로봇비전시스템
Assignment

과목명: 로봇비전시스템

담당교수: 조성인

제출일: 2022.06.03

학과: 기계로봇에너지공학과

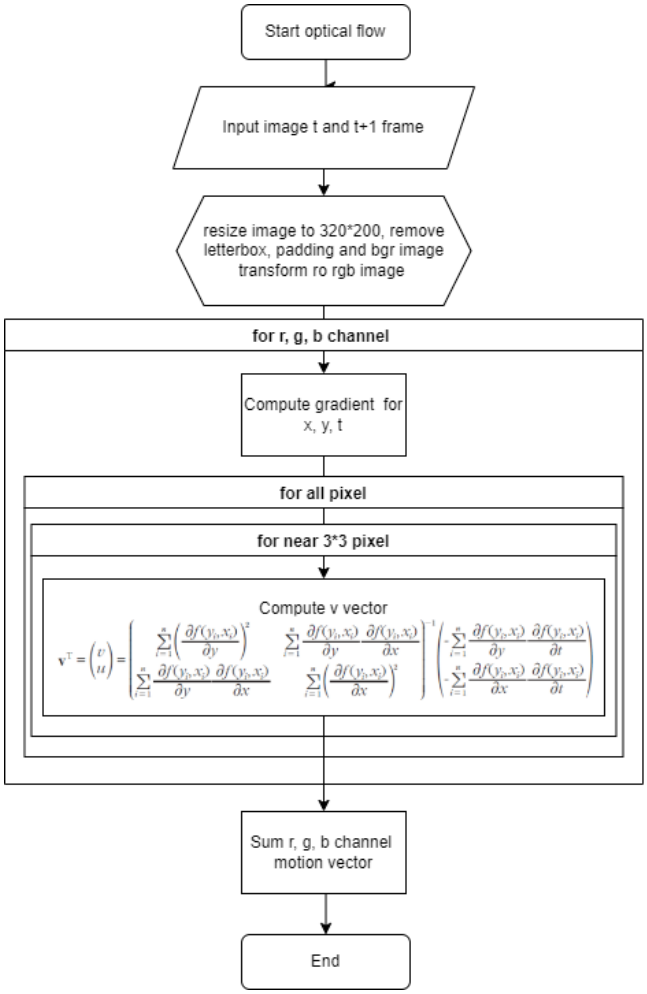
학번: 2017112387

성명: 박수웅

목차

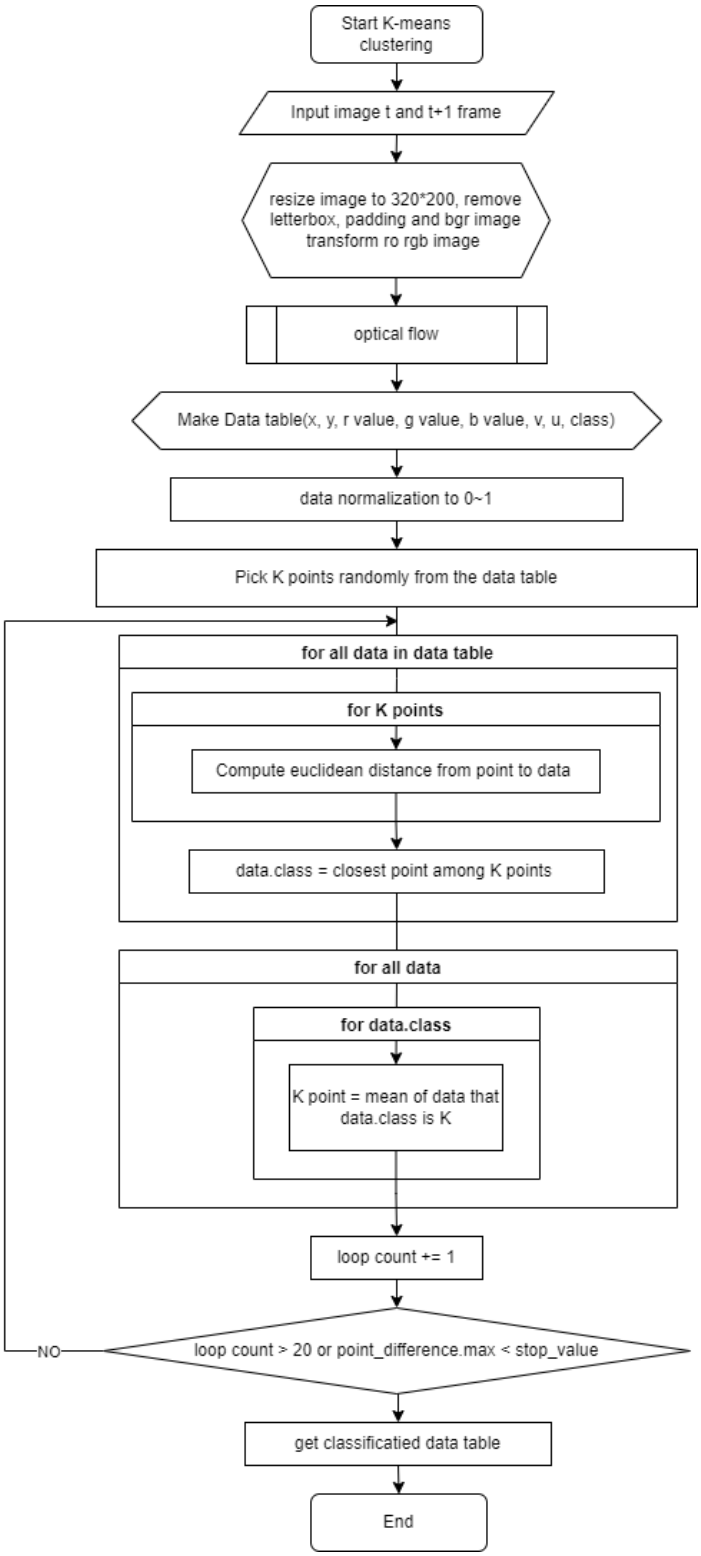
1. 개발 환경
   A. OS: Window11
   B. IDE: Visual Studio Code
   C. Programming Language: Python 3.10
   D. Library
      i. OpenCV
      ii. Numpy
      iii. Matplotlib
      iv. Scipy
      v. random

2. 진행 과정
   A. Flow chart
      i. Optical Flow



[그림 1] Optical Flow 의 flow chart

ii. K-mean clustering



[그림 2] K-mean clustering 의 flow chart

iii. Mean shift algorithm

B. 개발 과정

i. 이론적 배경

1. Optical Flow

Optical flow 는 t, t+1 의 영상에서 t 프레임 이미지의 픽셀의 움직임을 구하는 방법이다. Optical flow 의 가장 기본원리는 테일러 시리즈를 바탕으로 한 다음 수식을 따른다.

$$f(y+dy, x+dx, t+dt) = f(y,x,t) + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial t}dt$$

이러한 수식을 이용하여 optical flow 를 구하기 위해서는 여러 가정이 필요하다.

A. t, t+1 프레임의 두 이미지의 차이는 매우 작다.
B. 두 이미지 상의 밝기 차이는 없다.
C. rotation 은 없고, transformation 만 존재한다.

식으로 표현하면 다음과 같다.

$$\frac{\partial f}{\partial y}\frac{dy}{dt} + \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial t} = 0$$

이 때, 이 식에서 motion vector(u,v)에 대해서 dy/dt = v, dx/dt = u 가 된다. 따라서 위의 식은 정리하면 다음과 같고, 이 식을 optical flow conditional expression(gradient conditional expression)이라고 부른다.

$$\frac{\partial f}{\partial y}v + \frac{\partial f}{\partial x}u + \frac{\partial f}{\partial t} = 0$$

하지만 구해야 할 미지수는 두개(u, v)인데, 식은 하나이기 때문에 unique 한 해를 구하기 위해서는 추가적인 식이 필요하다.
이를 위해서 Lucas-Kanade algorith 을 사용한다. 이 알고리즘에서는 한 픽셀에 인접한 주변의 픽셀들은 같은 모션 벡터를 가진다고 가정한다. 이 때 N 은 주로 3*3 으로 한다.

$$\frac{\partial f(y_i, x_i)}{\partial y}v + \frac{\partial f(y_i, x_i)}{\partial x}u + \frac{\partial f(y_i, x_i)}{\partial t} = 0, \qquad (y_i, x_i) \in N(y, x)$$

To matrix form,

$$\begin{bmatrix}\frac{\partial f(y_i,x_i)}{\partial y} & \frac{\partial f(y_i,x_i)}{\partial x}\end{bmatrix}\begin{bmatrix}v\\u\end{bmatrix} = -\frac{\partial f(y_i,x_i)}{\partial t}, \qquad (y_i,x_i) \in N(y,x)$$

$$\begin{bmatrix}\frac{\partial f(y_1,x_1)}{\partial y} & \frac{\partial f(y_1,x_1)}{\partial x}\\ \vdots & \vdots \\ \frac{\partial f(y_n,x_n)}{\partial y} & \frac{\partial f(y_n,x_n)}{\partial x}\end{bmatrix}\begin{bmatrix}v\\u\end{bmatrix} = \begin{bmatrix}-\frac{\partial f(y_1,x_1)}{\partial t}\\ \vdots \\ -\frac{\partial f(y_n,x_n)}{\partial t}\end{bmatrix}, \qquad (y_i,x_i) \in N(y,x)$$

이는 Ax=b 형태와 동일하다. 따라서 v 에 대해 정리하면 아래와 같다.

$$\begin{bmatrix}v\\u\end{bmatrix} = \begin{bmatrix}\sum_{i=1}^{n}\left(\frac{\partial f(y_i,x_i)}{\partial y}\right)^2 & \sum_{i=1}^{n}\frac{\partial f(y_i,x_i)}{\partial y}\frac{\partial f(y_i,x_i)}{\partial x}\\ \sum_{i=1}^{n}\frac{\partial f(y_i,x_i)}{\partial y}\frac{\partial f(y_i,x_i)}{\partial x} & \sum_{i=1}^{n}\left(\frac{\partial f(y_i,x_i)}{\partial x}\right)^2\end{bmatrix}^{-1}\begin{pmatrix}-\sum_{i=1}^{n}\frac{\partial f(y_i,x_i)}{\partial y}\frac{\partial f(y_i,x_i)}{\partial t}\\ -\sum_{i=1}^{n}\frac{\partial f(y_i,x_i)}{\partial x}\frac{\partial f(y_i,x_i)}{\partial t}\end{pmatrix}$$

위의 수식을 모든 픽셀에 대해 계산해 보면 각 픽셀에 대한 motion field 를 얻을 수 있다.

Optical flow 는 위에서 설명한 여러 가정들 때문에 제약사항들이 존재한다.

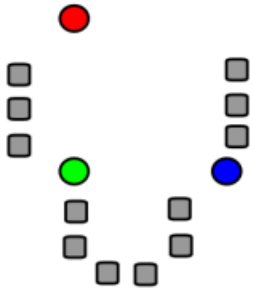
1. 256 gray level images with tremendous pixel numbers
   모든 픽셀에 대해 motion vector 를 계산하기 때문에 이미지의 사이즈가 커질수록 연산이 크게 증가한다.
2. Multiple geometric transformations
   두 이미지의 차이가 적다 라는 가정과 rotation 이 없다고 가정했기 때문에 이미지에 큰 geometric transformations 에 취약하다.
3. light source transformations
   픽셀 값의 변화가 없다고 가정했기 때문에 광원변화에 취약하다.
4. image noise occurrence
   아주 local 한 영역을 보기 때문에 작은 노이즈에도 결과에 큰 영향이 있을 수 있다.
5. Moving object and stationary object are mixed.
6. Occlusion.

Object 가 가려질 경우 두 이미지 사이의 매우 큰 변화가 일어나 제대로 작동이 안될 수 있다.

2. K-mean clustering

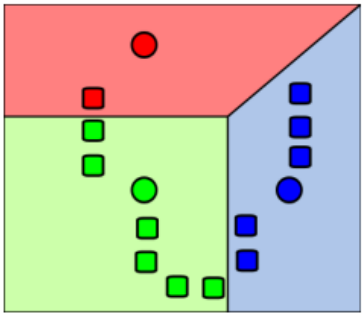
K-mean clustering 은 비지도학습 기반 clustering 의 한 방법이다. 이름 그대로 K 개의 Centroid 를 cluster 의 평균을 사용하여 최적화한다. 작동 순서는 다음과 같다.

A. Select initial centroids at random.



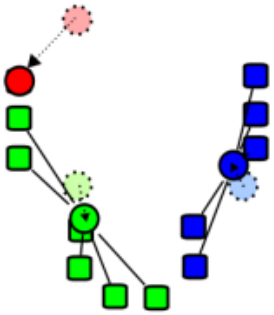
[그림 3-1] K-mean clustering 의 알고리즘

B. Assign each object to the cluster with the nearest centroid.



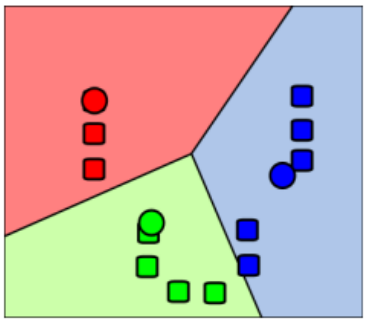
[그림 3-2] K-mean clustering 의 알고리즘

C. Compute each centroid as the mean of the objects assigned to it. Loop A~C until centroids does not move.



[그림 3-3] K-mean clustering 의 알고리즘

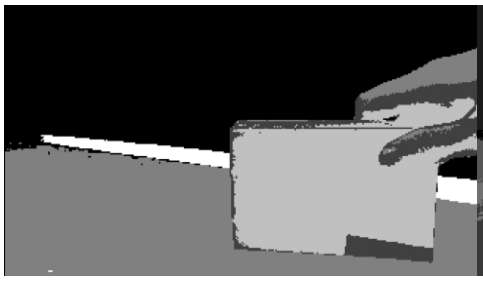
D. Assign each object to the cluster with the nearest centroid.



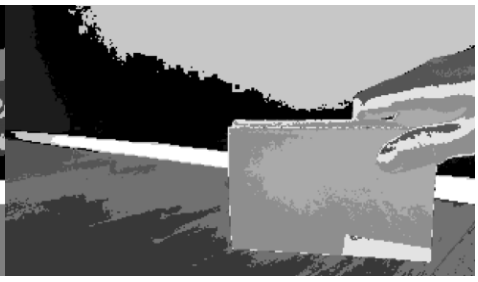
[그림 3-4] K-mean clustering 의 알고리즘

K-mean clustering 의 한계점도 존재한다. 가장 큰 문제는 K 값을 입력해주어야 한다는 점이다. K-mean clustering 은 K 값에 따라 결과가 매우 크게 좌우되는데, 특히 이미지에서는 이 K 값을 지정하는 것이 가장 큰 문제이기 때문에 활용되기가 쉽지 않다. 아래 이미지는 r, g, b 채널에 대해 각각 K=5, 10 으로 K-mean clustering 한 모습이다.

[그림 4-1] K-mean clustering 원본 이미지



[그림 4-2] K=5



[그림 4-3] K=10

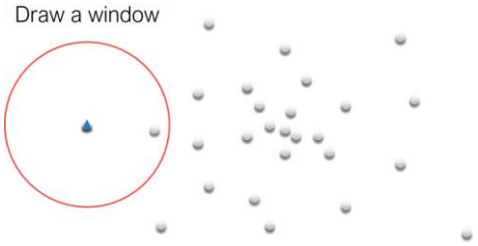
또한, initial centroids 를 랜덤하게 생성하기 때문에 최종 값이 local optimum 으로 빠질 가능성이 있다. 즉, initial centroids 에 따라 결과가 달라질 수 있다. 이러한 문제를 해결하기 위해 어느정도 적절한 initial centroids 를 지정해주는 K-mean++ 같은 알고리즘도 있다.

3. Mean shift algorithm

Mean shift, 말 그대로 평균을 옮겨가는 알고리즘이라는 뜻이다. 시작점에서 일정 거리 범위안에 들어오는 점들의 평균으로 평균점을

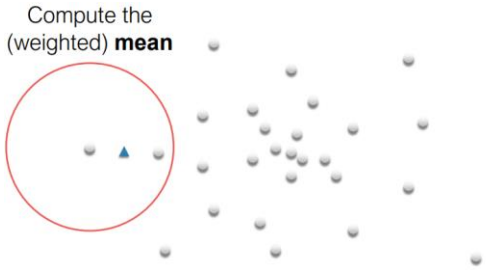
옮겨가고, 결국 가장 높은 밀도를 가진 지점으로 평균점이 움직이게 된다. 알고리즘의 순서는 다음과 같다.

A. 한 픽셀을 중심점으로 선택하고, 반경 r 에 들어오는 데이터를 구한다.



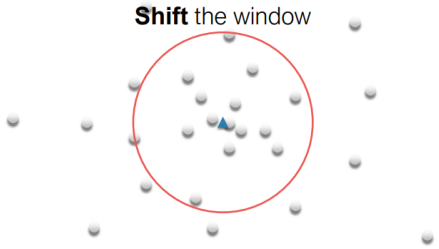
[그림 5-1] mean shift

B. 반경에 들어온 데이터들의 평균점으로 중심점을 옮긴다.



[그림 5-2] mean shift

C. 위 과정을 더 이상 중심점이 움직이지 않을 때 까지 반복한다.



[그림 5-3] mean shift

이 과정을 모든 픽셀에 대해 반복한 뒤, 최종적으로 비슷한 곳으로 중심점이 움직인 데이터들을 하나의 cluster 로 묶어주면 clustering 이 이루어 질 수 있다. 해당 과정을 수식으로 나타내면 다음과 같다.

Update $y_t \quad to \quad y_{t+1}$ until $\|y_{t+1} - y_t\| \leq \varepsilon$

$$y_{t+1} = \frac{\sum_{i=1}^{n} x_i k\left(\frac{x_i - y_t}{h}\right)}{\sum_{i=1}^{n} k\left(\frac{x_i - y_t}{h}\right)}$$

$$Gaussian\ kernel: k(x) = \begin{cases} e^{-\|x\|^2}, & \|x\| \le 1 \\ 0, & \|x\| > 1 \end{cases}$$

h: *scale of kernel* k

n: *the number pixels in the current kernel*

y_t: *current mode(mean)*

x_i: *pixel value(feature)*

ii. Code

1. Optical Flow

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as patches

## Padding function: 최외곽 픽셀 값과 동일한 값으로 padding 하는 함수
def padding(img_raw):
    img_padding = np.insert(img_raw, 0, img_raw[0], axis=0) # 위쪽 패딩
    img_padding = np.insert(img_padding, img_padding.shape[0],
img_padding[img_padding.shape[0]-1], axis=0) # 아래쪽 패딩
    img_padding = np.insert(img_padding, 0, img_padding[:,0], axis=1) #
왼쪽 패딩
    img_padding = np.insert(img_padding, img_padding.shape[1],
img_padding[:,img_padding.shape[1]-1], axis=1) # 오른쪽 패딩
    return img_padding # padding 된 이미지 반환

## Remove letterbox : 상하 래터박스를 제거해주는 함수
def rm_letterbox(img_raw):
    for i in range(img_raw.shape[0]):
        if np.mean(img_raw[i]) > 1.0: # 검은색 픽셀이 있는 행까지 카운트
            L_box = i
            break
    return img_raw[L_box:img_raw.shape[0]-L_box] # 카운트 된 행 삭제

## transform bgr image to rgb image
def bgr2rgb(bgr_img):
    rgb_img=np.zeros(bgr_img.shape)
    # bgr 순서를 rgb 순서로 변환
    for (i1, i2) in [[0, 2], [1,1], [2,0]]:
        rgb_img[:,:,i2] = bgr_img[:,:,i1]
    return rgb_img

## Convert BGR_img to Gray_img
def gray_conv(img_raw): # cv2.imread 는 bgr 로 읽어옴. 각 색상의 가중치는
밝기에 영향이 큰 정도를 고려
    img_gray = ((0.299 * img_raw[:, :, 2]) + (0.587 * img_raw[:, :, 1])
+ (0.114 * img_raw[:, :, 0]))
    return img_gray.astype(np.float64) # gray image 을 반환

## Get gradient of image
def diff_dx(img_raw): # gray 이미지를 입력으로 받음(shape = (y,x,1))
    df_dy = np.zeros((img_raw.shape[0]-2, img_raw.shape[1]-2))
    df_dx = np.zeros((img_raw.shape[0]-2, img_raw.shape[1]-2))
    for r in range(df_dy.shape[0]):
        for c in range(df_dy.shape[1]): # 모든 픽셀에 대해 연산
            df_dy[r,c] = img_raw[r+2, c+1] - img_raw[r+0, c+1]
            df_dx[r,c] = img_raw[r+1, c+2] - img_raw[r+1, c+0]
    return df_dy, df_dx # y,x 방향의 gradient image 를 반환

 ## Get df/dt
def diff_dt(img_t, img_tf): # gray 이미지 두장을 입력으로 받음(shape =
```

```
(y,x,1))
    df_dt = np.zeros((img_t.shape[0]-2, img_t.shape[1]-2))
    for r in range(df_dt.shape[0]):
        for c in range(df_dt.shape[1]):
            df_dt[r,c] = img_tf[r+1, c+1] - img_t[r+1, c+1]
    return df_dt # df/dt image 반환

## get dege image
def dect_edge(img, Thresholdvalue):
    dy, dx = diff_dx(img) # gradient image 를 받음
    edge_img = np.zeros(dy.shape).astype(np.uint8)
    for y in range(dy.shape[0]):
        for x in range(dy.shape[1]):
            # gradient 의 크기가 Thresholdvalue 보다 크면 value 를
edge(흰색), 작으면 0(검은색)으로 바꿈
            edge_img[y, x] = 255 if np.linalg.norm([dx[y,x], dy[y,x]]) >
Thresholdvalue else 0
    return padding(edge_img) # gradient image 는 이미지 사이즈가 줄었기
때문에 padding 해서 반환

## compute motion vector
def motion_vector(dy, dx, dt): # Ax=b 형태의 식 풀기. 3X3 patch 를
flatten 해서 list 로 받음
    AtA = np.zeros((2,2))
    Atb = np.zeros((2,1))
    for i in range(9): # 1~9 까지 3*3 행렬에 대해 H 와 Atb 행렬 계산
        AtA[0,0] = AtA[0,0] + dy[i]*dy[i]
        AtA[0,1] = AtA[0,1] + dy[i]*dx[i]
        AtA[1,1] = AtA[1,1] + dx[i]*dx[i]
        Atb[0,0] = Atb[0,0] - dy[i]*dt[i]
        Atb[1,0] = Atb[1,0] - dx[i]*dt[i]
    AtA[1,0] = AtA[0,1]
    if np.linalg.det(AtA) == 0: # H 의 역함수가 없으면 0,0 으로
        v = np.zeros((2,1))
    else:
        AtA_inv = np.linalg.inv(AtA)
        v = AtA_inv@Atb # v = H^(-1)*b
    return v.flatten() # v = [v, u] 값

## launch optical flow, get motion field
def optical_flow(A1_img, A2_img): # A1_img, A2_img 는 padding 된 같은
크기의 이미지
    df_dy, df_dx = diff_dx(A1_img)
    df_dt = diff_dt(A1_img, A2_img) # y, x, t 방향의 gradient 계산
    motion_vec =  np.zeros((df_dy.shape[0]-2, df_dy.shape[1]-2, 2))
    for r in range(1, motion_vec.shape[0]-1):
        for c in range(1, motion_vec.shape[1]-1): # 모든 픽셀에 대해서
motion vector 계산
            v =  motion_vector(df_dy[r:r+3,c:c+3].flatten(),
df_dx[r:r+3,c:c+3].flatten(), df_dt[r:r+3,c:c+3].flatten())
            if np.linalg.norm(v) > 5.0: v = [0, 0] # motion vector 의
크기 가 5.0 보다 크면 값에 문제가 있다고 판단, 0 으로 초기화
            motion_vec[r,c] = v # motion field 에 값 대입
    return padding(padding(motion_vec)) # 원본 이미지 크기와 같게 하기
위해 padding

## display motion field
def disp_result(img, edge_img, motion_vec): # 결과 출력
    plt.style.use('default')
    fig, ax = plt.subplots()
    for r in range(1, motion_vec.shape[0]-1):
        for c in range(1, motion_vec.shape[1]-1): # 모든 motion vector 에
대해서
            if np.any(edge_img[r-1:r+2,c-1:c+2] >= 100): # edge 근처의
vector 만 출력
```

```
                # vector의 크기가 1.0보다 작으면 검정색으로, 크면
deeppink로 출력
                color = 'black' if np.linalg.norm(motion_vec[r,c]) < 1.0
else 'deeppink'
                ax.add_patch(patches.Arrow(c, motion_vec.shape[0]-(r+1),
motion_vec[r,c,1], -motion_vec[r,c,0], width=0.3, edgecolor=color,
facecolor='white'))
    plt.xlim(-2,img.shape[1]+2)
    plt.ylim(-2,img.shape[0]+2)
    plt.imshow(np.flip(img, axis = 0)/225)
    plt.show()

## display motion field
## disp_result와 motion field의 rgb 화살표 색 빼고 동일
def disp_result_rgb(img, edge_img, motion_vec):
    plt.style.use('default')
    fig, ax = plt.subplots()
    for a in range(3):
        # r, g, b 채널에 맞게 motion vector색 설정
        clo = 'red' if a == 0 else 'green' if a == 1 else 'blue'
        for r in range(1, motion_vec.shape[0]-1):
            for c in range(1, motion_vec.shape[1]-1):
                if np.any(edge_img[r-1:r+2,c-1:c+2] >= 100):
                    color = 'black' if np.linalg.norm(motion_vec[r,c,a])
< 0.3 else clo
                    ax.add_patch(patches.Arrow(c, motion_vec.shape[0]-
(r+1), motion_vec[r,c,a,1], -motion_vec[r,c,a,0], width=0.2,
edgecolor=color, facecolor='white'))
    plt.xlim(-2,img.shape[1]+2)
    plt.ylim(-2,img.shape[0]+2)
    plt.imshow(np.flip(img, axis = 0)/225)
    plt.show()

def main():
    # 0: gray이미지로 변환 후 motion vector 계산
    # 1: r,g,b 채널에서 각각 motion vector를 계산 후 sum
    mode = 1
    file_set = "A"

    ## Load Image & down sampling
    A1_raw = cv2.resize(cv2.imread(filename=file_set+"1.jpg",
flags=cv2.IMREAD_COLOR).astype(np.float64), dsize=(320,200)) # 16:10
해상도(480,300) (640, 400)
    A2_raw = cv2.resize(cv2.imread(filename=file_set+"2.jpg",
flags=cv2.IMREAD_COLOR).astype(np.float64), dsize=(320,200))

    ## Remove letterbox & Padding & conversion
    A1_img = padding(rm_letterbox(A1_raw))
    A2_img = padding(rm_letterbox(A2_raw))

    ## transform bgr image to rgb image
    A1_img = bgr2rgb(A1_img)
    A2_img = bgr2rgb(A2_img)

    if mode == 0: # gray이미지로 변환 후 motion vector 계산
        ## Remove letterbox & Padding & conversion to gray_img
        A1_gray = gray_conv(A1_img)
        A2_gray = gray_conv(A2_img)

        ## Get edge image
        A1_edge_img = dect_edge(A1_gray, 15)

        ## cal motion vector using optical flow algorithm
        motion_vec_img = optical_flow(A1_gray, A2_gray)

        ## display motion vector and image
        disp_result(A1_gray, A1_edge_img, motion_vec_img)
```

```
    else: # r,g,b 채널에서 각각 motion vector 를 계산 후 sum
        ## Get edge image
        A1_edge_img = dect_edge(gray_conv(A1_img), 15)

        ## cal motion vector using optical flow algorithm
        motion_vec_set = np.zeros((A1_img.shape[0], A1_img.shape[1],
3 ,2))
        for i in range(3):
            motion_vec_set[:,:,i,:] = optical_flow(A1_img[:,:,i],
A2_img[:,:,i])
        motion_vec_img = np.sum(motion_vec_set, axis=2)

        ## display motion vector and image
        disp_result(A1_img, A1_edge_img, motion_vec_img)
        disp_result_rgb(A1_img, A1_edge_img, motion_vec_set)

if __name__ == '__main__':
    main()
```

2. K-mean clustering

```
import cv2
import numpy as np
from scipy.spatial import distance
from matplotlib import pyplot as plt
import random
from Optical_flow import *

## Make datatable
## 각 픽셀에 y, x, r value, g value, b value, v, u, class 정보가 0~1 로 정규화되어
출력
## scale 은 각 정보의 가중치
def set_datatable(img, motion_vec_img, scale):
    data_table = np.zeros((img.shape[0], img.shape[1], 8))
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            data_table[r, c, 0] = r/img.shape[0] * scale[0]
            data_table[r, c, 1] = c/img.shape[1] * scale[1]
            data_table[r, c, 2] = img[r, c, 0]/255.0 * scale[2]
            data_table[r, c, 3] = img[r, c, 1]/255.0 * scale[3]
            data_table[r, c, 4] = img[r, c, 2]/255.0 * scale[4]
            data_table[r, c, 5] = motion_vec_img[r, c,
0]/np.max(motion_vec_img[:,:,0]) * scale[5]
            data_table[r, c, 6] = motion_vec_img[r, c,
1]/np.max(motion_vec_img[:,:,1]) * scale[6]
            data_table[r, c, 7] = 0 # class 정보
    return data_table

## Picking randomly K points by centroid from the data
def get_centroids(data, K):
    centroids = np.empty((K, data.shape[2]-1))
    for k in range(K):
        centroids[k] = data[random.randrange(0, data.shape[0]),
random.randrange(0, data.shape[1]), 0:data.shape[2]-1]
    return centroids

## Assign each object to the cluster with the nearest centroid.
def classification(data_table, centroids, K):
    for r in range(data_table.shape[0]):
        for c in range(data_table.shape[1]):
            ud = list()
            #각 데이터들과 centroids 사이의 euclidean 거리를 구하고 가장 가까운
centroid 에 따른 정보를 class 에 0~K 값으로 저장
            for k in range(K):

ud.append(distance.euclidean(data_table[r,c,0:data_table.shape[2]-1],
centroids[k]))
            data_table[r,c,data_table.shape[2]-1] = np.array(ud).argmin()
    # cluster 된 data_table 반환
    return data_table
```

```
    ## Compute each centroid as the mean of the objects assigned to it.
    def update_centroids(data_table, centroids, K):
        point_dif = list()
        for k in range(K): #각 cluster 마다 반복
            u = list()
            # 모든 objects 중 k cluster 에 속한 데이터만 추출
            for r in range(data_table.shape[0]):
                for c in range(data_table.shape[1]):
                    if data_table[r,c,data_table.shape[2]-1] == k:
                        u.append(data_table[r,c,0:data_table.shape[2]-1])
            # 직전의 centroids 에서 얼마나 이동했는지 확인
            point_dif.append(np.mean(centroids[k] - np.mean(np.array(u), axis = 0)))
            # mean of the objects assigned
            centroids[k] = np.mean(np.array(u), axis = 0)
        # data_table 과 업데이트 된 centroids 와 가장 크게 이동했던 centroid 의 이동거리
    반환
        return data_table, centroids, np.array(point_dif).max()


    ## launch K-mean clustering Once
    def k_mean(data_table, centroids, K):
        data_table = classification(data_table, centroids, K)
        return update_centroids(data_table, centroids, K)

    def main():
        file_set = "A"
        K = 10
        stop_value = 0.000000001
        scale = [1,1,1,1,1,1,1] #각 픽셀에 y, x, r, g, b, v, u 에 대해 k-mean
    알고리즘에서 얼마나 많이 고려할지 scale 값(클수록 분류 확실 -> 고려 많이 함)

        ## Load Image & down sampling
        A1_raw = cv2.resize(cv2.imread(filename=file_set+"1.jpg",
    flags=cv2.IMREAD_COLOR).astype(np.float64), dsize=(320,200)) # 16:10
    해상도(480,300) (640, 400)
        A2_raw = cv2.resize(cv2.imread(filename=file_set+"2.jpg",
    flags=cv2.IMREAD_COLOR).astype(np.float64), dsize=(320,200))

        ## Remove letterbox & Padding
        A1_img = padding(rm_letterbox(A1_raw))
        A2_img = padding(rm_letterbox(A2_raw))

        ## transform bgr image to rgb image
        A1_img = bgr2rgb(A1_img)
        A2_img = bgr2rgb(A2_img)

        ## cal motion vector using optical flow algorithm
        motion_vec_set = np.zeros((A1_img.shape[0], A1_img.shape[1], 3 ,2))
        for i in range(3):
            motion_vec_set[:,:,i,:] = optical_flow(A1_img[:,:,i], A2_img[:,:,i])
        motion_vec_img = np.sum(motion_vec_set, axis=2)

        ## Make data table
        data_table = set_datatable(A1_img, motion_vec_img, scale)

        ## Get initial centroids
        centroids = get_centroids(data_table, K)
        count = 0

        ## K-mean clustering loop start
        while True:
            # K-mean clustering once
            data_table, centroids, point_dif = k_mean(data_table, centroids, K)
            # decide whether to stop
            # centroids 의 변화가 일정 값 이하이거나 20 번 이상 반복하면 끝
            if point_dif < stop_value or count > 50:
                print(count+1, "cal end")
                break
            else:
                count += 1
                print(count, "keep cal")
                print((point_dif-stop_value)/point_dif, point_dif, "process")

        ## display result
        ## gray image 로 K 개의 색으로 clustering 하여 출력
```

```
        k_mined_img = data_table[:,:,data_table.shape[2]-1]/(K-1)*255
        plt.imshow(k_mined_img, cmap='gray')
        plt.show()

    if __name__ == '__main__':
        main()
```

3. Mean shift

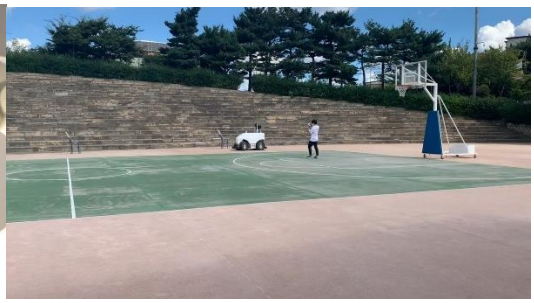
```

```

3. 결과 분석

A. 정확도 평가

테스트에는 직접 찍은 동영상에서 프레임을 추출하여 사용하였다. 먼저, 이상적인 상황이라 생각되는 이미지 세트 두장을 준비해 보았다. 테스트 이미지 1 번은 카메라는 최대한 고정되어있는 상황에서 손과 마우스만 1 시 방향으로 움직이는 이미지이다. 테스트 이미지 2 번은 차량과 사람이 왼쪽으로 이동하지만, 실제로 픽셀상 위치는 동일하고, 카메라가 왼쪽으로 회전하여 결과적으로는 배경이 오른쪽으로 움직이는 이미지이다. 사람의 왼쪽 다리가 위로 올라가는 움직임 또한 있다.

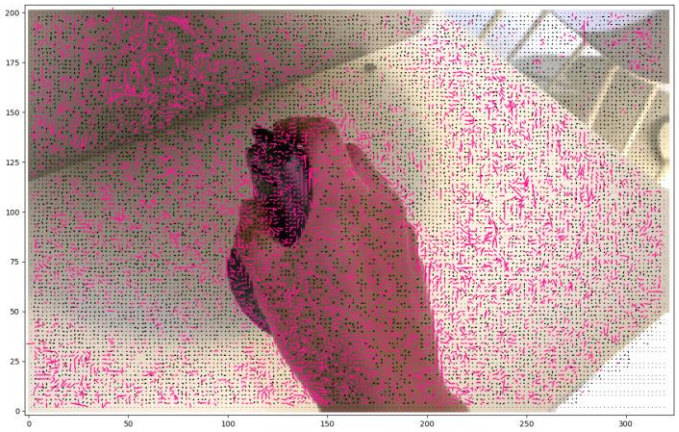
[그림 6-1] 테스트 이미지 1



[그림 6-2] 테스트 이미지 2

i. Optical Flow

1. 테스트 이미지 1



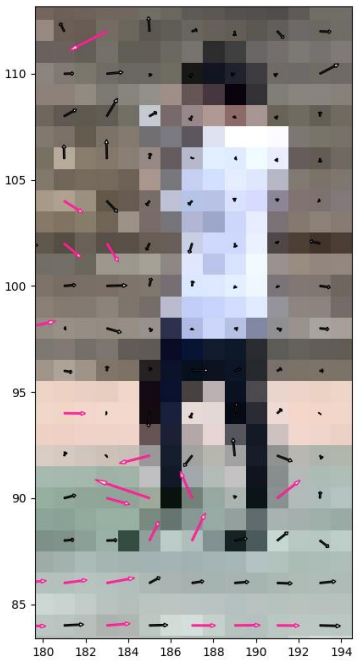
[그림 7] 테스트 이미지 1 의 motion field

정확한 테스트를 위해 모든 픽셀의 motion vector 를 출력해 보았다. 예상과는 다르게 edge 부분의 motion vector 들은 예상과 비슷하게 방향과 크기가 나왔지만, motion vector 가 거의 없어야하는 평면 부분에서 motion vector 들의 방향과 크기가 문제가 있어 보였다.

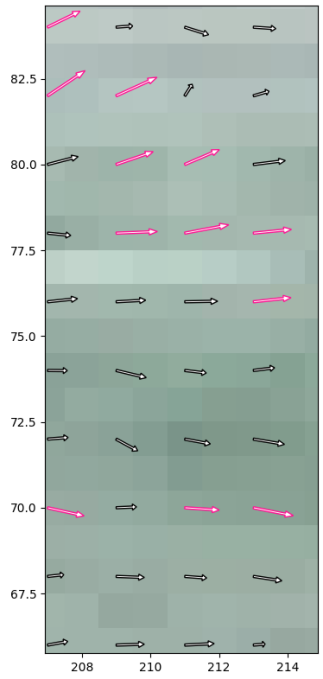
2. 테스트 이미지 2

[그림 8] 테스트 이미지 2 의 motion field

마찬가지로 평면에 가까운 부분에서도 큰 motion vector 가 나온다. 하지만 움직임이 멈춰있는 자동차와 사람의 motion vector 는 매우 작게나온 것을 확인 할 수 있었다. 다음은 사람과 농구 코트의 흰 선을 확대한 모습이다.



[그림 9-1] 사람 motion field



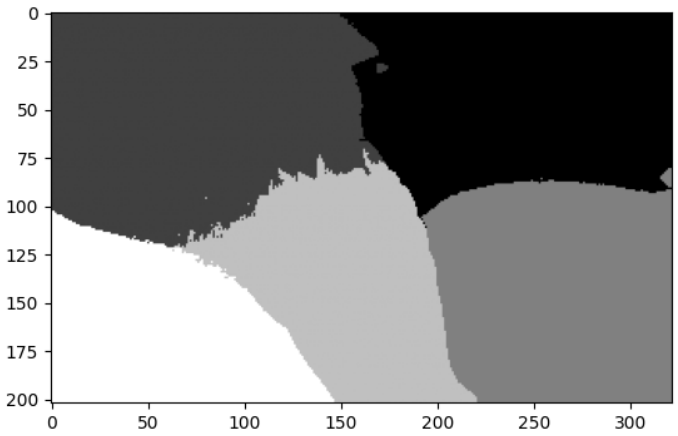
[그림 9-2] 농구장 motion field

사람의 왼쪽 다리가 올라가는 방향이고, 배경은 오른쪽으로 흘러가는 방향으로 매우 잘 나온 것을 볼 수 있다. 코트의 경우에도 매우 균일하게 motion vector 가 예상한 방향으로 나온 것을 볼 수 있다. 따라서 결론을 내리자면, 특징점이 있는 부분의 motion vector 는 상당히 잘 나온 것을 볼 수 있지만, 특징점이 없는 평면 부분에서는 상당히 부정확한 모습을 보이는 것을 확인하였다.

ii. K-mean clustering

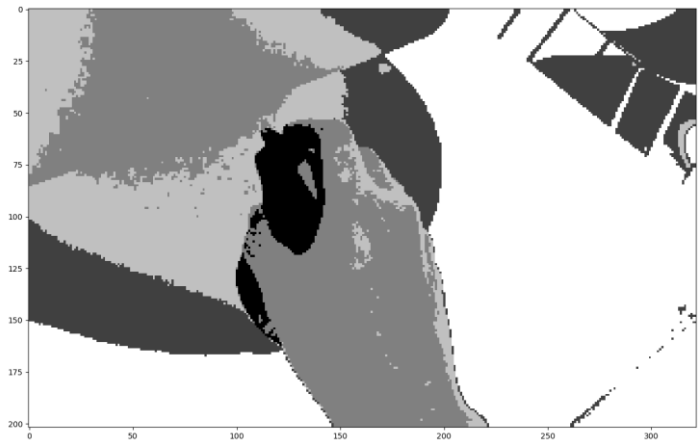
1. 테스트 이미지 1

먼저 첫번째 테스트 이미지의 모든 데이터(x,y,r,g,b,v,u)를 같은 크기로 고려하여 clustering 을 해보았다. (코드의 scale = (1,1,1,1,1,1,1)) K=5 를 사용해 보았다.



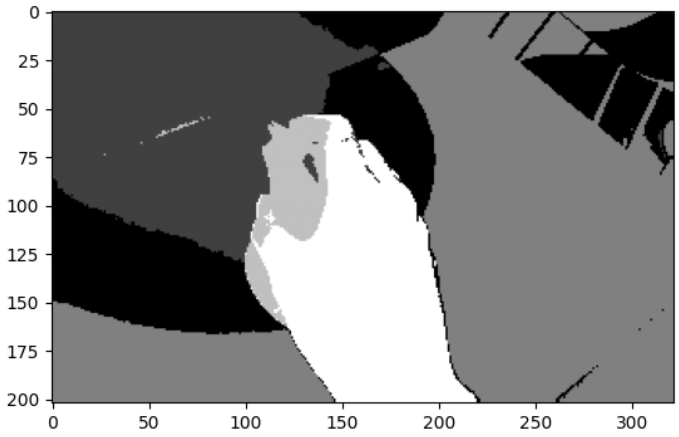
[그림 10] K-mean clustering by scale = (1,1,1,1,1,1,1)), K=5

매우 큰 위화감을 느낄 수 있었다. x ,y 의 영향을 매우 크게 받은 것 같은 모양이다. 이 예상이 맞는지 확인해 보고자 x, y 의 영향을 크게 줄여보았다. (scale = (0.2,0.2,1,1,1,1,1)



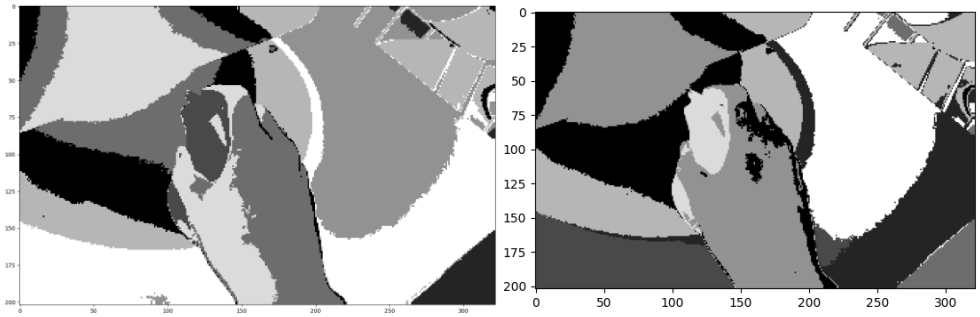
[그림 11] K-mean clustering by scale = (0.2,0.2,1,1,1,1,1)), K=5

의미론적인 사람의 관점에서 봤을 때, 확실히 이전의 clustering 에 비해 결과물이 좋아졌다. 여기서 직전의 Optical Flow 의 motion vector 에 문제가 있던 점을 고려해 v, u 의 영향력도 조금 낮추어 보았다. (scale = (0.2,0.2,1,1,1,0.3,0.3))



[그림 12] K-mean clustering by scale = (0.2,0.2,1,1,1,0.5,0.5)), K=5

큰 변화는 없지만, clustering 의 구역이 비교적 깔끔해진 것을 볼 수 있었다. 여기서 K 값을 조금 늘려보았다. (K=8)

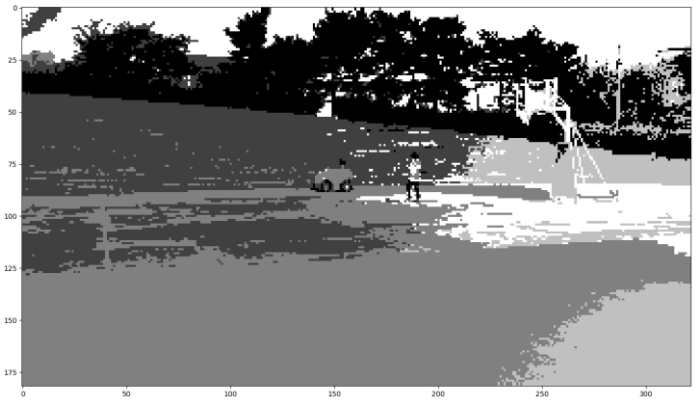


[그림 13] K-mean clustering by scale = (0.2,0.2,1,1,1,0.5,0.5)), K=8

오히려 손 부분이 불필요하게 clustering 이 나누어졌고, 노트북과 그림자 부분도 많이 나누어졌다. 혹시 초기값의 문제일 수 있어 동일한 파라미터로 한번 더 진행해 보았지만 크게 개선되는 모습은 보기 힘들었다.

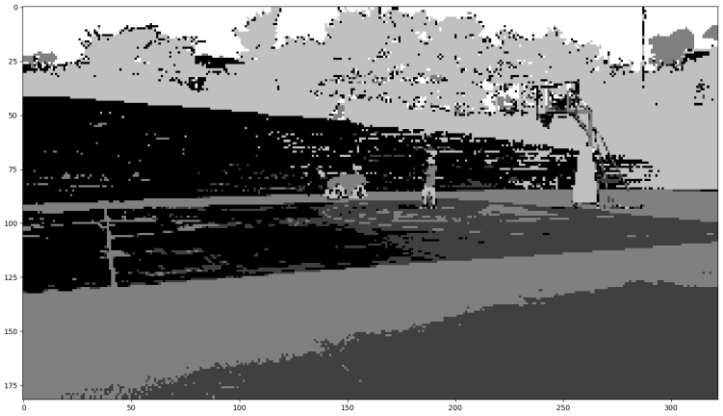
2. 테스트 이미지 2

두번째 테스트 이미지에서 첫번째 파라미터와 동일하게 모두 같은 scale 을 가진 상태로 K=5 에서 진행해 보았다.



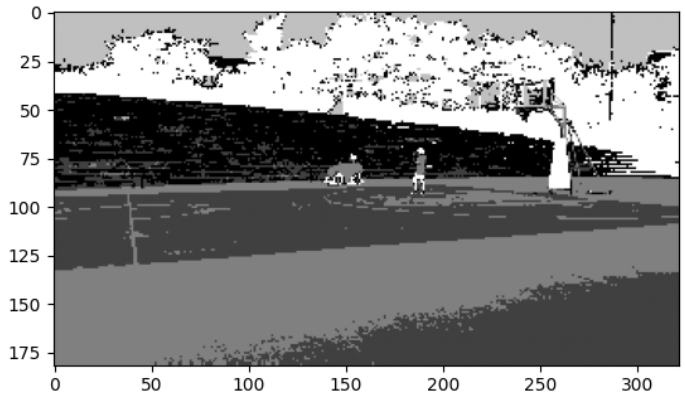
[그림 14] K-mean clustering by scale = (1,1,1,1,1,1,1)), K=5

첫번째 테스트 이미지보다 motion vector 가 이상적으로 나왔기 때문인지 상당히 나쁘지 않은 결과를 보여주었다. 하지만 전체적으로 1~4 분면을 기준으로 어느정도 나뉜 모습을 보여준다. 따라서 이전처럼 x, y 의 scale 을 조금 낮추어 보았다(scale = (0.5,0.5,1,1,1,1,1))



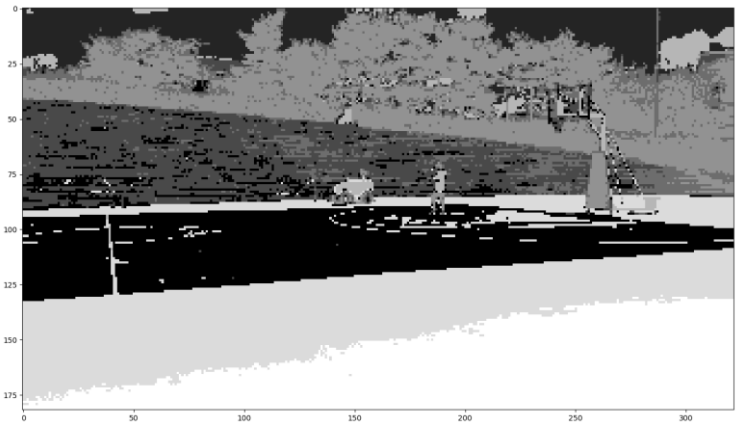
[그림 15] K-mean clustering by scale = (0.5,0.5,1,1,1,1,1)), K=5

쓸데없이 나누어져 있었던 계단부분이 하나로 clustering 되었고, 픽셀 위치에 따라 나누어 지는 경향이 있던 것이 완화되었다. 여기서 해당 이미지가 전체적으로 y 축을 기준으로 경계가 나누어져 있기 때문에 x 축의 영향을 더 줄이고, y 축의 영향을 늘려보았다. (scale = (0.6,0.3,1,1,1,1,1))



[그림 16] K-mean clustering by scale = (0.6,0.3,1,1,1,1,1)), K=5

좌우로 나누어져있던 농구 코트가 이전보다 잘 clustering 되어있는 경향을 볼 수 있었다. 마지막으로 여기서 K 값을 조금 올려보았다. (K = 8)



[그림 17] K-mean clustering by scale = (0.6,0.3,1,1,1,1,1)), K=8

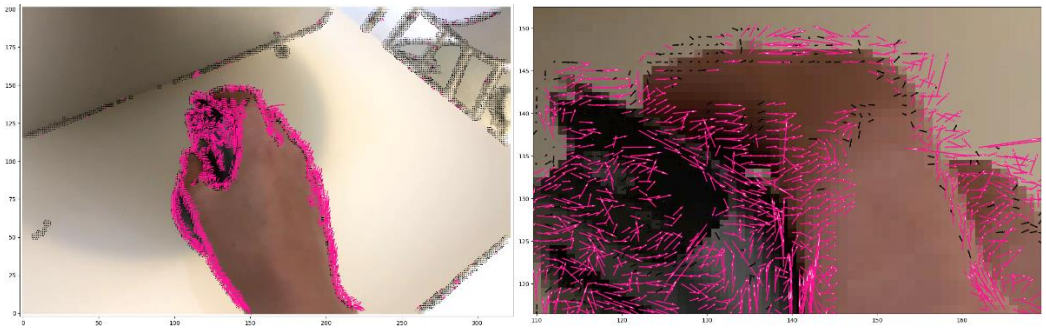
테스트 이미지 2 번의 경우 K=8 을 주었을 때 더 잘 clustering 되는 경향을 보여주었다.

iii. Mean shift

B. 동작 과정

i. Optical Flow

효과적으로 나타내기 위해 motion vector 의 크기가 0.5 보다 작다면 검정색으로, 크다면 핑크색으로 표현해 보았다. 또한 motion vector 를 edge 부분에서만 표현하면 더 효과적으로 구할 수 있을 것 같아 아래와 같이 출력해 보았다.



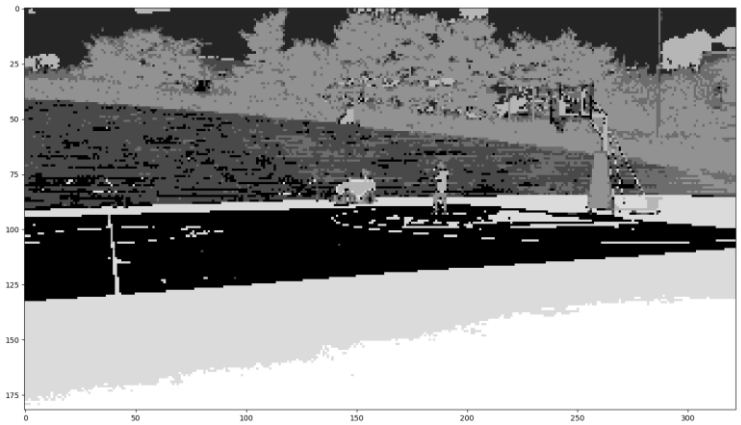
[그림 18] 테스트 이미지 1 의 edge 부분의 motion field

그림과 같이 움직임이 있는 손과 마우스부분의 motion vector 의 크기가 크기 때문에 분홍색 화살표로 나타나있고, 확대해보면 그림과 같이 오른쪽과 위쪽으로 motion vector 가 나타남을 볼 수 있다. 기존 테스트 때

이미지 출력의 연산량을 줄이기 위해 이렇게 시각화 했기 때문에 평평한 면의 모션벡터가 문제가 있다는 점을 간과했었다.

ii. K-mean clustering

Clustering 된 이미지를 gray scale 로 출력하였다. 각 cluster 의 색상을 (cluster index)*255/K 를 해주어 255 값의 범위에서 일정한 텀을 두고 value 를 조정하여 시각화 하였다.



[그림 19] 테스트 이미지 1 의 K-mean clustering(K=8) 시각화

*255/K 를 해주었기 때문에, 1 번 cluster 의 경우 검정색으로 나타나고, 차례로 밝아지는 방향으로, 가장 마지막 (그림 19 의 경우)8 번 cluster 가 흰색으로 나타나게 된다.

iii. Mean shift