- **허프 변환 기반 차선 인식 흐름도 요약**

1. 동영상에서 영상 프레임 획득
2. 영상 프레임을 OpenCV함수로 처리
3. 영상 처리
   A. GrayScale (흑백 이미지로 변환)
   B. Gaussian Blur (노이즈 제거)
   C. Canny Edge (외곽선 Edge 추출)
   D. ROI (관심 영역 잘라 내기)
   E. HoughLineP (선분 검출)
4. 차선 위치 찾고 화면 중앙에서 어느 쪽으로 치우쳤는지 파악
5. 핸들을 얼마나 꺾을지 결정 (조향각 계산)

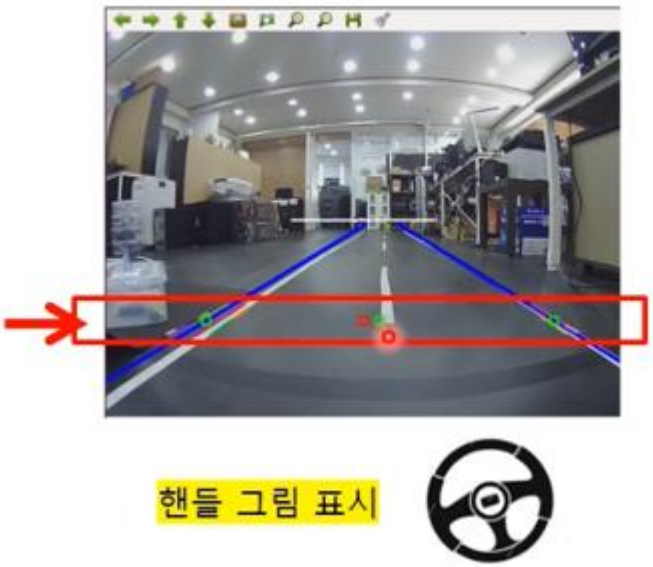



그림 1. 실행 화면

- **허프 변환 기반 차선 인식 구현 설명**

### 1. 동영상에서 프레임 획득

```
cap = cv2.VideoCapture('hough_track.avi')

while not rospy.is_shutdown():

    ret, image = cap.read()
    time.sleep(0.03)
```

동영상 파일을 열기

동영상 파일에서 이미지 한장 읽기

### 2. 동영상 프레임을 OpenCV함수로 처리

```
pos, frame = process_image(image)
```

허프변환 기반으로 영상 처리 진행.
차선 찾고 위치 표시하기

### 3. 영상 처리

```
def process_image(frame):
    global Width
    global Offset, Gap

    # gray
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    # blur
    kernel_size = 5
    blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size), 0)

    # canny edge
    low_threshold = 60
    high_threshold = 70
    edge_img = cv2.Canny(np.uint8(blur_gray), low_threshold, high_threshold)

    # HoughLinesP
    roi = edge_img[Offset : Offset+Gap, 0 : Width]
    all_lines = cv2.HoughLinesP(roi,1,math.pi/180,30,30,10)
```

Gray 색상으로 변환

Gaussian Blur 처리

Canny Edge 외곽선 따기

ROI 영역에서 선분 찾기

```
# divide left, right lines
if all_lines is None:
    return 0, 640
left_lines, right_lines = divide_left_right(all_lines)

# get center of lines
frame, lpos = get_line_pos(frame, left_lines, left=True)
frame, rpos = get_line_pos(frame, right_lines, right=True)

# draw lines
frame = draw_lines(frame, left_lines)
frame = draw_lines(frame, right_lines)
frame = cv2.line(frame, (230, 235), (410, 235), (255,255,255), 2)

# draw rectangle
frame = draw_rectangle(frame, lpos, rpos, offset=Offset)

return lpos, rpos
```

선분을 왼쪽 것과 오른쪽 것으로 분류

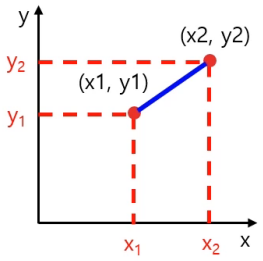
선분의 정보를 받아서 이미지에 차선 그리고, 위치 구하기

ROI 영역 안에서 허프변환을 통해 구한 차선을 랜덤한 색상으로 그리기

차선과 화면중앙에 사각형 그리기

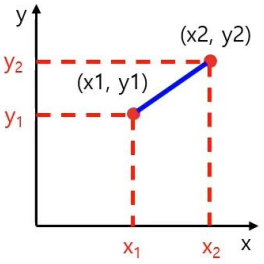
## ➔ 선분을 왼쪽과 오른쪽 것으로 분류하기

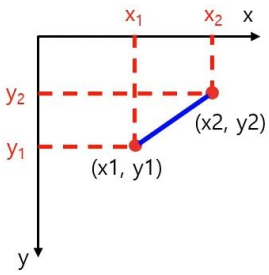
```
# left lines, right lines
def divide_left_right(lines):

    global Width
    low_slope_threshold = 0
    high_slope_threshold = 10

    slopes = []
    new_lines = []

    for line in lines:
        x1, y1, x2, y2 = line[0]
        if x2 - x1 == 0:
            slope = 0
        else:
            slope = float(y2-y1) / float(x2-x1)
        if (abs(slope) > low_slope_threshold) and (abs(slope) < high_slope_threshold):
            slopes.append(slope)
            new_lines.append(line[0])
```




선분의 기울기를 구해서
기울기 절대값이 10 이하인 것만 추출

허프변환 함수로 검출한 선분들의
기울기를 비교하여 왼쪽 차선과
오른쪽 차선을 구분




```
# divide lines left to right
left_lines = []
right_lines = []

for j in range(len(slopes)):
    Line = new_lines[j]
    slope = slopes[j]
    x1, y1, x2, y2 = Line
    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()])

return left_lines, right_lines
```

OpenCV 좌표계에서는 아래방향으로
y가 증가하므로 기울기 계산법이
다르다 (주의!)

화면의 왼쪽에 있는 선분 중에서
기울기가 음수인 것들만 모음

화면의 오른쪽에 있는 선분 중에서
기울기가 양수인 것들만 모음

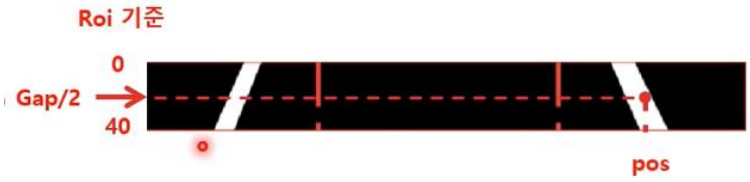
➔ 선분의 정보를 받아서 이미지에 차선을 그리고, 위치 구하기

```
# get lpos, rpos
def get_line_pos(img, lines, left=False, right=False):
    global Width, Height
    global Offset, Gap

    m, b = get_line_params(lines)

    if m == 0 and b == 0:
        if left:
            pos = 0
        if right:
            pos = Width
    else:
        y = Gap / 2
        pos = (y - b) / m

        b += Offset
        x1 = (Height - b) / float(m)
        x2 = ((Height/2) - b) / float(m)

        cv2.line(img, (int(x1), Height), (int(x2), (Height/2)), (255, 0,0), 3)

    return img, int(pos)
```

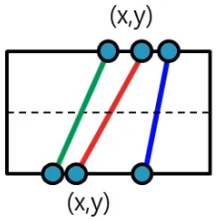

차선이 인식되지 않으면
left 는 0값을
right는 width(640)값으로 설정함 (화면의 끝 좌표)

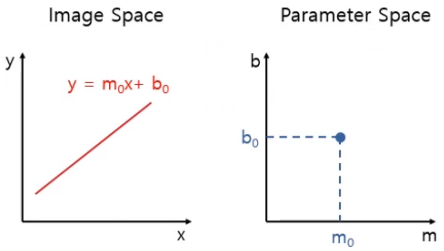
직선의 방정식에서 y=20 넣어서 x 좌표 찾음.

640x480 원본 이미지의 맨 아래(y 값이 480)의 x1과
이미지 중간(y값이 240)의 x2를 구해
(x1,480)와 (x2,320) 두 점을 잇는 파란색 선을 그린다.




```
# get average m, b of lines
def get_line_params(lines):
    # sum of x, y, m
    x_sum = 0.0
    y_sum = 0.0
    m_sum = 0.0

    size = len(lines)
    if size == 0:
        return 0, 0

    for line in lines:
        x1, y1, x2, y2 = line[0]

        x_sum += x1 + x2
        y_sum += y1 + y2
        m_sum += float(y2 - y1) / float(x2 - x1)

    x_avg = x_sum / (size * 2)
    y_avg = y_sum / (size * 2)
    m = m_sum / size
    b = y_avg - m * x_avg

    return m, b
```




허프변환 함수로 찾아낸 직선을 대상으로
Parameter Space (m, b 좌표계)에서
m의 평균값을 먼저 구하고,
그걸로 b의 값을 구함

m의 평균값을 구하는 이유는
허프변환 함수의 결과로 하나가 아닌
여러 개의 선이 검출되기 때문임.
찾은 선들의 평균값을 이용하고자 함.

$y = m_0x + b_0$

➔ ROI영역 안에서 허프 변환을 통해 구한 차선을 랜덤한 색상으로 그리기

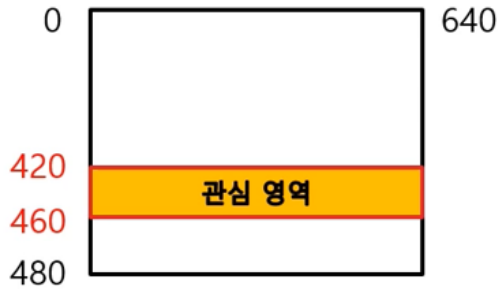
```
Width = 640
Height = 480
Offset = 420
Gap = 40
```

영상 사이즈는 가로세로 640x480

ROI 영역 = 세로 480 크기에서 420~460, 40픽셀 만큼만 잘라서 사용

```
# draw lines
def draw_lines(img, lines):
    global Offset
    for line in lines:
        x1, y1, x2, y2 = line[0]
        color = (random.randint(0, 255), random.randint(0, 255),
                 random.randint(0, 255))
        img = cv2.line(img, (x1, y1+Offset), (x2, y2+Offset), color, 2)
    return img
```

허프변환 함수로 검출된 모든 선분을 알록달록하게 출력




➔ 차선과 화면 중앙에 4각형 그리기

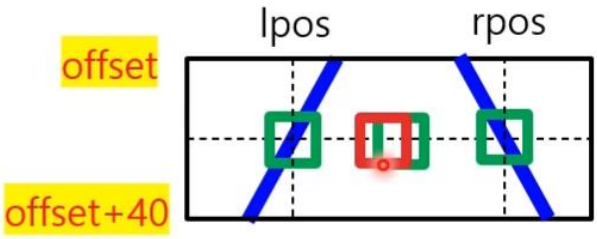
```
# draw rectangle
def draw_rectangle(img, lpos, rpos, offset=0):

    center = (lpos + rpos) / 2

    cv2.rectangle(img, (lpos - 5, 15 + offset),
                       (lpos + 5, 25 + offset),
                       (0, 255, 0), 2)
    cv2.rectangle(img, (rpos - 5, 15 + offset),
                       (rpos + 5, 25 + offset),
                       (0, 255, 0), 2)
    cv2.rectangle(img, (center-5, 15 + offset),
                       (center+5, 25 + offset),
                       (0, 255, 0), 2)
    cv2.rectangle(img, (315, 15 + offset),
                       (325, 25 + offset),
                       (0, 0, 255), 2)
    return img
```

lpos 위치에 녹색 사각형 그리기

rpos 위치에 녹색 사각형 그리기

lpos rpos 사이에 녹색 사각형 그리기

화면 중앙에 빨강 사각형 그리기




## 4-5. 차선 위치 찾고 화면 중앙에서 어느 쪽으로 치우쳤는지 파악 및 조향각 결정

```
center = (pos[0] + pos[1]) / 2
angle = 320 - center
steer_angle = angle * 0.4
draw_steer(frame, steer_angle)
```

왼쪽과 오른쪽 차선의 중간점과
화면 중앙과의 차이를 가지고
핸들 조향각을 결정해서
핸들 그림 표시하기