

창업연계공학설계입문

AD 프로젝트 수행 보고서

1분반 5조

〈주제〉

자율주행 자동차 Xycar의 다양한 교통환경적 미션 수행하기

〈진행 기간〉

2019.09.02 ~ 2019.12.18

〈 참여자 명단 〉

	학과	학번	이름
조장	소프트웨어학부	20191564	김신건
조원	소프트웨어학부	20191545	강경수
조원	소프트웨어학부	20191549	고현성
조원	소프트웨어학부	20191550	곽다윗
조원	소프트웨어학부	20191565	김예리

CONTENTS

1. 서론

- 1.1. 프로젝트의 배경
- 1.2. 프로젝트의 목적

2. 본론

2.1. 차선 추종 자율 주행

- 2.1.1. 요구사항
- 2.1.2. 상세설계
- 2.1.3. 코드의 구현

2.2. 차량 차단기

- 2.2.1. 요구사항
- 2.2.2. 상세설계
- 2.2.3. 코드의 구현

2.3. 신호등 판별

- 2.3.1. 요구사항
- 2.3.2. 상세설계
- 2.3.3. 코드의 구현

2.4. 자율 주행 버스

- 2.4.1. 요구사항
- 2.4.2. 상세설계
- 2.4.3. 코드의 구현

3. 결론

3.1. 프로젝트 결과

- 3.1.1. 김신건
- 3.1.2. 강경수
- 3.1.3. 고현성
- 3.1.4. 곽다윗
- 3.1.5. 김예리

1. 서론

1.1. 프로젝트의 배경

2019년 9월 2일부터 2019년 12월 19일까지, 창업연계공학설계입문 및 유레카 프로젝트의 조원들과 자율 주행 구동체에 대한 경험적/지식적 배경의 견문을 넓히고자, '다양한 교통환경적 미션 수행'을 주제로 AD 프로젝트를 진행하였다. 프로젝트의 주제인 차선 추종 자율 주행, 차량 차단기, 신호등 판별, 자율 주행 버스를 택하게 된 동기는 각각 다음과 같다.

1) 차선 추종 자율 주행

- 기존 창업연계공학설계입문 교과목에서 학습하였던 차선을 인식하여 주행하는 방식에서 자율주행 스튜디오의 환경적 문제점을 고려하여, 근본적인 단점을 없애고자 AD 프로젝트의 연구 과제에 포함시켰다.

2) 차량 차단기

- 장애물의 여부를 단순히 초음파 센서로 판단하기에는, 초음파 센서의 정확도가 떨어지기에 많은 어려움이 있었다. 따라서, 장애물의 여부를 초음파 센서가 아닌, 카메라로 인식하여 장애물을 인식하고 모터를 제어하자는 의견이 나오게 되었고, 교통 장애물의 대표적인 예시로, 차량 차단기가 떠올랐기에 차량 차단기를 연구 과제에 넣게 되었다.

3) 신호등 판별

- 실제 교통 상황에서 가장 흔히 볼 수 있으면서, 차선과 함께 가장 중요한 교통 신호 체계인 신호등을 연구 과제에 넣는 것은 당연한 일이었다. 또한, 신호등의 교통 신호인 적색/황색/녹색 원을 인식하는 과정에서 OpenCV를 이용한 도형 및 색상 판별에 대한 알고리즘에 대해 연구할 수 있는 기회로 여겨졌기 때문이다.

4) 자율 주행 버스

- 자율 주행에 대해 조사하던 중, 우리가 흔히 접할 수 있는 교통 수단 중 자율 주행 알고리즘이 탑재된 구동체 발전 순이, 비행기 > 선박 > 자동차&버스라는 것을 알게 되었고, 이 때문에 가장 발전이 느렸던 자율 주행 버스의 기본적인 기능을 구현하고자, 버스 알고리즘을 AD 연구 과제로 삼았다. 자율 주행 버스는, 2016년도 메르세데스 벤츠 기업의 '퓨처 버스'라는 이름의 대형 버스의 주행 시험을 성공적으로 마쳤으며, 2019년 우리나라의 경우, 완전 자율 주행 버스 '제로 셔틀'이 판교 지역에서 시범 운영 중에 있다.

1.2. 프로젝트의 목적

국민대학교 창업연계공학설계입문 교과목을 통해 아래 3가지의 목표를 달성하기 위해 프로젝트를 진행하게 되었다.

첫째, 현실 제약이 따른 상황에서서의 시스템 설계를 하고 설계 -> 구현 -> 시험 -> 설계 수정의 과정을 통해 목표를 달성하면서 공학 설계에 대해 이해한다.

둘째, 임베디드 시스템 프로그래밍의 기초인 센서와 액츄에이터에 대해 이해하고 자율주행 알고리즘을 공부하면서 자율 주행 모형차 실습을 진행한다.

셋째, 자율주행 자동차가 다양한 교통환경적 요인에 대한 현실적 제약 조건을 고려하는 알고리즘을 설계 및 구현한다.

2. 본론

2.1. 차선 추종 자율 주행

2.1.1. 요구사항

Xycar는 자율주행 스튜디오의 트랙 중 하얀색 외곽선을 차선의 각도로 인식하고, 이에 따라 차량의 주행 방향을 조정하여, 차선 이탈 및 충돌 없이 자율주행을 해야 한다.

창업연계공학설계입문에서 학습한 자율주행 스튜디오의 환경적 문제점을 개선하고자, 요구사항은 다음과 같다.

- 1) 트랙의 폭이 넓기에 발생하는 문제인, 좌/우회전 중 차선을 하나 잃게 되더라도 올바르게 주행한다.
- 2) 방해물로 여겨졌던 트랙의 정중앙 하얀색 선을 차선 추종 주행 알고리즘에서 주행 방향을 알려주는 역할로 개선한다.

2.1.2. 상세설계

클래스	메서드	input	return	기능
LineDetector	conv_image	이미지 메시지	-	이미지 데이터를 OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환
	detect_line	-	외곽선의 각도 최빈값	허프 변환을 거쳐 외곽선 각도의 최빈값을 반환
AutoDrive	trace	-	-	조향각과 속력을 이용해 Xycar의 모터를 조작
	steer	외곽선의 각도 최빈값	조향각	외곽선 각도의 최빈값을 기준으로 조향각 반환
	accelerate	-	속력	지정한 속력을 반환

2.1.3. 코드의 구현

참고자료: linedetector.py & autodrive.py

USB 카메라를 이용한 비전 처리는 LineDetector 클래스의 구현을 통해 이루어졌다. 성공적인 구현을 위해 주어진 여러 문제를 풀어나가야 했다. 본격적으로 비전 처리를 위한 과정 중 첫번째로 Bird's-eye view로의 변환을 구현하였다. 차량에 부착된 USB 카메라의 시선이 보는 차선은 원근이 적용되어 있기에 실제 차선과 비교해 심하게 왜곡되어 있고 이는 직관적인 값 검출에 어려움을 준다. 그러므로, 마치 위에서 보는 것처럼 변환을 하는 과정이 필요하다. 이 과정에는 source, destination 정점 배열을 이용해 source의 정점들을 destination으로 왜곡시키는 것이 포함된다.

```
self.before = np.array([[0, 315], [639, 305], [170, 260], [460, 250]], dtype='float32')
self.after = np.array([[0, 100], [100, 100], [0, -100], [100, -100]], dtype='float32')
<Bird's-eye view로의 변환을 위한 source, destination 정점 배열>
```

이를 달성하기 위해, cv2 모듈의 getPerspectiveTransform 함수로 source에서 destination으로 변환하는 행렬을 계산한 뒤 warpPerspective 함수를 통해 2차원인 픽셀의 위치가 변환된 3차원 상의 평면에 투영되어 실제 이미지의 왜곡 과정을 진행한다.

```
tdSize = (100, 210)
m = cv2.getPerspectiveTransform(self.before, self.after)
topdown = cv2.warpPerspective(frame, m, tdSize)
...
edges = cv2.warpPerspective(edges, m, tdSize)
<실제 Bird's-eye view로의 변환>
```

두번째로, 허프 변환을 통한 직선 검출을 위해 이미지 전처리 과정을 거쳤다. 카메라를 통해 받아온 이미지는 각종 노이즈가 있을 뿐만 아니라 허프 변환 알고리즘에 적합한 형태가 아니기 때문이다. 허프 변환에서 직선을 검출하는 원리는 극좌표계에서의 직선 표현에 사용되는 θ , ρ 를 좌표평면에 도입한 파라미터 공간을 discrete하게 만들어 두고 이미지의 각 픽셀에 대해 그 픽셀과의 교점이 있는 모든 직선의 파라미터를 구하여 파라미터 공간에서의 각 점들의 값을 늘려주고, 임계치를 두어 그 이상인 점들을 존재하는 직선으로 고려하는 것이다. 그렇기에, 우선 bgr 값을 갖는 색상을 가진 3채널 이미지를 cv2.cvtColor 함수를 통해 흑백의 1채널 그레이스케일 이미지로 변환하였다. 파라미터 공간에서의 한 점은 스칼라 값을 갖기에 bgr 형태의 3차원 벡터 값은 값 누적 결정에 모호하게 작용하기 때문이다. 그 후, 존재할 수 있는 노이즈를 opencv에서 구현되어 있는 가우시안 필터인 cv2.GaussianBlur 함수를 통해 제거하였다. 마지막으로, 정보의 강건성과 최적화를 위해 정보 최소화 과정을 거쳤다. 일반 이미지는 모든 픽셀이 밀집된 형태이므로 각각 픽셀이 값을 갖기에 외곽선만 표현된 이미지에 비해 연산 대상이 많아 성능에 부정적인 영향을 미칠 뿐만 아니라 직선은 외곽선의 포함될 가능성이 높기 때문에 외곽선만을 이용하는 것이 전체 픽셀을 이용하는 것보다 노이즈가 적어 더욱 강건한 결과를 가져오기 때문이다. 외곽선 검출을 위해 cv2.Canny 함수를 이용하여 그레이스케일 이미지에서 각 픽셀마다 이웃 픽셀과의 밝기 차이를 기준으로 이진화하였다.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (3, 3), 0)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
<올바른 형태로의 변환 및 외곽선 검출을 통한 정보 희소화>
```

마지막으로, 실제로 허프 변환을 거쳐 직선들의 파라미터를 얻어냈다. OpenCV에서 제공하는 cv2.HoughLines 함수를 사용하면 입력된 이미지와 임계치 등의 인자에 따라 조건에 맞는 직선들을 반환한다. 각각의 직선마다 고유의 theta, radius를 갖기 때문에 가장 빈도가 높은 theta 값을 차선의 것이라고 간주하고 처리하였다. 또, theta가 y축을 기준으로 0 ~ 2PI 값을 갖기에 좀 더 직관적인 -PI/2 ~ PI/2 형태로 변환하였다.

```
lines = cv2.HoughLines(edges, 1, np.pi / 180, 80)

...
index = cnts.index(max(cnts))
if cnts[index] == 0:
    break
theta = vals[index] / cnts[index]
if theta > math.pi / 2:
    theta -= math.pi
thetas.append(theta)
<허프변환 및 형태 변환, 최빈값 선택 과정>
```

주행 알고리즘이 완성되고 나서는 여러 문제가 발견되었고 그 문제를 해결하기 위해 다양한 시도를 해보았다. 그 중, 인자를 수정하여 가장 효과적이고 큰 변화를 준 것이 두 가지가 있다. 첫번째는, 시야를 좁히는 것이었다. 초기 코드에서는 시야가 과하게 먼 나머지 너무 빨리 조향을 하여 트랙을 벗어나는 경우가 빈번히 있었다. 이 문제는 Bird's-eye view 변환에 사용되는 source, destination 정점 배열의 수정을 통해 시야를 눈에 띄게 줄여 해결하였다. 두번째 문제는 가우시안 블러의 필터 크기가 너무 컸다는 것이었다. 노이즈를 제거하기 위해 삽입한 코드였는데, 너무 강도가 심했는지 차선과 바닥의 경계가 모호해져, Canny 외곽선 검출에서 선으로 인식되지 않았기 때문에 이 또한 차선을 이탈하는 문제를 빚어냈다. 이 문제도 간단하게 가우시안 블러 함수의 인자에서 필터의 크기를 7x7에서 3x3으로 낮추어 해결하였다.

프로젝트 진행 중, 모터의 조향과 가속은 AutoDrive 클래스에서 LineDetector가 인식한 차선 데이터로부터 이루어졌으며 여러 과정을 통해 그 데이터를 가공하여 모터 노드에 메시지를 발행하여 달성하였다.

가장 처음으로 시도한 방식은 얻은 차선의 theta 값에 따라 일정한 비율로 조향, 감속을 하는 것이었다. 조향을 담당하는 메서드인 steer와 가속을 담당하는 메서드인 accelerate에서 각각 theta에 대한 비율인 지역변수 K를 선언하고 값을 임의로 넣어 구성하였다. 간단한 방식이었기에 코드 작성 후 바로 시험주행을 할 수 있었고, 결과는 예상보다 뛰어났다. 간헐적으로 트랙을 벗어나거나 차선을 밟는 일이 있었지만 간단한 수정으로 개선할 수 있어 보였다.

그 후에는 theta의 값을 단계 별로 나눠 K 값을 정하는 방식으로 바꾸었다. theta의 절댓값이 작을 때는 너무 조금 조향하고 값이 너무 클 때는 과하게 조향하였기 때문이다. 그런 문제를 최소화하고자 theta가 작을 때는 K를 크게 설정하고 값이 클 때는 K를 작게 설정하는 식으로

수정하였다. 또, 가속의 정도에 따른 안정성은 θ 의 값과 큰 관계가 없다는 것을 관찰하고 항상 같은 속도로 주행하도록 하였다.

그렇게 수정한 뒤에는 좀 더 안정적으로 주행하게 되었지만 트랙의 중심으로 주행하지 않고 한 방향을 쏠려 주행하는 문제가 남아있었기에 여러 해결방안을 생각해보았고 시험하였다. 첫번째로 시도한 방법은 LineDetector에서 왼쪽, 오른쪽 차선을 각각 구해 두 차선의 중앙에서 주행하게 조향하는 것이었다. 이 방법은 기존의 θ 를 이용한 조향과 잘 맞지 않았고 각 방식이 서로 반대의 방향으로 조향하게 명령하여 값이 상쇄되는 문제가 있었다. 이 문제를 해결하기 위해 직진 상황에서만 중앙으로 주행하게 바꿔보았으나 잘 되지 않았다. 결국 이 방법을 사용하지 않기로 결정하였다. 대신, 직진 상태에서 그렇듯 매우 작은 θ 값이 입력되는 경우에는 K 의 값을 매우 크게 주어 쏠림을 보정하고, 기본적으로 조향 각도와 θ 값 자체에 오프셋을 주어 해결하였다.

이렇게 차선 추종 자율 주행이 완성되었다.

2.2. 차량 차단기

2.1.1. 요구사항

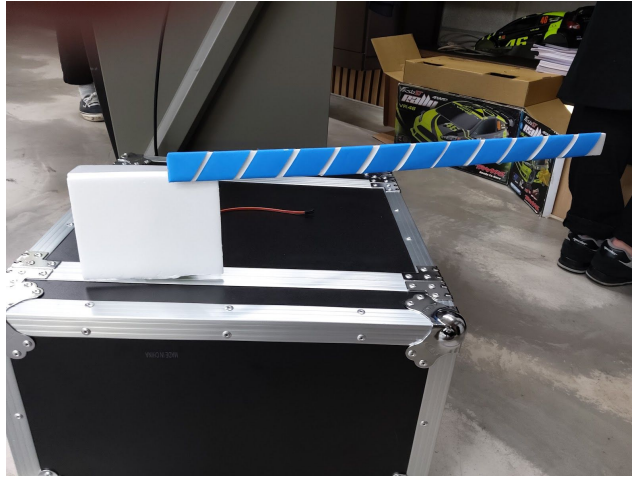
Xycar는 차량 차단기의 기울기에 따라, 모터를 제어해야 한다. 기울기에 따른 주행 시 고려 사항은 다음과 같다.

- 1) Xycar의 카메라에 파란색 픽셀이 많이 들어있다면, Xycar는 이를 차단기가 내려간 상태로 인식하고 정차한다.
- 2) Xycar의 카메라에 파란색 픽셀이 상당량&일부 들어있다면, Xycar는 이를 차단기의 기울기가 변화하고 있는 상태로 인식한다.
- 3) Xycar의 카메라에 파란색 픽셀이 아예 없거나, 소량만 존재할 경우, Xycar는 이를 차단기가 없거나, 차단기가 올라간 상태로 인식하고 정상 주행한다.

2.1.2. 상세설계

클래스	메서드	input	return	기능
VehicleBreaker Detector	conv_image	이미지 메세지	-	이미지 데이터를 OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환
	detect	-	차단기 발견 여부, 파란색 픽셀의 개수	lower_blue와 upper_blue를 기준으로 이진화 작업 후, 파란색 픽셀 개수 측정 및 차단기 발견 여부 판단
Autodrive	trace	-	-	차단기 발견 여부에 따른 주행 판단

서보 모터를 연결한 차량 차단기 (라즈베리 파이) ServoMotor.py



<서보모터를 연결한 차량 차단기(라즈베리 파이)>

2.1.3. 코드의 구현

참고자료: VehicleBreakerDetector.py & autodriven.py

먼저, USB 카메라로부터 이미지를 수신받기 위해 자이트론 측에서 구현한 노드가 발행하는 메시지 토픽인 '/usb_cam/image_raw' 을 conv_image 메서드를 콜백 함수로 하여 구독하였다.

```
self.bridge = CvBridge()
rospy.Subscriber(topic, Image, self.conv_image)
<CvBridge 객체의 생성과 '/usb_cam/image_raw' 토픽의 구독>
```

그리고 메시지가 발행 되면 호출되는 conv_image 메서드에서 이미지 정보를 OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환하기 위해 cv_bridge 모듈의 CvBridge 클래스의 객체를 생성하여 imgmsg_to_cv2 메서드를 이용해 rgb8 형식의 바이너리 데이터를 bgr8 형식의 numpy.ndarray로 변환하고 멤버변수인 cam_img에 저장하였다.

```
def conv_image(self, data):
    self.cam_img = self.bridge.imgmsg_to_cv2(data, 'bgr8')
    <conv_image 메서드의 구현>
```

그리고 본격적으로 차량 차단기를 검출하기 위해 detect 메서드에서 차단기 발견 여부 변수인 ret에 초기값으로 False를 넣고, OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환된 이미지 정보가 담긴 cam_img를 파란색을 검출하기 위해 HSV 변환 작업을 거쳤다. 이후,

lower_blue와 upper_blue로 Xycar가 인식할 '파란색'의 기준이 되는 색상/채도/명도 범위를 설정한 후, 이진화 작업을 거쳐, 파란색인 것과 파란색이 아닌 것으로 구분하였다. 다음으로, 이진화된 이미지를 cv2 모듈의 countNonZero 메서드를 이용하여, 파란색에 속하는 픽셀의 수를 구하여 변수 cnt에 저장하였다. 마지막으로 cnt의 값이 13000보다 많다면, 즉, 파란색 픽셀의 수가 13000개보다 많다면, 이를 차단기가 감지된 상황이라 판단하고 ret에 True를 저장하였다. 이후, ret과 count를 반환한다.

```
lower_blue = (90, 130, 130)
upper_blue = (130, 255, 255)

img_mask = cv2.inRange(img_hsv, lower_blue, upper_blue)
cnt = cv2.countNonZero(img_mask)
<파란색 범위 설정 및 이진화 작업 & 픽셀 수 카운팅>
```

2.3. 신호등 판별

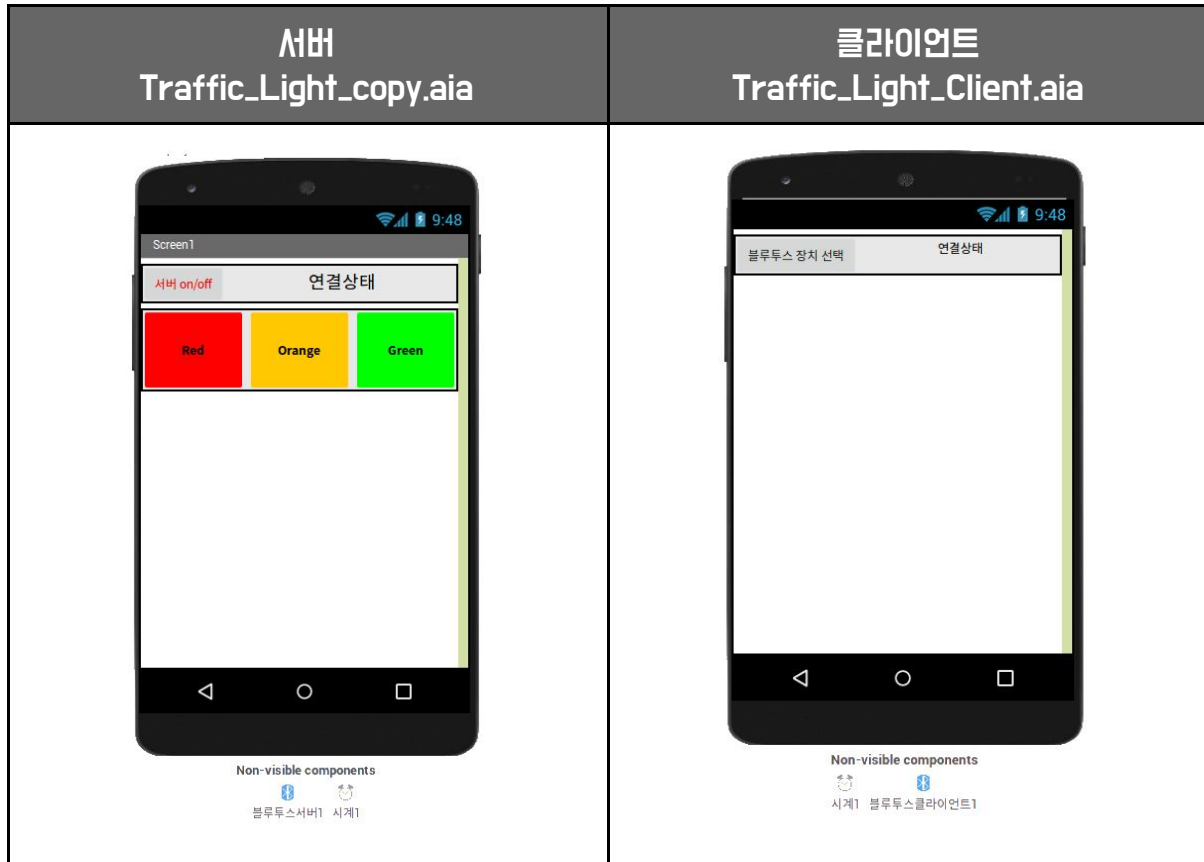
2.1.1. 요구사항

Xycar는 신호등의 교통 신호에 따라, 모터를 제어해야 한다. 신호등에 따른 주행 시 고려 사항은 다음과 같다.

- 1) Xycar는 적색/황색/녹색 원을 포착 시, 이를 신호등의 교통 신호로 인식한다.
- 2) 감지한 교통 신호의 주 색깔이 적색일 경우, Xycar는 이를 신호등의 빨간불로 인식한 후, 정차한다.
- 3) 감지한 교통 신호의 주 색깔이 황색일 경우, Xycar는 이를 신호등의 노란불로 인식한 후, 매우 느린 속도로 주행한다.
- 4) 감지한 교통 신호의 주 색깔이 녹색일 경우, Xycar는 이를 신호등의 초록불로 인식한 후, 정상 주행한다.

2.1.2. 상세설계

클래스	메서드	input	return	기능
TrafficLightDetector	conv_image	이미지 메시지	-	이미지 데이터를 OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환
	detect	-	None이나 켜진 신호등의 색	이미지에서 켜진 신호등을 찾아 색을 반환하거나 찾지 못했다면 None 반환
Autodrive	trace	-	-	켜진 신호등 색에 따른 주행



<앱 인벤터로 제작된 블루투스 통신 기반 서버-클라이언트 형식 신호등>

2.1.3. 코드의 구현

참고자료: TrafficLightDetector.py & autodrive.py

TrafficLightDetector에선 먼저 LineDetector와 같이 생성자에서 카메라 이미지 토픽을 구독하고 conv_image 메서드에서 받은 이미지 메시지를 OpenCV에서 처리할 수 있는 형태로 변환한다.

그리고, 신호등 인식의 메인 로직이 상세된 detect 메서드에서 그 이미지를 이용해 단일 채널인 그레이스케일 이미지로 변환 후, 원 허프 변환을 적용하여 명시된 조건에 맞는 이미지 상에서의 원들에 대한 정보를 반환한다.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 10, param1=200,
param2=40, minRadius=0, maxRadius=50)
```

<그레이스케일 변환 및 원 허프 변환 과정>

원 허프 변환으로부터 정상적인 원 정보를 획득했을 경우, 점등된 신호등의 색상을 구분하기 위해서 이미지의 BGR 색상 공간을 HSV 색상 공간으로 변환한다. 그 후, 얻어낸 원 정보를 이용하여 원본 이미지 내에서 각 원의 형태로 ROI를 설정하고, 색상 값을 이용해 이진화한 뒤 임계값을 이용해 점등된 신호등을 구분해낸다.

```
cv2.circle(mask, (i[0], i[1]), i[2], 255, -1)

...
m, M = (i[1] - i[2], i[0] - i[2]), (i[1] + i[2], i[0] + i[2])
circle = np.zeros((i[2] * 2, i[2] * 2), dtype=np.uint8)
cv2.circle(circle, (i[2], i[2]), i[2], 255, -1)
roi = h[m[0]:M[0], m[1]:M[1]] & cv2.inRange(s[m[0]:M[0], m[1]:M[1]], 50, 255) &
circle

cnts = [np.count_nonzero(cv2.inRange(roi, l, u)) for l, u in [(10, 30), (40, 60),
(60, 80)]]

if max(map(lambda x: x / (i[2] ** 2 * np.pi), cnts)) < 0.7:
    continue
    <원 ROI 설정 및 색상의 분포를 이용한 신호등 구분>
```

마지막으로, 그렇게 얻어낸 신호등 후보를 이용해 그 중 가장 색상값 일치 정도가 높은 것을 이용해 점등된 신호등의 색을 반환한다.

```
rog = [max(cand, key=lambda x: x[0][i]) for i in range(3)]
M = 0
for i in range(1, 3):
    if rog[i][0][i] > rog[M][0][M]:
        M = i
return ['red', 'orange', 'green'][M]
    <점등된 신호등 값의 반환>
```

AutoDrive 클래스에서는 TrafficLightDetector로부터 얻은 점등된 신호등의 색에 따라 녹색은 정상 주행, 황색은 서행, 적색의 경우엔 정차하게 차량을 조작한다.

```
light_color = self.traffic_light_detector.detect()
if light_color == 'green':
    self.driver.drive(90, 115)
elif light_color == 'orange':
    self.driver.drive(90, 110)
elif light_color == 'red':
    self.driver.drive(90, 90)
    <점등된 신호등의 색에 따른 주행>
```

2.4. 자율 주행 버스


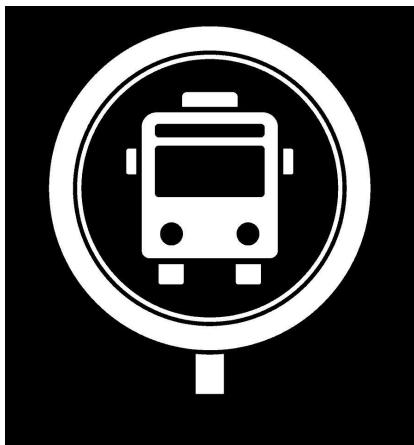
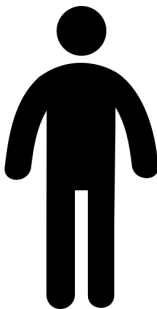
2.1.1. 요구사항

Xycar는 대중 교통인 버스의 기본적인 기능을 수행한다. 요구되는 기능은 다음과 같다.

- 1) Xycar는 정해진 버스 정류장에 정차할 수 있다.
- 2) Xycar는 버스 정류장에 정차 시, 정류장의 모든 사람이 탑승할 때까지 정차한다.
- 3) 정차한 정류장에 사람이 없더라도, 승객의 하차를 위해 2초간 정차한다.
- 4) 정해진 버스 정류장이 아닌 경우, 사람이 있더라도 이를 무시하고 주행한다.

2.1.2. 상세설계

클래스	메서드	input	return	기능
BusStopDetector	conv_image	이미지 메시지	-	이미지 데이터를 OpenCV와 NumPy가 처리할 수 있는 자료형으로 변환
	find_circles	이미지	원 이미지, 경계상자 리스트	이미지에서 원을 찾아 리스트에 원의 이미지와 원본 이미지 내에서의 경계상자를 적재하여 반환
	detect	-	표지판의 유무, 표지판 경계 상자, 인원수	이미지 내의 원 중에서 지정된 표지판과 유사도가 높은 것과 그 근처의 인원수를 세서 경계 상자와 함께 반환
Autodrive	trace	-	-	표지판의 유무와 인원수에 따른 정차 및 주행

올바른 버스 표지판(정차 O)	잘못된 버스 표지판(정차 X)	사람
		

우드락으로 제작된 버스 표지판과 사람 모형



2.1.3. 코드의 구현

참고자료: BusStopDetector.py & autodrive.py

BusStopDetector에서의 버스 표지판 인식은 TrafficLightDetector가 신호등의 원 ROI를 본뜨는 과정과 유사하게 흘러간다. find_circles 메서드에서는 아래와 같이 입력으로 주어진 이미지를 그레이스케일 이미지로 변환 작업 후, 원 허프 변환을 적용하여 명시된 조건에 맞는 이미지 상에서의 원들에 대한 정보를 반환한다.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 10, param1=200,
param2=70, minRadius=10, maxRadius=100)
```

<그레이스케일 변환 및 원 허프 변환 과정>

TrafficLightDetector의 detect 메서드와 마찬가지로, 원 허프 변환으로부터 정상적인 원 정보를 획득했을 경우, 버스 표지판의 이미지를 비교하기 위해서 이미지의 BGR 색상 공간을 HSV 색상 공간으로 변환한다. 이후, 얻어낸 원 정보를 이용하여 원본 이미지 내에서 각 원의 형태로 ROI를 설정한 후, 리스트에 원의 이미지와 경계상자를 적재하여 반환한다.

```
m, M = (i[1] - i[2], i[0] - i[2]), (i[1] + i[2], i[0] + i[2])
circle = np.zeros((i[2] * 2, i[2] * 2), dtype=np.uint8)
cv2.circle(circle, (i[2], i[2]), i[2], 255, -1)
roi = cv2.inRange(hsv[m[0]:M[0], m[1]:M[1]], np.array([0, 0, 200]), np.array([255, 50,
255]))
```

```
roi &= circle
results.append((roi, m, M))
<원 ROI 설정 및 리스트에 원의 이미지와 경계상자 적재>
```

이후, detect 메서드에서 find_circles 메서드에서 찾은 원의 이미지와 Xycar 내 미리 저장된 표지판 이미지인 template.jpg를 픽셀 단위로 비교해가며 유사도를 측정한다. 이중 template.jpg와 가장 차이가 적은 것, 즉 template.jpg와 가장 유사도가 높은 것을 찾는다.

```
if circles:
    minDiff, minRoi, min_, min_data = None, None, np.inf, None
    for circle_data in circles:
        circle = circle_data[0]
        diff = cv2.absdiff(cv2.resize(circle, self.template.shape[:2]), self.template)

        ...

        min_ = diffVal
        min_data = circle_data
<이미지 유사도 측정>
```

유사도가 가장 높은 이미지를 찾았을 경우, 버스에 탑승할 승객인 사람을 찾는 알고리즘을 실행한다. conv_image 메서드에서 처리한 이미지에서 사람이 있을만한 영역을 ROI로 설정하고, 해당영역을 그레이스케일 이미지로 변환 작업 후, Canny Edge Detection을 통해 Edge를 찾아낸다. 그리고, 이미지를 이진화 작업 이후, 흑백 이미지에서 검은 부분만을 남긴 후, OpenCV에서 제공하는 CascadeClassifier의 detectMultiScale 메서드를 이용해, 사람 형태를 인식하여, 사람의 인원 수를 구한다.

```
if minDiff is not None:
    cv2.waitKey(1)
    body_img = self.cam_img[circle_data[2][0]:min(circle_data[2][0]+120,
self.cam_img.shape[0] - 1), max(circle_data[1][1]-100, 0):min(circle_data[1][1]+100,
self.cam_img.shape[1] - 1)].copy()

    ...

    gray = ~gray
    body = self.body_cascade.detectMultiScale(gray, 1.01, 2)
    for (x,y,w,h) in body :
        cv2.rectangle(body_img,(x,y),(x+w,y+h),(0,0,255),3)
        <CascadeClassifier의 메서드 detectMultiScale을 이용한 사람 검출>
```

마지막으로, 표지판의 유무와 근처의 사람 인원 수, 표지판 경계상자를 반환한다.

3. 결론

3.1. 프로젝트 결과

3.1.1. 김신건

수행역할) AD 프로젝트를 진행하기 위한 준비물 준비, 차단기 액츄에이터 제작, 차단기 인식 코드 작성, 최종 AD 프로젝트 결과 발표, git 관리, 조장 역할

수행후기)

opencv에 대해서 수업 시간에 배운 내용을 복습하는 것 뿐만아니라 더 자세한 내용을 공부할 수 있는 기회가 되었습니다. 대학교에 와서 새로 배우는 임베디드 개념을 익히고 바로 적용해볼 수 있었으며, ROS에 대한 여러 개념을 배울 수 있었습니다. 그리고 조원들과 역할을 분배하고 프로젝트를 진행하면서 분업 프로젝트를 경험할 수 있었습니다. 분업이 잘 되어 프로젝트를 성공적으로 마무리지을 수 있었기 때문에 너무나 뿌듯하고 기쁩니다.

성찰)

성찰 항목	점수
자율주행과 관련된 주요개념, 원리 및 절차에 대한 이해 정도	9
프로젝트를 진행하면서 얻은 새로운 사실 혹은 지식의 정도	10
프로젝트 수행 시간	10
프로젝트의 개인 기여도	8

3.1.2. 강경수

수행역할) 차선 추종 주행 시 git issue 작성 및 아이디어 제시, 차량 차단기 라즈베리 파이 서보모터 연결, 차량 차단기 알고리즘, AD 프로젝트 ppt 및 보고서 작성, 자율주행 버스 아이디어 제공

수행후기)

아직 소프트웨어 햇병아리인 나에게 자율주행 자동차를 다루는 것은 교수님의 언급대로, 확실히 어려운 일이었다. 허나, 창업연계공학설계입문 실습을 하면서, 유레카 프로젝트 자율

주행 대회에 출전하면서, AD 프로젝트를 진행하면서 팀원들 덕분에 포기하지 않고, 나아갈 수 있었던 것 같다.

한 학기동안 자율주행 구동체를 움직이면서, 힘든 것도 많았지만 얻은 것이 상당히 많은 것 같다. 특히, 임베디드 시스템 쪽에 관심이 있던 나로선, 센서와 액츄에이터를 직접 다루본 경험이 목표로서의 밑거름이 될 것이라 생각한다. 또한, 프로젝트에서는 다루지 않았던 초음파센서나, IMU 센서를 더 활용해보고픈 마음이 든다.

성찰)

성찰 항목	점수
자율주행과 관련된 주요개념, 원리 및 절차에 대한 이해 정도	8
프로젝트를 진행하면서 얻은 새로운 사실 혹은 지식의 정도	10
프로젝트 수행 시간	10
프로젝트의 개인 기여도	8

3.1.3. 고현성

수행역할) 차선추종 자율주행, 신호등 판별, 자율주행 버스 내 알고리즘 및 로직 구현 및 코드 총괄 주도

수행후기) 자율주행과 AD 과제에서 알고리즘과 로직의 흐름을 설계하고 구현할 때 Bird's-eye view로의 시점 변환, 직선 허프 변환, 원 허프 변환을 OpenCV에서 사용하거나 구현해보며 소프트웨어를 통한 하드웨어로부터의 입력력을 통한 데이터 추출 및 가공과 출력을 통한 조작으로 임베디드 시스템에서의 소프트웨어 개발에 대한 많은 지식을 쌓게 되어 좋았다. 이러한 저명한 알고리즘을 OpenCV, NumPy와 같은 고수준 라이브러리에서 직접 구현 및 사용해본 경험 또한 차후의 소프트웨어 개발에 도움을 줄 것이라 생각한다. 또, 팀 프로젝트를 진행하며 조원들과 함께 협업하여 모든 과제를 성공적으로 해내는 좋은 경험을 얻었다.

성찰)

성찰 항목	점수
자율주행과 관련된 주요개념, 원리 및 절차에 대한 이해 정도	9
프로젝트를 진행하면서 얻은 새로운 사실 혹은 지식의 정도	8
프로젝트 수행 시간	9
프로젝트의 개인 기여도	9

3.1.4. 곽다윗

수행역할) 신호등 판별에 필요한 신호등앱을 만들었다. 신호등 판별과 자율주행 버스 코드 구현에 필요한 아이디어와 영상을 찍는데 필요한 시나리오를 제안하는 동시에 보조 역할을 수행하였다. 차선 인식 자율주행에서는 차를 옮기는 일과 코드 수정에 필요한 아이디어를 제공했다.

수행후기) 창연공 수업 때 배운 ROS, OpenCV, Python 등을 사용하여 xycar를 제어하는 것을 가지고 차선인식 자율주행, 신호등 판별, 버스 정류장을 만들면서 서드파티 라이브러리인 openCV, numpy에서 완성도가 높은 함수들을 사용하면서 도움을 많이 받았고, 동영상의 프레임을 가지고 무엇을 판단하는 코딩들을 하면서 openCV에 대한 이해도가 높아진 것 같다. 또한, 팀프로젝트를 진행하면서 팀원들이 서로 각자 맡은 역할을 맡아도 하고 있고 팀원들이 활동을 적극적으로 참여해서 조별과제의 좋은 기억으로 남을 거 같다.

성찰)

성찰 항목	점수
자율주행과 관련된 주요개념, 원리 및 절차에 대한 이해 정도	8
프로젝트를 진행하면서 얻은 새로운 사실 혹은 지식의 정도	10
프로젝트 수행 시간	9
프로젝트의 개인 기여도	8

3.1.5. 김예리

수행역할) 자율주행 알고리즘 의견 제시, 신호등 판별 알고리즘 의견 제시, 자율주행 버스 알고리즘 의견 제시, 영상 촬영

수행후기)

그동안 실습 과제로 나왔던 과제들을 해온 후에 이번 ad 프로젝트를 진행하여 배운 것을 바탕으로 더 한 단계 높아진 수준의 내용을 구현할 수 있었던 것 같다. Xycar가 실제 자동차처럼 다양한 교통환경적 미션을 수행하도록 하면서 처음에 구상했던 것 보다 점점 더 정밀하게 수정하여 더 좋은 결과물이 나온 것 같아 뿌듯하다. 그리고 처음 진행하기로 한 신호등 판별과 차량 차단기를 구현 한 후에 더 도전해볼 미션으로 자율 주행 버스를 기획안에 넣었는데 이 미션까지 해볼 수 있었고 실제로 구현까지 할 수 있었기에 의미있는 시간이었다. 또한 이 모든 프로젝트를 진행하면서 조원들과의 역할 분담이나 의견 조율이 잘 되었기 때문에 원활히 모든 미션을 구현할 수 있었다.

성찰)

성찰 항목	점수
자율주행과 관련된 주요개념, 원리 및 절차에 대한 이해 정도	8
프로젝트를 진행하면서 얻은 새로운 사실 혹은 지식의 정도	9
프로젝트 수행 시간	9
프로젝트의 개인 기여도	8