

# AD 프로젝트

## 자동 주차 시스템

---

4분반 4조

20181320 이정현, 20191623 엄석현

20191631 윤민상, 20191637 이민석, 20191643 이아연

# I 목차

- 01 주제 선정 이유
- 02 처음 구상
- 03 각 부분별 초반 구상
- 04 한계점과 해결 방안
- 05 시연 영상

## 1. 주제 선정 이유

주차 사고에 대한 뉴스를 접하게 되었다.

많은 사람들이 자율 주행에 대

**하차만**  
도로 위에서 '달리는'  
과정만을 생각

주차 중 사고 비율이  
전체 사고 비율의 30%

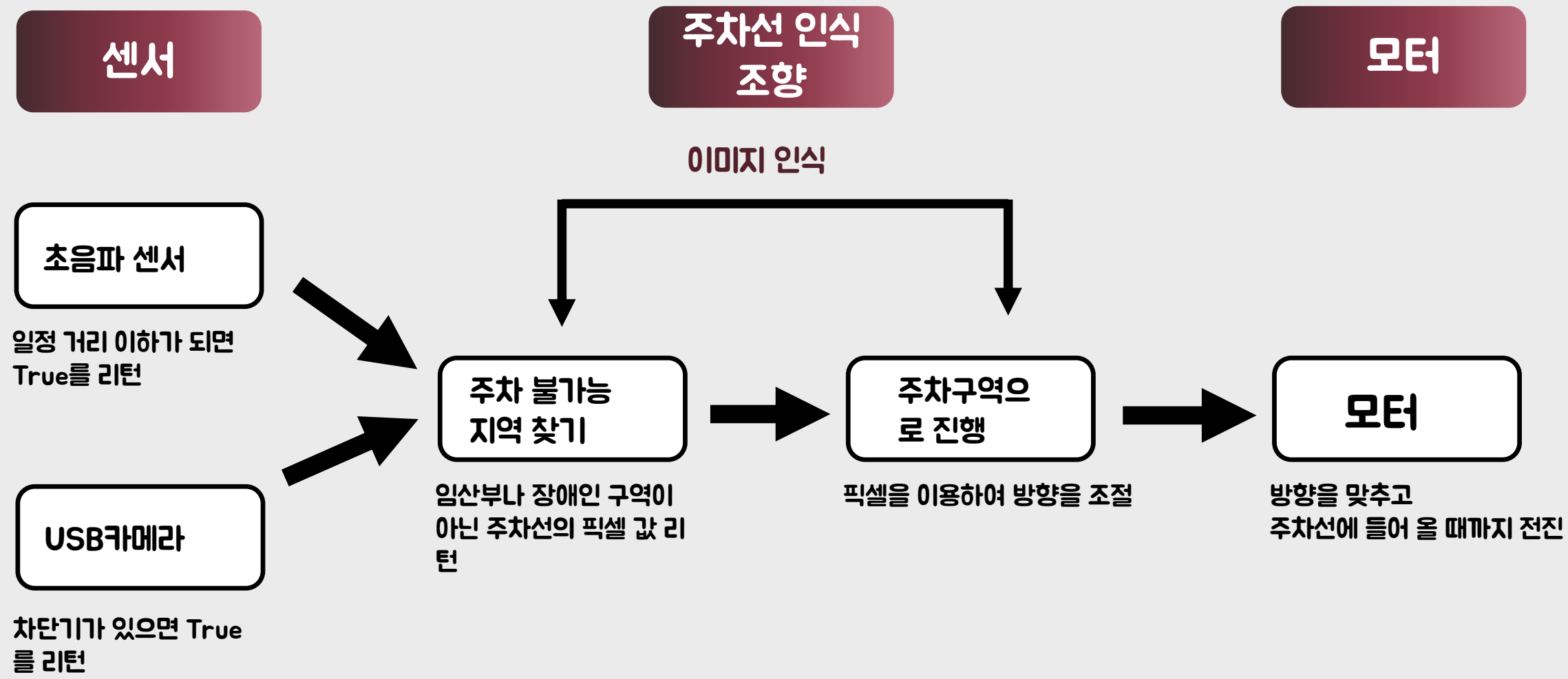
운전 초보자들이 가장  
어렵게 느끼는 것이 주차

자료 출처:

[http://imnews.imbc.com/replay/2017/nrc1200/article/4211110\\_21376.html](http://imnews.imbc.com/replay/2017/nrc1200/article/4211110_21376.html)



## 2. 처음 구상 구조들의 나열



### 3. 각 부분별 초반 구상

#### 초음파 센서



- 튀는 값 제거
- 평균과 가중치 평균 중 평균필터 선택



- 초음파 데이터를 받아 오기
- 일정 거리 이하(30cm)에 장애물이 있다면 True를 리턴

### 3. 각 부분별 초반 구상 초음파 센서

```
class MovingAverage:
    def __init__(self, n):
        self.samples = n
        self.data = []
        self.weights = list(range(1, n + 1))

    def add_sample(self, new_sample):
        if len(self.data) < self.samples:
            self.data.append(new_sample)
        else:
            self.data = self.data[1:] + [new_sample]
            # print("samples: %s" % self.data)

    def get_mm(self):
        return float(sum(self.data)) / len(self.data)
```

1. 데이터 리스트를 만들어서 인자로  $n$ 의 길이까지 데이터를 받기

2.  $n$ 개 만큼 데이터가 채워지면 가장 처음의 데이터는 버리고 계속 추가하기

3. 데이터들의 합을 데이터 리스트의 길이로 나누어서 평균을 구하기

### 3. 각 부분별 초반 구상

#### 초음파 센서

```
# def drive(angle, speed):  
#     global motor_pub  
#     drive_info = [angle, speed]  
#     pub_data = Int32MultiArray(data=drive_info)  
#     motor_pub.publish(pub_data)  
  
def callback(self, data):  
    global usonic_data  
    usonic_data = data.data
```

#### 1. 초음파의 데이터를 받아온다

```
rate = rospy.sleep(10)  
self.ma = MovingAverage(10)  
while len(self.ma.data) <= 10:  
    self.ma.add_sample(usonic_data[1])  
    rate.sleep()  
  
def exit_node(self):  
    return
```

#### 2. 앞서 만든 필터에 10개의 초음파 데이터 값을 먼저 넣어준다.

### 3. 각 부분별 초반 구상 초음파 센서

```
def obstruction_detect(self):  
    rate = rospy.Rate(10)  
  
    while True:  
        if self.ma.get_mm() - 5 < usonic_data[1] < self.ma.get_mm() + 5:  
            self.ma.add_sample(usonic_data[1])  
  
        else:  
            self.ma.add_sample(self.ma.data[-1])  
  
        rate.sleep()  
        print(self.ma.data)  
  
        if usonic_data[1] < 30 and (self.ma.getmm - 5 < usonic_data[1] < self.ma.getmm + 5):  
            return True
```

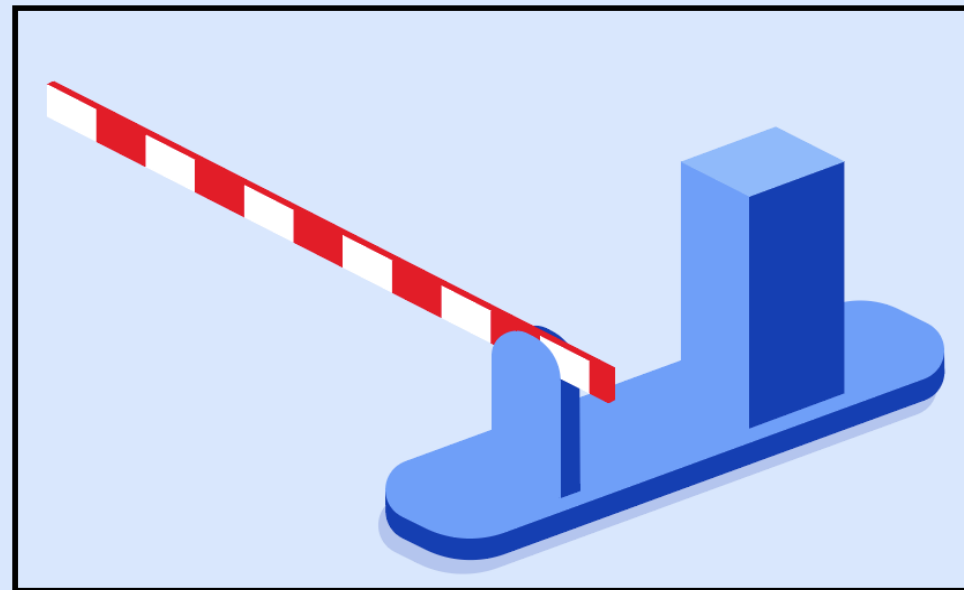
3. 초음파 데이터 값이 평균보다 5 미만으로 차이가 난다면 리스트에 추가해준다.
4. 만약 5보다 더 차이가 난다면 추가되는 값은 맨 뒤의 값이 추가 되게 한다.
5. 방해물과의 거리가 30 미만으로 떨어진다면 True를 반환한다.



### 3. 각 부분별 초반 구상

#### USB카메라

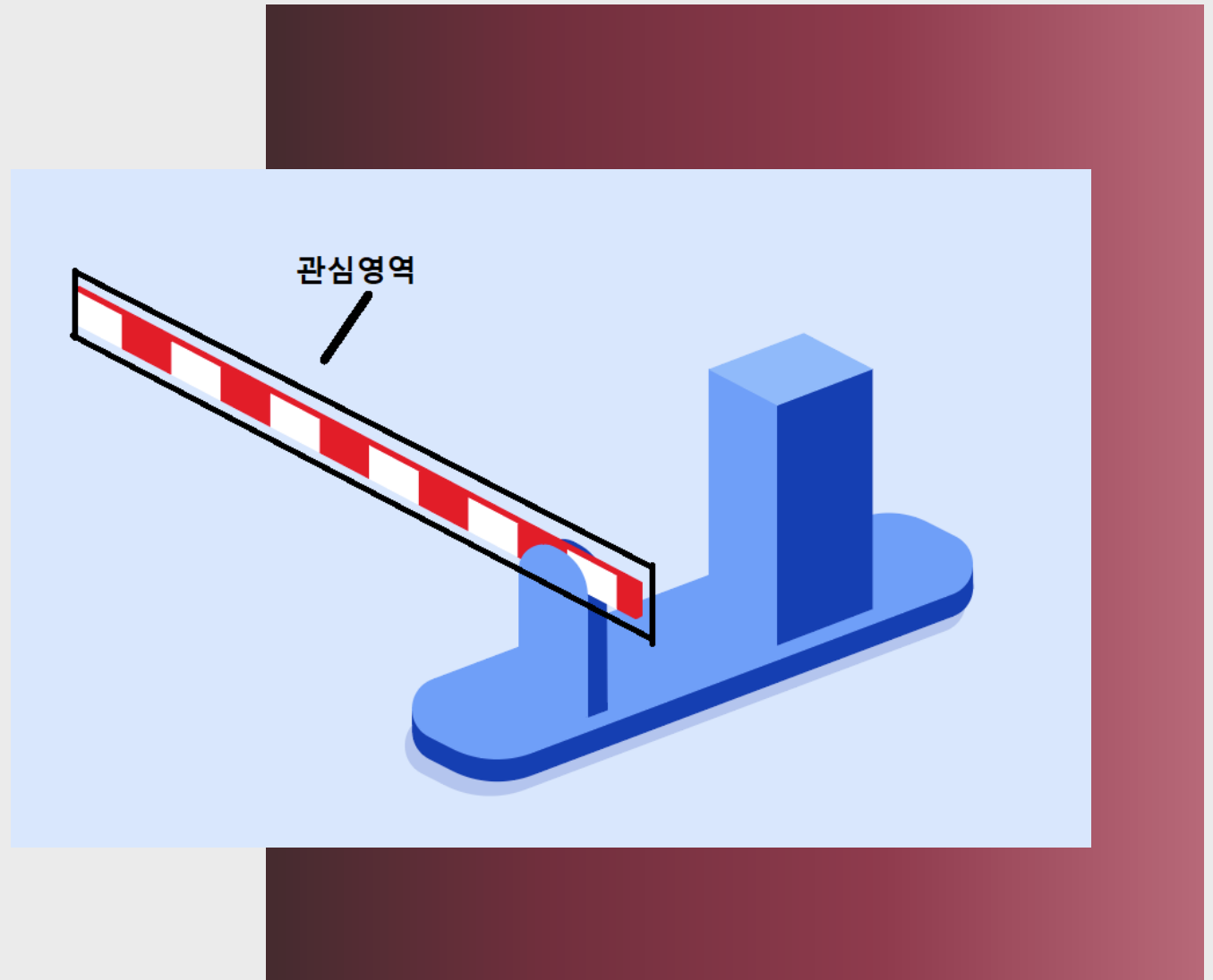
3rd party library



Breaker

### 3. 각 부분별 초반 구상 USB카메라

1. 관심영역 설정.
2. 차단기 색 평균값 설정
3. 관심영역 의 b, g, r의 색 평균을 값 계산
4. 계산한 색의 평균값과 관심영역 내 색 평균값 비교
5. 비교값이 일정범위 내 이면 차단기로 인식.



### 3. 각 부분별 초반 구상

#### USB카메라

```
roi = img[95:98 , 200:553]# 차단기를 인식할 범위
std_color = (129, 121, 214) # 차단기 색의 평균값
isbreaker = False #차단기 인식 여부
blue = 0
green = 0
red = 0
num_pixel = 4 * 353 #roi 픽셀수
```

```
#roi의 색 평균값 구하기
for i in range(95,99):
    for j in range(200,554):
        blue += img[i][j][0]
        green += img[i][j][1]
        red += img[i][j][2]

blue_avg = blue / num_pixel
green_avg = green / num_pixel
red_avg = red / num_pixel
```

#### 1. 관심영역 및 초기설정

프레임 마다 관심영역의 전체의 색  
평균값을 계산

#### 2. 관심영역의 색 평균값 계산

프레임 마다 관심영역의 전체의 색 평균  
값을 계산

### 3. 각 부분별 초반 구상

#### USB카메라

```
#roi의 색 평균값과 기준으로 잡아놓은 색의 값 비교 (오차범위 내에 들어오는지)
if (std_color[0] - 10 < blue_avg < std_color[0] + 10) and (std_color[1] - 10 < green_avg < std_color
[1] + 210) and (std_color[2] - 10 < red_avg < std_color[2] + 10) :
    isbreaker = True

return isbreaker
```

### 3. 기준 색 평균값 과 받아들이는 색 평균 비교

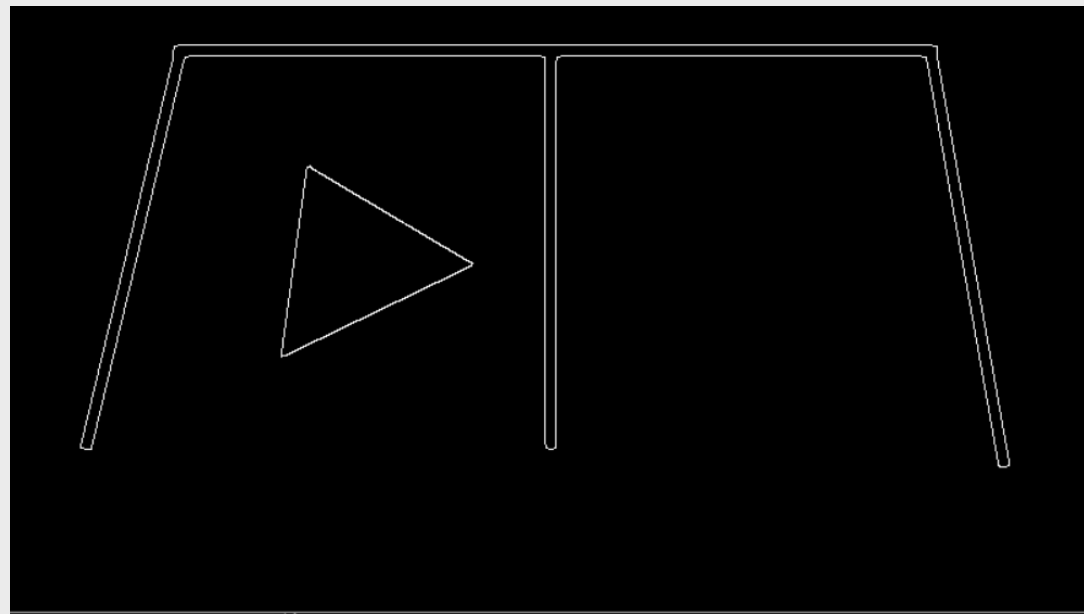
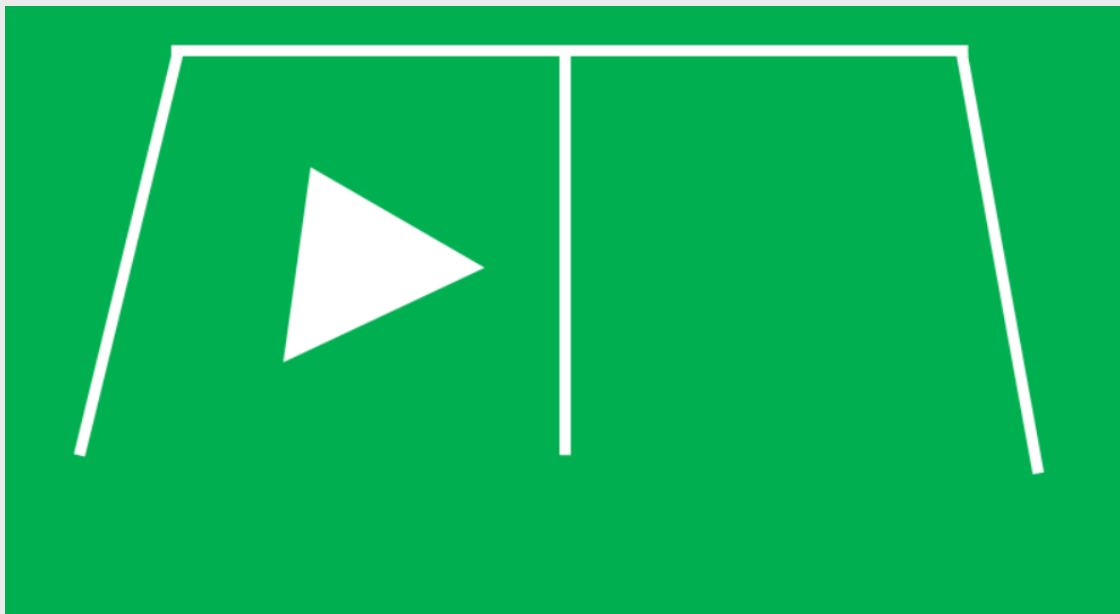
계산한 색의 평균과 초기 설정한 색 평균값 비교 범위 내에 있을시엔 True 리턴.

### 3. 각 부분별 초반 구상

주차 불가능 지역 찾기

#### 1. 영역을 잘라서 구역을 윤곽선 찾기

- 먼저 카메라로 받아 드린 이미지에서 윤곽선을 찾음

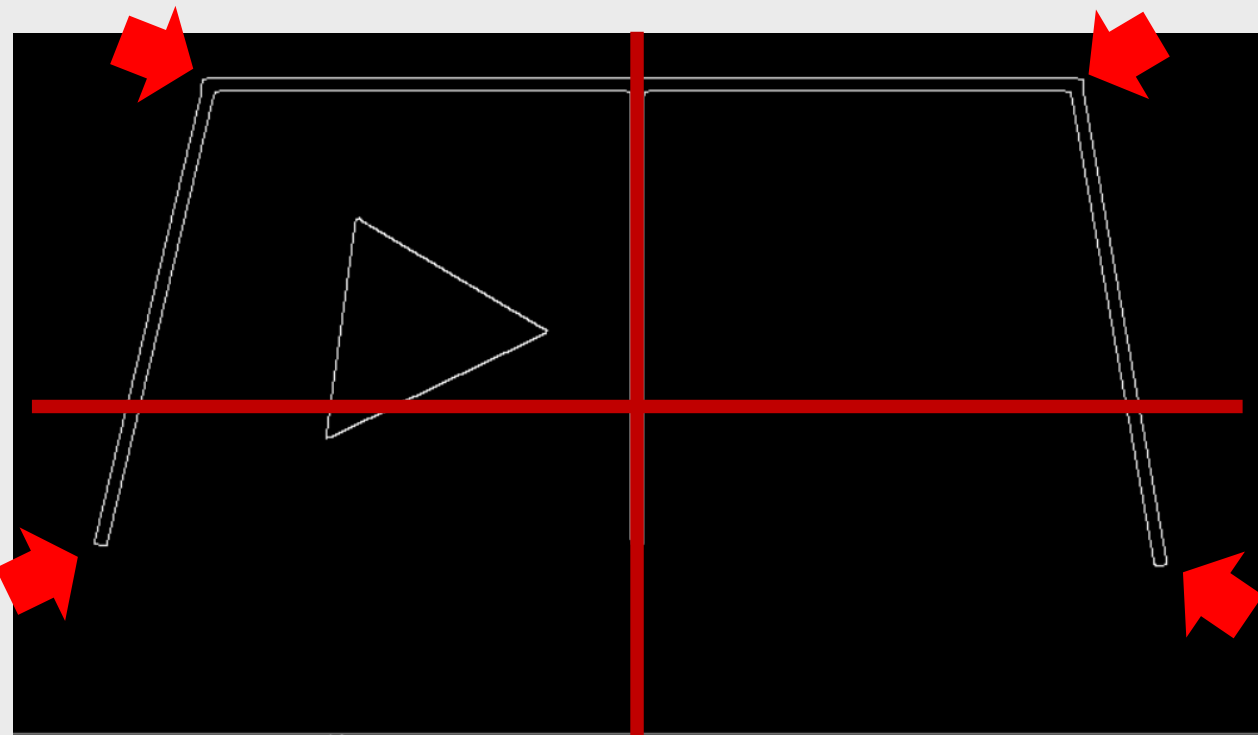


### 3. 각 부분별 초반 구상

#### 주차 불가능 지역 찾기

## 2. 주차 구역이 시작과 끝 부분을 찾기

- 이미지에서 구역을 4개로 나눈다
- 4개로 나눈 구역에서 주차선이 시작되는 부분을 찾는다.

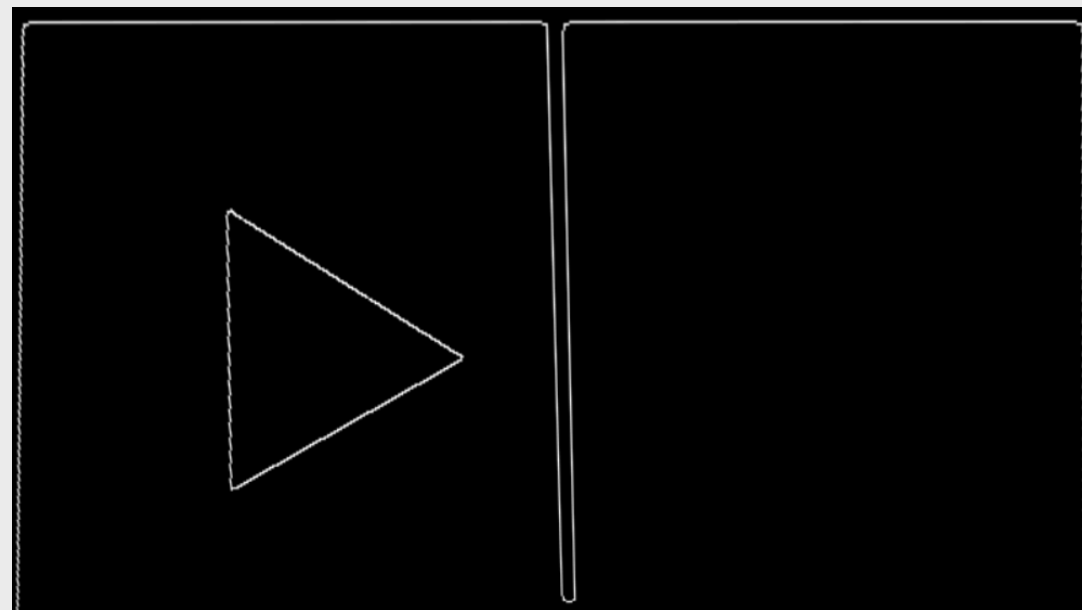
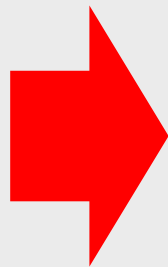
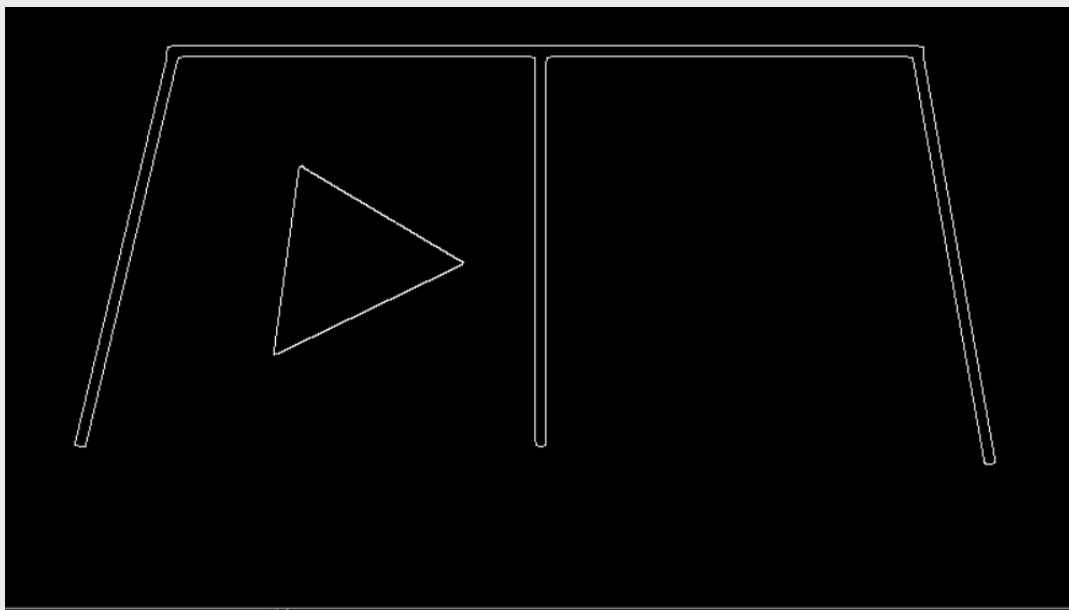


### 3. 각 부분별 초반 구상

주차 불가능 지역 찾기

### 3. 시작과 끝 부분을 이용하여 이미지 변환하기

- 앞에 과정에서 찾은 4개의 점을 기준으로 이미지를 변환

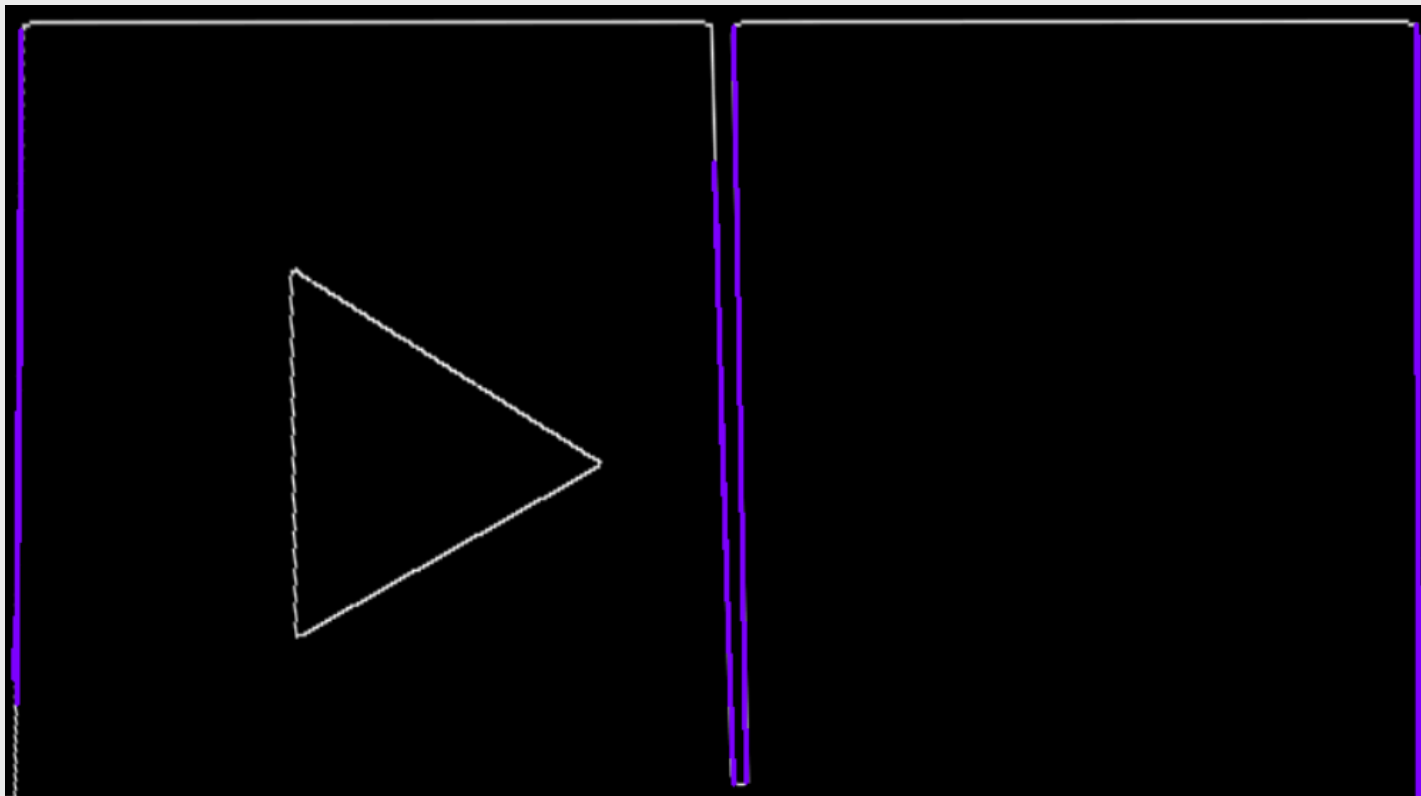


### 3. 각 부분별 초반 구상

주차 불가능 지역 찾기

## 4. 확률적 허프 변환으로 주차 차선 찾기

- 변환한 이미지에서 가로 줄 찾는다.





### 3. 각 부분별 초반 구상

#### 주차 불가능 지역 찾기

## 5. 찾은 차선 안에 이미지가 있는지 없는지 확인

- 찾은 선들의 x값을 모두 리스트에 저장
- 리스트를 한 번 정렬
- 정렬한 값들 중에 필요한 값들만 새로운 리스트에 추가
- 변환한 이미지에서 x값들을 기준 자르고 그 범위안에 흰색 점 계수로 이미지가 있는지 없는지를 확인

차선은 있는 지역은 [5, 398, 792]

5 부터 398 까지는 5796의 흰 점이 있습니다.

398 부터 792 까지는 4345의 흰 점이 있습니다.

## 6. 주차가 가능한 지역이 몇 픽셀에 있는지를 리턴하기

- 앞에서 찾은 흰 색 점의 개수가 일정 범위 이하에 있으면 그 값의 픽셀 값을 리턴한다.

### 3. 각 부분별 초반 구상

#### 주차 구역으로 진행

## 1. 앞의 과정에서 받은 픽셀을 이용해 이동

```
#수평까지 이동 ( 원래 코드에서 왼/오 canny값 대신 받아와준 픽셀값 사용 )
def gotoplace(self, pixel_left, pixel_right, mid):
    self.pixle_left= pixel_left
    self.pixle_right= pixel_right

    mid = (pixel_right + pixel_left) // 2

    if mid < 280:
        angle = -40
    elif mid > 360:
        angle = 40
    else:
        angle = 0
    return angle
```

1. 자율 주행 코드를 통해 배웠던 것을 바탕으로 angle값을 지정해줌.

2. 주차 라인의 픽셀을 받아오게 될 시, 양 쪽 라인의 값 대신 앞 과정에서 받아온 올바른 주차 위치의 양 쪽 픽셀값을 바탕으로 angle값을 지정해서 주차 공간 안으로 주행하게 해줌.

### 3. 각 부분별 초반 구상 주차 구역으로 진행

## 2. 수평이 될 시 True를 리턴해줌

```
#받아온 왼/오 canny와 인식한 왼/오 픽셀값이 일치 시 true를 리턴
if left == pixel_left and right == pixel_right
    return True
```

계속 받아오고 있는 차선의 좌우 값과 앞에서 받아온 좌우 픽셀값이 같아주다면 True를 리턴

### 3. 각 부분별 초반 구상 주차 구역으로 진행

## 3, 4. 직진 후 주차 완료까지

```
#ultrasonic에서 true 리턴해주면 운전 멈추도록
def stopsign(self, obstacle, recognize):
    obstacle = self.obstacle
    recognize = self.recognize
    if obstacle == recognize == True:
        speed = 0
        return speed
```

1. 1번 코드가 계속 돌아가며  
xycar는 계속 직진하고 있을  
것이다.

2. 직진 하다가 앞에서 짤 초  
음파 센서 코드가 주차막과의  
거리를 인식하고 True값을  
받아올 시 xycar의 speed를  
0으로 지정해주어 멈추고 주  
차를 완료해 줌.

## 4. 문제점과 해결 방안

### 초음파 센서와 USB카메라

# 한계점

초음파 센서의 필터 값을 제대로 사용하지 못함

세상에 많은 차단기가 존재하나 규격이 통일되어있지 않음  $\Rightarrow$  한정적인 조건

그러나 차단기를 인식하는 주된 방법은 초음파 센서를 이용한 것.

이 방법은 초음파 센서를 보조하는 기능이다.

## 4. 한계점과 해결방안

### 주차 불가능 지역 찾기

# 한계점

1. 주변 다른 물건들이 있거나 차선이 한 화면에서 인식 되지 않으면 주차 시작과 끝 부분을 제대로 찾지 못한다.

3. 제대로 된 변환을 하기 힘들기에 직선을 찾는 것에 어려움을 겪는다.

3. 또한 픽셀을 넘겨주지 못한다.

# 해결 방안

1. 이미지를 받고 나서 직선의 기울기를 측정하고 고정된 값으로 이미지를 변환했다.

2. 주차 지역이 몇 개 있는 지를 확인하고 그 개수에 맞게 영역을 잘라서 흰색 픽셀 수 검사했다.

3. 픽셀을 넘겨주는 것 대신해서 방향을 넘겨 주었다.

## 4. 한계점과 해결방안

### 주차 구역으로 진행

# 한계점

1. 트랙 위의 주행이 아닐 때 차선과 주차 구역을 비교 후 True를 리턴해주지 못한다.

2. 픽셀값을 넘겨주는 것이 불가능해 올바른 주차 구역의 방향을 넘겨주는 것으로 코드를 수정했다.

# 해결 방안

1. 트랙 위의 주행하는 경우엔 트랙의 차선을 인식하고, 트랙이 아닌 곳에서 if문이 만족되지 않는 상황에선 직진하도록 수정하였다.

2. 받아온 주차 방향을 이용해 조향각을 설정한 후 올바른 주차 구역까지 찾아가는 과정은 직접 일일이 해보면서 하게 되었다.

## 5. 시연 영상





# THANK YOU!

---

MADE BY HENDO