창업연계공학설계입문

2019

12 월 18 일

창연공 2 조

작성자: 이강민, 이소영, 이수아, 조민수

목차

1.AD 프로젝트 주제

2.목표 설정

3.실행환경

4.구현방식

5.활용방안

6.개선방안

1.AD 프로젝트 주제

표지판을 인식하는 자율주행 자동차를 구현한다. 과제로 진행했던 차선인식 자동차에 표지판에 적혀 있는 글자를 인식해서 각 표지판 마다 최선의 행동을 수행할 수 있도록 코드를 구현해 보려고 한다. 표지판은 나라에 따라 생김새도 주의하는 내용도 상이한 경우가 많다. 따라서 이번 프로젝트에서는 한국에 존재하는 표지판을 대상으로 표지판 인식 자율주행 자동차를 구현하려고 한다.

구현 방식은 크게 표지판에 적힌 속도를 인식하여 현재 속도가 표지판에 속도 보다 너무 느리다면 가속하고 너무 빠르다면 감속한다. 또한 감속 표지판의 경우 “어린이 보호구역”, “서행지역” 등 비슷한 범주에 묶인 표지판을 추가하여 다양한 경우에 표지판 인식을 수행할 수 있도록 한다. 마지막으로 정지(STOP) 표지판을 보게 되면 운행을 중지한다. 표지판 인식으로 인해 자율주행 자동차에 사고율을 줄이고 안정성을 높이려고 한다.

2.목표 설정

자율주행자동차는 처음 출발 속도를 40 로 설정하여 주행을 시작한다.
주행 중 인식하게 될 첫 번째 표지판은 어린이보호구역 표지판을 인식하고, 스쿨존(School Zone) 등과 같은 낮은 연령대가 많은 지역을 가정한다. 어린이 보호구역의 경우 제한속도가 30 이하이기 때문에 속도를 30 으로 감속한다. 어린이 보호구간동안 감속한 후 구간 종료 시 다시 기존속도 40 로 가속한다.

두번째 표지판은 서행 표지판으로 설정한다. 표지판으로 전방에 서행표지판을 설치한 뒤, 속도를 낮춰야 하는 밀집지역이나 사고다발지역의 경우를 가정해 본다. 이 경우 속도를 또다시 30 으로 감속한다. 그리고 서행구간이 끝나면 다시 기존속도 40 으로 돌아오도록 한다.

세번째 표지판은 최저속도 50 표지판이다. 고속도로와 같은 일정속도 이상이 필요한 경우의 지역을 가정하고, 자율주행자동차는 50 이상의 속도를 낸다.

그 다음 표지판은 최대속도 30 표지판이다. 최대속도가 30 이기 때문에 30 이하의 속도로 움직이도록 속도를 감속한다. 그 후 표지판 구간이 끝나면 정상속도 40 으로 복귀한다.
이후 운행을 진행하다가 설치된 STOP 표지판을 인식하면 최종 주행을 멈추게 한다.

3.실행환경

(1) 사용 표지판

도로 교통공단 교통안전표지일람표(2014.07.02 개정) 기준

324
어 린 이 보 호
(어린이보호구역안)

(1)

226
서 행

(2)

224
최고속도제한

(3)

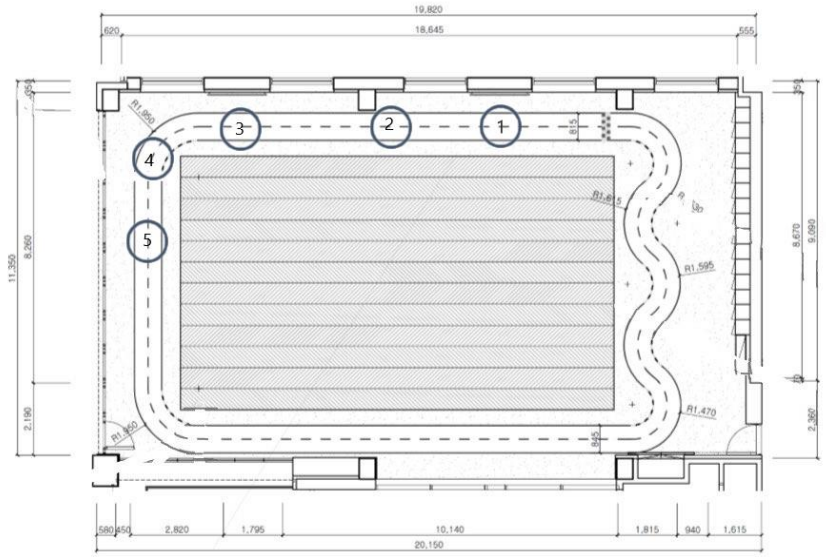
225
최저속도제한

(4)

227
일 시 정 지

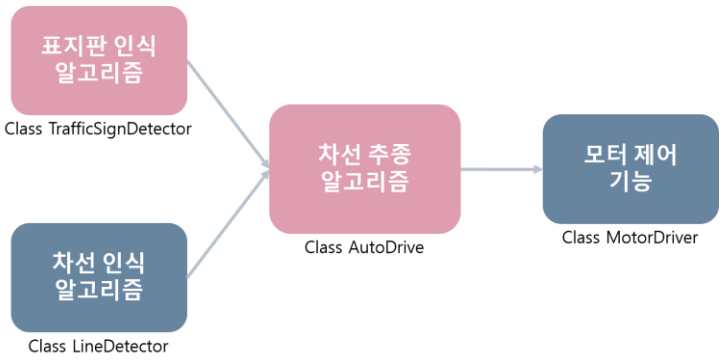
(5)

코스 구성



4.구현방식

(1) 프로그램 구조



기존의 LineDetector 클래스와 MotorDriver 클래스는 그대로 이용하고, 표지판 인식 알고리즘을 구현한 TrafficSignDetector 클래스를 새로 만들고, AutoDrive 클래스는 필요한 부분만 변경하여 사용하였다.

TrafficSignDetector 클래스의 경우 외부 모듈을 이용해야 하므로 셸 환경 변수 설정을 통해 개인 랩탑에서 구동하였고 다른 클래스들은 기존의 방법처럼 Xycar 에서 구동하였다.

(2) Class TrafficSignDetector

1. 역할

'signalDetect' 노드를 생성하고 '/usb_can/image_raw' 토픽을 구독하여 Xycar 의 카메라 영상을 받아 ROI 영역을 설정한다. 이 영역에서 pytesseract 모듈을 이용하여 표지판의 텍스트를 인식하고 인식된 표지판의 정보를 사용자 정의 토픽인 '/signal' 에 'std_msgs/String' 타입의 메시지로 발행한다.

2. 클래스 설계

Class TrafficSignDetector:

def __init__(self):

def conv_image(self, data):

- '/usb_cam/image_raw' 토픽에 대한 콜백 함수

def detect_signal(self):

- 원형 표지판의 인식 및 표지판 정보를 발행하는 함수

def adThresholding(self, im, val=31, val2=0):

- 그 외 표지판 인식을 위한 이미지 처리 (Adaptive Thresholding)

def tesseracting(self, name, mask, bitwising):

- 테서렉트로 인식하게 하는 메서드

def tesseract_start(self, filepath, im):

- 그 외 표지판을 인식하고 결과를 리턴하는 함수

def detecting(self, ans):

- 인식한 단어가 표지판 글자들 중에 일치하는 것이 있는지

def pub(self):

- 토픽을 발행하는 함수

def detectWord(self, org, word):

- 단어를 필터링하고 검출하는 함수

def __del__(self):

```
if __name__ == "__main__":
    rate = rospy.Rate(10)
    detector = TrafficSignDetector('/usb_cam/image_raw')
    while not rospy.is_shutdown():
        detector.pub()
        rate.sleep()
```

3. 원형 표지판 인식의 상세 구현

```
def detect_signal(self):
    circles = cv2.HoughCircles(self.gray, cv2.HOUGH_GRADIENT, 1, 20,
                               param1=50, param2=40, minRadius=30, maxRadius=45)

    circle_string_finded = False
```

cv2.HoughCircles 함수를 이용해 원을 검출한다. circle_string_finded 변수는 원형 표지판의 인식과 그 외의 표지판의 인식을 구별하기 위한 Boolean 타입 변수이다.

```
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
        offset_w = self.roi_horizontal_pos
        offset_h = self.roi_vertical_pos
        cnt = 23
        r = self.roi[i[1] - cnt:i[1] + cnt, i[0] - cnt:i[0] + cnt]
        cv2.circle(self.cam_img, (i[0] + offset_w, i[1] + offset_h), i[2], (255, 0, 0), 1)
        cv2.rectangle(self.cam_img, (i[0] - cnt + offset_w, i[1] - cnt + offset_h),
                      (i[0] + cnt + offset_w, i[1] + offset_h), (0, 255, 0), 1)
        try:
            self.pub_info = pytesseract.image_to_string(r)
            if "30" in self.pub_info:
                self.pub_info = "30"
                circle_string_finded = True

            elif "50" in self.pub_info:
                self.pub_info = "50"
                circle_string_finded = True
            else:
                self.pub_info = "not"

        except Exception as e:
            print(e)

if not circle_string_finded:
    self.pub_info = self.tesseract_start("python", self.roi)
```

원의 반지름과 중심 좌표를 이용하여 텍스트 검출 영역 r 을 지정한 후 이 영역에서 pytesseract 를 이용해 텍스트를 검출한다. 검출된 텍스트를 pub_info 라는 변수에 저장하고 try 문 아래의 if 문과 같은 방식으로 표지판의 종류를 인식한다. 원형 표지판이 인식되면 circle_string_finded 변수값을 True 로 바꿔 원형이 아닌 표지판 인식 알고리즘으로 넘어가지 않도록 한다. circle_string_finded 변수 값이 False 인 경우 원형이 아닌 표지판을 인식하는 알고리즘(self.tesseract_start())으로 넘어간다.

4. 그 외 표지판 인식의 상세 구현

우선 글자 검출에 가장 적합한 이미지 처리 기법을 알아내기 위해 다음과 같이 4 개의 메서드를 구현하고 테스트하였다. 앞에서부터 'HSV 색상 모델로 변경 후 이진화', '이미지 임계 처리: Threshold', '이미지 임계 처리: Adaptive Threshold', '경계선 검출'이다.

```
def hsvMasking(self, im, threshold=1):
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    avg_value = np.average(hsv[:, :, 2])
    value_threshold = avg_value * threshold
    lbound = np.array([0, 0, value_threshold], dtype=np.uint8)
    ubound = np.array([255, 255, 255], dtype=np.uint8)
    mask = cv2.inRange(hsv, lbound, ubound)
    return mask
```

```
def thresholding(self, im):
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    mask = cv2.threshold(im, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
    return mask

def adThresholding(self, im, val=31, val2=0):
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    mask = cv2.adaptiveThreshold(im, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, val,
val2)
    return mask

def canning(self, im, blur=5, val1=30, val2=70, dilate=3):
    im = cv2.GaussianBlur(im, (blur, blur), 0)
    mask = cv2.Canny(im, val1, val2)
    kernel = np.ones((dilate, dilate), np.uint8)
    mask = cv2.dilate(mask, kernel, iterations=1)
    return mask
```

경계선 검출의 경우 선이 너무 얇게 나와 글자 인식이 제대로 수행되지 않으므로, 이미지 형태 전환 기법 중 Dilation 을 이용하여 선을 굵게 만든다.

테스트 결과, 글자 인식이 제일 잘 되는 것은 Adaptive Thresholding 을 적용한 이미지였다. 하지만 그래도 여전히 Tesseract 가 글자를 부정확하게 인식하는 경우가 상당히 많았다. 다음은 pytesseract 로 Adaptive Thresholding 을 적용해 처리한 이미지에서 글자를 추출한 결과이다.

 # 정지 표지판

지{0)'』/2

8`0 흥

겉처뛰°줌`

징지`흠칭 0 흥~`I

/첨자흠칭 0 줌

칩자엎`햅 0 줌

참찌낚흠`0?칭/

/짐지₩뛰 0 흥

섬시쉘 0 줌

심시씽 810 줌

 # 어린이보호 표지판

`、쇄:、

혔어반허모호

`쉐`허턴미므흐

웬리미신

```
# 천천히 표지판
친미꼬꼬빰
친미뻐대|
친친미룸띠꺼
천천히튼따째
틱꼬다뻐
```

언뜻 보면 규칙이 하나도 없는, 잘못 인식된 결과로 보일 수 있다. 하지만 잘 살펴보면 '정지'를 인식할 때는 '징지', '첨자', '칩자' 등 '정지'와 비슷하게 생긴 문자열을 인식하며, '어린이보호' 또한 '어반허모호', '허턴미므흐', '천천히' 또한 '친친미' 등 모두 기준이 되는 문자열과 비슷하게 생긴 문자열을 인식한다.

Tesseract 가 추출한 결과를 정제하기 위해, 우선 '공백 및 특수문자 제거'를 진행하였다.

```
word = re.sub('[-=.#/?:$}{]`!%^&*()_+', '', word)
```

그리고 '인정 문자열'을 지정한다. 다음 코드는 stop(정지) 표지판인지 확인할 때는 '징, 진, 점, 잠...' 이어도 '정'으로 인정하고, 다른 것들도 마찬가지로 수행하기 위해 설정한 변수이다.

```
self.finds = []

stop = ("stop", [
    ['정', '징', '진', '점', '잠', '짐', '청', '칭', '친', '쳠', '참', '침', '성', '섬', '심'],
    ['지', '자', '저', '치', '차', '처', '시']
])
self.finds.append(stop)

slow = ("slow", [
    ['천', '칭', '전', '친', '진', '징'],
    ['쳔', '칭', '전', '친', '진', '징'],
    ['히', '이', '어', '허', '미', '머']
])
self.finds.append(slow)

child = ("child", [
    ['어', '허', '이', '히'],
    ['린', '턴', '반', '런'],
    ['이', '미', '허'],
    ['보', '브', '부', '모', '므', '무'],
    ['호', '흐', '후'],
])
self.finds.append(child)
```

판별은 다음 코드로 수행된다.

```
def detecting(self, ans):
    word = ans[0]
    word = re.sub('[-=.#/?:$}{]`!%^&*()_+', '', word)
    if word != '':
        print(word)

    for fn in self.finds:
        if self.detectWord(fn[1], word):
            print("========================================ITS", fn[0])
            return fn[0]

    return None

def detectWord(self, org, word):
    if len(word) < len(org):
        return False

    score = 0

    for i in range(len(org)):
        if word[i] in org[i]:
            score += 1
    if score >= len(org) // 2 + 1:
        return True

    if len(word) > len(org):
        p = len(word) - len(org)
        for i in range(len(org)):
            if word[i + p] in org[i]:
                score += 1
        if score >= len(org) // 2 + 1:
            return True
        else:
            return False
    else:
        return False
```

테서렉트로 인식한 글자를 detecting 메서드의 인자로 넣으면 표지판 인식 결과가 리턴된다. detecting 메서드는 다음 코드의 tesseract 메서드에서 호출한다.

```
def tesseracting(self, name, mask, bitwising):
    if mask is None:
        return ''

    cv2.imshow(name, mask)

    st = pytesseract.image_to_string(mask, lang='kor')

    if bitwising:
        mask_bitwise = np.full(shape=(mask.shape[0], mask.shape[1]), fill_value=255,
                               dtype=np.uint8)
        mask_bited = np.bitwise_xor(mask, mask_bitwise)
        st += "|" + pytesseract.image_to_string(mask_bited, lang='kor')

    st = st.replace(' ', '')
    st = st.replace('\n', '')
    return st

def tesseract_start(self, filepath, im):
    ans = []
    ans.append(self.tesseracting(filepath + "1",
                                 self.adThresholding(im, 101, 10),
```

```
                                False))
    return self.detecting(ans)
```

(3) Class AutoDrive

1. 역할

기존 자율주행 기능에 더해, /signal 토픽을 구독하여 표지판 인식 결과를 받아오고 그에 따라 속도를 조절하는 기능을 행한다.

```
class AutoDrive:

    def __init__(self):
        rospy.init_node('autodrive')
        rospy.Subscriber('/signal', String, self.detect_signal)
        (중략)
        self.end_time = -1
        self.signal_time = 3
        self.under_signal = "not"


    (메서드 중략)
    def detect_signal(self, data):
        if time.time() < self.end_time:
            return

        signal = data

        if signal != "not":
            self.end_time = time.time() + self.signal_time
            self.under_signal = signal
        else:
            self.under_signal = signal

    def accelerate(self):
        if self.under_signal != "not":
            if self.under_signal == "50":
                return 50
            else:
                return 30
        else:
            return 40
```

5.활용 방안

앞서 진행한 표지판 인식 자율주행 자동차를 어디에 활용할 것인지? 실생활에 어떻게 접목시키고 어떤 긍정적인 효과를 낼 수 있는지에 대해 말해보자. 2018 년판 OECD 회원국 교통사고 비교율을 보면 우리나라의 경우 OECD 평균 교통사고율 인 213.8 건 보다 약 2 배 높은 431.1 건이 발생했는데, 자료가 확인된 OECD 32 개국 중 우리나라는 4 번째로 교통사고율이 높은 것으로 확인되었다. 우리나라의 경우 교통사고율이 점차 나아지는 추세를 보이고 있지만, 그럼에도 불구하고 여전히 다른 나라에 비해 높은 교통사고 통계량을 가지고 있다. 또한 교통사고 사망률 또한 8.4 명으로 OECD 회원국 평균인 5.5 명에 비해서 약 1.5 배 높은 사망률을 가지고 있으며, 35 개국 중 4 번째로 높은 사망률을 보유하고 있다. 따라서 사고율과 사망률을 줄이기 위해 교통상황에 맞는 안전운행이 필요하다. 이러한 경우 우리가 진행한 표지판 인식 차량주행 프로그램이 도움을 줄 수 있다. 운전을 하다 보면 미쳐 표지판을 보지 못하고 속도를 줄이지 못해 과속 벌금을 내거나 사고로 이어지는 경우가 있다. 이 때 자동차가 표지판을 먼저 인식해서 속도를 줄여 준다면 운전자의 부담감도 줄여 줄 수 있고, 자동차 사고율 또한 줄일 수 있을 것이다.

6.개선 방안

1. 머신러닝을 활용한 다양한 표지판 인식

   현재 pyteseract 를 활용하여 차량이 인식할 수 있는 표지판이 글자나 숫자에 제한된다는 한계점이 있다. 이를 파이썬 모듈인 욜로(yolo)를 통해 그림이나 기호 등을 인식할 수 있도록 하여 다양한 표지판을 인식할 수 있도록 개선한다.

2. 예외처리

   우측에 보이는 횡단보도의 신호등이 파란신호여도 횡단보도를 건너는 사람이 없으면 우회전해도 되는 도로 교통법이 있음. 이에 따라 보행자가 보는 건너편 신호등의 색을 인식하고 보행자(장애물)이 없으면 우회전 주행을 하도록 한다.

3. 인식률 향상

   자율주행 스튜디오 라는 제한된 조건속에서 프로젝트를 진행했기 때문에, 실제 도로를 주행하게 된다면 날씨 또는 일조량에 의해서 표지판 인식률이 떨어질 가능성이 매우 높다. 따라서 예외상황에서 코드처리를 통해 표지판 인식률을 높일 방법을 찾아 개선하도록 한다.