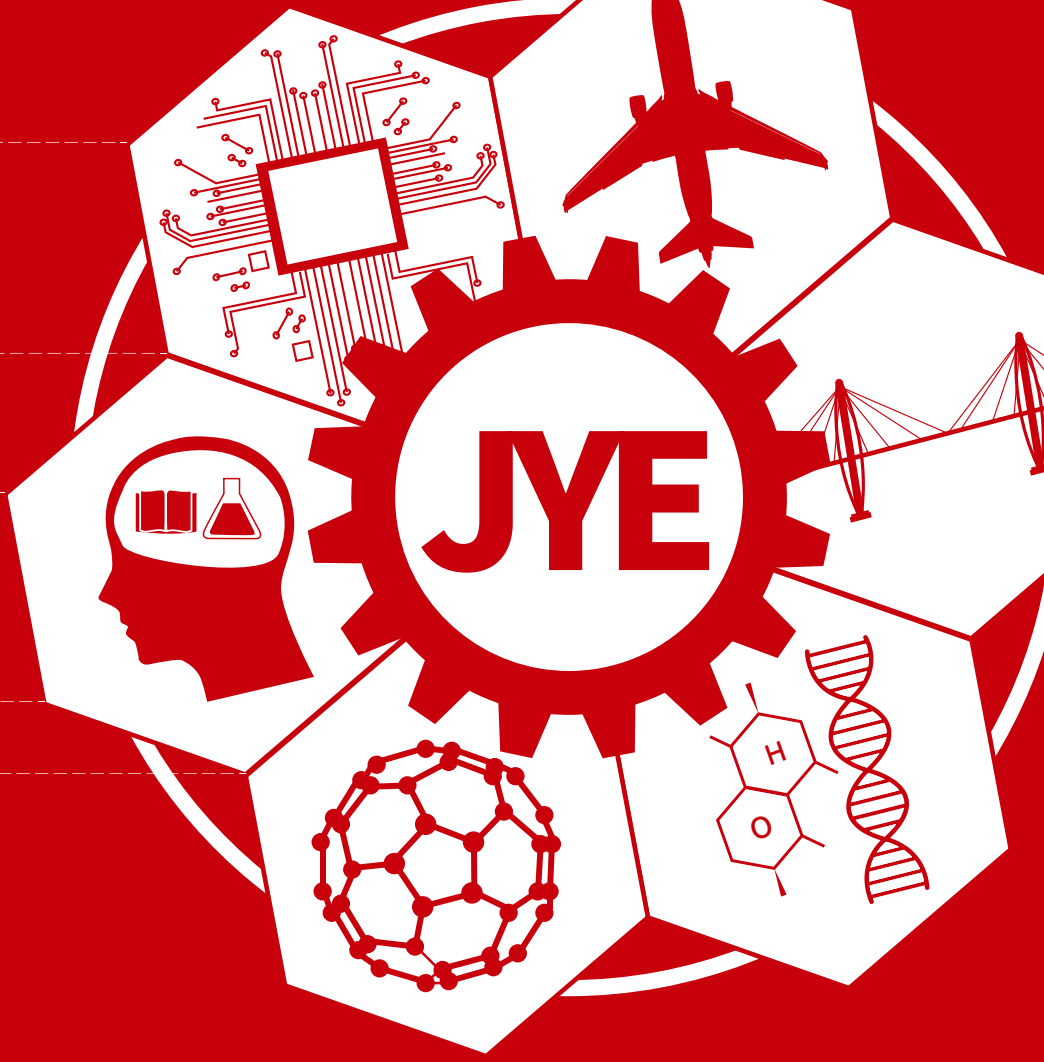


Journal of Youth Engineering

청년공학

제 4 집

—

2020년

청소년 공학리더 안내(클릭)

NAEK 한국공학한림원

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

청년공학

Journal of Youth Engineering

제4집

2020. 09.

NAEK 한국공학한림원
The National Academy of Engineering of Korea

※ 이 논문집은 산업통상자원부의 지원을 받아 발간되었습니다.

차례

발간사

최근 교육과정 개정에 따라 창의 인재 교육을 중시한 융합형 교육과 프로젝트 학습이 활성화되고 대학 입시가 교내 활동 중심의 학생부종합전형(입학사정관 전형)으로 바뀌게 되면서 단위학교에서는 수월성을 추구하는 교과목(수학, 국제, 과학계열의 전문교과목, 과제 연구 등)과 프로젝트 활동이 확산되고 있습니다.

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

한국공학한림원은 공학자(엔지니어)를 꿈꾸는 고등학생들이 공학 논문을 발표할 수 있는 학술지를 만들어 교육 현장의 연구 활동을 장려하고, 학술지에 투고하는 과정을 미리 체험해 보도록 2014년부터 <청년공학>을 펴내고 있었습니다. <청년공학>은 공정한 원칙과 절차에 따라 전문 학술지와 동일한 수준의 논문 심사 절차를 따르고 있습니다. 2020년 제3집부터는 한국공학한림원과 현대모비스가 주관하는 청소년 공학 리더 프로그램의 성과물을 논문으로 펴내고 있습니다.

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

2020년 9월

한국공학한림원 회장

권오경

1

공학 논문

1

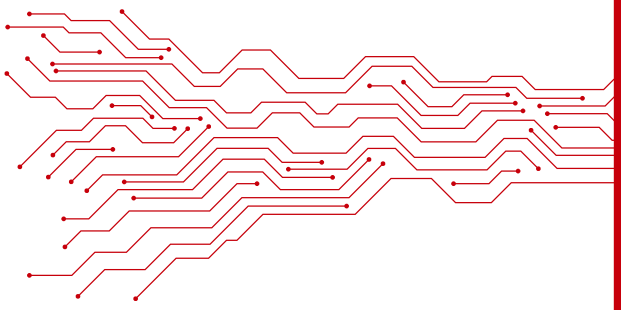
Journal of Youth Engineering

특별 논문

자율 주행 교육용 모형 자동차의 인식 및 주행 능력 개선

서울대학교 문주현, 이상규

※ 이 논문은 2020년 개발한 신형 모형 자주차 개발 연구진에 참여한 청소년 공학리더(하나고등학교) 출신 학생들이 작성한 것입니다.



자율 주행 교육용 모형 자동차의 인식 및 주행 능력 개선

A Model Car for Autonomous Driving Education with enhanced Perception and Maneuvering ability

문주현*, 이상규** Juhyeon Moon, Sanggyu Lee

요약

타이어의 미끄러짐이 적고 조향각의 세밀한 조정 및 전후방 다차로 차선 인식이 가능한 자율 주행 교육용 모형 자동차를 제안한다. 기존의 교육용 모형 자동차는 저속에서도 타이어가 미끄러져 kinematic 모델의 사용이 불가능했다. 또한, 조향부가 불안정해 같은 제어 명령을 입력해도 다른 결과가 나타나서 학생에게 혼란을 주었다. 새로운 모형 자동차는 아커만 조향과 차동 기어를 이용하여 저속에서 타이어가 거의 미끄러지지 않고, 랙 앤 피니언, 베어링을 이용하여 안정적인 조향이 가능하다. 또한, 기존 모형 자동차는 후진 시 뒤쪽을 볼 수 없고, 가장 안쪽 차선만을 13개 점의 형태로 제공해 다차로를 인식하기 어려웠다. 새로운 모형 자동차는 후방에 센서를 추가하고, 보이는 모든 차선마다 점을 제공하여 학생들이 다차로 상황을 분석할 수 있도록 한다. 새로운 모형 자동차를 활용하면 효과적인 코딩 교육이 가능할 것으로 기대된다.

Keywords: 자율 주행, 자동차, 코딩 교육, 차선 인식, 아커만 조향, 차동 기어

I. 서 론

자율 주행 자동차를 활용한 코딩 교육은, 코딩으로 실생활의 문제를 해결하고, 그 과정에서 이론을 배운다는 교육목표를 갖는다.([1]) 따라서 이 목표에 맞는 수업을 위해서는, 여러 개의 구동 모터로 로봇처럼 움직이는 기존의 교육용 하드웨어보다는, 실제 자동차와 유사한 원리로 움직이는 모형 자동차를 사용하는 것이 더욱 적절하다.

그래서 [1]에서는 바퀴가 4개이고 앞바퀴가 조향을, 뒷바퀴가 구동을 담당하는, 실제 자동차와 유사한 모형 자동차를 제작하여 이를 교육에 활용하는 방법을 제시하였다. 그러나 이 모델은 앞바퀴 조향 시스템의 한계와 뒷바퀴의 차동 시스템의 부재로 인해 주행 중에 바퀴가 계속해서 미끄러졌다.([2]) 이에 따라 차량의 움직임을 kinematic 모델([3])로 예측할 수 없어, 학생들이 코딩하는 데에 어려움이 있었다. 더 나아가, 앞바퀴의 유격으로 인해 같은 명령을 입력하더라도 조금씩 다른 주행을 하는 경우가 발생하여, 주행 테스트 이후 프로그램을 수정하고 개선해나가는 과정이 잘 이루어지기 어려웠다. 이외에도 카메라와 센서가 부족하여 코딩에 필요한 정보가 충분히 제공되지 않는 등 기존의 차량은 많은 개선이 필요하다.

따라서 본 연구에서는 아커만 조향 등을 활용한 조향부의 개선 방안을 다루고, 차동 기어와 스텝 모터로 개선된 구동부의 특징을 논한다. 마지막으로 후방 인식 기능 및 다차로 인식 기능의 추가를 언급하고, 결론과 추후 개선점을 논하며 글을 마무리할 것이다

II. 조향부

2.1. 아커만 조향 방식의 적용

기존의 모형 자동차 모델들은 평행사변형 방식의 조향을 사용한다. 즉, 4개의 막대를 관절로 연결하고(마주 보는 두 막대의 길이는 서로 같다), 그 중 조향용 막대를 좌우로 움직임에 따라 사각형이 기울어지는 것을 활용한 방식이다.([4]) 평행사변형 조향 방식에서는 평행사변형의 성질에 의해 두 앞바퀴의 조향각이 같다.

그림 1. 평행사변형 조향

그러나 앞바퀴가 2개인 차량은 윤거(앞바퀴 간의 간격)가 있어서 두 앞바퀴의 조향각이 그림 2와 같이 달라진다. 저속 주행 시 바퀴들이 모두 미끄러지지 않고 구르기 위해서는 바퀴들이 모두 하나의 점을 중심으로 하는 동심원과 접해야 하는데, 그러려면 회전하는 안쪽 앞바퀴가 더 큰 각도로 기울어져야 한다. 평행사변형 조향 방식의 차량은 이를 만족하지 않기 때문에 정확한 조향 경로를 따라 이동하지 못하고 오차를 갖게 된다.

* 문주현(서울대학교, juhyeun38@gmail.com)
* 이상규(서울대학교, sanggyu.developer@gmail.com)

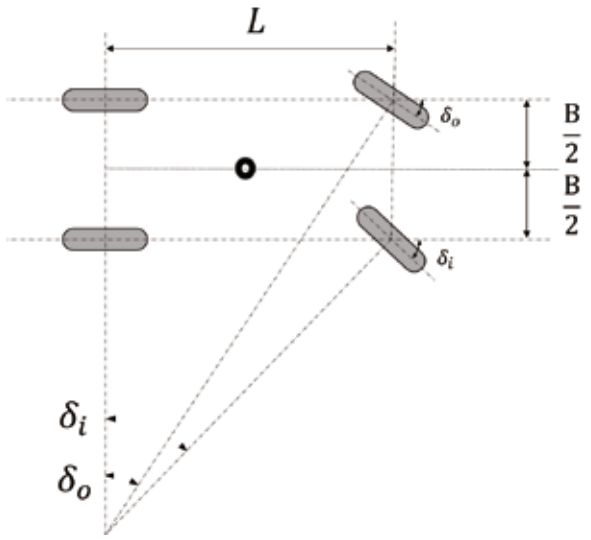


그림 2. 조향 시에 두 앞바퀴의 각도 차이를 나타낸 그림.([2]) 전혀 미끄러짐 없는 이상적인 조향을 하려면 위 그림의 $\cot\delta_o - \cot\delta_i = B/L$을 만족해야 한다. 이는 그림 4의 녹색 선을 그리는 데에 이용된 식이다.

본 연구에서 제안하는 차량은 이런 문제의 개선을 위해 아커만 조향을 사용하였다. 이는 조향용 막대와 앞바퀴 사이에 타이로드와 관절을 추가하여, 두 바퀴의 조향각이 서로 달라지도록 설계한 것이다. 이 차량에 적용한 아커만 조향(그림 3)의 경우 아래(그림 4)와 같이 6% 이내의 오차로 약 250mm의 최소회전반경을 가져, 평행사변형 조향에 비해 미끄러짐이 작고 정밀하다는 것을 확인할 수 있다.

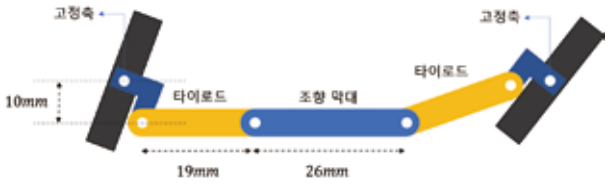


그림 3. 개선된 차량 조향부의 구조. 앞바퀴의 윤거는 70mm이며, 고정축(조향축)은 정확히 바퀴의 중심으로부터 연직 위에 있다. 고정축으로부터 10mm 뒤에 타이로드(19mm)와 조향 막대(26mm)가 일직선으로 연결되어 있고, 조향을 하는 경우에 양쪽 타이로드가 각각 기울어져 두 앞바퀴의 조향각에 차이를 만든다. 위의 각 부품 길이는, 조향 구조가 아커만 조향이 되도록(타이로드의 바깥쪽 관절이 고정축과 두 뒷바퀴의 중심을 이은 선 위에 있도록) 설계하였다.

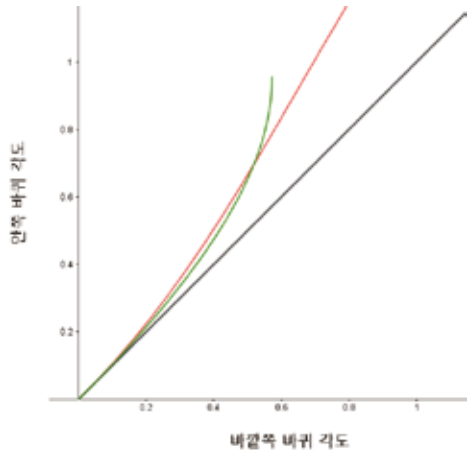


그림 4. 아커만 조향, 그리고 평행사변형 조향의 정확성 비교. 붉은색 선은 바퀴의 미끄러짐이 전혀 없는 이상적인 조향각 비율, 녹색 선은 차량에 적용된 아커만 구조의 조향각 비율, 가운데의 검은색 선은 평행사변형 조향의 조향각 비율이다.

새로운 차량의 경우, 타이어와 트랙의 재질 상 바퀴의 미끄러짐이 거의 없을 것이며, 300mm/s 이하의 저속 주행을 목적으로 설계했기에 위와 같은 정밀 아커만 조향을 채택한 것이다. 다만 실제 자동차는 고속주행을 염두에 두고 있기 때문에, 타이어가 미끄러지는 상황을 고려하여 일부러 오차를 만들거나 심지어는 안티-아커만 구조를 택하기도 한다.([2])

2.2. 랙 앤 피니언 방식의 적용

[1]에서 제안한 차량의 조향부는 서보모터에서 막대가 돌출되어 회전하고, 그 막대에 연결된 볼트가 조향용 막대를 좌우로 미는 방식으로 작동한다. 그러나 이 방식은 볼트가 기울어져서 오차가 발생할 수 있고, 더 나아가 조향 막대와 볼트 사이에 유격이 충분하지 않으면 조향이 불가능하다는 본질적 한계를 안고 있다. 요컨대 정밀한 조향이 불가능한 구조이다.

이 점을 개선하고자 조향 막대의 윗면을 톱니 모양으로 만들어서 래크의 형태를 지니도록 하고, 서보모터에 피니언을 연결하여 이 둘을 결합하였다. 그 결과, 서보모터와 조향 막대가 직접 연결된다는 특성 덕분에 조향의 유격이 줄어들고, 최소 조향 단위도 약 0.42도로 이전에 비해 줄어들었다.

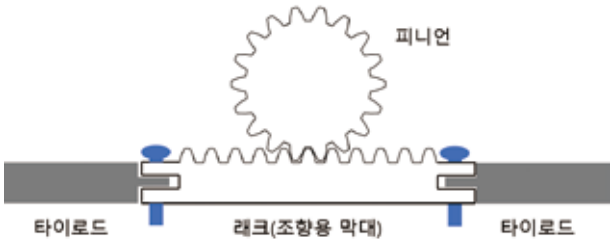


그림 5. 조향부의 랙 앤 피니언 구조. 래크(조향용 막대)와 타이로드를 결합할 때에는 요철 구조에 볼트를 끼워 넣는 방식으로 관절을 만들었다.

2.3. 베어링을 활용한 조향부 정교화

조향부의 정밀한 제어를 위해서는 최종적으로 앞바퀴의 회전축, 지지대 등의 유격이 작고 흔들리지 않아야 한다. 따라서 하나의 바퀴를 한쪽 면에서만 지지하던 기존 방식에서 벗어나, 바퀴 양쪽을 감싸는 지지대를 만들고 이 지지대가 조향축을 기준으로 회전하는 구조를 설계하였다. 이때 차량의 프레임을 1층과 2층으로 나누어 각각 조향축을 지지하고, 축과 프레임 사이에 베어링을 삽입하여 유격을 최소화하였다.

그림 6. 베어링 삽입 사진

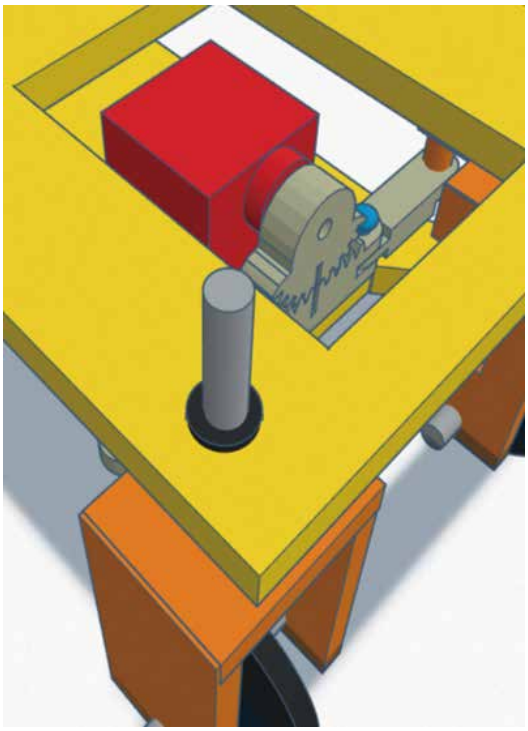


그림 7. 베어링 삽입 그림

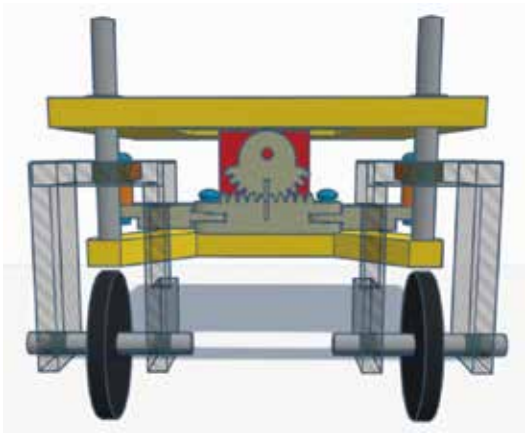


그림 8. 새로운 차량의 조향부 전체 그림

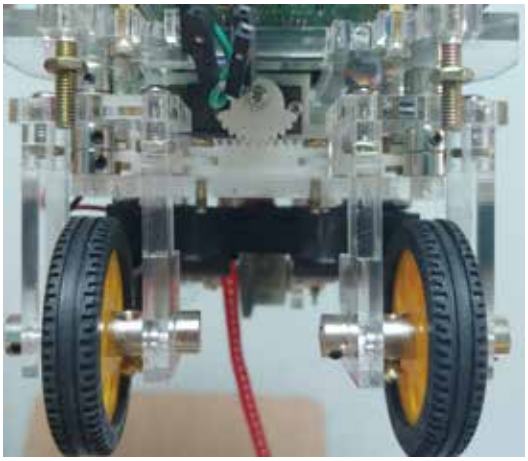


그림 9. 새로운 차량의 조향부 전체 사진

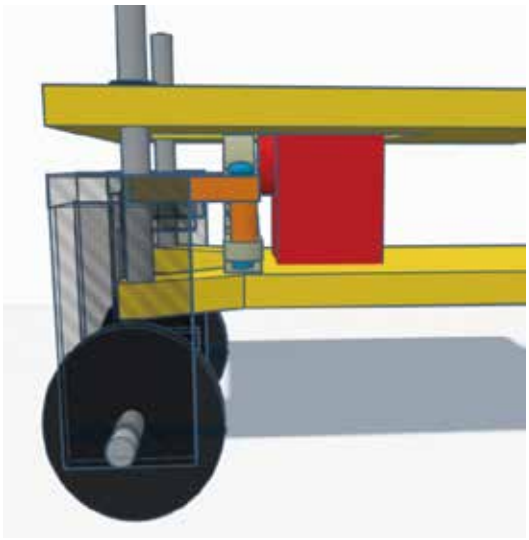


그림 10. 새로운 차량의 조향부 측면 그림

Ⅲ. 구동부

3.1. 구동부 차동 장치의 적용

이전 모형 자동차 모델들에서 모터 동력을 바퀴에 전달한 방식은 크게 두 가지로 구분할 수 있다. 첫째는, 두 뒷바퀴가 고정된 하나의 축과 모터 축 사이에 평기어를 삽입하여 동력을 전달하는 방식이다.([1]) 둘째는, 두 뒷바퀴에 각각 모터를 연결하여 개별적으로 동력을 전달하는 방식이다.

첫 번째 방식을 활용하면, 두 뒷바퀴가 항상 같은 각속도로 회전한다. 이 때문에 커브 길을 지날 때 두 바퀴 중 하나가 미끄러지면서 구르는 현상이 발생한다.

두 번째 방식을 활용하는 차량은 두 뒷바퀴의 속도를 각각 다르게 제어할 수 있다. 그러나, 트랙의 상황과 주행 방식에 따라서 유격과 미끄러짐으로부터 완전히 자유로운 조향과 차량 운행은 불가능하므로, 차동의 양을 계산하여 각 모터를 따로 제어하는 것은 비현실적이다.

이 문제점들을 해결하는 방안으로, 실제 차량에서는 차동 기어를 활용하고 있다. 모터와 바퀴 축 사이에 차동 기어를 사용하면, 두 뒷바퀴가 서로 다른 각속도로 운동할 수 있다. 본 연구에서는 RC 자동차에 쓰이는 차동 부품을 그림 12와 같이 도입하였고, 이에 따라 커브 길을 운행할 때에도 뒷바퀴가 미끄러지지 않고 구를 수 있다.

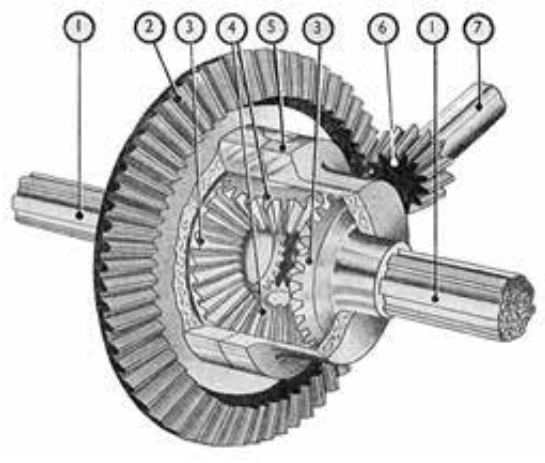


그림 11. 차동 기어의 세부구조([5])

① 좌우 뒷바퀴 샤프트
② 베벨기어 1
③ 베벨기어 2 (좌우 샤프트에 연결된 베벨)
④ 베벨기어 3 (차동용 자유회전 베벨)
⑤ 베벨기어 1과 3의 연결
⑥ 베벨기어 4 (모터의 동력을 전달)
⑦ 모터 샤프트

그림 12. 개선 차량 하부에 도입된 차동 장치. 구동 모터에 3d 프린팅 베벨기어(그림 11의 ⑥)를 장착하여 차동 기어의 베벨기어(그림 11의 ②)와 맞물리도록 하였다.

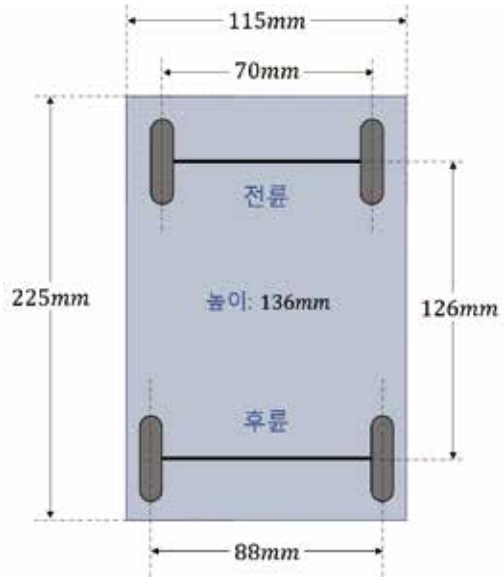


그림 13. 신형 차량의 세부 길이 정보

3.2. 스텝 모터의 이용

모형 자동차는 배터리를 이용하므로 DC 모터, 스텝 모터 등을 이용해 구동할 수 있다. 본 연구에서는 스텝 모터를 사용하였다. DC 모터의 경우 속도 제어를 위해서는 모터의 기전력 등을 바탕으로 모터의 속도를 측정하고, 이를 반영하여 전압을 조절하는 Closed-Loop 시스템이 필요하다.([6]) Closed-Loop 시스템은 Open-Loop 시스템에 비해 만들기 어렵고, PID 제어를 이용할 경우 여러 개의 계수 조정이 필요하다.([7]) 하지만 스텝 모터는 신호를 보내 위치를 정확하게 제어할 수 있도록 설계되었기 때문에 피드백 없이 Open-Loop 시스템으로 제어할 수 있다.([8]) 따라서 스텝 모터와 함께 정확한 시계를 이용하면 쉽게 모터의 속도를 제어할 수 있다.

그러나 라즈베리 파이 4B([11]) 내부의 시계와 스텝 모터 17HD5003, A4988 스텝 모터 드라이버를 이용하여 속도를 제어했을 때 모터가 매우 뜨거워지는 문제와 200mm/s 미만의 속도로 주행 시 모터에서 소음이 발생하는 문제가 있었다. 이를 3.3, 3.4에서 다룬다.

3.3. 스텝 모터의 발열

스텝 모터가 정지해 있을 때도 모터에 디텐트 토크를 만들기 위해 전류가 흐르는데, 모터가 일을 하지 않으므로 이 전력은 열에너지로 전환된다. 따라서 전원을 켠 상태로 정지해 있을 때 모터가 점점 뜨거워지는 문제가 발생했다.

모터 온도를 낮추기 위해 정지 상태에서는 A4988 드라이버([9])의 SLEEP 핀 또는 ENABLE 핀에 신호를 보내 모터에 전류가 흐르지 않도록 했다.

3.4. 마이크로 스테핑

본 연구에서 사용한 스텝 모터 17HD5003은 1.8°의 분해능을 갖고 있지만, 저속으로 작동시킬 때 모터에서 소음이 발생하였다. 스텝 모터가 저속으로 회전할 때 공명이 발생하고, 이를 마이크로 스테핑을 이용하여 개선할 수 있음이 알려져 있다.([10]) 따라서 A4988 드라이버([9])의 마이크로 스테핑 기능을 이용하여 전류의 세기 변화를 2, 4, 8, 16단계로 나누어 부드럽게 회전할 수 있도록 하였다.

Ⅳ. 인식부

4.1. 후방 카메라와 거리 센서

[1]에서 제작한 모형 자동차는 전진과 후진이 모두 가능했지만, 카메라와 거리 센서가 앞쪽에만 있어서 후진할 때 판단이 어려웠다. 따라서 뒤쪽에도 카메라와 거리 센서를 추가하여 후진할 때 상황을 파악할 수 있도록 했다.

본 연구에서 사용한 SoC인 라즈베리 파이 4B([11])는 MIPI CSI 카메라 포트가 1개이기 때문에 후방 카메라로는 MIPI CSI 카메라 대신 USB 웹캠을 연결하였다.

거리 센서는 [1]과 마찬가지로 VL53L0X([12])를 이용했다. 이 센서는 I2C를 이용하여 라즈베리 파이와 통신하는데, 라즈베리 파이 4B([11])는 SDA, SCL 포트가 각 1개이기 때문에 하나의 회선으로 두 센서가 통신해야 했다. 따라서 VL53L0X([12])의 XSHUT 핀을 이용하여 하나의 센서를 먼저 켜고, I2C 주소를 기본값인 0x29에서 0x2C로 바꾸어 하나의 회선에서 두 센서와 각각 통신할 수 있도록 했다.

그림 14. 후방 카메라와 거리 센서

4.2. 차선 인식

모형 자동차는 바닥이 검은색이고 차선이 흰색인 평면 도로에서 운동한다고 가정하고, 2차로 이상인 경우도 고려하여 이미지에서 각 차선 위의 점들의 좌표를 구하는 알고리즘을 설계했다.

4.2.1. 차선 후보 구하기

바닥과 차선의 경계에서 검은색과 흰색이 대비되므로 픽셀의 밝기가 크게 변한다. 캐니 에지 검출기([13])를 이용하여 차선과 도로의 경계를 찾아, 이를 차선의 후보로 두었다. 특히 검은색과 흰색이 대비되도록 HSL 색상모델의 L 채널(그림 15)에 대해 캐니 에지 검출기를 적용했다.

그림 15. 이미지의 L 채널

그림 16. 이미지에서 검출된 에지

4.2.2. 평면도 변환

4.2.1에서 검출한 에지는 차선뿐만 아니라 도로가 아닌 영역에서의 에지도 검출하고, 원근 효과에 의해 평행한 차선이 평행하지 않게 보인다. 따라서 차선을 찾기 위해, 아래 동차 변환 행렬을 이용하여 500x300의 평면도(그림 17)로 변환했다.([14])

$$T_i^g = \begin{pmatrix} \frac{1}{2} & 0 & 250 \\ 0 & -\frac{1}{2} & 350 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} -\frac{h}{f_u} & 0 & \frac{h}{f_u}c_u \\ 0 & 0 & -h \\ 0 & -\frac{1}{f_v} & \frac{c_v}{f_v} \end{pmatrix}$$

카메라의 지면으로부터의 높이 $h$mm는 자를 이용하여 측정했다. 본 연구에 사용한 모형 자동차의 카메라는 이미지 평면이 지면과 수직을 이루도록 설치했다. 즉, [14]에서 정의한 roll, pitch, yaw 각도가 모두 0이다. 가로, 세로 초점거리 $f_u$mm, $f_u$mm와 카메라 중심의 좌표 c_u, c_u는 체커보드 그림을 여러 방향과 각도에서 촬영하여 얻을 수 있다.([15, 16]) 변환된 평면도에서 1픽셀은 지면에서 2mm에 해당한다.

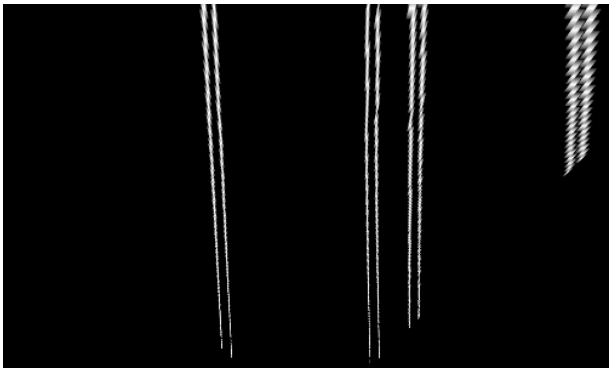


그림 17. 평면도

4.2.3. 차선 탐색

모형 자동차는 정상적으로 주행하는 동안 차선과 거의 평행하므로, 차선의 곡률이 작다면 평면도에서 차선은 평면도의 세로 방향(y 방향)과 거의 평행하다.

차선이 거의 평행하면 단순화된 Hough 변환을 이용하여 차선의 개수와 위치를 구할 수 있다.([14]) 단순화된 Hough 변환([14])은 평면도 각 열의 픽셀값의 합을 구한다. 이에 M=20인 hanning 윈도우([17])와의 합성 곱을 적용하여 노이즈를 제거한 히스토그램(그림 18)을 만들고 극대점을 찾으면 극대점의 개수가 차선의 개수이고, 극대점의 위치가 차선의 대략적인 위치이다. 그림 17에서 4개의 차선이 그림 18에서 4개의 극대점으로 나타난 것을 확인할 수 있다.

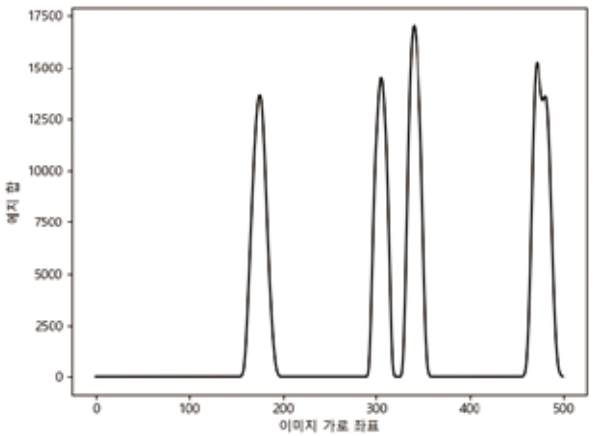


그림 18. 에지 히스토그램

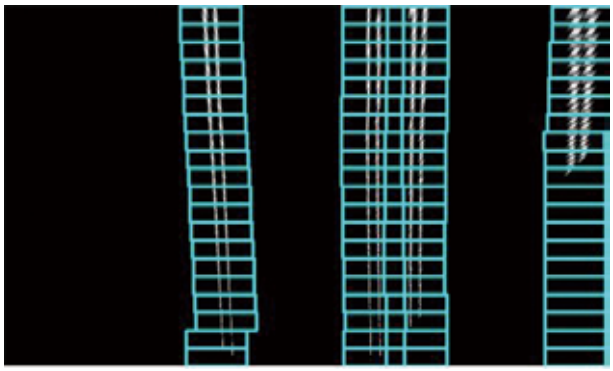


그림 19. 사각형 윈도우 탐색

차선은 [18]에서 제안한 것과 같이 50x15 크기의 사각형 윈도우를 위로 15만큼씩 이동하며 에지가 가운데에 오도록 하여 탐색하였다. 먼저 히스토그램에서 구한 극대점의 x좌표 x_0에 대해 $(x_0-25, 300-15)$, $(x_0+25, 300)$을 두 꼭짓점으로 하고 각 변이 x축, y축에 평행한 직사각형 S_0을 만들었다. 그 다음부터 직사각형 S_i내의 에지의 좌표 평균 x_{i+1}을 구하고, $(x_{i+1}-25, 300-15\times(i+2))$, $(x_0+25, 300-15\times(i+1))$을 두 꼭짓점으로 하고 각 변이 x축, y축에 평행한 직사각형 S_{i+1}을 만드는 것을 반복한다. $(i=0,1,2,\cdots,19)$ 만약 직사각형 S_i내에 에지가 30개보다 $x_{i+1}=x_i+(x_i-x_{i-1})$ 적다면 으로 두고 계속해서 탐색한다. $(x_{-1}:=x_0)$

그림 19에서 각 차선마다 청록색 사각형 20개가 그려진 것을 확인할 수 있다.

4.2.4. 차선 정보의 가공

4.2.3에서 구한 직사각형 S_i내부의 에지를 차선에 포함되는 점이라고 볼 수 있다. 이 점들에 가장 가까운 $x=p(y)$ 형태의 다항함수([18]), 3차 스플라인([19]), B-스플라인([20]) 등을 구하여 차선을 매개화된 곡선으로 나타내고 그로부터 곡률 등을 구하여 이용할 수 있다. 하지만 각각의 도로 모델은 모든 종류의 곡선을 나타낼 수는 없고, 코딩을 처음 배우는 사람이 다루기 쉽지 않다.

따라서 직사각형 S_i 중 내부에 에지가 30개 이상인 i에 대해 $\left(x_{i+1}, 300-15\times\frac{2i+1}{2}\right)$을 평면도에서 차선 좌표라 하여 좌표를 다룰 수 있도록 했다. 동차 변환 행렬

$$T_g^i=\begin{pmatrix} f_u & c_u & 0 \\ 0 & c_v & hf_v \\ 0 & 1 & 0 \end{pmatrix}\times\begin{pmatrix} 2 & 0 & -500 \\ 0 & -2 & 700 \\ 0 & 0 & 1 \end{pmatrix}$$

을 이용하면 평면도의 좌표를 이미지에서의 좌표로 변환할 수 있다.([14]) 그림 20은 평면도에서 차선 좌표의 위치를, 그림 21은 이미지에서 차선 좌표의 위치를 보여준다.

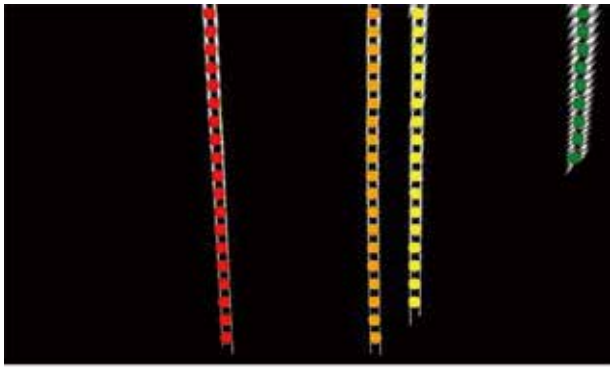


그림 20. 평면도에서 차선 점의 위치

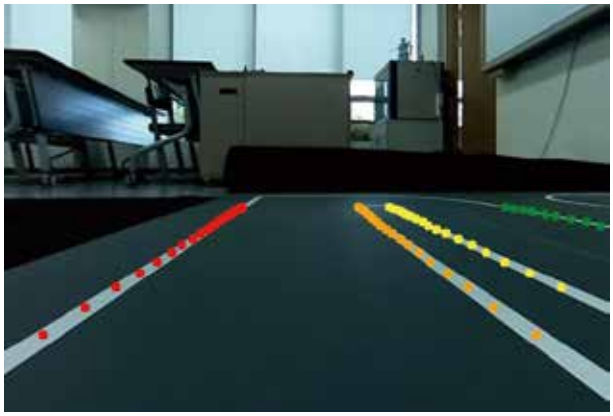


그림 21. 이미지에서 차선 점의 위치

Ⅴ. 결론 및 제언

5.1. 결론

본 연구에서 새롭게 제안한 자율 주행 교육용 모형 자동차는 더욱 정밀한 조향장치와 다차로 인식 기능을 제공한다. 아커만 조향과 차동 기어의 도입으로 커브 길에서도 바퀴의 미끄러짐이 거의 없으며, 샤프트와 프레임의 유격을 베어링으로 해소하여 작은 스케일의 명령(최소 0.42도의 조향)에 대해서도 정밀한 반응성을 가지게 되었다. 또한, 전후방의 두 카메라가 각각 3개 이상의 차선을 동시에 인식할 수 있도록 프로그래밍하였다. 이렇게 개선된 차량을 활용하여 자율 주행 교육이 더욱 효과적으로 이루어지리라 기대한다.

5.2. 제언

차량의 정밀도를 높인 대신에, 새로운 기계적 한계가 발생하여 특정 환경에서 주행할 때 어려움이 있었다. 예컨대 구동부에 장착한 스테핑 모터의 발열 문제를 해결하기 위해 디턴트 토크를 제거하였는데, 이 때문에 차량이 정지해 있는 동안 앞뒤로 밀릴 수 있고 경사로에서 정지할 수 없다. 그뿐만 아니라 울퉁불퉁한 표면 위에서 마이크로 스테핑을 이용하여 저속으로 주행할 때 구동 토크가 충분하지 않아서 앞으로 나아가지 못하는 경우가 발생했고, 한쪽 뒷바퀴가 바닥에서 들리면 차동 기어의 작동원리에 따라 공중에 뜬 바퀴만 계속 돌아가는 현상도 관찰하였다.

다차로 인식 기능 또한 상황에 따라 부정확한 결과를 내놓기도 하였다. 앞에서 언급했듯이 도로의 곡률이 큰 경우에는 차선 인식이 어려웠으며, 교차로처럼 차선이 많은 도로에서 여러 개의 차선을 하나로 인식하는 등의 오류도 발생했다.

추후 이 차량 모델을 개선할 시에는 모터제어 기능을 유지하면서 발열 문제를 해결할 수 있는 다른 방안을 모색해야 할 것이며, 다차로 인식 기능의 정확성을 높여 더욱 다채로운 트랙에서 경로계획을 할 수 있도록 개선해야 할 것이다.

Acknowledgements

차량의 형태와 기능에 관해 조언해 주신 하나고등학교 정형식, 이효근 선생님, 인천대학교 김평원 교수님께 감사드립니다. 또한, 시제품 제작을 위한 공간과 도구를 제공해 주신 하나고등학교와 로보티즈 메이커 스페이스에 감사의 말씀을 전합니다.

이 논문은 2020년형 신형 자주차 연구 개발비(인천대학교 산학협력단) 지원을 통해 작성되었습니다.

참고문헌

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

[2] Wong, Jo Yung. Theory of ground vehicles. John Wiley & Sons, 2008.

[3] Rajamani, Rajesh. Vehicle dynamics and control. Springer Science & Business Media, 2011.

[4] Halderman, James D, Automotive Chassis Systems, 2010

[5] McCarron, Bird, Manual of Driving and Maintenance for Mechanical Vehicles, 1939

[6] Bhagwat, Pradeep, et al. "Direct current motor speed control." U.S. Patent No. 4,893,067. 9 Jan. 1990.

[7] Huang, Guoshing, and Shuocheng Lee. "PC-based PID speed control in DC motor." 2008 International Conference on Audio, Language and Image Processing. IEEE, 2008.

[8] Zribi, M., and J. Chiasson. "Position control of a PM stepper motor by exact linearization." IEEE Transactions on automatic control 36.5 (1991): 620-625.

[9] Allegro MicroSystems. "A4988. DMOS Microstepping Driver with Translator and Overcurrent Protection." online].https://www.allegromicro.com/en/products/motor-drivers/brush-dc-motor-drivers/a4988 (2020)

[10] Baluta, Gheorghe. "Microstepping mode for stepper motor control." 2007 International Symposium on Signals, Circuits and Systems. Vol. 2. IEEE, 2007.

[11] Raspberry Pi. "Raspberry Pi 4 Model B specifications." online].https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/ (2020)

[12] STMicroelectronics. “VL53L0X. World's smallest Time-of-Flight (ToF) ranging sensor.” online].https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html

[13] Canny, John. "A computational approach to edge detection." IEEE Transactions on pattern analysis and machine intelligence 6 (1986): 679-698.

[14] Aly, Mohamed. "Real time detection of lane markers in urban streets." 2008 IEEE Intelligent Vehicles Symposium. IEEE, 2008.

[15] Zhang, Zhengyou. "A flexible new technique for camera calibration." IEEE Transactions on pattern analysis and machine intelligence 22.11 (2000): 1330-1334.

[16] Wang, Y. M., Y. Li, and J. B. Zheng. "A camera calibration technique based on OpenCV." The 3rd International Conference on Information Sciences and Interaction Sciences. IEEE, 2010.

[17] Oppenheim, Alan V., John R. Buck, and Ronald W. Schafer. Discrete-time signal processing. Vol. 2. Upper Saddle River, NJ: Prentice Hall, 2001.

[18] McLeod, Haidyn. “Driving Lane Detection.” online].https://www.haidynmcleod.com/driving-lane-detection (2017)

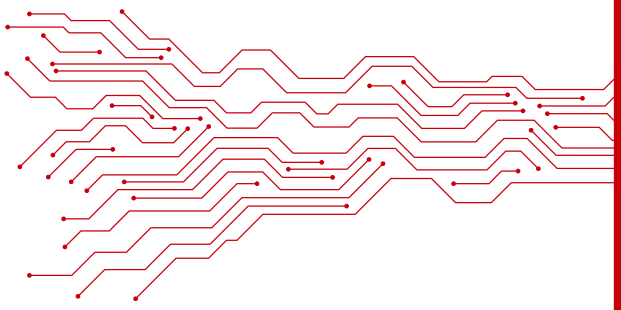
[19] Wang, Yue, Dinggang Shen, and Eam Khwang Teoh. "Lane detection using spline model." Pattern Recognition Letters 21.8 (2000): 677-689.

[20] Wang, Yue, Eam Khwang Teoh, and Dinggang Shen. "Lane detection and tracking using B-Snake." Image and Vision computing 22.4 (2004): 269-280.

최우수 논문

01 자율 주행 자동차의 안정적 주행을 위한 PD 제어 알고리즘 연구

선덕고등학교 김대환



자율 주행 자동차의 안정적 주행을 위한 PD 제어 알고리즘 연구

A Study on the PD Control Algorithm for the Stable Driving of Self-driving Cars

김 대 환* Kim Dae Hwan

요약

본 논문에서는 자율 주행 자동차의 주행알고리즘에 비례미분(PD) 제어 기법을 적용하여 차선과의 거리를 일정하게 유지하며, 안정적으로 주행할 수 있도록 개선하는 방법을 연구하였다. 기존의 자율 주행 알고리즘은 좌측, 우측 차선의 거리를 조건에 따라 몇 가지 경우(Case)로 지정하고 각각의 경우에 대해 전진, 좌회전, 우회전, 후진 등을 하도록 하였다. 그러나, 조건의 경계 부분에서는 자동차의 제어가 부드럽지 못하고, 차선을 이탈하는 경우가 종종 발생한다. 본 연구에서는 비례제어(P제어)를 통해 차선과의 이탈 정도에 비례하여 자동차의 회전 각도를 계산하며, 미분제어(D제어)로 자동차 앞쪽의 가까운 거리와 먼 거리의 미분값을 이용하여 차선의 급격한 변화에 빠르게 대응할 수 있도록 하였다. 본 연구에서는 비례제어와 미분제어를 함께 사용하여 직선 구간이나 S자 커브 구간 등 차선의 변화가 적은 부분에서는 비례제어 값이 직각 구간, M자 구간과 같이 차선이 급격하게 변하는 구간에서는 미분제어 값이 커지도록 하여 다양한 차선 상황에서도 부드러운 주행이 가능하도록 하였다.

Keywords: 자율 주행 자동차, 자율 주행 알고리즘, PD제어, 비례제어, 미분제어

I. 서 론

자율 주행 자동차는 컴퓨터가 자동차의 위치와 주변 정보를 수집하여 스스로 판단하고 주행 경로를 결정하는 자동차이다. 안전한 자율 주행을 위해서는 오류 없는 정보 수집이 필요하며, 수집된 정보에 대한 정확한 분석과 자동차 제어가 필요하다. 특히 이런 과정은 빠르게 이동하는 자동차의 특성에 맞게 실시간으로 처리되어야 할 필요가 있다.

본 연구에서 사용된 자율 주행 자동차는 카메라에서 인식한 원본 영상 이미지를 Canny 이미지로 전환하고 Hough Transform 과정을 거쳐 차선의 위치와 진행 방향을 인식하게 된다. 이 연구에서는 다루는 것은 영상을 통해 수집된 Grid 데이터를 수학적으로 계산하여 차선의 위치를 계산하고, 자동차의 목푯값을 설정하여 자동차의 현재 위치와 비교하면서 자동차를 제어하는 방법에 관한 것이다.

자동차를 제어하는 방법으로는 차선의 위치를 여러 가지 Case로 구분하여 주행명령을 내리는 방법과 PD(비례미분)제어방법을 통한 주행명령의 결과를 비교하여 PD제어를 통한 주행 알고리즘의 안정성을 개선하고자 한다.

* 김대환(선덕고등학교, kingdaehwan@naver.com)

II. 본론

2.1 카메라 영상을 이용한 차선 인식

자율 주행차는 카메라 영상을 분석하여 차선을 인식한다. 카메라의 원본 영상은 사람이 주변을 보는 것과 같이 복잡한 물체의 형태와 색깔로 표현되므로, 여기서 직접 차선의 위치를 찾아낼 수 없어 Canny Edge Detector를 이용한다.

그림 1. 2019년형 자주차의 카메라와 라이다 센서

Canny Image 원리는 원본 영상이 많은 점들과 각각의 색깔 정보를 가지고 있는데, 한점과 옆에 있는 점들의 정보를 비교하여 갑작스러운 변화가 나타나는 부분을 선으로 표시하면 물체의 경계면으로 나타나는 것이다. 예를 들어 자동차 도로의 경우라면 계속 검은색 도로가 이어지다가 갑작스럽게 흰색 차선으로 변하는 부분을 찾아낼 수 있다. 이렇게 갑작스럽게 변하는 부분을 선으로 이어주면 차선의 모양이 된다. 그러나, 자동차 제어를 위해서는 차선의 정보를 특정한 숫자로 표현할 수 있어야 한다. 따라서 Canny 이미지에 3개 수평선과 7개 수직선을 긋고 직선이 차선과 만나는 점을 행렬로 표현하면 컴퓨터가 계산할 수 있는 숫자로 나타낼 수 있다. 이 데이터를 Grid

데이터라고 부른다. 이렇게 카메라 영상 속의 차선을 더하기, 빼기, 곱하기, 나누기가 가능한 숫자로 표시되면 일정한 알고리즘을 통해 자동차를 제어할 수 있게 된다.

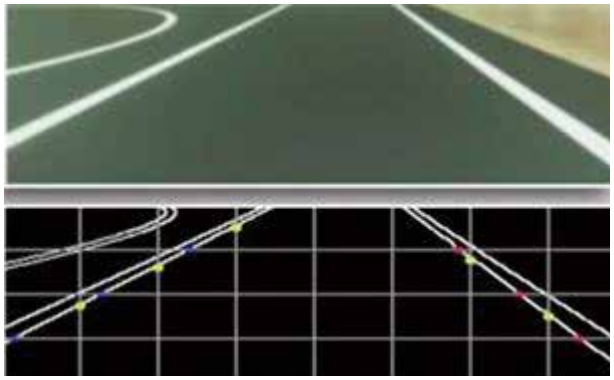


그림 2. 원본 이미지와 Canny 이미지

2.2 자동 제어

자동제어는 사람이 관여하지 않고 기계 스스로 목푯값과 실제값을 비교하여 실제값이 목푯값과 같아지도록 조정해 주는 것이다. 이렇게 시스템의 출력을 Feedback 하여 목푯값과 비교하고 다시 출력값을 조정하는 것을 Feedback 제어라고 한다. 대부분 자동제어는 피드백 제어의 한 종류이다. 아래에서 피드백제어의 종류를 살펴보자.

그림 3. 피드백 제어

2.2.1 단순 On/Off 제어

가장 간단한 제어방법으로 실제값이 목푯값보다 작으면 출력을 높이고, 실제값이 목푯값보다 크면 출력을 멈춘다. 제어의 조작량이 0% 아니면 100% 이기 때문에 실제값이 목푯값 범위에서 항상 오르락내리락 반복하게 되어 항상 큰 오차를 발생한다. 자율 주행 자동차의 경우 차선의 위치를 여러 가지 Case로 구분하여 목푯값을 다양하게 설정하여 제어에 사용할 수 있으나, Case별 경계 부근에서 제어가 부드럽지 못하고 차선이 이탈될 수 있어 좋은 자동제어 방법은 아니다.

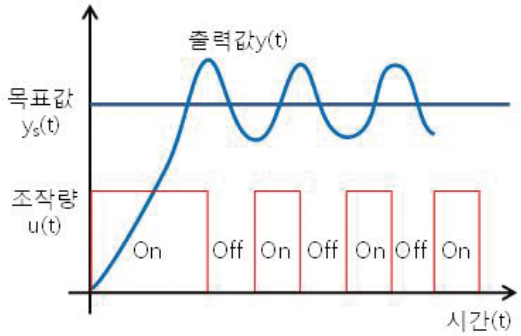


그림 4. 단순 On/Off 제어

2.2.2 비율제어(Proportional Control, P제어)

P제어는 출력값(실제값)과 목푯값의 차이에 비례하여 제어량을 변화시키는 방법이다. 목표값 (y(t))에서 출력값 (y(t))를 뺀 값에 제어 Gain값 kc를 곱하여 조작량을 결정하므로 두값의 차이가 클때는 조작량이 크고 빠르게 목푯값에 접근하지만, 차이가 작아지면 조작량이 줄어들어 영원히 목푯값과 같아지지 않을 수 있다.

$$P\text{제어 조작량 } u(t) = k_p \times (y_s(t) - y(t)) \quad (1)$$

(kc = 제어 Gain값, ys(t) = 목푯값, y(t) = Feedback값 (실제값))

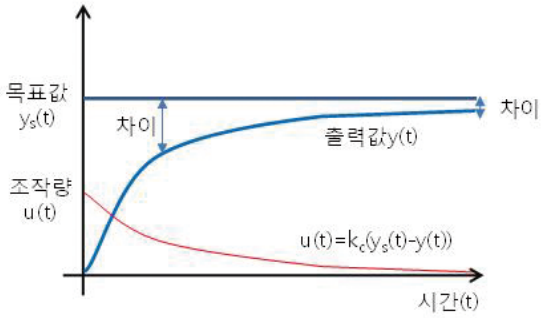


그림 5. 비율제어(P제어)

2.2.3 미분제어(Differntial Control, D제어)

자율 주행과 같은 시스템에서는 목푯값이 아주 빠르게 변하므로 자동제어도 아주 빠르게 동작해야 한다. 실제값이 목푯값과 같아진 후 목푯값이 빠르게 바뀌는 경우라도 둘 간의 차이가 적다면 P제어는 시간이 오래 걸린다. 반면 D제어는 목푯값과 실제값의 차이를 시간으로 미분하면 목푯값이 빠르게 변할수록 조작량이 더 큰 값을 출력하여 빠르게 시스템을 제어할 수 있다.

$$D\text{제어 조작량 } u(t) = k_d \times \frac{d((y_s(t) - y(t))}{dt} \quad (2)$$

(kd = D제어 Gain값, ys(t) = 목푯값, y(t) = Feedback값 (실제값))

위에서 살펴본 자동제어 방법은 각각의 장단점이 있어 단독으로 사용하지 않고 비례제어(P제어), 미분제어(D제어), 적분제어(I제어)를 조합하여 PD제어, PI제어, PID제어와 같이 함께 사용한다. 본 연구에서는 PD제어를 사용하여 제어하고자 한다.

$$PD\text{제어 } u(t) = k_p(y_s(t) - y(t)) + k_d\frac{d(y_s(t) - y(t))}{dt} \quad (3)$$

P 제어(비율제어+D제어(미분제어)

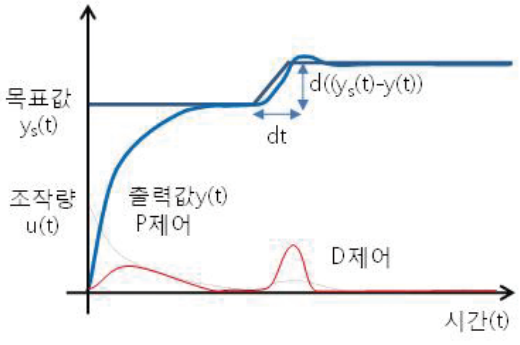


그림 6. 비율미분제어(PD제어)

2.3 자율 주행 자동차 회전분석

자동차가 주변에 충돌하지 않고 안전하게 회전하기 위해서는 자동차 앞쪽과 회전 방향에 최소한의 공간이 필요하다. 이때 앞바퀴의 회전각도와 축간거리에 따라 달라지는데, 본 연구에 적용한 자율 주행차를 시험한 결과 자동차 앞쪽 끝을 기준으로 앞쪽방향 165mm, 회전방향 355mm 이상의 공간이 필요한 것으로 확인되었다. 이것으로 자율 주행 자동차가 앞쪽 라인을 확인하였을 때 늦어도 165mm까지 접근하기 전에 자동차를 회전시켜야 한다는 것을 알 수 있었다.

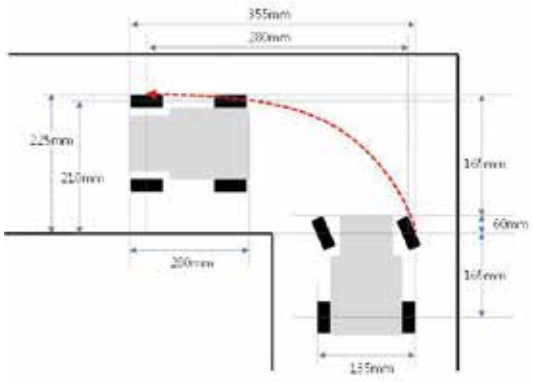


그림 7. 자동차 회전공간

2.4 자율 주행 비례미분 제어

주행을 결정하는 알고리즘은 Grid 데이터를 수학적으로 계산하여 결정하게 된다. 아래 그림은 비례미분제어 알고리즘을 설명하기 위해 편집한 그림이다.

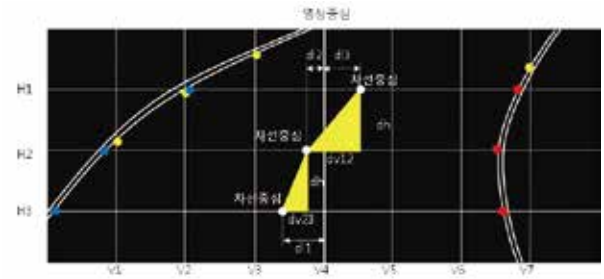


그림 8. 비례 미분제어 분석

2.4.1 비례제어

자주차의 중심은 영상의 중심인 V4이며, H1라인에서 차선의 중심은 H1LD의 x값과 H1RD의 x의 값의 평균값으로 구할 수 있다.

$$centerPoint\_X = \frac{(320 - H1LD + H1RD)}{2} \quad (4)$$

같은 방법으로 H2, H3라인의 차선 중심도 구할 수 있다. 이때, H2라인의 차선 중심과 영상 중심의 차는 그림8에서 dl2 가 되는데, 이 길이가 길수록 자동차가 차선에서 많이 벗어난 것이므로 이 길이를 비례제어에 사용할 것이다.

$$P제어=\left(160 - \frac{(320 - H1LD + H1RD)}{2}\right) \times K_p \quad (5)$$

여기서 Kp 는 비례제어의 비중을 얼마나 크게 할 것인가를 결정하는 Gain값으로 여러 시험을 통해 가장 적당한 값을 사용하면 된다. 본 연구에서는 0.5 값을 적용하였다.

2.4.2 미분제어

영상에서 자동차는 H3 -> H2 ->H1 방향으로 이동해 갈 것이므로, H3, H2, H1 라인에서 차선 중심 차이(dl), 거리 차이(dv)이라고 하면 $\frac{dv}{dh}$로 미분하면 차선이 다음 단계에 얼마나 급하게 변할지 예측할 수 있다. 그림에서 H3~H2 구간의 미분값보다 H2~H1 구간에서 미분값이 크므로 다음 단계에서 차선이 더 급하게 회전할 것이라고 예측할 수 있다. 이 미분값을 미분제어에 사용할 것이다.

$$D제어=\frac{dv}{dh} \times K_d \quad (6)$$

여기서 Kd 는 미분제어의 비중을 얼마나 크게 할 것인가를 결정하는 Gain값으로 여러 시험을 통해 가장 적당한 값을 사용하면 된다. 본 연구에서는 2 값을 적용하였다.

2.5 주행 알고리즘 코딩

앞에서 수학적으로 계산한 비례미분(PD)제어 값을 파이썬 코딩을 통해 자율 주행 자동차를 제어하도록 프로그래밍 하였다.

자율 주행 자동차의 제어명령은 총 6 Byte로 구성되며, 이중 조향 3 Byte 값을 변화시키면 자주차의 앞바퀴 방향을 제어할 수 있다.

시작명령 (1 Byte)	이동(DC 모터) (1 Byte)	조향(서보 모터) (3 Byte)	종료명령 (1 Byte)
S	0 : 정지 1 : 전진 2 : 후진 3 : 개발모드	110 : 최대 왼쪽 ~ 150 : 중립 ~ 190 : 최대 오른쪽	E

그림 9. 자주차 제어명령의 구조

PD제어 결과값을 자주차의 앞바퀴 방향을 제어할 조작량으로 사용할 것이므로, PD제어 값을 110 ~ 190 범위의 숫자로 만들어 줘야 한다.

이때 150을 입력하면 자동차는 전진하고, 150보다 작고 110보다 큰 범위에서는 자동차가 왼쪽으로 회전하게 된다. 이때 값이 작을수록 자동차의 좌회전 각도가 커진다. 반대로 150보다 크고 190보다 작은 범위에서는 자동차가 오른쪽으로 회전하게 된다. 이때 값이 클수록 자동차의 우회전 각도를 크게 할 수 있다. 따라서 PD제어 조작량을 제어명령으로 사용하려면, 중간값이 150이고 왼쪽 -40, 오른쪽 +40 범위의 3자리 숫자로 표현되도록 PD제어 조작량의 출력값을 조정하였다.

$$조향값=150+\left(160-\frac{(320-H2LD+H2RD)}{2}\right)\times 0.5+\frac{dv}{dh}\times 2 \quad (7)$$

이번 실험에서는 Kp 값을 0.5, Kd 값을 2로 적용하였다. 주행차선의 조건에 따라 적당한 값으로 변경할 수 있다.

```
if right and left:
    status = ONSTRAIGHT

centerPoint_X1 = (320 - H1LD + H1RD)/2
centerPoint_X2 = (320 - H2LD + H2RD)/2
centerPoint_X3 = (320 - H3LD + H3RD)/2
dv12 = centerPoint_X1 - centerPoint_X2
dv23 = centerPoint_X2 - centerPoint_X3
pd = int(150 + (160-centerPoint_X2) * 0.5 + dv23 * 2)

if status == ONSTRAIGHT:  # 주행 시작
    if 1 < LiDAR and LiDAR < 300:
        print('Obstacle Detected at %d mm' % LiDAR)
        command = 'S0150E'

    elif result != False and light == False:
        command = 'S0150E'
        print('신호등 빨간색')
    ...
    else:
        command = 'S1%dE' %(pd)
    ...
```

그림 10. PD제어 코딩(일부)

III. 결론

이전 자율 주행 알고리즘에서는 자동차와 차선의 현재 상황을 단계별 여러 가지 Case로 구분하여 각각의 Case별 자동차의 주행 방향을 결정하도록 했었다. 이런 알고리즘은 On/Off 제어방법에 해당되며, 단계별 제어값이 급격히 바뀌는 문제가 있어 주행 방향이 급격히 꺾이고 주행이 부드럽지 못한 단점이 있었다. 또한, 다음 단계의 차선 위치를 예측하지 못해 회전이 급하게 이뤄질 때 차선 이탈하는 경우가 많았었다. 그러나 이번 비례미분제어(PD제어) 알고리즘을 적용하면, 차선을 이탈한 정도에 따라 비율적으로 앞바퀴 회전각도가 변하므로 자동차 주행방향이 단계별로 꺾이는 현상이 발생하지 않으며, 자동차 앞쪽 거리에 따른 차선의 틀어짐을 미분하여 제어값을 조정하므로, 차선이 급회전 할수록 더 큰 값으로 바퀴각도를 회전시킬 수 있어 이번 연구의 비례미분제어 알고리즘이 이전의 On/Off 제어 알고리즘 보다 안전한 자율 주행 알고리즘으로 적용될 수 있을 것이다.

참고문헌

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

[2] S. Jung, J. Yoon, S. Sull, "Efficient Lane Detection Based on Spatiotemporal Images", IEEE Trans. Intelligent Transportation Systems, 17(1), 289-295, Jan. 2016

[3] 이경민, 인치호. (2018). 도로 환경에 효율적인 새로운 차선 검출 방법. , 17(1), 129-136.

[4] 김용재, 구글로 공부하는 파이썬, 비제이퍼블릭, 2018.

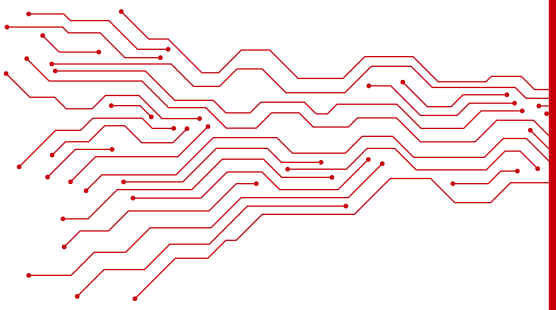
[5] 강봉주, 파이썬으로 실무에 바로 적용하는 머신러닝, 에이콘출판(주), 2019.

[6] 선우, PID제어란? (비례/적분/미분), 블로그, 2020.

최우수 논문

02 합성곱 신경망 및 이미지 인식을 이용한 스레드 풀에서의 자율 주행 알고리즘

인천하늘고등학교 배연욱, 서기범



합성곱 신경망 및 이미지 인식을 이용한 스레드 풀에서의 자율 주행 알고리즘

Self-Driving Algorithm Using Image Recognition and Convolutional Neural Network in Thread Pool

배 연 욱*, 서 기 범** YeonUk Pae, GiBeom Seo

요약

기존의 고등학교 과정의 자율 주행 자동차에서는 자율 주행 자동차에서 찍은 사진을 수학적으로 분석하여 구한 조향 값으로 자율 주행 자동차가 주행하도록 하였다 [1]. 본 연구의 자율 주행 자동차에서는 CNN(convolutional neural network, 합성곱 신경망) 인공지능이 자율 주행 자동차에서 찍은 사진을 분석하여 구한 servomotor(서보모터)의 조향 값으로 자율 주행 자동차가 자율 주행하도록 하였다. 본 연구에서는 사람이 원격 조정한 자동차에서 찍은 사진과 그 순간의 앞바퀴 조향 값, 즉 servomotor(서보모터)의 조향 값을 사용하여 CNN(convolutional neural network, 합성곱 신경망)을 훈련시켰다. 실제 자율 주행에서는 CNN(convolutional neural network, 합성곱 신경망)이 자율 주행 자동차에서 찍은 사진을 분석하여 servomotor(서보모터)의 조향 값, 즉 앞바퀴의 조향 값을 구하였고, 이를 바탕으로 자율 주행 자동차가 자율 주행하였다. 본 연구에서 제안하는 알고리즘은 기존의 수학적 계산에 근거한 알고리즘과 달리 인공지능 CNN(convolutional neural network, 합성곱 신경망)에 기반을 두고 있으므로 진정한 자율 주행 알고리즘이라고 할 수 있다.

Keywords: 자율 주행 자동차, 주행 알고리즘, 인공지능, 딥 러닝, CNN, 이미지 인식, 자율 주행

I. 서 론

고등학교에서 구현된 기존의 Raspberry Pi(라즈베리 파이)를 이용한 자율 주행 자동차의 주행 방식은 일반적인 자율 주행과는 거리가 있다. 카메라로부터 영상(원본 이미지)을 받고 calibration(캘리브레이션)을 통해 영상의 왜곡을 바로 잡아준 뒤, ROI(region of interest, 관심 영역)를 설정하는데, 이미지의 아랫부분을 차선 인식을 위한 ROI(region of interest, 관심 영역), 이미지의 윗부분을 신호등 인식을 위한 ROI(region of interest, 관심 영역)로 한다. 이때 ROI(region of interest, 관심 영역)란 특정 목적을 위해 data set (데이터 세트)에서 추출된 sample(샘플)이다 [2]. 이어서 차선 인식을 위한 ROI(region of interest, 관심 영역)에서 차선을 검출하기 위해 Canny edge detector (캐니 에지 디텍터)를 사용한다. Canny edge detector (캐니 에지 디텍터)란 다단계의 알고리즘을 사용하여 광범위한 edge(에지)를 이미지에서 검출하는 edge(에지) 검출 연산자이다 [3]. 차선과 도로의 경계가 검출된 Canny image (캐니 이미지)를 얻는다. Canny image (캐니 이미지)는 Hough transform (허프 변환)을 통해 Hough image (허프 이미지)가 되는데, 이 Hough image (허프 이미지)는 검출된 차선과 도로의 경계를 직선으로 인식한다. 이로부터 왼쪽, 오른쪽, 전방에 가로로 위치하는 3가지 차선이 검출된다. 이후 ROI(region of interest, 관심 영역) 위에 가로선, 세로선을 설정하여 격자 모양으로 나누고, 좌표를 이용한 수학적 계산 기반의 주행 알고리즘을 구현한다.

이러한 기존의 주행 방식은 이미지의 차선을 인식한 후, 좌표를 이용한 수학적 계산을 통한 주행 방식으로 원본 이미지를 Hough image (허프 이미지)로 변환해야 하며, 하나의 이미지에 대하여 처리해야 할 연산이 많아 비효율적이라는 한계가 있다. 일반적인 자율 주행의 의미와도 상응하지 못한다. 따라서 이러한 문제를 해결하여 더 정교한 주행이 가능해지고, 일반적인 자율 주행 자동차에 가까워질 수 있도록 본 연구에서는 인공지능 기반의 학습을 도입한 주행 알고리즘을 제안한다.

* 배연욱(인천하늘고등학교, emperorpae@gmail.com)
** 서기범(인천하늘고등학교, sgb1616@gmail.com)

II. 구조

본 연구에서 활용한 자율 주행 자동차 프로그램은 기본 구조로서 multithreading(멀티스레딩)을 이용한다(그림 1).

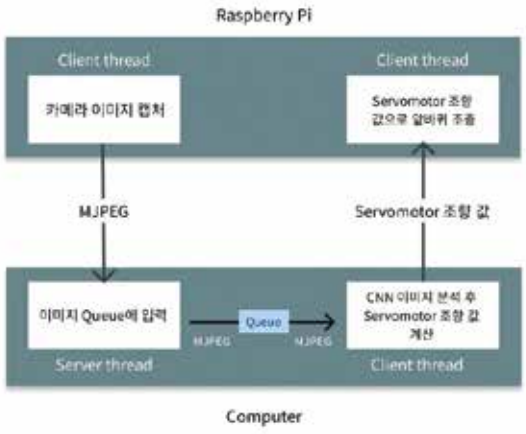


그림 1. 기본 구조

첫 번째 thread(스레드) 그룹에서는 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)가 client thread(클라이언트 스레드), 컴퓨터가 server thread(서버 스레드)가 된다. 자율 주행 자동차의 카메라에서 MJPEG(Motion JPEG, 모션 JPEG) 형식으로 사진을 찍어

Raspberry Pi (라즈베리 파이)가 사진을 컴퓨터로 전송한다. 컴퓨터에서는 사진을 받아 분석한 후 queue(큐)에 넣어 준다.

두 번째 thread(스레드) 그룹에서는 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)가 server thread(서버 스레드), 컴퓨터가 client thread(클라이언트 스레드)가 된다. 컴퓨터에서는 queue(큐)에 사진이 들어오면 기존에 훈련된 CNN(convolutional neural network, 합성곱 신경망) 인공지능 model(모델)을 이용하여 사진에 대한 servomotor(서보모터)의 조향 값을 결정하고, 조향 값을 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)로 보낸다. 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)는 컴퓨터가 보낸 조향 값에 따라 servomotor(서보모터)를 조향한다.

Ⅲ. 인공지능 Model 훈련

실제 주행 시 매 순간의 사진에 적합한 servomotor(서보모터)의 조향 값을 결정하는 인공지능 model(모델)을 훈련하기 위해서는 주행 시 매 순간에 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)가 찍은 사진과 상응하는 servomotor(서보모터)의 조향 값이 있어야 한다. 이를 위해 자율 주행 자동차를 키보드의 방향키를 이용하여 직접 원격으로 조종해 시험 도로 위를 주행하게 하고, 동시에 매 순간의 사진과 그때의 자율 주행 자동차의 servomotor(서보모터)의 조향 값을 함께 저장한다. 정교한 인공지능 model(모델)을 훈련시키기 위해 자율 주행 자동차를 직접 조종하여 시험 도로를 40여분간 주행하며 약 7만 장의 사진과 각 사진에 대한 servomotor(서보모터)의 조향 값을 저장하여 많은 데이터를 모았다.

인공지능 model(모델) 중 DNN(deep neural network, 심층 신경망)은 deep learning (딥 러닝, 심층 학습)에 기반을 둔 것으로, 본 연구에서는 DNN(deep neural network, 심층 신경망)의 종류 중 하나인 CNN(convolutional neural network, 합성곱 신경망)을 훈련시켰다. Deep learning (딥 러닝, 심층 학습)은 machine learning (머신 러닝, 기계 학습) 알고리즘의 한 종류로, 많은 layer(레이어)를 통해 raw input (초기 입력 값)으로부터 고차원의 특징을 추출하는 알고리즘이다 [4]. CNN(convolutional neural network, 합성곱 신경망)은 시각적 이미지 분석에 일반적으로 사용되는 DNN(deep neural network, 심층 신경망)의 한 종류이다. 이때 DNN(deep neural network, 심층 신경망)이란 input layer (인풋 레이어)와 output layer (아웃풋 레이어) 사이에 많은 layer(레이어)가 있는 ANN(artificial neural network, 인공 신경망)이다. ANN(artificial neural network, 인공 신경망)은 동물의 뇌를 구성하는 NN(neural network, 신경망)에 영향을 받아 만들어진 계산 시스템을 말한다 [5].

본 연구의 CNN(convolutional neural network, 합성곱 신경망)은 들어온 이미지에 대한 servomotor(서보모터)의 조향 값을 계산하는데, 이때 계산된 조향 값과 해당 이미지에 상응하는 실제 조향 값을 비교하여 계산된 조향 값이 실제 조향 값과 비슷해지도록 CNN(convolutional neural network, 합성곱 신경망)의 weight(가중치)가 조정된다. 이러한 weight(가중치) 조정은 backpropagation(역전파)을 이용하여 이루어진다. Backpropagation(역전파)은 보통 전체적인 학습 알고리즘을 의미하는데, stochastic gradient descent (확률 경사 하강법)와 같이 기울기가 어떻게 사용되는지도 포함된다 [6].

본 연구의 CNN(convolutional neural network, 합성곱 신경망)의 weight(가중치)는 예측 조향 값과 실제 조향 값의 mean squared error (평균 제곱 오차)를 최소화하도록 훈련된다. CNN(convolutional neural network, 합성곱 신경망)은 그림 2와 같이 1개의 normalization(노멀라이제이션, 정규화) layer(레이어), 5개의 convolutional layer (컨볼루셔널 레이어), 3개의 fully connected layer (완전 연결 레이어)로 총 9개의 layer(레이어)로 이루어져 있다.

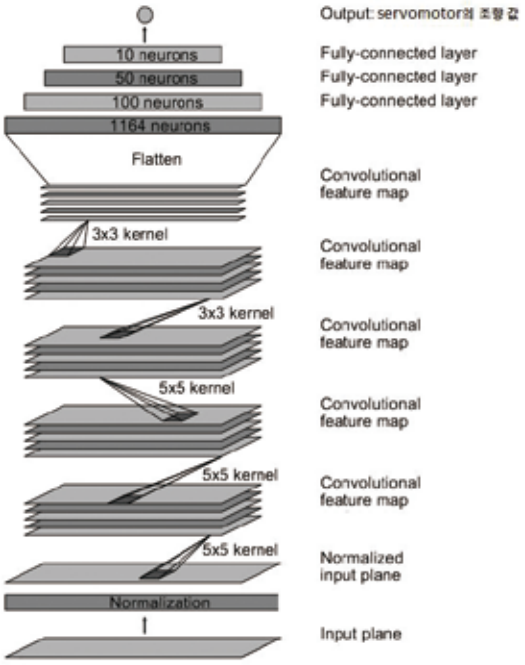


그림 2. CNN의 구조

첫 번째 layer(레이어)는 이미지 normalization(노멀라이제이션, 정규화)을 수행한다. 이를 위한 normalizer(노멀라이저, 정규화 코드)는 직접 설정하였으며, 학습 과정에서 조정되지 않는다.

5개의 convolutional layer (컨볼루셔널 레이어)는 feature extraction (특징 추출)을 수행하기 위해 디자인되었는데, 처음 3개의 convolutional layer (컨볼루셔널 레이어)에는 2×2 stride(스트라이드)와 5×5 kernel(커널)을 이용한 convolution(합성곱)을, 마지막 2개의 convolutional layer (컨볼루셔널 레이어)에는 stride(스트라이드) 없이 3×3 kernel(커널)을 이용한 convolution(합성곱)을 수행하게 하였다.

5개의 convolutional layer (컨볼루셔널 레이어) 뒤에 3개의 fully connected layer (완전 연결 레이어)가 있는데, 이는 결괏값, 즉 servomotor(서보모터)의 조향 값으로 이어진다. 3개의 fully connected layer (완전 연결 레이어)는 조향 관리를 위해 디자인되었지만, end-to-end(엔드 투 엔드)로 CNN(convolutional neural network, 합성곱 신경망)을 훈련시킨다는 점을 고려하면 CNN(convolutional neural network, 합성곱 신경망)에서 주로 feature extraction (특징 추출)을 수행하는 부분과 관리 역할을 하는 부분을 확실하게 구분할 수 없다는 것을 알 수 있다.

인공지능 model(모델)을 훈련하기 위해 필요한 이미지는 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)에서 30 FPS(frames per second, 초당 프레임)로 컴퓨터에게 보내는 이미지를 변환하여 사용한다. CNN(convolutional neural network, 합성곱 신경망)을 훈련할 때에는 데이터를 모으기 위해 직접 조종하며 저장한 이미지와 servomotor(서보모터)의 조향 값을 불러와 훈련한다.

Ⅳ. 주행 알고리즘

4.1 자율 주행

OpenCV(Open Source Computer Vision Library)란 주로 실시간 컴퓨터 비전에 사용되는 프로그래밍 함수를 포함한 library(라이브러리)이다 [7]. 본 연구에서는 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)가 컴퓨터로 보낸 원본 이미지로부터 가능한 한 차선만을 인식하기 위해 OpenCV에서 제공하는 Python(파이썬) 인터페이스 중 cv2의 cvtColor method(메서드)를 이용하여 원본 이미지(그림 3)의 color(색)를 gray(회색)로 변환한다. 즉, 원본 이미지를 흑백 사진(그림 4)으로 변환한다. 이후 cv2의 threshold method(메서드)로 흑백 사진의 일부는 흰색, 일부는 검은색으로 변환하여 흰색과 검은색만 존재하는 사진(그림 5)을 얻는다. 이때 최종적으로 변환한 사진에서의 흰색 부분이 차선인 것이다. 인공지능 model(모델)이 이렇게 변환한 사진에 적절한 servomotor(서보모터)의 조향 값을 예측한 뒤, 자율 주행 자동차의 Raspberry Pi (라즈베리 파이)로 조향 값을 보내 Raspberry Pi (라즈베리 파이)가 조향 값에 따라 servomotor(서보모터)를 조향하며 주행한다.

그림 3. 원본 사진

그림 4. 흑백 사진

그림 5. 흰색과 검은색만 존재하는 사진

4.2 Image Detection (이미지 인식)

자율 주행 중 LiDAR(라이다) 센서를 이용하여 장애물을 인식하여 멈추는 방법과 제공된 파일을 개발하여 신호등의 적신호를 인식하여 멈추는 방법은 불안정하였다. 따라서 본 연구에서는 image detection (이미지 인식)을 통해 장애물 및 신호등의 적신호를 인식하여 자율 주행 자동차의 움직임을 제어하였다. 자율 주행 시와 마찬가지로 OpenCV에서 제공하는 Python(파이썬) 인터페이스 중 cv2를 이용하였다. 장애물 및 신호등 적신호의 특징적인 부분을 cv2에서 제공하는 template matching (템플릿 매칭) 함수인 matchTemplate을 위한 template(템플릿)으로 하였다. Template matching (템플릿 매칭)은 scale(스케일) 변화에 적합하지 않기 때문에 template(템플릿)을 거리에 맞게 scale(스케일) 범위를 조정한 뒤 여러 template(템플릿) 이미지들을 만들고, 템플릿 매칭을 진행한다. 이를 이용하여 template scale (템플릿 스케일) 사이즈로 거리를 조종할 수 있었다. 상기한 과정을 통해 자율 주행 중 장애물 또는 신호등 적신호가 인식되면 자율 주행 자동차가 정지하고, 인식되지 않으면 다시 주행하도록 하였다.

Ⅴ. 결론 및 제언

5.1 결론

본 연구에서는 자율 주행 자동차에서 찍은 사진을 변형하여 미리 훈련된 인공지능 model(모델)인 CNN(convolutional neural network, 합성곱 신경망)에 넣고, servomotor(서보모터)의 예측 조향 값을 바탕으로 주행하는 알고리즘을 제안하였다. 자율 주행 자동차에 제안한 알고리즘을 적용하여 자율 주행 자동차용 시험 도로에서 자율 주행하도록 하였고, 안정적으로 완주하였다. 이를 통해 본 연구에서의 주행 알고리즘이 유효함을 검증하였다.

또한 실제 자동차를 기반으로 한 연구에서도 본 연구와 마찬가지로 사람이 정보를 분해하지 않고 이미지와 steering wheel (운전대)의 회전 각도만을 정보로 주었는데, 다양한 환경에서의 약 72시간의 주행 데이터만으로 매우 성공적인 결과를 도출하였다 [8]. 이로써 본 연구에서 제안한 알고리즘의 확장성 또한 검증되었다.

5.2 제언

5.2.1 환경에 민감한 자율 주행

제안한 알고리즘에서는 이미지를 검은색과 흰색만 존재하는 사진으로 변형하였다. 따라서 본 연구의 알고리즘은 빛의 양에 민감하게 반응한다. 원본 이미지를 흑백 사진으로 변환한 뒤 일정 기준보다 밝으면 흰색, 그렇지 않으면 검은색으로 변환하였기 때문에 차선 이외의 것이 흰색으로 나타나는 경우, 차선이 흰색으로 나타나지 않는 경우 등 완벽하게 차선만 흰색으로 보이는 사진을 도출하기 어려운 점이 있다. 따라서 제안한 알고리즘은 환경, 즉 빛의 양과 같은 조건에 민감하게 반응하여 그 안정성에 의문이 제기될 수 있다. 자율 주행 자동차가 주행하는 도로에서는 변수가 적어 주행이 성공적으로

이루어질 수 있었지만, 실제 자동차가 자율 주행을 하기 위해서는 사진의 색상을 변형하지 않고, 다양한 환경에서 조건을 달리 하며 주행하여 데이터를 모아야 한다. 본 연구에서도 주행 알고리즘의 안정성을 제고하기 위해서는 원본 이미지 그대로를 이용하되, 빛의 양 등을 조절하여 다양한 환경에서 주행한 데이터를 모아야 한다. 조건을 달리할 때 예컨대 빛의 양을 조절하는 경우 자율 주행 자동차가 주행할 때의 주변 조명의 개수, 밝기 등을 달리하거나 카메라 자체에서 받아들이는 빛의 양을 조절할 수 있다.

5.2.2 지연 발생

본 연구에서의 알고리즘은 모든 이미지의 색상을 변환하여 흰색과 검은색만 존재하는 사진을 얻기 위해 시간을 쓸 뿐만 아니라, 이미지를 처리(CNN(convolutional neural network, 합성곱 신경망)이 이미지로부터 servomotor(서보모터)의 값을 도출)하는 데 시간이 걸리므로 컴퓨터에서 정보를 받는 시점, 즉 사진을 받아 queue(큐)에 넣는 시점과 다른 thread(스레드)에서 queue(큐)에 들어와 있는 사진을 꺼내 처리를 시작하는 시점의 차가 미세하게 발생한다. 이는 queue(큐)의 FIFO(first in, first out, 선입 선출) 속성으로 인해 먼저 온 것이 우선 처리되고, 처리가 끝날 때까지 다음 것은 대기 상태에 놓이게 되기 때문이다. 이렇게 미세한 delay(딜레이)가 쌓여 결국 밀리고, 느려질 수 있다. 이러한 구조는 훈련을 위한 데이터를 모을 때는 영향이 없지만, 실제 자율 주행 시에는 매우 큰 문제가 된다. 따라서 이를 해결하기 위해 queue(큐)에서 정보를 꺼내는 시점에 queue(큐)에 들어온 가장 최근 정보(이미지)를 사용해야 한다. 주행 시 처리되지 못한 이미지가 미세하게 쌓여 정보 처리가 밀려 버리는 현상을 방지할 수 있을 것이다.

참고문헌

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

[2] Ron Brinkmann, The Art and Science of Digital Compositing, Morgan Kaufmann, p. 184, 1999.

[3] John Canny, "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 9, no. 6, pp. 679-698, 1986.

[4] Li Deng, Dong Yu, Deep Learning: Methods and Applications, Now Publishers, pp. 199-200, 2014.

[5] Yung-Yao Chen, Yu-Hsiu Lin, Chia-Ching Kung, Ming-Han Chung, I-Hsuan Yen, "Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes", Sensors, vol. 19, no. 9, p. 3, 02 May 2019.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, p. 200, 2016.

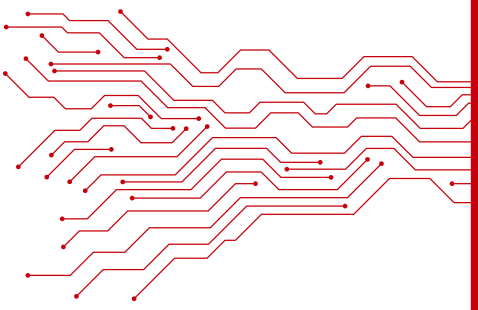
[7] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, Victor Eruhimov, "Realtime Computer Vision with OpenCV", Queue, vol. 10, no. 4, 1 April 2012.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, "End to End Learning for Self-Driving Cars", 25 April 201

최우수 논문

03 연속적 되먹임 체계를 통한 직선 및 곡선 차선 자율 주행과 다각형 추출을 통한 신호등 인식 알고리즘

통진고등학교 안중원, 박 진, 이승현, 채종은, 김진석



연속적 되먹임 체계를 통한 직선 및 곡선 차선 자율 주행과 다각형 추출을 통한 신호등 인식 알고리즘

Autonomous Driving Algorithm in Straight and Curved Lane through Continuous Feedback and Traffic Light Recognition Algorithm through Polygon Extraction

안중원*, 박 진**, 이승현***, 채종은****, 김진석*****
Ahn Joongwon, Park Jin, Lee Seunghyun, Chae Jongeun, Kim Jinseok

요약

본 연구에서는 실제 주행 상황을 재현한 트랙에서 모형 자동차가 차선을 따라 주행하고 교통신호 변화와 보행자 및 장애물 출현에 대응하여 멈추고 출발하기 위한 알고리즘을 제안한다. 알고리즘에서 모형 자동차는 카메라 이미지를 변환하여 차선의 위치 및 주행 상황 유형을 추출하여 조향하고 다각형 추출을 통해 신호등의 신호를 인식해 정지하고 출발한다. 또한 라이다 센서를 통해 보행자와 장애물까지의 거리를 측정하여 정지하고 출발한다.

Keywords: 자율 주행 자동차, 주행 알고리즘, 차선 인식, 다각형 추출, 되먹임

I. 서 론

자율 주행 자동차란 자동차관리법 제 2조에 따르면 운전자 또는 승객의 조작 없이 자동차 스스로 운행이 가능한 자동차를 말한다. 스스로 운행이 가능한 자동차를 만들기 위해서는 먼저 인간이 감각을 대신할 다양한 센서를 이용하여 하드웨어를 제작해야 한다. 또한 센서가 입력한 값을 토대로 명령을 내려야 자동차가 스스로 움직일 수 있다. 따라서 본 연구에서는 자율 주행 자동차의 카메라 이미지를 변환하여 차선의 위치를 인식하고 주행 상황을 나누어 움직이고 다각형 추출을 통해 신호등의 신호를 인식해 정지하고 출발하는 과정을 서술하고 있다. 본 연구의 과정은 다음과 같이 진행되었다. 먼저 토대가 되는 코드를 작성하고, 그 오류를 바탕으로 시험 주행을 하여 오류를 확인한다. 오류가 발생하면 그 오류를 분석하고 상수 보정을 통해 다시 시험 주행을 거쳐 오류를 줄이며 반복 실험과 수정을 수차례 거쳐 알고리즘이 완성되었다.

II. 교통 상황 인식 및 정지

트랙에는 신호등, 신호와 관계없이 신호등 앞을 건널 수 있는 보행자 모형, 차단기를 임의의 위치에 배치할 수 있다. 모형 자동차가 그 세 가지 기물로 인한 교통상황에 정지하고 출발할 수 있도록 하는 것이 주행 알고리즘의 목표 중 하나이다.

2.1 신호등 및 신호 인식 알고리즘

모형 자동차 전방 카메라를 통해 얻은 이미지의 상단 60 픽셀의 영역을 신호등 및 신호 인식의 관심영역으로 설정하고, 관심영역 이미지를 이진화한 후 윤곽선을 검출하여 다각형을 근사하였다. 근사한 다각형이 8 ~ 10각형일 경우 검은 신호등 상자와 그 위의 기둥을 인식한 것으로 간주하였다. 다각형의 꼭짓점 중 x 좌표가 최소인 점과 최대인 점의 x 좌표를 각각 사각형의 왼쪽과 오른쪽으로, y 좌표가 3번째로 작은 점과 최대인 점의 y 좌표를 각각 사각형의 위와 아래로 설정하여 다시 신호등 상자만을 추출하였다. 8 ~ 10각형 중에도 그 비율이 신호등과 비슷한 사각형만 처리하도록 했다. 높이에 대한 너비의 비가 1에서 2.5 사이일 때만 처리하도록 했을 때 신호등과 다른 사물을 가장 잘 구분했다. 신호등이 충분히 가까워졌을 때만 정지하기 위해 신호등 상자의 너비가 20 픽셀 이상일 때만 신호를 인식하게 하였다. 어떤 신호가 점등되었는지 구하기 위해 상자 내에서 그레이스케일 값이 가장 큰 점, 즉 흰색에 가장 가까운 점을 구했다. 구한 점이 충분히 밝아야 점등된 것으로 볼 수 있으므로 그레이스케일 값이 220 이상일 경우 현재 점등된 것으로 인식하도록 하였다. 신호등의 가장 왼쪽 등이 적색등이므로 구한 점의 x 좌표가 중앙보다 왼쪽일 때만 정지 명령을 내리도록 하였다.

그림 1. 신호등 이미지에서 검출된 다각형

* 안중원(통진고등학교, winterdawnlight123@gmail.com)
** 박 진(통진고등학교, smileskyj@hanmail.net)
*** 이승현(통진고등학교, sounghyun23@naver.com)
**** 채종은(통진고등학교, jachae0722@naver.com)
***** 김진석(통진고등학교, sadgod33@naver.com)

2.2 장애물 및 보행자 인식 알고리즘

모형 자동차 전방에 부착된 라이다(LiDAR) 센서를 통해 전방의 장애물이나 보행자까지의 거리를 측정할 수 있다. 측정된 값이 특정 상수보다 작으면 장애물이 충분히 가까워졌다는 뜻이므로 정지 명령을 내린다. 라이다 센서의 오차, 주행 상황 인식과 명령어 실행 사이의 시차 때문에, 정지를 명령할 거리는 실험적으로 구해야 했다. 여러 번의 실험을 통해 센서의 값이 380mm 미만일 때 정지 명령을 내리도록 하면 안정적으로 멈출 수 있었다.

Ⅲ. 차선 주행

주행상황유형은 "초기상태(ON_NOP)", "직선주행(ON_STRAIGHT)", "회전 준비 (ON_LEFT_READY, ON_RIGHT_READY)", "회전 (ON_LEFT, ON_RIGHT)", "커브 주행 (ON_SLEFT, ON_SRIGHT)", "정지(ON_STOP)"로 나누었다. "회전 준비", "회전", "커브 주행"은 좌우를 다른 상황 유형으로 처리하여 총 9가지의 상황 유형이 사용되었다. 처음에는 "초기 상태"로 시작한다. 알고리즘이 호출될 때마다 우선 교통상황을 인식한다. 정지해야 한다고 판단하면 정지하고, 계속 주행해야 한다고 판단하면 이전 주행 상황 유형에 따라 차선 위치 정보로부터 현재 주행 상황 유형을 판단한다. 이후는 판단된 상황 유형에 따라 주행한다. 상황 주행 판별과 주행에 쓰이는 상수의 값들은 개개의 모형 자동차마다 다를 수 있으며, 실제 코드에서 사용된 값들은 실험을 통해 반복적으로 보정되었다.

3.1 차선 위치 정보

좌우 차선(left, right)과 자동차 앞을 가로막는 끝 차선(end)은 미리 제공되었다. 이 세 변수는 각각 차선을 나타내는 직선에 관한 정보를 갖고 있지만 본 연구에서는 차선의 검출 여부만을 사용했다. 인식된 차선에 대한 단편적인 값인 HnLD, HnRD, VnD도 제공되어 차선에 대한 유의미한 정보를 쉽게 계산하였다. center는 그렇게 계산된 값들 중 하나이다. 좌우 차선의 중앙값을 나타내는 center 값은 아래 각 주행 상황의 판별과 주행에서 두루두루 쓰였다. center는 H1LD와 H1RD, H2LD와 H2RD, H3LD와 H3RD 각각의 차의 평균이다. center가 음수이면 좌우차선이 왼쪽으로 치우쳐있음을, 양수이면 오른쪽으로 치우쳐있음을 나타낸다. 단 이러한 단편적인 평균값은 각종 예외상황에서 오류를 일으켰다. 반복적인 주행을 통해 발견된 이런 오류 상황에서는 center 값을 계산하지 않았다.

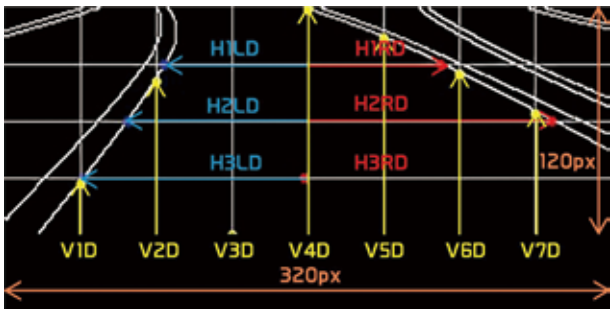


그림 2. HnLD, HnRD, VnD의 정의

3.2 주행 상황

이전 주행 상황과 차선 위치 정보에 따라 주행 상황을 판단한다. 이전 주행 상황과 차선 검출 여부를 주로 활용하고, 그 외 추가적인 정보를 계산하여 세세한 판단을 한다.

3.2.1 정지해야 할 때와 이전 주행 상황이 "정지"일 때

신호등 적색등이 점등되었거나 장애물이나 보행자까지의 거리가 가깝다고 판단되면 정지한다. 그 외의 경우에는 주행 상황 판단에 들어가게 된다. 그러나 신호등 적색등 점멸 여부는 주변의 빛에 따라 잘못 인식될 수 있고, 라이다 센서의 값도 종종 오차가 발생한다. 알고리즘이 호출될 때의 이미지와 센서에 따라서만 주행하면 아직 정지해 있어야 할 상황에 출발하는 오류가 발생한다. 이렇게 미리 출발하는 오류를 줄이기 위해서, 출발해도 된다는 판단이 연속해서 특정 횟수 이상 이루어지는 경우에만 출발하게 하였다. 여러 번 주행해본 결과 5번 연속으로 출발로 판단하도록 하면 미리 출발하는 오류가 거의 없었다.

주행 상황은 출발하는 순간의 카메라 이미지만으로는 판단하기 어렵고 이전 상황에 대한 정보가 있어야 정확하게 판단할 수 있다. 그래서 정지 명령을 내릴 때 이전 주행 상황에 관한 정보를 전역변수에 저장해 놓았다가 출발해도 된다는 판단이 이루어지면 이전 주행 상황을 복구하여 주행 상황을 판별하였다.

```
if stop:
    time_left = STOP_CYCLE
    if status is ON_STOP:
        print('STOP -> STOP')
        return ON_STOP
    else:
        global status_before_stop, survo_before_stop
        status_before_stop = status
        survo_before_stop = prev_survo
        print('* -> STOP')
        return ON_STOP

if status is ON_STOP:
    time_left -= 1
    if time_left >= 0:
        print('STOP -> STOP (WAIT)', time_left)
        return ON_STOP
    print('(STOP -> *)')
    status = status_before_stop
```

그림 3. 정지 알고리즘

3.2.2 "초기 상태"이거나 이전 주행 상황이 "직선 주행"일 때

왼쪽 차선과 오른쪽 차선의 검출 여부에 따라 "직선 주행", "회전 준비"를 한다. 왼쪽 차선과 오른쪽 차선이 둘 다 존재할 경우에는 "직선 주행"한다. 왼쪽 차선이나 오른쪽 차선 중 한쪽이 인식되지 않을 경우 차선이 인식되지 않는 방향으로 "회전 준비"한다.

3.2.3 이전 주행 상황이 "회전 준비"일 때

3.2.3.1 정상적인 회전 준비"

회전하려는 쪽의 차선이 검출되지 않고 나머지 두 차선 중 적어도 하나가 검출되는 경우 정상적인 "회전 준비" 상황이다. 예를 들어 "좌회전 준비" 상황에서 왼쪽 차선이 검출되지 않고 오른쪽과 앞 차선 중 적어도 하나가 검출되는 경우는 "좌회전 준비" 주행을 하며

조금씩 회전하다 보면 마주칠 것으로 예상할 수 있는 상황이다. V2D, V4D, V6D의 평균 값(endlane_distance)에 따라 회전을 시작할지 조금 더 "회전 준비"를 할지 결정한다. endlane_distance 값이 작을수록 끝 차선이 더 가깝다는 것을 의미하므로 이 값이 특정 기준보다 작으면 회전을 시작한다. 단, 그 기준은 세부적인 상황에 따라 다르다. 앞쪽 차선이 뾰족하게 꺾여있으면 자동차가 차선의 모서리를 향해 가고 있다는 것을 의미한다. 이럴 때는 직선 V2D-V4D와 직선 V4D-V6D의 기울기의 부호와 차를 계산해서 판별할 수 있다. 두 기울기의 부호가 다르고 두 기울기 값이 0.3 이상 차이나면 모서리를 향해가고 있다고 판단한다. 그럴 때는 endlane_distance의 기준값이 커서 일반적인 회전보다 일찍 "회전" 상황 유형으로 전환한다. 모서리를 향해가지 않을 때는 V3D-V5D 직선의 기울기에 따라 "회전"을 할지 "커브 주행"을 할지 결정했다. 기울기가 특정 값(STURN_GRADIENT=0.18)보다 크면 "커브 주행", 작으면 "회전"을 한다.

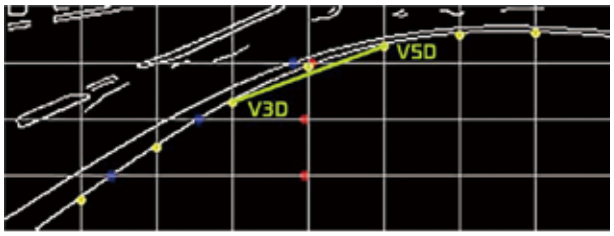


그림 4. V3D-V5D 직선의 기울기

3.2.3.2 "회전 준비"중단

왼쪽 차선과 오른쪽 차선이 모두 검출된 경우에는 "직선 주행"으로 전환한다. 간혹 그 방향의 회전을 준비할 상황이 아님에도 "회전 준비" 상황으로 판단하는 경우가 있다. 회전하려는 쪽의 차선이 검출되고 오히려 반대쪽 차선이 검출되지 않는 것을 그러한 상황이라고 볼 수 있다. 그럴 때는 반대쪽 방향으로 "회전 준비"하도록 한다. 예를 들어, "좌회전 준비" 상황에서 왼쪽 차선이 검출되고 오른쪽 차선이 검출되지 않으면, "우회전 준비" 상황으로 전환한다. 이렇게 하면 왼쪽으로 치우쳐 있는 주행 방향을 오른쪽으로 교정해서 다시 차선을 따라 주행하게 된다. 차선이 전혀 검출되지 않는 상황에서는 정확한 판단이 어려우므로 이전 상황을 유지한다.

3.2.4 이전 주행 상황이 "회전"이나 "커브 주행"일 때

회전하고 있는 쪽의 차선이 검출되지 않고, 반대쪽 차선과 끝 차선이 검출되면 맞게 회전하고 있는 것이므로 계속 회전한다. center 값이 존재하고 좌우 차선이 모두 존재하면 완전히 직선 차선에 오른 것으로 판단하고 "직선 주행"으로 전환한다. 회전하고 있는 쪽의 차선과 끝 차선이 검출되고 오히려 반대쪽 차선이 검출되지 않으면 너무 많이 회전한 것이므로 반대쪽 방향의 "회전 준비"로 전환하여 방향을 바로잡는다. 그 외의 상황은 정확한 상황을 판단하기 어려우므로 이전 상황을 유지한다.

3.3. 주행

3.3.1 "직선 주행"

Center 값이 정상적으로 계산된 경우 중앙을 향하는 서보모터 값에 center2servo(center) 값을 더한 값을 서보모터에 명령한다. center2survo는 아래와 같이 정의되는 함수로, 중앙에서 벗어난 정도에 따라 연속적인 보정 값을 가지게 하기 위해 사용된다. 선형 함수를 사용하지 않은 것은 오류가 누적되어 커진 뒤에 큰 값으로 방향을 바로잡는 것보다 작은 오류가 있을 때 즉시 방향을 수정하도록 하는 것이 더 안정적이었기 때문이다. center 값을 구하기 어려운 경우는 중앙으로 직진하도록 했다.

```
def center2survo(x):
    y = int(20 * math.pow(abs(x)/80, 0.7))
    if x < 0:
        y = -y
    return y
```
그림 5. 직진 중 서보모터 회전각 계산식

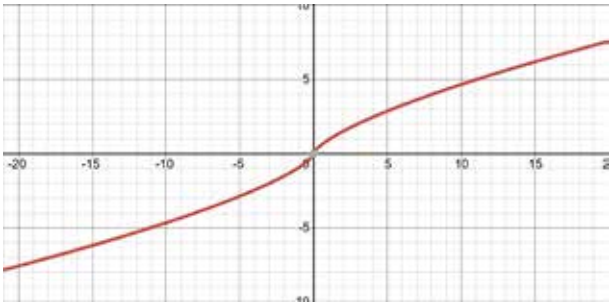


그림 6. HnLD, HnRD(x)와 서보모터 회전값(y)의 관계식 그래프

3.3.2 회전 준비

좌우 차선이 검출되지 않을 때 바로 회전할 경우 너무 안쪽으로 돌게 되고, 가까워질 때까지 계속 직진하면 너무 바깥쪽으로 돌게 되므로 잠깐 회전할 방향으로 조금 치우쳐 도는 것이 "회전 준비"이다. "회전 준비" 상황에서는 서보모터를 중앙으로부터 READY_TURN 값(=6)만큼 약간 치우쳐서 주행하도록 한다.

3.3.3 회전

90도 이상의 각을 단번에 회전해야 할 때는 "회전" 주행을 한다. 끝 차선이 가까우면 크게 회전하고 멀면 비교적 작게 회전할 수 있도록 endlane_distance(=distance) 값에 따라 서보모터의 회전각(=SURVO_MID±turn_angle(...))을 계산한다. fitin 함수는 fitin(m,x,M) = min(max(m, x), M)로 정의되는 함수이다. TURN과 EDFT(ENDLANE_DISTANCE_FOR_TURN)은 각각 "회전"/"커브 주행" 여부와 좌우에 따라 달라지는 상수이다.

```
def turn_angle(TURN, distance, EDFT):
    return TURN * fitin(-1,(EDFT - (distance + 12))//10,8)
```
그림 7. 회전각 계산식

3.3.4 "커브 주행"

S자 구간에서 "커브 주행"할 때에는 앞에 보이는 차선까지의 거리, 즉 endlane_distance의 값을 일정하게 유지하는 적극적인 음성 피드백 전략을 사용하였다. endlane_distance가 기준값(좌우 모두 62)보다 크면 너무 안쪽으로 돌아서 안쪽 차선을 밟을 수 있다. 이때는 서보모터의 회전각을 비교적 작게(=SURVO_MID ± 18(STURN_WEAK)) 했다. endlane_distance가 기준값보다 작으면 너무 바깥쪽으로 돌아서 바깥쪽 차선을 밟을 수 있다. 이때는 서보모터의 회전각을 endlane_distance 값에 따라 연속적으로(=SURVO_MID ± turn_angle(...)) 계산하였다.

```
def ON_SLEFT(center, endlane_distance):
    command_dc = DC_FORWARD
    if endlane_distance > ENDLANE_DISTANCE_FOR_S_FEEDBACK_LEFT:
        command_survo = SURVO_MID - STURN_WEAK
    else:
        command_survo = SURVO_MID - turn_angle(STURN,endlane_distance,EDFTL)
    return command_dc, command_survo
```

그림 8. 커브 알고리즘

V. 결론 및 제언

5.1 결론

본 연구에서는 모형 자동차가 차선을 따라 주행하고 교통신호 변화와 보행자 및 장애물 출현에 대응하는 알고리즘을 제안했다. 특히 상황에 따라 서보 모터의 값을 연속함수로 되먹여 유연하게 대처할 수 있었다. 또한 다각형 검출을 통해 신호등의 형태를 인식하여 교통신호 변화에 대응하여 정지 및 출발할 수 있었다. 알고리즘은 모형 자동차의 주행을 통해 검증되었다.

5.2 제언

5.2.1 제한적 모형의 한계와 이에 따른 과적합

본문의 알고리즘은 해당 모형 자동차와 해당 대회의 트랙에 과적합(Overfitting)되어 있다는 한계가 존재한다. 이러한 한계가 존재하는 이유는 모형과 트랙이 현실의 자동차와 차이가 존재하기 때문이다. 이러한 예시로는 속도 조절 능력, 트렉션 컨트롤 시스템(TCS), 디퍼런셜 기어 유무 등의 차이 등이 있다. 디퍼런셜 기어를 예로 들자면 실제 자동차에서는 선회 시 양쪽 뒷바퀴의 각속도가 서로 다르기 때문에 이를 보정하기 위해 디퍼런셜 기어를 사용하는 것이다. 그러나 모형 자동차에서는 단일 샤프트로 연결되어 있었기 때문에 미끄러짐이나 비틀림이 있었을 것이다. 이는 회전 각도를 정확하게 알 수 없게 만드는 요인이 된다. 이러한 하드웨어적 문제들로 인해 수학적인 계산에 어려움이 있었고 결과 중심적으로 디버깅한 결과 반복실험을 통한 상수 가중치 도입으로 귀결되었다. 이는 다양한 맥락 속에서 보편적으로 적용되기 어려울 수 있으며 수학적으로 설명되기 어렵다는 것을 의미한다.

5.2.2 알고리즘 호출 간 시간 지연 문제

알고리즘의 한 호출과 다음 호출 사이에는 통상 0.2 ~1.0초 정도의 시간 간격이 있었다. 명령 간에 지연은 알고리즘의 성능을 크게 제약했고, 그 시간 간격 조건에서 최소한의 오류가 발생하도록 하는 것이 과제였다. 그럼에도 종종 시간 간격이 1.0초 이상으로 크게 발생할 때에는 그 사이에 섬세한 주행 명령을 내릴 수 없어 속수무책으로 트랙을 이탈하곤 했다. 알고리즘 호출 간 시간 간격을 줄이려면 라즈베리 파이의 센서 인식이나 와이파이 통신에서의 지연 문제를 해결해야 할 것으로 보인다. 특히 와이파이 통신 문제의 경우 파이썬 코드를 라즈베리 파이에 직접 업로드해서 실행해서 해결할 수 있을 것이다.

5.2.3 인공 신경망 활용 가능성

본 연구에서 제안한 알고리즘은 HnLD, HnRD, VnD 값들을 합, 차, 실수배 등의 연산 후 기준치와 비교하여 다음 주행 상황과 주행 명령을 결정하는 구조이다. 때문에 알고리즘을 개발하는 과정은 대부분에 반복적인 시험 주행을 통해 상수를 조금씩 수정하는 기계적인 작업이었다. 이러한 과정은 인공신경망을 통한 기계학습으로도 구현이 가능하다. 자율 주행 알고리즘 개발에 인공신경망 학습을 활용하면 기계적인 작업에 드는 시간을 절약할 수 있을 뿐 아니라 더 효과적인 알고리즘을 발견하는 데까지 나아갈 수 있을 것이다.

참고문헌

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

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

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

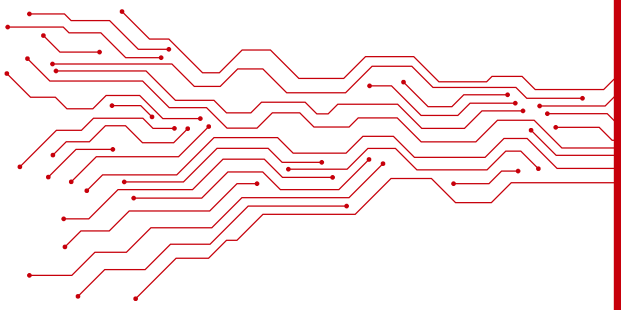
[4] 김건정, 김상동, 허수정, 이종훈, 박용완, Advanced Automotive Sensor Technologies: RADAR and LIDAR vol.25, no.3, SK telecom, pp. 383-405, 2015.

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

우수 논문

04 해석 기하적 기법을 활용한 직진 안정화 알고리즘 구현

하나고등학교 김준수, 이용욱, 장진영, 최민준



해석 기하적 기법을 활용한 직진 안정화 알고리즘 구현

Linear Stabilization Algorithm Design Using Analytical Geometric Techniques

김 준 수*, 이 용 욱**, 장 진 영***, 최 민 준****

요약

본 연구는 자율 주행 자동차의 안정적인 직진 운행을 위한 알고리즘을 찾는 것에 중점을 두었다. 이를 해석기하학적 기법을 이용하여 이미지 상에서 차선과 수직선의 교점의 기울기를 계산했다. 그리고 중앙선을 기준으로 차체가 왼쪽에 있는지 오른쪽에 있는지 판단했다. 차선의 연장선의 교점을 구하고 기존 점들의 중점을 구하여 두 점을 이은 선분의 기울기를 계산하였다.. 이를 통해 구한 기울기의 부호와 설정한 연장선의 교점의 x좌표의 부호를 통해 회전방향을 결정하였다. 시뮬레이션을 이용한 실험을 통해 곱해줄 상수 Ef의 적당한 값을 찾았다. 그 결과 Ef의 값이 0.060일 때 가장 최적한 된 회전을 통해 직진 주행을 할 수 있었다. 이 방법이 임의의 트랙에서 항상 성립하지는 않지만 비슷한 방법을 통해 특정 트랙에 최적화된 Ef값을 찾을 수 있을 것이다.

Keywords: 자율 주행, 직진 주행, 해석기하, 차선 인식, 시뮬레이션

I. 서 론

자율 주행 자동차는 운전자의 개입 없이 자동으로 주행하는 자동차를 말한다. 현재 자율 주행 자동차는 전서계 기업들의 기술력을 필두로 하여 빠르게 발전해 나가고 있다. 구글의 웨이모(Waymo)가 대표적이며 이미 도로 주행이 가능한 수준의 자동차가 개발되었고 자율 주행 택시 서비스도 선보였다. 자율 주행 기술은 미래의 자동차 시장과 더불어 연결된 무수한 산업들을 좌지우지할 중요한 요인이기에 전세계로부터 각광받고 있고 누가 더 정밀하고 안전한 기술을 개발할 수 있는지가 관건이 되었다. 자율 주행 자동차는 각종 센서들과 알고리즘이 복합적으로 어우러져 도로 주행 시 발생하는 다양한 문제 상황들을 해결함으로써 자율적인 주행이 가능해진다. 연구를 진행하면서 여러 번의 주행 테스트를 해본 결과 직진 주행의 안정화가 다른 문제 상황 해결을 위해서도 필수적임을 알 수 있었다. 따라서 본 연구는 도로 주행의 기본이 되는 직진 주행 알고리즘을 안정화하고 효율적으로 구성하는 것에 초점을 두었다. 이를 위해 알고리즘을 직접 구성해보면서 수학적 개념의 도입을 통해 보다 정밀한 알고리즘을 구축하는 방향으로 연구를 진행하였다.

II. 직진 주행 알고리즘

2.1. 직진주행

우리팀은 직진주행을 위해서 두가지 정보를 활용하여 계산했다. 첫번째 정보는 양쪽 차선의 기울기이며, 두번째 정보는 양 차선의 끝점의 중점이다. 위의 두 점이 모두 축 위에 오는 순간은 차가 중앙선에서 직진하는 경우로 유일하므로, 우리는 그 상황을 만드릭 위해 회전을 하면 된다. 먼저 각각의 차선 위에있는 두 점, 즉 총 4개의 점을 활용한다.

여기서 우리는 좌표계의 원점을 화면의 중심으로 설정하였고, 이를 기준으로 4개의 점을 좌표로 나타내었다. 이들을 각각 (x_{r1}, y_{r1}), (x_{r2}, y_{r2}), (x_{l1}, y_{l1}), (x_{l2}, y_{l2})라 하자. 그러면 왼쪽, 오른쪽 차선의 기울기는 각각 다음과 같이 표현된다.

$$m_r = \frac{y_{r2} - y_{r1}}{x_{r2} - x_{r1}}, m_l = \frac{y_{l2} - y_{l1}}{x_{l2} - x_{l1}}$$

그리고 위의 정보를 이용하여 두 차선의 직선의 방정식을 구할 수 있다.

$y = m_r(x - x_{r1}) + y_{r1}, y = m_l(x - x_{l1}) + y_{l1}$ 이제 교점의 x좌표를 나타낼 수 있다.

$$x' = \frac{(m_r x_{r1} - m_l x_{l1}) - (y_{r1} - y_{l1})}{m_r - m_l}$$

이제 두 끝점의 중점을 구하자.

$$x_m = \frac{x_{r1} + x_{l1}}{2}$$

우리는 여기서 두 x좌표를 이용해 중앙선을 조절하였다. 먼저 회전의 방향을 결정하자. 만약 x_m, x'이 모두 양수라면 차는 오른쪽으로 회전해야 한다. 이와 반대로 위의 두 좌표가 모두 음수라면 차는 왼쪽으로 회전해야 한다.

* 김준수(하나고등학교, junsu01041526634@gmail.com)
** 이용욱(하나고등학교, dldyddnr4285@gmail.com)
*** 장진영(하나고등학교, cjy6275@naver.com)
**** 최민준(하나고등학교, has_19184@sonline20.sen.go.kr)

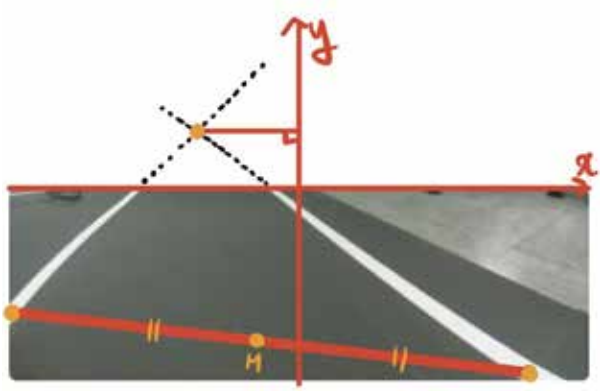


그림 1. 중앙선 조절 상황(1)

이제 이외의 경우를 따져보자.

$(x_m < 0 \text{ and } x' > 0)$ or $(x_m > 0 \text{ and } x' < 0)$인 경우에서 만약 중점보다 교점을 우선시한다면 차는 방향은 맞추더라도 중앙선으로 돌아가지 못할 것이다. 그래서 우리는 여기서 x_m에 가중치를 둔 평균을 이용하여 각각의 상황에서 차의 회전을 결정하였다. 이때 가중치를 로 설정하였다.

$$\text{가중치 고려 평균: } avg = \frac{x' + 2x_m}{3}$$

이후에 회전정도를 판단하기 위해서는 회전정도를$150 + \alpha(avg)$로 조절해야한다. 이 때는 이론적으로 구할 수 있는 값이 아니기 때문에 수차례의 실험을 통해서 조절했다.

2.2. 직선 주행 설계의 코딩에서의 구현

앞서 직선 주행 방법에 따라, 차선을 직선 방정식으로의 구현과 차선 양 끝 점의 중점을 코딩하는 것이 핵심이라고 할 수 있다. 우선 H2LD의 값과 H2RD의 값이 같을 경우 직진을 반환시킨다. 이 상황이 아닐 경우 본 알고리즘을 적용시킨다.

우선, 두 차선의 중점을 구하는 코딩은 H1RD와 H2LD 값을 통해 구할 수 있다. 중점을 midPoint 로 명명하여 코딩으로 구현하면 다음과 같다.

```
midPoint = (H2RD - H1LD)/2
```

그림 2. 두 차선의 중점을 구하는 코딩

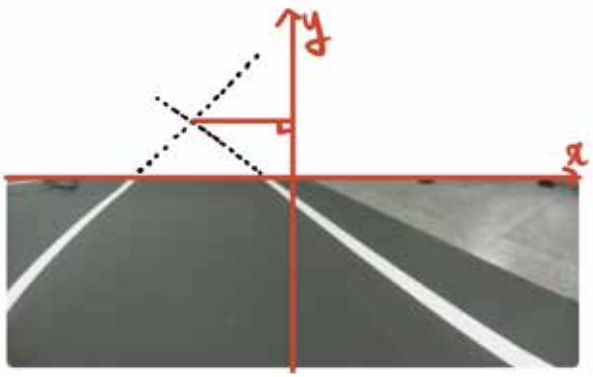


그림 3. 중앙선 조절 상황(2)

다음으로, 입력되는 영상에서의 두 차선을 직선 방정식으로 나타내고 교점을 구하는 과정은 총 n단계로 나눌 수 있다. 첫번째는 좌표계의 정의이다. 좌표계의 경우 입력되는 영상을 기준으로 중심을 (0,0)으로 설정하여 계산을 진행한다. 두 번째는 각각 차선이 지나는 점과 기울기의 값을 계산하는 것이다. 설정한 좌표계에 따라 왼쪽 차선 위의 두 점을 지정하는 코딩은 다음과 같다.

```
LP1x = -H1LD
LP1y = -30

LP2x = -H2LD
LP2y = -60

RP1x = H1RD
RP1y = -30

RP2x = -H2RD
RP2y = -60
```

그림 4. 왼쪽 차선 위의 두 점을 지정 하는 코딩

LP1x는 왼쪽 차선의 1번째 점을 의미한다. 각각 점의 약자는 (L또는R)+P+(번호)+(x 또는 y)형식으로 표기하였다. 이를 통해 두 차선 위의 점은 구해졌다. 다음으로 이 두 점을 활용하여 기울기를 구하는 코딩은 아래와 같다.

```
mL = (LP1y - LP2y)/(LP1x - LP2x)
mR = (RP1y - RP2y)/(RP1x - RP2x)
```

그림 5. 기울기를 구하는 코딩

이때 mL은 왼쪽 차선의 기울기, mR은 오른쪽 차선의 기울기이다. 이를 통해 두 차선을 설정한 좌표에서의 직선으로 생각하였을 때 지나는 점과 기울기가 모두 구해였다. 여기서 교점의 x 좌표를 Px 로 설정하고, 위에서 구한 일반해인

$x' = \frac{(m_r x_{r1} - m_l x_{l1}) - (y_{r1} - y_{l1})}{m_r - m_l}$ 를 이용하여 코딩으로 구현하면 아래와 같다.

```
Px = ( mL*LP1x - mR*RP1x + RP1y - LP1y ) / (mL - mR)

Tiltavg = ( 2*midPoint + Px ) / 3
```

그림 6. 교점의 X좌표를 구하는 코딩

따라서 최종적으로 구해지는 회전 정도인 Tiltavg는 위의 코딩을 통해 구해진다. 회전각에 대한 코딩에서의 구현은 모두 완료되었으므로, 이를 직접 자주차에 적용시켜 가장 적절한 K를 구하여 반영한다. 회전에 대한 반환값인 command까지 반영한 최종 직선 주행 알고리즘은 다음과 같다.

```
if sit == 1 :
    ### 직진 ###
    if left and right :
        if H2RD <= 0 and H3RD <= 0 :
            command = 'S1180E'

    elif V1D<=89:
        status = 'ONSTRAIGHT'
        print(status)

        midPoint = (H2RD - H1LD)/2

        LP1x = -H1LD
        LP1y = 30

        LP2x = -H2LD
        LP2y = 60

        RP1x = H1RD
        RP1y = -30

        RP2x = -H2RD
        RP2y = -60

        mL = (LP1y - LP2y)/(LP1x - LP2x)
        mR = (RP1y - RP2y)/(RP1x - RP2x)

        Px = ( mL*LP1x - mR*RP1x + RP1y - LP1y ) / (mL - mR)

        Ef = 0.1

        Tiltavg = Ef*(( 2*midPoint + Px ) / 3)

        Tilt = int(150+int(Tiltavg))

        command = ('S1%dE'% Tilt)
```

그림 7. 직선 주행 알고리즘

위 코딩에서 Ef의 값은 계산된 Tilt값에 곱해지는 상수 값으로, 시뮬레이션을 통해 최적화된 직선 주행을 위한 가장 적절한 값을 시뮬레이션을 통해 구하는 것을 계획하였다.

III. 결론

위에서 작성한 알고리즘은 상수 Ef에 따라서 회전하는 정도가 바뀐다. 그러나 이는 이론적으로 계산해낼 수 있는 값이 아닌 차가 다니는 트랙에 따라 달라지기 때문에 실험적으로 찾을 필요가 있었다. 따라서 다음과 같은 실험을 진행하여 시뮬레이션에 사용한 트랙에서의 최적의 Ef 값을 구하고자 하였다.

표 1. 교점의 X좌표를 구하는 코딩

조정값	결과
Ef = 0.015	조정이 필요한 상황에도 너무 Tilt값이 작게 산출됨을 관찰하였다.
Ef = 0.030	여전히 충분한 Tilt 값을 산출하지 못함을 확인하였다.
Ef = 0.045	적절한 Tilt 값에 근접하였지만 여전히 부족한 수준의 Tilt값을 산출하는 것을 관찰하였다.
Ef = 0.060	가장 적절한 Tilt 값을 산출함이 관찰되었다. 상황에 맞춰 돌아가야 할 각도에 맞게 회전값을 산출하였다.
Ef = 0.075	원래 돌아야 하는 각도보다 더 크게 명령을 산출함을 관찰하였다.
Ef = 0.090	너무 큰 각으로 회전하는 명령값을 산출함을 관찰하였다.

Ⅳ. 논의

본 연구는 해석 기하적인 방법을 알고리즘의 회전각 결정에 적용하여 직진 주행 안정화 알고리즘을 구현하였다. 자동차의 화면에 감지되는 선들의 기울기를 이용한 방법을 사용함으로써 실시간으로 차가 중앙선을 찾아가도록 하는 방법을 고안하는데 성공했다. 실험의 시뮬레이션에서는 안정적인 주행 명령을 내리는 적당한 상수 알파를 구해내는 것에 성공했지만 알고리즘에 들어가는 상수가 이론적으로 얻어지는 값이 아니라 반복적인 실험을 통해 도출되기 때문에 실험실을 벗어난 환경에서의 주행에서는 안정적이라고 볼 수 없다. 즉, 주변 환경 즉 트랙마다 적절한 상수 값이 다를 것이라 예상되므로 임의의 상황에서 안정적인 주행을 한다고 말하기는 어렵다. 따라서 사용자들이 각각 동일한 과정을 반복함으로써 상수를 구해야 한다. 그리고 상수의 범위를 더 좁게 조절함으로써 가장 안정적으로 직진 명령을 내리게 하는 알고리즘을 찾을 수 있을 것이다.

참고문헌

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

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

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

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

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

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

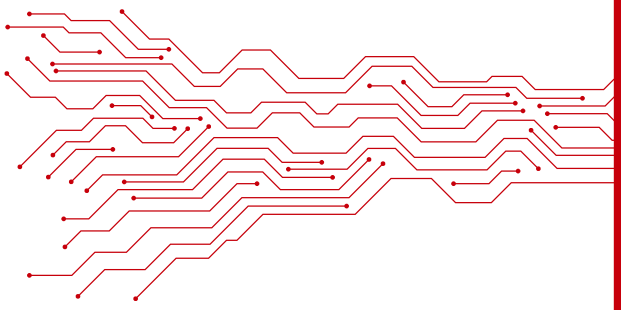
[7] C. Guo, J. Meguro, Y. Kojima, T. Naito, "A multimodal ADAS system for unmarked urban scenarios based on road context understanding", IEEE Trans. Intell Transp. Syst., 16(4), 1690-1704, Aug. 2015.

우수 논문

05 영상인식 기반 직진 주행 알고리즘에 대한 탐구

하나고등학교

최준혁, 박정우, 송노명, 김금평, 최은우, 윤준상



영상 인식 기반 직진 주행 알고리즘에 대한 탐구

A Research about Image Recognition-Based Straight Drive Algorithm

최준혁*, 박정우**, 송노명***, 김금평****, 최은우*****, 윤준상******
Junhyeok Choi, Jungwoo Park, Rhomyung Song, Keumpyeong Kim, Eunwoo Choi, Junsang Yun

요약

본 연구에서는 자율 주행 자동차의 주행 알고리즘, 차선 인식과 관련된 주행 개선 방안을 다루고, 카메라 캘리브레이션 과정에서의 관계식을 제안하였다. 본 연구에서 제안한 알고리즘은 직선주행에 특화된 알고리즘으로서, 실제 자율 주행 자동차가 주행 시 가중치 w에 따라 달라지는 값 t(w)를 다양하게 변화시키면서 적용하는 실험 과정을 통해 가장 최적의 가중치를 밝혀 낼 수 있었다. 본 연구에서 규명한 알고리즘을 적용한 결과, 데이터 상의 오류는 발견되지 않았고, 자주차의 영상 인식이 원활하게 이루어질 수 있었다.

Keywords: 자율 주행, 자동차, 캘리브레이션, 차선 인식, 알고리즘, 라이다 센서

I. 서 론

4차 산업혁명의 시대가 도래함에 따라 인공지능과 더불어 자율 주행 자동차는 점점 더 뜨거운 관심을 받고 있으며 이에 관련한 다양한 연구가 진행되고 있다. 이러한 자율 주행 자동차(이하 자주차)의 자율 주행 알고리즘을 설계하는 방법에는 여러 가지가 존재한다. 그 중에서도 가장 일반적인 프로세스는 외부 주행환경을 인식하고 이를 판단하여 주행전략을 수렴함으로써 차량을 제어하는 방법이다.

우선 자동차가 외부주행환경을 인식하는 과정에는 경로탐색, 고정 지물 인식, 변동 물체 인식 등 여러 가지가 있으며 자동차는 센서들을 이용해 이들과 관련된 데이터를 실시간으로 수집한다. 외부주행환경을 인식하고 나면 자동차는 여기에 대한 판단을 하고 상황에 맞는 주행전략을 수립해야 하는데, 구체적으로 회전이나 차선 변경 등과 같은 상황 판단 및 전략 수립을 위해서는 판단 및 제어시스템이 필요하다. 한국공학한림원으로부터 제공받은 자주차 키트는 카메라를 이용해 차선과 같은 고정 지물을 인식하며 라이다(Lidar) 센서로 변동하는 물체를 감지하여 운전자에게 보행자나 교통신호의 변화를 알려준다. 이와 같은 차량용 센서들을 기반으로 주행상황이 인지 되었다면 이에 알맞는 주행 시스템을 통해 자동차는 주행궤적을 생성한다. 본 연구에서는 위의 인지 시스템 중 카메라를 통한 차선 인식 방식에 대하여 연구하고 해당 방식을 통해 안정적인 직진주행 알고리즘 개발을 목적으로 하였다. 이에 따라 본 연구에서는 카메라를 이용한 차선 정보 인식에 대하여 그 구체적인 매커니즘을 알아보았으며, 좌우 차선 간의 거리의 차를 활용하여 보다 정확하고 안정적인 직선 주행 알고리즘을 고안하였다.

* 최준혁(하나고등학교, choijunhyeok781@gmail.com)
** 박정우(하나고등학교, jungwoopark03@gmail.com)
*** 송노명(하나고등학교, 20rks03@gmail.com)
**** 김금평(하나고등학교, rmavud123@gmail.com)
*****최은우(하나고등학교, soarhigh03@gmail.com)
****** 윤준상(하나고등학교, tim.yun.has@gmail.com)

따라서 본 연구에서는 주행 알고리즘, 차선 인식 등을 활용한 주행 개선 방안을 다루고, 카메라 캘리브레이션 과정에서의 관계식을 논한다. 마지막으로 해당 알고리즘 사용시의 유의점을 언급하고, 결론과 추후 개선점을 논하며 글을 마무리할 것이다.

II. 본론

직진 주행의 핵심은 중앙선에 있을 시 직진하며 중앙선에서 벗어나면 벗어난 정도에 따라 바퀴를 회전시켜 중앙선으로 돌아가는 데에 있다. 자주차 키트와 함께 나누어준 책에서는 경계값을 기준으로 중앙선을 벗어난 정도에 따라 중앙선을 많이 벗어난 경우, 중앙선을 조금 벗어난 경우, 두 개의 경우로 나누었고, 각각의 경우에 대하여 회전각을 상수로 정함으로써 직진 주행을 구현했다. 해당 알고리즘은 매우 간단하지만 상수로 정해진 회전각이 각각의 경우의 모든 값에 적합하다고 할 수 없었고 실제 주행 결과 이러한 점으로 인해 안정적인 직진 주행이 불가능했다. 해당 문제는 중앙선을 벗어난 정도(이하 오차)가 연속적인 값을 가지기에 발생하는 것이며 이에 따라 본 연구에서는 회전각을 오차에 따른 함수로 정의하여 상황에 가장 적합한 움직임을 취할 수 있도록 하였다.

2.1 Calibration

오차에 따른 함수로 회전각을 정의하기에 앞서 실제 우리의 주행이 이루어질 좌표계와 카메라로 보이는 좌표계 간의 상관관계를 파악하기 위해 관련해 탐구를 진행하였다. 카메라로 인식된 이미지는 캘리브레이션이라는 일종의 과정을 거쳐 보다 정확한 상을 보여주도록 수정된다.

보다 구체적으로, 카메라 캘리브레이션이란 3차원 점이 우리가 카메라를 통해 볼 2차원 영상에 투영된 위치를 구할 때, 영상을 왜곡하는 다양한 카메라 내부 요인을 제거하는 과정이다. 왜곡이 일어나는 요인에는 렌즈의 종류, 렌즈와 이미지 센서와의 거리, 렌즈와 이미지 센서가 이루는 각 등이 있다. 이 외에도 다양한 왜곡요인을 완벽히 제거하는 것은 매우 어렵기 때문에 이러한 값들을 구하는 과정을 거치는 것이다. 핀홀(pinhole) 카메라 모델이라 가정하면 변환 관계식은 다음과 같다.

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= A[R \mid t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

그림 1. 캘리브레이션의 행렬 변환

여기서 (X, Y, Z)는 World coordinate system 상의 3D 점의 좌표이며 A가 Intrinsic Camera Matrix, [R|t]가 좌표계의 변환에 사용되는 회전/이동변환 행렬이다.

위 행렬을 이용해 캘리브레이션 과정까지 고려해 회전각 관련 함수를 도출하기에는 카메라 자체에 대한 정보가 부족해 본 연구에서는 직접적으로 이용하지 않고 실험적으로 구한 오차에 따라 함수를 도출했다. 만약 영상 왜곡의 원인이 되는 카메라 내부 요소에 대한 정보를 얻을 수 있다면 카메라 영상 자체로부터 회전각 관련 정보를 도출해 이상적인 각을 연산할 수 있을 것이다.

2.2 Algorithm

우리는 다양한 회전각도를 변환시켜보며 시간을 최소하 시키는 지점을 찾았다.

```
1. algorithm.py
d=0, w=0.28 // d는 트랙 중앙(150)에서 벗어난 정도를 나타내는 difference 측정 변수. w는 가중치.
if left and right: // 직진 상황일 경우
        if debug: print('line is visible')
        d+=int(H2RD)-int(H2LD) // d값 계산. left와 right 거리 계산 후 트랙 내 위치 판단 (중앙 기준으로)
        r=int(150 +d*w) //가중치 w만큼 곱한 d값 기반으로 회전각도 값 r 을 산출 이 때 직진은 150도
        if (r>185 ):r=185 // 차체의 안정성을 위해  우측 최대 회전각도를 185도로 제한 했다
elif (145<r&&r<155)// 작은 오차는 큰 영향이 없고 속도를 저하시키기에 무시하고 직진하도록 하였다
        elif (r<115 ):r=115//차체의 안정성을 위해  우측 최대 회전각도를 115도로 제한하였다
        command=('S1%dE'%r)
```

2.3 그래프 분석

w값들을 나열하고, 그에 따른 t(w)를 도출한 결과, 다음과 같은 표가 만들어졌다. 이때 자료 3개가 추세선에서 크게 벗어나는 모습을 보여 "오류"로 분류하고 보다 정확한 측정을 위해 배제하기로 결정했다.

이렇게 도출된 그래프를 미분하여 얻은 도함수는 아래의 막대그래프의 모양과 같다. 최솟값을 구하기에 적합한 그래프로 막대그래프가 적절하다고 생각되었기에 다음과 같이 표현하였다. 완벽한 선형에서 분석하였다면 도함수가 0이 되는 지점이 존재하여야 한다. 하지만 주어진 수치 값을 분석하였기 때문에 도함수 값이 가장 작은 구간, w=0.282를 얻게 되었다. 수치 분석 방법은 다음과 같다.

(뒷 t(w)- 앞 t(w))/2*(뒷w-앞w)

이를 통해 구간 별 기울기를 구해냈고, 이 수치들을 이어서 다음과 같은 도함수 식을 도출했다.

표 1. 구간 별 기울기

w	0.028	0.074	0.1	0.141	0.192	0.212
t(w)	0.78	0.67	0.64	0.52	0.55	0.35
w	0.256	0.27	0.282	0.32	0.31	0.36
t(w)	0.3	0.81	0.24	0.29	0.66	0.38

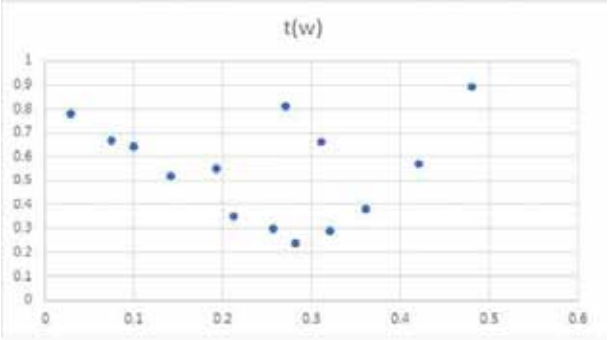


그림 2. 오차에 따른 t(w)의 그래프

표 2. t(w) 미분계수의 나열

w	0.028	0.074	0.1	0.141	0.212	0.256
t(w)	0.78	0.67	0.64	0.52	0.35	0.3
dy/dx		-0.97222	-1.1194	-1.29464	-0.95652	-0.78571
w	0.282	0.32	0.36	0.42		
t(w)	0.24	0.29	0.57	0.38		
y/dx	-0.07813	0.897436	1.4	2.125		

도출한 함수식은 다음과 같다

$$y = -55.002x^4 + 78.028x^3 - 24.664x^2 + 0.1616x + 0.7861$$

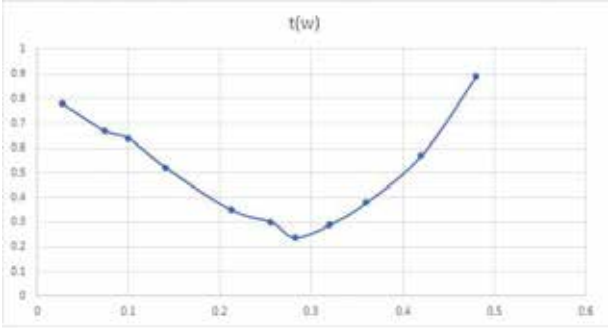


그림 3. 도출한 함수식의 그래프

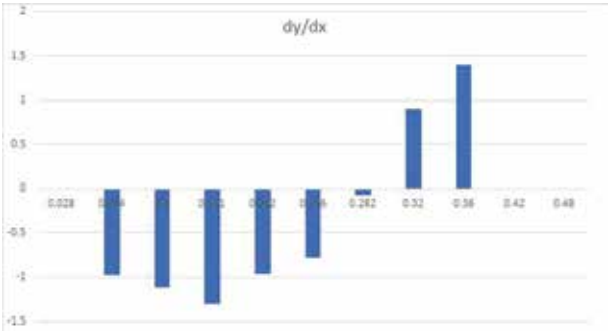


그림 4. 도함수의 막대그래프

2.4 해당 알고리즘 사용 시 유의점

자주차의 알고리즘을 구성하여 사용할 때 주의해야 할 사항은 다음과 같다. 우선 자주차의 조립 결과가 일치하지 않을 수 있기 때문에 위에서 구한 가중치는 늘 변동할 수 있음 염두해 두어야 한다. 특히, 본 연구를 진행함에 있어 사용된 트랙의 경우에는 SCORNER가 우측으로만 존재한다는 것이다.

이러한 트랙의 특성상 자주차의 알고리즘을 단축하기 위해 SCORNER의 좌측 회전은 구성하지 않았다. 즉, 효율적인 알고리즘 구성을 위해 트랙의 특성을 고려하였고, 이 때 만들어진 알고리즘을 모든 트랙에서 자율적으로 주행을 할 수 있는 것은 아니다. 특히, 가중치의 경우에는 주어진 트랙의 각도에 맞게 설계되어있기 때문에 다소 극단적일 수 있다. 실사에서 이용될 자주차의 경우에는 각 좌우 방향 코드를 모두 넣어야 하며, 더 안정적인 턴을 위해 위의 가중치를 줄여야 할 수도 있다.

둘째로는 제대로 된 알고리즘임을 확인하는 과정에서 다양한 에러를 이용할 수 있다. 에러 메시지는 알고리즘의 진행 순서에서 오류가 난 지점을 알려주는 역할을 한다. 예를 들어 &ModuleNotFoundError: No module named 'jajuchaUtil'&와 같은 에러가 났을 경우, 이는 첫 줄인 import의 과정에서 에러 난 것이다. 즉, 자주차와 컴퓨터가 연결되지 못했다는 것이므로, 자주차의 공유기를 reboot하거나 컴퓨터의 네트워크 관리자에 들어가서 정확한 ip주소로 자주차와 연결해야 한다. 이와 같은 방법으로 에러의 위치를 코드에서 확인한다면 하드웨어적, 소프트웨어적 문제를 해결하여 주행 알고리즘을 실험해볼 수 있을 것이다.

III. 결론

본 연구에서 개선할 점은 다음과 같다.일반적인 경우에는 알고리즘과 자주차의 주행 모두 정상적으로 작동하지만, 차체의 부품 등에 문제가 생기는 경우에는 이를 반영할 수 없다는 것이 가장 아쉬웠다. 연구를 진행하면서 자주차 운행을 많이 시험해보았는데, 알고리즘이 지닌 구조적 문제와 더불어 앞바퀴의 감도가 극단적으로 높거나 낮아서 발생하는 문제가 많이 발생했다. 이를 개선하기 위해 바퀴와 축의 조임 정도를 조절해보거나 운전 시 차량 바퀴가 돌아가는 감도를 향상시켜보는 등 다양한 시도를 해보았다. 그러나 초기부터 차체에 문제가 있는 경우, 알고리즘이 이를 매순간 반영할 수는 없었다는 점에서 아쉬움이 남는다.

또한 본 연구에서 주로 다룬 알고리즘은 직선주행에 특화된 알고리즘이고 실제 자율 주행 자동차가 주행 시 분석해야 할 도로는 직선 도로 이외에도 수백 가지 있는 것을 고려하면, 특정 경우에만 적용되는 제한적인 연구를 한 것이 아쉽다.

가중치 w에 따라 달라지는 값 t(w)를 수 차례에 걸쳐서 구하는 과정을 통해 가장 최적의 가중치를 찾을 수 있었다. 기계적 결함 또는 기타 이유에 의해서 발생한 오차값이 약 3개 정도 발생했는데, 이는 제외하고 w-t(w) 관계를 분석하였다. 발생한 오류의 시간 측정값이 기존의 가중치에 비해서 지나치게 큰 값을 지녔다고 판단하였고, 이는 가중치 및 알고리즘의 문제보다 차체의 센서 및 인식 단계에서 발생한 문제라고 생각했기 때문이다.

자주차의 영상 인식 자체가 차체가 안정적으로 주행할 수 있기 위해서는 가장 중요한 것이 외부환경 인식이며, 이는 자주차에서 영상 인식을 하는 것에서부터 시작한다. 이때 영상 인식 자체의 기술적 문제에 의하여 알고리즘의 분석 데이터가 처음부터 오류가 발생하면, 안정적 주행에 있어서 문제가 생길 수 있다. 본 연구에서 다룬 알고리즘의 분석 데이터 상의 오류는 발견되지 않았고 자주차의 영상 인식이 원활하게 이루어질 수 있었다.

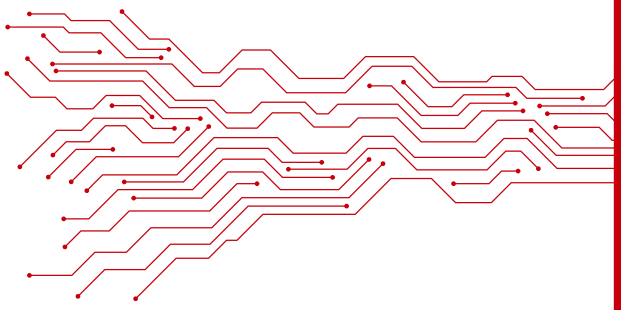
참고문헌

[1] 이병윤(2016), 국내외 자율 주행 자동차 기술개발 동향과 전망, 한국통신학회지(정보와 통신), 33(4), 10-16
[2] 청년공학 제 3집, 한국공학한림원(2019.09)

우수 논문

06 LiDAR 센서를 이용한 전방향 장애물 탐지 알고리즘의 개발

인천연송고등학교 김민구, 박주환, 백승범



LiDAR 센서를 이용한 전방향 장애물 탐지 알고리즘의 개발

Development of Omnidirectional Obstacle Detection System Using LiDAR Sensor

김 민 구*, 박 주 환**, 백 승 범*** Min-gu Kim, Ju-hwan Park, Seung-beom Baek

요약

기존의 자율 주행 자동차는 전방의 장애물만 인식하며, 장애물을 인식하면 정지하도록 제어되어 몇몇 특수한 상황만 대응가능하다는 단점이 있다. 본 연구에서는 장애물을 인식하는 센서인 LiDAR 센서를 차량의 다방면에 달아 다양한 방향에서 오는 장애물을 인식하도록 한다. 또한 전방, 측면, 후방으로 경우를 나누고, 미분을 이용하여 속도를 계산하는 등의 방법을 이용하여 장애물을 회피하는 알고리즘을 개선해 더욱 다양한 상황에 대처할 수 있도록 한다.

Keywords: 자율 주행 자동차, LiDAR, 장애물 회피

I. 서 론

2020년 7월부터 한국에서 레벨3 수준의 자율 주행 자동차의 출시, 판매가 가능해진다. 기존 안전 기준 상의 첨단조향장치(레벨2)는 운전자를 지원하는 수준의 기능이었지만, 이번에 안전 기준이 도입된 부분 자율 주행(레벨3)수준의 시스템은 특정한 영역에서는 자동차의 핸들에서 손을 떼어도 지속적인 차로 유지 및 자율 주행이 가능하다. 이와 같이 자율 주행 자동차와 관련된 기술은 점점 발전되어 가고 있고, 그에 따라 제도도 점차 만들어지고 있다.

현재 우리가 제작한 자율 주행 자동차는 도로를 감지하여 주행할 수 있다는 점은 실제 자율 주행차와 같지만, LiDAR 센서가 전면에만 달려있어 측면과 후방의 장애물을 탐지하기 힘들다. 실제 자율 주행차를 보면 360도로 장애물을 인식하는 기능이 탑재되어 있어 전방 이외의 다른 방향의 장애물도 탐지하여 회피할 수 있다.

우리는 제작한 자율 주행 자동차의 이런 한계점을 인식하여, 실제 자율 주행 자동차에 360도로 장애물을 인식하는 장치가 있다는 것을 이용해 다양한 방면에 LiDAR 센서를 부착함으로써 측면과 후방의 장애물을 탐지하기 어렵다는 단점을 개선하고자 한다.

* 김민구(인천연송고등학교, mingu_02@naver.com)
** 박주환(인천연송고등학교, matt1221@naver.com)
*** 백승범(인천연송고등학교, 02bsb777@naver.com)

II. 본론

1. 기본적 원리

1.1 LiDAR 센서

LiDAR(LiDAR)는'Light Detection And Ranging(빛 탐지 및 범위 측정)의 약자로 센서 시스템의 구성은 응용 분야에 따라 때로는 매우 복잡하게 구성되지만, 기본적인 구성은 (그림 1)에 보인 바와 같이 레이저 송신부, 레이저 검출부, 신호 수집 및 처리와 데이터를 송수신하기 위한 부분으로 단순하게 구분될 수 있다.

아울러 LiDAR 센서는 레이저 신호의 변조 방법에 따라 time-of-flight(TOF) 방식과 phase-shift 방식으로 구분될 수 있다. 본 자율 주행 자동차 연구에서 사용하는 TOF 방식은 (그림 1)에 보인 바와 같이 레이저가 펄스 신호를 방출하여 측정 범위 내에 있는 물체들로부터의 반사 펄스 신호들이 수신기에 도착하는 시간을 측정함으로써 거리를 측정하는 것이 가능하다.

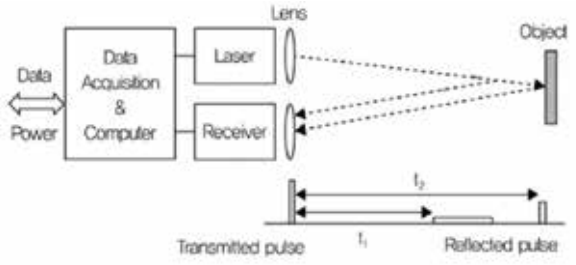


그림 1. LiDAR 센서의 원리

LiDAR 센서의 중요한 특성 중 하나는 센서의 측정 각도를 나타내는 Field Of View (FOV)에 따라 시야의 범위가 한정된다는 점이다.

1.2 속도 측정 원리

LiDAR 센서를 이용하여 속도를 측정하기 위해서는 미분의 원리가 들어간다. 어떤 물체의 순간적인 움직임을 계산하는 것이 미분인데, 여기서는 LiDAR 센서를 이용해 물체의 위치를 측정하고, 이를 이용해 물체의 속도를 계산한다.

t초일 때 LiDAR 센서가 감지한 값을 f(t)라고 할 때, 이를 이용하여 속도를 계산하는 식은 $\lim_{h\to 0}\frac{f(t+h)-f(t)}{h}$가 된다.

1.3 자율 주행 자동차 기본 코드

```
if status is ONSTRAIGHT:
    if 1 < LiDAR and LiDAR < 300:
        command = 'S0150E'
    elif left and right:
        if H2LD != 160 and H2RD != 160:
            centerPoint_X = (320 - H2LD + H2RD)/2
        elif H3LD != 160 and H3RD != 160:
            centerPoint_X = (320 - H3LD + H3RD)-67
        if centerPoint_X > center+5:
            if centerPoint_X > center+20:
                command = 'S1170E'
            else:
                command = 'S1160E'
        esfdflif centerPoint_X < center-5:
            if centerPoint_X < center-20:
                command = 'S1130E'
            else:
                command = 'S1140E'
        else:
            command = 'S1150E'
    if H1RD is -1 or H2RD is -1:
        status = ONRIGHT
    elif H1LD is -1 or H2LD is -1:
        status = ONLEFT
    elif H2LD is not -1 and H2RD is not -1:
        status = ONSTRAIGHT
```

그림2. 직진 코드

```
elif status is ONLEFT:
    if right and not left and end:
        if V4D <= 75:
            command = 'S1125E'
        if V3D is -1 and V7D is not -1:
            command = 'S1130E'
    elif left and right:
        if (H2LD > 60 and H2RD > 60) or (H3LD > 65 and H2RD > 65):
            command = 'S1150E'
            status = ONSTRAIGHT
    elif end:
        command = 'S1125E'
```

그림3. 좌회전 코드

```
elif status is ONRIGHT:
    if not right and left and end:
        if V4D <= 75:
            command = 'S1175E'
        if V5D is -1 and V1D is not -1:
            command = 'S1170E'
    elif left and right:
        if (H2LD > 60 and H2RD > 60) or (H2LD > 65 and H3RD > 65):
            command = 'S1150E'
            status = ONSTRAIGHT
    elif end:
        command = 'S1175E'
```

그림4. 우회전 코드

위와 같이 기존 자율 주행 자동차의 코드는 직진상황에서의 전방 감지 시스템만 있으므로 다른 방향의 장애물에 대해 취약하다.

2. 알고리즘 제작

2.1 전방 장애물

그림5. 전방 장애물

자율 주행 자동차가 진행하는 도중 전방에 장애물이 감지되었을 경우, 그 장애물은 정지하고 있거나 앞으로 다가오고 있을 수 있다. 이때, 정지한 장애물일 경우 기존의 코딩으로 문제를 해결할 수 있지만, 앞으로 다가오는 장애물의 경우 기존의 코딩으로 문제를 해결할 수 없다. 따라서 이때는 새로운 해결 방법이 필요하다.

앞으로 다가오는 장애물을 피하기 위해서는 후진하는 방법과 옆으로 피하는 방법이 있을 것인데, 후진을 하게 되면 계속 장애물이랑 같은 선상에서 운동하는 것이 되어버리므로 효과적이지 않다. 따라서 측면으로 피하는 방법이 가장 효과적일 것이다.

```
if LiDAR_front < 300:
    if LiDAR_left > 300 and LiDAR_right < 300:
        command = 'S1130E'
    elif LiDAR_left < 300 and LiDAR_right > 300:
        command = 'S1170E'
    elif LiDAR_left < 300 and LiDAR_right < 300:
        command = 'S2150E'
    else:
        command = 'S1170E'
```

그림6. 전방 장애물 회피 알고리즘

장애물을 피해 측면으로 이동하기 위해서는 우선 측면 장애물의 유무를 인지해야 한다. 이를 위해 좌측과 우측의 LiDAR 센서를 이용하였고, 장애물이 없는 쪽으로 회피하도록 하였다. 이때, 양쪽에 장애물이 있다면 계속 후진하되, 측면 장애물의 유무를 계속 탐지하여 가능한 한 빠르게 측면으로 피할 수 있게 하였다. 그리고 양쪽 다 장애물이 없다면, 우측통행인 우리나라의 특성상 좌측으로 가면 중앙선을 넘을 수 있기 때문에 우측으로 회피하도록 하였다.

2.2 측면 장애물

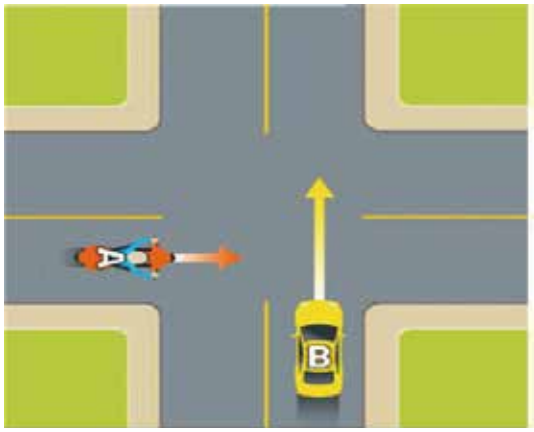


그림7. 측면 장애물

자동차의 진행방향과 수직으로 장애물이 오는 경우에는 회전하여 피하는 것보다 빠르게 움직여서 앞으로 피하는 것이 효율적이다. (자율 주행 자동차에 속도조정기능을 추가하였다고 가정한다.) 이때 장애물을 피하기 위해 무조건 최고 속도로 움직이게 되면, 너무 빠른 속도로 인한 사고가 추가적으로 발생할 가능성이 있다. 그러므로 장애물이 차량쪽으로 다가오는 속도를 계산해, 이에 맞춰서 속도를 내야 할 것이다.

$$v = \frac{L_C}{\frac{L}{\lim_{h \to 0} \frac{f(t+h) - f(t)}{h}}}$$

수식1. 속도 계산식(v = 장애물을 회피하기 위해 필요한 최저 속도, Lc = LiDAR 센서와 자율 주행 자동차 뒷부분 사이의 거리, L= 장애물과 자율 주행 자동차 사이의 거리)

장애물과 차량 사이의 거리를 LiDAR 센서를 이용해 파악하고, 거리의 변화량을 이용하여 속도를 구해 몇 초 후에 차량과 장애물이 충돌할지 계산한다. 그 후 충돌을 피할 수 있을 정도의 속도를 내서 장애물을 회피한다. 이때 계산과정에 오차가 있거나 다른 변수가 존재할 수 있으므로 충돌을 피할 수 있는 속도보다 약간 더 빠른 속도으로 주행해야 한다.

2.3 후방 장애물

그림8. 후방 장애물

자율 주행 자동차가 주행하는 도중 후방에 장애물이 감지된다면 그 장애물은 자율 주행 자동차와 같거나 그보다 빠른 속도로 진행하고 있는 것이기 때문에, 이를 회피하기 위해서는 더 빠른 속도로 나아가거나, 전방의 장애물을 인식했을 때와 같이 측면으로 빠져야 할 것이다.

```
if LiDAR_back < 300:
    if LiDAR_left > 300 and LiDAR_right < 300:
        command = 'S1130E'
    elif LiDAR_left < 300 and LiDAR_right > 300:
        command = 'S1170E'
    elif LiDAR_left < 300 and LiDAR_right < 300:
        command = 'S1150E' #빠른 속도
    else:
        command = 'S1170E'
```

그림9. 후방 장애물 회피 알고리즘(속도 조절 가능할 경우)

장애물을 피해 측면으로 이동하기 위해서는 우선 측면 장애물의 유무를 인지해야 한다. 이를 위해 좌측과 우측의 LiDAR 센서를 이용하였고, 장애물이 없는 쪽으로 회피하도록 하였다. 그리고 양쪽 다 장애물이 없다면, 우측통행인 우리나라의 특성상 좌측으로 가면 중앙선을 넘을 수 있기 때문에 우측으로 회피하도록 하였다. 만약 양쪽에 장애물이 있다면 계속 빠른 속도로 전진하되, 측면 장애물의 유무를 계속 탐지하여 가능한 한 빠르게 측면으로 피할 수 있게 하였다. 이때 속도는 LiDAR 센서 값이 줄어들지 않는 속도에 도달할 때까지 가속한다.

III. 결론

위와 같은 연구를 통해 다양한 방향의 장애물을 탐지하고 회피하는 알고리즘을 개발함으로써 기존 자율 주행 자동차보다 더욱 다양한 상황에 대처할 수 있게 되어 실제 도로 상황에 적용할 수 있는 가능성이 높아졌다.

본 연구에서는 연구의 복잡도를 낮추기 위해 전방, 좌측, 우측, 후방 중 한 방향에서만 장애물이 온다고 가정하였기 때문에, 다양한 방향에서 오는 장애물에 대해 대처하기 어렵다는 한계가 있다. 또한 LiDAR 센서의 한계로 상황을 자세하게 인식하지 못하므로, 카메라를 사용하는 등의 방법을 보완이 필요하다. 이와 관련하여 추가적인 연구를 하게 된다면 더욱 안정적인 장애물 회피 알고리즘을 설계할 수 있을 것이다.

참고문헌

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

[2] 윤태진, 김민구, 김민, 문동호, 이상학. (2020). SLAM알고리즘과 LiDAR를 이용한 자율 주행 로봇 개발. 한국컴퓨터정보학회 학술발표논문집 , 28(1), 289-290.

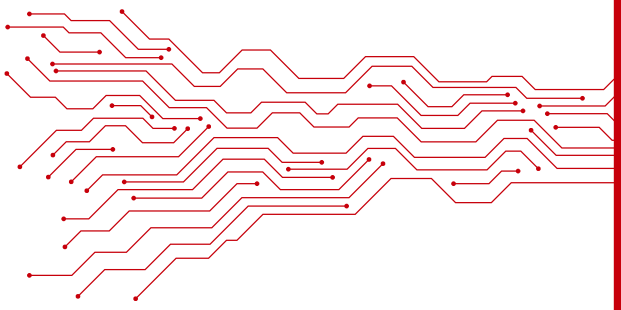
[3] 강병준, 김종원. (2018). 다양한 자율 주행 이동체에 적용하기 위한 장애물 회피의사 결정 시스템 연구. 한국산학기술학회 논문지, 19(6), 639-645.

[4] 김미선, 나유승, 박재성, 손희원, 김연욱, 박정근. (2020). 자율 주행을 위한 적외선 센서를 이용한 장애물 회피 기법 구현. 한국통신학회 학술대회논문집, (), 105-105.

우수 논문

07 차선의 기울기를 이용한 곡선 주행 알고리즘 개발

서울 오산고등학교 이정민, 정재욱, 정재원



차선의 기울기를 이용한 곡선 주행 알고리즘 개발

Curved driving algorithm design using lane slope

이 정 민*, 정 재 우**, 정 재 원*** Lee Jeong-min, Jeong Jae-Woo, Jeong Jae-Won

요약

이 연구에서는 기존의 자율 주행 자동차 주행 알고리즘이 갖는 문제점에 대해 분석하고 그 문제를 해결하기 위해 차선의 좌표를 이용해 기울기를 계산한 후, 기울기의 부호에 따라 우회전의 유무를 결정하도록 하는 알고리즘을 이용해 그 문제를 해결하였다.

Keywords: 자율 주행 자동차, 차선, 기울기, 알고리즘, 곡선 주행

I. 서 론

제4차 산업혁명은 만물 초지능 혁명이다. 이제는 일상생활 속에 사용되는 사물들이 사물 인터넷(IoT)이나 인공지능(AI) 등이 적용되어 지능을 갖고, 통신 인프라가 구축되어 모든 기기들이 연결되는 사회가 도래할 것이다.

그 중 주목받는 핵심 산업으로 자율 주행 자동차가 있다. 자율 주행 자동차(Self-Driving Car)는 말 그대로 스스로 도로 위를 주행하는 차이다. 자율 주행 자동차는 인공지능, 5G 네트워크, 센서 기술 등 최신 기술들의 집합체로 일컬어진다. 5G네트워크의 발전으로 V2X(Vehicle to Everything) 기술이 개발되고, 여러 센서들과 인공지능은 각각 팔과 다리, 그리고 뇌 역할을 하게 될 것이다. 가까운 미래에는 이러한 기술들이 발전되어 모두가 직접 운전하지 않고 도로를 누비는 날이 올 것이라고 예상된다.

현재 시중에서 판매되는 자율 주행 자동차는 2단계에 해당되며 LKAS(Lane Keeping Assist system)라 불리는 차선유지시스템과 고속도로와 같은 환경에서 차선을 유지하며 속도를 규정 속도에 자동 조정되는 ACC(Adaptive Cruise Control)도 장착되어 있다. 그러나 가까운 미래에는 4~5단계의 자율 주행 자동차가 개발되어 전세계 곳곳을 누빌 것으로 예상된다.

그리고 이러한 자율 주행은 '인식-판단-제어 기술'을 통해 실현된다. 인식 기술은 센서를 이용하여 차량의 위치, 속도 등을 파악하는 자차위치인식기술과 주변의 다른 차량 및 신호등, 보행자 등을 파악하는 주변인식 기술로 구분된다.

자차위치인식기술로 현재에는 보편적으로 GPS를 이용하고 있지만, 즉각적인 정보 변화를 정확히 인식해야하는 자율 주행 자동차에 경우에는 적합하지 않다. 따라서 현재 자율 주행 자동차는 LiDAR(Light Detection And Range) 센서와 카메라를 많이 사용하고, 아래와 같이 SLAM(Simultaneous Localization And Mapping), 즉 자율 주행용 지도가 연구 되고 있다.

인식 기술을 통해 획득한 데이터를 기반으로 하여 자율 주행 자동차는 수행할 행동을 판단한다. 특히 이러한 과정을 수행할 딥러닝 기반의 물체의 형상을 구분하는 시멘틱 세그멘테이션 기술에 대한 연구가 활발히 진행되고 있으며, 세부 지도를 바탕으로 차선이나 목적지까지의 경로 등을 만드는 기술 또한 개발되고 있다. 이러한 판단 기술에는 크게 3가지 주요 기능이 있다.

첫째, 움직이는 장애물을 정확히 인식한 후 위험 상황을 분석하는 것이다. 둘째, 위험도를 계산한 후 그에 따른 주행 전략을 수립한다. 세 번째, 어떤 경로를 통해 주행할 것인지와 속도를 계획한다. 이러한 3가지 기능을 바탕으로 자율 주행 자동차는 스스로 주행상황을 판단하여 사람들을 안전하게 목적지까지 이동시켜 준다.

하지만 자율 주행 자동차가 자율 주행만을 하는 것은 아니다. 필요에 따라서 분명히 운전자가 직접 운전하는 상황이 필연적으로 발생할 것이다. 그에 따라 자율 주행 자동차는 제어 플랫폼 기술을 지니고 있는데, 한국전자통신연구원(ETRI)에서 개발한 제어 플랫폼을 바탕으로 설명하면 다음과 같다.

그림 1. ETRI 자율 주행 자동차 플랫폼

* 이정민(오산고등학교, junsu01041526634@gmail.com)
** 정재우(오산고등학교, dldyddnr4285@gmail.com)
*** 정재원(오산고등학교, cjy6275@naver.com)

ETRI 자율 주행시스템은 크게 운전자 주행 모드와 자율 주행 모드를 가진다. 더 자세히 설명하자면 Manual, AutoReady, SystemFail 일 때는 운전자 주행 모드로 주행하며, Auto 상태일 때만 자율 주행 모드로 작동하는 것이다. 즉, 시스템에 조금이라도 문제가 있다면 운전자주행 모드로 전환하는 시스템을 갖추고 있다. 이렇듯 자율 주행 자동차는 상황에 따라서 자동차 시스템을 제어하며, 상황에 맞게 자율 주행 모드에 돌입하여 토크나 조향각을 스스로 조절하는 것이다.

우리가 제작한 자율 주행 자동차는 위에서 언급한 모든 기술을 적용시키기엔 한계가 존재한다. 따라서 우리는 자율 주행 자동차가 반드시 지녀야할 핵심적인 기술을 위주로 적용시킨 자동차를 제작하여 주행시켜 보았다. 자율 주행 자동차는 핵심 기술이라 할 수 있는 인지 기술과 판단 기술을 위주로 적용시켰다. 비록 세부 지도나 신호등 및 장애물 간 통신은 구현되지 않지만, 인식 기술로서 LiDAR 센서를 기반으로 하여 장애물을 감지하고, 카메라를 통해 차선 유지 및 신호등 인식을 수행한다. 또한 판단 기술은 우리가 작성한 파이썬 주행 코드와 제공된 신호등 딥러닝 데이터 세트를 기반으로 작동된다. 이를 기반으로 우리는 직선 도로 주행, 곡선 주행, 장애물 인식, 신호등 인식 등 도로 주행 중 발생할 수 있는 상황을 분류하여 각각의 알고리즘을 구축하였다.

Ⅱ. 본 론

1. 직진 주행(중앙값 맞추기)

직진 상황에서 자율 주행 자동차가 안정적으로 주행하기 위해서는, 스스로 차체를 차선의 중앙으로 위치시키는 시스템이 필요하다. 이 원리를 알기 전에 기본적으로 알아야 할 몇 가지가 있다.

'S1150E'

첫째로 이 코드는 자동차가 직진하도록 해준다. 이와 여기서 뒤의 숫자 50이 더 큰 숫자를 가질수록(51~90) 더욱 우회전하고(우회전하는 정도가 크고) 더 작은 숫자를 가질수록 (10~49) 더욱 좌회전한다(좌회전하는 정도가 크다). 또한 'S0150E'는 정지, 'S2150E'는 후진을 뜻한다. 다음은 H#RD 혹은 H#LD 값을 읽는 방법이다.

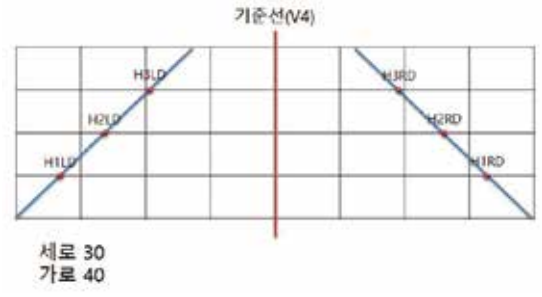


그림 2. H#RD 혹은 H#LD 값을 읽는 방법

H#RD 혹은 H#LD은 차선과 가로선과의 교점을 나타낸 데이터이다. H1, H2, H3 선은 각각 가장 아래에 있는 가로선, 중간 가로선, 가장 위에 있는 가로선을 뜻한다. R과 L은 각각 기준선(V4) 에서 오른쪽에 위치한 점인지, 왼쪽에 위치한 점인지를 나타내는 표시이다.

H3LD는 차선과 H3선과의 교점들 중 기준선을 기준으로 왼쪽에 위치한 점을 나타내고 이 데이터 값은 기준선과 이 점 사이의 거리로 정의한다. 즉, 위 그림에서 H3LD 는 60~80 사이의 값을 갖고, H1RD는 120~140 사이의 값을 갖는다. 이제 중앙값을 맞추는 원리에 대해 알아보자.

그림 3은 차체가 차선의 중앙에 위치해있을 때의 모습이다. center는 이 사진 자체에서 왼쪽 모서리로부터 V4까지의 거리를 의미한다. 즉 center 값은 항상 160으로 고정되어있다.

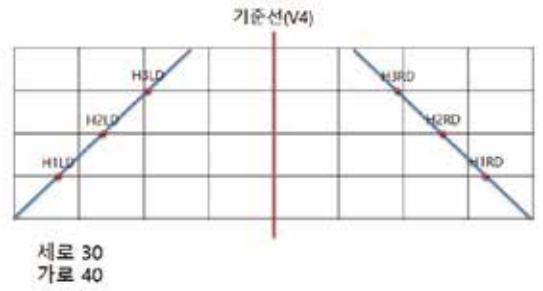


그림 3. 차체가 차선의 중앙에 위치해있을 때

centerPoint_X는 카메라에서 나타나는 실제 차선의 중앙의 좌표를 나타내고 이는 다음과 같이 나타낼 수 있다. (H2LD, H1LD, H2RD, H1RD 데이터 대신 H3LD, H3RD 값을 사용하였다)

{(160-H3LD)+(160+H3RD)}/2

160-H3LD는 H3LD점이 왼쪽 모서리로부터 떨어진 거리를 의미하고 160+H3RD 는 H3RD점이 왼쪽 모서리로부터 떨어진 거리를 의미한다. 즉, {(160-H3LD)+(160+H3RD)}/2는 H3LD H3RD 두 점의 중앙값이 카메라의 왼쪽모서리로부터 떨어진 거리를 의미한다.

그림 4는 centerPoint_X < center 인 상황이다. 이 때는 차가 차선의 중앙의 오른쪽에 위치해 있는 경우로 좌회전해주어야 한다.

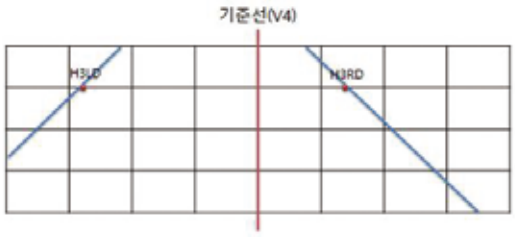


그림 4. centerPoint_X < center인 상황

그림 5는 centerPoint_X > center 인 상황이다. 이 때는 차가 차선의 중앙의 왼쪽에 위치해 있는 경우로 우회전해주어야 한다.

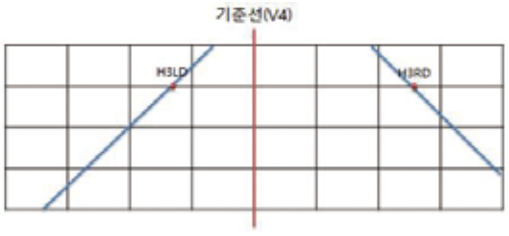


그림 5. centerPoint_X > center인 상황

이와 같은 원리로 다음과 같은 코드를 제작하였다

```
if debug:
    print('center point calculated by H2 points %d, %d' %(H3LD, H3RD))
    centerPoint_X = (920 - H3LD + H3RD)/2
    print('center = %d' %center)
    print('center point = %d' %centerPoint_X)
if centerPoint_X > center + 5:
    if centerPoint_X > center + 20:
        command = 'S1165E'
    else:
        command = 'S1157E'
elif centerPoint_X < center - 5:
    if centerPoint_X < center - 20:
        command = 'S1135E'
    else:
        command = 'S1143E'
else:
    command = 'S1150E'
```

그림 6. centerPoint_X > center인 상황에서의 코딩

그림 6에서 centerPoint_X와 center 값이 조금 차이 나는 정도는 무시할 수 있도록 해주기 위해 5mm 정도의 여유를 두었다. (centerPoint_X > center +5, centerPoint_X < center -5로 실현된다)

또한 centerPoint_X와 center 값이 많이 차이 날 경우에는 더욱 크게 꺾어야 하기 때문에 이에 대한 상황을 따로 정의 해 두었다. (centerPoint_X > center +20, centerPoint_X < center -20로 실현된다)

2. 장애물 인식과 정지

이는 라이다 센서로 실현된다. 라이다 센서는 적외선을 쏘아 적외선이 물체에 반사하여 돌아오는 시간을 계산하여 물체와의 거리를 측정한다. 장애물 인식과 정지는 아래와 같은 코드로 실현된다.

```
if 1< LiDAR and LiDAR < 300:
    print('Obstacle Detected at %dmm' %LiDAR)
    command = 'S0150E'
elif LiDAR >= 300 or LiDAR == 0:
```

그림 7. 장애물 인식과 정지 코드

LiDAR 값이 300 이내일 때 (물체와의 거리가 30cm 이내일 때) 'S0150E'를 명령하여 정지시킨다. LiDAR 값이 300 보다 크거나 0일 때 (0인 경우는 물체와의 거리가 너무 커 LiDAR 센서가 그 값을 불러오지 못한 경우이다) elif 이하의 알고리즘을 실행한다.

3. 신호등 인식

먼저 카메라로 신호등을 인지한다. 검정색 박스형태의 물체를 모두 신호등으로 인식하도록 딥러닝을 통해 학습되어있다. 다음으로, 신호등의 색 변화를 인지해 초록불과 빨간불을 인지한다. 색 자체를 인지하지는 못하고 색의 밝기만을 인식할 수 있다. 즉, 컬러의 그림을 흑백으로 전환하여 명도를 파악한다. 이에 따라 초록불 부분(신호등의 오른쪽)의 밝기가 높아졌을 때 초록불이 켜진 상태로 인식해 True로 데이터를 처리하고, 이외의 것들은 모두 False로 데이터를 처리하도록 되어 있다. 신호등 처리 코드는 다음과 같다.

```
if left and right:
    status = ONSTRAIGHT
    if H2LD != 160 and H2RD != 160:
        if light == 10:
            print('Traffic light = RED')
            command = 'S0150E'
        elif light is False or light is True:
```

그림 8. 신호등 처리 코드

Left 차선과 right 차선이 모두 존재하는 경우를 직진상황으로 인식한다. 직진 상황일 때 인식한 신호등의 오른쪽 부분에서 충분한 명도의 빛이 인식되면 초록불로 인식해 직진을 유지하고, 나머지 경우에는 빨간불로 인식하고 정지한다.

4. 곡선 주행

우리는 자율 주행 자동차 기본 트랙에서 마주치는 곡선 주행의 유형을 크게 2가지로 분류하여 알고리즘을 작성하였다. 첫 번째로 직진 도중 좌회전하는 경우와 두 번째로 S자 커브를 돌아야하는 상황으로 분류하였다. 이는 각각 주행 트랙에서 6번과 1번씩 나타났다. 이를 판단하기 위해 실제 주행 상황 중의 카메라 데이터를 분석하여 각 유형 별 공통적인 특징을 추출해내는 데 집중하였고, 기준으로 카메라에 인식되는 차선으로 잡았다. 이에 따라 우리는 처음에 왼쪽, 오른쪽, 앞쪽 차선의 유무를 통해 각 상황을 구별하며 알고리즘을 작성하였다.

첫 번째 일반적인 좌회전 상황은 if문을 활용하여 카메라를 통해 인식된 차선 중 오른쪽 차선과 앞 차선만 있을 때 위 상황으로 분류하도록 하였다. 그런 후 왼쪽 차선이 인식될 때까지 서브모터를 왼쪽 회전시키는 'S1122E'를 명령을 내렸다.

그리고 조향각을 크게 해야 하는 두 번째 S자 코스 상황은 인식된 차선 중 왼쪽 차선만 있을 때 인지하도록 if문을 통해 프로그램을 작성하였다. S자 코스임을 자율 주행 자동차가 인식한다면 'S1168E' 커맨드를 주어 오른쪽으로 크게 회전할 수 있도록 하였다.

하지만 기존의 곡선 주행 알고리즘을 이용하면 카메라 각도에 따라 좌회전 상황과 우회전 상황을 혼동하는 오류가 발생한다. 이는 단순히 인식되는 좌표만을 이용하여 차선의 유무만 가지고 판단하기 때문에 생기는 문제인데, 우리는 이 문제를 해결하기 위해서 각 차선의 좌표를 이용하여 차선의 기울기를 알아내어 이를 이용했고, 그 결과 위의 문제를 해결할 수 있었다. 자율 주행 자동차가 좌회전하고 난 직후, 다시 차선의 중앙값을 맞추어 직선 주행을 시작해야 하는데 회전하는 과정에서 일시적으로 오른쪽 차선이 카메라의 왼쪽에 잡혀 왼쪽 차선으로 인식되는 경우에 문제가 발생했다. 이 경우 카메라의 왼쪽에 잡힌 오른쪽 차선의 기울기는 직선 주행 과정에서 인식되는 실제 왼쪽 차선의 기울기와는 차이가 있으므로 이 점에 주목하여 문제를 해결하였다. 직선 주행 과정에서 잡히는 실제 왼쪽 차선의 기울기는 양의 값을 갖는다. 하지만 좌회전 직후 카메라의 왼쪽에 잡히는 오른쪽 차선의 기울기는 음수 값을 갖는다. 이 점에 주목하여 우리는 차선의 기울기를 얻기 위해 H1LD, H2LD, H3LD 값을 이용했다. H1LD, H2LD, H3LD의 높이는 각각 고정되어있으므로 그 값만 이용하여 각각의 기울기를 계산할 수 있다. 예를 들어 H1LD 값이 100이고 H3LD 값이 40이라 하면, 카메라의 중앙을 기준으로 (-100, 30)과 (-40, 90)을 이은 직선의 기울기를 구하는 것으로, 이 경우에 기울기는 (90-30)/(-40+100)=1로 양의 값을 갖는다. 하지만 좌회전 직후 문제가 생기는 상황을 가정하여 H1LD 값이 40이고 H3LD 값이 100인 경우 기울기는 -1이 되므로 음의 값을 갖는다. 하지만 이 과정을 알고리즘에 대입하여 실제로 작동시켰을 때, 가끔 정상적으로 작동하지 않는 경우가 있었다. 그 이유는 카메라가 차선을 인식하는 과정에서 차선이 카메라에서 벗어나 종종 H1LD나 H3LD 값이 없는 경우 때문이었다.

우리는 이 문제를 해결하기 위해서 H1LD와 H3LD 값만을 이용하는 것이 아니라 H2LD와 H1LD, H2LD와 H3LD를 이용하는 각각의 경우를 나누어 차선의 특정 부분이 인식되지 않아 그 값이 존재하지 않는 경우에도 나머지 두 값을 이용해 차선의 기울기를 구하는 알고리즘을 완성했다.

좌회전 직후 실제 오른쪽 차선이 카메라의 왼쪽에 잡혀 왼쪽 차선으로 인식되었다고 하더라도 그 기울기가 음수인 경우, 우회전을 실행하지 않고 그냥 직선 주행 알고리즘으로 넘어가도록 하는 내용을 작성해 이 문제를 해결할 수 있었다.

III. 결론

이 연구는 주어진 자율 주행 자동차의 주행 도로를 이용하는 경우에 발생하는 위와 같은 문제는 기울기를 이용해 해결할 수 있지만 다른 도로 유형에서는 사용할 수 없는 경우도 있을 수 있다는 단점을 갖는다. 하지만 곡선 주행 부분에서 단순히 차선의 유무를 기준으로 주행을 판단하는 것이 아니라 다른 여러 요소를 복합적으로 고려해 자동차의 주행을 판단하는 방법을 이용하였다는 점에서 의의를 갖는다.

참고문헌

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

[2] 성경복, 민경욱, 최정단 (2018). 자율 주행 자동차 기술동향 및 핵심기술. 정보와 통신 열린강좌, 35(1(별책7호)), 3-13

[3] 자율 주행차 국내외 개발 현황, KDB미래전략연구소 산업기술리서치센터, 백장균 연구위원

[4] SAE(미국 자동차 공학회), https://www.sae.org/

[5] R. Stewart, M. Andriluka, "End-to-end people detection in crowded scenes", Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2325-2333, 2016.

[6] M. Wolf, J. W. Burdick, "Artificial potential functions for highway driving with collision avoidance", Proc. Int. Conf. Robot. Autom., 3731-3736, 2008.

[7] T. Gu, J. M. Dolan, J.-W. Lee, "Human-like planning of swerve maneuvers for autonomous vehicles", Proc. IEEE Intell. Veh. Symp., 716-721, 2016.

[8] P. Raksincharoensak, T. Hasegawa, M. Nagai, "Motion planning and control of autonomous driving intelligence system based on risk potential optimization framework", Int. J. Autom. Eng., 7(1), 53-60, 2016.

포스터 논문

Journal of Youth Engineering

01 후진 주행 알고리즘 추가를 통한 자율 주행 자동차 주행 성능 개선 연구

Research on improving the driving performance of autonomous vehicles by adding a reverse algorithm

이윤수, 김선우, 정준호 (인천연송고등학교)

1. 서론

테슬라에서는 현재 미국 자동차 공학회(SAE: Society of Automotive Engineers) 자율 주행 기술 2단계 수준의 자율 주행 시스템인 오토파일럿 시스템을 자사의 차량에 옵션으로 제공하고 있다. 그리고 현대 자동차에서는 스마트 센스라는 이름으로 첨단 운전자 보조 시스템(ADAS: Advanced Driver Assistance System) 옵션을 제공하고 있다. 이처럼 자율 주행 시스템은 실생활에서 사용될 수 있는 단계에 이르렀다. 이와 같은 자율 주행 시스템의 발전에 따라 본 연구팀은 한국공학한림원으로부터 자율 주행 자동차 키트를 제공받아 모의 트랙을 완주하는 활동을 진행할 수 있었다[1]. 그러나 모의 트랙 주행 활동을 진행하던 도중 직진 주행만으로는 트랙 라인을 침범하지 않고 주행하는 것이 어려운 난주행 구간이 있다는 것을 발견하였다. 따라서 본 연구팀은 위와 같은 구간에서의 완벽한 주행을 위하여 후진 주행 알고리즘을 추가하는 연구를 진행하였다.

2. 본론

1. 후진 주행 상황과 직진 주행 전환 상황 설정

본 연구에서는 후진 주행 알고리즘을 추가하기 전 트랙 내에서 주행 중 후진 주행이 필요한 상황과 직진 주행으로 다시 전환하여 주행할 수 있는 상황을 분류하였다.

트랙 내에서의 자율 주행 자동차의 주행 상황은 제한되어 있으므로 이를 고려하여 상황을 설정하였다.

1.1 후진 주행 상황

모의 트랙 주행 활동을 통해 자율 주행 자동차는 주행 도중 외부 요인에 의해, 그리고 좌회전 혹은 우회전 상황일 때 차선을 인식하지 못하여 트랙 라인을 침범한다는 것을 발견하였다. 만약 자율 주행 자동차가 차선을 인식하지 못하는 상황이 계속된다면, 자율 주행 자동차는 오류로 인해 의도치 않은 주행을 하여 트랙을 완전히 이탈하는 문제가 발생하기도 한다는 것 또한 발견하였다. 따라서 본 연구에서는 자율 주행 자동차가 관심영역에서 차선을 Canny Edge Detector로 검출하지 못하는 상황을 후진 주행이 필요한 상황으로 판단하였다.

1.2 직진 주행 전환 상황

본 연구의 목적은 후진 주행을 통해 자율 주행 자동차가 차선을 다시 검출하게 하여 완벽한 주행을 하도록 이끌기 위함이다. 그렇기에 자율 주행 자동차가 언제 직진 주행을 재 시작해야 하는지에 대한 주행 상황 설정이 필요하다. 따라서 본 연구에서는 후진 주행을 하던 도중 관심영역에서 차선이 Canny Edge Detector로 검출되는 상황을 직진 주행으로 전환하는 상황으로 판단하였다.

2. 주행 알고리즘 구현

자율 주행 자동차가 차선을 검출하지 못하는 경우는 두 가지가 있다. 외부 요인으로 인해 관심영역에 필요 이상의 차선이 검출되는 경우, 좌회전 혹은 우회전 상황일 때 일시적으로 차선을 검출하지 못하는 경우이다. 본 연구에서는 전자의 경우 무수히 많은 외부 요인이 있을 수 있기에 후자의 경우를 다룰 것이다.

2.1 후진 주행 상황 판단 및 후진 주행 알고리즘

다음 코드와 같이 좌회전 혹은 우회전의 상황에서 Canny Edge Detector를 통해 차선이 검출되지 않을 때 'S2150E' 프로토콜을 호출하여 후진 주행을 실시한다. 좌회전 상황에서의 경우 우측과 전면의 차선이 검출되지 않는다면, 즉 right과 end 변수의 값이 모두 False라면 후진 주행을 실시한다. 우회전 상황에서의 경우 좌측과 전면의 차선이 검출되지 않는다면, 즉 left와 end 변수의 값이 모두 False라면 후진 주행을 실시한다.

```
elif status is ONLEFT:
    if right and not left and end:
        if V3D < 75:
            command = 'S1120E'
        if not right and not end: #차선이 인식되지 않는다면
            command = 'S2150E' # 후진
    elif left and right:
        if (H2LD > 60 and H2RD > 60) or (H3LD > 65 and H3RD > 65):
            command = 'S1150E'
            status = ONSTRAIGHT
    elif end:
        command = 'S1120E'

elif status is ONRIGHT:
    if not right and left and end:
        if V4D < 75:
            command = 'S1180E'
        if not left and not end:
            command = 'S2150E'
    elif left and right:
        if (H2LD > 60 and H2RD > 60) or (H2LD > 65 and H3RD > 65):
            command = 'S1150E'
            status = ONSTRAIGHT
    elif end:
        command = 'S1180E'
```

그림 1. 후진 주행 상황 판단 및 후진 주행 알고리즘

2.2 직진 주행 전환 알고리즘

후진 주행 상황 판단 및 후진 주행 알고리즘 또한 직진 주행 알고리즘과 마찬가지로 auto Drive_algorithm 함수 내에 구현하였다. 따라서 관심영역에서 차선이 Canny Edge Detector로 검출되는 상황이라면, 즉 자율 주행 자동차가 다시 차선을 검출한다면 자동으로 직진 주행을 실시한다.

3. 결론

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

02

자율 주행 자동차 주행 알고리즘의 문제점과 한계점 분석

Limit Point Analysis of Autonomous Driving Vehicle Algorithms

김민서, 김건우, 조광현, 천승원 (선덕고등학교)

1. 서론

기존의 알고리즘은 주행 상황을 직진, 곡선, S자로 구분했지만, 우리 조에서는 직진, 좌회전, 우회전으로 나누었다. 직진 코드에서 주로 차선의 중앙값을 이용했고, 좌회전과 우회전 구간에서 양 차선의 인식 여부에 따라 회전 방향을 결정했다. 이후 빈번한 주행을 통해 정확도를 향상시켰다. 결과적으로 회전 구간에서 안정적으로 회전하는 각도를 찾아내어 이용할 수 있었고, 실제 이용하게 될 주행 코스에 적응시킬 수 있었다. 그러나, 오히려 계산에서 벗어나는 급격한 회전 구간을 정확하게 통과할 수 없었고, 무수한 주행이 필요하게 되어 나중에는 큰 발전이 없었다. 이러한 문제점을 보완하기 위해 내장 코딩과 머신러닝 알고리즘의 도입을 논의했다.

2. 기존의 알고리즘

지정된 주행 트랙은 크게 직진, 곡선, S자의 총 3가지 상황으로 구분됐다. 그러나 우리 조에서는 곡선 도로 주행에서 핵심 기능은 좌회전이고, S자 도로 주행에서 핵심 기능은 우회전이라고 판단하여 직진, 좌회전, 우회전 총 3가지의 경우로 나누어 프로그래밍했다.

가장 먼저 해야 할 과제는 기본적인 직진 주행 알고리즘을 만드는 것이었다. leftLane, rightLane, endLane을 통해 각각 왼쪽 차선, 오른쪽 차선, 전방 차선 인식을 할 수 있게 프로그래밍했고, 곡선 차로 프로그래밍에서 이 세 변수들을 공유했다. leftLane과 rightLane을 이용하여 좌우 차선이 동시에 인식될 경우 직진하도록 했다. 그러나 이 알고리즘은 주행했을 때 도로의 상황과 라이다의 차선 인식 오류 때문에 차선 밖으로 벗어나는 것이 비일비재했다. 그래서 이용한 코드가 차선의 중앙값이었다. 차선의 중앙값을 centerPoint_X로 나타내고 차량의 위치와 비교하여 앞바퀴의 조향 각도를 정했다. 차량의 현재 위치와 차선의 중앙값의 차가 클수록 차량이 크게 회전하도록 설정하여 출발 시 차선의 중앙에 위치하고 있지 않을 때 빠르게 차선 중앙으로 이동하게 했다. 일반적인 직선 차로를 주행할 때에도 가운데로 주행하도록 했다. 이후 주행시키고 앞바퀴 조향 계수를 고치는 과정을 반복하여 정확도를 높였다.

자율 주행 대회 예선에 출전하기 전 마지막으로 신호등 인식 알고리즘을 추가했고, 직선·곡선·S자 주행 상황마다 다른 코드가 필요하다고 생각되어 주행상태를 status 변수에 저장하고 직진 - ONSTRAIGHT, 좌회전 - ONLEFT, 우회전 - ONRIGHT 총 3가지의 상황으로 분류했다. 전방 거리와 좌우 차선이 동시에 특정한 거리 내에 들어오면 우회전하도록 하여 기존 알고리즘에서 직진 상황과 우회전 상황을 혼동하던 오류를 해결했다.

본 프로젝트에서 이용했던 카메라 모델이 빛의 색을 인식하지 못하였기 때문에 ROI(관심 영역) 설정을 통해 초록 불/빨간 불의 위치를 지정하고, 지정된 위치의 밝기가 변화하는 것으로 신호등의 색을 추론했다. 그러나 실제 주행 상황에서는 신호등이 작동하는 알고리즘이 예상대로 작동하지 않았고, 결국 예선 때는 알고리즘에서 제거했다.

3. 기존 알고리즘의 문제점

좌회전, 우회전, 직진 세 가지 상태로 분기하는 기준을 양 차선의 중앙값으로 잡았다. 그래서 급격한 좌 우회전 구간에서는 상태를 유연하게 바꾸지 못했고, 같은 주행 상황에서도 시작 지점과 방향, 라즈베리파이의 정보 처리 시간 같은 우연한 요소들의 영향을 많이 받아서 꾸준한 주행 결과가 나오지 않았다.

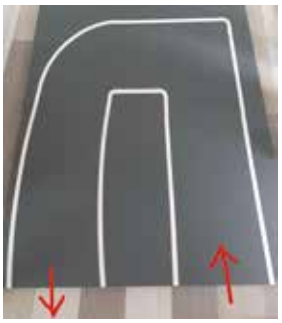


그림 1. 180도 이상의 좌회전 구간

그림 2. 90도 이상의 좌회전 구간

또한 자동차를 주행시켜 보면서 적절한 회전각 계수를 생각하고 하나하나 조정했는데, 시간이 정말 오래 걸렸고, 결과물도 만족스럽지 못했다.

4. 자주차의 한계점

라이다 작동 오류가 주행할 때 계속 발생해서 나중에는 검정색 판으로 도로 주변을 모두 둘러쌓아야 했다.

그림 3. 오류 예시 1번

그림 4. 오류 예시 2번

자주차를 운용하는 본체(라즈베리 파이)와 코드를 가지고 본체와 통신하는 노트북 사이에 통신 지연이 발생하여 자주차가 회전해야 할 타이밍을 놓치는 등의 문제가 생겼었다. 차의 앞바퀴 크기가 커서 양 방향으로 회전할 때 (특히 좌회전 - 90도 이상 코너에서) 앞바퀴가 밀리는 현상이 발생했다.

5. 한계점 극복 방향

'회전 각도 조정을 머신러닝 모델으로 해결할 수 있다.' 지금으로선 지도학습 또는 강화학습을 이용하는 것이 가장 나아 보인다. '자율 주행 자동차 내부에 코드를 저장한다.' 명령어 체계가 달라서 조금은 시간이 필요한 방안이다.

03 안전한 자율 주행을 위한 차선 인식 및 경로 추적 주행 알고리즘

Lane recognition and route tracking driving algorithms for safe autonomous driving

권혁제, 김종헌, 박건용, 윤동건 (인천하늘고등학교)

1. 서론

본 연구에서는 자율 주행 자동차에서 발생한 세 가지 문제점을 개선하는 알고리즘을 제안하는 것을 목적으로 한다.

① 좌우 모터의 차이로 발생한 직진 주행 오류
② 곡선 주행 시 차선 미인식으로 인한 오류
③ U자 곡선 주행 시 복수 차선 안식으로 인한 오류

2. 직진 주행 오류 개선

자율 주행 자동차의 왼쪽 모터와 오른쪽 모터의 힘을 정확하게 일치시키는 것은 불가능하기 때문에, 이를 보정하는 알고리즘이 없으면 왼쪽이나 오른쪽으로 치우치게 된다. 직선 주행 알고리즘에서는 왼쪽과 오른쪽 차선의 유무에 따라 직진을 하게 하였는데 이와 달리 중앙값 알고리즘에서는 중앙에서 왼쪽 차선까지의 거리가 H2LD, 오른쪽 차선까지의 거리가 H2RD이고 차가 정확히 중앙에 위치해 있다고 가정하였을 때 중앙은 160이다. 따라서 왼쪽 차선의 위치값은 160- H2LD이고 오른쪽 차선의 위치값은 160-H2RD이다. 따라서 차선의 중앙값은 (1)과 같다.

$$(160-H2LD)+\frac{(160+H2RD)}{2}=\frac{(320-H2LD+H2RD)}{2} \quad \cdots\cdots(1)$$

(1)을 이용하여 왼쪽과 오른쪽 차선이 모두 있을 때, 차선의 중앙값을 차의 중앙에 위치하도록 차선의 중앙값이 차의 중앙보다 왼쪽에 있을 때는 왼쪽 모터를 더 강하게 작동시켜 왼쪽으로 돌게 하여 차선의 중앙값과 차의 중앙을 일치시켰다. 오른쪽에 있을 때도 이와 마찬가지로 오른쪽 모터를 더 세게 작동시켜 오른쪽으로 돌게 하여 차선의 중앙값과 차의 중앙을 일치시키게 하며 결국 직진하도록 알고리즘을 구현하였다.

3. 곡선 주행 오류 개선

중앙값 알고리즘에서는 왼쪽과 오른쪽 차선이 모두 있을 경우, 양쪽 차선의 중앙값을 구하여 차의 중앙과 차선의 중앙값을 일치시키며 직진하도록 알고리즘을 제작하였다. 이 알고리즘이 직진 구간에서는 잘 작동하였지만 차선이 곡선 모양 즉 자회전이나 우회전을 해야하는 차선에서는 제대로 작동되지 않았다. 따라서 이의 이유를 찾아보니 차가 좌회전이나 우회전을 하기 직전에 차의 카메라가 이동해야 하는 방향 쪽의 차선이 카메라에 인식되지 않는 다는 것을 알 수 있었다. 따라서 이와 같은 문제점을 해결하기 위해 곡선 주행을 할 수 있는 곡선 주행 알고리즘을 제작하였다.

곡선 주행에서는 if 구문에서 상황(status) 변수를 이용하여 이전 자주차의 진행 상황이 우회전, 좌회전, 후진이었는지, 아니면 U자 곡선 주행었는지 확인한다. 자율 주행 자동차가 주행 중 상황이 바뀌는 것을 인지했다면 상황에 맞추어 현재의 상황을 바꾸어야 한다. 예를 들어 자주차가 직진 도중 좌회전이 필요한 상황이 온다면 그 부분에 'status=ONLEFT'라는 코드를 입력해주면 주행 상태가 직진에서 좌회전으로 바뀌게 된다.

직진 상황에서 한쪽 차선은 없고, 다른 차선과 앞 차선이 있는 경우를 생각해보면 좌회전 또는 우회전을 해야하는 상황이라는 것을 알 수 있다. 왼쪽 차선이 없는 경우는 좌회전을 해야하고 오른쪽 차선이 없는 경우는 우회전을 해야 한다. 우측 차선은 있으나 좌측 차선이 없고, 앞차선이 있는 경우를 case3으로 정의하고, 이를 직진하다가 좌회전을 하라는 것으로 표현한 후 반대로 우측 차선은 없으나 좌측 차선이 있고, 앞차선이 있는 경우를 case4라고 정의하고 이 경우를 직진하다가 우회전 하는 것으로 표현했다.

4. U자 곡선 주행 오류 개선

U자 곡선은 회전해야 하는 상황임에도 양쪽에 차선이 있기 때문에 직선 주행 알고리즘에 따라 직진을 하게된다. 따라서 양쪽에 차선이 있으면 무조건 직진하는 것이 아니라 회전해야 하는 상황까지 판단해야 한다. 따라서 일반적인 직진 상황과 U자 곡선을 돌 때의 상황을 비교해보니 일반적인 직진 상황에서는 그림과 같이 양쪽 차선만 인식되는 반면에 U자 곡선에서는 차선이 휘어져 있기 때문에 회전하는 쪽의 반대편 차선이 차량의 앞쪽 까지 휘어져 있는 것을 확인했다. 따라서 양쪽 차선이 모두 있을 때 한쪽 차선이 차량의 앞쪽까지 이어져 있을 경우를 U자 곡선을 돌아야 할 경우라고 지정하고, 이 상황에서 차선이 차량의 앞쪽까지 이어져 있지 않은 쪽 방향으로 크게 회전하도록 알고리즘을 구성하였다.

그림 1. U자 곡선 주행 상황

3. 결론

본 연구는 자율 주행 자동차를 제작하면서 자율 주행 자동차 차선인식 과정에서 발생한 문제점과 그를 해결하는 방안을 제시하였다. 본 연구에서 먼저 직진 상황에서 두 모터의 출력값이 전압 차이로 인해 달라져 모터만으로는 일자로 주행하기엔 어렵다는 문제점을 발견하였다. 이를 차선의 중앙값을 찾아 주행하는 중앙값 알고리즘의 고안으로 해결하였다. 하지만 차량의 중앙값 알고리즘은 곡선 부분에서 차선이 카메라 관심 영역에서 벗어나 캐니 이미지와 허프 이미지로 변환되지 않고 차가 차선의 중앙값을 못 찾는다는 문제점을 발견했다. 이를 조건문을 활용해 왼쪽, 오른쪽 차선이 모두 발견되기 전까지 회전한다는 곡선 주행 알고리즘의 고안으로 해결했다. 그러나 여러 곡선중 U자 곡선 주행 부분에서는 차선이 사라지지 않아 곡선 주행 알고리즘을 적용할 수 없었다. 그래서 두가지 상황을 비교해보니 일반적인 곡선 주행 상황과는 다르게 두 차선이 모두 있음에도 계속 회전을 하거나 직진을 해야함을 판단해야한다는 문제점이 발견되었다. 따라서 새로운 알고리즘이 필요하게 되었고 이를 판단하는 기준을 접선 벡터의 크기를 비교해 일반적인 곡선 트랙과 U자 트랙과의 차이를 인식하고 판단하는 기준을 만드는 U자 곡선 알고리즘을 고안하여 해결하였다.

04 곡선 주행 능력 개선을 위한 소프트웨어 및 하드웨어 고찰 및 개선 방안

A Study on the Software and Hardware for the Improvement of Curved Driving Capacity

김도경, 유서우, 반대준 (하나고등학교)

1. 서론

이 연구는 자율 주행 자동차가 곡선 주행을 하면서 발생하는 문제점을 소프트웨어와 하드웨어 부분으로 나누어 분석하였다.

2. 소프트웨어

자주차가 카메라로 차선을 인식하고 주행하는 과정에서의 판단은 프로그래밍을 통해 조절된다. 우리는 자주차의 차선 인식 시스템을 라인 트레이서에서 활용하는 방법을 적용하였다. 즉, 바로 가운데 선을 기준으로 자동차가 벗어난 만큼 방향을 전환하는 것이다. 직선 주행에서 H2RD와 H2LD의 차만큼 자동차의 방향을 전환하도록 하면 자동적으로 직선주행 중 자주차가 차선 가운데를 따라 주행함을 알 수 있다.

좌회전과 우회전 상황에서는 카메라가 전방에 있으므로 좌회전 도중에는 왼쪽 차선을 감지하지 못하고 우회전 도중에는 오른쪽 차선을 감지하지 못한다. 때문에 직진 상황과 같이 양쪽 차선 모두를 고려하지는 못한다. 따라서 우리는 가운데 선과 인식되는 차선 사이의 간격으로 방향 전환을 하도록 했다. 예를 들어 좌회전 상황일 경우 도중에 왼쪽차선은 사라지고 오른쪽 차선만 남는다. 때문에 H2RD나 H3RD를 이용하여 오른쪽 차선과의 간격이 짧아질 수록 더 방향을 크게 만들어야 한다. 다만 여기에서 가장 큰 문제점은 차선이 가운데 선보다 왼쪽으로 가는 경우가 생긴다. 이러할 경우 왼쪽차선으로 인식하여 오히려 차선을 벗어나는 경우가 있다. 때문에 좌회전 상황에서 H2RD가 0이 되었다면 H2RD의 기준을 V4가 아닌 V3로 두는 등 융통성 있는 변환이 필요하다. 다만 자주차 프로그램에서는 H2RD나 H3RD의 기준을 V4만으로 하기 때문에 H2RD가 0이 될 경우 H3RD를, H3RD 역시 0이 될 경우 최대한의 각도로 좌회전을 하는 방법 밖에 없었다.

이러한 문제는 S자 코너 주행에서도 마찬가지로 작용했다. S자 경로의 경우 오른쪽으로 U턴하는 구간이 존재하는데 U턴 막바지 구간에서 옆쪽 차선도 카메라에 인식이 된다. 때문에 기존에 있던 왼쪽차선이 V4를 넘어가서 오른쪽 차선으로 인식이 되고 옆쪽 차선이 왼쪽차선으로 인식이 되면 기존 왼쪽차선과 그 옆쪽 차선 사이를 경로로 인식하고 직진 상황으로 인식해 실제 경로를 이탈하게 된다. 이러한 문제는 V4만을 기준으로 하는 상황에서는 해결할 수 있는 뚜렷한 방법이 없다.

자율 주행 자동차가 원하는 대로 작동하지 않을 때 그 원인을 출력 함수를 통해 확인한다. 하지만 출력 함수는 반복되는 루프의 속력을 저하시키는 요인 중 하나이다. 자주차를 비롯한 여러 피지컬 컴퓨팅의 경우 굉장히 빠른 속도로 알고리즘을 반복하여 센서값을 얻고 알고리즘을 통해 올바른 행동을 실행하는 방법을 많이 이용하는데 출력 함수는 한 번 루프를 돌 때 시간을 많이 사용하게 한다. 이미 자주차는 필요한 연산을 다 끝내고 주행 명령을 내려야 하는데 출력 함수가 값을 출력하는 시간 때문에 루프가 속도가 늦어지면 얼마나 비효율적인 방법인가? 또 루프 속도가 늦어지면 자주차가 센서값을 읽고 행동을 출력하는 속도가 늦어지는 것이기 때문에 자주차의 질 역시 떨어지게 된다. 따라서 자주차가 어떻게 실행되고 있는지 알아보기 위해 한 두 번 사용하고 실제 주행시에는 주석처리를 하는 등 알고리즘 효율을 낮추지 말아야 한다.

3. 하드웨어

자주차의 뒷바퀴는 DC모터와 연결되어 있다. 자주차에서 큰 샤프트 기어와 DC모터가 연결되고, 작은 샤프트 기어가 끼워진 로드에 바퀴가 연결된다. 그리고 이때, 큰 샤프트 기어와, 샤프트 기어의 톱니가 서로 맞물리면서 뒷바퀴가 굴러갈 수 있게 된다. 하지만, DC모터를 작동시키는 과정에서 큰 샤프트 기어와 샤프트 기어가 제대로 맞물리지 않게 된다면, 큰 샤프트 기어만 헛돌게 되어 샤프트 기어가 연결된 바퀴를 움직일 수 없게 된다. 또는 샤프트 기어가 로드에 제대로 연결되어 있지 않을 경우 즉, 무두나사가 제대로 조여지지 않았을 경우에도 모터가 헛돌 수도 있다. 하지만, 후자는 해결하기 쉽지만, 전자는 해결하기에 까다롭다. 로드와 DC모터가 고정되어 있어서 둘 사이의 거리를 줄이기에 한계가 있어, 위치를 이동시키는 데에 한계가 있다. 이를 팀에서 생각해낸 결과, 스카치테이프를 반으로 접어 로드와의 마찰을 최소한으로 한 뒤, DC모터와 로드의 거리를 인위적으로 좁혀서 큰 샤프트 기어와 작은 샤프트 기어가 제대로 맞물릴 수 있게 하였다. 그러려면 서보모터와 앞바퀴의 연결방식에 대하여 알아보아야 한다. 일단 앞바퀴끼리 스티어링 바를 통하여 연결되어져 있고, 그리고 스티어링 바에 드라이 빙 바를 연결한 뒤 서모모터에 연결하는 방식으로 연결되어져 있다. 이제 우리는 바퀴가 어떻게 돌아가는지 상상이 가능할 것이다. 오른쪽으로 바퀴를 돌린다고 생각해본다면 서보모터가 설정한 각도까지 스티어링바를 돌릴 것이고 그에 따라 바퀴의 중심에 연결된 스티어링 바가 돌아가며 바퀴를 움직일 것이다.

자주차가 S자 곡선 주행을 할 때는 두 가지 문제점이 발생한다. 첫째 문제는 회전 각도의 한계이다. 자주차의 앞 바퀴는 비독립적이다. 따라서 조금이라도 틀어지면 바퀴가 헛돌거나 프로그램에서 설정한 각도까지 바퀴가 회전하지 못하게 된다. 둘째는 주행 중, 앞 바퀴 회전으로 인해 발생하는 마찰력이다. 마찰력을 통제할 수 없는 로 인하여 원하는 각도까지 바퀴가 돌아가지도 않고, 주행속도에까지 문제를 끼칠 수 있을 것이다. 이러한 문제점들을 해결하기 위해서는 방향 바퀴를 독립시키는 것이 좋은 해결책이다.

4. 결론 및 제언

본 연구에서는 자율 주행 교육용 모형 자동차에서 하드웨어 측면과 소프트웨어 측면으로 나눠서 어떻게 하면 곡선 주행의 차선 인식과 주행 능력을 개선 시킬지 연구하였다. 소프트웨어 측면에서는 H2RD 혹은 H2LD의 기준을 V4만이 아닌 V3나 V5등 융통성 있는 변환을 할 수 있도록 개선해야 할 것으로 판단한다.

하드웨어 측면에서는 뒷바퀴 즉, DC모터에 연결된 기어와 뒷바퀴 축과 연결된 샤프트 기어 간의 연결이 잘 되어야 DC모터로 얻은 동력이 잘 전될되고, 앞바퀴에서는 스티어링바의 마찰을 줄이면 방향 각도 변환을 더 효과적으로 될 것이라고 예상한다. 이와 같은 조건들을 고려하여 모형 자동차를 주행할 시 더욱 효과적으로 주행할 것이라고 예상된다.

05 이미지에서의 직선 기울기 변화에 따른 자율 주행 자동차의 위치 추정 및 차량의 직진 주행 알고리즘 개선

Estimating the Location of Self-driving Cars and Improving the Vehicle's Straight Driving Algorithm According to Changes in the Straight-line Slope in the Image

정영찬, 나지원, 김유석, 박찬서 (선덕고등학교)

1. 서론

기존의 자율 주행 알고리즘 연구에서는 운전자의 관점에서 촬영한 사진을 이용하여 차선을 인식하고 차량이 차로의 중앙에 위치하도록 제어하였다. 차량은 도로 중앙에 위치하나 카메라는 차량이 도로 중앙에 위치해 있지 않다고 인식하는 경우 차량이 도로 정가운데를 주행하지 않는다는 문제가 발생하기 쉽다.

본 탐구에서는 카메라를 통해서 차선을 인식하여 직진주행을 할 때 기존에 제시되었던 차선과 이미지 위에 덧붙여진 교점을 통한 직진주행 방안이 아니라 직선의 기울기를 통한 주행방법을 제안하고자 한다.

2. 기울기 인식의 원리

파이썬 개발 환경에서 그려지는 좌표계를 데카르트 좌표계로 변환하여 직선을 구한다.

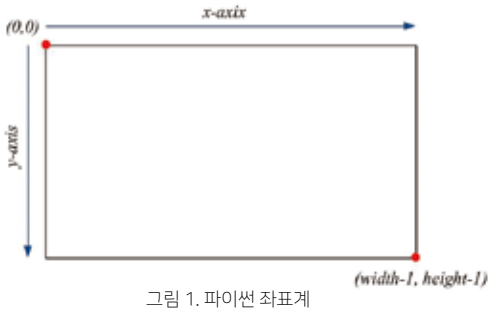


그림 1. 파이썬 좌표계

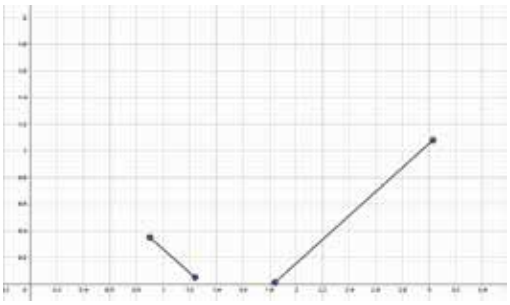


그림 2. 데카르트 좌표에 나타낸 인식된 직선

이때 인식되는 직선행렬에 대해 직선의 기울기를 구하면 오른쪽 차선은 양수의 기울기를, 왼쪽의 차선은 음수의 기울기를 갖는다. 각각의 직선은 y축 대칭된 직선이므로 만약 차량이 직선 주행 중이라면 두 직선의 기울기는 절댓값이 같아야 한다. 기울기 값에 따라 직진 상태, 오른쪽 틀어짐 상태, 왼쪽 틀어짐 상태로 나눌 수 있다. 다음은 직진일 때의 양 차선의 기울기이다.

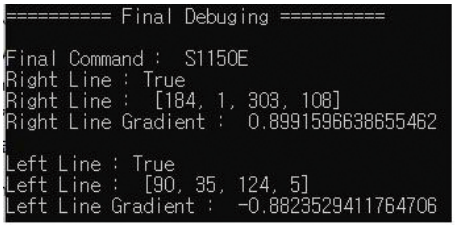


그림 3. 직진 상태의 디버깅 화면

양 차선의 기울기는 소수점 첫째자리를 기준으로 y축 대칭인 것이 확인된다. 따라서 직진일 때는 양 차선의 기울기의 절댓값이 0.8을 만족한다는 것을 알 수 있다. 다음은 차량이 우회전이 필요한 주행상황에서의 기울기이다.

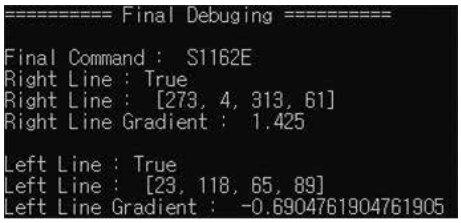


그림 3. 우회전이 필요한 상태의 디버깅 화면

오른쪽 차선의 기울기의 절댓값은 증가, 왼쪽 차선의 기울기의 절댓값은 감소하였음을 확인할 수 있다. 따라서 우회전이 필요한 경우에는 직선 주행보다 오른쪽 차선은 기울기의 절댓값이 증가, 왼쪽 차선의 기울기는 감소한 상황임을 알 수 있다. 이하 동일하게 좌회전이 필요한 상황인 경우 오른쪽 차선은 기울기의 절댓값이 감소, 왼쪽 차선은 기울기의 절댓값이 증가한 상황임을 알 수 있다.

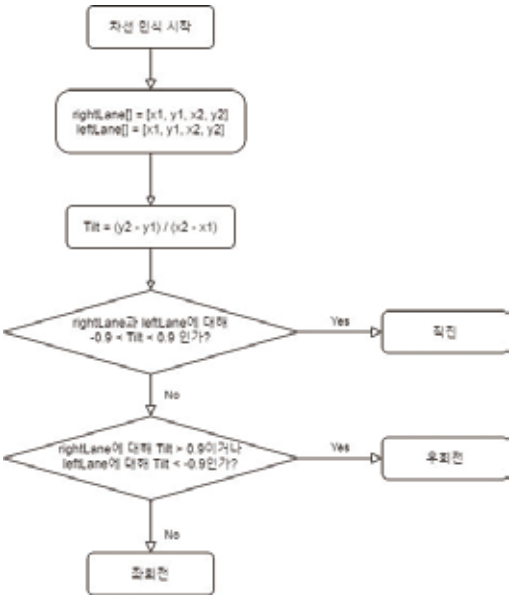


그림 2. 기울기 인식 알고리즘의 흐름도

3. 결론

차선의 기울기를 통해 차량의 직선주행을 보정할 수 있었다. 특히 카메라의 틀어짐에 따라 발생할 수 있는 직선주행 오류 상황에 더 직관적인 판단이 가능하도록 프로그래밍을 할 수 있었다.

06 자율 주행 자동차의 배향 곡선 구간의 안정적인 회전을 위한 주행 설계 제안

Proposal of Design for Stable Rotation of Reversed Curve Section for Self-Driving Car

차유경, 황지영 (미림여자고등학교)

1. 서론

실제 자율 주행 자동차는 다양한 고급 센서를 통해 주행 정보를 얻을 수 있으나, 모형 자율 주행 자동차는 주어진 제안된 센서를 최대한 활용할 수 밖에 없다. 이 연구는 주어진 센서의 한계를 인식한 후, 안정적인 배향 곡선 주행을 위한 알고리즘 설계를 제안하는 것을 목적으로 한다.

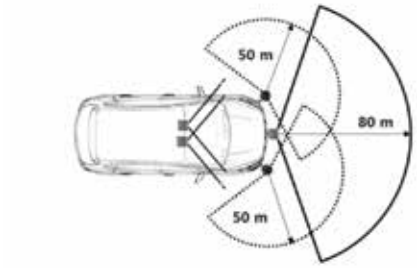


그림 1. 레이저 레이더 센서(laser radar sensors)의 스캔 범위

2. 차선 인식과 주행 알고리즘

(1) 통신 프로토콜

데이터를 통해 어떻게 주행해야 할지 판단하고 여러 프로토콜을 이용하여 라즈베리파이(Raspberry Pi) 로 명령을 보내게 한다. 제어명령 프로토콜은 그림 5와 같이 6byte로 구성된다. 그러나 주행 실험 결과, 서브 모터(조향)제어 값이 190이상 또는 110이하이면 서브모터의 정상적인 작동에 장애가 있음을 확인했다. 따라서 실제 주행 실험에서는 120 이상 185 이하의 값만 사용하였다.

시작명령 (1Byte)	이동(DC 모터) (1Byte)	조향(서보모터) (3Byte)	종료 명령 (1Byte)
S	0: 정지 1: 전진 2: 후진 3: 개발모드	110: 최대 왼쪽 150: 중립 190: 최대 오른쪽	E

그림 2. 제어 명령 프로토콜

(2) 장애물 인식

마이크로파를 이용해 전방의 물체를 감지하는 라이다 센서를 탑재한 모형을 제작해 장애물 인식에 라이다 센서를 활용한다. 라이다 값이 일정한 기준 이하라면 장애물이 근방에 있으므로 멈추는 알고리즘을 짜야한다. 그러나 일정한 값의 이하일 때 정지했을 경우 장애물이 제거된 후 차선이 제대로 인식되지 않아 오류가 나므로 바로 정지하지 않고 적절한 라이다 값이 나올 때까지 후진한다. 또한 코드 리딩 반복 주기의 길이로 인한 장애물 감지 오류의 해결을 위해 status 값에 따른 각 상황별 제어명령마다 장애물 인식 알고리즘을 삽입한다.

3. 배향 곡선 구간의 알고리즘 개선

배향 곡선 구간은 회전각이 매우 큰 구간으로, 자율 주행 자동차 모형이 어느 지점에서 이를 감지하는가에 따라 오차 범위가 몹시 크다. 따라서 배향 곡선 구간임을 감지한 지점에서 전방의 차선까지의 거리에 따라 조향(서보모터)의 제어명령 프로토콜 값을 다르게 해야 한다.

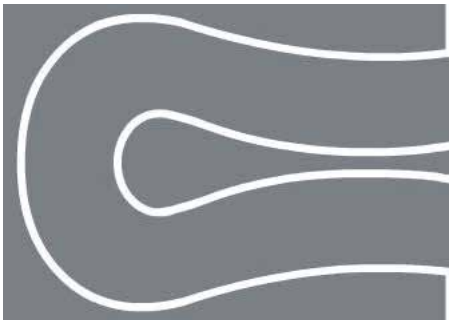


그림 3. 대표적인 배향 곡선 구간

최초 감지 지점은 V4D의 값을 기준으로 측정한다. V4D 값을 기준으로 큰 회전각을 가지고, V4D의 값이 일정한 기준보다 작을 경우 후진을 통해 안정적인 거리를 확보한다.

회전 중 차선이 카메라 밖으로 벗어나 오류가 나는 빈도가 높다. 따라서 회전 궤도의 반경을 늘려야한다. 카메라 상에 양쪽 차선이 보이는 반경을 안정적으로 유지하기 위해 후진 코드를 사용한다. 차선이 일정 기준 이상으로 가까울 때 후진하여 회전이 끝나고 양 차선이 보이도록 한다.

4. 결론 및 제언

기존 자율 주행 자동차 모형의 외골격 및 차체는 부피가 커, 중량이 크다는 한계점이 있다. 또한 단순한 직사각형의 구조로 공기 저항을 많이 받는다는 한계점도 가진다. 따라서 주행 시간의 단축 및 모터 과부하 방지를 위해 가볍고 얇은 플라스틱의 유선형 구조를 가진 새로운 차체를 제안하는 바이다.

최근 센서 데이터를 처리하기 위해 딥러닝 기법을 활발히 사용 중이다. 딥러닝이란 사람의 신경망을 모방한 인공신경망 구조를 많은 데이터를 이용하여 학습시켜서, 최적의 알고 리즘을 컴퓨터 스스로 만들 수 있도록 하는 기계학습 기법이다. 카메라는 사람의 눈과 가장 유사한 환경정보를 취득할 수 있기에 사람의 신경을 모방하여 만든 neural network를 활용하여 자율 주행 자동차의 딥러닝 기반 인지 기술을 구현하면 본 연구에 사용된 자율 주행 자동차 모형의 성능 개선에 도움이 될 것이다. 주행 중 직면하는 다양한 변수들에 대해 일일히 대응 알고리즘을 설계하는것은 비효율적이다. 따라서 주행환경의 많은 변수를 딥러닝 기술을 이용하여 효율성을 높일 수 있다.

07

자율 주행 자동차 모델 제작 및 주행 알고리즘 설계

Proposal of Design for Stable Rotation of Reversed Curve Section for Self-Driving Car

손수민, 김정환, 이융현, 남동균, 김용우, 신동혁 (인천하늘고등학교)

1. 서론

이 연구에서는 자율 주행 자동차가 회전을 할 때에 시간에 따라 조향각이 달라지는 점을 극복하기 위한 방법을 구안하여 안정적이고 유동적인 주행이 가능하도록 하는 것이다.

2. 알고리즘 설계

카메라에서 보내주는 영상 데이터를 OpenCV에서 분별하여 만들어진 차선 데이터를 좌표평면에 옮겨 상황에 따른 알맞은 조향값을 할당하였다. 시간 변화를 고려하여 차체가 커브 진입부터 조향각도가 커지다가 커브 탈출 시작 구간에서 다시 작아지는 양상으로 정교한 알고리즘을 작성하였고, 커브 진입 시 커브와 반대되는 방향으로 우선으로 방향을 틀어 커브를 돌 때 커브 진입각을 넓히었다. 'Ω' 형태의 헤어핀 도로는 구간을 지나갈 때의 차선 데이터가 커브보다 상대적으로 일정하므로, 알고리즘 설계는 더욱 수월하였지만, 헤어핀 탈출 단계에서 지속해서 차선을 밟게 되어 세세한 코딩이 요구되었다.

실제 자율 주행 자동차에 자주 쓰이는 라이다 센서를 이용해 차단기를 인식하고 정지하도록 하였다. 정면에 라이다 센서가 장착된 차체 앞에 장애물이 놓이게 되면 라이다의 거리값이 낮아지는데 임계값을 설정하고 거리값 그 값보다 작아지게 되면 자동차가 정지하게 된다. 임계값을 설정하는 방식을 차용했기 때문에, 차단기가 올라가 거리값이 높여졌을 때 자동차는 자동적으로 출발하게 된다.

신호등 인식 알고리즘은 라즈베리파이와 신호등 인식 학습 파일의 호환성 문제로 인해 구동되지 않아 본 연구팀은 장애물 인식과 차선 인식 주행 알고리즘의 정확성을 높이는 쪽으로 방향을 잡았다.

3. 문제점 개선

컴퓨터 코딩을 이용한 파이썬 기반의 알고리즘 설계로 자율 주행 자동차를 설계하였다. 라즈베리파이를 기용하였고, 일종의 운영체제 소프트웨어를 통해 알고리즘을 구동시켜 보다 정교하고 빠른 알고리즘 구동이 가능하였다. 자율 주행 자동차를 설계하면서 차선 이탈이나 라이다 센서의 계속되는 인식 불가 현상으로 인한 시행착오를 겪었지만, 지속해서 개선한 결과, 주행 부분에서는 차선을 밟지 않으면서 깔끔한 움직임을 만들어내었다.

또한, 라이다 센서를 통한 장애물 인식에서는 오류가 나지 않을 만한 최적의 제동 거리를 여러 번의 시도를 통해 찾아내었으며, 본 연구팀이 설계한 자율 주행 자동차는 신호등을 제외한 도로 내 총 6개의 커브와 1개의 헤어핀 형태의 커브 구간 및 차단기 인식 구간을 성공적으로 통과할 수 있었다.

4. 결론 및 제언

본 연구팀이 자율 주행 자동차를 설계하면서 여러 어려움을 겪음과 동시에 해결해 나갔지만 해결되지 못한 문제들이 남아있었다. 우선, 신호등 알고리즘의 호환성 충돌에 대한 대안을 찾지 못하였다. 신호등 인식 학습 파일인 "trafficlight_cascade.xml" 에서 라즈베리파이 및 자율 주행 자동차 본체 전부 인식이 되지 않아 코딩 후 알고리즘 작동이 되지 않았다. 본 연구팀은 추후 신호등 인식 학습 파일을 파이썬 파일 형태로 변환하여 호환성 문제를 해결해 나아갈 예정이다. 또한, 유동적인 도로 환경에 대처할 수 없다는 문제점이 있다. 본 연구팀의 자율 주행 자동차 개발은 총 6개의 커브로 이루어진 정해진 도로에서 개발되어, 다른 도로에서와 같이 직각이 아닌 둔각 형태의 커브 및 급격한 커브는 안정적으로 통과할 수 없었다. 개발에 바탕이 된 커브는 전부 좌회전, 헤어핀 커브만이 우회전 커브였기 때문에, 변화된 도로에 차체를 구동시킬 시 우회전 커브는 차선을 통과하여 지나가는 현상이 발생하였다. 본 연구팀은 머신 러닝을 이용한 커브 선회 학습을 통해 다양한 형태의 도로에서 더욱 안정적인 주행을 가능하게 할 예정이다.

2020년 <청년공학> 제4집 투고 안내 공고

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 교육방법론이 대두되면서 탐구활동을 통한 수준 높은 실험 및 연구보고서가 만들어지고 있으나 단위학교 발표에 그쳐 널리 알려지지 않고 사장되고 있음. 고등학생이 발표할 수 있는 전문 학술지는 전무한 상태임.

·국내외 저널에 미성년자가 논문을 게재한 경우 대입에 활용할 수 없도록 조치했기 때문에 학술 논문을 대체할 수 있는 소논문이나 발표 포스터와 같은 형태의 결과물을 만들고 낸 후, 이를 공유하고 선의의 경쟁을 할 수 있는 발표의 장이 필요함.

1. 주제

※ 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

2. 자격

- 2019년 한국공학한림원 선정 청소년 공학 리더 프로젝트 선정 학교(8개 교)
- 2019년 자율 주행 자동차 경진 대회 본선에 참여한 팀 소속 개인이나 단체(2인 이상 8인 이하)
- 청소년 공학 리더 출신 대학생 (새로운 방식의 자율 주행 자동차를 개발한 경우 게재)

3. 출판 일정

① 마감 : 2019년 5월 31일까지 제출.
② 심사 : 7월 31일까지, 심사 점수를 합산하여 최우수 논문, 우수 논문, 포스터 논문 등급으로 분류
③ 출판 : 9월 예정

4. 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, 저자소개 순으로 작성한다.

5. 그림과 표는 그림 1, 그림 2, 표 1, 표 2 등으로 표시하고 본문을 읽지 않고도 이해할 수 있도록 상세한 설명을 첨부해야 한다. 그림의 제목은 그림 밑에, 표의 제목은 표 위에 기입하며, 설명문은 한글과 영문으로 표기한다. 표, 그림캡션은 8.5point, 줄 간격은 150%, 서체는 중고딕, 왼쪽 여백은 6ch, 내어 쓰기 6ch를 한다(단, 표 1, 그림 1만 진하게 한다).

6. 인용된 참고문헌은 원고의 끝에 기재하며, 인용번호를 본문의 인용 장소에 반드시 기입하고, 인용순서대로 다음과 같이 표시한다.
   가. 단행본의 경우 : 저자명, 책명, 출판사, 인용페이지, 출판년도.
   [1] : 홍길동, 전기기기공학, 동명사, pp. 186-195, 1990.
   [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
   [1] : 홍길동, 김유신, "2상8극형 HB형 리니어 펄스모터의 자속분포와 정특성 해석", 대한전기학회논문지, 제42권, 9호, pp. 9-18, 1993. 9.
   [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.
   [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. 아래와 같은 요령으로 최종 파일을 작성하여 제출하되, 최종 편집과 교정은 한국공학한림원에서 진행한다.
   ① 원고 내 모든 항은 양쪽혼합으로 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, 논문내용부터 2단 편집형식이다. 첫 페이지의 편집과정은 다음과 같다.
   1) 초기화면에서 "모양-편집 용지"를 선택한 후 용지 종류 A4, 용지방향 보통, 여백주기에서 위쪽 19, 아래쪽 19, 왼쪽 18, 오른쪽 18, 머리말 10, 꼬리말 10, 제본은 0으로 한다.
   2) "모양-다단"을 선택한 후 단수 2단, 단 간격 8, 단구분선 투명으로 한다.

⑥ 표 박스에는 한글제목, 영문제목, 저자한문 영문이름, Abstract, Key Words 순으로 편집이 되는데 표 박스를 만드는 방법은 다음과 같다.

1. 1) "표-표 만들기"를 선택한 후 칸 수 1, 줄 수 9를 지정한 후 Enter Key를 치고 표 메뉴에서 안 여백 0, 0, 0, 0, 밖 여백 0, 0, 0, 10을, 위치는 문단, 틀 배치는 자리차지를 지정한 후 완료한다.

2) 화면에 9줄 박스가 나오면 표 안에서 F5 키를 누른 후 F3을 누르고 ↓화살표 키를 이용하여 9째 줄까지 블록을 지정한다. 키를 눌러 선 종류에서 전부와 투명을 지정하여 선 모양이 보이지 않게 한다.

3) 표 박스 첫째 줄에는 한글제목이 들어가는데 이줄 오른쪽 끝에는 논문 Volume과 Number 표시를 하기 위해 표 나누기를 한다. 길이는 가로 20mm, 세로 15mm로 만든다. 두 줄을 만들어 윗줄에는 논문 또는 속보논문 표시를 하고, 아래 줄에는 Volume과 Number, 논문게재순번을 표시한다. 10point, 서체는 중고딕으로 한다.

⑦ 한글제목은 16point, 서체는 중고딕, 줄 간격은 130%, 진하게로 한다.

⑧ 영문제목은 12point, 서체는 중고딕, 줄 간격은 130%, 진하게로 한다.

⑨ 저자명은 10point, 서체는 신명조, 줄 간격은 130%로 한다.
저자소속이 다를 경우 구분은 "*, **, ***, §, §§, §§§, #"순으로 하여 윗첨자로 표시하며 1단 왼쪽 하단의 저자 순서와 동일하게 한다.

⑩ Abstract는 8.5point, 서체는 신명조, 줄 간격은 130%로 한다.
단, Abstract 표시만 진하게로 하고 '-' 을 넣어 내용을 게재한다.

⑪ Key Words는 8.5point, 서체는 신명조, 줄 간격은 130%로 한다.
단, Key Words 표시만 진하게로 하고 ':' 을 넣어 내용을 게재한다.

⑫ 표 박스 안의 한글제목과 영문제목 사이, 영문제목과 저자이름 사이, 저자이름과 Abstract 사이, Abstract와 Key Words 사이 여백은 전부 1cm로 통일한다. 이는 본문크기 8.5point로 Enter Key를 두 번 사용하는 것과 같다.

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 제4집

인쇄일 2020년 9월 3일
발행일 2020년 9월 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