Judul **PRAKTEK I8-FACIAL LANDMARK**

Deskripsi

Pada praktek ini kita akan mencoba mengekstraksi fitur area wajah dengan menggunakan library dlib.

Estimasi waktu 15 menit

Beberapa citra berwarna wajah

Install Instructions for dlib

- ***Download and Install Dlib***

https://sourceforge.net/projects/dclib/

- Extract files in C:/dlib
- Use command prompt to Cd to folder and run "python setup.py install"

Download the pre-trained model here

http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

simpan di folder project Anda

Prerequisite

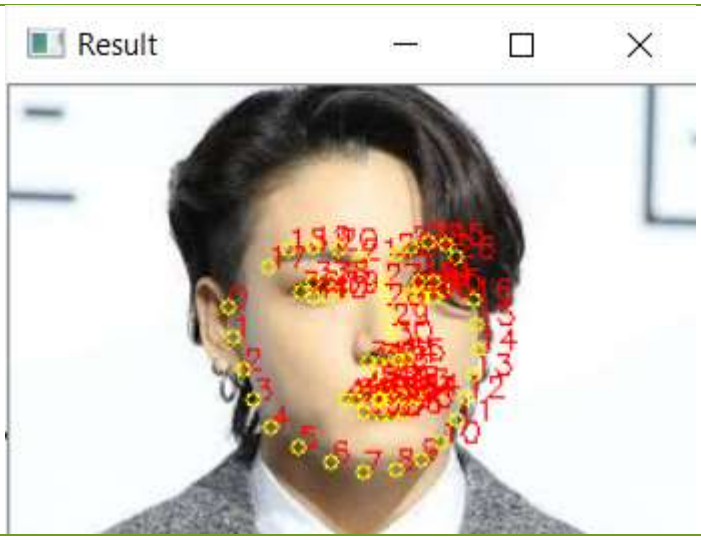
```
import cv2
import dlib
import numpy

PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(PREDICTOR_PATH)
detector = dlib.get_frontal_face_detector()
```

Listing Program

```
class TooManyFaces(Exception):
    pass


class NoFaces(Exception):
    pass
```

```
def get_landmarks(im):
    rects = detector(im, 1)

    if len(rects) > 1:
        raise TooManyFaces
    if len(rects) == 0:
        raise NoFaces

    return numpy.matrix([[p.x, p.y] for p in predictor(im,
rects[0]).parts()])


def annotate_landmarks(im, landmarks):
    im = im.copy()
    for idx, point in enumerate(landmarks):
        pos = (point[0, 0], point[0, 1])
        cv2.putText(im, str(idx), pos,
                    fontFace=cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
                    fontScale=0.4,

                    color=(0, 0, 255))
        cv2.circle(im, pos, 3, color=(0, 255, 255))
    return im


image = cv2.imread('jungkook.jfif')
landmarks = get_landmarks(image)
image_with_landmarks = annotate_landmarks(image, landmarks)

cv2.imshow('Result', image_with_landmarks)
cv2.imwrite('image_with_landmarks.jpg', image_with_landmarks)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result



Tugas

Lakukan analisa terhadap beberapa gambar yang telah di ekstraksi area wajah. Pada fitur ke berapakah objek alis, hidung dan bibir

Judul PRAKTEK I9-SWAP FACE

Deskripsi

Pada praktek ini kita akan mencoba mendeteksi area wajah dengan menggunakan library dlib. Praktek yang akan dilakukan kali ini adalah menukarkan 2 wajah dari 2 image

(Sumber: https://matthewearl.github.io/2015/07/28/switching-eds-with-python/)

Proses dibagi menjadi 4 step:.

1. Detecting facial landmarks.
2. Rotating, scaling, and translating the second image to fit over the first.
3. Adjusting the colour balance in the second image to match that of the first.
4. Blending features from the second image on top of the first.

Estimasi waktu 30 menit

Prerequisite Beberapa citra berwarna wajah

Listing Program

```
import cv2
import dlib
import numpy
from time import sleep
import sys

PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
SCALE_FACTOR = 1
FEATHER_AMOUNT = 11

FACE_POINTS = list(range(17, 68))
MOUTH_POINTS = list(range(48, 61))
RIGHT_BROW_POINTS = list(range(17, 22))
LEFT_BROW_POINTS = list(range(22, 27))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
NOSE_POINTS = list(range(27, 35))
JAW_POINTS = list(range(0, 17))

# Points used to line up the images.
ALIGN_POINTS = (LEFT_BROW_POINTS + RIGHT_EYE_POINTS + LEFT_EYE_POINTS +
                RIGHT_BROW_POINTS + NOSE_POINTS + MOUTH_POINTS)
```

```
# Points from the second image to overlay on the first. The convex hull of
each
# element will be overlaid.
OVERLAY_POINTS = [
    LEFT_EYE_POINTS + RIGHT_EYE_POINTS + LEFT_BROW_POINTS +
RIGHT_BROW_POINTS,
    NOSE_POINTS + MOUTH_POINTS,
]

# Amount of blur to use during colour correction, as a fraction of the
# pupillary distance.
COLOUR_CORRECT_BLUR_FRAC = 0.6

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)


class TooManyFaces(Exception):
    pass


class NoFaces(Exception):
    pass


def get_landmarks(im):
    # Returns facial landmarks as (x,y) coordinates
    rects = detector(im, 1)

    if len(rects) > 1:
        raise TooManyFaces
    if len(rects) == 0:
        raise NoFaces

    return numpy.matrix([[p.x, p.y] for p in predictor(im,
rects[0]).parts()])


def annotate_landmarks(im, landmarks):
    # Overlays the landmark points on the image itself

    im = im.copy()
    for idx, point in enumerate(landmarks):
        pos = (point[0, 0], point[0, 1])
        cv2.putText(im, str(idx), pos,
                    fontFace=cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
                    fontScale=0.4,
                    color=(0, 0, 255))
        cv2.circle(im, pos, 3, color=(0, 255, 255))
    return im


def draw_convex_hull(im, points, color):
    points = cv2.convexHull(points)
    cv2.fillConvexPoly(im, points, color=color)


def get_face_mask(im, landmarks):
    im = numpy.zeros(im.shape[:2], dtype=numpy.float64)

    for group in OVERLAY_POINTS:
        draw_convex_hull(im,
                         landmarks[group],
                         color=1)
```

```
    im = numpy.array([im, im, im]).transpose((1, 2, 0))

    im = (cv2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0) > 0) *
1.0
    im = cv2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0)

    return im


def transformation_from_points(points1, points2):


    points1 = points1.astype(numpy.float64)
    points2 = points2.astype(numpy.float64)

    c1 = numpy.mean(points1, axis=0)
    c2 = numpy.mean(points2, axis=0)
    points1 -= c1
    points2 -= c2

    s1 = numpy.std(points1)
    s2 = numpy.std(points2)
    points1 /= s1
    points2 /= s2

    U, S, Vt = numpy.linalg.svd(points1.T * points2)

    R = (U * Vt).T

    return numpy.vstack([numpy.hstack(((s2 / s1) * R,
                                       c2.T - (s2 / s1) * R * c1.T)),
                         numpy.matrix([0., 0., 1.])])


def read_im_and_landmarks(image):
    im = image
    im = cv2.resize(im, None, fx=1, fy=1, interpolation=cv2.INTER_LINEAR)
    im = cv2.resize(im, (im.shape[1] * SCALE_FACTOR,
                         im.shape[0] * SCALE_FACTOR))
    s = get_landmarks(im)

    return im, s


def warp_im(im, M, dshape):
    output_im = numpy.zeros(dshape, dtype=im.dtype)
    cv2.warpAffine(im,
                   M[:2],
                   (dshape[1], dshape[0]),
                   dst=output_im,
                   borderMode=cv2.BORDER_TRANSPARENT,
                   flags=cv2.WARP_INVERSE_MAP)
    return output_im


def correct_colours(im1, im2, landmarks1):
    blur_amount = COLOUR_CORRECT_BLUR_FRAC * numpy.linalg.norm(
        numpy.mean(landmarks1[LEFT_EYE_POINTS], axis=0) -
        numpy.mean(landmarks1[RIGHT_EYE_POINTS], axis=0))
    blur_amount = int(blur_amount)
    if blur_amount % 2 == 0:
        blur_amount += 1
    im1_blur = cv2.GaussianBlur(im1, (blur_amount, blur_amount), 0)
    im2_blur = cv2.GaussianBlur(im2, (blur_amount, blur_amount), 0)

    # Avoid divide-by-zero errors.
```

```
    im2_blur += (128 * (im2_blur <= 1.0)).astype(im2_blur.dtype)

    return (im2.astype(numpy.float64) * im1_blur.astype(numpy.float64) /
            im2_blur.astype(numpy.float64))


def swappy(image1, image2):
    im1, landmarks1 = read_im_and_landmarks(image1)
    im2, landmarks2 = read_im_and_landmarks(image2)

    M = transformation_from_points(landmarks1[ALIGN_POINTS],
                                   landmarks2[ALIGN_POINTS])

    mask = get_face_mask(im2, landmarks2)
    warped_mask = warp_im(mask, M, im1.shape)
    combined_mask = numpy.max([get_face_mask(im1, landmarks1),
warped_mask],
                              axis=0)

    warped_im2 = warp_im(im2, M, im1.shape)
    warped_corrected_im2 = correct_colours(im1, warped_im2, landmarks1)

    output_im = im1 * (1.0 - combined_mask) + warped_corrected_im2 *
combined_mask
    cv2.imwrite('output.jpg', output_im)
    image = cv2.imread('output.jpg')
    return image


## Enter the paths to your input images here
image1 = cv2.imread('Trump.jpg')
image2 = cv2.imread('siwon.jpg')

swapped = swappy(image1, image2)
cv2.imshow('Face Swap 1', swapped)

swapped = swappy(image2, image1)
cv2.imshow('Face Swap 2', swapped)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Result

Judul	**PRAKTEK I10-SWAP FACE REAL TIME**
Deskripsi	Pada praktek ini kita akan memodifikasi praktek I9 swap face namun dilakukan secara realtime
Estimasi waktu	30 menit
Prerequisite	Beberapa citra berwarna wajah

Listing Program

```
import cv2
import dlib
import numpy
from time import sleep
import sys

## Our pretrained model that predicts the rectangles that correspond to the
facial features of a face
PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
SCALE_FACTOR = 1
FEATHER_AMOUNT = 11

FACE_POINTS = list(range(17, 68))
MOUTH_POINTS = list(range(48, 61))
RIGHT_BROW_POINTS = list(range(17, 22))
LEFT_BROW_POINTS = list(range(22, 27))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
NOSE_POINTS = list(range(27, 35))
JAW_POINTS = list(range(0, 17))

# Points used to line up the images.
ALIGN_POINTS = (LEFT_BROW_POINTS + RIGHT_EYE_POINTS + LEFT_EYE_POINTS +
                RIGHT_BROW_POINTS + NOSE_POINTS + MOUTH_POINTS)

# Points from the second image to overlay on the first. The convex hull of
each
# element will be overlaid.
OVERLAY_POINTS = [
    LEFT_EYE_POINTS + RIGHT_EYE_POINTS + LEFT_BROW_POINTS +
RIGHT_BROW_POINTS,
    NOSE_POINTS + MOUTH_POINTS,
]

# Amount of blur to use during colour correction, as a fraction of the
# pupillary distance.
COLOUR_CORRECT_BLUR_FRAC = 0.6
cascade_path = 'haarcascade_frontalface_default.xml'
cascade = cv2.CascadeClassifier(cascade_path)
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)


def get_landmarks(im, dlibOn):
    if (dlibOn == True):
        rects = detector(im, 1)
        if len(rects) > 1:
            return "error"
        if len(rects) == 0:
            return "error"
        return numpy.matrix([[p.x, p.y] for p in predictor(im,
rects[0]).parts()])

    else:
```

```
        rects = cascade.detectMultiScale(im, 1.3, 5)
        if len(rects) > 1:
            return "error"
        if len(rects) == 0:
            return "error"
        x, y, w, h = rects[0]
        rect = dlib.rectangle(x, y, x + w, y + h)
        return numpy.matrix([[p.x, p.y] for p in predictor(im,
rect).parts()])


def annotate_landmarks(im, landmarks):
    im = im.copy()
    for idx, point in enumerate(landmarks):
        pos = (point[0, 0], point[0, 1])
        cv2.putText(im, str(idx), pos,
                    fontFace=cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
                    fontScale=0.4,
                    color=(0, 0, 255))
        cv2.circle(im, pos, 3, color=(0, 255, 255))
    return im


def draw_convex_hull(im, points, color):
    points = cv2.convexHull(points)
    cv2.fillConvexPoly(im, points, color=color)


def get_face_mask(im, landmarks):
    im = numpy.zeros(im.shape[:2], dtype=numpy.float64)

    for group in OVERLAY_POINTS:
        draw_convex_hull(im,
                         landmarks[group],
                         color=1)

    im = numpy.array([im, im, im]).transpose((1, 2, 0))

    im = (cv2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0) > 0) *
1.0
    im = cv2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0)

    return im


def transformation_from_points(points1, points2):


    points1 = points1.astype(numpy.float64)
    points2 = points2.astype(numpy.float64)

    c1 = numpy.mean(points1, axis=0)
    c2 = numpy.mean(points2, axis=0)
    points1 -= c1
    points2 -= c2

    s1 = numpy.std(points1)
    s2 = numpy.std(points2)
    points1 /= s1
    points2 /= s2

    U, S, Vt = numpy.linalg.svd(points1.T * points2)


    R = (U * Vt).T
```

```
    return numpy.vstack([numpy.hstack(((s2 / s1) * R,
                                       c2.T - (s2 / s1) * R * c1.T)),
                         numpy.matrix([0., 0., 1.])])


def read_im_and_landmarks(fname):
    im = cv2.imread(fname, cv2.IMREAD_COLOR)
    im = cv2.resize(im, None, fx=0.35, fy=0.35,
interpolation=cv2.INTER_LINEAR)
    im = cv2.resize(im, (im.shape[1] * SCALE_FACTOR,
                         im.shape[0] * SCALE_FACTOR))
    s = get_landmarks(im, dlibOn)

    return im, s


def warp_im(im, M, dshape):
    output_im = numpy.zeros(dshape, dtype=im.dtype)
    cv2.warpAffine(im,
                   M[:2],
                   (dshape[1], dshape[0]),
                   dst=output_im,
                   borderMode=cv2.BORDER_TRANSPARENT,
                   flags=cv2.WARP_INVERSE_MAP)
    return output_im


def correct_colours(im1, im2, landmarks1):
    blur_amount = COLOUR_CORRECT_BLUR_FRAC * numpy.linalg.norm(
        numpy.mean(landmarks1[LEFT_EYE_POINTS], axis=0) -
        numpy.mean(landmarks1[RIGHT_EYE_POINTS], axis=0))
    blur_amount = int(blur_amount)
    if blur_amount % 2 == 0:
        blur_amount += 1
    im1_blur = cv2.GaussianBlur(im1, (blur_amount, blur_amount), 0)
    im2_blur = cv2.GaussianBlur(im2, (blur_amount, blur_amount), 0)

    # Avoid divide-by-zero errors.
    im2_blur += (128 * (im2_blur <= 1.0)).astype(im2_blur.dtype)

    return (im2.astype(numpy.float64) * im1_blur.astype(numpy.float64) /
            im2_blur.astype(numpy.float64))


def face_swap(img, name):
    s = get_landmarks(img, True)

    if (s == "error"):
        print("No or too many faces")
        return img

    im1, landmarks1 = img, s
    im2, landmarks2 = read_im_and_landmarks(name)

    M = transformation_from_points(landmarks1[ALIGN_POINTS],
                                   landmarks2[ALIGN_POINTS])

    mask = get_face_mask(im2, landmarks2)
    warped_mask = warp_im(mask, M, im1.shape)
    combined_mask = numpy.max([get_face_mask(im1, landmarks1),
warped_mask],
                              axis=0)

    warped_im2 = warp_im(im2, M, im1.shape)

    warped_corrected_im2 = correct_colours(im1, warped_im2, landmarks1)
```

```
    output_im = im1 * (1.0 - combined_mask) + warped_corrected_im2 *
combined_mask

    # output_im is no longer in the expected OpenCV format so we use openCV
    # to write the image to diks and then reload it
    cv2.imwrite('output.jpg', output_im)
    image = cv2.imread('output.jpg')

    frame = cv2.resize(image, None, fx=1.5, fy=1.5,
interpolation=cv2.INTER_LINEAR)

    return image


cap = cv2.VideoCapture(0)

# Name is the image we want to swap onto ours
# dlibOn controls if use dlib's facial landmark detector (better)
# or use HAAR Cascade Classifiers (faster)

filter_image = "images/Trump.jpg"  ### Put your image here!
dlibOn = False

while True:
    ret, frame = cap.read()

    # Reduce image size by 75% to reduce processing time and improve
framerates
    frame = cv2.resize(frame, None, fx=0.75, fy=0.75,
interpolation=cv2.INTER_LINEAR)

    # flip image so that it's more mirror like
    frame = cv2.flip(frame, 1)

    cv2.imshow('Our Amazing Face Swapper', face_swap(frame, filter_image))

    if cv2.waitKey(1) == 13:  # 13 is the Enter Key
        break

cap.release()
cv2.destroyAllWindows()
```

Judul	**PRAKTEK I11-YAWN DETECTION**
Deskripsi	Pada praktek ini kita akan memanfaatkan library dlib untuk yawn detection sebagai indikasi mengantuk pada seseorang
Estimasi waktu	30 menit
Prerequisite	Kamera/webcam

Listing Program

```
import cv2
import dlib
import numpy as np

PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(PREDICTOR_PATH)

detector = dlib.get_frontal_face_detector()


def get_landmarks(im):
    rects = detector(im, 1)

    if len(rects) > 1:
        return "error"
    if len(rects) == 0:
        return "error"
    return np.matrix([[p.x, p.y] for p in predictor(im, rects[0]).parts()])


def annotate_landmarks(im, landmarks):
    im = im.copy()
    for idx, point in enumerate(landmarks):
        pos = (point[0, 0], point[0, 1])
        cv2.putText(im, str(idx), pos,
                    fontFace=cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
                    fontScale=0.4,
                    color=(0, 0, 255))
        cv2.circle(im, pos, 3, color=(0, 255, 255))
    return im


def top_lip(landmarks):
    top_lip_pts = []
    for i in range(50, 53):
        top_lip_pts.append(landmarks[i])
    for i in range(61, 64):
        top_lip_pts.append(landmarks[i])
    top_lip_all_pts = np.squeeze(np.asarray(top_lip_pts))
    top_lip_mean = np.mean(top_lip_pts, axis=0)
    return int(top_lip_mean[:, 1])


def bottom_lip(landmarks):
    bottom_lip_pts = []
    for i in range(65, 68):
        bottom_lip_pts.append(landmarks[i])
    for i in range(56, 59):
        bottom_lip_pts.append(landmarks[i])
    bottom_lip_all_pts = np.squeeze(np.asarray(bottom_lip_pts))
    bottom_lip_mean = np.mean(bottom_lip_pts, axis=0)
    return int(bottom_lip_mean[:, 1])
```

```
def mouth_open(image):
    landmarks = get_landmarks(image)

    if landmarks == "error":
        return image, 0

    image_with_landmarks = annotate_landmarks(image, landmarks)
    top_lip_center = top_lip(landmarks)
    bottom_lip_center = bottom_lip(landmarks)
    lip_distance = abs(top_lip_center - bottom_lip_center)
    return image_with_landmarks, lip_distance

    # cv2.imshow('Result', image_with_landmarks)
    # cv2.imwrite('image_with_landmarks.jpg',image_with_landmarks)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()


cap = cv2.VideoCapture(0)
yawns = 0
yawn_status = False

while True:
    ret, frame = cap.read()
    image_landmarks, lip_distance = mouth_open(frame)

    prev_yawn_status = yawn_status

    if lip_distance > 25:
        yawn_status = True

        cv2.putText(frame, "Subject is Yawning", (50, 450),
                    cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)

        output_text = " Yawn Count: " + str(yawns + 1)

        cv2.putText(frame, output_text, (50, 50),
                    cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 127), 2)

    else:
        yawn_status = False

    if prev_yawn_status == True and yawn_status == False:
        yawns += 1

    cv2.imshow('Live Landmarks', image_landmarks)
    cv2.imshow('Yawn Detection', frame)

    if cv2.waitKey(1) == 13:  # 13 is the Enter Key
        break

cap.release()
cv2.destroyAllWindows()
```

DAFTAR PUSTAKA

Kadir, A., & Susanto, A. (2013). *Teori dan Aplikasi Pengolahan Citra Digital.* Yogyakarta: Andi Publisher.

Munir, R. (2004). *Pengolahan Citra Digital dengan Pendekatan Algoritmik.* Bandung: Informatika.

Putra, D. (2010). *Pengolahan Citra Digital.* Yogyakarta: Andi Publisher.