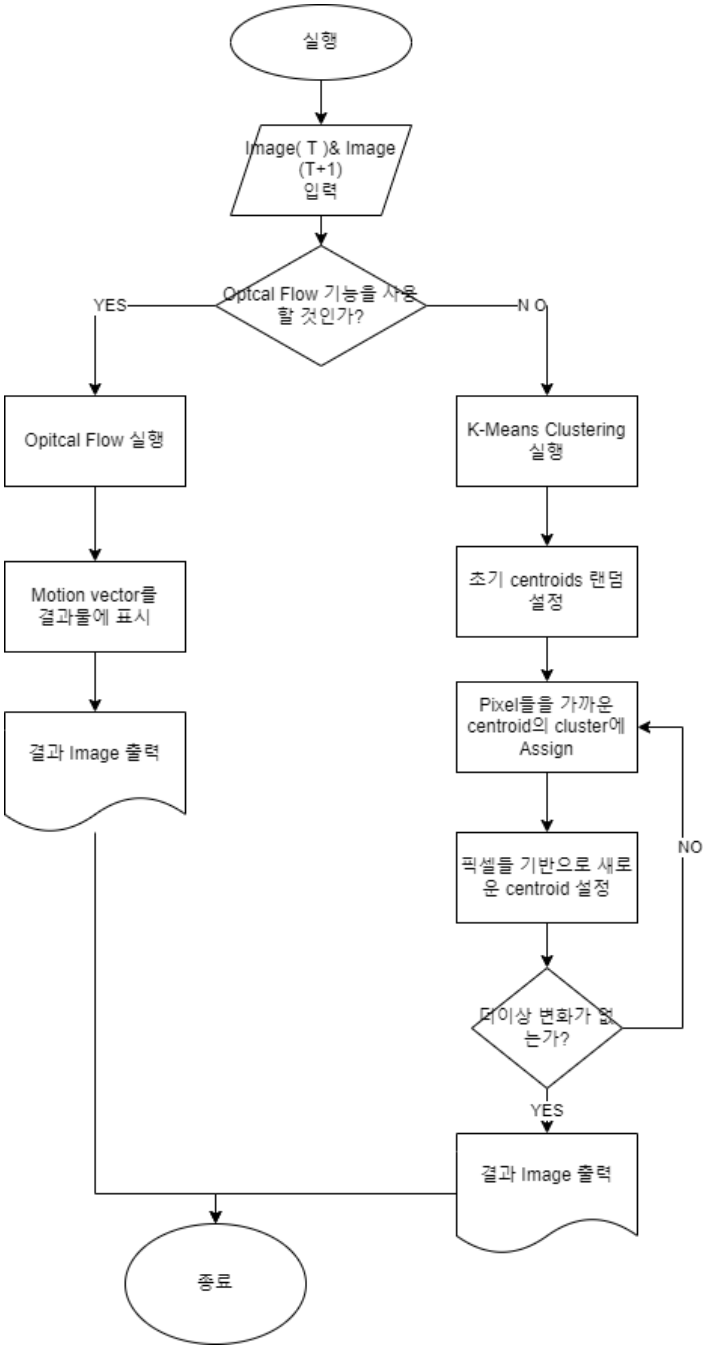
로봇비전시스템 Assignment

2016112529 신민규

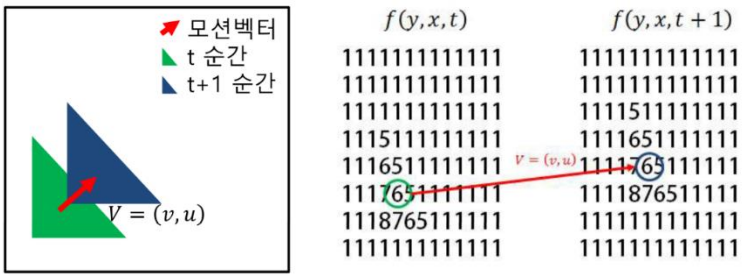
1. 개발환경

Image 를 다루는데 유용한 CV2 와 배열 및 계산에 유리한 Numpy 의 사용을 위하여 Pycharm 을 이용한 Python 으로 개발

Flowchart



2. Optical Flow (Lucas-Kanade Algorithm)



Optical Flow 를 추정하기 위한 2 가지 전제조건

1. color/brightness constancy : 어떤 픽셀과 그 픽셀의 주변 픽셀의 색/밝기는 같음을 가정

2. small motion : Frame 간 움직임이 작아서 어떤 픽셀 점은 멀리 움직이지 않는 것을 가정

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

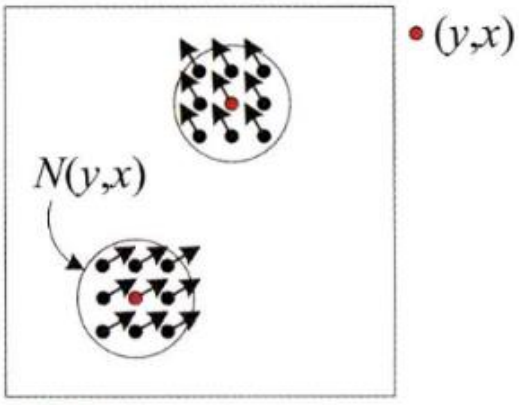
$\partial f/\partial y$, $\partial f/\partial x$ 그리고 $\partial f/\partial$t 는 y, x, t 의 gradient

dy/dt, dx/dt = 구해야 하는 motion vector

문제점: gradient 3 개의 값을 모두 알아도 미지수는 2 개이지만 방정식은 1 개인 문제가 발생한다

해결: 이를 해결하기 위해 추가적인 가정(Lucas-Kanade Algorithm)을 통해 식을 추가하여 계산한다

Lucas-Kanade Algorithm



새로운 전제조건

3. 픽셀 (y,x)를 중심으로 하는 윈도우 영역 N(y,x)의 optical flow 는 같다

이웃 영역에 속하는 모든 픽셀 $(y_i,x_i), i=1,2,\cdots,n$ 은 같은 motion vector $v=(v,u)$를 가져야한다.

따라서,

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

결과식

$$\mathbf{v}^{\mathrm{T}}=(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

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

CODE

```python
def optical_flow(frame_t, frame_t_1, window_size): #window_size u,v 구하기위한 근접픽셀수 설정

    corners = cv2.goodFeaturesToTrack(frame_t, 0, 0.01, 0.1) #코너 찾기함수 (이미지,최대코너갯수(0=무한대),코너점 결정을 위한 값,코너점 사이의 최소 거리

    frame_t = frame_t / 255
    frame_t_1 = frame_t_1 / 255

    kernel_x = np.array([[-1, 1], [-1, 1]]) #edge detection을 위한 Kernel들
    kernel_y = np.array([[-1, -1], [1, 1]])
    kernel_t = np.array([[1, 1], [1, 1]])

    fx = cv2.filter2D(frame_t, -1, kernel_x)              #X,Y,T 의 gradient 구하기
    fy = cv2.filter2D(frame_t, -1, kernel_y)
    ft = cv2.filter2D(frame_t_1, -1, kernel_t) - cv2.filter2D(frame_t, -1, kernel_t)


    u = np.zeros(frame_t.shape)
    v = np.zeros(frame_t.shape)

    n = int(window_size / 2)

    for feature in corners:
            j, i = feature.ravel()
            i, j = int(i), int(j)

            I_x = fx[i-n:i+n+1, j-n:j+n+1].flatten()  # 이웃픽셀 들 추가
            I_y = fy[i-n:i+n+1, j-n:j+n+1].flatten()
            I_t = ft[i-n:i+n+1, j-n:j+n+1].flatten()

            b = np.reshape(I_t, (I_t.shape[0], 1))
            A = np.vstack((I_x, I_y)).T  #수직으로 행렬 결합

            V = np.matmul(np.linalg.pinv(A), b)     #/matmul =행렬곱 함수/ linalg.pinv 의사 역행렬

            u[i, j] = V[0][0]
            v[i, j] = V[1][0]

    return (u, v)
```

```python
def getresultimg(frame, U, V, output):

    line_color = (0, 255, 0)

    for i in range(frame.shape[0]):
        for j in range(frame.shape[1]):
            u, v = U[i][j], V[i][j]

            if u and v:
                frame = cv2.arrowedLine(frame, (i, j), (int(round(i+u)), int(round(j+v))), line_color, thickness=1)
    cv2.imwrite(output, frame)
```

결과

Input image (t) – 480 x 360

Input image (t+1) – 480 x 360

Result image (window size=3)

Result image (window size=6)

이웃픽셀인 Window 의 크기를 paramater 로 비교해 보았다

Window size 가 더 작은 결과값에서 오히려 더 큰 motion vecetor 가 제대로 검출되는 경향을 보였다.

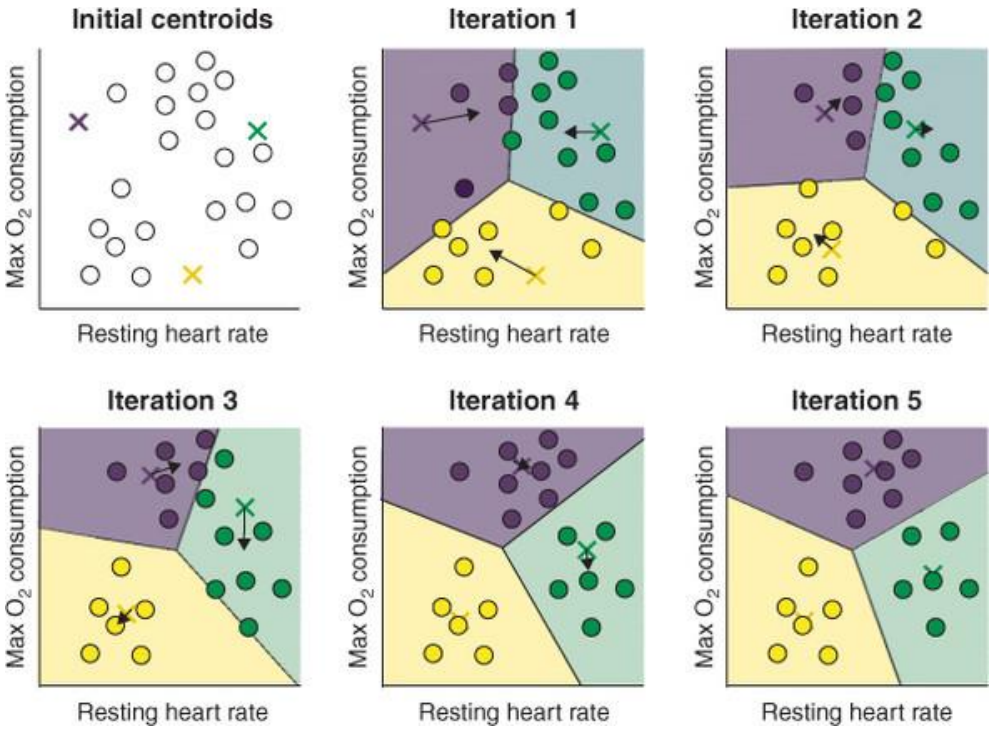
Window size 가 커지면 더 넓은 범위의 pixel 이 같은 rgb 값과 방향을 가진다는 가정이 더해지기 때문에 Blur 된 현상과 비슷한 결과가 도출되었다.

3.K-Means Clustering Segmentation

Cluster 란 비슷한 특성을 가진 데이터끼리의 묶음.

Clustering 이란 어떤 데이터들이 주어졌을 때, 그 데이터들을 Cluster 로 assign 시켜주는 것을 의미합니다

각 클러스터의 중심을 Centroid 라고 한다.



1. Cluster 수 설정(K 결정)

2. 초기 Centroid 랜덤하게 설정

3. 모든 데이터를 순회하며 각 데이터마다 가장 가까운 Centroid 가 속해있는 클러스터로 assign

4. Centroid 를 클러스터의 중심으로 이동

5. 클러스터에 assign 되는 데이터가 없을 때까지 스텝 3, 4 를 반복

CODE

```
def segmentation(self, image, k=2):
    centroids = []
    clusters = {}
    i = 1

    while (len(centroids) != k):
        cent = image[np.random.randint(0, image.shape[0]), np.random.randint(0, image.shape[1])]
        if (len(centroids) >= 1):
            if (cent.tolist() not in centroids):
                centroids.append(cent.tolist())
        else:
            centroids.append(cent.tolist())      #centroid의 초기값을 랜덤으로 정해준다 centroids list에 추가
    print("Initial centroids {}".format(centroids))

    clusters = self.kmeans(clusters, image, centroids, k)   #kmean clustering 실행
    new_centroids = self.get_new_centroids(clusters, k)     #새로운 centroids 할당

    while (not (np.array_equal(new_centroids, centroids))) and i <= 15: # 수렴할 때까지 반복(max치 15)
        centroids = new_centroids
        clusters = self.kmeans(clusters, image, centroids, k)
        new_centroids = self.get_new_centroids(clusters, k)
        i = i + 1
    else:
        print("END")

    image = self.assignPixels(clusters, image, k)

    return image
```

```
def kmeans(self, clusters, image, centroids, k): #각각의 pixel들을 가장 가까운 centroids의 clusters에 assgin

    def add_cluster(minIndex, pixel):
        try:
            clusters[minIndex].append(pixel)
        except KeyError:
            clusters[minIndex] = [pixel]

    for x in range(0, image.shape[0]):
        for y in range(0, image.shape[1]):
            pixel = image[x, y].tolist()
            minIndex = self.findMinIndex(pixel, centroids)
            add_cluster(minIndex, pixel)        #가장 가까운 cluster에 pixel의 RGB값을 저장

    return clusters
```

```python
def get_new_centroids(self, clusters, k): # cluster에 clustering된 RGB값의 mean을 통해 새로운 centroid를 계산
    new_centroids = []
    keys = sorted(clusters.keys())
    for k in keys:
        n_mean = np.mean(clusters[k], axis=0)
        cent_new = (int(n_mean[0]), int(n_mean[1]), int(n_mean[2]))
        new_centroids.append(cent_new)

    return new_centroids
```

```python
def findMinIndex(self, pixel, centroids):#pixel에서 가장 가까운 cluster 찾기
    dist = []
    for i in range(0, len(centroids)):
        d = np.sqrt(int((centroids[i][0] - pixel[0])) ** 2 + int((centroids[i][1] - pixel[1])) ** 2 + int(
            (centroids[i][2] - pixel[2])) ** 2)                                    #pixel의 RGB값과 centroid을 거리를 계산
        dist.append(d)
        minIndex = dist.index(min(dist))              #pixel가 가장 가까운 cluster의 index 찾기

    return minIndex
```

```python
def assignPixels(self, clusters, image, k):  #cluster의 RGB값을 통해 output image의 RGB값을 계산
    cluster_centroids = []
    keys = sorted(clusters.keys())
    for k in keys:
        n_mean = np.mean(clusters[k], axis=0)
        cent_new = (int(n_mean[0]), int(n_mean[1]), int(n_mean[2]))
        cluster_centroids.append(cent_new)

    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            Value = cluster_centroids[self.findMinIndex(image[x, y], cluster_centroids)]
            image[x, y] = Value

    return image
```

결과

Input image 480 x 360

Result

K=2

K=3

K=4

K 의 값에따라 Cluster 의 개수가 정해지므로 K 가 커지면 result 가 더 디테일 하게 나누어져 표현되었다

인물, 나무, 하늘 ,땅 등의 개체가 segment 되었지만 단순히 pixel 값을 기반으로 clustering 한 만큼 개체별로 눈에 띄게 확실히 segment 되지 못하였다

개체의 RGB 값이 하나로 이루어지지 않기 때문에 일어나는 문제점 이다