AD프로젝트 보고서

2019년 12월 15일

작성자
20191660 전병우
20191668 정태성
20191654 이현준
20175163 박정환

목차

서론

목표

바쁜 부모님들을 위해 자율주행 자동차 xycar를 이용해 자녀들의 통학을 도와주는 스쿨버스를 구현한다. 학교부터 집까지 안전한 자율주행을 위하여 스쿨버스에는 첫번째로 과속방지턱을 전방 카메라를 통해 인식하여 진입시 감속주행하고 구간을 통과한 후에는 다시 가속하는 기능을 구현한다. 두번째로는 전방카메라를 이용해 설치된 QR코드를 인식하고 스쿨존 구간에서 서행하고 학교앞에서 정지하는 기능을 구현한다.마지막으로 전방카메라를 이용해 색을 감지하여 신호등을 인식하고 신호에 맞는 알맞은 상호작용을 하도록 구현한다.

연구동기

하나금융경영연구소,K-ICT빅데이터센터의 자료에 따르면 직장인들이 저녁 6시 이전에 퇴근하는 비율은 20%에 불과한 것으로 나타났다. 맞벌이 부모 가정을 흔하게 볼 수 있는 현재 대한민국 사회에서 자녀들을 학교에서 집까지 데려다 주기위한 시간을 마련하기란 쉽지 않은 일이다. 하지만 여러가지 위험상황에 대해 성인에 비해 훨씬 취약한 청소년 자녀를 홀로 통학시키는 방법은 자녀에게는 부담감을 부모에게는 불안감을 주기때문에 이를 해결하기 위한 대안이 필요하다 생각했다. 업무로 인해 자녀의 통학시간을 함께 할 수 없는 부모님들을 위하여 생각한 대안은 자녀들이 안전하게 학교를 오갈 수 있도록 자율주행 스쿨버스를 만든다면 인건비와 시간 같은 물리적인 자원을 효과적으로 절약하면서도 부모님들과 자녀 모두의 불안감을 해소해주는 효율적인 복지수단이 될 수 있다는 이유로 다음과 같은 프로젝트를 진행하게 되었다.

역할분담

팀원	역할 분담
이현준	메인주행노드 담당 , 회의 참가, 보고서 작성
정태성	신호등인식 노드 담당, 회의 참가, 보고서 작성
전병우	QR코드 인식 노드 담당, 회의 참가, 보고서 작성
박정환	과속방지턱 인식 노드 담당, 회의 참가, 보고서 작성
장민석	회의 참가

본론

- ## 노드

1.QR코드 인식

1.1 코드 구현

QR코드를 인식하기 위한 클래스를 만들어 전방 카메라를 이용해 QR코드를 인식하여 차량이 스쿨존에선 QR코드를 읽어 'slow'문자열을 반환하여 서행시키도록 하며 차량이 학교 앞에선 'stop'문자열을 반환하여 학교앞에서 정차하도록 코드를 구현하였다.

먼저 전방카메라 노드로 부터 받은 카메라 메세지를 cv이미지로 변경하여 흑백이미지로 바꾸어 주고

pyzbar모듈에 decode함수를 이용하여 흑백화한 이미지를 해석한다. 해석한 이미지의 모서리 위치를 찾아 해당하는 화면 위치에 사각형으로 나타내주어 QR코드를 읽었음을 나타내어 주고 utf-8방식으로 읽어낸 데이터 값을 문자열로 리턴하여준다.

1.2 소스코드

```python
import cv2
import numpy as np
from pyzbar import pyzbar
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import rospy

# qr코드를 읽고 사용하기 위해 pyzbar를 사용한다.
class QRcode:
    def __init__(self,topic):
        self.bridge = CvBridge()
        self.cv_image = np.empty(shape=[0])
        self.scan_node = rospy.Subscriber(topic, Image, self.QRcode)

    def QRcode(self, data):
        self.cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        gray = cv2.cvtColor(self.cv_image, cv2.COLOR_BGR2GRAY)
        #불러온 이미지를 gray 이미지로 바꾸어준다.
        decoded = pyzbar.decode(gray)
        #pyzbar에 있는 decode라는 함수로 gray라는 이미지를 해석한다.
        for d in decoded:
            x, y, w, h = d.rect
        #발견한 qrcode의 왼쪽 상단 모서리, 오른쪽 상단 모서리, 왼쪽 하단 모서리
        #오른쪽 하단 모서리의 위치를 찾는다.

            barcode_data = d.data.decode("utf-8")

```

QR코드를 읽기위해 init 메소드에서 전방카메라를 구독한 다음 QRcode를 콜백하여준다. cv_image로 이미지 메세지를 cv이미지로 바꾸어준다음 이를 흑백으로 바꾸어 pyzbar.decode를 이용해 QR코드를 읽어온 다음 rect를 이용해 카메라속 QR코드의 위치를 얻고 data.decode를 이용해 QR코드 속에 들어있는 문자열 정보를 utf-8방식으로 읽어온다.

```
29          cv2.rectangle(self.cv_image, (x, y), (x + w, y + h), (0, 0, 255), 2)
30          #파란색 사각형을 검출된 qr코드 위에다 그린다.
31          self.text = str(barcode_data)
32          #바코드의 데이타(slow or stop)을 self.text에 보내준다.
33          print(self.text)
34          return self.text
35          #self.text를 리턴해준다.
```

QR코드에 위치에 해당하는 부분에 붉은색 사각형을 만들어 표시해주고 읽어온 정보는 문자열로 자료형을 바꾸어 리턴해준다.

<class QRcode>

메소드	역할
QR코드	영상속 QR코드를 인식하여 읽고 해당하는 문자열을 리턴

변수	역할
bridge	CvBridge 클래스를 호출하여 할당받음
scna_node	카메라 노드를 구독하여 정보를 받아오고 QRcode 메소드를 콜백
cv_image	노드 메세지를 cv이미지로 바꾼 정보를 받음
gray	QR코드를 읽기위한 영상을 흑백으로 바꿈
decoded	pyzbar.decode를 이용해 영상속 바코드를 해석
x, y, w, h	인식된 바코드의 모서리 좌표값을 각각 위치에 할당
barcode_data	data_decode를 이용해 utf-8 방식으로 QR코드속 정보를 읽어옴
text	읽어온 정보를 문자열로 변환

2.신호등 인식

2.1 코드구현

전방 카메라에 이미지에서 신호등을 감지하기 위해 신호등색에 해당하는 색을 추출한다. 이를 위하여 전방카메라 노드로 부터 영상정보를 구독하여 메세지 형태로 받은 영상정보를 cv 이미지로 변경하여 roi를 추출하고 추출한 영상을 hsv로 변환하여 색에 해당하는 hsv 범위를 정하여 이진화 시킨다음 이진화된 정보를 원본 영상과 합쳐 원하는 색이 추출되었는지 디버깅 하는 용도로 사용할 수 있는 이미지를 얻어 낸다. 신호등 감지를 위해서는 이진화된 이미지에 픽셀 수 를 검사하여 픽셀 수가 일정량 이상 감지될 경우 빨간불일때는 0이 주황불 일때는 1이 초록불 일때는 2인 정수값이 반환되도록 설정하였고 빨간색-주황색-초록색순으로 검사하며 만약 조건을 만족하는 조건문이 생기며 즉시 return 시켜 코드를 빠져나가도록 설계하였다.

2.2 소스코드

```python
import cv2
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import rospy

class TrafficDetect:

    def __init__(self, topic):
        self.bridge = CvBridge()
        self.cam_img = np.zeros(shape=(480, 640, 3), dtype=np.uint8)
        self.detect_node = rospy.Subscriber(topic, Image, self.makeROI)

    def makeROI(self, data):
        self.cam_img = self.bridge.imgmsg_to_cv2(data, "bgr8")
        self.roi = self.cam_img[100:300, 500:630]
        self.img_hsv = cv2.cvtColor(self.roi, cv2.COLOR_BGR2HSV)
```

init 메소드에서 카메라를 구독하여 영상정보를 얻어올 수 있도록 한 후 makeROI를 콜백하여준다 makeROI에서는 이미지를 원하는 부분으로 잘라 roi를 설정한 후 hsv방식으로 영상을 변경하여 수정이 용이하도록 설정한다.

```python
def detectTraffic(self):
    lower_red = (0, 20, 20)
    upper_red = (5, 255, 255)
    img_mask_red = cv2.inRange(self.img_hsv, lower_red, upper_red)
    img_result_red = cv2.bitwise_and(self.roi, self.roi, mask=img_mask_red)
    gray = cv2.cvtColor(img_result_red, cv2.COLOR_BGR2GRAY)
    ret, check = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)
    if cv2.countNonZero(check) >8000:
        return 0

    lower_orange = (5, 30, 30)
    upper_orange = (20, 255, 255)
    img_mask_orange = cv2.inRange(self.img_hsv, lower_orange, upper_orange)
    img_result_orange = cv2.bitwise_and(self.roi, self.roi, mask=img_mask_orange)
    gray = cv2.cvtColor(img_result_orange, cv2.COLOR_BGR2GRAY)
    ret, check = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    if cv2.countNonZero(check) > 5000:
        return 1

    lower_green = (60 - 10, 30, 30)
    upper_green = (60 + 10, 255, 255)
    img_mask_green = cv2.inRange(self.img_hsv, lower_green, upper_green)
    img_result_green = cv2.bitwise_and(self.roi, self.roi, mask=img_mask_green)
    gray = cv2.cvtColor(img_result_green, cv2.COLOR_BGR2GRAY)
    ret, check = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    if cv2.countNonZero(check) > 100:
        return 2
```

가공한 영상에서 신호등을 감지하기 위해 lower, upper 범위를 hsv방식으로 설정하여 원하는 색을 제외하고 검은색으로 바뀌도록 이진화 시킨다. inrange를 이용하여 이진화 시킨 영상을 원하는 색이 잘 추출되었는지 확인하기 위해 bitwise_and를 이용하여 원본영상과 이진화 영상중 이진화영상에서 하얀부분만 원본과 합쳐져 색을 가지도록 만들어 디버깅할 수 있는 영상을 얻는다. 원하는 색이 검출되었는지 확인하기 위해서 countNonZero를 이용하는데 이진화한 이미지만 사용할 수 있는 방법이므로 만들어놓은 결과물을 다시 threshold를 이용해 이진화 시켜 이진화 이미지속 하얀색 픽셀수를 세어 원하는 값이 나온다면 바로 값을 리턴하여 코드가 종료될 수 있도록 구현하였다.

<class TrafficDetect>

메소드명	역할
makeROI	받아온 노드 메세지를 cv 이미지로 바꾸어 주고 roi를 설정하여 hsv 방식으로 변경
detectTraffic	이미지를 이진화하여 원하는 색을 추출하여 이진화한 이미지의 픽셀수를 세어 신호를 감지함 이진화한 이미지를 원본과 합쳐 원하는 색이 정확히 추출되었는지 디버깅에 사용

변수명	역할
bridge	CvBridge를 호출하여 할당받음
detect_node	전방카메라 노드로 부터 구독을 요청함

cam_img	전방 카메라 노드로부터 받아온 이미지 메세지를 cv 영상으로 바꾼 정보를 할당받음
roi	영상의 roi를 설정함
img_hsv	roi영상을 hsv방식으로 변경
lower_color	색 추출을 위한 hsv 범위의 최솟값
upper_color	색 추출을 위한 hsv 범위의 최댓값
img_mask_color	cv2.inrange를 이용하여 해당하는 색 부분만 이진화해 추출함
img_result_color	cv2.bitwise_and를 이용해 원본과 img_mask_color 를 합쳐 색이 추출되었는지 디버그에 활용하기 위한 영상을 얻음
gray	결과값을 흑백이미지로 변경
ret, check	흑백 이미지를 이진화 시켜 ret에는 할당여부를 boolean으로 check에는 이진화한 정보를 각각 할당하여줌

3.과속 방지턱

3.1 기능 구현

하얀색과 노란색으로 조합되어있는 과속방지턱을 전방카메라를 통해 감지하여 구간을 진입할때 속도를

감속하고 IMU센서(pitch값만을 이용)를 이용해 구간을 통과한 후에는 다시 정상적인 주행을 할 수 있도록

하는 boolean값을 리턴한다. 메인 주행 노드에서는 isBump라는 메소드를 호출하고 isBump 메소드는 과속방지턱 구간에서는 True값을 과속방지턱 구간이 아닌곳에서는 False값을 리턴한다. 과속 방지턱 구간에 진입시 판단의 근거는 카메라 프레임에서 과속방지턱을 감지하기 위한 ROI영역을 설정한 후 해당 영역을 HSV변환한다. 이후 노란색 범위값에 대하여 이진화 한다. ROI영역의 30%이상이 흰픽셀일 경우 과속방지턱 구간으로 판단한다. 과속 방지턱 구간 진입이후 탈출에 대한 판단의 근거는 다음과 같다. 과속방지턱 구간 진입시 pitch 값을 측정하고 기록한다. 이후 체크 포인트 변수 3개를 통해 판단을 한다. 체크 포인트는 과속방지턱을 올라갈때 한번, 내려갈때 한번, 마지막으로 평지에 내려 왔을때이다. pitch값이 정확하지 않으므로 오차범위를 설정해두었다.

3.2 소스코드

ImuRead 클래스는 에서 IMU센서의 정보를 읽어온다.

```python
import rospy
from diagnostic_msgs.msg import DiagnosticArray
from Camera import Camera


class ImuRead:

    def __init__(self, topic):
        self.roll = -1
        self.pitch = -1
        self.raw = -1
        rospy.Subscriber(topic, DiagnosticArray, self.read_data)

    def read_data(self, data):
        status = data.status[0].values
        self.roll = status[0].value
        self.pitch = status[1].value
        self.yaw = status[2].value

    def get_data(self):
        # return float(self.roll), float(self.fitch), float(self.yaw)
        return float(self.fitch)
```

ImuRead 클래스는 imu센서가 감지한 데이터 값들을 get_data 메소드를 통해 반환해준다.

과속방지턱 판단에서는 pitch값만 필요하므로 get_data메소드 호출시 pitch값만 반환하도록 해주었다.

```python
class Camera:

    def __init__(self, topic):
        self.scan_width, self.scan_height = 200, 90  # 200, 90
        self.bridge = CvBridge()
        self.cv_image = np.empty(shape=[0])
        self.detect_node = rospy.Subscriber('usb_cam/image_raw', Image, self.conv_image)
        self.mask = np.zeros(shape=(self.scan_height, self.scan_width),dtype=np.uint8)
        self.pixel_cnt_threshold = 0.3 * self.scan_width * self.scan_height

    def conv_image(self, data):

        roi_vertical_pos = 200
        h = 480
        w = 640
        roi_horizon_pos = (w - self.scan_width) // 2
        lower_yellow = np.array([20, 100, 100])
        upper_yellow = np.array([30, 255, 255])  # 노란색의 HSV 범위값

        self.cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        roi = self.cv_image[roi_vertical_pos:roi_vertical_pos + self.scan_height, roi_horizon_pos: w-roi_horizon_pos]
        hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
        self.mask = cv2.inRange(hsv, lower_yellow, upper_yellow, 50)

        # 노란색 비율이 30% 이상일때 과속방지턱이라 여긴다

    def detectBumper(self):
        if cv2.countNonZero(self.mask) > self.pixel_cnt_threshold:
            return True

        return False
```

camera.py 의 Camera클래스는 카메라로부터 읽어들인 영상 데이터를 과속방지턱을 감지하기 위한 ROI영역으로 자른 다음 자른영역을 HSV변환하고 이진화 한다. 과속방지턱의 색상은 흰색과 노란색이 섞여 있다. 노란색을 이진화하기 위해 임계값을 H:20~30, S:100~255, V:100~255 로 설정하였다. detectBumper 메소드는 이진화한 영역에서 흰색의 비율이 영역의 30%이상일때 True (과속방지턱을 발견)를 리턴한다. 디폴트 반환값은 False이다.

```
class SpeedBump:

    def __init__(self):
        self.status = False
        self.pitch = -1
        self.imu = ImuRead('')
        self.checkUp = False
        self.checkDown = False
        self.checkNormal = False

    def isBump(self):
        if self.status == False:
            ### 여기서 roi에서 노흰 비율 학인하고
            switch = Camera('usb_cam/image_raw').detectBumper()

            if switch:
                self.status = True
                self.pitch = self.imu.get_data()
        else:
            if self.imu.get_data() > self.pitch + 8:
                self.checkUp = True

            if self.imu.get_data() < self.pitch - 9:
                self.checkDown = True

            if self.checkUp and self.checkDown:
                if self.pitch - 2 < self.imu.get_data() < self.pitch + 2:
                    self.checkNormal = True
                if self.checkNormal:
                    self.resetData()

    def resetData(self):
        self.checkUp = False
        self.checkDown = False
        self.checkNormal = False
        self.status = False
        self.pitch = -1

    def getStatus(self):
        self.isBump()
        return self.status
```

<클래스 SpeedBump>

메소드명	역할
isBump	카메라노드와 imu센서로 부터 받아온 데이터를 바탕으로 과속방지턱

	구간인지를 판단한다.
resetData	과속방지턱 구간에서 벗어날 때 판단의 근거가 됐던 변수값들을 초기화 시킨다.
getStatus	메인주행코드에 현재 주행구간이 과속방지턱 구간인지 아닌지를 알려준다

변수명	역할
status	boolean변수,True- 과속방지턱 주행 구간을 의미/ False- 일반 주행 구간을 의미
pitch	imu센서로 부터 읽어들인 pitch값을 저장하는 변수
checkUp	과속방지턱 상승 구간에 진입했는지 판단하는 boolean변수, true-상승구간을 지났음을 의미
checkDown	과속방지턱 하강 구간에 진입했는지 판단하는 boolean변수, true-하강구간을 지났음을 의미
checkNormal	과속방지턱 구간을 지나고 평지에 도달했는지 판단하는 boolean변수, true-평지에 도달했음

getStatus를 autodrive에서 호출하면 해당하는 boolean값을 리턴하며 주행에 반영할 수 있도록 하였다.

```
56      def resetData(self):
57          self.checkUp = False
58          self.checkDown = False
59          self.checkNormal = False
60          self.status = False
61          self.pitch = -1
62
63      def getStatus(self):
64          self.isBump()
65          return self.status
```

4.차선 검출

4.1 기능 구현

카메라에서 받아온 영상을 처리하여 왼쪽 차선과 오른쪽 차선을 인식하게 해준다.
왼쪽 차선과 오른쪽 차선의 위치 값을 리턴해줘 주행코드에서 쓰이게 한다.

4.2 소스코드

메소드명	역할
conv_image	이미지를 차선주행이 용이하도록 변환
detect_lines	roi로 잘라온 이미지에서 차선을 검출함
show_image	화면에 디버깅을 위한 사각형을 표시

변수명	역할
cv_image	이미지 메세지를 cv 이미지로 변경
frame	이미지를 원하는 크기로 조정함
blur	가우시안 블러를 적용하여 윤곽선을 부드럽게함
dst	Canny를 이용하여 차선을 따냄
cdst	이미지를 BGR방식으로 변경
lines	이미지를 허프변환 함
bin	허프 변환한 이미지를 inrange를 이용해 이진화함
view	디버깅을 위해 색을 BGR방식으로 변경
area	차선 검출을 위해 허프변환한 bin을 잘라 roi로 만듬
left,right	검출된 차선의 좌표를 할당받음
last_l, last_r	받은 차선의 좌표를 저장해 다음 차선검출에 활용함

```python
import rospy, time
import cv2
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge


class LineDetector:

    def __init__(self, topic):

        self.bridge = CvBridge()
        self.cv_image = np.empty(shape=[0])
        self.value_threshold = 180
        self.image_width = 640
        self.image_middle = 320
        self.image_middle = 320
        self.scan_height = 40
        self.area_width, area_height = 20, 10
        self.roi_vertical_pos = 310
        self.row_begin = (self.scan_height - area_height) // 2
        self.row_end = self.row_begin + area_height
        self.pixel_cnt_threshold = 0.4 * self.area_width * area_height
        # 차선을 인식하기 위해 설정해준 값으로 self.pixel_cnt_threshold 값보다 인식되는
        # 값이 더 크면 차선으로 인식한다.
        self.detect_node = rospy.Subscriber(topic, Image, self.conv_image)
        self.dleft, self.dright = 0, 0
        # 왼쪽 차선과 오른쪽 차선의 전값을 저장해주기 위한 값을 초기화 해준 것이다.

```

```python

    def conv_image(self, data):

        cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        frame = cv_image[self.roi_vertical_pos:self.roi_vertical_pos + self.scan_height, :]
        # ROI를 설정해준다.
        blur = cv2.GaussianBlur(frame, (5, 5), 0)
        # frame을 GaussianBlur처리 해준다.
        dst = cv2.Canny(blur, 50, 200, None, 3)
        # blur한 이미지를 캐니 처리를 통해 윤곽선을 그려준다.
        cdst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)
        # GRAY그림을 BGR로 바꾸어준다.
        lines = cv2.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)
        # 캐니 처리가 된 dst를 허프변환 처리를 해준다.
        if lines is not None:
            for i in range(0, len(lines)):
                l = lines[i][0]
                cv2.line(cdst, (l[0], l[1]), (l[2], l[3]), (0, 0, 255), 5, cv2.LINE_AA)

        lbound = np.array([0, 0, self.value_threshold], dtype=np.uint8)
        ubound = np.array([131, 255, 255], dtype=np.uint8)
        # 이진화를 하기 위해 하한값과 상한값을 잡아준다.

        hsv = cv2.cvtColor(cdst, cv2.COLOR_BGR2HSV)
        # BGR그림을 HSV처리를 해준다.
        self.bin = cv2.inRange(hsv, lbound, ubound)
        # HSV처리가 된 그림을 이진화 해준다.
        self.view = cv2.cvtColor(self.bin, cv2.COLOR_GRAY2BGR)
        # 녹색 사각형을 화면에 표시하기 위해서 BGR로 바꾸어준다.

```

이미지를 변환시켜주는 메소드이다 카메라 노드에서 정보를 받아와 이를 roi로 자르기도 하고 허프변환을 통해 차선인식을 위한 이미지로 바꾸어 차선인식이 용이하도록 가공하는 과정을 거친다. 처음에는 bgr로 이미지를 불러와 이를 gaussianblur 처리로 윤곽선을 부드럽게 만들어 canny edge 방식으로 선을 추출해내 허프변환을 통해 그림을 다시 색칠한다. 이러한 그림을 다시 이진화 처리해주고 roi가 정확하게 표시되었음을 알려주기 위한 사각형을 그리기 위해 이미지를 색 표현이 가능한 hsv 방식으로 다시 바꾸어 주며 작업을 마무리한다.

```
58
59      def detect_lines(self):
60          # Return positions of left and right lines detected.
61
62          for l in range(self.image_middle, self.area_width, -1):
63              # self.image_middle값을 기준으로 왼쪽으로 1이 움직인다.
64              area = self.bin[self.row_begin:self.row_end, l - self.area_width:l]
65              # 바뀌는 l값에 따라 area가 바뀌게 된다.
66              if cv2.countNonZero(area) > self.pixel_cnt_threshold:
67                  self.left = l
68                  break
69              # area 의 흰색 값이 self.pixel_cnt_threshold 보다 값이 크면 self.left를
70              # l로 설정해 주고 for 문을 탈출한다.
71              else:
72                  self.left = -1
73              # area 의 흰색 값이 self.pixel_cnt_threshold 보다 값이 작으면 self.left를
74              # -1로 설정해 준다.
75
76          for r in range(self.image_middle, self.image_width - self.area_width):
77              # self.image_middle 을 기준으로 오른쪽으로 r이 움직인다.
78              area = self.bin[self.row_begin:self.row_end, r:r + self.area_width]
79              # 바뀌는 r값에 따라 area 가 바뀌게 된다.
80              if cv2.countNonZero(area) > self.pixel_cnt_threshold:
81                  self.right = r
82                  break
83              # area 의 흰색 값이 self.pixel_cnt_threshold 보다 값이 크면 self.right를 r로
84              # 설정해 주고 for 문을 탈출한다.
85              else:
86                  self.right = -1
87              # area 의 흰색 값이 self.pixel_cnt_threshold 보다 값이 크면 self.right를
88              # 1로 설정해 준다.
89
90          if self.left > 0 and self.right > 0 and \
91                  self.right - self.left < 450:
92              if self.dleft == -1:
93                  self.left = -1
94              elif self.dright == -1:
95                  self.right = -1
96          # 왼쪽 차선과 오른쪽 차선 두개다 검출 되고 오른쪽 차선의 값과 왼쪽 차선의 값의 차가
97          # 450보다 크면(중앙선을 인식하지 않도록함이다)
98          self.dleft, self.dright = self.left, self.right
99          # 이전 좌표값을 저장
100         return self.left, self.right
101
```

추출된 roi 이미지에서 바깥쪽 부터 안쪽으로 들어가며 이진화된 하얀색 부분을 검사한다. 그 구분이 일정량을 넘어갈 경우 차선으로 인식하고 해당하는 좌표를 리턴해주도록 하였으나 중앙선까지 차선으로 인식하는 버그가 발생하여 이를 해결하기 위해 차선이 일정 거리이상 가까우며 차선을 잃었던 상태라면 이를 무시하도록 코드를 작성해 중앙선을 인식하여도 값을 리턴하지 않도록 하여 오류를 수정했다.

```
101
102     def show_images(self, left, right):
103
104         if left != -1:
105             self.lsquare = cv2.rectangle(self.view,
106                                          (left, self.row_begin),
107                                          (left - self.area_width, self.row_end),
108                                          (0, 255, 0), 3)
109         # 왼쪽 차선이 발견되었을때 왼쪽 차선 주변에 초록색 사각형을 그리는 코드이다.
110
111         else:
112             print("Lost left line")
113         # 왼쪽 차선을 잃었을때 콘솔창에 "Lost left line"이라는 문구를 띄우는 코드이다.
114
115         if right != -1:
116             self.rsquare = cv2.rectangle(self.view,
117                                          (right, self.row_begin),
118                                          (right + self.area_width, self.row_end),
119                                          (0, 255, 0), 3)
120         # 오른쪽 차선이 발견되었을때 오른쪽 차선 주변에 초록색 사각형을 그리는 코드이다.
121         else:
122             print("Lost right line")
123         # 오른쪽 차선을 잃었을때 콘솔창에 "Lost right line"이라는 문구를 띄우는 코드이다.
124
125
126 if __name__ == "__main__":
127     det = LineDetector()
128     time.sleep(1)
129     while not rospy.is_shutdown():
130         det.show_images(det.detect_lines()[0], det.detect_lines()[1])
```

roi 이미지에 문제가 발생하는지 디버깅 하기위한 화면으로 roi에 차선이 인식되면 해당 부분에 사각형을 추가하기 위한 코드이지만 컴퓨터 자원 절약을 위해 화면을 띄우는 코드는 제거한 상태이다.

5.자율 주행

5.1 기능 구현

차선 검출 코드에서 받아온 왼쪽 차선과 오른쪽 차선의 위치값을 통해 자율주행차를 도로 가운데에서 주행 할 수 있게한다.

또한 방지턱, QR코드, 신호등의 값과 조향각을 통하여 속도를 변환 시켜준다.

5.2 소스코드

```python
#!/usr/bin/env python

import rospy, time
from linedetector import LineDetector
from motordriver import MotorDriver
from obstacledetector import ObstacleDetector
from trafficDetect import TrafficDetect
from speedBump import SpeedBump
from QRcode import QRcode

class AutoDrive:
        def __init__(self):
            rospy.init_node('xycar_driver')
            self.line_detector = LineDetector('/usb_cam/image_raw')
            self.driver = MotorDriver('/xycar_motor_msg')
            self.obstacle_detector = ObstacleDetector('/ultrasonic')
            self.traffic = TrafficDetect('usb_cam/image_raw')
            self.bump = SpeedBump()
            self.code = QRcode('usb_cam/image_raw')

        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)
            angle = self.steer(line_l, line_r)
            speed = self.accelerate(angle, obs_l, obs_m, obs_r)
            if speed == 0:
                angle = 0
            self.driver.drive(angle + 90, speed + 90)
```

위의 코드는 자율 주행을 위한 값들을 준비하는 코드이다. 이 코드를 바꾼다고 자율 주행이 원활히 되지는 않는 코드임으로 초반부터 마지막까지 변화가 없었다

.

```
33          def steer(self, left, right):
34              print(left, right)
35              if left == -1:
36                  if 320 < right < 390:
37                      angle = -50
38                  else:
39                      angle = (550 - right)/(-3)
40              elif right == -1:
41                  if 250 < left < 320:
42                      angle = 50
43                  else:
44                      angle = (left - 90)/(3)
45              else:
46                  angle = 0
47
48              return angle
49
```

초반의 주행코드는 단순하고 수학적이지 못한 코드이다. 단순히 왼쪽 차선만을 보고 주행하는 코드로써 차선 인식이 잘 되지 않을 때는 주행이 잘 안되어 차선 이탈을 한다. 또한 수학적으로 짜지 않았고 왼쪽 차선이 움직일 때마다 조향각을 바꾸는 코드이다. 이 코드로 주행시에 차선 이탈등의 문제점이 많아 값 수정으로는 자율 주행을 완성 시키지 못할 것으로 판단하여 코드를 엎고 다른 코드를 짜보기로 판단했다.

```python
    def accelerate(self, angle, obs_l, obs_m, obs_r):
        traffic_color = self.traffic.detectTraffic()
        # 받아온 신호등 값
        text = self.code.QRcode()
        # 받아온 QR코드 값
        bump = self.bump.getStatus()
        # 받아온 방지턱 값
        QRcnt = 0
        # QR코드로 "slow"를 받을 때 다시 전진 할 수 있도록 변수를 줬다.
        speed = 0
        if text == "slow":
            speed = 30
        # "slow"를 받아오면 speed값을 30으로 받아온다.
        elif text == "stop":
            speed = 0
        # "stop"을 받아오면 speed값을 0으로 받아오고
            QRcnt += 1
            # QRcnt 값을 1씩 더한다.
            # QRcnt 값이 40이 됐을 경우 text값을 "slow"로 줘서 다시 전진하도록 한다
            if QRcnt == 40:
                text = "slow"
        if bump == True:
            speed = 20
        #방지턱을 인식하면 20으로 감속한다.
        elif bump == False:
        #방지턱을 인식하지 않았을 때
            if traffic_color == 0:
                print("red")
                speed = 0
            # 신호등 값이 0이면 빨간 불이므로 정지한다.
            elif traffic_color == 1:
                print("orange")
                speed = 20
```

```
83                  #신호등 값이 1이면 주황 불이므로 감속한다.
84                  else:
85                      speed = 30
86              # 그 외의 상황은 어린이 보호 구역이 아니므로
87              # 빠른 속도로 주행하도록 한다.
88              else:
89                  if 0 < obs_m < 50 and 0 < angle:
90                      speed = 0
91                  elif angle <= -40 or angle >= 40:
92                      speed = 35
93                  elif angle <= -25 or angle >= 25:
94                      speed = 45
95                  else:
96                      speed = 50
97
98              return speed
99
100         def exit(self):
101             print('finished')
```

```
103 if __name__ == '__main__':
104     car = AutoDrive()
105     time.sleep(1)
106     rate = rospy.Rate(10)
107     while not rospy.is_shutdown():
108         car.trace()
109     rate.sleep()
110     rospy.on_shutdown(car.exit)
```

accelerate함수는 QR코드, 방지턱, 신호등들의 값과 조향각에 따라 곡선인지 직진인지 판단하여 속도를 그에 맞게 변화시켜주는 코드이다. 속도 또한 바꿔주어 차선인식에 도움을 줄 수 있는 부분이 있어 값 수정을 어느정도 해주어야 한다.

위의 코드는 최종 코드로 그 전의 코드보다 수학적이고 계산적으로 짜여진 코드이다. 다양한 상황에 맞게 여러 변수상황에 대처 가능하며 조향각을 부드럽게 바꿀 수 있기에 xyCar가 갑작스럽게 조향각을 바꾸는 일이 없게끔 했다. 초반의 코드에서 너무나도 단순한 코드의 문제점을 느끼고 수학적으로 접근한 결과 자율주행 스튜디오의 트랙을 완주 할 수 있는 코드를 완성시켰다.

<class AutoDrive>

메소드	역할
trace	steer와 accelerate 메소드에서 속도와 조향각을 얻어와 주행에 반영
steer	차선인식 정보를 바탕으로 알맞은 조향각을 리턴
accelerate	조향각에 따른 속도를 조절

변수명	역할
line_detector	LineDetector 클래스를 호출
driver	MotorDriver 클래스를 호출
obstacle_detector	ObstacleDetector 클래스를 호출
traffic	TrafficDetect 클래스를 호출
bump	SpeedBump 클래스를 호출
code	QRcode 클래스를 호출
obs_(l,m,r)	초음파 센서의 거리를 각각 할당 받은 값
line_(l.r)	카메라에서 인식된 양쪽 차선의 좌표값
angle	자동차의 조향각
speed	자동차의 속력값
traffic_color	감지한 신호의 해당하는 정수 값
text	감지한 QR코드의 해당하는 문자열 값
bump	과속 방지턱을 지나는 중인지에 대한 boolean값
QRcnt	정지된 상태에서 다시 주행하기 위해 세어주는 정수 값

결론

xycar라는 모형자동차를 이용해 자율주행을 구현해 나가면서 공학적 목표를 도달하기 위해 실제적으로 필요한 다양한 요소를 배울 수 있는 시간이었다. 소프트웨어적인 논리성과 더불어 물리적 공간에 있는 다양한 요소들을 고려하고 이를 반영하여 코드를 정교하게 수정하는 과정이 자율주행을 구현하기 위해 중요한 요소임을 깨달을 수 있었다.수 없이 많은 테스트를 거치며 코드를 정교하게 바꾸어가고 현실적으로 불가능한 요소를 포기하는 과정속에서 목표에 점점 근접해나갔고 xycar안에 내장되어 있는 다양한 하드웨어를 활용하여 AD프로젝트에서 다양한 기능들을 구현할 수 있었다.