1.

|  |
| --- |
| 순방향 사상 : 입력 영상의 좌표를 중심으로 목적 영상의 좌표를 계산하여 화소의 위치를 변환하는 방식이다. (홀이나 오버랩 문제발생가능)  역방향 사상 : 목적 영상의 좌표를 중심으로 역변환을 계산하여 해당하는 원본 영상의 좌표를 찾아서 화소 값을 가져오는 방식 |

2.

|  |
| --- |
| 홀(hole) : 입력 영상의 좌표들로 목적 영상의 좌표를 만드는 과정에서 사상되지 않은 화소를 가르킨다.  오버랩(overlap) : 입력 영상의 여러 화소들이 목적 영상의 한 화소로 사상되는 것을 말한다. |

3.

|  |
| --- |
| 순방향, 역방향 사상을 사용하면 홀이나 오버랩 문제가 발생하게 되는데 보간법을 이용하면 이러한 문제들을 해결할 수 있다.  Cv2.INTER\_NEAREST : 이웃 보간법  Cv2.INTER\_LINEAR : 쌍 선형 보간법  Cv2.INTER\_NINEAR\_EXACT : 비트 쌍 선형 보간법  Cv2.INTER\_CUBIC : 바이큐빅 보간법  Cv2.INTER\_AREA : 영역 보간법  Cv2.INTER\_LANCZOS4 : Lanczos 보간법 |

4.

|  |
| --- |
| 최근접 이웃 보간법 : 목적영상을 만드는 과정에서 홀이 되어 화소값을 할당받지 못한 위치에 값을 찾을 때 그 위치에 가장 가깝게 이웃한 입력 영상의 화소 값을 가져오는 방법이다. |

5.

|  |
| --- |
| 양선형 보간법 : 선형 보간을 두 번에 걸쳐서 수행한다.  1. 목적영상의 화소를 역변환하여 가장 가까운 위치에 있는 입력 영상의 4개 화소를 가져온다  2. 4개의 화소를 2개씩 이어 직선을 만든다.  3. 직선상의 목적 영상 화소의 좌표로 중간 위치를 찾고 그 위치의 화소값을 계산한다.  4. 중간 화소값을 잇는 직선을 구성하고 중간 화소값과 거리비율을 바탕으로 직선의 수식을 이용해서 최종 화소값을 계산한다. |

6.

|  |
| --- |
|  |

7.

|  |
| --- |
| cv2.warpAffine(src, M, dsize [, dst [, flags[,borderMode [, borderValue]]]])  입력한 영상을 어파인 변환하여 반환한다.  src : 입력영상  dst : 반환영상  M : 어파인 변환 행렬  dsize : 반환영상의 크기  flags : 보간 방법  borderMode : 경계지정 방법  cv2.getAffineTransform(src, dst)  어파인 변환행렬을 생성한다.  src : 입력영상 좌표 3개  dst : 반환영상 좌표 3개  cv2.getRotationMatrix2D(center, angle, scale)  회전, 크기 변환을 할 수 있는 어파인 행렬을 생성한다.  center : 회전의 중심점  angle : 회전각도  scale : 변경 크기  cv2.invertAffineTransform(M [, iM])  어파인 변환행렬의 역행렬을 반환한다.  M : 어파인 반환 행렬  iM : 어파인 역변환 행렬 |

8.

|  |
| --- |
| 원근법을 영상 좌표계에서 표현하는 것으로서 3차원의 실세계의 좌표 P를 투영 스크린의 2차원좌표로 표현할 수 있도록 변화하는 것을 말한다. |

9.

|  |
| --- |
| cv2.getPerspectiveTransform(src, dst [, solveMethod])  4개의 좌표쌍을 원근 변환 행렬로 바꾼다.  src : 입력영상 4개좌표  dst : 목적영상 4개좌표  borderMode : 경계 지정 방법  cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, boderValue]]]])  영상에 원근변환 적용  Src : 입력 영상  M : 원근 변환 행렬  Dsize : 결과여상의 크기  Dst : 결과 영상  Flags : 보간방법  borderMode : 경계지정 방법  borderValur : 상수 경계일 때, 경계 화소값  cv2.transform(src, M)  입력 좌표행렬에 원근변환을 수행한 결과를 반환한다.  src : 입력 좌표 행렬  M : 원근 변환 행렬  dst : 결과 좌표 행렬 |

10.

|  |
| --- |
| import numpy as np, cv2  def contain(p, shape):  return 0 <= p[0] < shape[0] and 0 <= p[1] < shape[1]  def translate(img, pt):  dst = np.zeros(img.shape, img.dtype)  for i in range(img.shape[0]):  for j in range(img.shape[1]):  x, y = np.subtract((j, i), pt)  if contain((y, x), img.shape):  dst[i, j] = img[y, x]  return dst  image = cv2.imread("c:/computervision/chap08/image/1.jpg")  dst1 = translate(image, (50, 60))  M = np.float64([[1, 0, 50], [0, 1, 60]]) dst2 = cv2.warpAffine(image, M, (image.shape[0], image.shape[1]))  cv2.imshow('1', dst1) cv2.imshow('2', dst2) cv2.waitKey() |

11-1.

|  |
| --- |
| import numpy as np, cv2  def contain(p, shape):  return 0<=p[0] < shape[0] and 0<=p[1]<shape[1]  def billinear\_value(img, pt):  x, y = np.int32(pt)  if x>= img.shape[1]-1:x=x-1  if y>= img.shape[0]-1:y=y-1   P1, P2, P3, P4 = np.float32(img[y:y+2, x:x+2].flatten())   alpha, beta = pt[1] - y, pt[0] - x  M1 = P1 + alpha \* (P3 - P1)  M2 = P2 + alpha \* (P4 - P2)  P = M1 + beta \* (M2 - M1)  return np.clip(P, 0, 255)  def rotate(img, degree, pt):  dst = np.zeros(img.shape[:2], img.dtype)  radian = (degree/180) \* np.pi  sin, cos = np.sin(radian), np.cos(radian)   for i in range(img.shape[0]):  for j in range(img.shape[1]):  jj, ii = np.subtract((j ,i), pt)  y = -jj \* sin + ii \* cos  x = jj \* cos + ii \* sin  x, y = np.add((x, y), pt)  if contain((y, x), img.shape):  dst[i, j] = billinear\_value(img, (x, y))  return dst  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE)  pt = [100, 100] dst = rotate(image, 30, pt)  cv2.imshow("image", image) cv2.imshow("dst", dst) cv2.waitKey() |

11-2.

|  |
| --- |
| import numpy as np, cv2  def contain(p, shape):  return 0<=p[0] < shape[0] and 0<=p[1]<shape[1]  def billinear\_value(img, pt):  x, y = np.int32(pt)  if x>= img.shape[1]-1:x=x-1  if y>= img.shape[0]-1:y=y-1   P1, P2, P3, P4 = np.float32(img[y:y+2, x:x+2].flatten())   alpha, beta = pt[1] - y, pt[0] - x  M1 = P1 + alpha \* (P3 - P1)  M2 = P2 + alpha \* (P4 - P2)  P = M1 + beta \* (M2 - M1)  return np.clip(P, 0, 255)  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE) image = image[100:400, 100:400]  pt, angle, scale = (100, 100), 30, 1 size = image.shape[::-1]  pt1 = np.array([(20, 70), (20, 240), (300, 110)], np.float32) pt2 = np.array([(120, 20), (10, 180), (280, 260)], np.float32) aff\_mat = cv2.getAffineTransform(pt1, pt2)  dst = cv2.warpAffine(image, aff\_mat, size, cv2.INTER\_LINEAR) cv2.imshow('image', image) cv2.imshow('dst', dst) cv2.waitKey() |

12.

|  |
| --- |
| import numpy as np, cv2  def contain(p, shape):  return 0<=p[0] < shape[0] and 0<=p[1]<shape[1]  def billinear\_value(img, pt):  x, y = np.int32(pt)  if x>= img.shape[1]-1:x=x-1  if y>= img.shape[0]-1:y=y-1   P1, P2, P3, P4 = np.float32(img[y:y+2, x:x+2].flatten())   alpha, beta = pt[1] - y, pt[0] - x  M1 = P1 + alpha \* (P3 - P1)  M2 = P2 + alpha \* (P4 - P2)  P = M1 + beta \* (M2 - M1)  return np.clip(P, 0, 255)  def rotate(img, degree, pt):  dst = np.zeros(img.shape[:2], img.dtype)  radian = (degree/180) \* np.pi  sin, cos = np.sin(radian), np.cos(radian)   for i in range(img.shape[0]):  for j in range(img.shape[1]):  jj, ii = np.subtract((j ,i), pt)  y = -jj \* sin + ii \* cos  x = jj \* cos + ii \* sin  x, y = np.add((x, y), pt)  if contain((y, x), img.shape):  dst[i, j] = billinear\_value(img, (x, y))  return dst  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_COLOR)  pt = [100, 100]  b, g, r = cv2.split(image) dst\_b = rotate(b, 30, pt) dst\_g = rotate(g, 30, pt) dst\_r = rotate(r, 30, pt)  dst = cv2.merge((dst\_b, dst\_g, dst\_r))  cv2.imshow("image", image) cv2.imshow("dst", dst) cv2.waitKey() |

13.

|  |
| --- |
| import numpy as np, cv2, math  def draw\_point(x, y):  pts.append([x, y])  def onMouse(event, x, y, flags, param):  global pts  if(event == cv2.EVENT\_LBUTTONDOWN and len(pts) == 0): draw\_point(x,y)  if(event == cv2.EVENT\_LBUTTONUP and len(pts) == 1):draw\_point(x,y)   if len(pts) == 2:  x1, y1, x2, y2 = pts[0][0], pts[0][1], pts[1][0], pts[1][1]  print("직선의 길이 : ", math.sqrt((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2), "기울기 : ", abs((x1 - x2) / (y1 - y2)))  pts.append([1, 1])  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE)  pts=[] cv2.imshow('image', image) cv2.setMouseCallback('image', onMouse, 0) cv2.waitKey() |

14.

|  |
| --- |
| import numpy as np, cv2, math  def draw\_point(x, y):  pts.append([x, y])  def onMouse(event, x, y, flags, param):  global pts  if (event == cv2.EVENT\_LBUTTONDOWN and len(pts) == 0): draw\_point(x, y)  if (event == cv2.EVENT\_LBUTTONUP and len(pts) == 1): draw\_point(x, y)   if len(pts) == 2:  x1, y1, x2, y2 = pts[0][0], pts[0][1], pts[1][0], pts[1][1]  if(x2-x1 > 0 and y2-y1 > 0) : pts.append([abs(x1-x2), abs(y1-y2)])  elif(x2-x1 > 0 and y2-y1 < 0): pts.append([abs(x1-x2), -abs(y1-y2)])  elif(x2-x1 < 0 and y2-y1 > 0): pts.append([-abs(x1-x2), abs(y1-y2)])  elif(x2-x1 < 0 and y2-y1 < 0): pts.append([-abs(x1-x2), -abs(y1-y2)])   if len(pts) == 3:  M = np.float64([[1, 0, pts[2][0]], [0, 1, pts[2][1]]])  dst = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))  cv2.imshow('dst', dst)  cv2.waitKey()  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE)  pts = [] cv2.imshow('image', image) cv2.setMouseCallback('image', onMouse, 0) cv2.waitKey() |

15.

|  |
| --- |
| import numpy as np, cv2, math  def draw\_point(x, y):  pts.append([x, y])  def onMouse(event, x, y, flags, param):  global pts, angle  if (event == cv2.EVENT\_LBUTTONDOWN and len(pts) == 0): draw\_point(x, y)  if (event == cv2.EVENT\_LBUTTONUP and len(pts) == 1): draw\_point(x, y)   if len(pts) == 2:  x1, y1, x2, y2 = pts[0][0], pts[0][1], pts[1][0], pts[1][1]  dx = abs(x1 - x2)  dy = abs(y1 - y2)   radian = math.atan(dx / dy)  angle = 180 / np.pi \* radian  pts.append([1, 1])   if len(pts) == 3:  pt, scale = (0, 0), 1  size = image.shape[::-1]   rot\_mat = cv2.getRotationMatrix2D(pt, angle, scale)  dst = cv2.warpAffine(image, rot\_mat, size, cv2.INTER\_LINEAR)  pts.append([1, 1])   cv2.imshow('dst', dst)  cv2.waitKey()  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE)  pts = [] cv2.imshow('image', image) cv2.setMouseCallback('image', onMouse, 0) cv2.waitKey() |

16.

|  |
| --- |
| import numpy as np, cv2  def contain(p, p1, p2):  return p1[0] <= p[0] <p2[0] and p1[1] <=p[1] <p2[1]  def draw\_rect(img):  rois = [(p - small, small \* 2) for p in pts1]  for(x, y), (w, h) in np.int32(rois):  roi = img[y:y+h, x:x+w]  val = np.full(roi.shape, 80, np.uint8)  cv2.add(roi, val, roi)  cv2.rectangle(img, (x, y, w, h), (0, 255, 0), 1)  cv2.polylines(img, [pts1.astype(int)], True, (0, 255, 0), 1)  cv2.imshow("select rect", img)  def warp(img):  perspect\_mat = cv2.getPerspectiveTransform(pts1, pts2)  dst = cv2.warpPerspective(img, perspect\_mat, (350, 400), cv2.INTER\_CUBIC)  cv2.imshow("perspective transform", dst)  def onMouse(event, x, y, flags, param):  global check  if event == cv2.EVENT\_LBUTTONDOWN:  for i, p in enumerate(pts1):  p1, p2 = p - small, p + small  if contain((x, y), p1, p2): check = i  if event == cv2.EVENT\_LBUTTONUP: check = -1   if check >=0 :  pts1[check] = (x, y)  draw\_rect(np.copy(image))  warp(np.copy(image))  capture = cv2.VideoCapture(0)  small = np.array([12, 12]) check = -1 pts1 = np.float32([(100, 100), (300, 100), (300, 300), (100, 300)]) pts2 = np.float32([(0, 0), (400, 0), (400, 350),(0, 350)])  while True:  ret, image = capture.read()  if not ret: break  if cv2.waitKey(30) >= 0 : break   draw\_rect(np.copy(image))  cv2.setMouseCallback("select rect", onMouse, 0)  capture.release() |

17.

|  |
| --- |
| import numpy as np, cv2  image = cv2.imread("c:/computervision/chap08/image/1.jpg", cv2.IMREAD\_GRAYSCALE)  h, w = image.shape  flip1 = np.array([[-1, 0, w], [0, 1, 0]], np.float32) flip2 = np.array([[1, 0, 0], [0, -1, h]], np.float32) flip3 = np.array([[-1, 0, w], [0, -1, h]], np.float32)   dst1 = cv2.warpAffine(image, flip1, (w, h)) dst2 = cv2.warpAffine(image, flip2, (w, h)) dst3 = cv2.warpAffine(image, flip3, (w, h)) cv2.imshow("image", image) cv2.imshow("1", dst1) cv2.imshow("2", dst2) cv2.imshow("3", dst3) cv2.waitKey() |

18.

|  |
| --- |
|  |

19.

|  |
| --- |
|  |