## 차선 추출(Yellow, White 차선 추출)

| 1. HSV에서 Yellow를 추출한다.  - BGR에서 HSV로 변환  - inRange함수를 사용하여 Yellow영역을 지정  - bitwise\_and를 사용하여 원래이미지에 적용  2. RGB에서 White를 추출한다.  - inRange함수를 사용하여 White영역을 지정  - bitwise\_and를 사용하여 원래이미지에 적용  3. 병합한다.(cv2.addWeighted(white\_image, 1., yellow\_image, 1., 0) |
| --- |

비디오 읽어오는 코드

| import cv2  capture = cv2.VideoCapture('images/challenge.mp4')  if capture.isOpened() == False:  print("카메라를 열 수 없습니다.")  exit(1)  while True:  ret, img\_src = capture.read()  img\_dst = img\_src.copy()  # if ret == False: # 동영상 끝까지 재생  # print("동영상 읽기 완료")  # break  # 동영상이 끝나면 재생되는 프레임의 위치를 0으로 다시 지정  if capture.get(cv2.CAP\_PROP\_POS\_FRAMES) == capture.get(cv2.CAP\_PROP\_FRAME\_COUNT):  capture.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  cv2.imshow('Video-dst', img\_dst)  key = cv2.waitKey(25) # 33ms마다  if key == 27: # Esc 키  break  capture.release()  cv2.destroyAllWindows() |
| --- |

HSV에서 Yellow추출

| import cv2  img\_src = cv2.imread('images/tomato.jpg', cv2.IMREAD\_COLOR)  # 이미지를 BGR에서 HSV로 색변환  img\_hsv = cv2.cvtColor(img\_src, cv2.COLOR\_BGR2HSV) # HSV  # HSV로 노란색 정보를 좀 더 구체적으로 표시  lower\_yellow = (15,100,100) # 자료형은 튜플형태로(H,S,V)  upper\_yellow = (40,255,255) # 자료형은 튜플형태로(H,S,V)  yellow\_mask = cv2.inRange(img\_hsv, lower\_yellow, upper\_yellow) # 초록 정보 추출  img\_yellow = cv2.bitwise\_and(img\_src, img\_src, mask=yellow\_mask)  cv2.imshow('yellow',img\_yellow)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

BGR에서 White추출

| import cv2  img\_src = cv2.imread('images/tomato.jpg', cv2.IMREAD\_COLOR)  # BGR에서 white를 좀 더 구체적으로 표시  lower\_white = (150,150,150) # 자료형은 튜플형태로(B,G,R)  upper\_white = (255,255,255) # 자료형은 튜플형태로(B,G,R)  white\_mask = cv2.inRange(img\_src, lower\_white, upper\_white) # 초록 정보 추출  img\_white = cv2.bitwise\_and(img\_src, img\_src, mask=white\_mask)  cv2.imshow('white',img\_white)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

합치기

| img\_dst=cv2.addWeighted(img\_white, 1., img\_yellow, 1., 0) |
| --- |

폴리곤 그리기

| import cv2  import numpy as np  trap\_bottom\_width = 0.8  trap\_top\_width = 0.1  trap\_height = 0.4  img\_bgr = cv2.imread('images/lane.png', cv2.IMREAD\_COLOR)  img\_mask = np.zeros\_like(img\_bgr)  height, width = img\_mask.shape[:2]  mask\_color = (255, 255, 255)  pts = np.array([[  ((width \* (1-trap\_bottom\_width))//2, height),  ((width\*(1-trap\_top\_width))//2, (1-trap\_height)\*height),  (width-(width\*(1-trap\_top\_width))//2, (1-trap\_height)\*height),  (width-(width \* (1-trap\_bottom\_width))//2, height)]],  dtype=np.int32)  cv2.fillPoly(img\_mask, pts, mask\_color)  cv2.imshow("src", img\_mask)  cv2.waitKey()  cv2.destroyAllWindows() |
| --- |

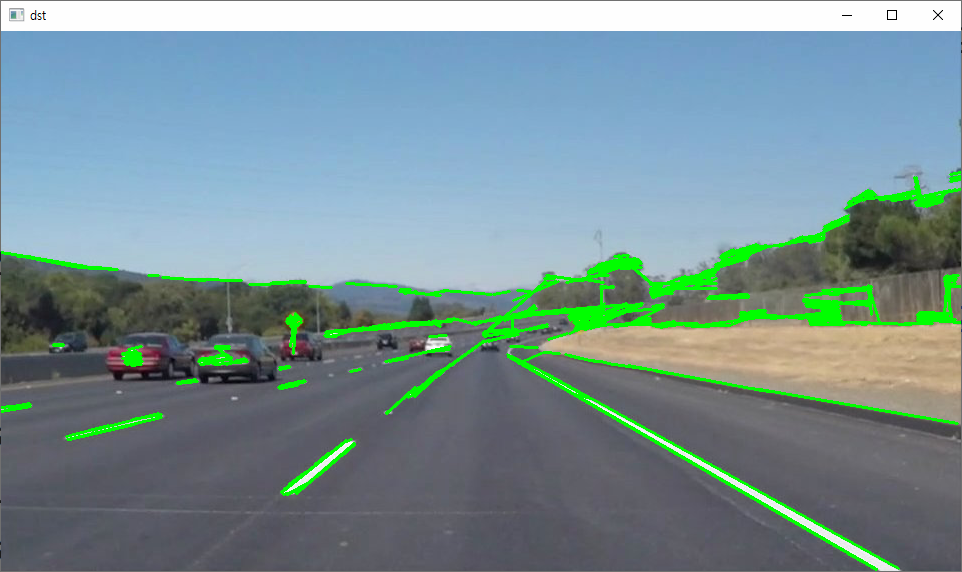
원본 이미지에 적용

| img\_src = cv2.bitwise\_and(img\_src, img\_mask) |
| --- |

기존의 방법들을 조합하여 차선 추출하기

| capture=cv2.VideoCapture("videos/challenge.mp4")  while(cv2.waitKey(33)<0):  ret, frame = capture.read()  #1. 흰색 노란색 차선을 추출 : 함수 filter\_colors()  frame\_crop = filter\_colors(frame)  #2. 그레이로 변환  frame\_gray = cv2.cvtColor(frame\_crop,cv2.COLOR\_BGR2GRAY)  #3. 블러(흐림)을 사용해서 노이즈 제거  frame\_gray = cv2.GaussianBlur(frame\_gray, (5,5), 0) #마스크 크기가 커지면 더 흐려짐  #4. 임계값(Threshold)를 사용하여 이진화  \_, frame\_binary = cv2.threshold(frame\_gray,0, 255,cv2.THRESH\_OTSU+cv2.THRESH\_BINARY)  #5. 외각선(에지) 구하기 : Canny 에지를 사용  frame\_canny = cv2.Canny(frame\_binary,50,150)  #6. fillPoly를 사용하여 원하는 사다리꼴을 생성 : roi  # 케니에지 외각선 영상에 원하는 영역(사다리꼴) 적용하기  # 사다리꼴 안에 있는 외각선만 남은 상황  # 함수를 생성 region\_of\_interst()  frame\_roi = region\_of\_interst(frame\_canny)  #7. HoughLinesP 함수를 사용하여 외각선에서 선 긋기  capture.release()  cv2.destroyAllWindows() |
| --- |

| import numpy as np  import cv2  img\_src = cv2.imread("images/lane.png")  # 노란색, 흰색 차선 추출  img\_gray = cv2.cvtColor(img\_src, cv2.COLOR\_BGR2GRAY)  # 이미지 블러  img\_gray = cv2.GaussianBlur(img\_gray,(5,5), 0)  # 이진화 수행  \_, img\_binary = cv2.threshold(img\_gray,0,255,cv2.THRESH\_OTSU+cv2.THRESH\_BINARY)  img\_canny = cv2.Canny(img\_binary, 50, 150)  #사다리꼴 적용하기  img\_roi = img\_canny  rho = 2  theta = 1 \* np.pi / 180  threshold = 15  min\_line\_length = 10  max\_line\_gap = 20  lines = cv2.HoughLinesP(img\_roi, rho, theta, threshold,  minLineLength = min\_line\_length,  maxLineGap = max\_line\_gap)  for i, line in enumerate(lines):  cv2.line(img\_src, (line[0][0], line[0][1]),  (line[0][2], line[0][3]), (0, 255, 0), 2)  cv2.imshow("dst", img\_src)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |



| def draw\_lines(img, lines, color=[255, 0, 0], thickness=12):  # 예외처리  if lines is None:  return  if len(lines) == 0:  return  draw\_right = True  draw\_left = True  # 모든 선의 기울기 찾기  # 기울기의 절대값이 임계값 보다 커야 추출됨  slope\_threshold = 0.5  slopes = []  new\_lines = []  for line in lines:  x1, y1, x2, y2 = line[0] # line = [[x1, y1, x2, y2]]  # 기울기 계산  if x2 - x1 == 0.: # 기울기의 분모가 0일때 예외처리  slope = 999. # practically infinite slope  else:  slope = (y2 - y1) / (x2 - x1)  # 조건을 만족하는 line을 new\_lines에 추가  if abs(slope) > slope\_threshold:  slopes.append(slope)  new\_lines.append(line)  lines = new\_lines  # 라인을 오른쪽과 왼쪽으로 구분하기  # 기울기 및 점의 위치가 영상의 가운데를 기준으로 왼쪽 또는 오른쪽에 위치  right\_lines = []  left\_lines = []  for i, line in enumerate(lines):  x1, y1, x2, y2 = line[0]  img\_x\_center = img.shape[1] / 2 # 영상의 가운데 x 좌표  #기울기 방향이 바뀐이유는 y축의 방향이 아래로 내려오기 때문  if slopes[i] > 0 and x1 > img\_x\_center and x2 > img\_x\_center:  right\_lines.append(line)  elif slopes[i] < 0 and x1 < img\_x\_center and x2 < img\_x\_center:  left\_lines.append(line)  # np.polyfit()함수를 사용하기 위해 점들을 추출  # Right lane lines  right\_lines\_x = []  right\_lines\_y = []  for line in right\_lines:  x1, y1, x2, y2 = line[0]  right\_lines\_x.append(x1)  right\_lines\_x.append(x2)  right\_lines\_y.append(y1)  right\_lines\_y.append(y2)  if len(right\_lines\_x) > 0:  right\_m, right\_b = np.polyfit(right\_lines\_x, right\_lines\_y, 1) # y = m\*x + b  else:  right\_m, right\_b = 1, 1  draw\_right = False  # Left lane lines  left\_lines\_x = []  left\_lines\_y = []  for line in left\_lines:  x1, y1, x2, y2 = line[0]  left\_lines\_x.append(x1)  left\_lines\_x.append(x2)  left\_lines\_y.append(y1)  left\_lines\_y.append(y2)  if len(left\_lines\_x) > 0:  left\_m, left\_b = np.polyfit(left\_lines\_x, left\_lines\_y, 1) # y = m\*x + b  else:  left\_m, left\_b = 1, 1  draw\_left = False  # 왼쪽과 오른쪽의 2개의 점을 찾기, y는 알고 있으므로 x만 찾으면됨  # y = m\*x + b --> x = (y - b)/m  y1 = img.shape[0]  y2 = img.shape[0] \* (1 - trap\_height)  right\_x1 = (y1 - right\_b) / right\_m  right\_x2 = (y2 - right\_b) / right\_m  left\_x1 = (y1 - left\_b) / left\_m  left\_x2 = (y2 - left\_b) / left\_m  # 모든 점은 정수형이어야 함(정수형으로 바꾸기)  y1 = int(y1)  y2 = int(y2)  right\_x1 = int(right\_x1)  right\_x2 = int(right\_x2)  left\_x1 = int(left\_x1)  left\_x2 = int(left\_x2)  # 차선그리기  if draw\_right:  cv2.line(img, (right\_x1, y1), (right\_x2, y2), color, thickness)  if draw\_left:  cv2.line(img, (left\_x1, y1), (left\_x2, y2), color, thickness) |
| --- |