blob : 사용자가 제시한 조건에 맞는 파일명을 리스트 형식으로 반환

* 키보드를 누르면 다음파일이 재생되도록 하는 코드
* Esc를 누르면 이전 그림으로 이동
* 0일 경우 처음 그림이 그대로 있음
* 마지막 그림 다음은 종료

| import cv2  import numpy as np  import glob  images = glob.glob('images/\*.jpg')  total\_images = len(images)  idx = 0  while True:  fname = images[idx]  img = cv2.imread(fname)  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  out\_str = f'{idx}/{total\_images}'  cv2.putText(gray,out\_str,(10,25),  cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, 255, 1)  cv2.imshow('dst' ,gray)  key = cv2.waitKey(0)  if key == 27: #Esc  idx -= 1  else:  idx += 1  if idx < 0:  idx = 0  cv2.destroyAllWindows() |
| --- |

캘리브레이션 영상을 가지고 정보 구하여 dump로 저장하기

| import cv2  import numpy as np  import glob  import pickle  # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)  objp = np.zeros((6\*9,3), np.float32)  objp[:,:2] = np.mgrid[0:9, 0:6].T.reshape(-1,2)  # Arrays to store object points and image points from all the images.  objpoints = [] # 3d points in real world space  imgpoints = [] # 2d points in image plane.  # Make a list of calibration images  # 사용자가 제시한 조건에 맞는 파일명을 리스트 형식으로 반환  images = glob.glob('camera\_cal/\*.jpg')  # Step through the list and search for chessboard corners  total\_images = len(images)  for idx, fname in enumerate(images):  img = cv2.imread(fname)  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  # Find the chessboard corners  ret, corners = cv2.findChessboardCorners(gray, (9,6), None)  # If found, add object points, image points  if ret == True:  objpoints.append(objp)  imgpoints.append(corners)  # Draw and display the corners  cv2.drawChessboardCorners(img, (9,6), corners, ret)  write\_name = 'camera\_cal/result/corners\_found'+str(idx)+'.jpg'  cv2.imwrite(write\_name, img)  out\_str = f'{idx}/{total\_images}'  cv2.putText(img, out\_str, (10, 25),  cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0,255,255), 1)  cv2.imshow('img', img)  cv2.waitKey(500)  cv2.destroyAllWindows()  img = cv2.imread('camera\_cal/test\_cal.jpg')  height, width = img.shape[:2]  img\_size = (width, height)  # Do camera calibration given object points and image points  ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,  img\_size, None,None)  dst = cv2.undistort(img, mtx, dist, None, mtx)  cv2.imwrite('camera\_cal/result/test\_undist.jpg',dst)  # Save the camera calibration result for later use (we won't worry about rvecs / tvecs)  dist\_pickle = {}  dist\_pickle["mtx"] = mtx  dist\_pickle["dist"] = dist  pickle.dump( dist\_pickle, open( "camera\_cal/wide\_dist\_pickle.p", "wb" ) )  img\_result = cv2.hconcat([img,dst])  img\_result = cv2.pyrDown(img\_result)  cv2.imshow('dst',img\_result)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

저장된 피클 데이터를 읽어와서 왜곡 보정하기(undistort)

| import cv2  import pickle  with open('camera\_cal/wide\_dist\_pickle.p', mode='rb') as f:  file = pickle.load(f)  mtx = file['mtx']  dist = file['dist']  img = cv2.imread('camera\_cal/test\_cal.jpg')  dst = cv2.undistort(img, mtx, dist, None, mtx)  img\_result = cv2.hconcat([img,dst])  img\_result = cv2.pyrDown(img\_result)  cv2.imshow('dst',img\_result)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

| import cv2  import pickle  def undistort(img, cal\_dir='camera\_cal/wide\_dist\_pickle.p'):  with open(cal\_dir, mode='rb') as f:  file = pickle.load(f)  mtx = file['mtx']  dist = file['dist']  dst = cv2.undistort(img, mtx, dist, None, mtx)  return dst  img\_src = cv2.imread('images/lane/test1.jpg',cv2.IMREAD\_COLOR)  img\_undist = undistort(img\_src)  cv2.imshow('dst',img\_undist)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

| import cv2  import numpy as np  import pickle  def undistort(img, cal\_dir='camera\_cal/wide\_dist\_pickle.p'):  with open(cal\_dir, mode='rb') as f:  file = pickle.load(f)  mtx = file['mtx']  dist = file['dist']  dst = cv2.undistort(img, mtx, dist, None, mtx)  return dst  img\_src = cv2.imread('images/lane/test1.jpg',cv2.IMREAD\_COLOR)  # 왜곡 보정  img\_undist = undistort(img\_src)  # BGR -> HSL 색좌표계로 변환  img\_hls = cv2.cvtColor(img\_undist, cv2.COLOR\_BGR2HLS)  img\_hls\_h, img\_hls\_l, img\_hls\_s = cv2.split(img\_hls)  #소벨 필터 적용  img\_sobel\_x = cv2.Sobel(img\_hls\_l, cv2.CV\_64F, 1,1)  img\_sobel\_x\_abs = abs(img\_sobel\_x)  img\_sobel\_scaled = np.uint8(img\_sobel\_x\_abs\*255/np.max(img\_sobel\_x\_abs))  sx\_threshold = (15,255)  sx\_binary = np.zeros\_like(img\_sobel\_scaled)  sx\_binary[(img\_sobel\_scaled >= sx\_threshold[0]) & \  (img\_sobel\_scaled <= sx\_threshold[1])] = 255  s\_threshold = (100,255)  s\_binary = np.zeros\_like(img\_hls\_s)  s\_binary[(img\_hls\_s >= s\_threshold[0]) &(img\_hls\_s <= s\_threshold[1])] = 255  img\_binary\_added = cv2.addWeighted(sx\_binary,1.0, s\_binary, 1.0, 0)  cv2.imshow('dst',img\_binary\_added)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |

Perspective

| import cv2  import numpy as np  img\_src = cv2.imread('images/lane/test1.jpg',cv2.IMREAD\_COLOR)  height, width = img\_src.shape[:2]  dst\_size=(1280,720)  src=np.float32([(0.43,0.65),(0.58,0.65),(0.1,1),(1,1)])  dst=np.float32([(0,0), (1, 0), (0,1), (1,1)])  src = src \* np.float32((width,height))  dst = dst \* np.float32(dst\_size)  M = cv2.getPerspectiveTransform(src, dst)  img\_warp = cv2.warpPerspective(img\_src,M,dst\_size)  cv2.imshow('dst',img\_warp)  cv2.waitKey(0)  cv2.destroyAllWindows() |
| --- |