

# 창업연계공학설계입문

## AD 프로젝트 보고서

3분반 5조

백현식 성창업 신혜은 박준서 오탁봉

# 목차

1. 개요
2. 프로젝트 설계
3. 프로젝트 구현
4. 테스트
5. 피드백

# 개요

## 프로젝트 주제 : 컨트롤러로 자이카 운전 면허 따기

어디서나 쉽게 구할 수 있으며 타는 방법 또한 간단한 전동킥보드와 같은 이동수단도 면허증이 있어야 이용이 가능한 법률이 있는 것처럼 이 시대는 발전하는 기술에 따라 그에 맞는 안전 수칙과 법률 등이 중요시 되고 있다.

본 프로젝트의 컨셉은 “자이카는 소중한 물건이고 다루기도 쉽지 않으며, 자동차의 일종이므로 면허를 소지한 자격 있는 사람만이 제어를 해야 한다” 이다. 이를 중심으로, 본 프로젝트는 컨트롤러를 이용하여 운전면허 시험을 보는 ‘자이카 운전면허 시스템’을 구축함을 목적으로 한다.

# 프로젝트 설계

본 프로젝트에서 구현해야 할 것은 실제 운전면허 기능시험과 같으므로,

1. 코스 완주시간
2. 제한 속도
3. 차선 이탈 유무
4. 올바른 주차

등을 파이썬을 이용하여 ros 프로그래밍을 통해 구현할 것이다. 하지만 컨트롤러와 자이카가 통신하는 동시에 자이카의 모터노드를 같이 실행시키는 것은 불가능하므로(노드가 실행되지 않음), 모터노드가 필요한 2번을 제외하고 프로젝트를 진행한다.

# 프로젝트 설계 - 코스 완주 시간

실제 운전면허와 같이 제한 시간내에 완주를 하지 못할 시 **불합격** 처리한다.

메인 함수가 실행될 때의 시간을 저장하고, 코스 주행이 완료되면 시간을 계산하여 합격, 불합격을 정한다. 파이썬 내장 라이브러리인 time 모듈을 import해서 시간을 계산한다.

# 프로젝트 설계 - 차선 이탈 유무

차선을 이탈하면 점수를 감점한다.

창연공 수업 때 배운 차선인식 알고리즘을 적용하여 차선을 검출한다. 자이카에 장착 되어있는 카메라를 통해 받아온 영상에서 ROI(Region Of Interesting)를 설정하고, Grayscale변환, Gaussian blur처리, Canny변환을 거쳐 ROI 구역 내의 차선의 외곽선을 검출한다. 명도값을 기준으로 차선을 구분한 뒤 값을 할당한다. 좌측과 우측 차선이 모두 검출되지 않으면 차선을 이탈한 것으로 간주, 감점 처리한다. 단, 5초의 복귀시간을 제공한다.

# 프로젝트 설계 - 올바른 주차

올바른 주차(여유공간 충분)를 하면 주차성공. **실패 시 감점**

코스의 마지막 부분에 주차장을 설치하여 주차능력을 시험한다. 주차는 전면주차로 하며 자이카에게 신호를 보내면 전방, 좌측, 우측의 초음파 값을 받아와 거리를 측정한다. 초음파 값이 일정 미만일 시 주차능력 미달로 간주하여 감점 처리한다.

# 프로젝트 구현

A screenshot of a gedit text editor window. The title bar shows the file path as 'license\_proto.py (~/xycar/src/ad/src) - gedit'. The editor has a dark theme. The code inside the editor is as follows:

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

import rospy, time, cv2
from linedetector_proto import LineDetector
from obstacledetector_proto import ObstacleDetector
```

메인함수를 포함하는 license 파일을 실행 파일로 지정하고,  
차선 검출을 하는 linedetector 파일과  
초음파 감지를 하는 obstacledetector 파일을 import 해온다.



# 프로젝트 구현 - License(클래스)

```
class License_test:
    def __init__(self):
        rospy.init_node('xycar_driver')
        self.line_detector = LineDetector('/usb_cam/image_raw')
        self.obstacle_detector = ObstacleDetector('/ultrasonic')
        self.count = 0
        self.depart_start = -5
```

클래스 License\_test 를 만든다.

\_\_init\_\_ 함수는 파이썬 파일 노드의 선언과, 각 모듈에 토픽 경로를 할당하고 객체를 생성한다.

self.count는 차선 이탈의 횟수, self.depart\_start는 추후 복귀시간 if 문을 통과할 수 있도록 초기화를 해주었다.

# 프로젝트 구현 - License(클래스)

```
def line_test(self):
    line_l, line_r = self.line_detector.detect_lines()
    self.line_detector.show_images(line_l, line_r)
    if 5 < time.time() - self.depart_start:
        if line_l == -1 & line_r == -1:
            self.count += 1
            self.depart_start = time.time()

def parking_test(self):
    return self.obstacle_detector.get_distance()
```

차선 이탈의 횟수를 세는 메소드 - line\_test

line\_detector 객체로부터 좌측, 우측 차선을 받아오고 스크린 출력을 위해 값을 전달한다.

현재시각에서 차선을 이탈한 시각의 차가 5초 이하일 경우엔 차선 이탈 감지를 활성화하고, 그렇지 않을 경우 비활성화한다.

주차장에서의 초음파 값을 가져오는 메소드 - parking\_test

obstacle\_detector 객체로부터 전방, 좌측, 우측 총 세개의 값을 리턴한다.

# 프로젝트 구현 - License(클래스)

```
def totalScore(self, line, parking_l, parking_m, parking_r, time):  
    if time > 60:  
        return 0, "Fail"  
  
    parking = "Success"  
    if (parking_l > 3) and (parking_m > 3) and (parking_r > 3):  
        totalscore = 100  
    else:  
        totalscore = 80  
        parking = "Fail"  
  
    totalscore -= 5 * line  
  
    return totalscore, parking  
  
def exit(self):  
    print('finished')
```

운전면허시험 점수를 계산하고 점수와 주차성공 여부를 리턴하는 메소드 - totalScore

먼저 완주 시간을 측정하고 60초 이상이면 빵점과 "Fail"을 출력한다(주차를 성공했어도 타임아웃으로 불합격).

만약 시간 내에 완주했다면, 받아온 초음파 값을 비교해 주차성공여부를 결정한다. 실패 시 20점을 감점한다.

마지막으로 차선이탈 횟수인 line을 5와 곱해서 이탈 횟수 당 5점을 감점한다.

Rospy가 shutdown 됐을 때 "finished"를 출력하는 메소드 - exit

# 프로젝트 구현 - License(메인)

```
if __name__ == '__main__':
    startTime = time.time()
    test_car = License_test()
    time.sleep(3)
    rate = rospy.Rate(15)
    while cv2.waitKey(1) & 0xFF != 27: #escape key
        test_car.line_test()
        rate.sleep()
    while cv2.waitKey(1) & 0xFF != 32: #space bar key
        left, mid, right = test_car.parking_test()
        rate.sleep()

    cv2.destroyAllWindows()
    finishTime = time.time()
    totalTime = finishTime - startTime
    score.result = test_car.totalScore(test_car.count, left, mid, right, totalTime)
```

실행파일인 License 파일의 메인.

객체를 생성하고 두번의 while문을 거친다. 이때 각 while문은 15hz를 주기로 반복된다.

첫 번째 while 문 : Esc키를 누를 때 까지 line\_test 메소드를 반복호출하면서 차선 이탈 횟수를 센다.

두 번째 while 문 : 첫 번째 while문이 종료되면 Space bar를 누를 때 까지 좌측, 전방, 우측 초음파 값을 받아온다.  
Space bar를 누르면 제일 마지막으로 감지된 초음파 값으로 주차성공여부를 판단한다.

이후 윈도우를 닫고, 걸린 시간을 계산하여 점수를 계산하고 주차결과를 리턴하는 totalScore메소드에 인자를 전달한다.

# 프로젝트 구현 - License(메인)

```
#show license result
print("-" * 40)
print("Lane Departure : \t%d" %test_car.count)
print("Pakring_mid : \t\t%d" %mid)
print("Pakring_left : \t\t%d" %left)
print("Pakring_right : \t%d ... %s" %(right, result))
print("Runnung Time : \t\t%.2fsec" %totalTime)

if score > 60:
    print("\n## PASS ##")
else:
    print("\t## FAIL ##")
print("-" * 40)

rospy.on_shutdown(test_car.exit)
```

실행파일인 License 파일의 메인 - 면허시험 결과 출력

Lane Departure : 차선 이탈 횟수

Parking\_mid, left, right : 전방, 좌측, 우측 초음파 값(주차장 벽과의 거리)

Running Time : 완주시간

총점이 60점을 넘으면 성공, 넘지 못하면 실패. 결과를 출력하고 rospy를 shutdown한다.

# 프로젝트 구현 - linedetector

```
import rospy
import cv2, time
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.frame = np.empty(shape=[0])
        self.image_width = 640
        self.scan_width, self.scan_height = 270, 40
        self.lmid, self.rmid = self.scan_width, self.image_width - self.scan_width
        self.area_width, self.area_height = 10, 5
        self.roi_vertical_pos = 310
        self.row_begin = (self.scan_height - self.area_height) // 2
        self.row_end = self.row_begin + self.area_height
        self.pixel_cnt_threshold = 0.04 * self.area_width * self.area_height
        rospy.Subscriber(topic, Image, self.conv_image)

    def conv_image(self, data):
        self.frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
```

## 차선 검출 - \_\_init\_\_과 콜백함수

차선이탈 감지를 위한 차선 검출 모듈이다. \_\_init\_\_함수에서 ROI 설정과 ROI 내에서 차선 검출 범위를 모두 정한다. 카메라로부터 받아온 프레임 데이터와 토픽을 콜백함수에 전달하여 변수에 프레임을 저장한다.



# 프로젝트 구현 - linedetector

```
def detect_lines(self):
    self.left, self.right = -1, -1
    self.roi = self.frame[self.roi_vertical_pos:self.roi_vertical_pos + self.scan_height, :]
    self.hsv = cv2.cvtColor(self.roi, cv2.COLOR_BGR2HSV)
    self.avg_value = np.average(self.hsv[:, :, 2])
    self.value_threshold = self.avg_value * 0.8
    self.gray = cv2.cvtColor(self.roi, cv2.COLOR_BGR2GRAY)
    self.blur = cv2.GaussianBlur(self.gray, (5, 5), 0)
    self.edges = cv2.Canny(self.blur, 50, 150)
    self.edges = cv2.cvtColor(self.edges, cv2.COLOR_GRAY2BGR)
    lbound = np.array([0, 0, self.value_threshold], dtype=np.uint8)
    ubound = np.array([100, 255, 255], dtype=np.uint8)
    self.edges = cv2.cvtColor(self.edges, cv2.COLOR_BGR2HSV)
    self.bin = cv2.inRange(self.edges, lbound, ubound)
    self.view = cv2.cvtColor(self.bin, cv2.COLOR_GRAY2BGR)

    for l in range(self.area_width, self.lmid):
        area = self.bin[self.row_begin:self.row_end, l - self.area_width:l]
        if cv2.countNonZero(area) > self.pixel_cnt_threshold:
            self.left = l
            break

    for r in range(self.image_width - self.area_width, self.rmid, -1):
        area = self.bin[self.row_begin:self.row_end, r:r + self.area_width]
        if cv2.countNonZero(area) > self.pixel_cnt_threshold:
            self.right = r
            break

    return self.left, self.right
```

차선 검출 - 좌, 우측 차선 검출하는 메소드

Frame을 ROI영역으로 자르고, 흑백 변환과 블러 처리, Canny처리를 거치고 명도를 기준으로 이진화시킨다.

차선 검출 범위에서 이진화된 픽셀이 기존에 설정한 커트라인을 넘으면 차선으로 인식하고 변수에 값을 할당한다.

검출된 좌, 우측 차선을 반환한다.

# 프로젝트 구현 - linedetector

```
def show_images(self, left, right):  
    if left != -1:  
        lsquare = cv2.rectangle(self.view,  
                                (left - self.area_width, self.row_begin),  
                                (left, self.row_end),  
                                (0, 255, 0), 3)  
  
    if right != -1:  
        rsquare = cv2.rectangle(self.view,  
                                (right, self.row_begin),  
                                (right + self.area_width, self.row_end),  
                                (0, 255, 0), 3)  
  
    cv2.imshow("view", self.view)
```

차선 검출 - 좌, 우측 차선 검출 사각형 표시, frame을 스크린에 출력하는 메소드

검출된 좌, 우측 차선 값을 인자로 가져와 검출 영역을 사각형으로 표시하고 출력한다.

단, 검출되지 않았을 시 표시하지 않는다.



# 프로젝트 구현 - obstacledetector

```
import rospy
from std_msgs.msg import Int32MultiArray

class ObstacleDetector:

    def __init__(self, topic):
        self.left = -1
        self.mid = -1
        self.right = -1
        rospy.Subscriber(topic, Int32MultiArray, self.read_distance)

    def read_distance(self, data):
        self.left = data.data[7]
        self.mid = data.data[1]
        self.right = data.data[6]

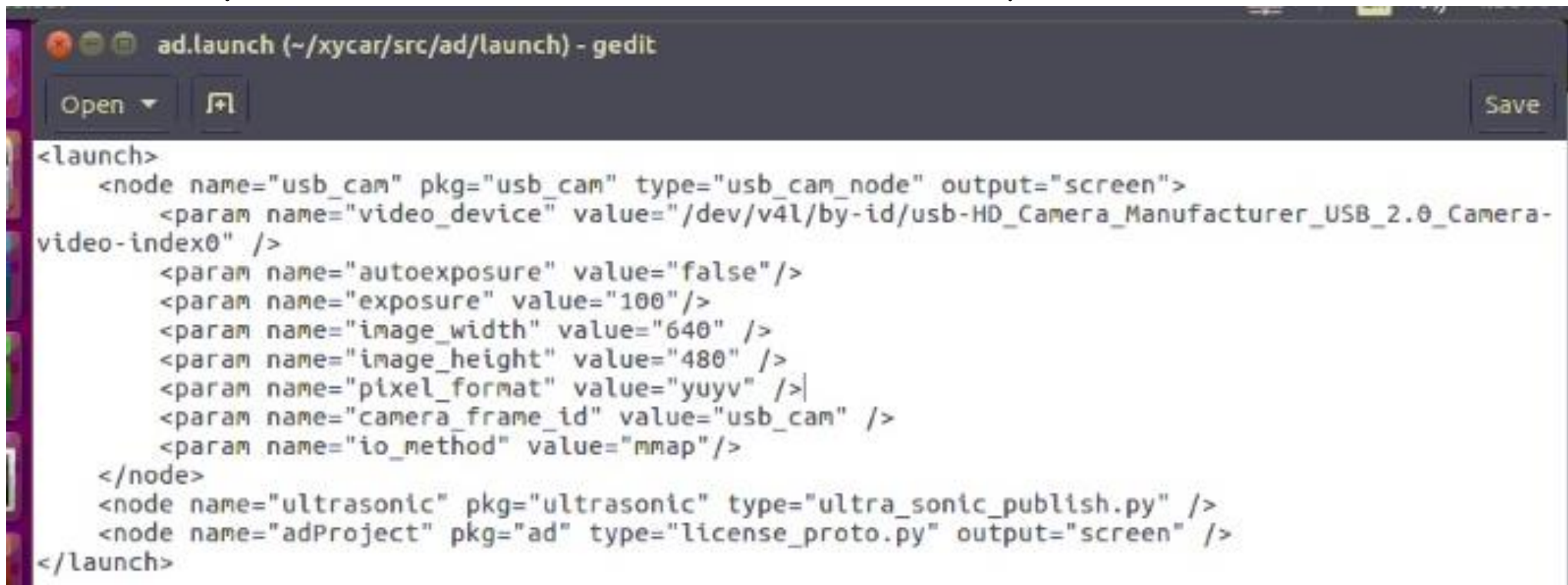
    def get_distance(self):
        return self.left, self.mid, self.right
```

자이카와 주차장 벽 사이의  
거리를 측정하는 모듈

각 변수를 -1로 초기화해  
주고, read\_distance 메소  
드를 통해 초음파 값을 받  
아오면 새로 할당한다.

Get\_distance 메소드로 초  
음파 값을 반환한다.

# 프로젝트 구현 - launch 파일

A screenshot of a gedit text editor window titled 'ad.launch (~/xycar/src/ad/launch) - gedit'. The editor contains XML code for a ROS launch file. The code defines three nodes: 'usb\_cam', 'ultrasonic', and 'adProject'. The 'usb\_cam' node is configured with various parameters like video\_device, autoexposure, exposure, image\_width, image\_height, pixel\_format, camera\_frame\_id, and io\_method. The 'ultrasonic' node is configured with pkg and type. The 'adProject' node is configured with pkg, type, and output. The entire code is enclosed in <launch> and </launch> tags.

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen">
    <param name="video_device" value="/dev/v4l/by-id/usb-HD_Camera_Manufacturer_USB_2.0_Camera-
video-index0" />
    <param name="autoexposure" value="false" />
    <param name="exposure" value="100" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap" />
  </node>
  <node name="ultrasonic" pkg="ultrasonic" type="ultra_sonic_publish.py" />
  <node name="adProject" pkg="ad" type="license_proto.py" output="screen" />
</launch>
```

## 런치파일

카메라와 초음파센서, license 파이썬 파일의 노드를 생성하고 사용하기 위한 launch 파일을 작성한다.

# 테스트(시연 영상 참고)



차선 이탈 2회



주차 완료

```
-----  
Lane Departure :      2  
Pakring_mid :      12  
Pakring_left :      36  
Pakring_right :      16 ... Fail  
Runnung Time :      83.93sec  
    ## FAIL ##  
-----  
finished
```

면허 시험 결과

# 피드백

차선이탈의 기준 개선 - 한쪽 바퀴만 조금 이탈할 경우 차선이탈로 간주하지 않을 수 있음. 차선을 조금만 이탈할 경우도 잘 감지할 수 있도록 차선이탈 인식 알고리즘을 고도화 할 필요가 있음.

터미널 인터페이스 개선 - 실시간 랩타임과 차선을 이탈한 순간 등 이벤트에 대한 출력문을 작성했으면 더 좋았을 것. 한 바퀴 주행이 끝나고 Space bar 를 누르면 주차테스트를 시작하겠다는 메시지를 출력하면 좋았을 것.