

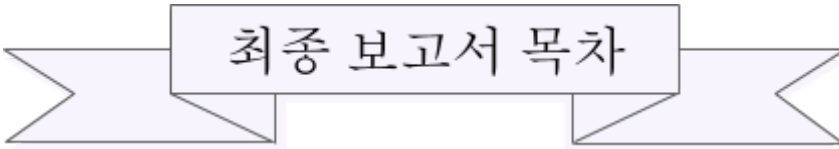
## 2019학년도 창업연계공학설계입문 조별 과제 최종 보고서

조번호					
조장/조원	성명	학과	학번	학년	비고
조 장	장정안	소프트웨어학부	20191658	1	
조 원	정승우	소프트웨어학부	20191664	1	
	정민지	소프트웨어학부	20191662	1	
	정수환	소프트웨어학부	20191663	1	
	장승훈	소프트웨어학부	20191656	1	

해당 조는 국민대학교 소프트웨어학부 창업연계공학설계입문의 규정을 준수하여 성실하게 과제에 참여하였으며, 사실을 토대로 조별 최종 연구보고서를 작성하여 최종보고서를 제출합니다.

2019년 월 일

국민대학교 소프트웨어학부 창업연계공학설계입문



## 최종 보고서 목차

### I. 과제 수행 결과 보고서

1. 서론
  2. 연구의 목적, 내용, 방법
  3. 연구의 결과
  4. 결론
- 참고문헌

### II. 주차별 활동 보고서

(기존 제출한 주차별 보고서 통합)

### III. 프로젝트 수행 후기

2019학년도 2학기

## I. 과제 수행 결과 보고서

2019학년도 창업연계공학설계입문연구 보고서

# 창업연계공학설계입문 AD Project

국민대학교 소프트웨어융합대학  
소프트웨어학부

조번호 2

2019 년 12 월 18 일

## 1. 서론

자율주행 자동차는 ‘자동차 스스로 주변 환경을 인식, 위험을 판단해 운전자 주행조작을 최소화하고, 목적지까지 주행경로를 스스로 계획해 안전주행이 가능한 자동차’(이재관 2015; 국토교통부 2015). 자율주행자동차에는 국제자동차기술자협회가 2016년에 분류한 총 6단계가 있다. 레벨0부터 레벨2까지는 운전자의 개입이 필요하지만 레벨3부터는 기본적으로 시스템이 전체 주행을 수행한다. 현재 자율주행자동차의 레벨은 3이다. 레벨 3는 차량 제어와 주행환경을 동시에 인식하지만, 비상 상황 시 운전 제어권을 운전자에게 준다. 레벨 4는 시스템이 전체 주행을 수행하는 점이 레벨 3와 동일하나 위험 상황 발생 시에도 안전하게 대응해야 한다. 레벨 5는 레벨4에 비해 제약이 없다.

자율주행자동차가 상용화가 될 시기에 도로교통체계에 많은 변화가 예상되고 있다. KPMG의 조사결과에 따르면 자율주행자동차가 상용화하면서 2040년까지 자동차 사고율이 약 80% 정도 감소할 것이라고 전망한다. 많은 대중교통수단이 필요 없어지고, 도로용량의 증가로 많은 도로나 주차장도 필요 없어진다. 제한속도, 어린이 보호구역 등의 주의 규제 정보에 따라 자동으로 운행하는 자율주행자동차로 인해 사람에게 맞추어진 도로의 표지판들이 없어지고 기계가 인식 할 수 있는 방식으로 변경이 될 것이다.

이렇듯 자율주행의 전망이 높아지고 있는 가운데 우리는 이런 자율주행자동차가 가져올 변화가 실질적으로 가능한지 확인하기위해 자이카에 필요한 코드들을 집어넣어 실제로 일어나고 있는 교통 상황들을 한번 적용해 보도록 하려고 한다. 비록 실제 자율주행자동차보다 필요한 부품들이 적고 비교적 간단한 방법으로 실행하는 거지만 이렇게 함으로써 자율주행 하는데에 최소한으로 알고 있어야할 기술들을 배워나가기에 매우 좋은 기회이다.

## 2. 2장 연구의 목적, 내용, 방법

### 2.1 신호등 검출

#### 2.1.1 HoughCircles를 이용한 원 검출

show\_imges 함수에서 이미지에 원이 있는 지 확인한다. 원이 검출되면 이미지에 원을 그린다. 원은 Hough Circle을 사용하여 검출한다.

원본 이미지에서 일정 부분을 roi로 설정하고, 이미지를 그레이스케일 이미지로 변환한다. OpenCV에서 제공하는 HoughCircles함수를 이용하여 원을 검출한다. 함수의 매개변수의 값을 조정하여 원 검출의 정밀도를 높인다.

`Cv2.HoughCircles(image, method, dp, minDist, parameter1, parameter2, minRadius, maxRadius)`

-dp : 이미지 해상도에 대한 accumulator 해상도의 비율의 역수이다.

- minDist는 검출된 원 사이의 최소거리이다. 이 값을 높이면 검출되지 못한 원들이 발생할 수 있고, 값을 낮추면 그 값보다 작은 위치에 있는 원들을 검출할 수 없게 된다.

- parameter1 : HOUGH\_GRADIENT의 경우 canny edge detector의 높은 threshold값이다. 낮은 thresh값은 0.5배해서 사용하게 된다.

- parameter2 : HOUGH\_GRADIENT의 경우 accumulator threshoding값이다. 이 값이 너무 낮으면 거짓 원이 검출된다. 가장 큰 accumulator값의 원을 가장 먼저 리턴한다.

- minRadius : 검출하고자 하는 원의 최소 반지름의 값이다. default값으로 0으로 지정할 수 있다.

- maxRadius : 검출하고자 하는 원의 최대 반지름의 값이다. default값으로 0으로 지정할 수 있다.

원이 검출되지 않으면 pass하고 원이 검출된다면 center값과 radius값을 추출해서 이미지에 원을 그린다. 이 이미지를 통해 원이 검출되었는지 확인할 수 있다. point라는 변수에 검출한 center값을 이용해 원본 이미지에서 검출한 원의 중심 좌표의 픽셀 값을 point에 담고 리턴한다. point에는 원의 중심의 픽셀의 BGR값이 담긴 리스트가 반환된다.

```
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 20,
                           param1=150, param2=70, minRadius=50, maxRadius=0)
point = []
if circles is None:
    pass
else:
    circles = np.uint(np.around(circles))
    for c in circles[0, :]:
        center = (c[0], c[1])
        radius = c[2]

    self.cam_img = cv2.circle(self.cam_img, center, radius, (255, 0, 0), 2)
```

```
print("point", self.cam_img[center[1]][center[0]])
point = self.cam_img[center[1]][center[0]]

cv2.imshow("check", frame)
cv2.imshow("des30", self.up_roi)
cv2.imshow("circle", self.cam_img)
cv2.waitKey(5)
return point
```

### 2.1.2 속도 제어

trace함수에서 리턴한 point값을 받아서 accelerate함수의 파라미터에 point를 추가하고 값을 넣는다. if문으로 point리스트의 3번째 요소, BGR에서 R값이 180보다 큰지 확인해서 만일 들어온 값이 180보다 크다면, red로 인식하고 속도를 90으로 지정한다. point리스트의 2번째 요소, G값이 150보다 큰지 확인하고 만일 크다면 green으로 인식하고 속도를 105로 지정한다. 여기서 표지판을 인식해서 멈추면 안되므로 맨 처음에 G와 R값이 155이상일 때는 pass한다.

```
def accelerate(self, angle, left, mid, right, point):
    if len(point) == 0:
        pass
    else:
        if point[1] >= 155 and point[2] >= 155:
            pass
        elif 180 <= point[2]:
            print("red")
            self.speed2 = 0
            self.speed = 0
        elif 150 <= point[1]:
            print("green")
            self.speed2 = 15

    return self.speed2 + self.speed
```

## 2.2 속도 표지판 검출

속도 표지판을 검출하기 위해서 이미지의 특징점을 검출하는 법을 알아낸다. xycar의 카메라로 들어오는 이미지와 저장해놓은 속도 표지판 이미지의 특징점을 서로 매치시키는 방법을 알아낸다.

### 2.2.1 ORB와 BF(Brute Force)매치를 이용한 이미지의 특징점 매치

먼저 xycar에 인식시킬 속도 표지판 이미지를 준비한다. 그 후 이미지의 특징점을 찾기 위한 ORB를 사용한다. (ORB는 특허권이 있어 자유롭게 사용 할 수 없는 SURF 와 SIFF를 대체 할 수 있도록 OpenCV Labs가 개발한 이미지 특성 검출 알고리즘이다.) 이 ORB를 사용하면 이미지의 키포인트와 디스크립터를 검출 해 낼 수 있는데 이를 검출 해 낸 후 두 이미지의 디스크립터를 BF매치를 이용하여 매치시킨다. 매치시킨 데이터를 파이썬파일의 리스트형태로 저장하는데 저장하는 기준을 매치시킨 데이터의 distance값을 기준으로 저장한다. 이 distance값이 작을수록 두 이미지의 특징점이 많이 일치하는것을 의미한다. 따라서 일정이하의 값만을 저장하고 이 리스트의 길이가 일정이상이면 두 이미지는 비슷하다고 보고 이미지 매치의 결과를 True로 리턴한다.

```
self.orb = cv2.ORB_create()
self.bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

self.img30 = cv2.imread('/home/nvidia/xyca/src/auto_drive/src/30km.jpg', cv2.IMREAD_GRAYSCALE)
self.kp30, self.des30 = self.orb.detectAndCompute(self.img30, None)

self.img40 = cv2.imread('/home/nvidia/xyca/src/auto_drive/src/40km.jpg', cv2.IMREAD_GRAYSCALE)
self.kp40, self.des40 = self.orb.detectAndCompute(self.img40, None)

self.img50 = cv2.imread('/home/nvidia/xyca/src/auto_drive/src/50km.jpg', cv2.IMREAD_GRAYSCALE)
self.kp50, self.des50 = self.orb.detectAndCompute(self.img50, None)

def img_match(self):
    kp, des = self.orb.detectAndCompute(self.up_roi, None)
    if des is None:
        print("des is None")
        return 0, 0, 0
    elif self.des30 is None:
        print("des 30 is None")
        return 0, 0, 0
    matches30 = self.bf.match(des, self.des30)
    matches30 = sorted(matches30, key=lambda x: x.distance)
    matches40 = self.bf.match(des, self.des40)
    matches40 = sorted(matches40, key=lambda x: x.distance)
    matches50 = self.bf.match(des, self.des50)
    matches50 = sorted(matches50, key=lambda x: x.distance)

    dist30 = [m.distance for m in matches30 if m.distance < 38]
    dist40 = [m.distance for m in matches40 if m.distance < 38]
    dist50 = [m.distance for m in matches50 if m.distance < 38]

    return len(dist30), len(dist40), len(dist50)
```

### 2.3.1 주차 표지판 인식

주차 표지판 인식 또한 이미지 매칭의 orb 방식을 사용했다. OpenCV에서는 이런 특징점 매칭을 Brute-Force 매칭이라고 한다. 이는 하나의 이미지에서 발견한 특징점을 다른 또 하나의 이미지의 모든

특징점과 비교해 가장 유사한 것이 동일한 특징점이라고 판단한다. xycar의 카메라를 통해 받아오는 실시간 영상을 한 이미지에 저장을 해서 매 순간마다 그 이미지에 저장된 프레임과 주차 표지판과 서로 비교해서 매칭되는 점들을 나타내준다. 서로 가까울수록 매칭되는 점들이 많아진다. 만약 매칭되는 점들의 수들이 일정 수 이상이 된다면 매칭이 되었다고 보고 True를 반환한다.

```
def parkingMatch(self):
    self.img1 = cv2.imread('/home/nvidia/xycar/src/auto_drive/src/f.jpg',self.cam_img)
    self.img2 = cv2.imread('/home/nvidia/xycar/src/auto_drive/src/f.jpg',
cv2.IMREAD_GRAYSCALE)
    self.kp2, self.des2 = self.orb.detectAndCompute(self.img2, None)
    self.imgTrainColor =
cv2.imread('/home/nvidia/xycar/src/auto_drive/src/parking.jpg',cv2.IMREAD_GRAYSCALE)
    self.kp1, self.des1 = self.orb.detectAndCompute(self.imgTrainColor, None)
    self.matches = self.bf.match(self.des1, self.des2)
    self.matches = sorted(self.matches, key = lambda x:x.distance)
    self.dist = [m.distance for m in self.matches if m.distance < 50]
    print(self.dist)
    if len(self.dist) >= 12:
        self.result = True
    return self.result
```

#### 2.4.1 주차하기

자율주행차의 주차는 레이더나 라이다를 사용하여 장소를 인식한 후 주차할 장소를 찾거나 발렛파킹시스템을 만들어서 주차 공간의 정보를 입력받아 빈 공간에 주차를 하는 것이다. 하지만 소프트웨어학부에서 제공하는 자율주행차인 XYCAR-B2에는 레이더나 라이다 같은 장치가 없다. 대신 초음파와 카메라가 장착되어 있다. 저번 도로 주행 과제를 하면서 같은 코드로 차를 운행시켰는데도 매번 다른 결과가 나오는 것을 보고 카메라에 인식되는 차선은 오류가 날 확률이 높다고 생각했다. 때문에 카메라 보다는 초음파 이용하는 것을 택했다. self.parkingMatch()에서는 주차 표지판을 인식하는 함수이다. return 값은 boolean이다. 주차 표지판이 인식되면 True를 return한다. 만약 boolean의 값이 True라면 self.parking()을 실행한다. 제대로 된 주차 코드는 parking()이다. self.obstacle\_detector.for\_parking\_distance()는 obstacle에서 filter를 거친 후의 장애물과의 거리이다. 장애물과의 거리를 이용하여 주차 공간을 찾고 왼쪽으로 직진한다. 후에 후진과 직진을 반복하여 도로와 차와의 각도를 수직으로 만든 후에 주차공간으로 후진하여 들어간다. 처음에는 직진하면서 parking\_6(6번 초음파센서)를 이용하여 오른쪽에 있는 주차공간을 인식한다. 후에 parking\_0(0번 초음파센서)를 이용하여 벽에 부딪히기 전까지 왼쪽으로 직진한다. 다음 뒤로 후진할 때는 parking\_3(3번 초음파센서) 그 후에 직진과 후진을 반복할 때는 다시 0번 초음파센서를 사용한다. 마지막 후진으로 주차공간을 진입할 때는 parking\_4(4번 초음파센서)를 이용하여 뒤쪽 벽에 부딪히지 않도록 한다.

```
def trace(self):
# if self.line_detector.parkingMatch() == True:
    self.parking()

def parking(self):
# parking_0 , parking_3, parking_4, parking_6 = self.obstacle_detector.for_parking_distance()
    while parking_6 <= 50:
```



```
for go1 in range(5):
    drive(85,110)
    if parking_6 >= 50:
        time.sleep(1)
        break
    time.sleep(0.1)

for stop1 in range(2):
    drive(90, 90)
    time.sleep(0.1)
    drive(65,110)
    time.sleep(0.1)

for left1 in range(50):
    drive(65,110)
    if data_0 <= 5:
        time.sleep(1)
        break
    time.sleep(0.1)

for stop2 in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(180,70)
    time.sleep(0.1)

for back1 in range(30):
    drive(180,70)
    if data_3 <= 5:
        time.sleep(1)
        break
    time.sleep(0.1)

for stop3 in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(70,110)
    time.sleep(0.1)
for go2 in range(20):
    drive(70,110)
    if data_0 <= 5:
        time.sleep(1)
        break
    time.sleep(0.1)

for stop4 in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(140,70)
    time.sleep(0.1)
```

```
for back2 in range(5):
    drive(140,70)
    time.sleep(0.1)

for stop5 in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(90,110)
    time.sleep(0.1)

for go3 in range(5):
    drive(90,110)
    time.sleep(0.1)

for stop6 in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(90,70)
    time.sleep(0.1)

for back3 in range(50):
    drive(90,70)
    if data_4 <= 5:
        time.sleep(1)
        break
    time.sleep(0.1)

for finish in range(2):
    drive(90,90)
    time.sleep(0.1)
    drive(90,90)
    time.sleep(0.1)
rate.sleep()
rospy.on_shutdown(exit_node)
```

### 3. 3장 연구의 결과

#### 3.1 Hough Circles를 이용한 원 검출 결과

받아들인 이미지를 Hough Circles를 이용해 원을 검출하는 과정에서 원이 검출되지 않는 케이스를 따로 고려하지 않았을 때 에러가 발생한다. 따라서 원이 검출되지 않은 경우 pass하는 코드를 작성해야 한다. 원을

정확하게 검출하기 위해 검출된 결과를 확인해가며 hough circles의 매개변수의 값을 조정한 결과 정확하게 원을 검출할 수 있었다. 마찬가지로 검출된 point의 값들을 확인해가며 색의 threshold값을 조정했다.

### 3.2 ORB와 BF매치를 이용한 이미지 처리의 결과

최종적으로 이미지의 특성을 찾아 이미지를 매치시키는데에는 성공했다. 하지만 xycar에 탑재된 컴퓨터의 연산속도, 카메라의 프레임 등의 하드웨어적 요건 때문인지 정지되어있는 표지판을 xycar가 주행하면서 검출해내는것은 실패했다. 코드에 프린트 등을 넣어 확인해 보니 표지판이 화면의 약 1/4을 덮을 수준정도가 되어야 인식을 하는것을 확인했다. 이는 xycar에 저장된 표지판의 이미지 자체가 너무 크기 때문이라고 생각된다. 자신이 검출할 이미지의 크기를 조절한다면 정지되어있는 물체를 움직이면서 검출 해 낼수도 있다고 생각한다.

### 3.3 주차표지판 인식과 주차하기 결과

결과적으로는 주차 표지판 인식과 주차하는 것에서 모두 성공했다. 하지만 라이다나 레이더를 이용하여 더욱 정확한 주차공간 인식과 주차를 구현하였더라면 더 좋았을 것이다. 특히 카메라의 성능이나 도로의 변색, 반사광 등 때문에 카메라 또한 이용하지 못한 점이 아쉽다. 이번 프로젝트와 시험 기간이 겹쳐 프로젝트에만 몰두하지 못한 것도 아쉽다. 시간이 조금만 더 있었다라면 앞으로 움직이면서 차와 도로 사이의 각도를 수직으로 맞출 때의 코드를 수동적이 아니라 초음파 센서를 더 이용하여 능동적으로 움직일 수 있게 했을 것이다. 그래도 제안발표에서 어렵다고 생각했던 주차하기를 어떤 방법으로든 성공하여 긍정적으로 생각한다.

## 4. 결 론

자율주행자동차가 상용화되는 날이 이제 그다지 멀지 않았다. 자율주행자동차가 상용화되면 사고율도 줄이게 되고 차 공용화의 시대가 온다. 자율주행자동차는 주어지는 정보를 바탕으로 스스로 도로의 상황을 파악해 자기 스스로 판단을 내린다. 운전자에게는 그 동안 다른 일을 할 수 있는 시간을 제공하고 사고율도 현재에 비해 많이 감소할 것이다. 우리는 이런 자율주행자동차의 기능을 작게나마 구현을 시도해 보고자 한다.

총 몇 주간의 시간을 써서 자이카를 우리 뜻대로 움직이게 하려고 시도해보았다. 시도하는 중간에 많은 오류들이 나타났고 때론 답답할때도 있었고 문제가 무엇인지 모르기 때문에 막막하기도 했었다. 자이카의 카메라와 opencv를 사용해 특정 표지판과 신호등을 인식하게 하고 그 인식 결과에 따라 각각 다른 행동을 할 수 있도록 해보았다. opencv의 orb과 BF매치를 사용해서 카메라에서부터 받아오는 이미지와 우리가 지정한 특정 표지판의 이미지의 특징점을 찾아 몇 개의 특징점이 인식이 되면 자이카 앞에 그 표지판이 있다고 보고 그에 따른 행동을 하도록 하거나 Hough Circles를 이용해 받아들인 이미지에 원을 검출해서 원의 색깔에 따라 멈추거나 출발하는 기능을 성공적으로 실현시켰다.

자율주행자동차가 상용화 되었다고 해서 꼭 장점만 있는 것이 아니다. 차가 자동화가 됨에 따라 우리 주위의 몇 몇 직업이 사라질 것이고 현재까지 계속적으로 의논이 되고있는 윤리적 문제들도 아직 해결이 되지 못한 상태이다. 하지만 이런 점들을 참고하면서도 자율주행자동차가 가져올 변화는 긍정적으로 판단이 된다. 이미 자율주행자동차 실험도 다른 나라들에서 충분히 검증이 되었고 현재 외적인 요소들을 협상하고 있을 것이다. 대부분의 시민들도 이미 자율주행자동차가 가져올 영향에 대해서 어느 정도 알고 있다. 우리처럼 자율주행자동차의 기능을 실현시키는 것이 간접적으로 체험할 수 있는 좋은 기회가 아닌가 생각한다.

## 참고문헌

- [1] 남두희, "자율주행자동차와 사회 변화", 한국교통연구원, 2016년, 6

### III. 프로젝트 수행 후기

연번	팀장(원)	역할 수행 내용	수행 후기
1	장정안	이미지 매칭 및 발표자료 작성.	창업연계공학설계입문 AD Project를 하면서 많은 시행 착오들이 있었다. 특히 같은 코드를 xycar에서 돌려 보아도 그게 작동이 될때가 있고 되지 않을때가 있었다.
2	정승우	이미지 매칭	임베디드 시스템이란 무엇인가를 생각 할 수 있는 활동이었다. 항상 컴퓨터로만 실행시키면 컴퓨터의 스펙이 너무좋아서 내 코드가 좋지 않아도 별로 상관이 없었는데 이 활동에서는 하드웨어를 고려하면서 많은 생각을 해봤던 것 같다.
3	정민지	원 검출	파이참에서 테스트해본 코드를 ros위에서 돌려보면 예상치 못한 에러가 많이 발생했다. 이 프로젝트를 진행하면서 에러가 발생했을 때 대처하는 능력을 키울 수 있었다.
4	정수환	주차하기	평소 컴퓨터만 사용하여 느끼지 못했던 하드웨어의 부족이 성과에 영향을 미친다는 것을 느꼈다. 또한 시간의 부족함도 느꼈다. 하지만 제안 발표에서 어렵다고 느껴졌던 주차를 성공해냈다는 것에서는 뿌듯함을 느낀다. 결과적으로 이번 프로젝트에서 나 자신에게 아쉬움과 대견함을 느꼈다.
5	장승훈	주차 표지판 인식	코드를 짜면서 매번 일어나는 카메라, opencv 오류를 볼 때마다 무엇이 문제인지 알지 못해 매우 답답했다. 결국 완성시켜서 한편으로는 뿌듯하고 신기하면서도 많은 시간을 쓰면서 나의 부족한 부분들을 알 수 있는 좋은 기회였다.
6			

※ 개인별 역할 수행 내용과 프로젝트를 수행하면서

2019학년도 2학기

느낀 점, 애로사항, 프로젝트 전·후의 팀원 개개인의 변화 모습 등을 서술