창업연계공학설계입문
AD Project

프로젝트 명	신호등 인식 Xycar
팀 명	5분반 1조
문서 제목	최종보고서

팀원	20152524	이윤재
	20165159	이준영
	20191653	이현승
	20191647	이종엽
	20191667	정지환

목차

국민대학교 소프트웨어융합대학	최종보고서	
	프로젝트 명	신호등 인식 Xycar
	팀 명	5분반 1조

1. 프로젝트 소개

- 주제 : 신호등 인식 Xycar

  도로를 주행하다 보면, 다양한 표지만 표식, 그림등을 만나게 된다. 이들은 모두 그 해당 도로에서 준수해야하는 규칙, 법 등을 명시해놓은 것이며 불순할 경우 처벌은 물론 각종 교통사고를 유발할 가능성을 높이게된다.
  이러한 상황에 적합한 Xycar를 만들기 위해서 센서를 다각도로 활용해 도로교통상황을 인식하고 적절한 행동을 취하는 차량을 제작하는 것으로 목표를 설정했다.
  추가적으로 고려한 부분은, 완전자율주행차량에서는 탑승자가 도로교통정보에 무관심해지는 현상이 발생할 수 있다는 점이다. 따라서 탑승자가 존재할 때의 상황을 고려해, 탑승자와 차량간의 원활한 상호작용을 위해서 차량 주행 정보를 안내하는 프로그램을 추가하기로 결정했다.

- 요구사항 명세서 (Software Requirement Specification, SRS)

  1. 기능적 요구사항
     1-1. 카메라의 영상에서 교통신호를 검출해 내도록 한다. 검출된 표지판에 대응하는 행동조건이 있다.
     1-2. IMU센서에서 roll, pitch, yaw를 읽어들인다.
     1-3. 차선인식 자율주행은 2차 주행평가에서 사용한 바를 그대로 이용한다.

  2. 사용자 인터페이스 요구사항
     2-1. xycar내에 별다른 GUI모듈이 없으므로, SSH로 접속한 Desktop의 terminal에 print되도록 구현한다.

  3. 비기능적 요구사항
     3-1. 이 소프트웨어의 구현에는 OpenCV3.7과 python3.5, Github를 이용한다.
     https://github.com/hyeonseung-lee/Xycar_ADproject
     ~~3-2. Github에 있는 표지판 인식 open source를 활용한다.~~
     ~~https://github.com/ghostbbbmt/Traffic-Sign-Detection~~
     3-3. 교통 법규 설정

: 2차평가에서 진행했던 순서대로 주행한다. 1구간에서는 기본 속도로 주행하고, 2구간은 어린이 보호구역, 3구간 끝에는 건널목으로 설정한다.

2. 수행 내용

2 - 1. 문제점

첫 번째로 진행해야하는 부분은 open source를 활용해서 표지판 인식을 원활히 진행하는 것이다. git hub에 게시되어있는 코드를 활용해서 개발을 진행했다. 하지만 다음과 같은 문제점이 발생했다.

- 높은 난이도

  구현되어있는 코드의 난이도 자체가 굉장히 높았다. 협업을 위해서는 5명 모두가 비슷한 이해 수준을 갖춰야한다. 하지만 그러기엔 시간이 너무나 부족했다. main class의 길이는 400줄 정도였고, impoort해 오는 class와 module들의 수도 굉장히 많았다. 타 과목의 AD 프로젝트와 기말고사 지필평가를 병행해야하는 입장에서 굉장히 부담스러운 수준의 코드였다.

- 떨어지는 정확도

  open source에 제시되어있는 코드를 실행환경만 맞춰서 실행해 보았다. 하지만 우리가 원했던 수준에 비해서 표지판이 인식되는 속도가 많이 떨어졌고, 그 정확도 또한 기대에 미치지 못했다.

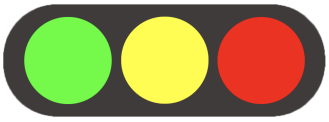
  이런 상황에서 우리는 정확한 표지판 인식을 위해서 tensorflow와 같은 machine learning이 필요함을 느꼈다. 이는 부족한 실력 탓에 공부가 필요했다. 하지만 이 또한 시간이 너무나 모자라서 다른 방법을 찾기로 결심했다.

2 - 2. 목표 재설정

2-1에서 언급한 바와 같이 open source의 활용과 실제 표지판 인식에 많은 어려움을 느꼈다. 그래서 우리는 별다른 모듈을 사용하지 않고, 수업시간에 배운 내용을 재구성해서 표지판을 인식할 수 있는 방법을 찾았다.

- 표지판의 단순화

  현존하는 표지판을 그대로 읽어들이는게 굉장히 어렵기 때문에, 단순한 색상으로 대체했다.



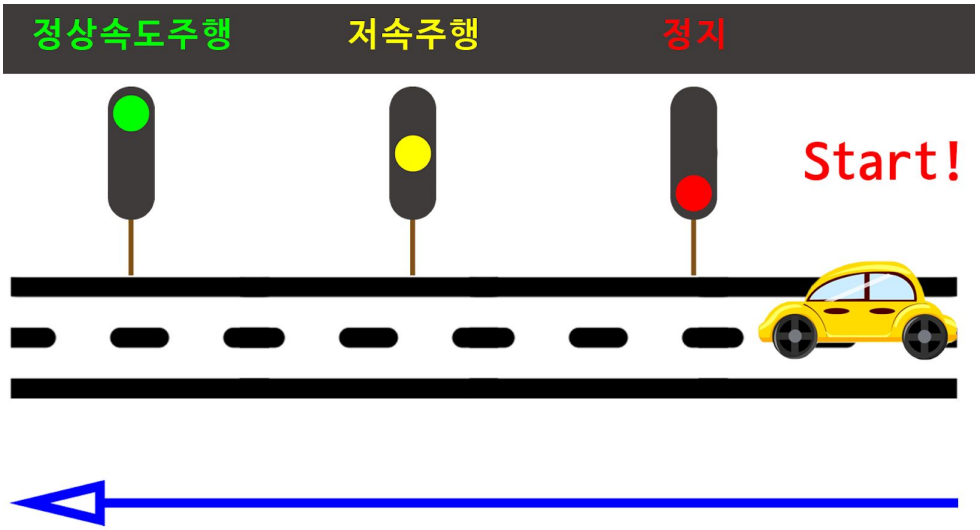
노란색 : 어린이 보호구역 (감속)

초록색 : 일반 구역 (정상 속력)

빨간색 : 건널목 등으로 인한 정지상황 (정지)

이렇게 목표를 재설정하고 다시 개발에 돌입했다.

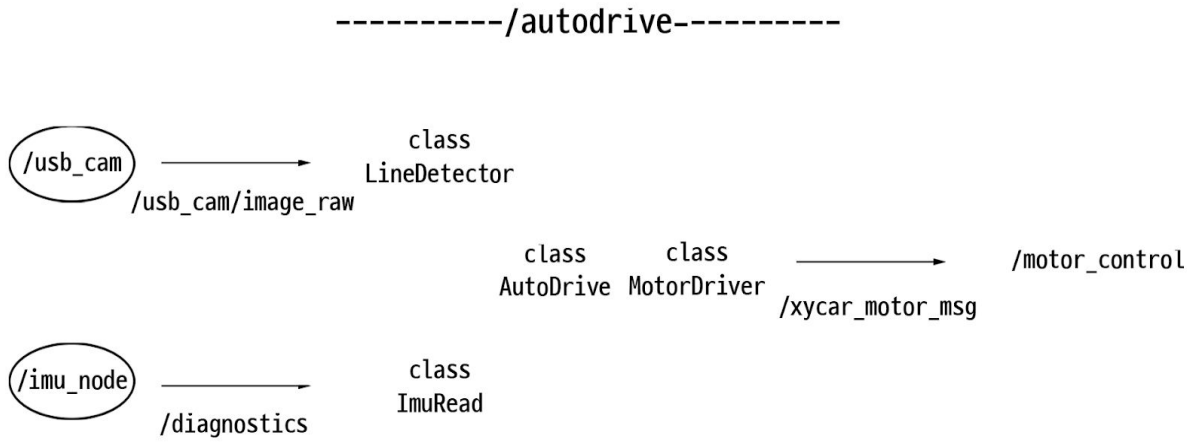
- 최종 결정된 교통법규



2 - 3. 소프트웨어 구조 설계

최종 설정한 목표와 함께 node와 topic사이의 관계를 설계했다.

- 소프트웨어 구조 설계서 (Architecture Design Specification, ADS)



모듈	클래스	역할
autodrive.py	AutoDrive	타 class에서 반환하는 데이터를 기반으로 조향각과 속력을 설정하여 MotorDriver로 전달함
		탑승자에게 전달하고자 하는 정보 (속력, 교통법규, imu 센서값)을 출력
linedetector.py	LineDetector	차선을 인식해서 차선 위치를 반환함. 동시에 표지판을 인식해서 표지판의 종류를 반환함
imuread.py	ImuRead	IMU 센서에서 topic을 받아오고, 이 topic에서 roll, pitch, yaw에 해당하는 값을 추출, 이를 반환함
motordriver.py	MotorDriver	autodrive에서 전달받은 조향각과 속력으로 차의 움직임을 제어함

위와 같은 방식으로 진행될 예정이다.

기존 차선인식 자율주행 알고리즘에서 크게 변화한 바는 없다. 주요 변화만 꼽자면 이렇다.

class LineDetector에는 색상을 인식하는 기능이 추가되었다. 또 ImuRead class에서는 /imu_node 에서 /diagnostics라는 topic을 가져와서 roll, pitch, yaw 값을 추출, AutoDrive class로 넘겨주게 된다.

이후 AutoDrive에서는 LineDetector에서 검출한 신호와 신호에 따른 반응을 하고, 그 신호의 의미를 print한다. 동시에 IMU센서의 값을 print하면서 주행하게 된다.

3. 결과

3 - 1. 완성된 코드

LineDetector.py

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

import rospy, time

from linedetector import LineDetector
from motordriver import MotorDriver
from imuread import ImuRead


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.imu = ImuRead('/diagnostics')
          self.information = {"speed": -1, "signal": "", "=>": "", "pitch": -1, "roll": -1, "yaw":
-1}

    def trace(self):
        line_l, line_r, red, yellow, green = self.line_detector.detect_lines()
        self.r, self.y, self.g = red, yellow, green
        self.line_detector.show_images(line_l, line_r)
                self.information["roll"],  self.information["pitch"],  self.information["yaw"]  =
self.imu.get_data()
        angle = self.steer(line_l, line_r)
        speed = self.accelerate(angle, red, yellow, green)
        self.information["speed"] = speed + 90
        self.driver.drive(angle + 90, speed + 90)

        print(speed + 90)
```

```
    print(r, p, y)

def steer(self, left, right):
    if left == -1:
        if 320 < right < 390:
            angle = -50
        else:
            angle = (550 - right) / (-3)
    elif right == -1:
        if 250 < left < 320:
            angle = 50
        else:
            angle = (left - 90) / 3
    else:
        angle = 0
    return angle

def accelerate(self, angle, red, yellow, green):

    # if red is detected, stop
    if red and (not yellow) and (not green):
        speed = 0
    # if yellow is detected, go low speed
    elif (not red) and yellow and (not green):
        speed = 20
    # if green is detected, go origin speed
    elif (not red) and (not yellow) and green:
        speed = 50

    # This is origin code
    else:  # (not red) and (not yellow) and (not green):
        if angle <= -40 or angle >= 40:
            speed = 30
        elif angle <= -25 or angle >= 25:
            speed = 40
        else:
            speed = 50

    print(red, yellow, green)
```

```
        return speed

    def printInformation(self):

        if self.r:
            self.information["signal"] = "red"
            self.information["=>"] = "stop!"
        if self.y:
            self.information["signal"] = "yellow"
            self.information["=>"] = "slow"
        if self.g:
            self.information["signal"] = "green"
            self.information["=>"] = "just go"

                print("speed : " + str(self.information["speed"]) + " ", "signal : " +
str(self.information["signal"]) + " ",
              "=> " + str(self.information["=>"]))
                    print("pitch : " + str(self.information["pitch"]) + " ", "roll : " +
str(self.information["roll"]) + " ",
              "yaw : " + str(self.information["yaw"]))
        print("")

        self.r, self.y, self.g = False, False, False
    def exit(self):
        print('finished')


if __name__ == '__main__':
    car = AutoDrive()
    time.sleep(3)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        car.trace()
        car.printInformation()
    rate.sleep()
    rospy.on_shutdown(car.exit)
```

imuread.py

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

import rospy
from diagnostic_msgs.msg import DiagnosticArray


class ImuRead:

    def __init__(self, topic):
        self.roll = -1
        self.pitch = -1
        self.yaw = -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.pitch), float(self.yaw)
```

autodrive.py

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

import rospy, time

from linedetector import LineDetector
from obstacledetector import ObstacleDetector
from motordriver import MotorDriver
from imuread import ImuRead


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.imu = ImuRead('/diagnostics')

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, y = self.imu.get_data()

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

    print(speed + 90)
    print(r, p, y)

def steer(self, left, right):
    if left == -1:
        if 320 < right < 390:
            angle = -50
        else:
            angle = (550 - right) / (-3)
    elif right == -1:
        if 250 < left < 320:
            angle = 50
        else:
            angle = (left - 90) / 3
    else:
        angle = 0
    return angle

def accelerate(self, angle, left, mid, right):
    print(mid)

    if mid < 40:
        speed = 0
```

```
            return speed

        if angle <= -40 or angle >= 40:
            speed = 30
        elif angle <= -25 or angle >= 25:
            speed = 40
        else:
            speed = 50

        return speed

    def exit(self):
        print('finished')

if __name__ == '__main__':
    car = AutoDrive()
    time.sleep(3)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        car.trace()
    rate.sleep()
    rospy.on_shutdown(car.exit)
```

motordrive.py

```
import rospy
from std_msgs.msg import Int32MultiArray

class MotorDriver:

    def __init__(self, topic):
        self.motor_pub = rospy.Publisher(topic,
                                         Int32MultiArray,
                                         queue_size=1)

    def drive(self, angle, speed):
        drive_info = [angle, speed]
        pub_info = Int32MultiArray(data=drive_info)
```

self.motor_pub.publish(pub_info)

3 - 2. 시연

https://youtu.be/aU4tlvHaWko

3 - 3. 후기

여전히 표지판 인식에 대한 아쉬움이 남았다. 조금의 시간적 여유가 있거나 팀원 개개인의 능력이 지금보다 조금만 더 높았더라면 더욱 질이 높은 프로그램을 구현할 수 있지 않았을까 하는 아쉬움이 남았다.