Lane Detection

2022년 2학기

목차

개요

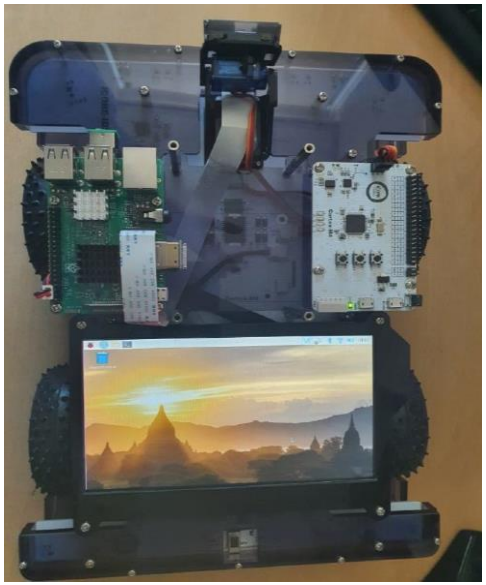
- **차선인식**
  - ✓ 차선을 찾을 수 없다면 어디로 운전해야 하는지에 대한 판단이 모호해진다. 이를 위해서 정확한 차선인식이 필요
  - ✓ 하지만, 카메라로 들어온 영상에는 노이즈, 렌즈에 의한 왜곡현상, 빛에 의한 그림자와 반사 등 여러 해결해야 할 사항들이 존재

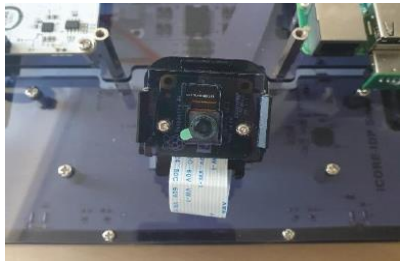
- **차선인식 알고리즘 목적**
  - ✓ 카메라로부터 들어오는 영상에서 차선을 추출 후 추출된 차선을 기준으로 차량의 조향각을 계산하고 이 값을 직렬 통신으로 제어기에 보내 차량을 제어

라즈베리 파이 설정

- **실습환경**

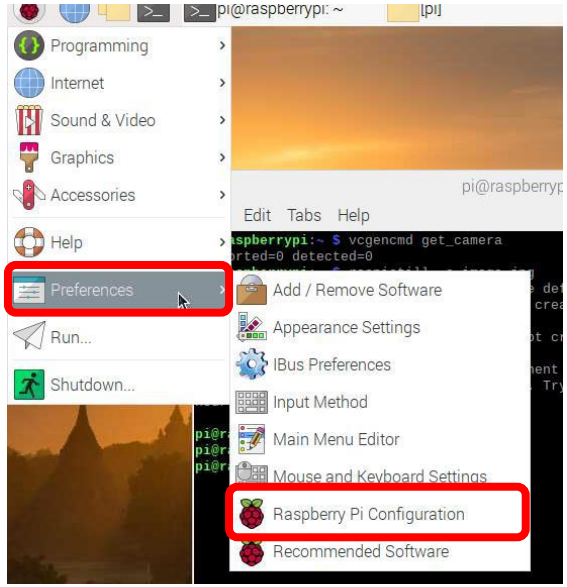
  ✓ 카메라와 라즈베리 파이가 연결된 상태의 스마트 자동차의 보드에 전원을 인가한다. (화면이 켜짐과 동시에 빨간 불이 점등)

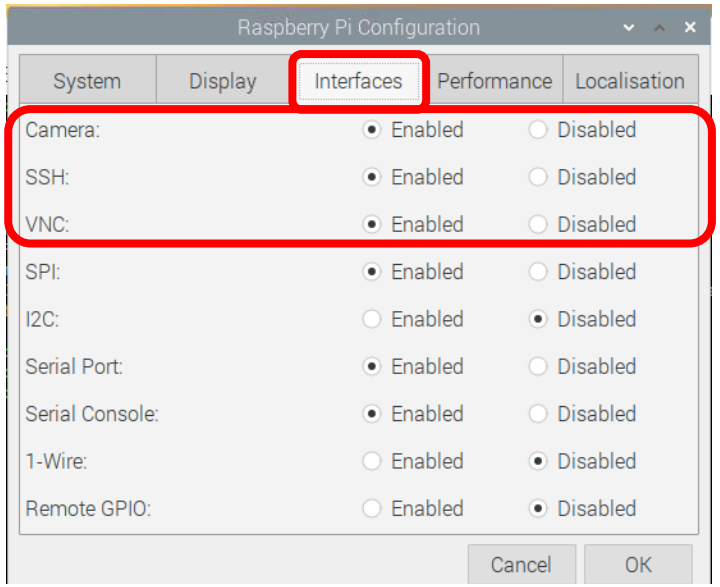




라즈베리 파이 설정

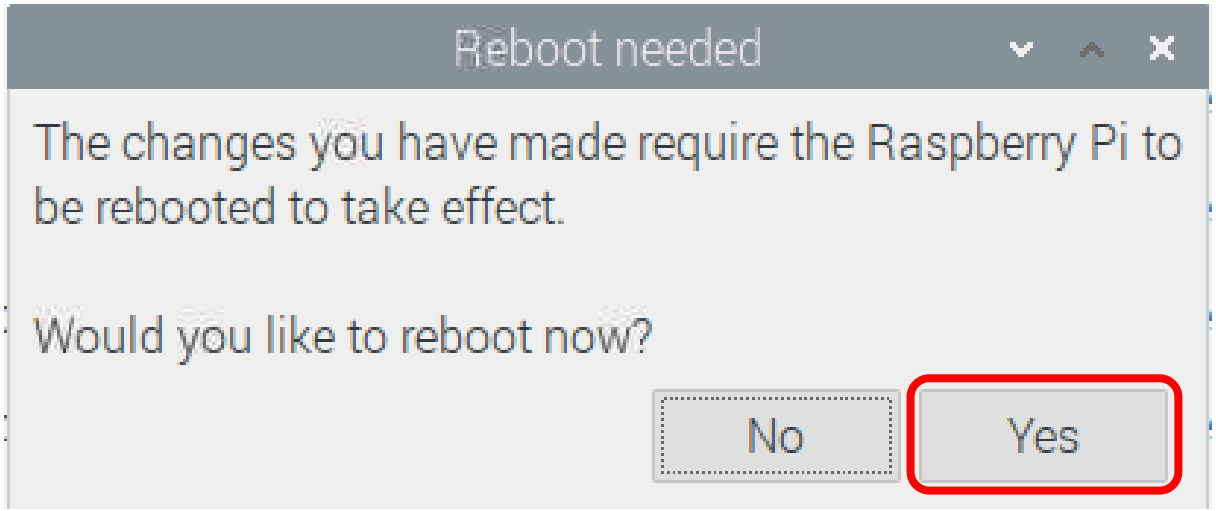
- **실습환경**
  - ✓ 터치 스크린을 이용하여 왼쪽 상단의 라즈베리 파이 메뉴를 선택하여 Preferences → Raspberry Pi Configuration으로 이동
  - ✓ 상단 메뉴의 Interfaces를 선택하고 Camera, SSH, VNC를 모두 Enabled로 변경
  - ✓ Reboot 메시지가 출력되면 Yes 클릭





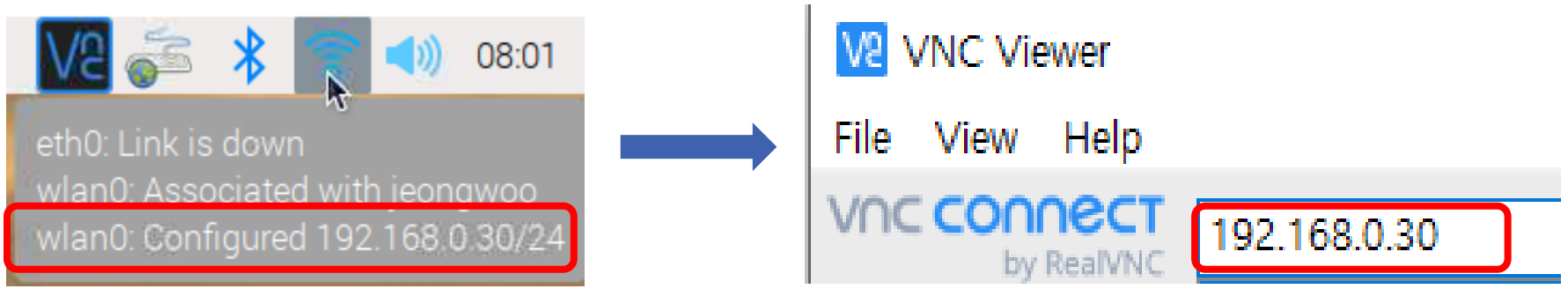
라즈베리 파이 설정

- **실습환경**
  - ✓ Reboot 메시지가 출력되면 Yes 클릭



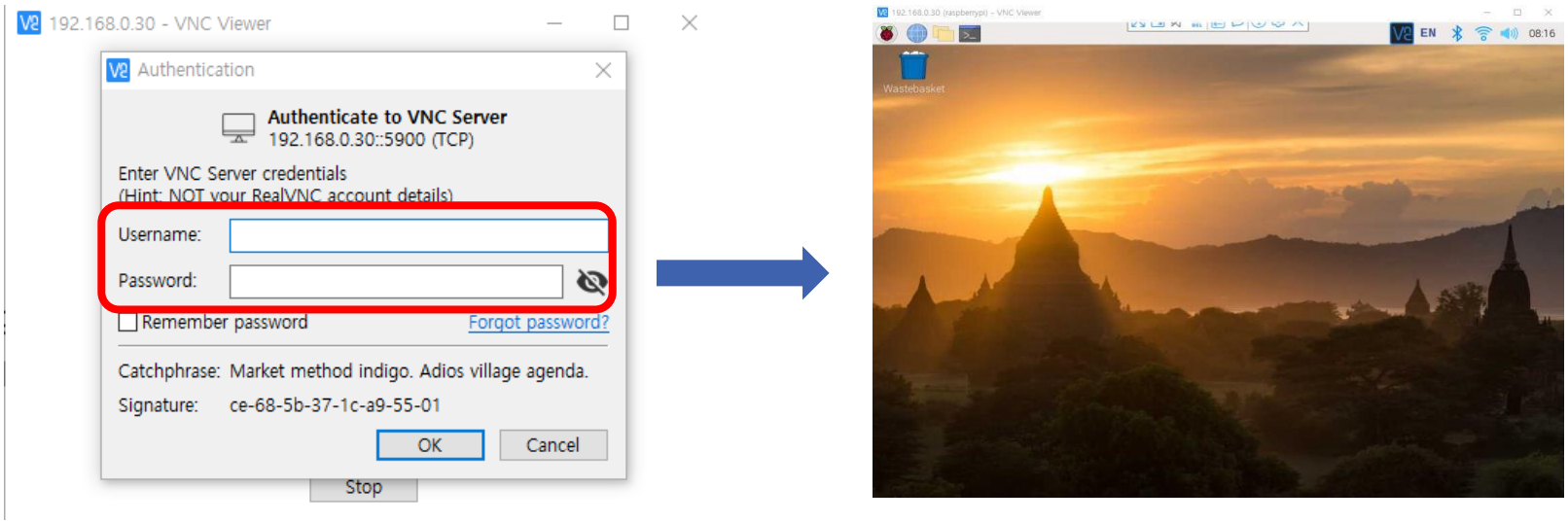
라즈베리 파이 설정

- **실습환경**
  - ✓ 터치 스크린을 이용하여 마우스 커서를 Wifi 아이콘에 올려놓는다.
  - ✓ 얻은 IP주소를 이용하여 컴퓨터의 VNC에 접속한다. VNC 접속방법은 다음과 같다.
    - VNC앱을 열고 상단의 입력위치에 IP주소를 입력한 후 Enter를 누른다.



라즈베리 파이 설정

- **실습환경**
  - ✓ Username: pi, Password: raspberry
  - ✓ 위의 과정을 마치면 라즈베리 파이의 화면을 개인 PC에서 확인할 수 있다.



차선인식(영상처리)

- **Gray Scale 변환**
  - ✓ RGB의 3채널 이미지에서 Gray Scale의 1채널 이미지로 변환
  - ✓ 연산량을 줄이기 위한 목적

```
def gray_scale(self, img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

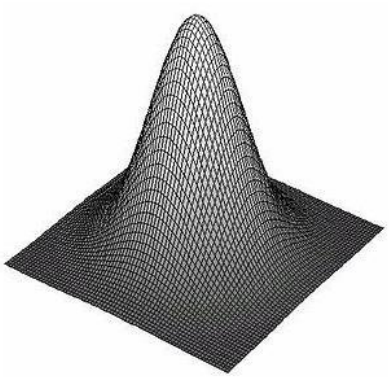
차선인식(영상처리)

- **노이즈 제거**

  ✓ 노이즈 제거를 위한 가우시안 블러링, 메디안 필터링 적용

```
def noise_removal(self, img):
    img = cv2.GaussianBlur(img, (5, 5), 0)
    return img
```

- **가우시안 블러링**
  - ✓ 노이즈를 최소화 시키는 가우시안 블러링 적용
  - ✓ 중심에 있는 픽셀에 높은 가중치를 부여하여 전체 영상에 마스크를 적용



$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

마스크

차선인식(영상처리)

- 엣지 검출

```
def edge_detection(self, img):
    img = cv2.Canny(img, 100, 250)
    return img  # 이미지 반환
```

차선인식(영상처리)

- **엣지 검출**
  - ✓ Canny Edge알고리즘 사용
  - ✓ 차선과 도로 사이에는 밝기 변화가 존재, 밝기가 급격하게 변하는 곳의 경사 값이 큰 특성을 이용하여 이러한 픽셀들을 따라서 그리면 Edge가 검출됨

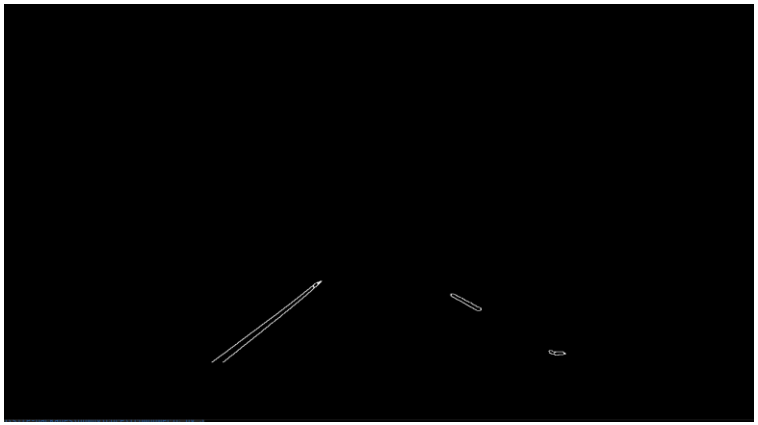
- **Canny Edge 알고리즘 순서**
  - ✓ 가우시안 필터링
  - ✓ Sobel 마스크를 사용하여 기울기 벡터의 크기를 계산
  - ✓ 기울기 벡터 방향에서 기울기 크기가 최대값인 화소만 남기고 나머지는 0으로 억제
  - ✓ 연결된 엣지를 얻기 위해 두개의 임계 값을 사용, 높은 값의 임계 값을 사용하여 기울기 방향에서 낮은 값의 임계 값이 나올 때까지 추적

차선인식(영상처리)

- 관심영역(Region Of Interest)

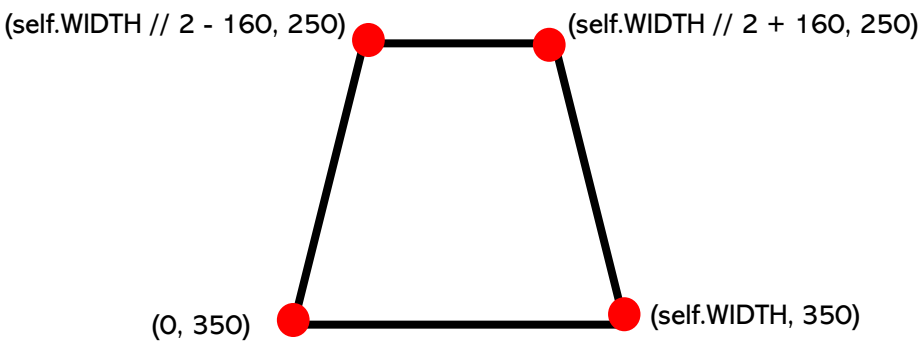
```
self.__vertices = np.array([(0, 350), (self.WIDTH // 2 - 160, 250),
                            (self.WIDTH // 2 + 160, 250), (self.WIDTH, 350)], dtype=np.int32)
```

```
def roi(self, img):
    mask = np.zeros_like(img
    cv2.fillPoly(mask, [self.__vertices], 255)
    masked_img = cv2.bitwise_and(img, mask)
    return masked_img
```



차선인식(영상처리)

- **관심영역(Region Of Interest)**
  - ✓ 영상 내에서 관심이 있는 영역을 의미함
  - ✓ 특정 오브젝트, 특이점을 찾는 것을 목표로 할 때 주로 사용되며 연산의 부하를 줄이고 차선 영역만을 관심영역으로 설정
  - ✓ 영상 크기의 마스크를 만들고 관심영역으로 된 마스크의 픽셀 값을 1로 채워 AND 연산하면 나머지 비 관심영역은 모두 0이 됨
  - ✓ 위의 코드에서는 사다리꼴 형태로 ROI를 설정

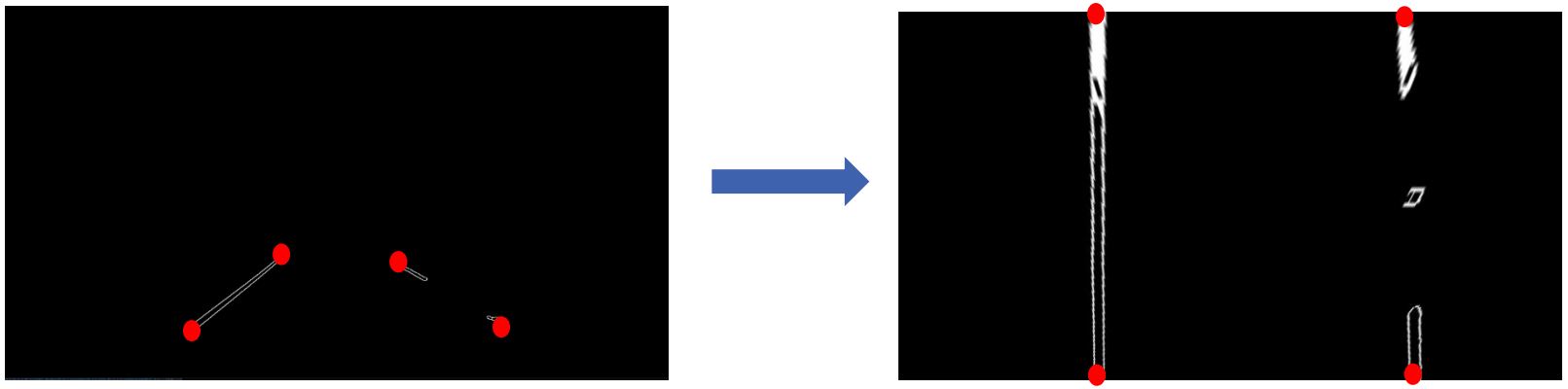


차선인식(영상처리)

- Bird Eye View(Perspective Transform)

```
# Bird's eye View 변환을 위한 src, dst point 설정 (src 좌표에서 dst 좌표로 투시 변환)
self.points_src = np.float32(list(self.__vertices))
self.points_dst = np.float32(
    [(100, self.HEIGHT), (100, 0), (self.WIDTH - 100, 0), (self.WIDTH - 100, self.HEIGHT)])

# 만든 src, dst point를 이용하여 투시 변환 행렬 생성
self.transform_matrix = cv2.getPerspectiveTransform(self.points_src, self.points_dst)
```

```
def perspective_transform(self, img):
    result_img = cv2.warpPerspective(img, self.transform_matrix, (self.WIDTH, self.HEIGHT))
    return result_img
```



차선인식(영상처리)

- **Bird Eye View(Perspective Transform)**
  - ✓ 멀리 있는 것은 작게 보이고, 가까이 있는 것은 크게 보이는 원근감은 카메라의 렌즈의 의한 왜곡 현상으로 인해 발생
  - ✓ 원근 변환(Perspective Transform)은 원근감을 표현하기 위한 변환을 의미함
  - ✓ 반대로, 실생활에서 적용되는 원근 변환은 원근 이미지를 평면 이미지로 변경하기 위해 적용 됨. 따라서, 원근감에 의한 왜곡을 최소화 하기 위해 원근 변환을 적용할 수 있음

- **Bird Eye View(Perspective Transform)**

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & b_0 \\ a_{10} & a_{11} & b_1 \\ a_{20} & a_{21} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00}x_1 + a_{01}y_1 + b_0 \\ a_{10}x_1 + a_{11}y_1 + b_1 \\ a_{20}x_1 + a_{21}y_1 + 1 \end{bmatrix}$$

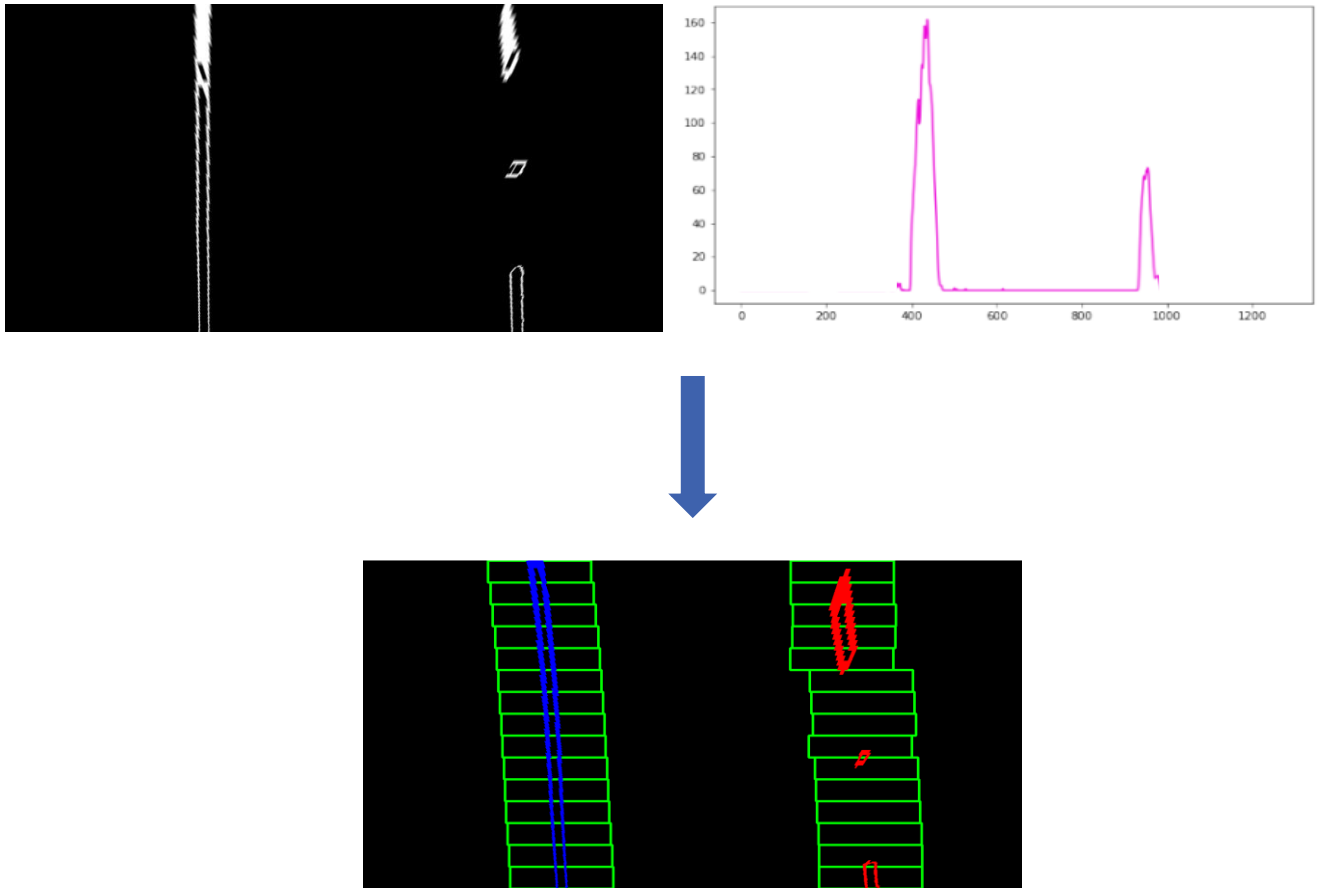
✓ 행렬의 x1, y1은 변환 전 원본 이미지의 픽셀 좌표

✓ X2, y2는 변환 후의 결과 이미지의 픽셀 좌표를 의미함

✓ 변환 후의 픽셀 좌표를 계산하기 위해서는 미지수 a00, a01, a10, a11, a20, a21, b0, b1을 알아야 하며 이 여덟개의 미지수를 구하기 위해 4개의 좌표를 활용해 미지수를 계산

차선인식(영상처리)

- **Sliding Window**
  - ✓ 고정 사이즈의 윈도우가 255의 픽셀 값을 가지는 위치를 기준으로 평균을 내어 차례대로 다음 조사 창의 위치를 결정하는 차선 인식 알고리즘



차선인식(영상처리)

- **Sliding Window**
  - ✓ 곡선 차선인식의 핵심 파트
  - ✓ 데이터에 대한 히스토그램으로 조사창의 시작점을 정한 뒤, 필요한 조사창의 개수만큼 반복문을 돌려가며 조사창의 위치를 재설정 및 탐색
  - ✓ 차선으로 인식되는 부분의 데이터를 따로 저장하여, 반복문을 빠져나온 뒤 모아 놓은 데이터를 토대로 방정식을 계산

차선인식(영상처리)

- Sliding Window

```
def sliding_window(self, img, nwindows=10, margin=60, minpix=50, draw_windows=True):  # sliding window
    left_fit_ = np.empty(3)
    right_fit_ = np.empty(3)
    out_img = np.dstack((img, img, img)) * 255

    # 슬라이딩 윈도우를 출력하기 위한 가로로 긴 이미지를 추가로 만듦
    # 차선이 ROI 영역 밖으로 넘어가며 슬라이딩 윈도우도 함께 잘려서 출력되는 것을 방지하고자 함
    window_img = np.zeros((self.HEIGHT, self.WIDTH + 2 * self.window_margin, 3))
    # 인식된 차선 이미지를 담고 있는 out_img를 window_img의 중앙에 동일하게 가져옴
    window_img[:, self.window_margin:self.window_margin + self.WIDTH, :] = out_img

    midpoint = self.WIDTH // 2

    # 조사창의 높이 설정
    # 전체 높이에서 설정한 조사창 개수만큼 나눈 값
    window_height = self.HEIGHT // nwindows

    # 0이 아닌 픽셀의 x,y 좌표 반환 (흰색 영역(차선)으로 인식할 수 있는 좌표)
    nonzero = img.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # 양쪽 차선 확인 위치 업데이트
    leftx_current = self.leftx_base
    rightx_current = self.rightx_base

    # 차선 확인 시 검출이 되지 않는 경우를 대비해서 바로 이전(화면상 바로 아래) 조사창으로부터 정보를 얻기 위한 변수
    # 이를 이용해서 조사창 위치의 변화량을 파악
    leftx_past = leftx_current
    rightx_past = rightx_current
    # 양쪽 차선 픽셀 인덱스를 담기 위한 빈 배열 선언
    left_lane_inds = []
    right_lane_inds = []
```

차선인식(영상처리)

- **Sliding Window**

```
# 설정한 조사창 개수만큼 슬라이딩 윈도우 생성
for window in range(nwindows):
    # 조사창 크기 및 위치 설정
    win_y_low = self.HEIGHT - ((window + 1) * window_height)
    # n번째 조사창 윗변 y 좌표 : (전체 높이) - (n * 조사창 높이)
    win_y_high = self.HEIGHT - (window * window_height)
    # 양쪽 차선의 조사창의 너비를 현재 좌표로부터 margin만큼 양 옆으로 키움
    win_xleft_low = leftx_current - margin
    win_xleft_high = leftx_current + margin
    win_xright_low = rightx_current - margin
    win_xright_high = rightx_current + margin

    if draw_windows == True:
        # 왼쪽 차선
        cv2.rectangle(out_img, (win_xleft_low, win_y_low), (win_xleft_high, win_y_high), (0, 255, 0), 3)
        # 오른쪽 차선
        cv2.rectangle(out_img, (win_xright_low, win_y_low), (win_xright_high, win_y_high), (0, 255, 0), 3)

        # 양쪽으로 각각 window_margin 만큼 넓힌 이미지이므로 x좌표를 window_margin 만큼 이동시켜 그려주어야 함
        # 왼쪽 차선
        cv2.rectangle(window_img, (win_xleft_low + self.window_margin, win_y_low),
                      (win_xleft_high + self.window_margin, win_y_high), (255, 100, 100), 1)
        # 오른쪽 차선
        cv2.rectangle(window_img, (win_xright_low + self.window_margin, win_y_low),
                      (win_xright_high + self.window_margin, win_y_high), (255, 100, 100), 1)

    # 조사창 내부에서 0이 아닌 픽셀의 인덱스 저장
    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high)
                      & (nonzerox >= win_xleft_low) & (nonzerox < win_xleft_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high)
                       & (nonzerox >= win_xright_low) & (nonzerox < win_xright_high)).nonzero()[0]
```

차선인식(영상처리)

- **Sliding Window**

```
# 양쪽 차선의 인덱스 저장
left_lane_inds.append(good_left_inds)
right_lane_inds.append(good_right_inds)

# 조사창 내부에서 0이 아닌 픽셀 개수가 기준치를 넘으면 해당 픽셀들의 인덱스 평균값(x좌표 평균)으로 다음 조사창의 위치(x좌표)를 결정
if len(good_left_inds) > minpix:
    np.nan_to_num(nonzerox[good_left_inds])
    leftx_current = int(np.mean(nonzerox[good_left_inds]))
if len(good_right_inds) > minpix:
    np.nan_to_num(nonzerox[good_right_inds])
    rightx_current = int(np.mean(nonzerox[good_right_inds]))
# 양쪽 차선 중 하나만 인식된 경우 반대편 차선에서 나타난 인덱스 변화량과 동일하게 인덱스 설정
if len(good_left_inds) < minpix:
    leftx_current = leftx_current + (rightx_current - rightx_past)
if len(good_right_inds) < minpix:
    rightx_current = rightx_current + (leftx_current - leftx_past)
# 가장 하단에 있는 첫번째 조사창에서 결정된 두번째 조사창의 좌표를 다음 프레임의 기준점으로 결정
if window == 0:
    # 왼쪽 차선의 기준점이 중앙 기준 우측으로 넘어가지 않도록 제한
    if leftx_current > midpoint + 40:
        leftx_current = midpoint + 40
    # 왼쪽 차선의 기준점이 왼쪽 화면 밖으로 나가지 않도록 제한
    if leftx_current < 0:
        leftx_current = 0
    # 오른쪽 차선의 기준점이 중앙 기준 좌측으로 넘어가지 않도록 제한
    if rightx_current < midpoint - 40:
        rightx_current = midpoint - 40
    # 오른쪽 차선의 기준점이 오른쪽 화면 밖으로 나가지 않도록 제한
    if rightx_current > self.WIDTH:
        rightx_current = self.WIDTH
    # 두번째 조사창의 현재 좌표를 다음 프레임의 기준점으로 설정
    self.leftx_base = leftx_current
    self.rightx_base = rightx_current
# 현재 인덱스 값을 이전 값으로 저장
leftx_past = leftx_current
rightx_past = rightx_current
```

차선인식(영상처리)

- Sliding Window

```
left_lane_inds = np.concatenate(left_lane_inds)
right_lane_inds = np.concatenate(right_lane_inds)

# 양쪽 차선 픽셀 추출
leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]

rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]

# 차선으로 인식된 픽셀 수가 일정치 이상일 경우에만 차선이 인식된 것으로 판단
# 왼쪽 차선으로 인식된 좌표가 5000개 미만이라면 False, 이상이라면 True
if leftx.size < 500:
    left_lane_detected = False
else:
    left_lane_detected = True
# 오른쪽 차선으로 인식된 좌표가 5000개 미만이라면 False, 이상이라면 True
if rightx.size < 500:
    right_lane_detected = False
else:
    right_lane_detected = True

# 차선이 인식된 것으로 판단되었다면 검출된 좌표로부터 차선의 2차 곡선을 구함
# 왼쪽 차선이 인식된 경우
if left_lane_detected:
    # 검출된 차선 좌표들을 통해 왼쪽 차선의 2차 방정식 계수를 구함
    left_fit = np.polyfit(lefty, leftx, 2)
    np.nan_to_num(left_fit)

    # 왼쪽 차선 계수
    self.left_a.append(left_fit[0])
    self.left_b.append(left_fit[1])
    self.left_c.append(left_fit[2])

# 오른쪽 차선이 인식된 경우
if right_lane_detected:
    # 검출된 차선 좌표들을 통해 오른쪽 차선의 2차 방정식 계수를 구함
    right_fit = np.polyfit(righty, rightx, 2)
    np.nan_to_num(right_fit)

    # 오른쪽 차선 계수
    self.right_a.append(right_fit[0])
    self.right_b.append(right_fit[1])
    self.right_c.append(right_fit[2])
```

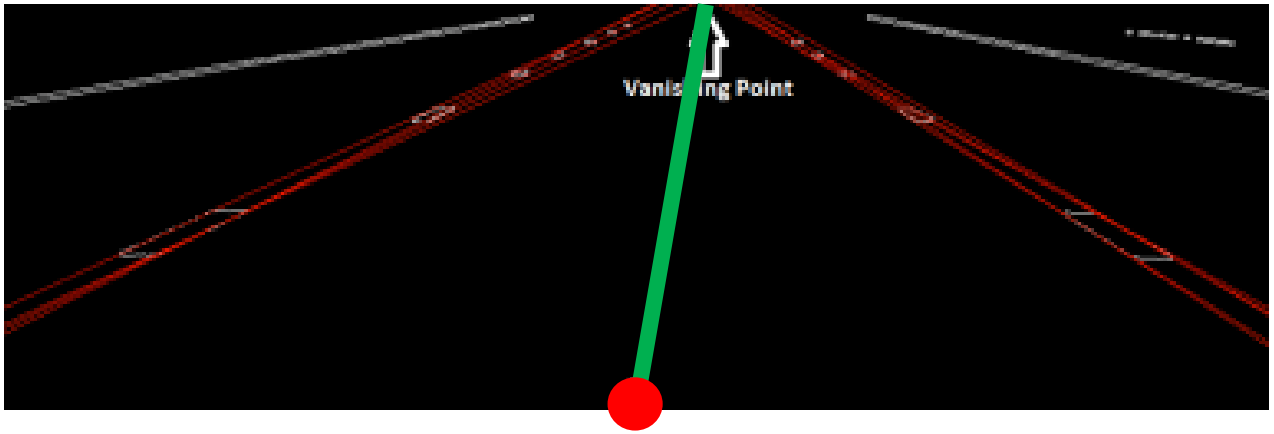
차선인식(영상처리)

- **Sliding Window**

```
# 차선으로 검출된 픽셀 값 변경
# 왼쪽 차선은 파란색, 오른쪽 차선은 빨간색으로 표시
out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]
# window_img : 양쪽으로 각각 window_margin 만큼 넓힌 이미지이므로 x좌표를 window_margin 만큼 이동시켜 그려주어야 함
window_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds] + self.window_margin] = [255, 0, 0]
window_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds] + self.window_margin] = [0, 0, 255]

# 계수마다 각각 마지막 10개의 평균으로 최종 계수 결정
# 왼쪽 차선의 계수 결정
left_fit_[0] = np.mean(self.left_a[-10:])
left_fit_[1] = np.mean(self.left_b[-10:])
left_fit_[2] = np.mean(self.left_c[-10:])

# 오른쪽 차선의 계수 결정
right_fit_[0] = np.mean(self.right_a[-10:])
right_fit_[1] = np.mean(self.right_b[-10:])
right_fit_[2] = np.mean(self.right_c[-10:])

# y 값에 해당하는 x 값 결정
# 왼쪽 차선
left_fitx = left_fit_[0] * self.ploty ** 2 + left_fit_[1] * self.ploty + left_fit_[2]

# 오른쪽 차선
right_fitx = right_fit_[0] * self.ploty ** 2 + right_fit_[1] * self.ploty + right_fit_[2]

# 양쪽 모두 차선인식이 안됐다면 슬라이딩 윈도우 조사창 재설정
if (left_lane_detected is False) and (right_lane_detected is False):
    self.leftx_base = self.leftx_mid - 30
    self.rightx_base = self.rightx_mid + 30

# 출력 이미지, 양쪽 곡선 x 좌표, 차선 인식 여부 반환
return out_img, left_fitx, right_fitx, left_lane_detected, right_lane_detected
```

차선인식(영상처리)

- **조향각 계산**
  - ✓ 소실점(Vanishing Point)는 양쪽 차선의 방정식의 교점
  - ✓ 카메라 중앙 하단점과 소실점과의 각도 차이를 이용하여 차량의 조향각을 결정



차선인식(영상처리)

- 조향각 계산

```
def get_angle_on_lane(self, left_fitx, right_fitx, left_lane_detected, right_lane_detected):
    # 두 차선 인식 모두 안되는 경우 이전 조향각 반환
    if (left_lane_detected is False) and (right_lane_detected is False):
        return self.prev_angle

    # 양쪽 차선 중 하나만 인식된 경우
    elif (left_lane_detected is False) or (right_lane_detected is False):
        # 왼쪽 차선이 인식된 경우 왼쪽 차선의 기울기를 그대로 적용
        if left_lane_detected is True:
            left_lane = list(np.polyfit(left_fitx, self.ploty, deg=1))  # 왼쪽 차선 좌표로부터 직선 방정식의 계수와 절편을 구함
            slope = left_lane[0]  # 나온 기울기 값 저장

        # 오른쪽 차선이 인식된 경우 오른쪽 차선의 기울기를 그대로 적용
        else:
            right_lane = list(np.polyfit(right_fitx, self.ploty, deg=1))  # 오른쪽 차선 좌표로부터 직선 방정식의 계수와 절편을 구함
            slope = right_lane[0]  # 나온 기울기 값 저장
```

차선인식(영상처리)

- 조향각 계산

```
# 양쪽 차선이 모두 인식된 경우
else:
    left_lane = list(np.polyfit(left_fitx, self.ploty, deg=1))  # 왼쪽 차선 직선 방정식의 계수와 절편을 구함
    right_lane = list(np.polyfit(right_fitx, self.ploty, deg=1))  # 오른쪽 차선 직선 방정식의 계수와 절편을 구함

    # 두 차선의 기울기가 평행한 경우 소실점이 존재하지 않음
    # 두 차선의 중앙 선을 그리고, -> y = ax + (b1+b2)/2
    # 카메라 최상단(y=0)에서의 좌표를 구한 후 해당 좌표를 소실점으로 결정
    # x = -(b1+b2)/(2*a), y = 0
    # 두 차선이 벌어진 정도와 카메라 중앙으로부터 치우친 정도를 고려하여 조향각 계산이 가능
    if left_lane[0] == right_lane[0]:
        Inter_X = -(right_lane[1] + left_lane[1]) / (2 * left_lane[0])
        Inter_Y = 0

    else:  # 두 차선의 기울기가 평행하지 않은 경우 소실점 추출
        Inter_X = (right_lane[1] - left_lane[1]) / (left_lane[0] - right_lane[0])  # 두 직선 방정식의 교차 X점 구하기
        Inter_Y = np.poly1d(left_lane)(Inter_X)  # 두 직선 방정식의 교차 Y점 구하기
    slope = (self.HEIGHT - Inter_Y) / ((self.WIDTH / 2) - Inter_X)  # 카메라 하단 중앙점과 차선 상단 중앙점 사이의 기울기 차이를 구하기

steering_angle = math.atan(slope) * 180 / math.pi  # 조향각 추출

# 수직선 기준으로 기준값이 90도로 계산됨, 기준값을 0도로 변환
# 왼쪽 방향 : -20 ~ 0  /  중앙 : 0  /  오른쪽 방향 : 0 ~ +20
if steering_angle > 0:
    steering_angle = steering_angle - 90
    if steering_angle <= -20:
        steering_angle = -20  # 최저 각도 범위를 20도로 제한

elif steering_angle < 0:
    steering_angle = steering_angle + 90
    if steering_angle >= 20:
        steering_angle = 20  # 최대 각도 범위를 -20도로 제한

steering_angle = int(steering_angle)  # 정수형으로 변환
self.prev_angle = steering_angle  # 현재 조향각을 이전 조향각 변수에 저장
return steering_angle  # 조향각 반환
```

차선인식(영상처리)

- **최종 결과**
  - ✓ 조향각 범위
    - -20: 좌회전 최대
    - 20: 우회전 최대
    - 0: 직진

Serial Communication

2022년 2학기

시리얼 통신

- 라즈베리 파이 터미널에서 연결된 USB포트의 이름을 확인하기 위해 아래의 명령어 입력

```
$ dmesg | grep tty
```

```
pi@raspberrypi:~ $ dmesg | grep tty
[    0.000000] Kernel command line: coherent_pool=1M 8250.nr_uarts=1 cma=64M snd_bcm2835.enabl
e_compat_alsa=0 snd_bcm2835.enable_hdmi=1 snd_bcm2835.enable_headphones=1 cma=256M video=HDMI-
A-1:1280x720M@60,margin_left=48,margin_right=48,margin_top=48,margin_bottom=48 smsc95xx.macadd
r=DC:A6:32:AD:2F:7B vc_mem.mem_base=0x3ec00000 vc_mem.mem_size=0x40000000  console=ttyS0,11520
0 console=tty1 root=PARTUUID=054adc8a-02 rootfstype=ext4 elevator=deadline fsck.repair=yes roo
twait quiet splash plymouth.ignore-serial-consoles
[    0.000267] console [tty1] enabled
[    0.511795] fe201000.serial: ttyAMA0 at MMIO 0xfe201000 (irq = 34, base_baud = 0) is a PL01
1 rev2
[    0.517030] console [ttyS0] disabled
[    0.517062] fe215040.serial: ttyS0 at MMIO 0x0 (irq = 36, base_baud = 62500000) is a 16550
[    0.517117] console [ttyS0] enabled
[    4.802444] usb 1-1.1: FTDI USB Serial Device converter now attached to ttyUSB0
```

시리얼 통신

```
# ========================================
# 포트 선언
# ========================================
def __init__(self):
    try:
        self.port = serial.Serial(port='/dev/ttyUSB0', baudrate=9600)
        self.port.open()
        if not self.port.is_open():
            raise serial.SerialException

    except serial.SerialException:
        print('Fail to Open Serial Port! Check Port Number.')
        exit()
```

포트 이름 입력

시리얼 통신 쓰기 함수

```
# ==========================================
# 4개의 변수 값을 입력받아 각 모터에 보냄
# left_top_speed: 왼쪽 상단 바퀴 속도 (ex: 055)
# right_top_speed: 오른쪽 상단 바퀴 속도 (ex: 045)
# left_bottom_speed: 왼쪽 하단 바퀴 속도 (ex: 055)
# right_bottom_speed: 오른쪽 하단 바퀴 속도 (ex: 045)
# ==========================================
def serial_write(self, left_top_speed, right_top_speed, left_bottom_speed,
right_bottom_speed):
    packet = bytes("{0:03d}#{1:03d}${2:03d}%{3:03d}^".format(left_top_speed,
right_top_speed, left_bottom_speed, right_bottom_speed)).encode('ascii')
    self.port.write(packet)
    time.sleep(0.001)
```

시리얼 통신 읽기 함수

```
# ========================================
# 시리얼 포트로 부터 값을 읽음
# ========================================
def serial_read(self):
    self.port.readline()
```

시리얼 통신 해제

```
# ========================================
# 시리얼 통신 해제
# ========================================
def serial_clear(self):
    self.port.close()
```

참고 사항

- 시리얼 통신 참고 사이트
  - ✓ https://pyserial.readthedocs.io/en/latest/index.html

- 병렬적으로 프로그램 수행이 필요한 경우 해당 사이트 참고
  - ✓ http://pythonstudy.xyz/python/article/24-%EC%93%B0%EB%A0%88%EB%93%9C-Thread

참고 사항

- 영상처리 속도가 느릴 경우 시각화 함수들을 주석처리
  - ✓ cv2.imshow()
  - ✓ cv2.line()
  - ✓ cv2.rectangle()
  - ✓ cv2.addWeighted()
- 카메라에 맞게 ROI 좌표를 수정
- 조향각이 급격하게 변화할 경우 필터 알고리즘 적용 고려