	JOBSHEET PENGELOLAAN CITRA DIGITAL	Nomor: 04b/pcd/tid3/2023
		Kode M.K. : TI.302
		Pengampu: Dr. Raswa, M. Pd.
		Revisi: 0
Topik Bahasan	Memahami Deteksi Objek	
Target	Mahasiswa dapat: <ol style="list-style-type: none"> 1. memahami apa itu objek, mengapa objek penting, dan lainnya. 2. memahami konsep di balik ObjectDetection. 3. Memahami deteksi objek, mendeteksi objek yang diinginkan (wajah kucing) secara real-time dan terus melacak objek yang sama. 	
Teori	<p>Deteksi Objek</p> <p>Deteksi Objek adalah teknologi komputer yang terkait dengan visi komputer, pemrosesan gambar, dan pembelajaran mendalam yang berhubungan dengan pendeteksian contoh objek dalam gambar dan video. Pendeteksian objek dapat menggunakan sesuatu yang dikenal sebagai haar cascades. Haar Cascade adalah pendekatan berbasis pembelajaran mesin di mana banyak gambar positif dan negatif digunakan untuk melatih classifier. Gambar positif – Gambar-gambar ini berisi gambar-gambar yang kita inginkan untuk diidentifikasi oleh pengklasifikasi kita. Gambar Negatif – Gambar segala sesuatu yang lain, yang tidak mengandung objek yang ingin kita deteksi.</p> <p>Langkah-langkah untuk mengunduh persyaratan di bawah ini:</p> <p>Jalankan perintah berikut di terminal untuk menginstal opencv.</p> <pre>pip instal opencv-python</pre> <p>Jalankan perintah berikut untuk menginstal matplotlib di terminal.</p> <pre>pip instal matplotlib</pre> <p>Untuk mengunduh file dan gambar haar cascade yang digunakan dalam kode di bawah ini sebagai file zip, klik di https://drive.google.com/file/d/1amieL09Q8G5auRC2_I9hwehuqFkKDHwf/view</p> <p>Catatan: Letakkan file XML dan gambar PNG di folder yang sama dengan skrip Python Anda.</p> <p><code>cv2.goodFeaturesToTrack()</code> metode mencari N sudut terkuat pada citra dengan metode Shi-Tomasi. Perhatikan bahwa gambar harus berupa gambar skala abu-abu. Tentukan jumlah sudut yang ingin Anda temukan dan tingkat kualitasnya (yaitu nilai antara 0-1). Ini menunjukkan kualitas sudut minimum di mana setiap orang ditolak. Kemudian berikan jarak Euclidean minimum antara sudut yang terdeteksi.</p> <p>Sintaks : <code>cv2.goodFeaturesToTrack(gambar, maxCorners, qualityLevel, minDistance[, corner[, mask[, blockSize[, useHarrisDetector[, k]]]])</code></p> <p>Gambar deteksi sudut:</p>	



Deteksi Wajah adalah teknologi untuk mengidentifikasi wajah dari gambar. Jika seseorang dapat melihat lebih dekat pada repositori OpenCV, direktori haar cascades menjadi spesifik (di mana OpenCV menyimpan semua pengklasifikasi haar yang telah dilatih sebelumnya untuk mendeteksi berbagai objek, bagian tubuh, dll.), ada dua file:

haarcascade_frontalcatface.xml

haarcascade_frontalcatface_extended.xml

Mengenali Plat Nomor Mobil adalah tugas yang sangat penting untuk sistem keamanan berbasis kamera pengintai. Ekstrak pelat nomor dari gambar menggunakan beberapa teknik penglihatan komputer dan kemudian digunakan Pengenalan Karakter Optik untuk mengenali nomor lisensi.

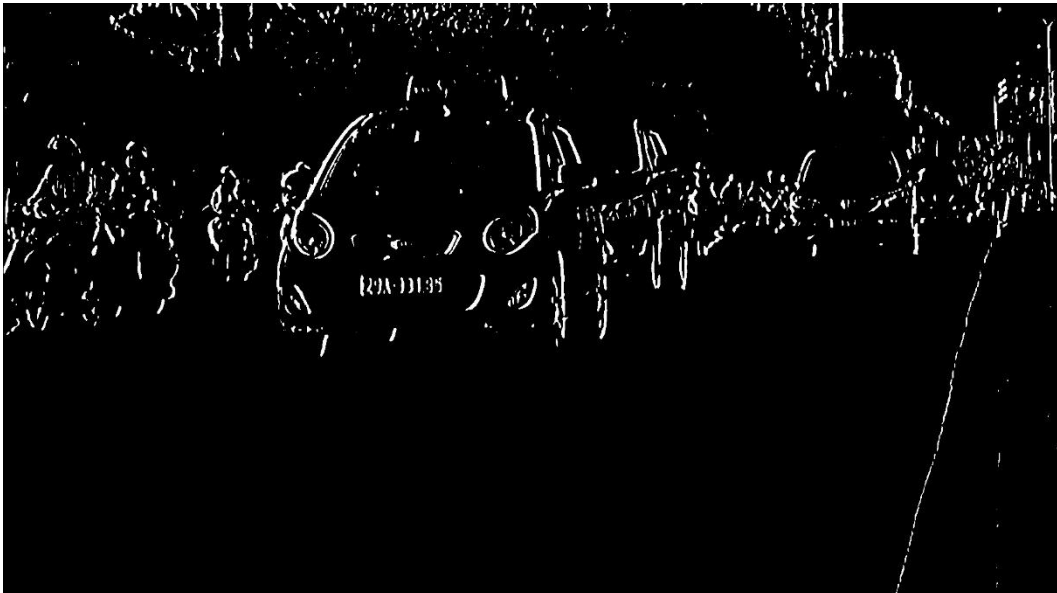
Untuk mengurangi noise kita perlu mengaburkan input Gambar dengan Gaussian Blur dan kemudian mengubahnya menjadi skala abu-abu.



Temukan tepi vertikal pada gambar:



Untuk mengungkap pelat kita harus membuat **binarisasi gambar**. Untuk ini terapkan **Thresholding Otsu** pada gambar tepi vertikal. Dalam metode thresholding lainnya, kita harus memilih nilai threshold untuk binarisasi citra tetapi Thresholding Otsu menentukan nilainya secara otomatis.



Terapkan **Closing Morphological Transformation** pada gambar yang di-threshold. Penutupan berguna untuk mengisi area hitam kecil di antara area putih pada gambar yang di-threshold. Ini mengungkapkan kotak putih persegi panjang pelat nomor.




Untuk mendeteksi plat kita perlu **mencari kontur pada citra**. Penting untuk melakukan binerisasi dan morph pada gambar sebelum menemukan kontur sehingga dapat menemukan kontur yang lebih relevan dan lebih sedikit pada gambar. Jika Anda menggambar semua kontur yang diekstraksi pada gambar asli, akan terlihat seperti ini:



Untuk mengenali karakter pada plat nomor dengan tepat, kita harus menerapkan segmentasi gambar. Langkah pertama itu adalah mengekstrak saluran nilai dari format HSV gambar pelat.

Sekarang terapkan ambang batas adaptif pada gambar saluran nilai pelat untuk membuat biner dan menampilkan karakter. Gambar pelat dapat memiliki kondisi pencahayaan yang berbeda di area yang berbeda, dalam hal ini, adaptive thresholding lebih cocok untuk binari karena menggunakan nilai ambang yang berbeda untuk area yang berbeda berdasarkan kecerahan piksel di area sekitarnya. Setelah binerisasi terapkan operasi bitwise not pada citra untuk menemukan komponen yang terhubung pada citra sehingga kita dapat mengekstrak kandidat karakter. Bangun topeng untuk menampilkan semua komponen karakter dan kemudian temukan kontur di topeng. Setelah mengekstraksi kontur, ambil yang terbesar, temukan persegi panjang pembatasnya dan validasi rasio sisi. Setelah memvalidasi rasio samping, temukan convex hull dari iklan kontur yang digambar pada mask kandidat karakter. Topeng akan terlihat seperti. Sekarang temukan semua kontur di mask

	<p>kandidat karakter dan ekstrak area kontur tersebut dari gambar ambang batas nilai pelat, Anda akan mendapatkan semua karakter secara terpisah.</p> <p>Kode pengembangan diperoleh melalui link berikut: https://github.com/hritik7080/Car-License-Plate-Recognition</p>
Instruksi	<p>Tugas 4b1, tulis kode program:</p> <p>Membuka gambar</p> <pre>import cv2 from matplotlib import pyplot as plt # Opening image img = cv2.imread("image.jpg") # OpenCV opens images as BRG # but we want it as RGB and # we also need a grayscale # version img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Creates the environment # of the picture and shows it plt.subplot(1, 1, 1) plt.imshow(img_rgb) plt.show()</pre> <p>Keluaran:</p>  <p>Kami akan menggunakan detectMultiScale()fungsi OpenCV untuk mengenali tanda besar maupun kecil:</p> <pre># Use minSize because for not # bothering with extra-small # dots that would look like STOP signs found = stop_data.detectMultiScale(img_gray, minSize =(20, 20)) # Don't do anything if there's # no sign amount_found = len(found)</pre>

```

if amount_found != 0:

    # There may be more than one
    # sign in the image
    for (x, y, width, height) in found:

        # We draw a green rectangle around
        # every recognized sign
        cv2.rectangle(img_rgb, (x, y),
                       (x + height, y + width),
                       (0, 255, 0), 5)

```

Tugas 4b2, tulis kode program berikut:

```

import cv2
from matplotlib import pyplot as plt

# Opening image
img = cv2.imread("image.jpg")

# OpenCV opens images as BRG
# but we want it as RGB We'll
# also need a grayscale version
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Use minSize because for not
# bothering with extra-small
# dots that would look like STOP signs
stop_data = cv2.CascadeClassifier('stop_data.xml')

found = stop_data.detectMultiScale(img_gray,
                                    minSize=(20, 20))

# Don't do anything if there's
# no sign
amount_found = len(found)

if amount_found != 0:

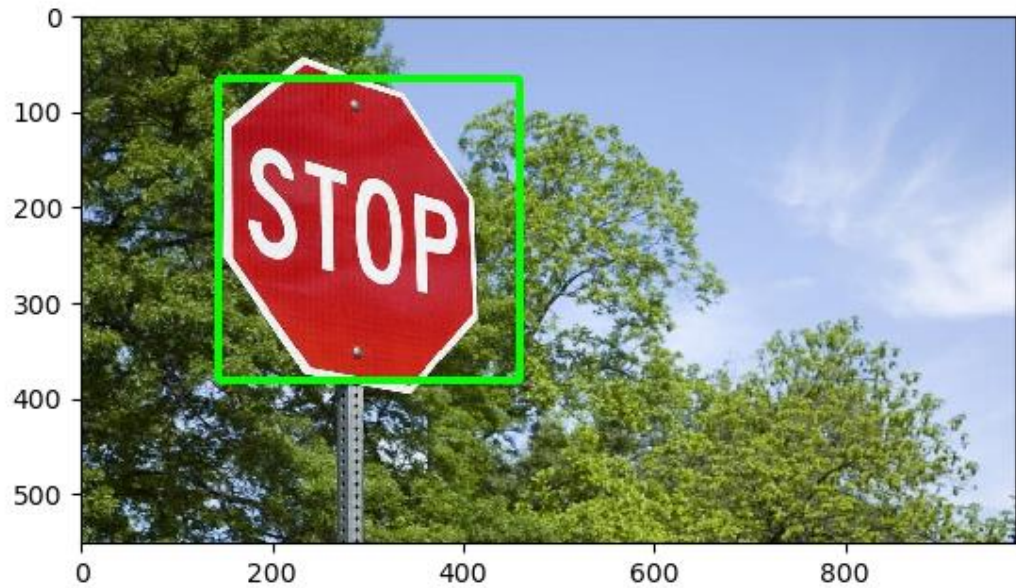
    # There may be more than one
    # sign in the image
    for (x, y, width, height) in found:

        # We draw a green rectangle around
        # every recognized sign
        cv2.rectangle(img_rgb, (x, y),
                       (x + height, y + width),
                       (0, 255, 0), 5)

# Creates the environment of
# the picture and shows it
plt.subplot(1, 1, 1)
plt.imshow(img_rgb)
plt.show()

```


Di bawah ini hasilnya:



Tugas 4b3, tulis kode program berikut:

```
# import the required library
import numpy as np
import cv2
from matplotlib import pyplot as plt

# read the image
img = cv2.imread('corner1.png')

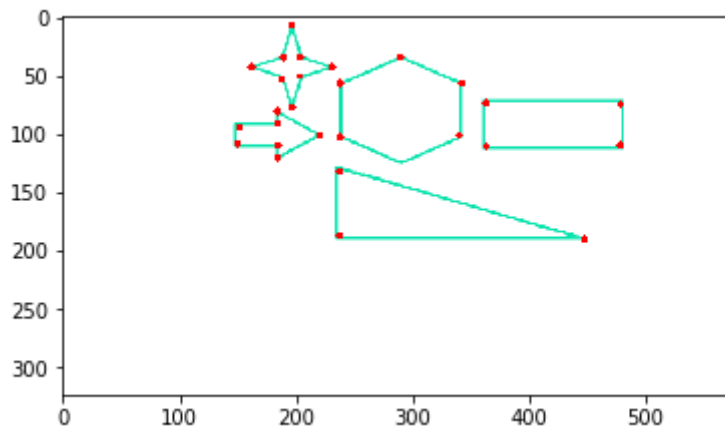
# convert image to gray scale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# detect corners with the goodFeaturesToTrack function.
corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
corners = np.int0(corners)

# we iterate through each corner,
# making a circle at each point that we think is a corner.
for i in corners:
    x, y = i.ravel()
    cv2.circle(img, (x, y), 3, 255, -1)

plt.imshow(img), plt.show()
```

Gambar setelah dideteksi sudut:



Tugas 4b4, tulis kode program berikut:

```
# OpenCV program to detect cat face in real time
# import libraries of python OpenCV
# where its functionality resides
import cv2

# load the required trained XML classifiers
# https://github.com/Itseez/opencv/blob/master/
# data/haarcascades/haarcascade_frontalcatface.xml
# Trained XML classifiers describes some features of some
# object we want to detect a cascade function is trained
# from a lot of positive(faces) and negative(non-faces)
# images.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalcatface.xml')

# capture frames from a camera
cap = cv2.VideoCapture(0)

# loop runs if capturing has been initialized.
while 1:

    # reads frames from a camera
    ret, img = cap.read()

    # convert to gray scale of each frames
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detects faces of different sizes in the input image
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        # To draw a rectangle in a face
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,255,0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

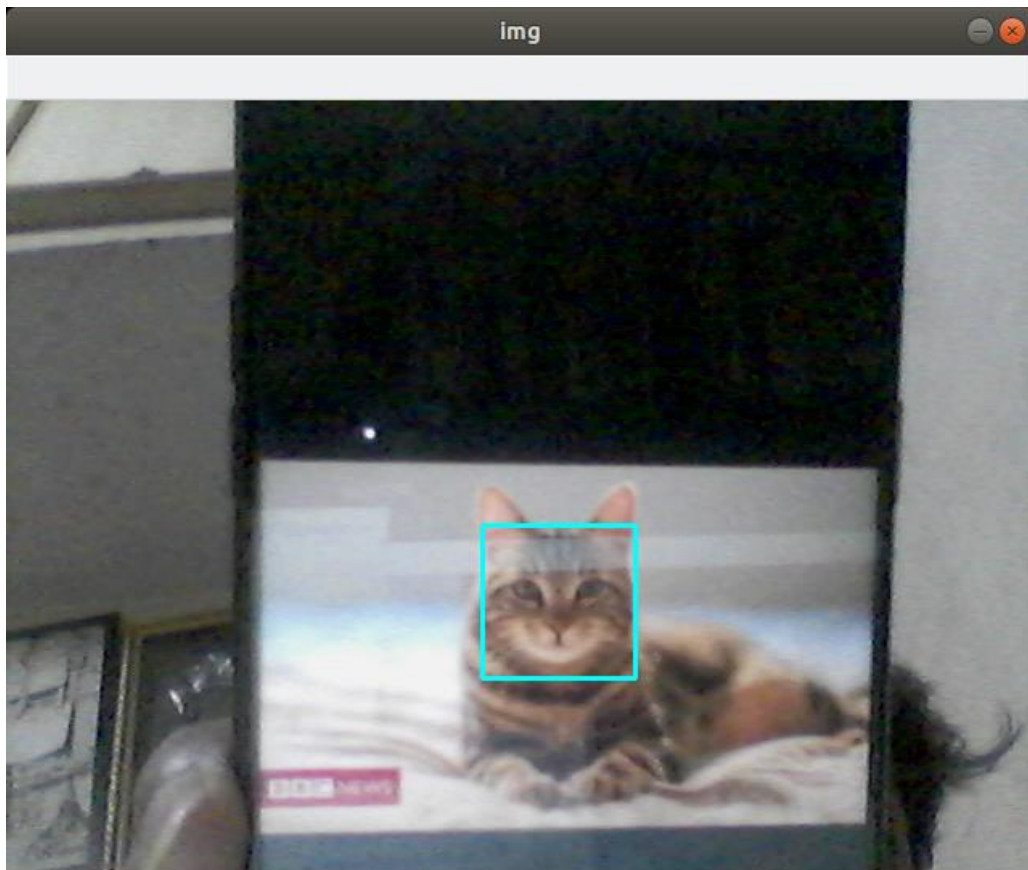
    # Display an image in a window
    cv2.imshow('img',img)

    # Wait for Esc key to stop
    k = cv2.waitKey(30) & 0xff
```



```
if k == 27:  
    break  
  
# Close the window  
cap.release()  
  
# De-allocate any associated memory usage  
cv2.destroyAllWindows()
```

Keluaran:



Tugas 4b5, tulis kode program berikut:

Persyaratan:

opencv-python 3.4.2
numpy 1.17.2
skimage 0.16.2
tensorflow 1.15.0
imutil 0.5.3

contoh:



Keluaran:

29A33185



Kode: Langkah ini dilakukan dengan **metode clean_plate** dan **ratioCheck** dari class **PlateFinder** .

```
def clean_plate(self, plate):

    gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255,
                                   cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)

    _, contours, _ = cv2.findContours(thresh.copy(),
                                       cv2.RETR_EXTERNAL,
                                       cv2.CHAIN_APPROX_NONE)

    if contours:

        areas = [cv2.contourArea(c) for c in contours]

        # index of the largest contour in the
        # areas array
        max_index = np.argmax(areas)

        max_cnt = contours[max_index]
        max_cntArea = areas[max_index]
        x, y, w, h = cv2.boundingRect(max_cnt)

        if not self.ratioCheck(max_cntArea,
                               plate.shape[1],
                               plate.shape[0]):
            return plate, False, None
```


11, 2)

```
thresh = cv2.bitwise_not(thresh)

# resize the license plate region to
# a canonical size
plate_img = imutils.resize(plate_img, width = fixed_width)
thresh = imutils.resize(thresh, width = fixed_width)
bgr_thresh = cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR)

# perform a connected components analysis
# and initialize the mask to store the locations
# of the character candidates
labels = measure.label(thresh, neighbors = 8, background = 0)

charCandidates = np.zeros(thresh.shape, dtype = 'uint8')

# loop over the unique components
characters = []
for label in np.unique(labels):

    # if this is the background label, ignore it
    if label == 0:
        continue
    # otherwise, construct the label mask to display
    # only connected components for the current label,
    # then find contours in the label mask
    labelMask = np.zeros(thresh.shape, dtype = 'uint8')
    labelMask[labels == label] = 255

    cnts = cv2.findContours(labelMask,
                           cv2.RETR_EXTERNAL,
                           cv2.CHAIN_APPROX_SIMPLE)

    cnts = cnts[0] if imutils.is_cv2() else cnts[1]

    # ensure at least one contour was found in the mask
    if len(cnts) > 0:

        # grab the largest contour which corresponds
        # to the component in the mask, then grab the
        # bounding box for the contour
        c = max(cnts, key = cv2.contourArea)
        (boxX, boxY, boxW, boxH) = cv2.boundingRect(c)

        # compute the aspect ratio, solidity, and
        # height ration for the component
        aspectRatio = boxW / float(boxH)
        solidity = cv2.contourArea(c) / float(boxW * boxH)
        heightRatio = boxH / float(plate_img.shape[0])

        # determine if the aspect ratio, solidity,
        # and height of the contour pass the rules
        # tests
        keepAspectRatio = aspectRatio < 1.0
        keepSolidity = solidity > 0.15
        keepHeight = heightRatio > 0.5 and heightRatio < 0.95

        # check to see if the component passes
        # all the tests
        if keepAspectRatio and keepSolidity and keepHeight and boxW > 14:
```

```

        # compute the convex hull of the contour
        # and draw it on the character candidates
        # mask
        hull = cv2.convexHull(c)

        cv2.drawContours(charCandidates, [hull], -1, 255, -1)

_, contours, hier = cv2.findContours(charCandidates,
                                     cv2.RETR_EXTERNAL,
                                     cv2.CHAIN_APPROX_SIMPLE)

if contours:
    contours = sort_cont(contours)

    # value to be added to each dimension
    # of the character
    addPixel = 4
    for c in contours:
        (x, y, w, h) = cv2.boundingRect(c)
        if y > addPixel:
            y = y - addPixel
        else:
            y = 0
        if x > addPixel:
            x = x - addPixel
        else:
            x = 0
        temp = bgr_thresh[y:y + h + (addPixel * 2),
                          x:x + w + (addPixel * 2)]

        characters.append(temp)

    return characters

else:
    return None

class PlateFinder:
    def __init__(self):

        # minimum area of the plate
        self.min_area = 4500

        # maximum area of the plate
        self.max_area = 30000

        self.element_structure = cv2.getStructuringElement(
            shape = cv2.MORPH_RECT, ksize =(22, 3))

    def preprocess(self, input_img):

        imgBlurred = cv2.GaussianBlur(input_img, (7, 7), 0)

        # convert to gray
        gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)

        # sobelX to get the vertical edges
        sobelx = cv2.Sobel(gray, cv2.CV_8U, 1, 0, ksize = 3)

```

```

# otsu's thresholding
ret2, threshold_img = cv2.threshold(sobelx, 0, 255,
                                   cv2.THRESH_BINARY + cv2.THRESH_OTSU)

element = self.element_structure
morph_n_thresholded_img = threshold_img.copy()
cv2.morphologyEx(src = threshold_img,
                 op = cv2.MORPH_CLOSE,
                 kernel = element,
                 dst = morph_n_thresholded_img)

return morph_n_thresholded_img

def extract_contours(self, after_preprocess):

    _, contours, _ = cv2.findContours(after_preprocess,
                                     mode = cv2.RETR_EXTERNAL,
                                     method = cv2.CHAIN_APPROX_NONE)

    return contours

def clean_plate(self, plate):

    gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray,
                                  255,
                                  cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                  cv2.THRESH_BINARY,
                                  11, 2)

    _, contours, _ = cv2.findContours(thresh.copy(),
                                     cv2.RETR_EXTERNAL,
                                     cv2.CHAIN_APPROX_NONE)

    if contours:
        areas = [cv2.contourArea(c) for c in contours]

        # index of the largest contour in the area
        # array
        max_index = np.argmax(areas)

        max_cnt = contours[max_index]
        max_cntArea = areas[max_index]
        x, y, w, h = cv2.boundingRect(max_cnt)
        rect = cv2.minAreaRect(max_cnt)

        if not self.ratioCheck(max_cntArea, plate.shape[1],
                               plate.shape[0]):
            return plate, False, None

        return plate, True, [x, y, w, h]

    else:
        return plate, False, None

def check_plate(self, input_img, contour):

    min_rect = cv2.minAreaRect(contour)

```



```

        if self.validateRatio(min_rect):
            x, y, w, h = cv2.boundingRect(contour)
            after_validation_img = input_img[y:y + h, x:x + w]
            after_clean_plate_img, plateFound, coordinates =
self.clean_plate(
                                                    after_validation_img)

            if plateFound:
                characters_on_plate = self.find_characters_on_plate(
                                                    after_clean_plate_img)

                if (characters_on_plate is not None and
len(characters_on_plate) == 8):
                    x1, y1, w1, h1 = coordinates
                    coordinates = x1 + x, y1 + y
                    after_check_plate_img = after_clean_plate_img

                    return after_check_plate_img, characters_on_plate,
coordinates

            return None, None, None

def find_possible_plates(self, input_img):

    """
    Finding all possible contours that can be plates
    """
    plates = []
    self.char_on_plate = []
    self.corresponding_area = []

    self.after_preprocess = self.preprocess(input_img)
    possible_plate_contours =
self.extract_contours(self.after_preprocess)

    for cnts in possible_plate_contours:
        plate, characters_on_plate, coordinates =
self.check_plate(input_img, cnts)

        if plate is not None:
            plates.append(plate)
            self.char_on_plate.append(characters_on_plate)
            self.corresponding_area.append(coordinates)

    if (len(plates) > 0):
        return plates

    else:
        return None

def find_characters_on_plate(self, plate):

    charactersFound = segment_chars(plate, 400)
    if charactersFound:
        return charactersFound

# PLATE FEATURES

```

```

def ratioCheck(self, area, width, height):

    min = self.min_area
    max = self.max_area

    ratioMin = 3
    ratioMax = 6

    ratio = float(width) / float(height)

    if ratio < 1:
        ratio = 1 / ratio

    if (area < min or area > max) or (ratio < ratioMin or ratio >
ratioMax):
        return False

    return True

def preRatioCheck(self, area, width, height):

    min = self.min_area
    max = self.max_area

    ratioMin = 2.5
    ratioMax = 7

    ratio = float(width) / float(height)

    if ratio < 1:
        ratio = 1 / ratio

    if (area < min or area > max) or (ratio < ratioMin or ratio >
ratioMax):
        return False

    return True

def validateRatio(self, rect):
    (x, y), (width, height), rect_angle = rect

    if (width > height):
        angle = -rect_angle
    else:
        angle = 90 + rect_angle

    if angle > 15:
        return False

    if (height == 0 or width == 0):
        return False

    area = width * height

    if not self.preRatioCheck(area, width, height):
        return False
    else:
        return True

```

Penilaian

Buat laporan hasil pengerjaan jobsheet ini dan dipresentasikan sebagai tugas kelompok.