

**과목명 : 컴퓨터 비전**

**과제명 : Fast Region Labeling**

**학 과 : 소프트웨어학과**

**제 출 일 자 : 2022.05.31**

**학 번 : 1726088**

**이 름 : 최민수**

**2. 알고리즘의 개요**

**Region Label은 이미지의 개체를 인식할 때 이미지를 영역으로 분할한 다음 영역에 레이블을 지정하는 것이다. 또 모든 같은 지역의 화소들은 같은 레이블 값을 가졌으며 좌측부터 끝까지 스캔하는 과정을 두번이나 거쳤다. 첫번째 스캔에서는 수평적 스캔이 이루어지며 임의의 레이블이 지정되고 두번째 스캔에서는 테이블의 레코드에 따라 레이블이 번호가 재할당된다.**

**Fast Region Label의 경우 한번의 raster스캔과 경계를 동시에 추적한다. Propagartion Procedure을 필요로 하지 않으며 연결성을 기록하는 메모리도 필요로 하지 않는다. Region Label에 비해 속도가 빠르며 영역의 hole을 감지할 수 있다.**

**Fast Region Label 알고리즘을 설명하기 앞서 그 전에 전제조건이 있다.**

**첫째로 4방향으로 구현을 할 것인지 8방향으로 구현을 할 것인지 정해야 한다. 이 방향의 개수에 따라 영역이 분리되어 나타나는 개체의 수가 결정된다. 두번째로 배경은 ‘0’, 레이블은 ‘R’로 초기화하며 레스터 스캔이 시작되면 양수로 라벨링이 된다. 마지막으로 조건에 따라 배경부와 홀을 추적할 때 시계 또는 반시계 방향으로 추적할지를 정해야한다.**

**Fast Region Label 알고리즘의 순서는 크게 4단계로 나뉜다.**

**1. 왼쪽 포인트가 ‘0’일 경우 0으로 채우면서 레스터 스캔을 진행한다. 새로운 레이블을 만나면 값이 ‘R’인 점을 찾을 떄까지 시계 방향으로 검색하며 경계의 다음 점을 찾고 K로 라벨링한다.**

**2. 왼쪽 레이블이 0인 경우인 경우의 ‘K’를 시작점으로 경계가 완전히 추적될 때까지 스캔을 계속하며 라벨링한다.**

**3. 홀의 경계에서도 4방향, 8방향 조건에 따라 라벨링을 진행한다.**

**4. 나머지 값들을 라벨링한다.**

**3. 구현코드 ( 8방향성 )**

import cv2,numpy as np,copy

def BoundaryTracking(mode,imgtrack,rr,cc,pix,label): # 주변 hole 경계 추적하는 함수

    prevT =0 # 이전탐색방향

    nextT =0 # 다음탐색방향

    r = rr  # row좌표

    c = cc # column좌표

    d = [0,0,0,0,0,0,0,0] #방향설정

    flag = False

    while True: # 여기선 시계방향으로 탐색

        d[0] = imgtrack[r][c + 1]; # 1행 2열

        d[1] = imgtrack[r + 1][c + 1]; # 2행 2열

        d[2] = imgtrack[r + 1][c]; # 2행 1열

        d[3] = imgtrack[r + 1][c - 1]; # 2행 0열

        d[4] = imgtrack[r][c - 1]; # 1행 0열

        d[5] = imgtrack[r - 1][c - 1]; # 0행 0열

        d[6] = imgtrack[r - 1][c]; # 0행 1열

        d[7] = imgtrack[r - 1][c + 1]; #0행 2열

        if(d[0]==0 and d[1]==0 and d[2]==0 and d[3]==0 and # 찾으려는 값이 모두 0이면 멈춤

           d[4]==0 and d[5]==0 and d[6]==0 and d[7]==0):

            break

        #마스크내의 탐색시작 방향 설정

        nextT =prevT - 3  # 규칙 설정

        if (nextT ==-1):

            nextT =7

        elif (nextT ==-2):

            nextT =6

        elif (nextT ==-3):

            nextT =5

        #마스크내의 탐색을 시계방향으로 수행

        while True:

            if((d[nextT] ==pix) or (d[nextT] ==label)):

                flag = False

                if(prevT == 1):

                    if(nextT==5):

                        flag = True

                elif(prevT == 2):

                    if(nextT==5 or nextT == 6):

                        flag = True

                elif (prevT == 3):

                    if (nextT == 5 or nextT == 6 or nextT == 7):

                        flag = True

                elif (prevT == 4):

                    if (nextT == 0 or nextT == 5 or nextT == 6 or nextT == 7):

                        flag = True

                elif (prevT == 5):

                    if (nextT != 2 and nextT != 3 and nextT !=4):

                        flag = True

                elif (prevT == 6):

                    if (nextT != 3 and nextT != 4):

                        flag = True

                elif (prevT == 7):

                    if (nextT != 4):

                        flag = True

                if(flag):

                    imgtrack[r][c] = label

                prevT = nextT

                break

            else: #다음 탐색방향 설정

                nextT = nextT+1

                if(nextT>7):

                    nextT = 0

        # 위치이동하면서 스캔

        if(nextT == 0):

            c+=1

        elif(nextT==1):

            r+=1

            c+=1

        elif(nextT==2):

            r+=1

        elif(nextT==3):

            r+=1

            c-=1

        elif(nextT==4):

            c-=1

        elif(nextT==5):

            r-=1

            c-=1

        elif(nextT==6):

            r-=1

        elif(nextT==7):

            r-=1

            c+=1

        if((r ==rr) and (c ==cc)): # 제자리로 돌아오면 멈추기

            break

    return imgtrack

img = cv2.imread("C:/Users/chlasltn/Desktop/rice.raw.jpg",1) # 이미지는 그레이스케일로 불러옴

img2 = cv2.imread("C:/Users/chlasltn/Desktop/rice.raw.jpg",0)

ret,thresh\_cv = cv2.threshold(img2, 127, 255, cv2.THRESH\_BINARY) # 이미지 이진화

cv2.imshow('Original', thresh1)

cv2.waitKey(0)

cv2.destroyAllWindows()

maxX = thresh\_cv.shape[1]

maxY = thresh\_cv.shape[0]

imgtrack = np.zeros((maxY,maxX))

dd = [[0]\*3 for i in range(3)]

print(thresh\_cv[50])

for y in range(0,maxY,1):

    for x in range(0,maxX,1):

        c = 0

        if(x ==0 or y ==0 or x ==(maxX-1) or y ==(maxY-1)):

            c = 0

        else:

            c =thresh\_cv[y][x]

            if(c==0):

                c =0

            else:

                c =-255

        print(x,y)

        imgtrack[y][x] = c

pixValue = 0

label = 0

for y in range(1,maxY -1,1):

    for x in range(1,maxX -1 ,1):

        pixValue = imgtrack[y][x]

        if(imgtrack[y][x] < 0):

            if ((imgtrack[y][x - 1]<= 0) and (imgtrack[y - 1][x - 1]<= 0)):

                label+=1

                print("라벨",label, ' ', pixValue,x,y) # 레이블마다 위치 설정

                imgtrack[y][x] = label

                imgtrack = BoundaryTracking(1,imgtrack, y, x, pixValue, label)

            elif (imgtrack[y][x-1] > 0) :

                imgtrack[y][x] = imgtrack[y][x-1]

            elif ((imgtrack[y][x-1] <= 0) and (imgtrack[y-1][x-1] > 0)) :

                imgtrack[y][x] = imgtrack[y-1][x-1]

                imgtrack =BoundaryTracking(2, imgtrack, y, x, pixValue, imgtrack[y-1][x-1])

for y in range(0,maxY,1):

    for x in range(0, maxX, 1):

        c = int(imgtrack[y][x] \* (255 / (label+1))) # 레이블의 수에 따라 밝기값을 균등분할

        if (c == 0):

            c = 255

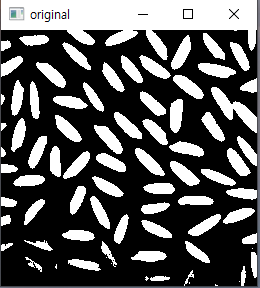
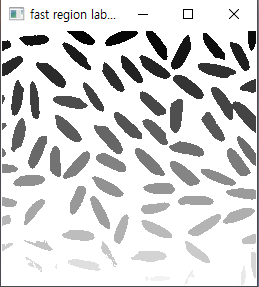
        img[y][x] = [c,c,c]

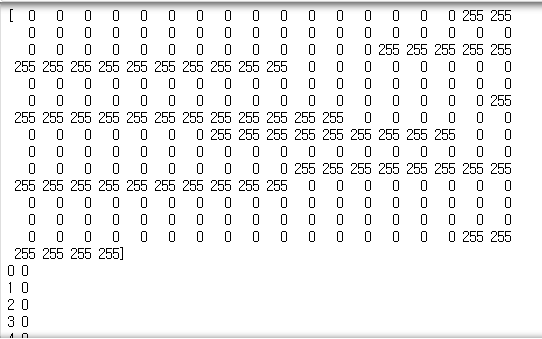
cv2.imshow('Fast Region Labeling', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**4. 실험결과**

** **

****

**코드 실행 결과 래스터 스캔을 2번 거치는 Region Label과 달리 외곽 경계와 래스터 스캔을 한번에 걸치는 Fast Region Label이 배경부가 ‘0’으로 흰색, 물체부가 라벨링되어 확실하게 객체를 인식할 수 있었다.**

**5. 소감**

**영상분할 강의시간에 간단한 알고리즘 설명과 특징들을 배우고 과제 설명을 듣고 간단한 논문이었지만 내가 해석하고 구현해낼 수 있을까가 의문이었다.**

**Fast Region Labeling의 경우 Region Labeling과 다르게 영상 실시간 처리에 있어 앞서 논문에도 나왔듯이 빠른 속도와 효율성을 느낄 수 있었고 특히 8방향성을 파이썬으로 구현함으로서 4방향과는 다른 물체인식을 확연히 느낄 수 있었다.**

**컴퓨터비전의 OpenCV를 CCTV, 얼굴인식, 자율주행 자동차에 쓰임을 알고 있었지만 기본이 되는 Labeling 과제를 진행하면서 관심을 가지게 되었다. 현재는 이미지만 다뤘지만 3차원 영상에 있어서 라이다와 레이더 각종 센서들을 찾아볼 수 있는 기회였다.**