계절학기 발표자료

필수 1~6, 선택 2,14

손한기- 부울경 2반

Contents

목차

0) 학습목표

1) 필수 프로젝트 소개

2) 선택 프로젝트 소개

3) 선택 14 프로젝트(얼굴인식)

4) 후기

Contents

0. 학습목표

- 새로운 기술을 확인하고 실습해보기
- Ssafy git 의 사용방법을 익히기
- 데이터 베이스에 대한 이해 향상시키기

Contents

1. 수행 프로젝트 소개

1) Git 실습

2) DB설계, ERD 작성 실습

3) 프론트엔드 프로젝트

4) 테스트 케이스 만들기

5) Dockerize project

6) 알고리즘 순서도와 의사코드

Contents

2. 선택 프로젝트 소개

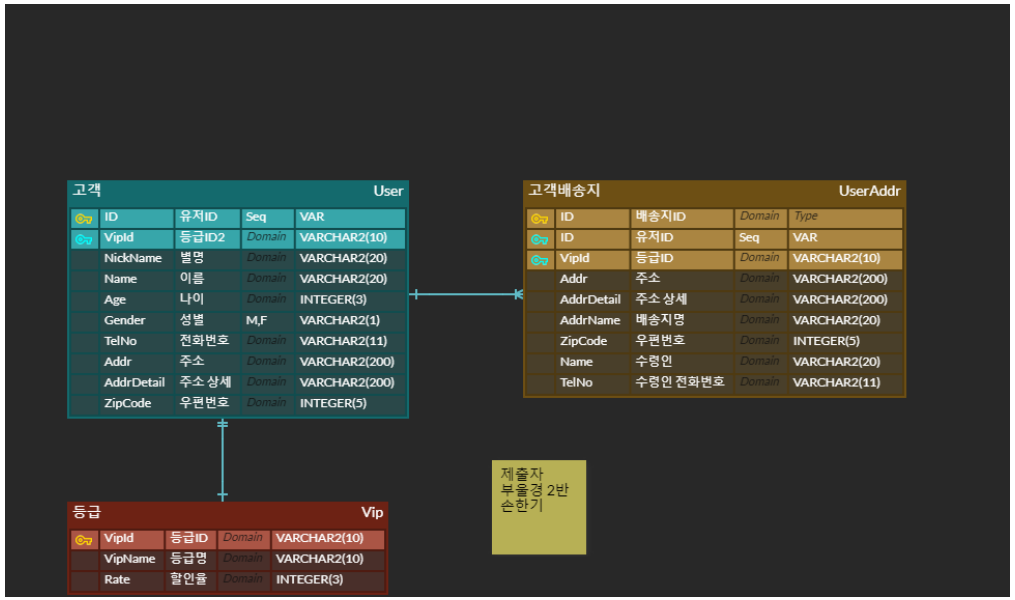
1) 관계형 데이터 베이스

2) Google colab 얼굴인식

Contents

2. 선택 프로젝트 소개

1) 관계형 데이터 베이스



Contents

3. 선택14 얼굴인식 프로젝트

```
image_path = "/gdrive/My Drive/gg1.jpg"
```

```
image = fr.load_image_file(image_path)
face_locations = fr.face_locations(image)
```

```
for (top, right, bottom, left) in face_locations:
  cv2.rectangle(image, (left, top), (right, bottom), (0,255,0), 1)
```

```
# 이미지 버퍼 출력
plt.rcParams['figure.figsize'] = (16,16)
plt.imshow(image)
plt.show()
```

Contents

3. 선택14 얼굴인식 프로젝트

Face Recognition from Multi Angled Images(Rohan Naik, Kalpesh Lad, (IJERT) - 2015)

IV. PROPOSED APPROACH FOR FACE RECOGNITION

Our goal is to identify people even with their multi angled or side profile views. That requires efficient frameworks that construct the frontal face construction from side view single face image. In this approach the input image having a face without external accessories.

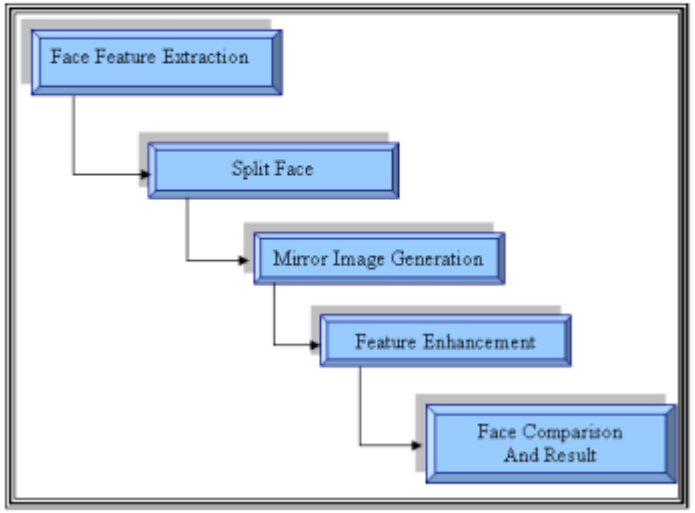


Figure 6. Block diagram of Face Recognition Process

A. Face Feature Extraction

In input image, face feature like nose, left eye, right eye, lips and mouth should be fully or partially visible. To extract these features we use Viola-Jones method. All human faces share some similar properties, so using "Haar features" obtain characteristic of human face based on subsequent constraint. The eyes region is darker than the upper-cheeks. The nose bridge region is brighter than the eyes. Size of eyes and nose seems bridge region. These features are applied onto a face for human any side.

B. Split Face

Using Viola-Jones method, it provide coordinate of nose and mouth. Taken center point of nose and mouth using "Haar feature" will mark virtual line that divides face in two parts. If face orientation is not very straight then virtual line's angle may be changed, that will not exactly 90. Choose finest part that contains clarification feature of human face as that facilitate to build 2D face.

C. Mirror Image Generation

It generates frontal face image that is constructed using side view and creating its mirror image. Here in XY plane value of Y coordinate remain same but X value gets change based on wideness of face that was splits on above phase. In merge image, there may be noise in nose, mouth and eye.

D. Feature Enhancement

Noticeably it's not always possible that angle of face is 45 angles. So during 2D face creation there can be some noise in final image and it require to have smoother face feature like nose mouth and eye. For that we have use Local Binary Patterns (LBP) and nearest neighbor pixel intensity method along with performing scaling or shifting transformation.

E. Face Comparision And Result

After Successful creation of 2D face image, will have to compare image with dataset image. Here we used feature-based (structural) method that compare face boundary along with local feature of human face.

Contents

3. 선택14 얼굴인식 프로젝트

Tracking Face in a Video FileFarah Saad Al (Mukhtar, Scientific International Conference,2017)

4. Kanade Lucas Tomasi Algorithm

Kanade-Lucas-Tomasi (KLT) algorithm tracking approach calculate the movements of object in consecutive video frames when the image brightness constancy constraint is fulfilled and image movement is small, [13].

The KLT algorithm is used here for tracking human faces continuously in a video frame. This method is accomplished by them finding the parameters that allow the reduction in dissimilarity measurements between feature points that are related to original translational model, [6].

Face tracking method is categorized into two parts:

1. Detect a face.
2. Identify facial features to track.

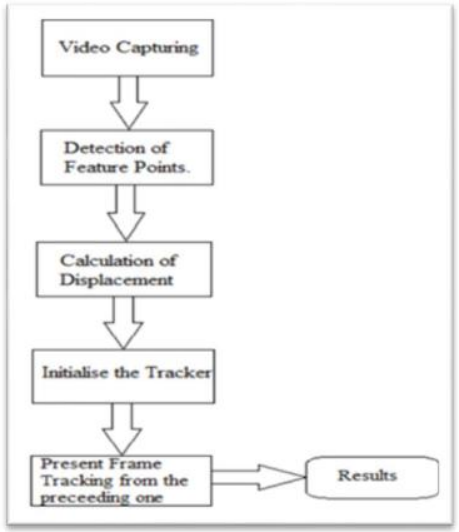


Figure (7) KLT Algorithm Flow Chart, [6].

4.2 Identify facial features to track:

The KLT algorithm detects a set of object points across the video frames. Once the detection trace the face, the next step detects feature points that can be constantly tracked. The basic information has now been established to solve the displacement d of a feature from one window to the next, [12]. For simplicity, we redefine the second window as $B(x) = I(x,y,t+)$ and the first window as $A(x-d) = I(x-d) = I(x-\Delta x, y-\Delta y, t)$ where $x = (x,y)$. The relationship between the two images is given by:

$$B(x) = A(x - d) + (x) \quad \ldots(4)$$

where (x) is a noise function caused by interference. An error function which has to be minimized in order to minimize the effect of noise:

$$€ = [A(x-d) - B(x)]\,[A(x - d) - B(x)]\,wdx \quad \ldots(5)$$

As we see € is a quadratic equation of d. To minimize €, we need to differentiate € with d and set the result to 0.

$$d€/dd = [A(x) - B(x) - d]\,gwdA = 0 \quad \ldots(6)$$

where A refers to area of window W. If the terms of 4 are rearranged and if we use the fact that $(d)g = (g)d$ it follows that:

$$(g\,wdA)d = [A(x) - B(x)]\,gwdA \quad \ldots(7)$$

Using a similar derivation as for the KLT, Shi and Tomasi showed that the search can be performed using the formula $T_Z = a$, where T_a matrix of gradients is, Z, is a vector of affine coefficients and a, is a vector of affine coefficients and $\nabla d = e$

$$I(x; y; t + ¢t) = I(x + ¢x; y + ¢y; t) \quad \ldots(8)$$

Let:

$$x = (x; y)T \text{ and } v = (¢x; ¢y) \quad \ldots(9)$$

In the presence of image noise

$$rI(x; t + ¢t) = I(x + d; t) + r \quad \ldots(10)$$

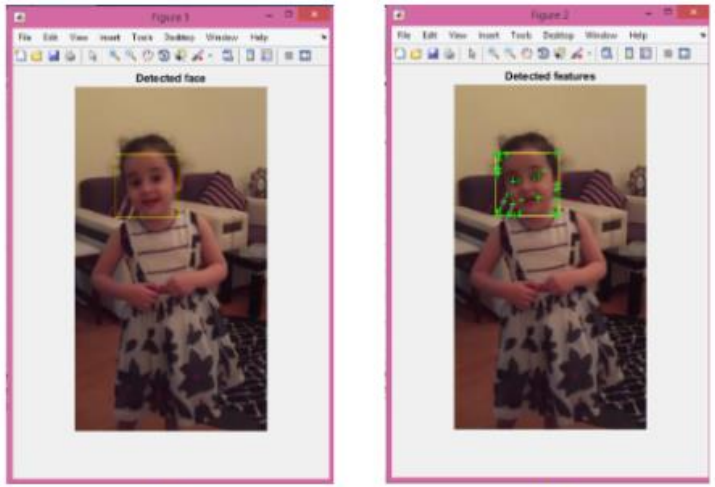
KLT will compute the displacement vector d that minimizes the following error [14]

$$r = XW(I(x + d; t) ¡ I(x; t + ¢t)) \quad \ldots(11)$$

Contents

3. 선택14 얼굴인식 프로젝트

Tracking Face in a Video FileFarah Saad Al (Mukhtar, Scientific International Conference,2017)



(a) (b)

Figure (8) (a) Detect the face; (b) Identify facial features to track.

(a) (b) (c) (d)

Figure (9) a,b,c,d Track the face.

Contents

3. 선택14 얼굴인식 프로젝트

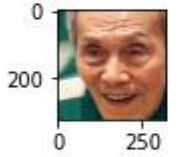
Contents

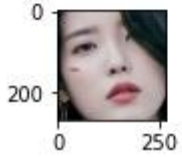
3. 선택14 얼굴인식 프로젝트

```
# Req-8
plt.rcParams["figure.figsize"] = (1,1)

# 이미지 파일을 로드 하여 known_person_list 생성
known_person_list = []
known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p1.jpg"))
known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p2.jpg"))
known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p3.jpg"))
```

```
# 얼굴을 인식을 하여 감지된 부분을 잘라낸 다음 known_face_list 에 저장
Known_face_list = []
Known_face_list.append(fr.load_image_file("/gdrive/My Drive/colab/p4.jpg"))
for person in known_person_list:

  # 얼굴좌표를 확인 후 잘라낸다
  top, right, bottom, left = fr.face_locations(person)[0]
  face_image = person[top:bottom, left:right]

  # known_face_list 에 잘라낸 face_image를 저장
  Known_face_list.append(face_image)
```

```
[14]  # known_face_list 에 저장된 얼굴들 출력
      for face in Known_face_list:
        plt.imshow(face)
        plt.show()
```





Contents

3. 선택14 얼굴인식 프로젝트

```
[15]  # 새로운 사진을 열어서 unknown face 로 저장
      unkonwn_person = fr.load_image_file("/gdrive/My Drive/colab/unknown.jpg")
      top, right, bottom, left = fr.face_locations(unkonwn_person)[0]
      unknown_face = unkonwn_person[top:bottom, left:right]

      # known_face_list 에 잘라낸 face_image를 저장
      plt.title("unknown_face")
      plt.imshow(unknown_face)
      plt.show()
```

face_recognition.api.face_distance(***face_encodings*** , *face_to_compare*) [원천]

얼굴 인코딩 목록이 주어지면 알려진 얼굴 인코딩과 비교하고 각 비교 얼굴에 대한 유클리드 거리를 얻습니다. 거리는 얼굴이 얼마나 유사한지를 알려줍니다.

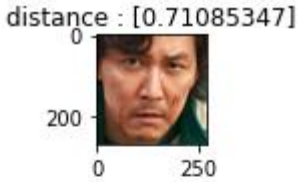
매개변수:
- **face_encodings** - 비교할 얼굴 인코딩 목록
- face_to_compare - 비교할 얼굴 인코딩

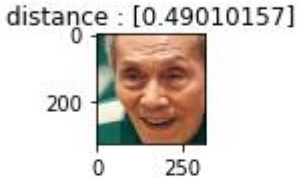
보고: 'faces' 배열과 동일한 순서로 각 면에 대한 거리가 있는 numpy ndarray

```
# 등록된 얼굴 리스트를 비교
for face in Known_face_list:
  enc_known_face = fr.face_encodings(face)
  # print(enc_known_face)

  distance = fr.face_distance(enc_known_face, enc_unknown_face[0])
  plt.title("distance : " + str(distance))
  plt.imshow(face)
  plt.show()
```

Contents

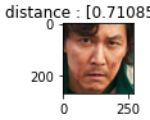
3. 선택14 얼굴인식 프로젝트



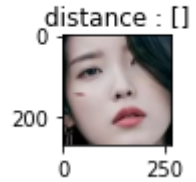



```
print(enc_known_face)
```

```
[array([-9.37026143e-02,  1.22053154e-01,  5.50722666e-02, -8.63128901e-03,
        -1.11417770e-01, -3.48836184e-04, -4.10753712e-02, -1.55098677e-01,
         1.06800668e-01, -8.49139318e-02,  2.00057507e-01, -3.17351967e-02,
        -2.07854286e-01, -1.17437899e-01, -3.14608812e-02,  1.93679884e-01,
        -1.68634146e-01, -1.80081278e-01, -3.12738717e-02, -2.35105306e-03,
         3.21285948e-02,  2.84338631e-02,  2.63148174e-02,  7.98773393e-03,
        -6.79649040e-02, -3.22997928e-01, -5.22756651e-02, -7.54092783e-02,
         3.11728567e-02, -5.85984886e-02, -1.22343779e-01,  2.27723345e-02,
        -2.23417401e-01, -1.06610015e-01,  7.20614940e-02,  8.90366510e-02,
        -1.16916057e-02, -4.42182310e-02,  1.49242342e-01, -2.85591856e-02,
        -2.18356267e-01,  2.19150186e-02,  8.73382390e-02,  2.61924982e-01,
         1.97437033e-01,  2.31458247e-02,  1.00524537e-02, -1.54730052e-01,
         8.83784518e-02, -9.06093866e-02,  1.11331359e-01,  1.26795337e-01,
         5.51826358e-02,  9.13950801e-02,  1.26795471e-03, -1.53849646e-01,
         3.66993882e-02,  8.30609053e-02, -6.20351881e-02,  3.55098546e-02,
         1.59379810e-01, -2.41081454e-02, -2.12288573e-02, -2.89980173e-02,
         2.05865920e-01, -9.80732590e-03, -6.74561188e-02, -2.53727794e-01,
         5.62616885e-02, -9.52294320e-02, -1.17944866e-01,  7.11781383e-02,
        -1.83437973e-01, -1.27670377e-01, -2.99785018e-01,  1.54749602e-02,
         3.82162422e-01,  8.48049372e-02, -2.25217924e-01,  3.09128389e-02,
        -4.50727157e-02, -1.09857731e-02,  1.96616530e-01,  1.40006900e-01,
         4.55974601e-02,  3.24091837e-02, -1.24923773e-01,  2.42310353e-02,
         2.30784297e-01, -3.17062363e-02, -1.00027084e-01,  1.72383785e-01,
        -1.61086172e-02,  1.26389787e-02,  4.40619290e-02,  8.47362652e-02,
        -4.93802354e-02,  8.28161240e-02, -1.77515775e-01, -1.00701727e-01,
         1.13836125e-01, -4.74460423e-04, -3.46737914e-02,  1.64218590e-01,
        -1.57977313e-01,  1.17446661e-01,  8.75830092e-03,  1.14603415e-01,
         3.76829207e-02, -3.98566872e-02, -1.11077987e-01, -2.71215886e-02,
         1.13641009e-01, -1.97018594e-01,  2.57479042e-01,  1.65708601e-01,
         6.50572479e-02,  4.86146361e-02,  1.18239105e-01,  4.34983373e-02,
         2.43059136e-02, -1.91578269e-02, -2.12147325e-01, -4.56020981e-02,
         9.94317234e-02,  3.33902240e-03,  1.39007077e-01,  3.63283269e-02])]
```




```
[]
```



Contents

3. 선택14 얼굴인식 프로젝트

face_recognition.api.face_encodings(*face_image , known_face_locations=None , num_jitters=1 , model='small'*) [원천]

이미지가 주어지면 이미지의 각 얼굴에 대해 128차원 얼굴 인코딩을 반환합니다.

매개변수:

- **face_image** – 하나 이상의 얼굴을 포함하는 이미지
- **known_face_locations** - 선택 사항 - 이미 알고 있는 각 면의 경계 상자입니다.
- **num_jitters** – 인코딩을 계산할 때 얼굴을 다시 샘플링하는 횟수입니다. 높을수록 정확하지만 느림(예: 100은 100배 느림)
- **model** – 선택 사항 – 사용할 모델입니다. "large" 또는 "small"(기본값)은 5포인트만 반환하지만 더 빠릅니다.

보고: 128차원 얼굴 인코딩 목록(이미지의 각 얼굴에 대해 하나씩)

 # 얼굴인식을 못하기 때문!(원본파일로 넣어준다)

Contents

3. 선택14 얼굴인식 프로젝트

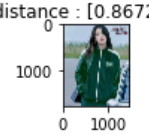
 # 얼굴인식을 못하기 때문!(원본파일로 넣어준다)

```
[12]  # Req-8
      plt.rcParams["figure.figsize"] = (1,1)

      # 이미지 파일을 로드 하여 known_person_list 생성
      known_person_list = []
      known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p1.jpg"))
      known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p2.jpg"))
      known_person_list.append(fr.load_image_file("/gdrive/My Drive/colab/p3.jpg"))
```

```
      # 얼굴을 인식을 하여 감지된 부분을 잘라낸 다음 known_face_list 에 저장
      Known_face_list = []
      Known_face_list.append(fr.load_image_file("/gdrive/My Drive/colab/p4.jpg"))
      for person in known_person_list:

        # 얼굴좌표를 확인 후 잘라낸다
        top, right, bottom, left = fr.face_locations(person)[0]
        face_image = person[top:bottom, left:right]

        # known_face_list 에 잘라낸 face_image를 저장
        Known_face_list.append(face_image)
```

```
[array([-0.0890424 ,  0.09433681,  0.03409908, -0.11403722, -0.11935287,
        0.01478769, -0.07777508, -0.12667419,  0.12208377, -0.1918841 ,
        0.17199694, -0.11374497, -0.22161837, -0.03603877, -0.06998833,
        0.2409479 , -0.15474039, -0.14458737,  0.00529955, -0.00116163,
        0.08185569,  0.05277438, -0.03769866,  0.06958115, -0.05987198,
       -0.32858992, -0.13260803, -0.02647047, -0.10590751, -0.04681502,
       -0.02554895,  0.08662975, -0.1505256 , -0.00898757,  0.03346893,
        0.14409795,  0.02964243, -0.0876739 ,  0.07261015,  0.00801483,
       -0.25576508,  0.08498754,  0.14236969,  0.26776874,  0.13735852,
       -0.06213002, -0.01015154, -0.13847153,  0.10964894, -0.15574241,
       -0.0212263 ,  0.0828068 ,  0.03330866,  0.02518047,  0.07805018,
       -0.079217  ,  0.0724863 ,  0.10655513, -0.07299788, -0.00202417,
        0.13986628, -0.07603754, -0.00118054, -0.13565893,  0.16382781,
        0.04646973, -0.08086886, -0.24691039,  0.08077797, -0.13736367,
       -0.12459821,  0.08410179, -0.1702349 , -0.15896894, -0.29867837,
        0.03331203,  0.35443857,  0.15164372, -0.16636448,  0.11414991,
        0.0494956 ,  0.00515935,  0.15912303,  0.2039203 ,  0.08046415,
        0.05210287, -0.10080945,  0.04317079,  0.25437915, -0.05855869,
       -0.00914937,  0.21445835, -0.01560993,  0.05756907,  0.02449654,
        0.00190332, -0.07320251,  0.03698622, -0.21392728, -0.00323763,
       -0.02109896, -0.03478204, -0.07564439,  0.11530932, -0.14185479,
        0.09181039, -0.02934203, -0.01046006, -0.04697392,  0.01303829,
       -0.09299578,  0.02186313,  0.12903941, -0.18392456,  0.11523867,
        0.14923894,  0.09076092,  0.05622119,  0.14553806,  0.07482734,
       -0.00474439, -0.04104838, -0.22014794, -0.03030328,  0.10704862,
       -0.05983794,  0.0523726 , -0.02446231])]
```

```
distance : [0.8672138]
```



Contents

4. 후기

이미지를 활용한 머신을 써봐서 굉장히 재미있었다.

배치 사이즈 ,정밀도 등을 조절하면서 여러 사진을 확인 해볼 수 있었다

이미지 처리 관련한 여러 지식을 얻을 수 있었다.