Fit Ellipses on an Image

—Homework2 for computer vision

Name: Wang Jun

Student ID：3170100186

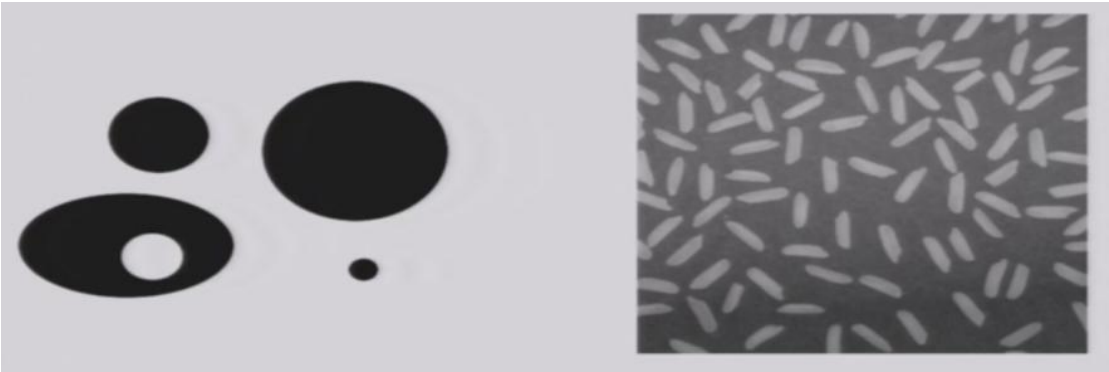
Date: 2021-12-03

Course: Computer Vision

Instructor: Mingli Song

Chapter 1: The purpose and requirements of the experiment

Call OpenCV's CvBox2D and cvFitEllipse2(const CvArr* points) to fit ellipses on an image. Actually, the interfaces provided here are already deprecated. In OpenCV's C++ API, the fitting function is changed to cv:： fitEllipse In Python API, the fitting function is cv2.fitEllipse



Chapter 2: Experimental content and principle

OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

Some functions in CV2

- **Get the image**

cv2.imread:

cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

cv2.cvtColor()

cv2.cvtColor() method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV. We will use some of color space conversion codes below.

cv2.Canny()

cv2.Canny()'s function in OpenCV is used to detect the edges in an image.

- **Get the features**

cv2.findContours：

findContour() function that helps in extracting the contours from the image. It works best on binary images, so we should first apply thresholding techniques, Sobel edges, etc.

cv2.fitEllipse：

Filter by Area – This is to avoid any identification of any small dots present in the image that can be wrongly detected as a circle.

Filter by Circularity – This helps us to identify, shapes that are more similar to a circle.

Other packages

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import os
import sys  # for commandline arguments
import cv2
import random  # random color generator
import logging
import coloredlogs
import numpy as np
```

Chapter 3: Experimental procedure and analysis

Read the image

```
if len(sys.argv)>1:
    imgName = sys.argv[1]
else:
    imgName = "./resources/21.jpg"
```

```
# Or he/she might mistype the file name
if not os.path.exists(imgName):
    log.error(f"Cannot find the file specified: {imgName}")
    return 1
```

Convert the image

Convert the image to rgb 3 channel image, grayscale image and detect edge using Canny operator:

```
image = cv2.imread(imgName)
igray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
irgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
iedge = cv2.Canny(igray, 100, 200)  # using Canny to get edge
return image, irgb, igray, iedge
```

Find contours and fit the contours with ellipses

- Detect contours in an image
  Get minimum area box / best fit ellilpse of contours

```
contours, _ = cv2.findContours(edge, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
ellipses = [None]*len(contours)  # [None, None, None, ...]
```

- All detected contour will have a min area box, but only those with more than 5 points will have a ellipse

```
for index, contour in enumerate(contours):
    if contour.shape[0] > 5:
        ellipses[index] = cv2.fitEllipse(contour)
return contours,ellipses
```

Render the contours and ellipses

```
draw = np.zeros(shape, dtype='uint8')
log.info(f"Drawing on shape: {shape} with type {draw.dtype}")
for index, contour in enumerate(contours):
    color = randcolor()
    log.debug(f"Getting random color: {color}")
    cv2.drawContours(draw, contours, index, color, 1, cv2.LINE_AA)

    # All detected contour will have a min area box
    # But only those with more than 5 points will have a ellipse
    if contour.shape[0] > 5:
        cv2.ellipse(draw, ellipses[index], color, 3, cv2.LINE_AA)

return draw
```

Display results

```
plt.figure("Fitting")
plt.suptitle("Fitting ellipses and finding min-area rectangles", fontweight="bold")
plt.subplot(221)
plt.title("original", fontweight="bold")
plt.imshow(rgb)
plt.subplot(222)
plt.title("grayscale", fontweight="bold")
plt.imshow(gray, cmap="gray")
plt.subplot(223)
plt.title("canny Edge", fontweight="bold")
plt.imshow(edge, cmap="gray")
plt.subplot(224)
plt.title("contour & ellipses ", fontweight="bold")
plt.imshow(drawing)
plt.show()
```

Chapter 4: Experimental environment and operation method

Development environment

- macOS 10.14.6

- python 3.9

- opencv-python 4.5.1.48

Operation mode

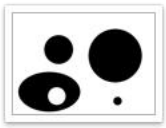
- python hw2.py ./resources/ 或者根据文件夹选择

-python hw2.py 不输入参数默认是./resources 文件夹

Chapter 5: Experimental results

IO Screenshots

INPUT:

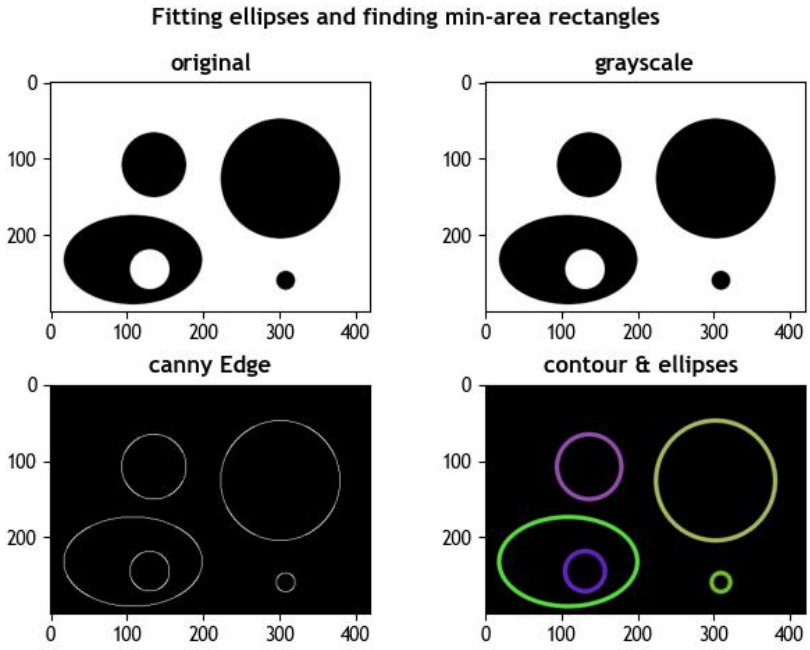


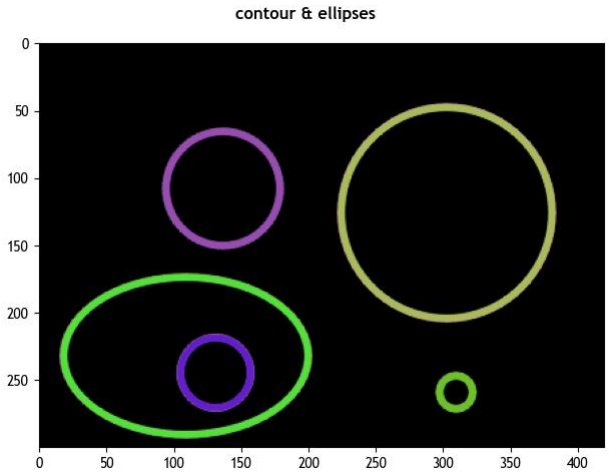
22.jpg

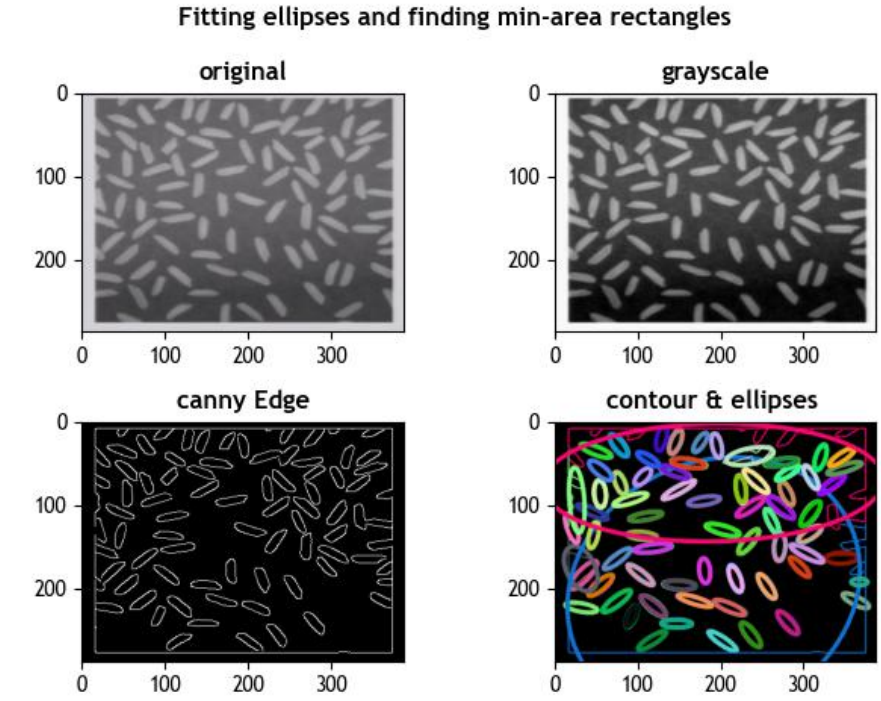
23.jpg

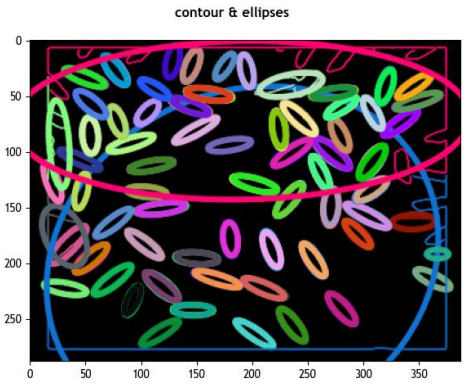
21.jpg

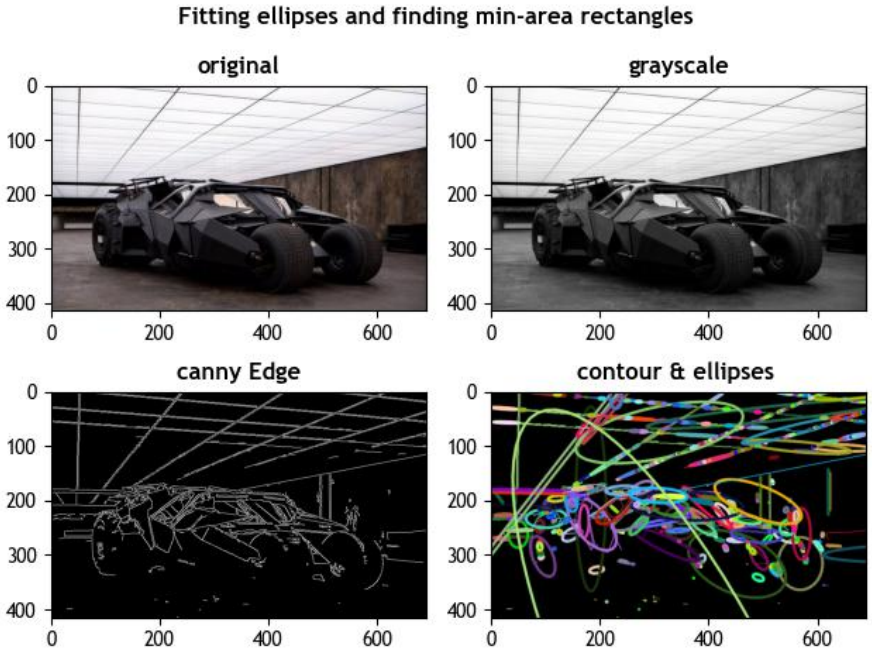
OUTPUT:

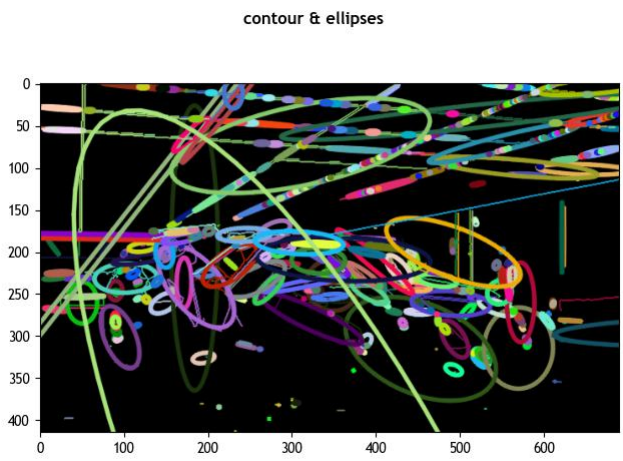





Fitting ellipses and finding min-area rectangles
original
grayscale
canny Edge
contour & ellipses


contour & ellipses


Fitting ellipses and finding min-area rectangles
original
grayscale
canny Edge
contour & ellipses



Chapter 6: Thoughts

OpenCV in python and C++

I use mac-os to finish my work, and find the visual studio in mac is hard to use in C++, so I try to use python instead. And surprisingly, python is much easier than C++ that not only in its readable, but also it did not require you to learn about a new class Mat to be able to operate on images. In a nutshell, for man using a mac os, python would be a better choice, and the deployment of environment is easier.

Although it's a fact that C++ is much faster. But we're not developing a full fledge application here. We're only defining some execution logic to make a toy example to experiment on things and explore around, which will not bring too much performance impact because OpenCV is doing the heavy lifting with C++ implementation in the background for us to enjoy.

The results look okay, but for a well detailed image, a lot of ellipses will pop up. So a possible improvements among this is try to set a lower bound for the radius in the ellipses (like in hough transform)