2019 창업연계공학설계입문
AD Project 보고서

[6분반 1조]

20151772(김명호) 20191681(최정훈) 20191675(조현진)
20171703(정태원) 20191687(한수연)

1. 서론

“운전자 또는 승객의 조작 없이 자동차 스스로 운행이 가능한 자동차” 이는 자율주행 자동차의 정의이다. 요즘 이 자율주행자동차에 대한 기업들의 관심이 크게 상승했는데, 그 이유는 자율주행 자동차가 제 4차 산업혁명의 핵심기술 중 하나이기 때문이다. 제 4차 산업혁명을 주도하는 선두주자가 되기 위해 이름난 자동차회사인 테슬라, 벤츠 등은 물론 구글이나 NVIDIA와 같은 IT기업들도 앞다투어 이 분야에 뛰어들고있다.

우리는 흔히 자율주행 자동차라고 하면 운전석 없이 자동차 스스로 주행하는 자동차를 떠올린다. 하지만, 이는 궁극적으로 완벽한 자율주행자동차이다. 현재의 기술로는 이 정도까지 구현이 불가하다.

미국 자동차기술 학회(SAE)의 자율주행기술 발전 6단계

자동화단계	특징	내용
사람이 주행환경을 모니터링 함		
Level 0	비자동 (No Automation)	운전자가 전적으로 모든 조작을 제어하고, 모든 동적 주행을 조장하는 단계
Level 1	운전자 지원 (Driver Assistance)	자동차가 조향 지원시스템 또는 가속/감속 지원시스템에 의해 실행되지만 사람이 자동차의 동적 주행에 대한 모든 기능을 수행하는 단계
Level 2	부분 자동화 (Partial Automation)	자동차가 조향 지원시스템 또는 가속/감속 지원시스템에 의해 실행되지만 주행환경의 모니터링은 사람이 하며 안전운전 책임도 운전자가 부담
자율주행 시스템이 주행환경을 모니터링 함		
Level 3	조건부자동화 (Conditional Automation)	시스템이 운전 조작의 모든 측면을 제어하지만, 시스템이 운전자의 개입을 요청하면 운전자가 적절하게 자동차를 제어해야하며, 그에 따른 책임도 운전자가 보유
Level 4	고도 자동화 (High Automation)	주행에 대한 핵심제어, 주행환경 모니터링 및 비상시의 대처 등을 모두 시스템이 수행하지만 시스템이 전적으로 항상 제어하는 것은 아님
Level 5	완전 자동화 (Full Automation)	모든 도로조건과 환경에서 시스템이 항상 주행 담당

※ 자료 : 자율주행기술동향-기술수준 구분, 한국교통연구원, 2016.04.

위 자료를 보면 자율주행자동차에는 그 기능에 따라서 여러가지 단계가 있다는 것을 알 수 있다. 현재는 주로 부분자동화나 조건부 자동화단계에 머물러있는 상태이다. 자율주행자동차가 주행하기 위해서는 환경인식, 위치인식, 판단, 제어 등 여러 기술들이 필요한데, 하나하나 쉽지 않기 때문에 쉬운 분야가 아니다.

이번기회를 통해서 우리도 지급받은 Xycar를 통해 세기의 관심사인 자율주행자동차의 구동원리에 대해서 알아보고, 주행방식을 직접 설계해볼 수 있었다. 이번 AD프로젝트에서 우리는 자율주행자동차라면 지녀야할 핵심기능들을 생각해보았다. 그 결과 도로 위에 있는 물체인식, 도로의 상태 그리고 정차 이 세 가지 기능들이 자율주행자동차가 지녀야 할 핵심기능이라고 생각했으며, Xycar에 탑재되어 있는 각종센서와 우리의 코딩을 통해서 이를 구현했다. 본 문헌에서는 우리가 구현한 핵심기능들에 대한 설명과 그 기능을 구현하기위한 코드들에 대한 설명을 다룰 예정이다.

2. 초음파를 활용한 안전거리를 확보하고 자동 정지시키기

-실험과제

전면에 정지하고 있는 물체를 초음파를 통해서 감지하고 이후에 정지해서 보행자의 안전을 확보시키자

-실험설계

목적 : 초음파 탐지를 활용하여 차를 정차시키기.

목표 : 과제1의 장애물 탐지 전후진을 활용하여 목적 수행하기

-요구사항 분석

초음파의 값이 변동이 심할 수 있으므로 필터 값을 적용(과제1에도 적용했던 방법)

속도의 값에 따라 초음파를 인지하는데 걸리는 시간의 폭을 자동차의 속도변화(차를 정지)와 어떻게 연관지어 설정을 할 것인지 실험 설계 반복하기.ex) 전면 중간 초음파가 인지 한거리 < 50 => 속도를 0으로

← **실제로 이동중에 차가 안전하게 정지한 모습**

self.obstacle_detector = ObstacleDetector('/ultrasonic') 퍼블리셔가 발행하는 초음파 거리의 값들(list)을 객체에 담고

obs_l, obs_m, obs_r = self.obstacle_detector.get_distance()

전면의 초음파 발생기(HC-SR04) 의 값 중 전면 초음파 3개(좌, 중, 우)만 사용하기 => 목표내에서 자동차가 후진할 일이 없기 때문에 필요한 자원만 사용 초음파의 값들은 튀는 경우가 많아서 필터가 꼭 필요함

L = [67 ,65 , 64]

에서 평균의 값(가중치 이동평균)을 또 마지막 값과 4번째의 값이 큰 차이를 가져올 때는 마지막 값을 잘못된 값으로써 인지하고 무시하는 알고리즘을 적용함

- **실험 변수**

정해진 방향으로만 초음파가 전달되는 것이 아니고

다른 센서에서 발사한 초음파가 물체에 반사되어 수신될 수도 있음

3. 안정적인 경사면 주행 기능

- **개요**

세번째로 우리조가 만들 기능은 경사면 주행 기능이었다. 만약 양쪽 측면에 경사를 주행 할 때, 속도를 줄이지 않고 빠른 속도로 주행하면 차가 뒤집어 질 수 있다. 따라서 자율주행 자동차에서는 운전자의 안전을 위해선 경사면 주행시 감속하는 기능이 필요하다고 생각하여 이 기능을 구현하게 되었다.

- **코드의 구현**

이 기능을 구현하기 위해, Xycar의 IMU센서를 이용하기로 하였다. 경사면을 타게 되면 옆으로 기울여지기 때문에 만약 차의 IMU센서가 옆으로 일정 각도 이상 기울어지게 된다면 속도를 줄이는 방식을 사용하기로 하였다.

```
def trace(self):
    obs_l, obs_m, obs_r = self.obstacle_detector.get_distance()
    line_l, line_r = self.line_detector.detect_lines()
    self.line_detector.show_images(line_l, line_r)
    r, p = self.imu.get_data()
    angle = self.steer(line_l, line_r)
    speed = self.accelerate(angle, obs_l, obs_m, obs_r, r, p)
    if self.origin and p > -6.5:
        speed = -7
        angle = 0
    elif 4<abs(r)<180 and self.origin == False:
        speed = 28  # 30
```

우선, IMU_Subscriber를 활용하여 Roll값이 기울어진 방향과 정도에 따라 얼마나 변하는지 확인하였다. 확인 결과, 왼쪽으로 기울어졌을 때는 Roll값이 1~180의 양수값, 오른쪽으로 기울어졌을 때는 -1~-180의 음수값을 나타냈다. 그래서 맨 처음에는 단지 r값이 변하기만 하면 속도를 변화시키면 될것이라고 생각했다. 하지만, 차를 돌려보니 IMU센서가 불안정해서 직진할때도 Roll값이 튀는 경우가 발생하였다. 따라서, 약간의 오차를 감안하여 r의 절댓값이 5~180 사이일 때, 속도를 줄이도록 코드를 작성하였다.

- **장애물 제작**

마지막으로, 차가 올라갈 수 있는 경사면을 만들어주었다. 재료는 우드락을 사용해서 만들었으며, 생각보다 차가 무겁기 때문에 좀 두꺼운 우드락을 사용하고 여러개의 받침대를 만들었다. 또한, 차가 좀더 쉽게 경사면 위로 올라갈 수 있도록 양쪽에 경사면을 설치하고 우드락 아랫부분을 깎아서 제작하였다.

4. 오르막주행과 정차기능

- 개요

마지막으로 우리조가 구현하여야 할 기능은 pitch 값을 이용한 오르막길 주행과 주행후 정차 기능이다. xycar가 오르막길을 주행하려면 평지에서와 같은 속도로 주행을 하더라도 차 자체의 무게와 경사면의 각도 때문에 상대적으로 느리게 느껴진다. 하지만 그렇다고 경사면에서 더 빠른 속도로 주행을 하게 된다면 오르막길 후 정차 하여야 하는 평지 구간에서 더 빠른 속도로 주행이 되기때문에 정차하기가 어려워진다. 이를 방지하기 위하여 우리조에서는 경사면에서는 평지와 비슷한 속도를 유지하되, 경사면의 각도가 완만해질수록 속도를 줄이는 것으로 정차를 쉽게 하기로 정하였다.

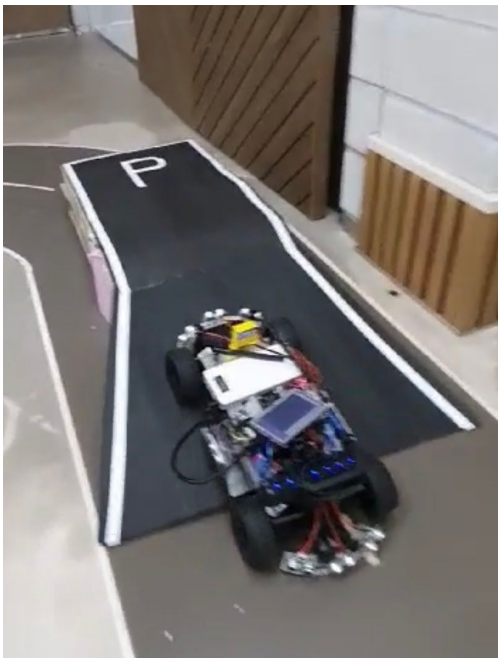
- 구현

```
def trace(self):
    obs_l, obs_m, obs_r = self.obstacle_detector.get_distance()
    line_l, line_r = self.line_detector.detect_lines()
    self.line_detector.show_images(line_l, line_r)
    r, p = self.imu.get_data()
    angle = self.steer(line_l, line_r)
    speed = self.accelerate(angle, obs_l, obs_m, obs_r, r, p)
```

①먼저 speed를 주는 함수에서 IMU센서로부터 pitch값을 받아와서 p변수에 저장한다.

```
    if self.origin and p > -6.5:
        speed = -7
        angle = 0
    elif ((r > 4 and r < 175.0) or (r < -4 and r > -175.0)) and self.origin == False:
        speed = 28  # 30
        angle = -5
    elif p <= -6.5 and p > -7.5:
        speed = -1
        angle = 0
    elif p <= -7.5 and p > -8.5:
        speed = 24
        angle = 0
    elif p <= -8.5 and p >= -15:
        speed = 26  # 28
        angle = 0
    elif p < -15:
        speed = 34.5
        angle = 0
        self.origin = True
```

② 그후 주행을 하며 pitch 값이 변화함에 따라 speed를 줄이도록 설정한다. 이때 속도를 같은 값으로 줄여버리면 오르막 주행후 정차가 제대로 되기 힘들기때문에 천천히 줄이도록 정한다. 정차한 후에는 IMU센서의 오차로 인해 다시 직진할수 있기때문에 self.origin값에 의해 다시 직진하기 않게 정하여 준다.



< 오르막길 후 정차를 하는 모습

5. 주차의 다른 방법 시도(OpenCV)

1) 다른 방법이 필요한 이유

- 우리가 주차에 사용한 방법은 IMU센서를 활용한 방법이다. 차가 언덕에 올라갈 때, pitch값이 일정량 이상 증가하면 self.origin(언덕을 올라갔다는 것을 체크)값이 True가 된다. 언덕에 다 올라간 후, pitch값이 다시 0에 가까워졌을 때 self.origin=True를 통해 차가 멈추게 된다.

- 본 AD 프로젝트는 언덕에 올라가 주차를 하고 끝나므로 이 알고리즘에 별 문제가 없다. 하지만 한 번 주차를 하면 아예 멈춰버리기 때문에 시범주행을 할 때마다 프로그램을 껐다 켜야 하는 등, 불편한 부분이 있었다.

- 또한, 원래 pitch값으로 주차를 구현하는 경우는 극히 드물다. 따라서, 좀 더 이상적인 방법으로 OpenCV를 사용하는 방법을 생각해 보았다.

2) OpenCV를 활용한 방법 절차

① 카메라를 통해 영상 정보를 받는다.

② cam_img -> hsv -> mask 과정.

③ mask에서 주차라인이 보이는 부분에서 흰 점이 일정 퍼센트 이상일 때, 주차라인으로 인식한다.

④ 정지(주차).

*세로에서 가로로 바뀌었을 뿐, 주행 시 차선을 인식하는 방법과 거의 같다.

3) 관련 코드 및 설명

```
#1)
self.cam_img = self.bridge.imgmsg_to_cv2(data,'bgr8')
```

① 카메라를 통해 영상을 받는다.

```
#2)
hsv = cv2.cvtColor(self.cam_img, cv2.COLOR_BRG2HSV)

avg_value = np.average(hsv[:,:,2])
value_threshold = avg_value * 0.6
lbound = np.array([0,0,value_threshold+50], dtype=np.uint8)
ubound = np.array([131,255,255], dtype=np.uint8)
self.mask = cv2.inRange(hsv, lbound, ubound)
```

② cam_img -> hsv -> mask 과정.

```
#3)
stop_value = False
stop_uline = 400
stop_dline = 450
stop_lline = 100
stop_rline = 540
stop_cnt = (stop_rline - stop_lline) * (stop_dline - stop_uline) * 0.4

area = self.mask[stop_uline:stop_dline, stop_lline:stop_rline]
if cv2.countNonZero(area) > stop_cnt:
    stop_value = True
    break

return left, right, stop_value
```

③ mask에서 주차라인이 보이는 부분에서 흰 점이 일정 퍼센트 이상일 때, 주차라인으로 인식한다.

- stop_value는 정지할지 체크하는 부분이다. stop_uline과 stop_dline은 영상에서 위아래 좌표이고, stop_lline, stop_rline은 왼쪽, 오른쪽 좌표이다. 다만, 이 값은 임의로 설정한 것으로, 실제 주행을 통해 적절한 값을 찾는 과정이 필요하다.

- stop_cnt는 해당 영역 안의 흰색 픽셀 수를 어느정도로 할지 정하는 값이다. 이 값 또한 임의의 값이다.

- 기존의 roi 영역 안에서 for문을 통해 작은 영역을 확인하는 방법이 아닌, 해당 영역을 countNonZero 함수로 한 번만 확인는 방법이다. 주차의 경우 빨리 멈추는게 중요한데, for문을 사용하지 않는 것이 더 효율적이지 않을까 싶어 조금 다르게 짜 보았다.

- 주차선을 발견하고 멈출 때까지 어떻게 주차선을 계속 인식할 수 있을지, 각각 값들의 조정이 중요하다.

```
#4)
line_l, line_r, line_s = self.line_detector.detect_lines()

if line_s = True:
    angle = 0
    speed = 0
```

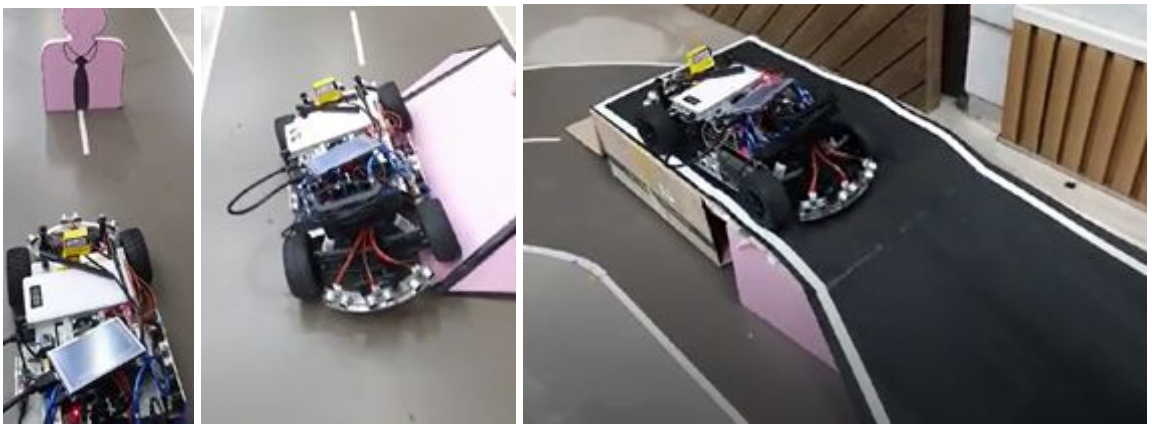
④ 정지(주차).

- autodrive.py 부분이다. 차선 왼쪽, 오른쪽을 받는 부분에서 stop_value를 line_s로 받는다. line_s가 True라면 정지하도록 계속 speed = 0을 주게 된다.

6.결과

- 사진

1)차량 보고 정지, 2)경사면에서 속도 감량, 3)경사면에서 속도 감량 +주차



- 실험결과

대부분의 코드는 원활히 작동했으나, 직접 제작한 경사면(2번째 사진)이 opencv의 선을 잡는 알고리즘에 변수로 작용(선을 가리게 됨), 또 roll 값을 인지하는 imu 센서를 활용한 알고리즘 또한 원활히 작동했으나, 모터의 배터리와 박스로 만든 주차장의 경도가 매우 낮아 차가 정지함에도 불구하고 박스가 눌리면서 시연과정에서 문제점으로 발견.

- 해결과제

실험의 환경변수의 영향을 많이 받음 ex) 배터리, 경사면이 본래의 흰 선을 가리기 때문에 발생하는 opencv 알고리즘 오류(opencv는 roi 영역내에 좌우측의 선을 확인하고 차의 속도 조향각도를 조정). 이부분은 본래 opencv만에 의지하여 가는 xycar차량에게 있어서 큰문제로

발견. 이를 해결하기 위해서 흰 선을 경사면에 부착하거나 아예 바닥의 선을 인식하지 못하도록 깔아 뒀었음.

관련 항목	실험성공여부	과제	시연성공여부	변수
UltraSonic	O	일정거리 인지후 정지	O	초음파
OpenCV	O	Roi영역 내에서 선을 발견 후 차량의 모터 값을 조정	X	구조물이 흰선을 가림
pitch	O	경사면을 다 올라갔을 때 차가 pitch값이 0임을 인지하고 정지	△	구조물의 경도 (박스)