학회 학술대회,(),561-564.

06

자율 주행 교육용 모형 자동차의 주행능력 개선

하나고등학교

신상우, 신우현, 장이환



자율 주행 교육용 모형 자동차의 주행능력 개선

Improvement of Driving Capacity of Self-Driving Educational Model Cars

신상우, 신우현, 장이환
Sangwoo Shin, Woohyun Shin, Yihwan Jang

요약

4차 산업 혁명 시대에서 각광 받고 있는 자율주행차는 IT와 자동차가 접목된 것으로, 이름 그대로 운전자 없이 차량 스스로 판단하여 주행하는 자동차이다. 본 논문에서는 자율 주행 자동차에 대해 분석하고 개선 방안을 제시할 예정이다. 본 팀이 자율 주행 자동차(이하 자주차)에 대해 분석하면서 개선한 부분은 총 4가지이다. 첫 번째로, 일반적인 자동차들과 비슷하게 이륜 조향으로 작동하는 자주차의 조향을 사륜 조향으로 전환하여 방향 조절을 더욱 원활하게 수행하도록 할 것이다. 두 번째로, 구동부에서는 모터로부터 동력을 전달해주는 기어를 제롤 베벨기어로 변경하여 동력 전달의 효율성을 높일 것이다. 세 번째로, 구성부에서는 선행연구에서 베어링을 통해 각 부품간의 접합력을 늘린 것처럼 진동 감쇠 가황 고무를 활용하여 자주차 작동 시 발생하는 진동을 줄일 것이다. 마지막으로 인식부에서는 기존에 사용하던 카메라를 이용한 인식 방법에 적외선 센서를 추가하여 실제 주행 시 도로 인식률을 높이며 센서 오작동에 대한 대비책을 세울 것이다.

Keywords: 자율주행자동차, 사륜 조향, 제롤 베벨기어, 적외선 센서

I. 서 론

지금까지 교육용 자율주행자동차 모델은 4세대까지 개발되며 많은 수정과 발전을 거쳤다. 그러나 본 자주차 모델은 여전히 문제점과 개선할 점이 남아있다. 본 팀은 모델 제작 및 구동 과정에서 조향부, 구동부, 구성부, 인식부 등에서 개선 방안을 고안하였다. 조향부의 경우, 조향각이 자유롭지 못하며, 구동부의 기어는 과사용 시 마모 등의 문제가 발생하였다. 또한, 차량의 진동으로 인해 부품이 떨어지는 등의 문제가 발생했으며, 인식부는 인식 방법의 수가 하나여서 문제 상황시의 대처 유연성이 적었다. 본 팀은 이번 연구에서 이들을 각각 사륜 조향, 제롤 베벨기어, 진동 감쇠 가황 고무, 라인 트레이싱 기술의 사용 및 적용으로 해결할 것을 제안한다.

II. 조향부

1. 사륜조향을 통한 조향 용이성 증대

현재 대부분의 자동차는 이륜조향, 그중에서도 전륜조향 방식을 채택한다. 이륜 조향은 앞바퀴만을 이용해서 차량의 방향을 조절하는 것으로 운전자가 핸들을 이용해 쉽게 운전할 수 있다는 장점이 있지만 차선 이동이나 회전 시의 반경 등에서 여러 단점을 지닌다. 따라서 본 팀은 전륜뿐 아니라 후륜에도 조향 기능을 포함한 사륜 조향 방식을 자율 주행 자동차에 적용할 것을 제안한다. 본래는 후륜 조향부의 형태를 전륜과 동일하게 할 계획이었다. 조향막대, 타이로드, 바퀴로 이루어진 아커만 형식의 조향부가 그것이다. 그러나 관련 탐구를 하며 후륜은 전륜과 달리 모터가 기어를 이용해 바퀴에 동력을 전달해주기 때문에 후륜조향을 하려면 모터 또한 같이 회전해야 할 필요성을 느꼈다. 이 경우, 후륜의 조향을 위한 서브모터와, 스텝모터를 움직이기 위한 서브모터 두 개의 추가 설치가 필요하기 때문에 무게가 크게 늘어나게 된다는 단점이 있다. 이는 곧 차량 안정성 저하로 이어질 것이다. 따라서 본 팀은 상황에 따라 왼쪽의 좌측의 후륜과 우측의 후륜의 회전 속도 및 방향을 다르게 하여 조향을 하는 방식을 채택하기로 하였다. 이를 위해서는 양쪽의 모터를 각각 설치하여 따로 제어를 하는 형태를 지닐 것이다. 모터는 전과 같이 모두 스텝 모터를 이용할 것이다. 회전하고픈 방향의 바퀴를 느리게, 또는 앞쪽으로 회전시키고, 반대쪽 바퀴를 빠르게, 또는 뒷쪽으로 회전시키는 방식이다. 이 경우 추가적인 모터의 수가 2개에서 1개로 적어 비교적 차량의 무게를 크게 늘리지 않고 조향이 가능하다는 장점이 있다.

사륜조향에는 전륜과 후륜이 같은 방향으로 회전하는 동위상 조향과 서로 다른 방향으로 회전하는 역위상 조향이 있다(그림 1).

그림 1. 사륜 조향의 두 가지 종류

신상우 has_21098@hana.hs.kr
신우현 paciharry@naver.com
장이환 jehwan1031@naver.com

동위상 조향의 경우, 고속 운전을 할 때, 부드러운 차선 이동을 가능하게 하며 요잉(yawing)을 줄인다. 요잉이란 물체의 수직 방향을 중심으로 물체가 회전 진동하는 현상으로 이를 경감시킨다면 사고의 위험도 감소하고 차체 손상의 위험 또한 줄어든다. 역위상 조향의 경우, 저속으로 운전할 때 회전 반경을 크게 줄여주는 장점이 있다. 오늘날, 사륜 조향 기술이 상용화되지 않은 이유에는 여러 가지가 있지만 운전자의 조향 어려움이 가장 대표적이다. 그러나 자율주행자동차는 운전자의 조향이 필요하지 않고 인공지능이 계산 과정을 통해 주행을 하기 때문에 해당 기술을 적용하기에 적합하다. 본 팀의 차량은 저속주행을 목적으로 하기 때문에 역위상 조향을 주로 사용하여 차량이 주행선을 침범하지 않고 침범 했을 때에는 빠르게 벗어날 수 있다.

또한, 차량의 대표적인 3가지 회전운동인 롤(Roll), 피치(Pitch), 요(yaw)를 고려하여 사륜 조향을 위한 장치를 사용한다면 더욱 안정적인 주행을 할 수 있다.[2] 롤은 차량이 종 방향 축(x축), 피치는 횡 방향 축(y축), 요는 수직 방향(z축)을 중심으로 회전하는 현상으로 회전운동의 중심점을 중점으로 회전운동을 한다. 이때 이 중심점의 위치에 따라 차량의 동적 성능이 바뀌게 되기 때문에 롤과 피치의 중심점은 낮게, 요의 중심점은 후륜에 가깝도록 설계하는 것이 좋다. 따라서 차체를 낮추고 무게중심을 조금 뒤쪽에 두는 것이 적절할 것이다. 이 또한, 본 팀의 차량의 저속주행에서는 주행에 큰 무리가 없지만 실제 자율주행자동차는 큰 롤, 피치, 요를 동반하므로 이를 고려한 후 사륜조향장치를 사용한다면 더욱 사륜조향의 효과가 클 것이다.

Ⅲ. 구동부

1. 제롤 베벨기어를 통한 자율주행자동차의 효율성 향상

현재 사용하고 있는 자율 주행 자동차에서 구동부로 분류되는 부분은 크게 두 부분으로 나뉜다. 첫 번째는, 차량 앞쪽 조향부에 적용된 렉 기어이고 두 번째는 모터에서 동력부로 연결되는 부위에 적용된 동력 전달용 베벨 기어이다. 그중 본 논문에서는 동력부에 적용된 베벨 기어의 문제점과 이에 대한 개선점을 탐구해보고자 한다. 4세대 자율주행자동차가 필요요하는 베벨 기어는 2개의 축이 수직으로 만나는 교차 축 기어로 베벨 기어, 스트레이트 베벨 기어, 스파이럴 베벨 기어 등의 기어를 사용할 수 있다. 그중에서 현재 자율 주행 자동차 모형에 적용된 베벨 기어는 톱니 줄기가 피치 원뿔면에 일치하는 기어인 직선 스트레이트 베벨 기어이다. 이 기어의 물림은 서로 맞물릴 때 이의 위쪽에서 시작하여 뿌리 방향으로 물림이 진행된다. 스트레이트 베벨 기어는 베벨 기어 가운데 가장 만들기 쉽고 간단하고 제작비가 적게 들며 주로 공장 기계, 인쇄 기계, 차동기어 장치 등에 사용한다. 이렇듯 직선 베벨 기어는 제작의 용이성과 적은 비용을 통한 생산 가능이라는 장점을 지니고 있지만, 자율 주행 자동차를 이용하여 탐구를 진행하다가 Python을 이용하여 직접 코딩 후 자주차를 작동시키는 과정에서 코딩과 모터가 정상적이었음에도 자주차가 움직이지 않는 문제를 발견하였다. 본 팀은 자주차가 움직이지 않는 원인이 기어의 과사용으로 인한 문제로 판단하였다. 이에 기존 베벨 기어에서 발전된 형태의 베벨 기어들을 찾아보았고 대체할 기어로 제롤 베벨 기어(그림 3)를 선정하였다. 제롤 베벨 기어는 직선 베벨 기어와 달리 나선형의 치형을 가진 베벨 기어로, 치형 물림율이 높아 직선 베벨 기어보다 효율, 강도, 진동 및 소음 면에서 우수하다. 일본의 기어회사인 KHK 사의 성능평가표(그림 2)를 살펴보면, 특정 회사의 제품인 것을 감안하더라도 제롤 베벨 기어의 호환성, 내구성 등에서 더 좋은 평가를 받았음을 알 수 있다[3].본 팀에서는 이런 제롤 베벨 기어의 장점을 활용하여 자주차에서 기어로 인해 발생하는 구동 문제를 해결하고자 한다. 제롤 베벨 기어는 직선 베벨 기어보다 기어 이빨 물림이 더욱 원활하여, 자주차 기어에 마모가 일어났더라도 기어가 헛도는 문제를 해결할 수 있다. 자주차는 Python을 통한 코딩을 이용해 정교한 방향 전환을 요구받는다. 자율 주행 자동차가 움직일 때 가장 중요한 역할을 하는 동력 전달에서 베벨 기어는 스텝모터가 발생시킨 동력을 조향부로 전달한다. 그렇기에 본 팀은 제롤 베벨 기어로의 교체를 통해 오차 없이 정확하게 작동하는 모델을 구상하였다.

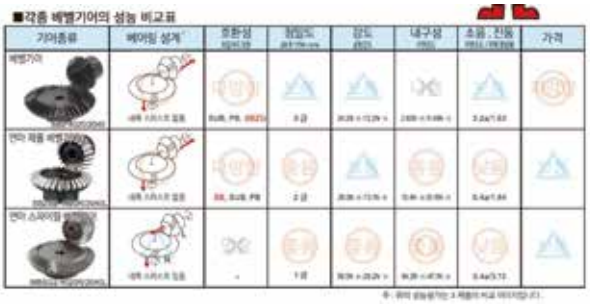


그림 2. KHK사의 베벨기어 비교표(자사의 베벨기어의 성능을 비교함)

그림 3. 제롤 베벨기어

본 논문에서는 기존의 기어를 단순 교체하는 것에서 나아가 교체한 제롤 베벨 기어의 백래시를 조정하여 앞서 언급했던 자주차의 효율적 움직임을 이룰 것이다. 대부분의 기어는 기본적으로 백래시를 가지고 있는데, 백래시란 한 쌍의 기어를 맞물렸을 때 기어이와 이 사이에 생기는 틈새를 의미한다. 백래시는 설계과정에서 의도적으로 만들어질 수도 있으나 의도하지 않게 발생하는 경우도 있다. 현재 사용되는 기어에서 백래시는 너무 크지도 작지도 않은 적당함을 유지해야 한다. 이때 베벨 기어의 경우 '축방향 틈새'라는 백래시가 발생하는데, 베벨 기어의 물림 측 치면과 반대 물림 측 치면이 접촉하도록 하였을 때 정해진 조립 거리로부터의 감소량(이동량)을 의미한다. 본 팀은 제롤 베벨 기어

적용 시 베벨 기어의 조립 거리를 작게 함으로써 축 방향의 틈새를 조정하고 이를 통해 백래시를 작게 할 것이다.

Ⅳ. 구성부

1. 진동 감쇠 가황 고무를 통한 차량 안정성 증대

본 팀이 연구 중에 찾은 또 하나의 문제점은 나사나 볼트 등의 부품들이 주행을 거듭할수록 조금씩 풀려 정기적으로 다시 조여야 한다는 것이었다. 이에 본 팀은 볼트나 너트의 헐거워짐과 부품의 마모 문제를 해결하기 위해 자율주행자동차에서 발생하는 진동을 감쇠시키는 방안을 생각했다. 진동 감쇠 가황 고무(그림 6)를 차량의 프레임 부분에 방진고무 마운트를 삽입하는 것으로써 실행하려고 한다. 조향부와 같이 회전이 필요로 하는 곳은 고무가 회전운동에 영향을 끼칠 수 있기 때문에 마운트를 삽입하지 않는다. 본 팀은 진동 감쇠 가황 고무를 그림 5와 같이 자주차에 적용하여 다음과 같은 효과를 얻을 것이다. 우선, 고무, 그중에서도 가황 고무(첨가제를 넣어 가교결합을 형성한 고무)는 대개 탄성 변형이 크고 탄성율은 작으며 형상 자유도가 높다. (그림 4) 따라서 상하, 좌우, 전후의 3가지 스프링 정수를 가지고 있으며 이를 자유롭게 설정할 수 있다. 따라서 상황에 따른 적용을 유동적으로 가져갈 수 있다. 또한, 비압축성을 지니기 때문에 응력과 변형 간 시간 지연이 있어 비선형적 성질을 지녀 효과적이다. 금속에 비해 감쇄가 1000배 이상 크며 재료 배합을 통해 효율성을 높일 수 있다. 구상한 차량은 기름을 사용하지 않고 발열이 크지 않지만 실제 자율주행자동차는 구동을 위해 기름이 사용될 가능성이 있고 발열도 상당하기 때문에 이에 대한 내구성을 갖춘 고무를 선정하는 것이 중요할 것이다.

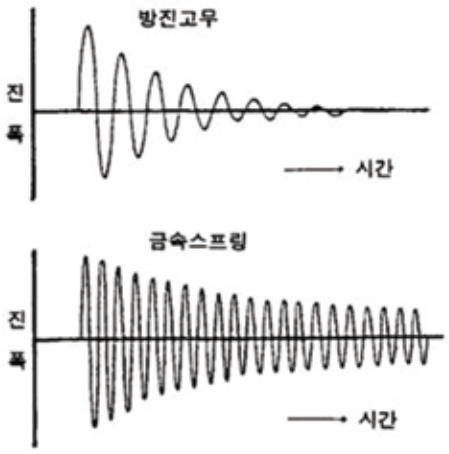


그림 4. 금속스프링와 방진고무의 진동 감쇠성 비교

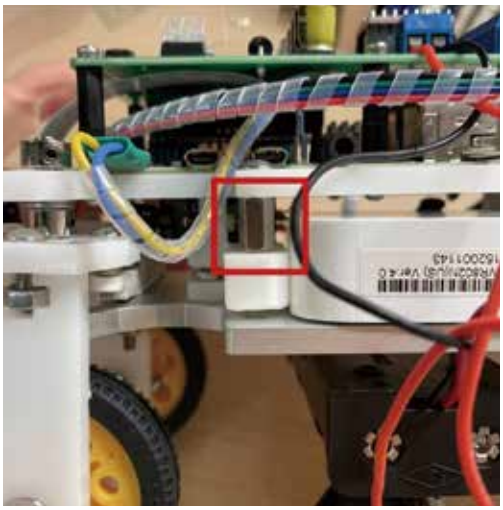


그림 5. 진동 감쇠 가황 고무 적용 위치 예시

그림 6. 가황 고무 구조체

Ⅳ. 인식부

2. 라인 트레이싱 기술을 이용한 차선 인식 개선

기존의 자율주행자동차 모델은 카메라를 활용해 인식한 이미지만을 사용하여 차선을 파악하였다. 하지만 전방의 라이더 센서와 카메라에만 의존할 경우, 카메라에 문제가 생겼을 때 자율주행에 큰 문제가 발생할 것이고 실제 탐구 중 카메라의 이미지가 전송되지 못하는 순간 때문에 자율 주행에 어려움을 겪기도 했다. 외부에 있는 실제 도로에서의 주행을 고려하면, 안개나 강우 등 카메라가 주변을 인식할 때 방해받을 수 있는 요소가 많고 이러한 요인으로 인한 잘못된 인식은 실제 상황에서 사고로 이어질 수 있다. 따라서 본 팀은 카메라 이미지 기반 자율주행 시스템에 추가로 적외선 센서를 도입한 '라인 트레이싱 기술'의 도입을 제안한다.

그림 7. 적외선 센서

라인 트레이싱 기술을 활용하기 위해서는 자율주행자동차 모형이 바닥의 색깔을 인식해야 한다. 이를 위해 적외선 센서를 활용하여 적외선을 바닥으로 쏘아 반사되는 빛의 양으로 색깔을 인식할 것이다. 적외선 센서는 그림 7과 같이 발광부와 수광부로 이루어져 있다. 발광부에서 적외선을 쏘게 되면 물체에 반사되어서 수광부로 들어오는 반면, 검은색 부분은 빛을 흡수하여 수광부에 적외선이 도달하지 않는다. 이 원리를 이용하면 검은색 바닥(도로)에서 흰색 선(차선)을 인식할 수 있게 된다. 본 팀에서는 그림 8과 같이 자외선 센서를 자주차에 설치하여 라인 트레이싱 기술을 활용할 것이다. 라인 트레이싱 기술을 활용한 자주차 적외선 센서 코드의 작동 원리와 기본 구조도는 다음과 같다. 간단하게 원리를 설명하자면, 왼쪽 센서에 차선이 감지되면 차를 오른쪽 방향으로 주행하고 오른쪽 센서가 차선을 감지하면 왼쪽으로 주행하여 모델의 차선 이탈을 막아주며 차선을 밟는 것을 최소화하는 것이다.

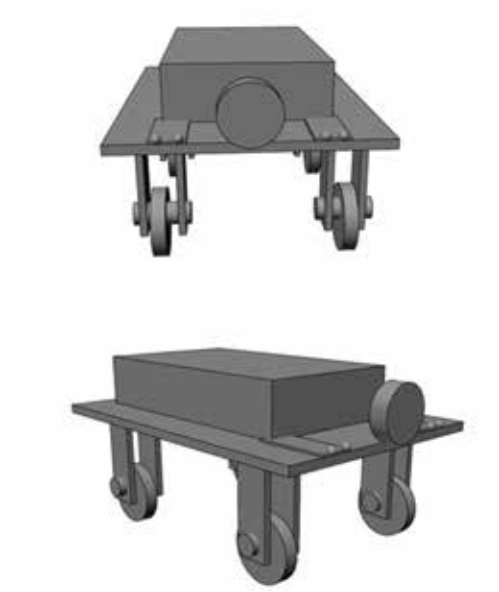


그림 8. 적외선 센서를 자주차에 적용한 모습을 3D 모델링을 통해 나타낸 모습

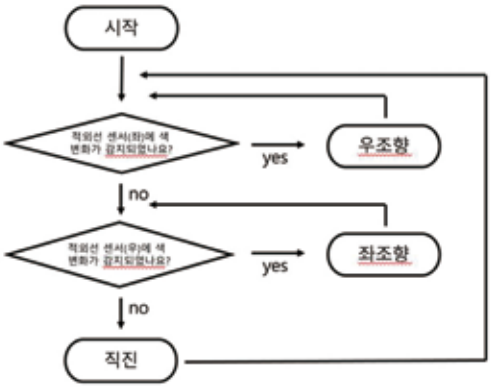


그림 9. 라인트레이싱 기술을 적용한 자주차의 작동 원리

```
void loop(){
 int aSensor = digitalRead(2);
 int bSensor = digitalRead(3);
 int Trig = LOW; //감지 되었을때가 LOW 이고 이를 Trig 함수로 치환
 if(aSensor == Trig && bSensor == Trig){
  //양쪽 다 감지 되었을때
  analogWrite(5,0); // 모터 직진
  analogWrite(5,0); // 모터 직진
 }
 else if(aSensor != Trig && aSensor != Trig){
  //양쪽 다 감지 되지 않았을 때
  analogWrite(5,255); // 모터 직진
  analogWrite(6,255); // 모터 직진
 }
 else if(aSensor != Trig && bSensor == Trig){
    // 오른이 감지 되었을 때 좌회전을 위해 왼쪽 바퀴의 속도를 줄이거나 멈춘다.
  analogWrite(5,0);
  analogWrite(6,255);
 }
 else if(aSensor == Trig && bSensor == Trig){

  // 왼쪽이 감지 되었을 때 우회전을 위해 오른쪽 바퀴의 속도를 줄이거나 멈춘다.
  analogWrite(5,255);
  analogWrite(6,0);
 }
}
```

Ⅳ. 결론 및 제언

1. 결론

본 연구에서 새롭게 제안한 자율 주행 교육용 모형 자동차는 더욱 정밀한 조향장치 및 주행에서 마모 및 고장 감소, 추가된 인식 기능을 제공한다. 제롤 베벨 기어로 바꾸면서 안정성과 효율성이 증가하며, 진동 감쇠 가황 고무를 추가함으로써 진동으로 발생하는 차량의 고장 및 마모량을 줄였다. 또한, 전방부에 적외선 센서를 2개 설치하여 카메라 인식에 오류가 생기거나 어둡거나 흐린 이유로 카메라 인식이 어려울 때 차선 인식에 도움을 받을 수 있도록 개선하였다. 이렇게 개선된 차량을 활용하여 자율 주행 교육이 더욱 효과적으로 이루어지리라 기대한다.

2. 제언

사륜 조향을 위해서는 뒷바퀴에 스텝 모터 2개를 설치해야 한다. 즉, 원래에서 추가로 1개의 모터를 더 설치해야하는 것이다. 따라서 아무리 경량 모터를 사용한다고 하더라도 차량의 전체적

인 무게 증가는 막을 수 없으며 이는 원활한 구동에 방해 요소가 될 수 있다. 이를 해결하기 위해서 한 가지 동력원에서 양쪽 바퀴에 모두 동력을 전달하는 동력전달 방식을 고안해 사용할 수 있을 것이나 실제 자율주행자동차에서는 궁극적으로 전륜과 후륜 각각의 동력원을 사용할 것이기 때문에 이보다는 배터리를 비롯한 다른 부품들을 최적화하여 성능을 유지 및 향상시키는 방안이 적절할 것이라 전망한다. 다음으로 제롤 베벨 기어를 통해 자율주행자동차를 개선할 시, 기존 자주차에서의 확실한 개선점이 발견되지 않을 수 있다. 이를 해결하기 위해서는 제롤 베벨기어를 통해 연계할 수 있는 개선점을 찾아야 할 것이다.

참고문헌

[1] https://terms.naver.com/entry.naver?docId=1654985&cid=42330&categoryId=42330(2022.05.19.)

[2] 유승범, 백훈열, 최광수.(2017).후륜조향 시스템 “RWS”적용을 통한 차량 조종 안정성 개선.한국자동차공학회 춘계학술대회,(),485-490.

[3] https://khkgears.net/korea/bevel-gears.html(2022.06.12.)

[4] 모재현, 장재훈. 통합 사륜 조향장치의 차량 핸들링 특성에 대한 실험적 고찰. 한국자동차공학회. 2013.11.

[5] 조성현, 최원택, 정대종, 김종갑. 유압식 사륜조향 시스템의 개발. 한국자동차공학회. 1994.11.

[6] 송정훈. 능동전륜조향장치를 채택한 사륜조향차량의 횡방향 안정성 강화에 대한 연구. 한국자동차공학회. 2012.03.

[7] 봉우종, 이상호, 이언구, 한창수. 기존모델 추종제어를 이용한 독립 후륜조향 차량의 조향 특성 해석. 2000.3.

[8] 유승범, 백훈열, 최광수. 후륜조향 시스템 "RWS" 적용을 통한 차량 조종 안정성 개선. 2017.05.

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

[10] P. Eswaran, A. V. M. Manikandan, S. Godha, "Prototype of an underground multi-storied automated car parking system", in Proc. IEEE International Conference ON Emerging Trends in Computing Communication and Nanotechnology (ICECCN), 674-677, 2013.

[11] B. T. Nugraha, S.-F. Su, "Towards self-driving car using convolutional neural network and road lane detector", in Proc. 2017 2nd International Conference on Automation Cognitive Science Optics Micro Electro Mechanical System and Information Technology (ICACOMIT), 65-69, 2017.

[12] P. Gurjashan, A. Mohammad and G. Pritha, “Design and Implementation of Autonomous Car using Raspberry Pi”, International Journal of Computer Applications, 113, 22-29.

[13] K. Satyanarayana. B. Tapasvi. P. KanakaRaju. and G. RameshBabu, “Based on machine learningAutonomous car using raspberry-pi.”, International Journal of Engineering Research and Application (IJERA), 7(12), 76-82, Dec 2017.

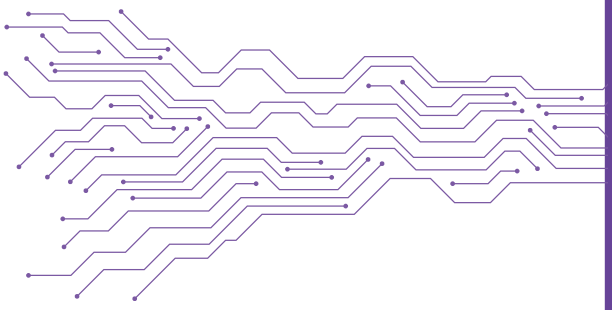
[14] E. Bertolazzi, F. Biral, P. Bosetti, M. De Cecco, R. Oboe, F. Zendri, "Development of a reduced size unmanned car", in Proc. 2008 10th IEEE International Workshop on Advanced Motion Control, 763-770, 2008.

[15] S. Verghese, "Self-driving cars and lidar", in Proc. Lasers and Electro-Optics (CLEO) 2017 Conference on, 2017.

[16] J. Y. Wong, Theory of Ground Vehicles., John Wiley & Sons, 2008.

07 도로 요소 오염 상황에서의 인식 개선

인천 동산고등학교
유태원, 임도현, 정태연, 황규성



도로 요소 오염 상황에서의 인식 개선

A Study on the Improvement of Recognition for Road Element Contamination

유태원*, 임도현**, 정태연***, 황규성****
Tae-Won Yoo, Do-hyeon Lim, Tae-yeon Jeong, Gyu-sung Hwang

요약

기존의 자율주행 자동차 알고리즘에서는 도로 요소가 오염될 경우, 차선을 인식하는 능력이 떨어지는 경향을 보였다. 본 연구에서는 직선 상황에서 차선이 훼손된 경우, 도로에 요소들로 인해 인식이 방해받는 경우로 문제 상황을 설정하고 각각 기울기를 이용하여 훼손된 차선을 원래 차선을 이용하여 예측하여 인식하는 방법, 장애물을 지나는 동안 조향 값을 고정하는 방법을 통하여 도로 요소 오염 상황에서의 인식 개선방안을 제시한다.

Keywords: 자율주행 자동차, 오염, 인식, 기울기, 분산

I. 서 론

자율주행 자동차는 알고리즘과 LiDAR, Rader와 같은 센서를 통해 운전자를 보조하여 안정성과 편의성을 향상시키는 시스템을 탑재하거나 탑승자의 조작 없이 자동차 스스로 운행이 가능한 자동차를 말한다.

이러한 자율주행 자동차는 자신의 위치와 주변의 상황을 통해 도로를 인식하고 주행한다. 그러므로 센서를 통해 도로를 인식하는 것은 자율주행 자동차를 주행시키는 것에 매우 중요한 요소이다. 그러나 사람이 직접 자동차를 운행하는 경우 사람이 도로의 오염과 같은 상황을 판단하여 안정적으로 주행을 할 수 있는 반면에 자율주행 자동차의 주행에 경우에는 LiDAR와 Rader와 같은 센서를 통해 정보를 받아들이고, 이 정보를 기반으로 차선과 기타 도로의 요소를 파악하여 알고리즘을 구동하는데, 실제 운전 상황에서는 도로 오염 요소로 인한 인식 장애와 같은 다양한 현장 변수들로 인해, LiDAR와 Rader와 같은 센서로 들어온 정보를 인식하는 것에 문제가 생길 수 있다.

이러한 오류는 자율주행 자동차의 구동에 있어서 차선을 이탈하거나 다른 자동차와 충돌하는 것과 같은 큰 문제들을 야기할 수 있으므로, 본 연구에서는 도로 요소의 오염 상황에서 생기는 문제점을 개선하고 보완하여 제시한다.

II. 기존 알고리즘

기존 주행 알고리즘은 가로 (rows), 세로 (cols)로 선을 그려 차선과 맞닿는 부분의 위치값을 받아와 양쪽 차선을 인식한다. [그림 1]과 같은 직선 상황에서, [그림 3]과 같이 차선과 맞닿는 부분의 위칫값 차이를 통해 직선 상황이라고 인식하고 양쪽 차선의 위치 차이를 이용해 차의 주향을 설정해서 도로의 중앙으로 주행하도록 알고리즘을 구성하였다.

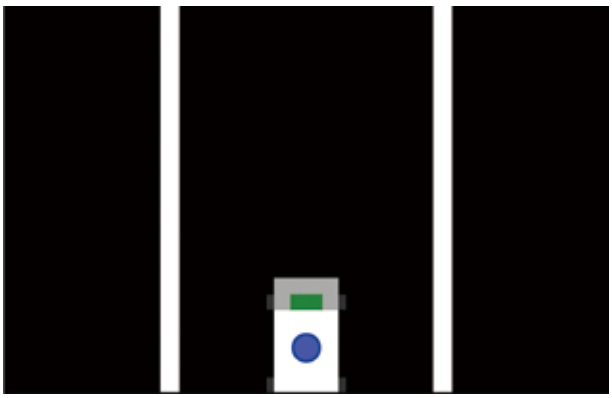


그림 1. 정상 코스

유태원 (동산고등학교, black-shark15@naver.com)
임도현 (동산고등학교, imdo0602@naver.com)
정태연 (동산고등학교, jtaeyeon05@gmail.com)
황규성 (동산고등학교, usdh0430@naver.com)

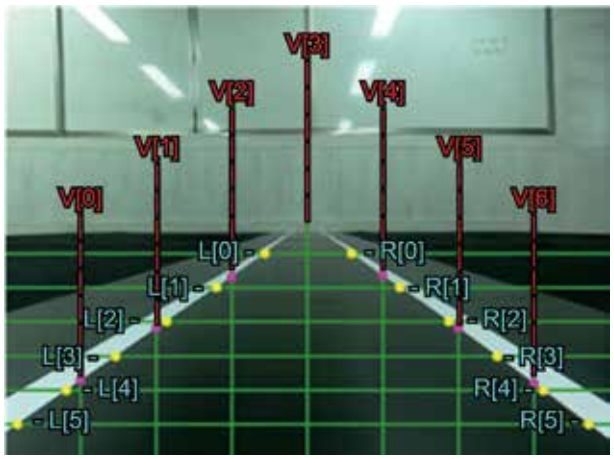


그림 2. 직선 도로 차선 인식


그림 3. 기존 주행 코드

자율주행 자동차의 차선 인식 방법은 차선과 세로로 그은 선의 교점을 V[x], 차선과 가로로 그은 선의 교점을 화면의 중앙을 기준으로 왼쪽은 L[x], 오른쪽은 R[x]의 값으로 가져온다. 이때 V[x]의 최댓값은 255, L[x]의 최댓값은 320, R[x]의 최댓값은 321이다. 이를 통해 V[x], L[x], R[x]가 각각 최댓값인지 아닌지를 판단하여 값이 정상 범위 내에 있는지 확인한다.

III. 도로가 훼손된 상황

1. 문제 상황

실제 주행 상황에서 차선의 페인트가 닳거나, 차선이 오염되어서 차선을 인식하기 힘들어지고, 이로 인해 가져오는 차선의 위치 값이 기존의 주행해야 하는 차선의 값과 다르게 가져와서 주행 알고리즘에 문제가 발생한다. 우리는 이러한 상황을 [그림 4]와 같이 재현하여, 3.2에서 서술할 알고리즘을 통해 개선하였다.

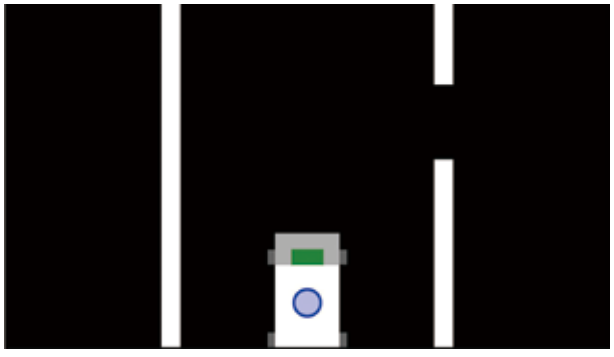


그림 4. 훼손된 도로 코스

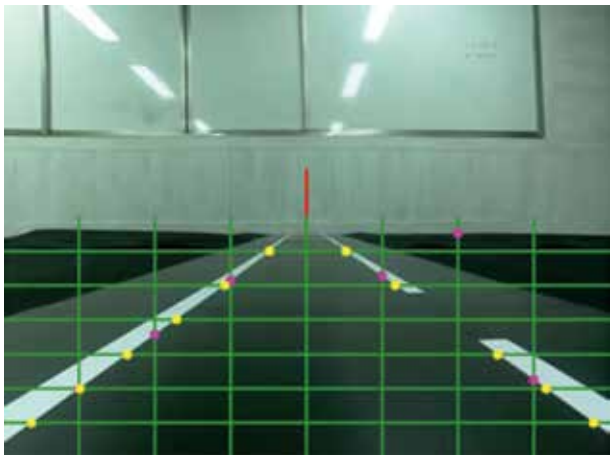


그림 5. 훼손된 차선 인식

[그림 4]와 같이 트랙을 놓고 주행한 결과 [그림 5]와 같이 도로 상황을 인식하였다. [그림 5]와 같이 상황을 인식하면 일정 차선의 값(R[2])을 정상적으로 받아오지 못해, 즉 R[2] 값을 321로 받아와 기존의 차선이 훼손되지 않았던 정상적인 직선 도로와 같은 방식으로 인식할 수 없었다.

2. 알고리즘 개선방안

우리는 정상적인 차선인지 판단하기 위해서 차선의 기울기를 측정했는데, 기울기는 y 값 / x 값이고, 이 상황에서는 가로선을 통해 기울기를 측정하는데, 가로선 간의 간격, 즉 y 값이 일정하므로 L[x] 또는 R[x]의 변화량 (L[i + 1] - L[i] 또는 R[i + 1] - R[i])을 기울기의 역수로 하였다.

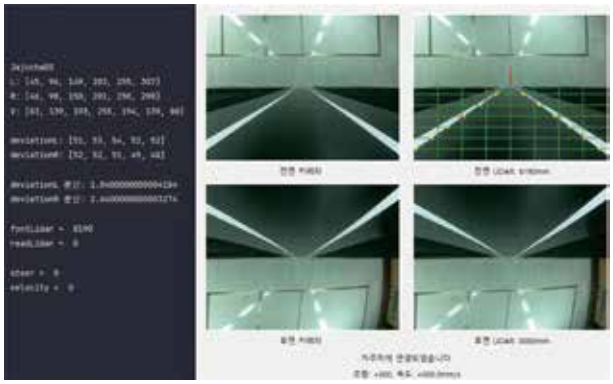


그림 6. 정상적인 도로일 때의 값

정상적인 차선인지 판단하기 위해 가로줄이 7줄인 (rows=7) 상황에서 미리 정상적인 직선코스의 도로를 주행하면서 도로의 기울기의 역수를 측정했고 ([그림 6]의 deviationL과 deviationR), 이 값이 50 정도로 거의 일정함을 알아내었다. 그리고 이를 self.defaultDeviation에 상수로 저장해서 정상적인 주행 도로를 판단하는 기준으로 사용하였다.

그림 7. 현재 코스에서의 도로 기울기 측정하는 코드

그리고 현재 코스에서 도로의 기울기의 역수를 측정하여 왼쪽 기울기의 역수 값은 deviationL이라는 배열에 저장하였고, 오른쪽 기울기의 역수 값은 deviationR이라는 배열에 저장하였다. 이때 카메라와 가까운 차선은 인식되지 않을 수도 있으므로, 마지막 기울기의 역수 값이 0이거나, 마지막 가로줄 (L[-1] 또는 R[-1])에 정상적인 값이 인식되지 않은 상황, 즉 최댓값이 인식된 상황에서는 각각의 배열에서 마지막 기울기의 역수 값을 제거하였다.

그림 8. 분산 구하는 코드 (getDispersion(array): Float)

그림 9. 차선에 문제가 있는지 판단하는 코드

한쪽 차선이 직선 도로이고, 다른 차선의 기울기의 역수의 분산이 일정 값보다 클 때 차선에 문제가 있다고 판단하였다. 이때 분산은 [그림 8]의 getDispersion(array)를 이용하여 구하였다. ([그림 9]의 코드에서는 왼쪽 차선이 직선 도로이고, 오른쪽 차선에 훼손이 일어난 경우이다.)

그림 10. 잘못된 L[x] 또는 R[x]를 구하고 해결하는 코드

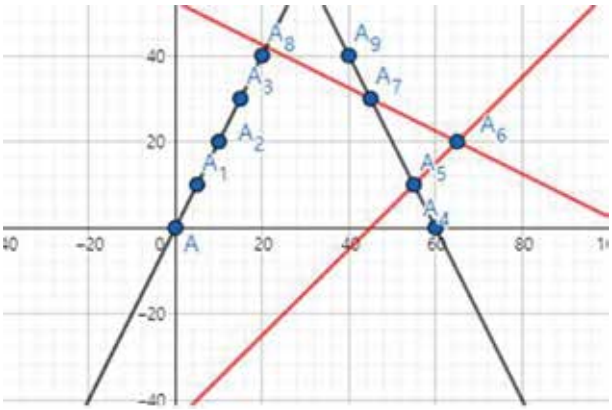


그림 11. 기울기 변화 추이

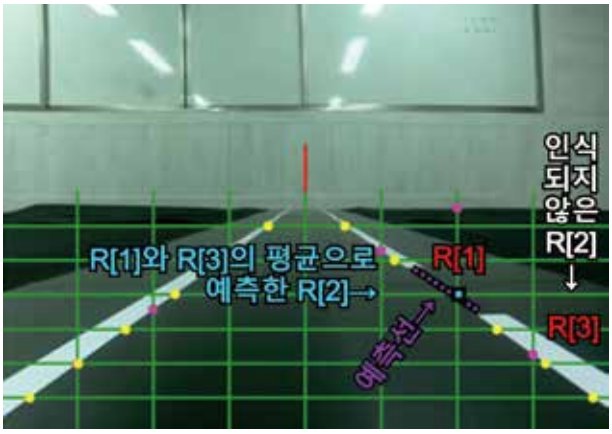


그림 12. 오류가 있는 L[x] 또는 R[x]를 예측하는 방법

이렇게 차선에 오류가 있는 것을 판단했으면, 오류가 있는 차선의 기울기의 역수가 나열되어 있는 배열 (deviationL 또는 deviationR)에서 기울기의 역수를 하나씩 가져와 기본 기울기의 역수(self.defaultDeviation)와 비교해, 차이가 크면 그에 해당하는 R[x] 또는 L[x]을 구해온다.

이렇게 문제가 있는 L[x] 또는 R[x] 값, 즉 problemIndex 값을 구했다. 이때 이 값이 가장 위에 있는 값, 즉 L[0] 또는 R[0] 라면, L[1] 또는 R[1]에서 기본 기울기의 역수 (self.defualtDeviation = 50)을 빼서 그 값을 예측하고, 가장 아래 있는 값, 즉 L[-1] 또는 R[-1] 라면, L[-2] 또는 R[-2]에서 (self.defualtDeviation = 50)을 더해서 그 값을 예측한다.

만약 problemIndex (i로 설명하겠다.)가 가장 위나 가장 아래가 아닌 중간의 값이라면, 그 윗값과 아랫값의 평균을 이용해 그 값을 예측한다. (L[i] = (L[i − 1] + L[i + 1]) / 2, R[i] = (R[i − 1] + R[i + 1]) / 2)

이 같은 방법으로 우리는 잘못된 L[x] 또는 R[x]를 찾아내고, 예측하여 정상적으로 도로를 주행할 수 있게 하였다.

IV. 도로가 오염된 상황

1. 문제 상황

실제 운전 상황에서는 도로에 납작한 물체가 덮여있거나, 색이 다르게 도색되어 있을 수도 있고, 이로 인해 가져오는 차선의 위치 값이 기존의 주행해야 하는 차선의 값과 다르게 가져와서 주행 알고리즘에 문제가 발생한다. 비닐봉지나 종이가 도로에 버려져 있거나, 맨홀 등이 차선과 겹칠 때가 이와 같은 상황이다.

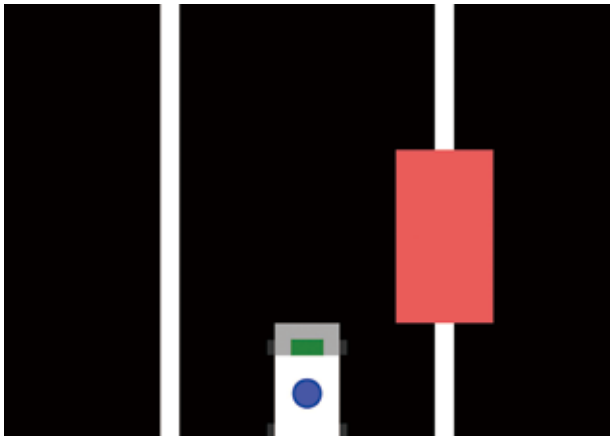


그림 13. 도로에 오염물이 덮여있는 코스

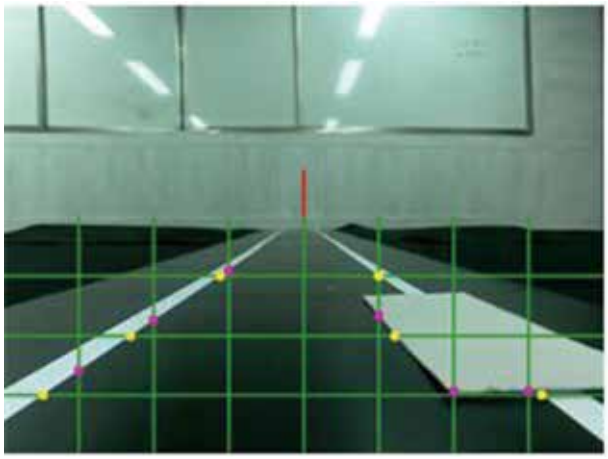


그림 14. 도로에 오염물이 덮여있는 상황의 도로 인식

기존 알고리즘에서는 [그림 13]와 같은 상황일 때, [그림 14]와 같이 인식하는데, 이렇게 되면 L[x] 또는 R[x]가 기존 값보다 작아지고 차선을 기존보다 안쪽에 있다고 측정하여, 중앙을 장애물이 있는 상황과 다르게 인식하여 한쪽으로 치우치게 측정한다. 이 상황이 도로가 훼손된 상황과 다른 점은 도로가 훼손된 상황에서는 하나의 L[x] 또는 R[x] 값만 바깥쪽으로 튀는 경우가 대부분이었으나, 현재의 상황에서는 여러 개의 L[x] 또는 R[x]가 안쪽으로 값이 튀고 있다.

2. 알고리즘 개선방안

```
self.LastL = [0, 0, 0, 0, 0, 0]
self.LastR = [0, 0, 0, 0, 0, 0]

self.isDetectingLeftObstacle = False
self.isNearLeftObstacle = False
self.isLeftObstacle = False

self.isDetectingRightObstacle = False
self.isNearRightObstacle = False
self.isRightObstacle = False
```

그림 15. 전역 변수 설정

우리는 본격적으로 알고리즘을 구현하기 앞서 총 8개의 전역변수를 추가로 설정하였다. self.LastL과 self.LastR은 각각 이전 주행 (이전 process 함수)에서의 L과 R을 저장해놓은 변수이다.

self.isDetectingLeftObstacle과 self.isDetectingRightObstacle, isNearLeftObstacle과 isNearRightObstacle, isLeftObstacle과 isRightObstacle은 각각 전방에 장애물이 감지되었는지, 장애물이 근접했는지, 장애물을 지나고 있는지를 표현하는 변수이고 기본값은 False이다.

그리고 추가로 본 연구에서는 오른쪽에 장애물이 있는 경우로 코드를 작성하고 설명할 것이지만, 왼쪽에 장애물이 있는 경우도 이와 유사하게 설명하고 구현할 수 있다.

```
if V[0] < V[1] < V[2] \
        and -10 < 2 * V[1] - (V[0] + V[2]) < 10 \
        and -7 < V[4] - V[5] < 7 and V[4] < 150 \
        and V[5] < 150:
    self.isDetectingRightObstacle = True
    print("우측 장애물 인식")
    steer = self.defaultSteer
```

그림 16. 전방 장애물 감지

한쪽 차선이 V[0]. V[1], V[2] 또는 V[6], V[5], V[4]가 순서대로 커지고, 이들의 중앙값에서 양쪽 사이드 값을 뺀 값이 0에 가까운 정도를 확인하여, 직선 도로임을 확인하고 반대쪽 차선에 V[4], V[5] 또는 V[2], V[1] 값이 150보다 작을 때, 즉 물체가 감지될 때, 전방에 장애물이 있다고 판단하고, self.isDetectingLeftObstacle 또는 self.isDetectingRightObstacle을 True로 변경한다.

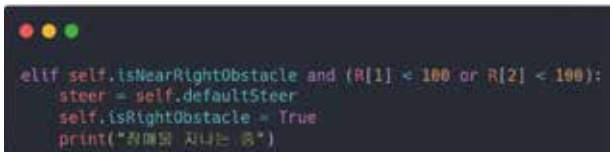
이때 장애물을 차선으로 인식하고 조향 값을 변경하지 않도록 조향 값을 기본값으로 고정해 준다.

```
if self.isDetectingRightObstacle and V[4] < 60 and V[5] < 60:
    self.isNearRightObstacle = True
    steer = self.defaultSteer
```

그림 17. 장애물 근접 감지

self.isDetectingLeftObstacle 또는 self.isDetectingRightObstacle이 True인 상황에서 V[4], V[5] 또는 V[2], V[1] 값이 150보다 작을 때, 즉 장애물이 매우 근접했다고 판단될 때, self.isNearLeftObstacle 또는 self.isNearRightObstacle을 True로 변경한다.

이때도 마찬가지로 장애물을 차선으로 인식하고 조향 값을 변경하지 않도록 조향 값을 기본값으로 고정해 준다.

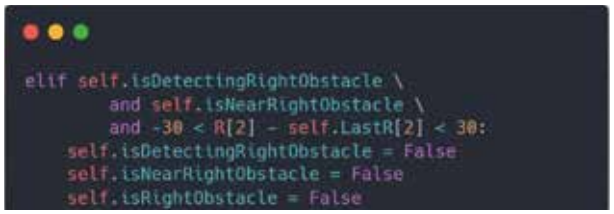


```
elif self.isNearRightObstacle and (R[1] < 100 or R[2] < 100):
    steer = self.defaultSteer
    self.isRightObstacle = True
    print("장애물 지나는 중")
```

그림 18. 장애물을 지나고 있음을 감지

self.isNearLeftObstacle 또는 self.isNearRightObstacle이 True인 상황에서 (이때 self.isDetectingLeftObstacle 또는 self.isDetectingRightObstacle도 True이다) 가로선으로 측정된 값(L[1]과 L[2] 또는 R[1]과 R[2])이 100보다 작아지면, 마침내 장애물 위를 지나고 있다고 판단하고, self.isLeftObstacle 또는 self.isRightObstacle을 True로 변경한다.

이때도 마찬가지로 장애물을 차선으로 인식하고 조향 값을 변경하지 않도록 조향 값을 기본값으로 고정해 준다.



```
elif self.isDetectingRightObstacle \
        and self.isNearRightObstacle \
        and -30 < R[2] - self.LastR[2] < 30:
    self.isDetectingRightObstacle = False
    self.isNearRightObstacle = False
    self.isRightObstacle = False
```

그림 19. 장애물을 통과했음을 감지하는 코드

self.isDetetingLeftObstacle과 self.isNearLeftObstacle 또는 self.isDetetingRightObstacle과 self.isNearRightObstacle이 모두 True인 상황에서 L[2]와 이전 process에서의 L[2], 즉 self.LastL[2] 또는 R[2]와 이전 process에서의 R[2], 즉 self.LastR[2]의 변화량이 급격하게 크다면, 장애물을 무사히 지나갔다고 판단하고 self.isDetectingLeftObstacle과 self.isNearLeftObstacle과 self.isLeftObstacle 또는 self.isDetectingRightObstacle과 self.isNearRightObstacle과 self.isRightObstacle을 모두 False로 바꾸고, 장애물을 지나기 이전 상황과 동일하게 변경한다.

```
if V[0] < V[1] < V[2] \
        and -10 < 2 * V[1] - (V[0] + V[2]) < 10 \
        and -7 < V[4] - V[5] < 7 and V[4] < 150 \
        and V[5] < 150:
    self.isDetectingRightObstacle = True
    print("우측 장애물 인식")
    steer = self.defaultSteer
if self.isDetectingRightObstacle and V[4] < 60 and V[5] < 60:
    self.isNearRightObstacle = True
    steer = self.defaultSteer
elif self.isNearRightObstacle and (R[1] < 100 or R[2] < 100):
    steer = self.defaultSteer
    self.isRightObstacle = True
    print("장애물 지나는 중")
elif self.isDetectingRightObstacle \
        and self.isNearRightObstacle \
        and -30 < R[2] - self.LastR[2] < 30:
    self.isDetectingRightObstacle = False
    self.isNearRightObstacle = False
    self.isRightObstacle = False
```

그림 20. 도로에 덮여있는 장애물 인식 및 대응 코드

이렇게 완성된 “직선 도로에 무언가 덮여있거나, 색이 다르게 칠해져 있을 경우 직선 도로로 인지하고 대응하는” 코드는 [그림 20]과 같다.

V. 결론

본 연구에서는 도로가 오염된 상황을 다루었는데, 차선 훼손 상황과 도로에 높이가 낮은 물체(자동차 주행에는 방해가 되지 않는 장애물)가 덮여있는 상황이 있었다.

차선이 훼손된 상황에서는 일단 차선이 훼손되어 있는지를 기울기의 분산을 이용해 판단하고, 차선이 훼손되었을 때 훼손된 차선이 어디 있는지를 판단하고 정상 차선의 평균값을 구하여 차선이 훼손되어 있어도 차선이 있는 것으로 예측하여 기존의 알고리즘을 적용하여 주행할 수 있게 하였다.

그리고 도로에 높이가 낮은 물체가 있는 상황에서는 차량 주행에 문제가 없을 정도의 장애물이면 조향을 고정하여 안정적으로 주행하게 하였다. 우선 세로선 값(V[x])과 가로선 값(L[x] 또는 R[x])을 통해 전방에 장애물이 있는지를 판단하고, 세로선 값(V[x])이 작아진 것을 확인함을 통해 장애물이 근접했는지 판단하고, 마지막 가로선 값(L[-1] 또는 R[-1]) 값이 급격히 작아졌음을 통해 장애물을 지남을 확인했다. 이 과정에서 조향 값(steer)는 기본 조향 값으로 고정했다. 이후 마지막 가로선 값(L[-1] 또는 R[-1]) 값이 급격히 변경된 순간을 장애물을 통과한 시점으로 판단하고 변수들을 기본 값으로 되돌렸다.

이를 통해서 우리는 도로가 오염된 상황에서도 자율주행 자동차 주행을 개선할 수 있었다.

참고문헌

[1] 이민채, 한재현, 장철훈, 선우명호, “영상 및레이저레이더 정보융합을 통한 자율 주행자동차의 주행 환경인식 및 추적 방법”, Journal of Korean Institute of Intelligent Systems, Vol. 23, NO. 1, February 2013, pp. 35-45

[2] 문석준, 최석원, 오정현, “자율 주행자동차를 위한 차선 인식을 통한 픽셀 기반 PID 제어”, 2021년도 대한전기학회 정보 및 제어 학술대회 논문집, 2021.10, pp. 363 - 364

3

포스터 논문

Journal of Youth Engineering

01 여러 상황에서의 주행 방법 연구: 기울기 알고리즘을 중심으로

A Research on Driving Methods in Different Situations: Focusing on The Gradient Algorithm

권예훈, 김유진, 유호언, 이서진 (배제고등학교)

Ⅰ. 서론

본 연구팀은 자율주행 자동차 모듈 하드웨어가 파이썬 프로그램 기반 알고리즘을 통해 여러 상황에서 차선 상황을 인식하고 장애물의 여부를 판단해서 트랙을 주행할 수 있도록 하였다.

전면 카메라에서 얻어진 실시간 이미지의 왜곡을 보정하여 형성된 FrontImage와 라이더에서 얻어진 FrontLider값을 기반으로 코드를 작성하였다. 이미지가 자율주행 자동차에 입력되면 frontImage에 가로선 3개를 그려 차선과의 교점 6개를 형성한다. 중앙선에서 교점까지의 가로거리를 차례대로 L[0], L[1], L[2], R[0], R[1], R[2]로 정의하며, L은 좌측 거리, R은 우측 거리를 의미한다.

canny 이미지에 의해 가로를 나타내는 L, R 값과 세로를 나타내는 V 값이 정해졌으므로 조향 값을 설정해주는 steer와 속도를 나타내는 velocity를 이용하여 자율주행 자동차의 경로를 설정한다. 이때 steer 값과 픽셀 사이의 유기적 연결을 위해 e 값을 도입하였다.

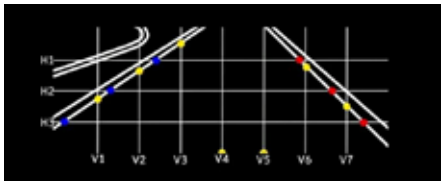


그림 1. 자율주행 자동차의 차선 인식

그러나 자율주행 자동차마다 default가 다르고, 그 때문에 경우에 따라서 상숫값을 일일이 정해야 하는 번거로운 과정이 발생하였다. 이 문제점을 해결하고자 본 연구팀에서는 '기울기 알고리즘'을 고안하였다.

Ⅱ. 주행 알고리즘

1. 직진 주행

본 모듈에서 L 값의 최댓값은 325, R 값의 최댓값은 316으로, 왼쪽으로 더 치우쳐져 있다. 때문에 L 값을 기반으로 하는 것이 더 안정적이었고, 본 연구팀은 L[2] 값을 기반으로 직진 코딩을 작성하였다.

L[2] 값과 R[2] 값이 모두 인식이 될 때(L[2] < 325, R[2] < 316), 자율주행 자동차가 왼쪽 차선에 치우쳐져 오른쪽 차선을 인식하지 못할 때(L[2] < 325, R[2] >= 316), 자율주행 자동차가 오른쪽 차선에 치우쳐져 왼쪽 차선을 인식하지 못할 때(L[2] >= 325, R[2] < 316), 3가지 경우로 나누어 알고리즘을 작성하였으며, L[2] 값과 R[2] 값에서 상숫값을 빼 e 값을 설정하게 하여 자율주행 자동차가 한쪽 차선에 많이 치우쳐 있을수록 절댓값이 더 큰 steer 값으로 조정하여 차선의 중앙으로 차량의 중심을 맞추도록 하였다.

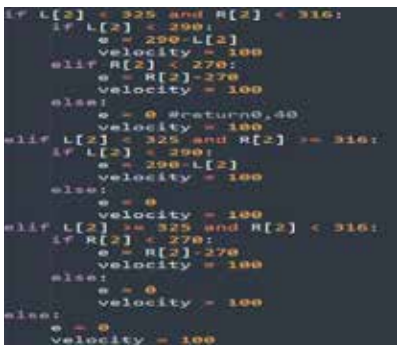


그림 2. 직선 주행 코딩

자율주행 자동차에 장착된 전면 카메라가 정중앙에 위치하지 않은 경우가 있으므로 직진에 있어서 기본적인 상숫값 조정이 필요하다.

```
steer = int(e / 3) - 11
```

그림 3. steer 값 조정 함수

L/R 값을 통해 얻은 e 값과 설정된 상수를 활용하여 조향 값을 도출한다.

2. 삼거리 (├ 자형) 주행

R[1] 값이 인식될 때 (R[1] < 314, 원활한 회전을 위해 R 값의 최댓값인 316에서 2를 뺌) 에는 직선 코딩을 따르도록 하였고, R[1] 값이 인식되지 않을 때 (R[1] >= 314) 에는 오른쪽 차선이 없는 것으로 간주하여 오른쪽으로 회전하게 한 후, 직각 곡선 코딩에 따라 주행하도록 하였다.

또한, 부드러운 회전을 위해서 V[5]-V[6]값을 사용하여 삼거리 코딩을 진입부와 중반부로 나누어 작성했다. 먼저 삼거리 진입부 (0 <= V[5]-V[6] <= 10) 에서는 e 값을 170으로 설정해 빠르게 회전하게 하였고, 삼거리 중반부 (10 <= V[5]-V[6] <= 20) 에서는 e 값을 120으로 설정해 자율주행 자동차가 회전하면서 차선을 밟는 문제를 해결하였다.

3. 삼거리 (┳ 자형) 주행

┳자형 삼거리에 가까워질수록 L[1] 값과 R[1] 값이 모두 사라지고, V[3]값이 점점 작아진다는 점을 이용해 L[1] 값이 322, R[1] 값이 313(원활한 회전을 위해 L, R 값의 최댓값에서 3을 뺌) 이상이고, V[3]값이 158 이하일 때, e 값을 120으로 설정해 자율주행 자동차가 ┳자형 삼거리 진입 시 오른쪽으로 약간 회전하게 하였고 이후 직각 곡선 코딩을 따라 주행하게 하였다.

그림 4. 삼거리 주행 코딩

4. 직각 곡선 주행

곡선 코딩의 조건을 만족시키면서 V[2]값과 V[3]값의 차이가 작을 때 (-10 <= V[2]-V[3] <= 10) 인 특수한 경우를 이용하여 직각 곡선과 일반 곡선을 구분하여 코딩을 작성하였다.

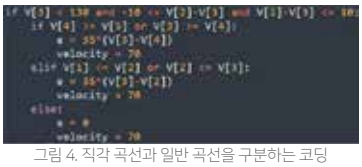


그림 4. 직각 곡선과 일반 곡선을 구분하는 코딩

V[3]값과 V[4]값의 차이에 상숫값 K를 곱해주어서 직각 곡선 상황에서 조향 값을 유동적으로 바꾸어 줄 수 있다. 또한, 일반 곡선에서 적용되는 V 값 알고리즘을 사용하여 거리가 먼 경우에는 직각 곡선을 감지하지 않도록 제작하였다.

Ⅲ. 기울기 알고리즘

1. 기울기 알고리즘

1) L, R 값을 이용하여 새로운 변수인 A, B를 정의한다.
2) A는 L[0], L[1], R[1], R[0]로 만들어지는 사다리꼴의 가운뎃점이고, B는 L[1], L[2], R[2], R[1] 사다리꼴의 가운뎃점이다. (그림 9 참고)
3) 좌회전 구간에서 L[2] 값이 잡히지 않으면 L[1]-R[2]의 중앙점을 B로 계산한다.
4) 우회전 구간에서 R[2] 값이 잡히지 않으면 R[1]-L[2]의 중앙점을 B로 계산한다.
5) A와 B를 연결한 직선을 기울기로 정의한다.
6) 기울기를 기반으로 조건문을 작성하여 주행 차선에 맞는 적절한 조향 값을 얻는다.

1.1 L[2] 값과 R[2] 값이 모두 인식되는 경우

$$m = \frac{64\left(\frac{L[1]+R[1]}{L[1]+R[1]+L[0]+R[0]} + \frac{L[1]+R[1]}{L[1]+R[1]+L[2]+R[2]}\right)}{\frac{(L[1]+R[1])\times(L[1]-R[0])}{L[0]+R[0]+L[1]+R[1]} - \frac{(L[1]+R[1])\times(L[1]+R[2])}{L[1]+R[1]+L[2]+R[2]}}$$

그림 5. L[2] 값과 R[2] 값이 인식될 때의 기울기

1.2 L[2] 값이 인식되지 않는 경우

$$m = \frac{32+64\times\frac{L[1]+R[1]}{L[0]+R[0]+L[1]+R[1]}}{\frac{(L[1]+R[2])\times(L[0]-R[0])}{L[0]+R[0]+L[1]+R[1]} - \frac{2L[1]-L[2]+R[1]}{2}}$$

그림 6. L[2] 값이 인식되지 않을 때의 기울기

1.3 R[2] 값이 인식되지 않는 경우

$$m = \frac{32+64\times\frac{L[1]+R[1]}{L[0]+R[0]+L[1]+R[1]}}{\frac{(L[1]+R[2])\times(L[0]-R[0])}{L[0]+R[0]+L[1]+R[1]} - \frac{2L[1]-L[2]+R[1]}{2}}$$

그림 7. R[2] 값이 인식되지 않을 때의 기울기

2. 기울기 알고리즘의 한계

직선 관련 코딩에서만 적용된다는 점에서 한계가 있다. 따라서 일반 곡선, 직각 곡선과 같은 상황에서는 다른 알고리즘을 적용해야 한다. 또한, 상수를 도입했을 때보다 코드가 길어지고, 계산 과정도 복잡하여 시행 속도가 느려질 수 있다는 단점이 있다.

Ⅳ. 결론

차선 중앙에 추가로 가상의 직선을 그린 다음 입력값으로 기울기를 계산하여 주행 알고리즘을 구현하였다. 기울기 알고리즘을 적용하면 비교적 단순한 조건문으로 로직 구현이 가능하다. 또한, 기울기가 차선의 방향을 의미하므로 기존에 주행해보지 않았던 새로운 형태의 차선에도 같은 알고리즘이 적용되어 자율주행이 가능하다는 점에서 큰 가치를 가진다.

02

교통 신호체계 교육용 4색 신호등 개발

Development of Four-Color Traffic Signals for Traffic Signaling System Education

김선구, 방유빈, 이호현, 최윤채 (하나고등학교)

Ⅰ. 서론

본 연구에서는 (十)자형 도로(사거리) 도로, T자형도로 2가지의 상황에서의 신호체계를 분석하고 이를 아두이노 알고리즘으로 구현하여 4색 신호등 모형을 스마트폰과 연동하여 사용자가 직접 신호체계 유형을 변경하고 이에 맞춰서 실제 도로에서의 신호체계와 동일하게 LED를 점등 시키는 알고리즘을 구현하고, 이를 제품화 (키트화) 하기 위한 방안에 대하여 연구를 진행하였다.

또한 실제로 자율주행차가 작동하는 도로에서 신호등이 쓰일 수 있도록 자율주행차가 신호를 인식할 수 있도록 고려하여 하드웨어를 설계했다.

Ⅱ. 4색 신호등 하드웨어 설계 및 제품화 방안

1. 작동방식 설계

1.1 컨트롤러

오픈소스 마이크로컨트롤러인 아두이노 UNO를 기반으로 하는 하드웨어를 사용한다. 아두이노에는 블루투스 모듈이 함께 설치되어 스마트기기와 통신한다.

1.2 스마트기기 연동

앱인벤터로 프로그래밍된 전용 애플리케이션 "HAS Traffic Light" 제작을 통해 안드로이드OS 기반의 다양한 스마트 기기로 신호등을 제어할 수 있도록 하여 편의성을 증대하였다. 앱에는 도로 유형별 신호체계 교육이라는 목적에 맞게 T자형 도로와 십(十)자형 도로(사거리)에서 작동하는 두 개의 신호 알고리즘이 내장되어 있으며, 사용자는 두 개의 버튼을 통해 알고리즘을 선택하여 실행할 수 있다. 돌발상황 발생 시 적색 신호등을 켜도록 하는 멈춤 버튼을 포함해 총 3개의 버튼으로 모든 조작이 이루어진다.

1.3. 배터리

본 제품에서는 AA 배터리를 사용한다. 아두이노의 전원소켓과 호환되는 배터리 케이스가 신호등 프레임 내부에 설치되어 사용자의 필요에 따라 배터리 교체가 가능하다.

2. 아두이노 모듈 및 회로 설계

2.1 전원공급 및 통신 모듈

아두이노 UNO 보드의 전원 소켓에 배터리 케이스가 연결되어 있고, HC-06 블루투스 모듈이 보드의, 디지털 2번~3번 핀, 전원 단자, 그리고 GND에 연결되어 있다. 배터리의 전압은 총 6볼트이므로 5V 단자를 활용한다.

2.2. 회로설계

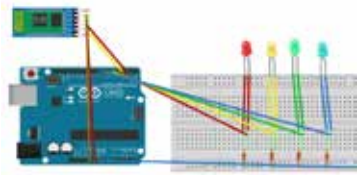


그림 1. 회로 사진

LED 4개가 나란히 연결된 구성이다. 각 LED는 브레드보드에 220Ω 저항과 직렬로 연결되어 있으며, 점프케이블과 양극을 연결하여 보드와 멀리 떨어진 곳에서도 점등 가능하게 설계하였다. 멈춤을 지시하는 적색, 멈춤을 알리는 황색, 직진을 지시하는 녹색, 좌회전을 지시하는 청색 LED의 음극이 순서대로 아두이노 보드의 디지털 핀 4~7번에 연결되어 있다. 블루투스 모듈의 경우, 컴퓨터와 아두이노의 시리얼 통신 포트인 RX와 TX핀을 통신 용도로 사용할 시에 블루투스 신호가 제대로 전달되지 않아 디지털 핀 2번과 3번을 대신 이용하였다. 브레드보드 버스 띠의 (+)극은 보드의 GND와 연결되어 있다.

3. 프레임 설계 및 제품 제작

3.1 프레임 규격 및 조립 방식 설계

4색 신호등의 구조는 받침대, 기둥, 부착대로 구분할 수 있다. 받침대는 17cm × 13cm × 5cm의 규격을 맞추어 배터리 케이스, 블루투스 모듈, 아두이노 UNO 보드, 미니 브레드보드를 모두 포함할 수 있도록 하였다. 기둥은 받침대와 부착대를 연결하는 전선이 충분히 지나갈 수 있으며 부착대를 지지할 수 있도록 하였다. 규격은 2cm × 2cm × 17cm로 자율주행 자동차가 충분히 지나갈 수 있는 공간을 확보한 높이로 설계하였다. 부착대는 2cm × 2cm × 15cm의 규격을 지키어 자율주행 자동차가 신호를 인지할 수 있는 거리에 4개의 LED와 고정판이 위치할 수 있도록 하였다.

그림 2. 받침대 내부 사진

전체 프레임을 보면 받침대의 윗면에는 기둥과 연결할 수 있는 2cm × 2cm 크기의 홈이 파져있어 이 홈에 기둥을 끼워 전체 프레임을 조립할 수 있다. 또한, 받침대의 밑면은 열고 닫음이 가능하게 하여 배터리를 쉽게 교체할 수 있도록 설계하였다.

3.2 산업공학적 고찰

교통 신호체계 교육용 4색 신호등 키트 개발 시 비용을 줄이고 효율과 품질을 높이기 위해 다음 사항들을 고려하였다. 키트 1개에는 위 규격에 맞는 아크릴 프레임, 아두이노 UNO 보드, 미니 브레드보드, 블루투스 모듈, 배터리 케이스, LED 4개(빨간색 1개, 노란색 2개, 초록색 2개), 점퍼 케이블 등이 포함된다.

신호등 프레임의 재질은 아크릴을 사용하였다. 이후 프레임 작업은 받침대, 기둥, 부착대 부분이 각각 진행되어 키트 사용자가 3개의 프레임을 조립하여 신호등을 완성할 수 있도록 함으로써 생산 속도 측면에서 효율을 높이고자 한다. 4색 신호등 제작에 필요한 전자부품 구성은 기존 아두이노 키트에 비해 훨씬 간소화되었으며 미니 브레드보드 등 부품의 크기 또한 축소되었다는 점에서 비용 측면의 이점을 얻을 수 있을 것으로 보인다.

Ⅲ. 4색 신호등 알고리즘 구현 및 앱 개발

1. 도로 유형별 4색 신호체계 분석

본 연구에서는 두가지 도로 상황별 신호 순서에 초점을 맞춰 신호 간격은 임의로 설정하고 알고리즘 제작을 하였다. (十)자형 도로(사거리)는 빨간색 (정지신호) → 초록색 (직진신호) → 노란색 → 빨간색 + 좌회전 신호 → 노란색 → 빨간색 순으로 점등이 되는 신호체계를 갖추고 있고, 오른쪽 그림에 해당하는 T자형 도로의 경우 빨간색 (정지신호) → 초록색 (직진신호) → 초록색+좌회전 신호 → 노란색 → 빨간색 에 해당하는 신호체계를 갖추고 있다.

2. 아두이노 알고리즘 개발

그림 3. 아두이노 sketch 코드 전체 사진

아두이노 sketch 프로그램에서 4색 신호체계 알고리즘을 개발했다. 앱 인벤터로 만든 앱과 연동하기 위해 블루투스 통신을 사용하여 사용자가 앱에서 도로 유형을 선택하는 것에 따라 블루투스 통신으로 받은 message 값을 다르게 하여 해당 message (숫자)를 받았을 때 그에 따라 신호체계 사이클을 작동시키는 형태로 알고리즘을 구현했다. 앱에서는 'Mode1', 'Mode2', 'STOP' 버튼 3가지가 있는데 각각의 버튼을 눌렀을 때 10자형 도로에서의 신호체계, T자형 도로에서의 신호체계 알고리즘에 따라 신호등의 LED가 일정한 시간간격 (패턴) 에 맞게 점등되고 이 체계가 STOP 버튼을 누르기 전까지 무한 반복되는 형태로 코드를 구성했다.

3. 앱인벤터 기반 스마트폰 어플 개발

그림 4. 앱 화면 (왼쪽) / 앱인벤터 작동 코드 (오른쪽)

아두이노와 스마트폰이 블루투스를 이용한 통신 방식으로 작동하는 방법을 택했기에 호환성을 높이고자 '앱 인벤터' 프로그램을 통해 어플리케이션을 개발했다. 앱을 동작시키는 코드는 블록코딩을 통해 구현했으며 사용자가 'Bluetooth Connection' 버튼을 눌러 자신의 스마트폰을 신호등과 연동시킨 후, Auto1 / Auto2 버튼을 누르면 각 모드에 맞는 신호체계가 작동하고, 아래에 있는 STOP 버튼을 통해 LED의 반복되는 점등을 멈출 수 있는 형태로 구성했다.

Ⅳ. 결론

우리나라의 신호 체계는 10자형, 로터리 등 다양한 형태가 있으며 이를 위한 교육은 필수적으로 진행되어야 한다. 그렇기에 본 연구에서 (十)자형 도로(사거리) 도로, T자형도로 2가지의 상황에서의 신호체계를 분석하고 이를 아두이노 알고리즘으로 구현하였으며, 4색 신호등 모형을 스마트폰과 연동하여 사용자가 직접 신호체계 유형을 변경하고 이에 맞추어 실제 도로에서의 신호체계와 동일하게 LED를 점등 시키는 알고리즘을 아두이노로 구현하였다.

03 주행 중 화면 인식 개선을 위한 삼중 타원 기법 및 주행 알고리즘

Automated Vehicles: The use of Triple Ellipse Method and driving algorithm for the improvement of screen recognition

강성주, 안동연 (하나고등학교)

Ⅰ. 서론

지금까지 자율주행 자동차의 차선 인식은 바둑판식 기법을 이용해 왔다. 바둑판식 기법은 3행 7열의 인식 선을 사용하여 주행 알고리즘에 필요한 L값 3개, R값 3개, V값 7개를 측정한다. 이처럼 여러 개의 값을 사용하는 기법은 일부 인식 값 측정에 오류가 발생하더라도 주행 경로를 복구할 수 있는 장점이 있다. 하지만 이는 너무 널리 알려진 방법이고 자율주행을 위한 코드 작성 시 복잡하게 짜인다는 단점이 발생한다. 이를 해결하기 위해 본 연구에서는 인지할 수 있는 점의 개수는 최소화하되 정교하게 운행할 수 있는 차선 인식 기법을 제안한다.

Ⅱ. 삼중타원 기법

1. 삼중 타원 기법의 특징 (Features of Triple Ellipse Method)

해당 기법은 3행 7열을 사용하는 바둑판식 기법과는 다르게 3개의 타원을 이용해서 인식 값을 유동적으로 받아들인다. 바둑판식 기법은 직선 주행 시에는 필요 없는 값들도 항시 측정되어 데이터의 과도한 축적으로 통신에 무리가 가는 바가 있었지만, 삼중 타원 기법은 타원과 차로의 교차점만을 인식하여 교차점에 대한 기울기 값만 반환하기 때문에 데이터를 최소화할 수 있어 유용하다. 해당 기법은 3개의 타원을 바깥쪽에 위치하는 순서대로 C0, C1, C2라 정의한다. 이때 C0와 C1은 직진과 회전 주행을 위해 사용되고 각각의 값은 0 ~ (타원의 반지름 수치) 범위를 가진다. C2는 돌발상황 발생 시 장애물 감지 및 정지에 사용된다. 이는 수치적인 측정을 통한 주행 조정보다는 안전장치 및 정지거리 유지에 이용되기 때문에 차선이나 장애물이 해당 타원과 교차했는지의 여부를 Boolean Type으로 반환한다. 각각의 타원은 중앙의 빨간 선을 기점으로 우측과 좌측이 구별되며 이를 중심으로 L값과 R값이 각각 중앙으로부터 얼마나 떨어져 있는지를 측정하여 반환한다. 타원의 가장 밑부분을 기준으로 하여 가장 가까운 교차점부터 순서대로 Am_n의 형식으로 A는 방향, m은 타원의 크기, n은 몇 번째 교차점인지를 나타낸다. [그림 4]에 보이는 C0의 왼쪽 교차점을 예시로 보면 L0_0, L0_1, · · ·, L0_n과 같이 정의하여 나타낼 수 있다. 위 방법을 통해 측정한 L값과 R값을 타원의 방정식에 대입하면 중심에 대한 x좌표와 y좌표를 구할 수 있고 이를 이용하여 L값과 R값의 기울기가 어떻게 되는지 측정해 기울기 S로 반환한다.

2. 삼중 타원 기법의 주행 알고리즘 (Driving algorithm of Triple Ellipse Method)

2.1 직선 주행 알고리즘 (Straight driving algorithm)

타원 C0에는 차선과의 교차점이 [그림 4]와 같이 C0 좌측에 L0_n값 2개, 우측에 R0_n값 2개로 최대 4개의 값이 측정된다. 삼중 타원 기법은 인식 값들을 이용해 L0_0와 R0_0의 기울기 값인 S0_0와 L0_1과 R0_1 값의 기울기인 S0_1값을 반환한다. 차선의 중앙에서 직선 주행 시 동일한 y축 선상에 L값과 R값이 존재하게 되고 기울기 값이 0에 수렴한다. 이러한 점을 이용하여 측정된 기울기의 부호를 판단하여 중앙에서 직선 주행을 할 수 있도록 조절한다. 또한 주행 시 L값과 R값의 차를 이용하여 도로의 폭을 측정해 기존 도로의 폭에서 현재 주행 중인 도로의 폭이 좁아졌는지 판단해 본래 값을 유지할 수 있도록 조절한다.

그림 1. 직선 주행 중 인식되는 화면 (삼중 타원 기법)

2.2 직각 곡선 주행 알고리즘 (Right-angled curve driving algorithm)

곡선 주행의 경우 C1의 사용률이 가장 높다. 한쪽 차선이 안보이고 반대쪽 차선이 직각 곡선 형태를 이룰 때 C0, C1에는 L값과 R값이 1개씩, 각각의 타원에 총 2개가 인식된다. C0에서의 교차점이 2개 이하임과 동시에 C1에서의 교차점이 2개 이하가 되었다면 회전 구간에 진입하였다고 판단한다. 회전 구간을 인식했다면 C1에서의 교차점의 좌표로 기울기 값을 구해 일정 각도로 회전한다. 직각 곡선의 경우 처음에 경사가 완만하다가 차츰 직선 도로로 돌아오면 경사가 급해지기 때문에 S1_0의 절댓값에 반비례하여 회전한다. C0의 기울기 값 S0_0에 대해 S0_0 > 0 라면 우회전을, S0_0 < 0 라면 좌회전하게 된다.

2.3 둥근 곡선 주행 알고리즘 Curve driving algorithm)

둥근 곡선 주행의 경우 직각 곡선 주행 알고리즘과 유사하게 작동된다. 한쪽 차선이 안보이고 반대쪽 차선이 둥근 곡선 형태를 이룰 때 C0, C1에는 차선과의 교차점이 L값 1개, R값 1개로 각각 2개씩 만들어진다. [그림 5]와 같이 C0, C1과 차선의 교차점이 각각 2개 이하라면 회전구간에 진입하였다고 판단한다. 회전구간을 인식했다면 C1을 통해 구한 S1_0의 절댓값에 반비례하여 일정 각도로 회전한다. C0의 기울기 값 S0_0에 대해 S0_0 > 0 라면 우회전을, S0_0 < 0 라면 좌회전하게 된다. 둥근 곡선 주행의 경우, 처음 진입로부터 S1_0의 절댓값이 매우 느리게 증감하므로 직각 곡선 주행과는 달리 조향 값이 천천히 변화한다. 이를 이용하면 직각 곡선에 비해 상대적으로 경사가 완만한 둥근 곡선 트랙을 주행할 수 있다.

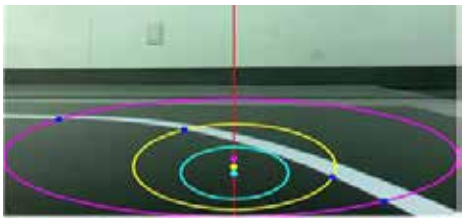


그림 2. 곡선 주행 중 인식되는 화면 (삼중 타원 기법)

2.4 장애물 감지 및 정지

장애물 감지 및 정지의 경우, 3개의 원 중 C2의 비중이 가장 높다. 장애물이 없는 도로를 주행하는 동안에는 일반적으로 C2에 교차점이 생길 일이 없다. 하지만 전방에 장애물이 존재하게 된다면 [그림 6]과 같이 C2 위에 1개 이상의 교차점이 생긴다. 이때 장애물이 주행 안전거리보다 가까이에 존재한다 판단하고 자율주행 자동차가 정지하도록 알고리즘을 설계한다.

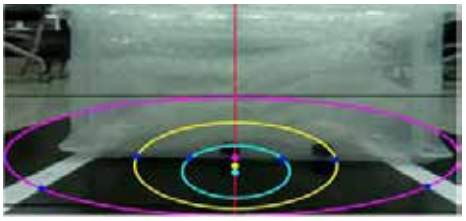


그림 3. 장애물 감지 및 정지 사진 (삼중 타원 기법)

3. 삼중 타원 기법의 장단점 (Pros & Cons of Triple Ellipse Method)

삼중 타원 기법은 알고리즘 작성에 있어 S값만으로도 코드 구성이 가능하다. 알고리즘 설계 방법에 따라 L값과 R값의 세세한 수치를 추가로 사용하여 안정적인 주행 코드를 작성할 수도 있다. 또한 불필요한 수치 사용을 최소화함으로써 기기의 과부하 또는 시간지연 현상이 발생하는 것을 방지할 수 있다. 기존의 방식은 유한개의 인식 값을 가지고 있어 차선이 동시에 하나의 인식 선에 교차하면 둘 중 선명한 값 1개만 인식된다는 단점이 있었지만, 삼중 타원 기법은 타원과 교차하는 모든 점을 사용하여 S값을 반환하기 때문에 필요한 정보가 누락되지 않는다는 장점이 있다. 장애물을 감지해 정지하는 과정에서도 차량과 가까운 내부 원에 교차하는 지점이 생겼는지를 이용해 Boolean Type으로 간단하게 반환하여 알고리즘 구성이 가능하다.

Ⅲ. 결론

본 연구에서는 3행 7열의 인식 선을 사용하는 바둑판식 기법의 단점들을 보완한 삼중 타원 기법을 제안한다. 기존 알고리즘과는 다르게 3개의 타원을 이용한 차선 인식으로 주행 알고리즘 구성에 필요한 인식 값들의 유효성이 사라지지 않도록 고안했다. 제안된 알고리즘은 바둑판식 기법의 주행 결과에 대한 내용을 토대로 이론적으로 접근하는 과정을 거쳐 유효성을 검증하였다.

04 곡선 차선에서의 원활한 주행을 위한 기본 알고리즘 및 후진 알고리즘의 적용

Application of Basic and Reverse Algorithms for Smooth Driving in Curved Lane

강동현 (선덕고등학교)

Ⅰ. 서론

자율주행 자동차는 실시간으로 차선을 인식해 알고리즘의 코드를 적용해 자신의 주행 방향 및 속도를 계산하고, 이를 통해 사람의 개입 없이 주행을 완수하는 것을 목표로 하는 자동차이다. 본 탐구에서 사용된 자율주행 자동차는 차선 인식 데이터를 네트워크를 통해 전송해 계산과정을 거치고, 그 결과값을 다시 전송해 주행을 진행한다. 이 과정에서 실시간의 상황을 적용한 주행을 실행하지 못해 차선이탈 및 주행 오류가 발생하게 된다. 본 탐구에서는 이런 문제를 해결하고자 하였다.

Ⅱ. 문제점 인식

차선 인식에서의 데이터를 전송하는 과정에서 시간적 딜레이가 생기고, 이에 따라 실제 상황보다 늦은 화면을 보고 계산을 진행해 곡선 구간에서 차선을 벗어난 이후 회전하게 되는 문제가 발생하였다. 이는 완만한 곡선 차선과 직각 곡선 차선 중 직각 곡선 차선에서 회전을 인식하는 것이 더 늦어 그 문제가 더욱 부각되어 나타났다.

Ⅲ. 연구방법

1. 직선 차선 인식 시의 기본 알고리즘 작성

그림 1. 자주차 직선 차선 알고리즘

자주차의 V[3]값을 기준으로 직선 차선과 곡선 차선의 주행을 구분하였다.

표 1. V[3]값에 따른 직선 및 곡선 주행 차선 구분

V[3] > 120	직선 차선 주행
65 < V[3] < 120	곡선 차선 주행
V[3] < 65	후진 알고리즘 적용

직선 차선 판단 후 차선의 중앙으로 주행하기 위해 양측 차선의 기울기 차이를 통해 차량의 기울어짐을 판단하였다. 이후 steer 변수의 미세 조정을 통해 차량의 주행에서 차선 침범이 없도록 하였다.

2. 곡선 주행 알고리즘 작성

그림 2. 곡선 주행 알고리즘

V[3]값을 토대로 곡선 주행을 판단한 후, V[3]값을 다시 한번 적용해 완만한 회전 주행과 급격한 회전 주행을 구분하였다. 완만한 회전 주행의 경우에서는 차선이탈을 최소화하기 위해 steer값을 V[2]값과 V[4]값의 차로 결정하였고, 이에 따라 차선 기울기에 따라 차량 조향값을 유동적으로 설정할 수 있었다. 급격한 회전 주행에서는 차선과 차량의 거리가 매우 가까운 것을 전제로 하기에 steer 값을 최대로 설정하여 곡선 구간을 주행하고자 하였다.

3. 회전각 한계의 문제점

직각 곡선 주행에서는 차량의 데이터에서 V[3]값이 회전 판단 범위 내로 들어와도 좌회전과 우회전을 구분하는 것이 늦고, 곡선 주행이라 분류를 한 상태에서도 V[2]와 V[4]의 차이 값이 차량의 회전을 만들 만큼의 조향값을 설정할 수 없었다. 연속 곡선 주행에서는 차선을 실시간으로 추적하기에 정면에 있는 차선을 넘어가도 뒷 차선을 정상 차선이라 인식해 차선을 이탈한 상태로 주행을 계속한다는 문제와 유턴과 같은 차선에서는 단순 조향값 조정으로만 해결하려고 하면 차선이 일부만 보일 때, 차선에 따라 주행을 결정하지 못하는 경향이 있었다. 이 문제를 해결하고자 후진 알고리즘을 적용하는 방안을 고안하였고, 이를 적용하고자 하였다.

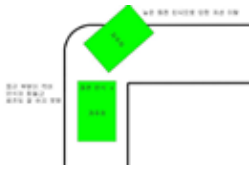


그림 3. 직각 차선에서의 문제점

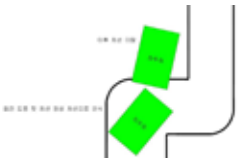


그림 4. 연속 곡선 차선에서의 문제점

4. 네트워크 지연으로 인한 문제점

자율주행 자동차에서 사용되는 기기 중 와이파이 모듈이 노트북과의 통신을 통해 차선 분석을 진행하도록 돕는다. 그 과정에서 통신상의 딜레이가 존재하고, 이는 때에 따라 다르지만, 주위 네트워크의 영향을 쉬게 받는다. 평균적으로 1-3초의 딜레이가 생기며, 딜레이가 심한 경우 1분간의 딜레이가 생기는 경우도 있었다. 딜레이를 해결하기 위해 주위의 네트워크를 제거한 상태로 알고리즘을 실행하였고, 그럼에도 1-3초의 딜레이가 유지되어 이에 맞는 알고리즘을 고안해 내어야 했다.

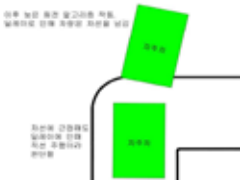


그림 5. 딜레이가 생기는 경우의 주행 문제

5. 후진 알고리즘의 적용

```
elif (V[3]<= 65):
    my_velocity = -100
    my_steer = 0
    print('후진')
```

그림 6. 후진 알고리즘

후진 알고리즘 또한 이전의 알고리즘의 판단 조건에 맞추어 v[3]값을 기본 판단 조건으로 사용하였고, V[3]값이 65보다 작은 경우 후진 알고리즘이 작동하도록 코딩하였다. 조향값을 기존의 조향값의 반대가 아닌 0으로 설정한 이유는 알고리즘이 한번 실행되는 동안 V[3]값이 65보다 큰 값이 된다면 문제 없지만, 그렇지 않고 계속 차선에 가까워 65 이하의 값이 유지되는 경우에는 후진하고 있는 조향값의 반대값이 되어 직진할 때의 조향값으로 후진하게 되므로 0으로 설정해 주행의 안정성을 높이고자 하였다. 후진 알고리즘을 처음 설계할 때는 시간 변수를 고려해 몇 초간 후진을 유지하는 코드를 작성하고자 하였다. 딜레이가 존재하지 않는 상황이라면 위와 같은 코드를 실행시킨다면 V[3]이 65인 지점에서 전진과 후진만을 반복시킬 것이다. 하지만, 딜레이를 고려한다면, V[3]값이 정확히 65에서 후진하는 것이 아니라 65에서 더 주행을 진행했을 때 V[3]값이 65로 측정되어 후진 알고리즘이 실행되고, 차량이 65를 벗어난 순간에도 코드상에서 감지되는 화면에서는 V[3]값이 65를 벗어나지 않은 것으로 측정되어 후진 알고리즘을 어느정도 더 실행하게 된다. 이에 코드상에 시간 변수를 적용할 필요가 없었으며 변수를 적용한 코딩을 진행해 보았을 때는 시간을 감지하는 동안 화면인식 코드가 멈추는 문제가 생겨 코드를 수정해 후진 알고리즘을 그림 6의 코드로 완성시켰다.

Ⅳ. 결론

자주차의 주행에서 회전각 한계의 문제가 연속곡선 차선에서의 주행과 직각 곡선 차선의 주행의 걸림돌이 되었고, 네트워크 지연은 기본적으로 코드를 실행할 때, 과거의 화면을 보고 현재의 상황을 예측해 주행시켜야 하는 문제들이 발생하였다. 이러한 문제를 해결하기 위해 기존의 직선 알고리즘과 회전 알고리즘에 추가적으로 후진 알고리즘을 적용하였고, 딜레이가 없었다면 시간변수를 적용해야 했던 후진 알고리즘은 주행에 방해되는 딜레이를 이용해 더 안정적이고 간단한 코드를 이용해 작성해 주행 안정성을 높일 수 있었다.

05 자주차 차량의 조향장치의 하드웨어적 개선 및 차량구조 변경 모델링

Hardware Improvement and Vehicle Structure Change Modeling of Steering System of Jajucha

서준하 (하나고등학교)

Ⅰ. 서론

자동차는 기본적으로 바퀴와 모터, 차체로 구성된다. 그리고 차량이 실제 주행을 하기 위해서는 안정적인 조향장치도 필수적이다. 과거 자동차가 발명된 이래로, 자동차의 조향장치에는 많은 변화가 있었다. 특히 현대의 자동차 형태가 만들어진 이후부터는 조향장치와 더불어 현가장치와 같이 차량의 승차감과 조향축의 안정성을 위한 구조가 개발이 되어 왔고, 이 과정에서 자동차의 조향축의 형태도 [그림 1]과 같이 조향휠, 조향축, 조향기어를 지나 릴레이 로드와 타이로드, 너클암, 휠 피벗으로 연결되는 순환 볼 식 형태와 조향축에서 피니언으로 연결되어 래크기어, 타이로드, 너클암, 휠 피벗으로 연결되는 래크/피니언 식 형태 두 가지의 기본적인 구성으로 이루어져 있다.

오늘날 거의 모든 자동차에는 사다리꼴 기구(trapezoidal mechanism)를 기본으로 하는 애커먼 - 조향장치 (Ackermann steering)가 사용되고 있다. 현재 자주차의 경우에는 이러한 조향장치와 비슷한 구동 방식으로 사다리꼴 기구를 변형시킨 형태로 구성된 조향장치를 가지고 있으나, 현재의 자주차 차량의 경우 조향장치를 제작하는 과정에서 더 많은 부품이 정밀한 조립 과정을 필요로 하여 조립 후 오차 발생 가능성이 높다. 뿐만아니라 전면부 바퀴의 안정성의 측면에서 바라보았을 때도, 바퀴 사이의 중심링크가 존재하지 않고, 바퀴보다 높은 위치에서 두 바퀴가 연결되어 결과적으로 주행 과정에서 바퀴가 많이 흔들리는 경향을 보인다.

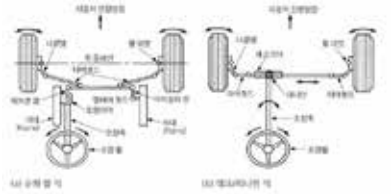


그림 1. 조향장치 기본 구성

이와 더불어 회전 반경의 측면에서도 기존 차량의 형태에서는 넓은 회전 반경을 기대하기는 어려운 형태이다. 이에 본 연구에서는 기존의 자주차 차량의 전면부 조향장치의 구조를 보다 안정적으로 모델링하여 기존 차량에 적용하며, 적용하는 과정에서 예상되는 자주차 차량의 개선점에 대하여 연구하였다. 이와 더불어 후륜 조향장치의 구조를 제안하여, 현재 자주차의 회전 반경을 개선한 주행 구조를 제작하여 자율주행 자동차의 주행 및 적용과정에서, 더욱더 다양한 기능을 기대하고, 이에 따라 얻을 수 있는 효과에 대해 제안한다. 연구 과정에서 생기는 자주차 차량구조의 일원화를 예방하고자 전체 차량의 구조에 대한 모델링은 진행하지 않고, 조향장치의 기본적인 모델링만 진행하였다.

Ⅱ. 조향장치

1. C - MDPS 구조

현재 자주차 차량의 형태를 분석해본 결과 앞바퀴 조향을 하기 위한 구조[그림 2]로 실제 차량에 적용되고 있는 조향시스템과 비교하자면 C-MDPS 구조를 채택하고 있는 것으로 확인되었다.

그림 2. 자주차 조향장치 구조

자주차 차량의 형태에서 이러한 조향 방식을 사용함으로써 차량에 사용되는 부품의 수와 공간을 줄이고, 결과적으로 차량의 주행효율을 높이기 위해 구조되었다. 그러나 결과적으로 C - MDPS 기반 구조화를 진행하는 과정에서 모터와 바퀴 사이의 부품들의 오차로 인한 반응속도의 지연이 발생하여 결과적으로 차선이탈 등의 주행능력에 영향을 미치는 단점이 발견되었다.

1.2 R-MDPS 구조

R-MDPS는 전동기가 바퀴를 조향하는 렉(rack)에 연결되어있는 구조로, 기존 유압 방식 스티어링과 조향방식이 비슷하다는 특징을 가지고 있다. 차량을 구성할 때 렉 부위에 충분한 공간이 필요하기 때문에 중형 이상의 차종에 주로 적용되고 있고, 직접 휠을 구동하는 방식이기 때문에 이를 구동 시키는데 사용되는 출력이 크다는 특징이 있다. 그러나 R - MDPS 구조 기반 구조화 시에 모터와 바퀴 사이의 거리가 줄어들게 되고 결과적으로 조향의 안정성 향상에 초점을 맞추고 개선된 모델링을 진행하고자 하기 때문에 최대한 적은 면적에 R - MDPS 기반 구조를 적용시키는 모델링을 진행하였다.

2. 자주차 구조 변경

Autodesk Fusion360 프로그램을 이용해 다음과 같은 기본 축을 구상하였다. 바퀴 사이를 잇는 축을 3개 배치하여 기존의 바퀴 윗부분에 배치되었던 연결부분을 아래로 내리는 구조를 채택하여 바퀴의 안정성을 높이고자 하였다.

그림 3. 기본 바퀴 축

축 3개를 연결하고, 후에 모터와 연결하여 평행사변형 조향 방식으로 차량의 방향 전환을 진행하기 위해서 중간축의 길이를 다른 두 축 보다 길게 하여 양 끝부분에 삼각형 형태의 프레임을 연결하여 1번 축(그림 3 기준 위에서부터 1,2,3) 위치에 조향 모터와 결합되는 톱니 형태를 모델링하여 평행사변형 조향 구조를 적용시키는 구조를 형성하였다. 2번 축은 삼각형 프레임과 고정되어있고, 1, 3번 축은 삼각형 프레임에 베어링을 연결하여 유동적인 움직임이 가능하도록 하여 바퀴의 방향 조정이 가능하도록 설계하였다.

그림 4. 바퀴 축 구조 전면부

그림 5. 바퀴 축 구조

2.1 조향장치 설계에 따른 바퀴 구조변경

앞선 모델링을 통해 설계한 바퀴 축 구조를 기반으로 하여 기존의 원판 형태의 바퀴를 실제 차량의 바퀴와 비슷한 형태로 모델링을 진행하였다. 기존의 원판형태의 바퀴의 경우 내구성이 부족하며, 지면에 닿는 타이어 부분의 면적이 적어서 바퀴당 받는 자주차 무게에 대한 부담이 비교적 높은 점을 함께 고려하여 설계하였다. 다음 그림은 앞선 바퀴 축 구조를 기반으로 하여 새로 설계한 바퀴를 장착한 형태의 모델링이다.[그림 6,7] 다음 형태에서 볼 수 있듯, 바퀴의 경우 별도 베어링을 통해 삼각프레임에 결합 되어있는 상태이고, 추후 4륜구동 차량을 제작하는 과정에서 삼각 프레임 부분에 모터를 장착하는 등의 활용이 가능하도록 설계하였다.

그림 6. 조향장치 구조

그림 7. 조향장치 구조 III

Ⅲ. 후륜조향

본 후륜 조향의 경우 인공지능 기반 자율 주행 자동차에 적용되는 것이기 때문에, 차속 감응식 총륜 조향 시스템을 적용 시키고자 한다. 따라서 프로그래밍 과정에서 속도에 따른 후륜 조향 모터 값의 변화 각을 전면부 각도와 함께 고려하여 조절하도록 하고, 저속주행 시 전륜과 반대 방향, 즉 역 위상으로 조향되고, 고속주행 시 전륜과 같은 방향, 즉 동 위상으로 조향하도록 설정하며, 차체의 안정성을 위해 최대 변화각은 5°로 설정하여 후륜 조향 시스템을 적용시켰다.

그림 8. 후륜 조향장치 구조

후륜 조향의 경우 미세한 조향각 조절이 필요하기 때문에, 삼각 프레임이 꼭짓점이 안쪽으로 오게 하여 축 1, 3번의 양 끝과 삼각 프레임의 양변을 실린더로 연결하여 설정된 값에 따라 조정 되도록 설계하였다.

Ⅳ. 결론

바퀴 축과 바퀴의 설계를 새롭게 진행하여 기존의 불안정한 조향장치에 대한 개선을 제안하였고, 이를 구조적 근거를 통해 설계된 모델에 대한 신뢰도를 높였다. 이와 더불어 자율주행과 관련하여 함께 적용되어 차량의 이동성을 극대화 할 수 있는 후륜 조향을 함께 제안하였다. 후륜의 안정성을 위해 실제 진행되고 있는 후륜 조향 연구에 기반하여 최대 변화 각과 후륜 시스템 개발 시 고려되어야 하는 부분들에 대해 제안하였다.

06

유한 상태 기계를 이용한 돌발 상황에서 자율주행자동차의 장애물 회피 알고리즘에 관한 연구

Obstacle Avoidance Algorithm for Autonomous Driving Car in Unexpected Situatuon Using Finite State Machine

임서진 (통진고등학교)

Ⅰ. 서론

자율주행자동차에서 가장 기본이 되는 원리는 흰색 차선을 따라 주행하는 것이다. 전 세계적으로 통용되는 도로 규정에 따라 흰색 실선 또는 점선을 따라 주행하고, 노란색 중앙선을 넘지 않는 등 자율주행자동차 또한 기존의 도로 형태에 의존하여 주행하게 된다. 지금까지의 라즈베리 파이(Raspberry Pi)를 이용한 자율주행자동차 관련 연구는 이와 같이 가장 기본이 되는 도로 형태를 전제로 하여 진행되어 왔다 [1]. 하지만 실제 도로 상황을 적용하고자 한다면, 당장 지금까지는 흰색 실선 차선을 따라 진행해왔던 곡선 주행도 유도선에 의존하거나 이마저도 없는 경우가 대다수이며, 정형적인 도로 형태에서 벗어날 필요성이 있는 예외 상황이 있기 마련이다. 예를 들어, 차량 통행량이 많은 도로를 구급 차량이 지나가고자 할 경우 자율주행자동차가 상용화된 시점에선 오히려 구급 차량의 이동이 원활하지 못하게 될 수 있다. 이러한 문제점에 주목하여, 본 연구에서는 돌발 상황에서 라즈베리 파이를 이용한 자율주행자동차 모델이 어떻게 판단하여 주행해야 하는지에 관하여 Finite State Machine을 통해 제안하고, 이를 알고리즘화하여 주행에 적용한다.

Ⅱ. 상황설정

1. 의도성과 즉각 인지 판단 여부

실제로 자율주행자동차가 상용화된 시점에서 앞서 말한 예외 상황을 적용하기 위해선, 우선 예외 상황을 의도성과 인지 여부에 따라 크게 두 가지로 분류해야 한다. 첫 번째 예외 상황은 상황을 주도하는 한 차량이 의도성을 가지며, 도로 위를 주행 중인 주변 차량들이 돌발 상황으로 인지하지 않는 상황이다(이하 의도된 상황). 서론에서 말한 예시 상황과 같이 구급 차량의 신속한 주행이 필요한 경우가 그렇다. 이와 같은 경우에는 신속한 주행을 필요로 하는 '주도 차량'이 자율주행자동차 통신망에 정보를 전송하여 GPS (Global Positioning System)상의 주변 차량들이 받은 정보에 따른 특수한 움직임을 가져야 할 것이다. 두 번째 예외 상황은 주도 차량이 존재하지 않으며, 자율주행차량의 자체적인 인지 판단에 의해 움직임을 가지는 경우이다(이하 돌발 상황). 실제 도로 위의 관점에서 보면 일시적인 통신 오류 등의 문제로 발생하는 교통사고가 이에 해당한다. 본 연구에서 사용된 자율주행자동차 모델의 경우는 차단기를 마주하였을 때 차량이 정지하는 '라이더 스탑(Lidar Stop!)' 알고리즘이 이에 해당한다.

1.1. 의도된 상황

주도 차량이 존재하며 GPS를 기반으로 움직임을 가져야 하는 상황에서는 서버 통신으로 전송받은 정보와 주변 인식을 통해 위기를 모면해야 한다. 주도 차량이 자율주행자동차 서버로 '구급 차량 진행 중, 최고 위급 단계'라는 정보를 전송하면 이는 GPS를 통해 구급 차량 진행 경로 내의 차량들이 받게 된다. 이후 주변 차량들은 GPS와 차량 자체 카메라 및 센서를 통해 주도 차량이 인접하였을 경우 주변 상황 판단 하에 차선 유지 등의 기본 주행 원칙으로부터 벗어날 수 있다.

1.2. 돌발 상황

자율주행 차량이 직접적인 인지 판단을 통해 돌발 상황으로 결론 내리는 경우에 차량이 할 수 있는 판단은 '멈추기'와 '회피하여 주행 유지하기'로 나눌 수 있다. '멈추기' 판단은 차단기를 마주한 상황에서 차량이 정지할 때의 알고리즘과 동일하지만, '회피하여 주행 유지하기' 판단은 자율주행자동차 모델의 알고리즘으로 다룬 전례가 없다. 따라서 본 연구에서는 직선 도로 중앙에 장애물이 존재하여 이를 피하며 주행을 유지하는 상황을 제시하고(그림 1), 이에 대한 알고리즘을 제시한다.

그림 1. 장애물 회피 상황

Ⅲ.주행 알고리즘

1. Finite State Machine

1.1. Finite State Machine의 정의 및 활용

본 연구에서는 돌발 상황에서의 알고리즘과 앞서 가정한 돌발 상황과 같은 특수 상황 이외에서의 자율주행 자동차 모델의 일반 주행 상황에 대한 알고리즘의 배경으로 Finite State Machine (유한 상태 기계, 이하 FSM)을 사용하였다. FSM이란 동기 순차 회로를 기술하는 추상화 모델로, 입력에 의해 상태를 바꾸면서 출력된다. FSM은 논리 설계의 복잡도를 단순화하여 인식을 용이하게 한다는 장점이 있다. 본 연구에서는 FSM을 <그림 2>의 구조로 장애물 회피 알고리즘에 적용하였다. (FSM 알고리즘은 다른 영향을 주는 명령이 존재하지 않으며 다음 State로의 전사가 이루어지지 않을 경우 전사 조건 충족 시까지 계속해서 이전 State가 반복 실행되나, 아래의 <그림 2>에서는 생략하였다.)

그림 2. FSM 장애물 회피 알고리즘

1.2. FSM의 알고리즘 적용

1.2.1. Status 0, 1, 2, 3의 정의

기존 직선 및 곡선 주행에 적용되는 알고리즘을 'status 0 (직진 조정)', 'status 1 (우회전 조정)', 'status 2(좌회전 조정)'으로 FSM 알고리즘 정의하였으며, 전방 라이더 센서를 통해 장애물을 감지하여 장애물이 차량과 일정 거리 내로 가까워질 때를 status 3으로 정의하였다. status 3부터 time 함수를 이용하여 매 state마다 새로운 시작 시간 값을 설정해주고, 이를 'status n'에 상응하여 'start_n'으로 이름 짓고 전역 변수로 지정해주었다(그림 3).

그림 3. FSM을 이용한 status 설정 1

1.2.2. Status 4, 5, 6, 7의 정의

'status n (4 =< n =< 6)'은 조건이 충족될 시 'status n + 1'로 전사되도록 구성하였다. 여기서 status 4는 이전의 status 3에 의해 '장애물 회피하여 주행 유지' 알고리즘이 실행될 경우 이 때의 시간 값과 status 4의 현재 시간 값의 차를 구하는 식을 통해 상태 지속 시간을 지정한다(그림 4).

그림 4. FSM을 이용한 status 설정 2

Ⅳ. 결론

본 연구에서는 시험 도로와 돌발 상황을 설정하고, FSM 알고리즘을 통해 예외 상황 중 돌발 상황의 경우에 자율주행 자동차 모델이 인식된 장애물을 회피하여 주행하던 상태를 유지하는 방식의 주행을 제안하였다. 제안된 알고리즘은 자율주행자동차 모델이 돌발 상황의 경우 차선 유지의 틀에서 벗어나 위기를 모면할 수 있도록 하는 목적으로 주행을 완료함으로써 알고리즘의 유효성을 검증하였다

07 자율 주행 자동차의 안정적인 주행을 위한 효과적인 중앙 보정 알고리즘

Effective Cental Correction Algorithm for Stable Traveling of Automat

김진형, 윤영서 (통진고등학교)

Ⅰ. 서론

도로에서의 차량 충돌 위험을 줄이고 도로 안전을 높이는 것은 엔지니어의 오랜 과제였다. 도로에서 차량 간 교통사고의 주된 원인은 차량의 기계적인 결함보다 운전자의 핸들 조작 실수와 갑작스러운 차선변경으로 인해 발생한다. 이러한 인적인 요인에 대한 해결 방안 기술로 운전자 보조 시스템이 개발되고 있으며 이 기술은 돌발적인 상황의 대처와 자율적인 차선 유지에 더 안전하고 편리한 기능으로 적용되고 있다. 본 연구팀은 인식한 차선의 위치에 따라 바퀴의 기울기를 달리하여 중앙을 유지하는 기존 방식의 중앙 보정 알고리즘을 여러 차례 사용하였다. 이때 여러 문제점이 발견되었지만 그중 경우에 따라 곡선 코스에 진입하기전 차체의 위치가 계속해서 바뀐다는 한계가 있다는 점에 집중하여 더 효과적인 중앙 보정 알고리즘을 제작하여 문제를 해결하고자 하였다. 고안한 중앙 보정 알고리즘은 차체를 효과적으로 중앙에 위치시킴에 따라 기존보다 더 안정적이고, 이상적인 주행을 하게 하였다.

Ⅱ. 본론

1. 중앙 보정 알고리즘

1.1 중앙 보정 알고리즘의 필요성

기존 코드도 직진할 때 문제없이 잘 가지만 조금씩 비틀리다가 중간에 곡선이 나올 경우 약간의 오차가 생겨 결과에 차이가 나는 상황이 발생한다. 그러나 중앙 보정 알고리즘을 추가할 경우 이러한 미세한 오차를 줄여 그만큼 직진, 곡선 주행이 좀 더 안정적으로 주행하게 되어 차선을 이탈하는 오류가 발생할 확률을 적게 하는 역할을 한다.

1.2 기존의 중앙 보정 알고리즘의 문제점

기존의 중앙 보정 알고리즘은 차체를 중심으로 좌우측 차선의 거리 변화를 인식하여 차체를 중앙에 유지시키는 방식을 갖고있었다. 하지만 이런 방식의 중앙 보정 알고리즘은 외부적인 요인으로 인해 차선의 폭이 달라지는 경우 실제 차체와 차선 사이의 거리가 이론상의 거리와 달라져 차체를 중앙에 유지시키지 못하는 경우가 생기게 된다. 이 밖에도 실제 차선은 완벽한 직선이 아니기 때문에 이 부분에서 또한 문제가 생길 수 있다.

1.3 중앙 보정 알고리즘의 개선 방향

차선의 중앙값을 차선의 폭이 변하더라도 알아낼 수 있도록 좌우측 차선의 위치를 확인하고 계속해서 중앙값을 찾는 방향으로 개선해야 할 것이다.

2. 개선된 중앙 보정 알고리즘

2.1 초기 변수 설정

그림 1. e를 전역변수로 바꾸는 코드

e값이 지역 변수로 설정되어 있다면 함수가 종료되고 부터 다시 시작될 때 까지 e값이 메모리에서 없어지게 된다. 이때 e값이 없음으로 인해 차량이 다른 방향을 향하게 될 수 있고 이는 주행에서 큰 문제를 이르킬 수 있다. 따라서 global e를 추가하여 지역변수였던 e를 전역변수로 바꾸어 혹시 모를 경우를 대비할 수 있다.

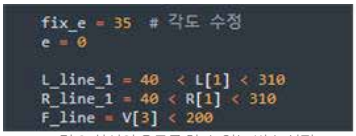


그림 2. 차선의 유무를 알 수 있는 변수 설정

차량의 전방 및 좌우측의 차선의 유무를 알아내기 위해 위와 같은 초기 변수를 설정해준다. L[1], R[1]의 최댓값과 최솟값을 이용하여 L_line_1과 R_line_1을 True와 False의 경우로 나누어 좌우측에 차선의 유무를 알아낸다. V[3]의 값이 200보다 작을 때 차선이 앞에 있음을 감지하고 F_line을 활성화 하게 된다

2.2 개선된 중앙 보정 알고리즘의 구조

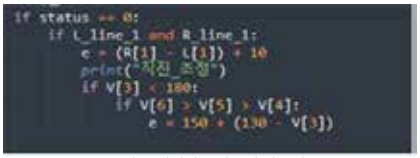


그림 3. 개선된 중앙보정 알고리즘

만약 L_line_1 and R_line_1 즉 L(1)과 R(1)에 차선이 인식된다면 계속해서 e값을 바꿔주며 중앙을 유기할 수 있도록 도와준다. 또한 L(1)과 R(1) 모두에서 차선이 인식되지만 직선주행이 아닌 곡선주행을 해야하는 직각 코너의 경우를 고려하여 L(1)과 R(1) 모두에서 차선이 인식됨과 동시에 앞에 차선이 있다면 e값을 급격하게 바꾸어 직각 코너를 돌 수 있게 하였다

Ⅲ. 결론 및 한계

1. 결론

본 연구에서 고안한 개선된 중앙보정 알고리즘은 2개의 차선을 이용하여 차선의 중앙을 찾아내어 차체를 중앙에 유지시키는 방법을 사용하였다. 이는 차선의 중앙을 계속해서 찾아내기 때문에 기존의 중앙 보정 알고리즘의 문제점을 해결했다고 할 수 있을 것이다.

2. 한계

효율적인 곡선 주행을 위해서는 곡선 주행을 하기 전 차체가 완벽한 중앙보다는 차선의 바깥쪽 부분에 위치한 후 주행하는 것이 좋다. 하지만 위 코드는 항상 차선의 중앙을 찾고 유지시키기 때문에 이런 부분은 구연할 수 없다는 한계가 있다.

또한 좌우측의 차선중 한가지 차선을 인식하지 못하였을 때 차선의 중앙을 찾아 낼 수 없기 때문에 차체를 중앙에 유지 시키지 못한다는 한계점 또한 있었다. 이를 해결 하기 위해서는 한 개의 차선 만으로도 차체를 중앙에 유지시킬 수 있는 알고리즘을 개발하여야 할 것이다.

08 자율 주행 자동차의 정확한 'T' 자형 곡선 주행을 위한 알고리즘 연구

A Study of Regulation Algorithms Accurate T-Shape Curve Driving in self-driving cars

이은혜, 남다현, 노여송, 심은성 (미림여자고등학교)

Ⅰ. 서론

현재 레벨 2(부분적 자율주행)의 상용화 단계에 있으며, 레벨 2의 첨단 운전자 보조시스템(ADAS)의 주요 기술 중 LKAS는 카메라(camera)와 적외선 센서를 사용하여 도로 영상을 실시간으로 수집하고, 영상 처리 장치(ECU, Electronic Control Unit)로 보내진다. ECU는 운전자가 사전에 방향지시등을 조작하지 않거나 의도하지 않은 차선침범, 차선이탈이 발생하였을 때, 조향 보조장치를 제어하여 주행 중인 차선으로 복귀하는 시스템이다. [1]

이 LKAS 기술을 이용하여 자율주행 자동차에 대한 알고리즘을 입력할 때 안전한 주행을 위해 실제 우리나라 교통 사고 다발 지점을 조사해보았다. 교통 안전 정보 관리 시스템에 따르면 서울의 대부분의 구에서 교통 사고가 가장 많이 일어난 지점은 사거리라는 통계를 확인할 수 있었다. [2]

따라서 앞으로 자율주행 자동차의 상용화를 위해서는 사거리에서의 주행이 안전하게 이루어져야함을 깨달았고, 사거리와 비슷한 형태의 차선인 'T'자형 곡선 차선을 완벽하게 주행하고자 이 연구를 진행하게 되었다. 이 연구를 바탕으로 현대 모비스에서 주최한 청소년 공학리더 자율주행 자동차 경진대회에 출전해 'T'자형 곡선 차선의 알고리즘을 작성하고 자율주행 자동차를 실행해본 결과 'T'자 차선에 진입하는 데는 성공하였으나, 정확한 커브를 성공시키지 못했다. 이를 해결하기 위해 원인을 분석해 본 결과 자동차가 커브

후에 차선과 나란한 위치에 있지 않았기 때문에, 3차선 인식을 혼란스러워 한 것이 원인이라고 판단하였고, 이를 수정하여 더 완벽한 알고리즘을 구현해보고자 한다.

Ⅱ. 본론

1. 'ㄱ'자형 곡선

1.1 'ㄱ'자형 곡선과 일반 곡선의 차이점

회전을 하는 동안 곡선은 ㄱ자 코스를 돌 때와 비교해 비교적 완만하게 돌게 된다. 따라서 곡선 코스는 V값이 완만하지만 지속적으로 줄어들었다가 늘어나게 된다. 그에 반면 ㄱ자 코스는 V의 값이 급격하게 변하지만 지속적으로 변하지 않는다.

1.2'ㄱ'자형 곡선 알고리즘 분석

만약 앞쪽이 직선이고 왼쪽이 오른쪽보다 높은 값이면 직각 좌회전이라는 코딩을 보면,

그림 1.

if -2 < V[5] - V[4] < 2 :

라는 것을 보면 v의 값이 직각 회전에 영향을 준다는 것을 알 수 있는데,V 값을 잡아 직선 회전을 할 때, 어느 정도의 값이 나오는지 확인하여 조절하는 것이다. V값의 한계를 두어 그 값이 된다면 회전을 하게 만든 알고리즘이다.

그 뒤에 'and'가 있다. 'a and b'를 보면 a, b 모두 True 일 때 성립한다.

elif 값에서 elif는 a가 아니면 b라는 뜻이다. if문이 성립되지 않는다면, elif 문을 써야한다는 뜻이다. 반대로 이것은 우회전도 적용할 수 있는데 우회전을 할 때 좌회전과 반대로 오른쪽이 왼쪽보다 높은 값이면 직각 우회전이다, 좌회전과 모든 부분을 같은 방식으로 하는 것이다.

또한 회전 코스로 eilf문을 써서 꺾을 시기를 조정하는데 V[3]이 100보다 작으면 꺾으라는 신호이다.

2. 'T'자형 코스 출입의 차이점

그림 2.

들어갈 때에는 좌차선이 막혀 있고 정면의 차선은 막혀 있지 않다. 우차선 또한 막혀있지 않다. 따라서 우회전을 할 때, L의 값이 잡히지 않거나 매우 큰 값이 나타난다. 나갈 때에는 들어갈 때와 반대로 좌차선과 우차선이 막혀 있지 않고, 정면의 차선이 막혀있어 대칭의 형태이다. 따라서 좌차선과 우차선 중 어떤 쪽으로 회전을 해야 할지 선택을 해야 한다는 점에서 차이가 있다.

2.1 'T'자형 들어갈 때

직선코스를 주행하고서 앞길이 막혀 있지 않아도 우회전을 하는 알고리즘을 작성해야 했다. 처음에는 L[2]와 R[2]값의 변화를 기준으로 하여 작성하였지만, 원하지 않는 구간에서 우회전을 하는 일이 발생하였다. 원하는 진입 구간에서 자동차가 인식하는 값들을 측정해보니, V[4] > V[5] > V[6]인 특징을 나타내고 있었다. 이는 곡선 코스 두 개 중 윗부분을 인식하는 것이었다. 따라서 V[4] > V[5] > V[6] 일 때 L[2] < 321 and R[2] > 319를 만족하면 우회전을 실행하게 했다.

2.2 'T'자형 나갈 때

'T'자 들어가는 코스보다 간단하게 주행시킬 수 있었다. 좌회전과 우회전처럼 앞길이 막혀 있었고, 상황에 따라서 회전 방향이 바뀌지 않았기 때문이다. 원하는 커브구간에서의 값들을 측정하고, 이 코스는 양 옆으로 차선이 연결되었기 때문에 V[3] < 140 일 때 R[2]> 319 and L[2] > 31을 만족하면 'T'자 나가는 코스를 설명하기 충분했고, 우회전을 실행시킬 수 있었다.

Ⅲ. 결론

코스들을 분석한 차선의 특징을 토대로 상황에 알맞은 주행을 할 수 있었지만, 여전히 주행 도중에 여러 if문의범위에 해당이 되는 값이 겹치는 문제가 존재한다. 앞으로 이러한 문제를 해결하기 위해 더 자세한 알고리즘을 작성에 대한 추가 연구를 할 계획이다.

2022년 〈청년공학〉 제6집 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

2021년 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 : 2022년 6월 30일까지, 청소년 공학 리더 담당 교사에게 제출
② 논문 심사 : 2022년 7월 31일까지 논문 심사 및 PDF 출판 완료
③ 출판 : 2022년 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 제6집

인쇄일 2022년 8월 6일
발행일 2022년 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.

NAEK 한국공학한림원

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

95
9 772383 885000
ISSN 2383-885X