프로그래머스 자율주행 데브코스

차선인식 모의 경진대회

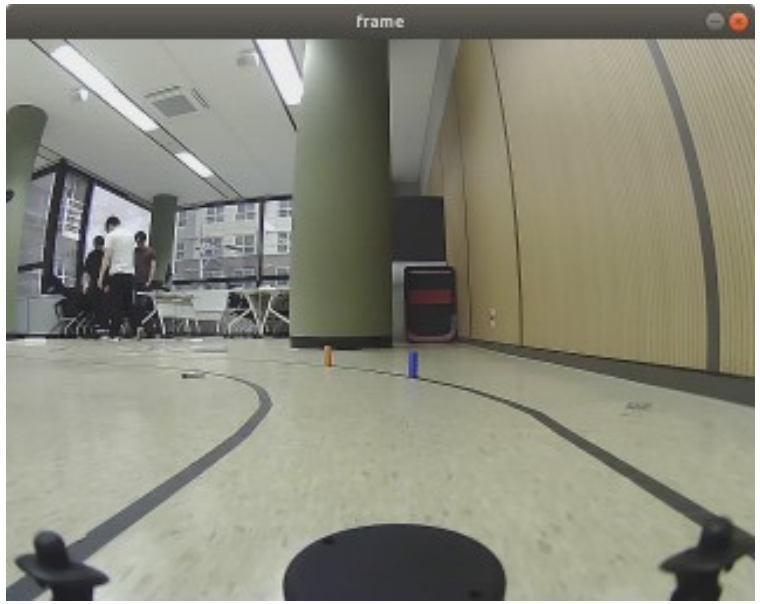
Team B1

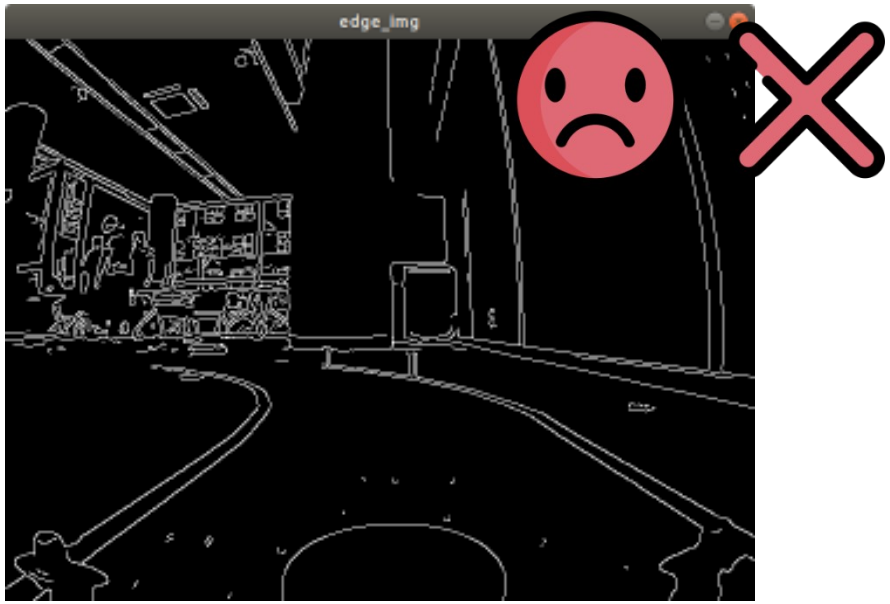
박한음
송미희
이상철
이창준

목차

1-1. Canny Edge 개선

```
edge_img = cv2.Canny(np.uint8(blur_gray), 60, 70)
```

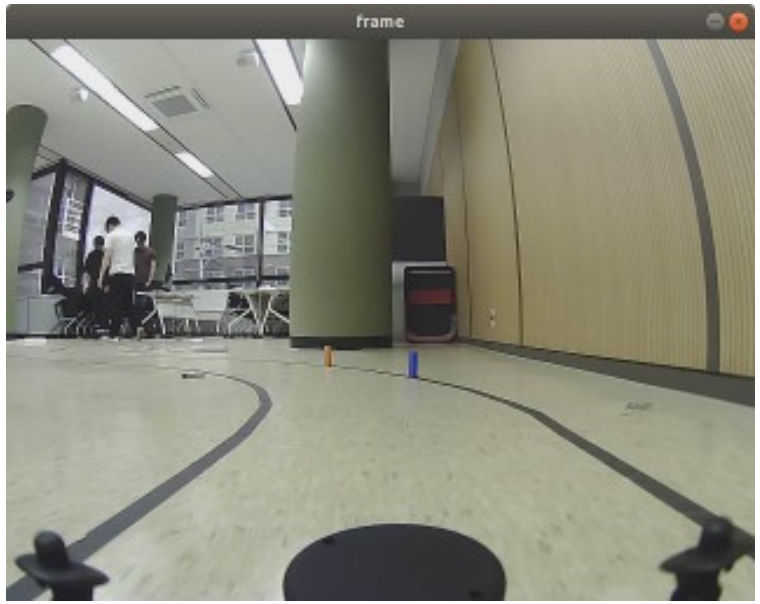




기존의 threshold 값은 노이즈가 많이 발생한다.

1-1. Canny Edge 개선

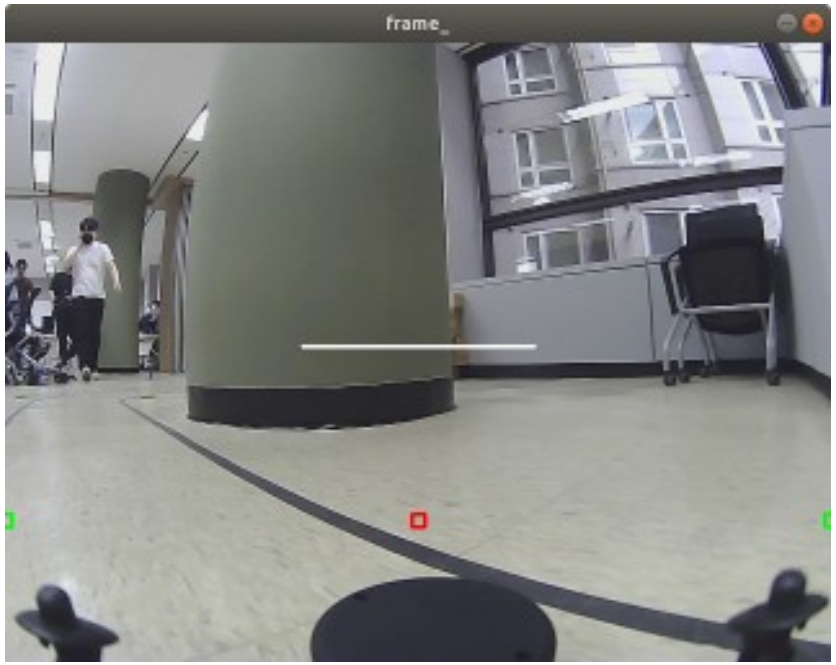
```
edge_img = cv2.Canny(np.uint8(blur_gray), 100, 200)
```



low threshold와 high threshold 값을 적절하게 수정하여 해당 문제를 해결했다.

1-2. Lines 검출 개선

```
if (slope < 0) and (x2 < Width/2 - 90):
    left_lines.append([Line.tolist()])

elif (slope > 0) and (x1 > Width/2 + 90):
    right_lines.append([Line.tolist()])
```



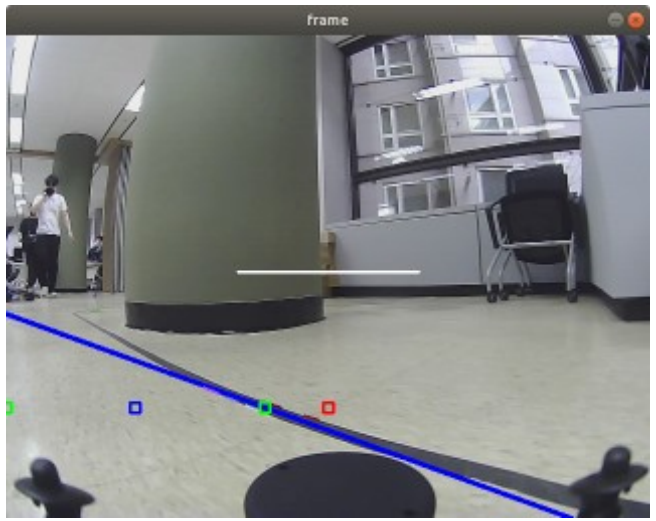
기존의 코드는 왼쪽 차선과 오른쪽 차선을 구분할 때 기울기와 검출한 차선의 x좌표를 모두 고려했다.

→ 곡률이 큰 곡선 구간을 주행할 때 왼쪽 차선과 오른쪽 차선이 정확하게 구분되지 않는 문제 발생

1-2. Lines 검출 개선

```
if slope < 0:
    left_lines.append([Line.tolist()])

else:
    right_lines.append([Line.tolist()])
```

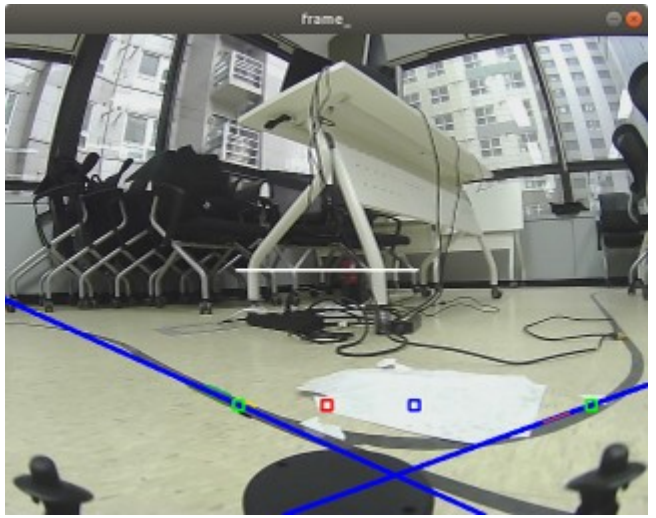
차선의 x좌표를 제외하고 오직 기울기만 고려하도록 변경



1-2. Lines 검출 개선

```
if slope < 0:
    left_lines.append([Line.tolist()])

else:
    right_lines.append([Line.tolist()])
```

하지만 기울기로만 판단하자 왼쪽 차선과 오른쪽 차선을 잘못 인식하는 문제가 발생했다.

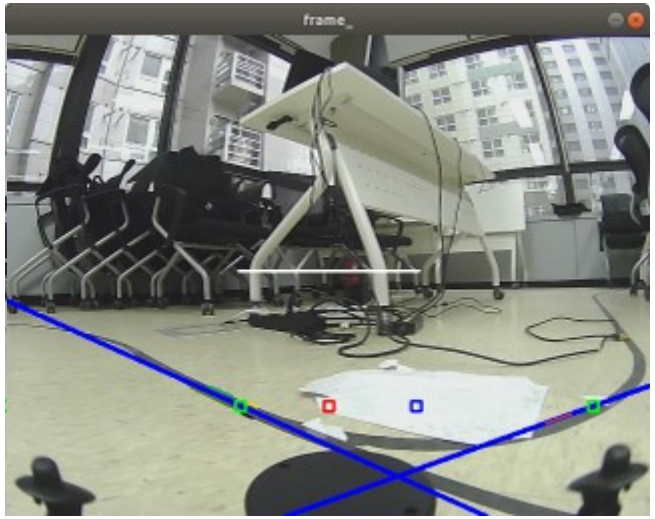


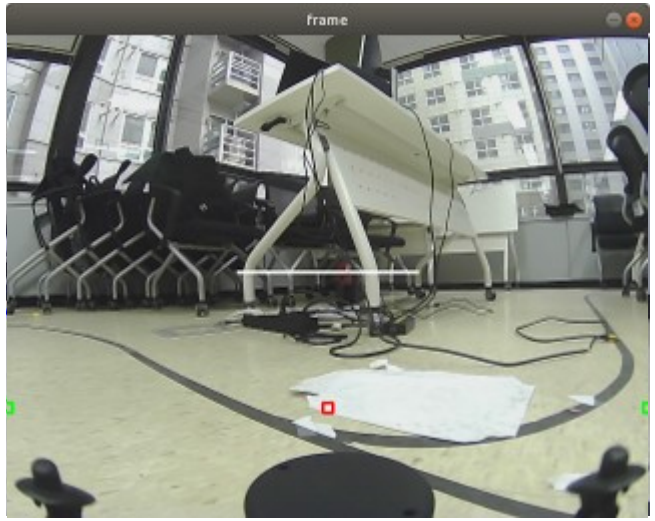
1-2. Lines 검출 개선

```
if slope < 0:
    left_lines.append([Line.tolist()])
    left_x_sum += (x1 + x2)/2

else:
    right_lines.append([Line.tolist()])
    right_x_sum += (x1 + x2)/2
```

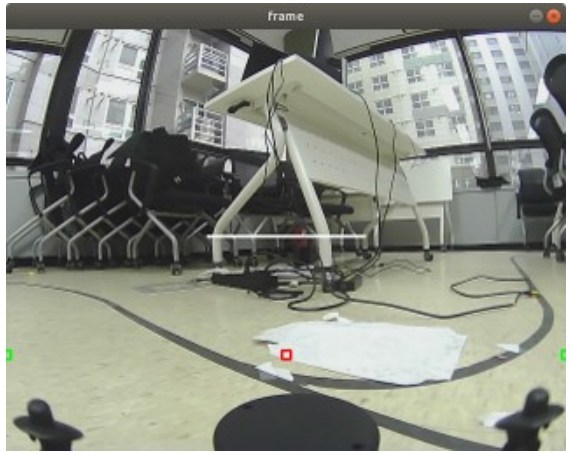
```
left_x_avg = left_x_sum / len(left_lines)
right_x_avg = right_x_sum / len(right_lines)

if left_x_avg > right_x_avg:
    left_lines = []
    right_lines = []
```

검출한 왼쪽 차선의 x좌표와 오른쪽 차선의 x좌표가 서로 엇갈리면 해당 차선을 전부 초기화





1-2. Lines 검출 개선



차선을 전부 초기화하여 왼쪽 차선과 오른쪽 차선을 모두 인식하지 못하면 lpos는 0, rpos는 640을 반환하여 조향각이 0이 된다.

→ 차선을 인식하지 못하면 차선을 이탈하는 문제 발생

```
prev_angle = 0
```

```
if lpos == 0 and rpos == 640:
    steer_angle = prev_angle
```

```
drive(wmm_angle, wmm_speed)

prev_angle = steer_angle
```

차선을 인식하지 못하는 상황 (lpos == 0, rpos == 640)
에서는 이전의 조향각을 유지하며 주행하도록 함

2-1. 종방향 제어

```
def velocity_control(l_slope, r_slope):
    global Offset

    if abs(l_slope) < 0.45 or abs(r_slope) < 0.45:
        Offset = 370
        speed = 17

    else:
        Offset = 340
        speed = 50

    return speed
```

차선의 기울기가 일정 값보다 작다면?

→ **곡선 주행**

→ **ROI를 밑으로 내리고 속도는 느리게**

차선의 기울기가 일정 값보다 크다면?

→ **직선 주행**

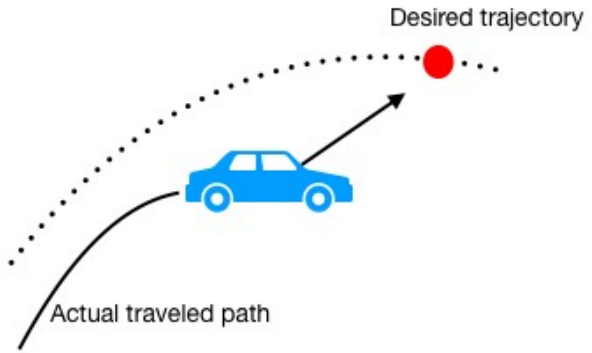
→ **ROI를 위로 내리고 속도는 빠르게**

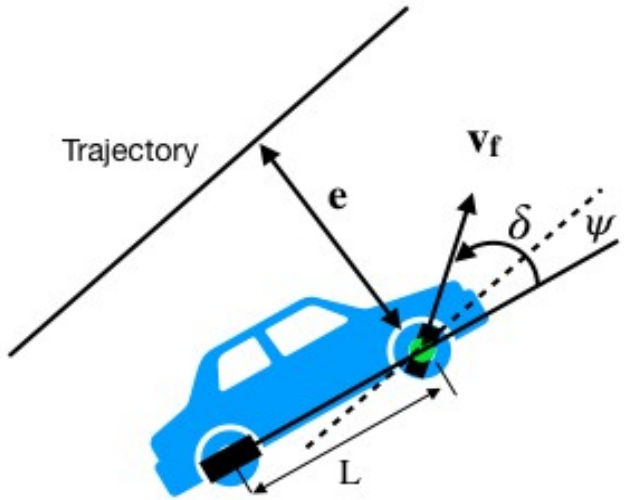
속도가 빠른 구간에서는 더욱 먼 곳을 바라봐야
횡방향 제어가 안정적이기 때문이다.

2-2. 횡방향 제어

Stanly Method

경로 추종 알고리즘 중 하나로, 차량 전륜축의 중심을 차량의 기준점으로 사용하여 경로에서 가장 가까운 지점을 결정한 다음 해당 경로로 추종하는 방식이다.





ψ Yaw 각도 오차(Heading error)에 대한 Term이다.

$\tan^{-1}(\frac{ke}{v_f})$ 횡방향 오차(Cross track error - CTE)에 대한 Term이다.

$$\delta = \psi + \tan^{-1}(\frac{ke}{v_f})$$

Gain, *Lateral error*, *Steering angle*, *Heading error*, *Velocity*

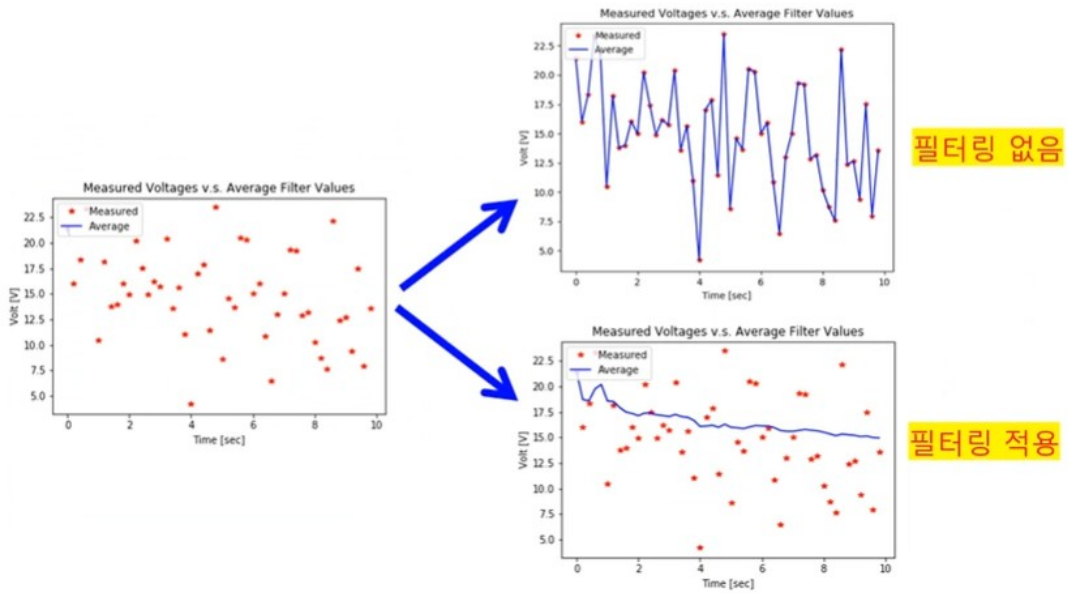
사진 출처: https://dingyan89.medium.com/three-methods-of-vehicle-lateral-control-pure-pursuit-stanley-and-mpc-db8cc1d32081

2-2. 횡방향 제어

$$\delta = k * \tan^{-1}(cte)$$

```python
def get_steer_angle(curr_position):
    k = 1.0

    if -0.2 < curr_position < 0.2 :  # 좀 더 천천히 조향해도 괜찮은 상황
        k = 1.0

    elif curr_position > 0.4 or curr_position < -0.4 :  # 신속하게 가운데로 들어와야 함
        k = 4.0

    else:   # 그 중간의 경우 계수는 linear 변화
        k = 20.0 * abs(curr_position) - 3.0

    steer_angle = k * math.atan(curr_position)* 180 / math.pi

    return steer_angle
```

2-3. 필터 기반 조향각 제어




```
mm_angle.add_sample(steer_angle)
wmm_angle = mm_angle.get_wmm()

mm_speed.add_sample(speed)
wmm_speed = mm_speed.get_wmm()

drive(wmm_angle, wmm_speed)
```

속도와 조향각을 필터링하여 자동차가
흔들리지 않고 부드럽게 주행하도록 했다.

THANK YOU