

# # Homework 2

## ## Basic Info

Student ID: 3180105504

Student Name: Xu Zhen

Instructor Name: Song Mingli

Course Name: Computer Vision

Homework Name: **Fit Ellipses on an Image**

## ## Homework Purposes and Requirements

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`

## ## Homework Principles

We use `Python` to implement this homework.

In this project, we used these packages

```
``` python
1 #!python
2 import matplotlib.pyplot as plt
3 import matplotlib as mpl
4 import os
5 import sys # for commandline arguments
6 import cv2
7 import random
8 import logging
9 import coloredlogs
10 import numpy as np
```
```

Some of them are free since they come with a clean python installation

But `numpy`, `matplotlib`, `opencv-python`, `coloredlogs` should be installed using either `pip` or `conda` manually.

Note that you don't have to explicitly install them all since there're dependency in between. For example, when `matplotlib` is installed before `numpy`, the latter will be automatically installed by `pip` or `conda`.

These're the core functionalities we used from `OpenCV` for this homework:

```
``` python
1 cv2.Canny(gray, 100, 200) # using Canny to get edge
2 cv2.findContours(edge, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
3 cv2.minAreaRect(contour)
4 cv2.fitEllipse(contour)
````
```

The functionality for them are quite self explanatory.

And they're listed here in the sequence in which they will be called by our homework project.

## ## Homework Procedure

In this homework, we do:

1. Read the image specified by the first user defined parameter

```
``` python
1 def main():
2     if len(sys.argv) > 1:
3         imgName = sys.argv[1]
4     else:
5         imgName = "camaro.jpg"
6
7     if not os.path.exists(imgName):
8         log.error(f"Cannot find the file specified: {imgName}")
9         log.info(f"This program fits ellipses to an image,
usage:\npython hw2.py <image name>")
10        return 1
11    ...
12    ...
13````
```

2. Convert the image to `rgb` 3 channel image, `grayscale` image and detect edge using `Canny` operator

```
``` python
1 def getImg(imgName):
2     img = cv2.imread(imgName)
3     rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5     # img = cv2.imread(imgName, cv2.IMREAD_GRAYSCALE) # assuming the
6     # first user defined arg as img name
7     edge = cv2.Canny(gray, 100, 200) # using Canny to get edge
8
```
```

3. Find **contours** on the detected **edge** map and fit the contours with **rectangles** and **ellipses** using *OpenCV*

```
``` python
1 def getFeatures(edge):
2     contours, _ = cv2.findContours(edge, cv2.RETR_TREE,
3     cv2.CHAIN_APPROX_SIMPLE)
4     rects = [None]*len(contours)
5     ellipses = [None]*len(contours)
6     for index, contour in enumerate(contours):
7         rects[index] = cv2.minAreaRect(contour)
8         if contour.shape[0] > 5:
9             ellipses[index] = cv2.fitEllipse(contour)
10    return contours, rects, ellipses
```
```

4. Render the **contours**, **rectangles**, and **ellipses** using *OpenCV*

```
``` python
1 def render(shape, contours, rects, ellipses):
2     # plt.figure()
3     drawing = np.zeros(shape, dtype='uint8')
4     log.info(f"Drawing on shape: {shape}")
5     for index, contour in enumerate(contours):
6         color = randcolor()
7         log.debug(f"Getting random color: {color}")
8         cv2.drawContours(drawing, contours, index, color, 1,
9         cv2.LINE_AA)
10        if contour.shape[0] > 5:
11            cv2.ellipse(drawing, ellipses[index], color, 3,
12            cv2.LINE_AA)
13
14        box = cv2.boxPoints(rects[index])
15        box = box.astype(int)
16        if box.shape[0] > 0:
```

```
15         cv2.drawContours(drawing, [box], 0, color, 2,
16         cv2.LINE_AA)
17
18     # plt.imshow(drawing)
19     # plt.show()
20
21     return drawing
```

```

## 5. Display the `rgb`, `grayscale`, `edge` map and `render` result using `matplotlib`

```
``` python
1 def main():
2     ...
3     ...
4     ...
5
6     # plt.figure(figsize=(16, 9))
7     plt.figure("Fitting")
8     plt.suptitle("Fitting ellipses and finding min-area
9     rectangles", fontweight="bold")
10    plt.subplot(221)
11    plt.title("Original", fontweight="bold")
12    plt.imshow(rgb)
13    plt.subplot(222)
14    plt.title("Grayscale", fontweight="bold")
15    plt.imshow(gray, cmap="gray")
16    plt.subplot(223)
17    plt.title("Canny Edge", fontweight="bold")
18    plt.imshow(edge, cmap="gray")
19    plt.subplot(224)
20    plt.title("Contour & Ellipses & Rects", fontweight="bold")
21    plt.imshow(drawing)
22    plt.show()
```

```

## 6. Utilities

```

``` python
1 mpl.rcParams["font", family=["Josefin Sans", "Trebuchet MS",
2 "Inconsolata"])
3 log = logging.getLogger(__name__)
4 coloredlogs.install(level='INFO') # Change this to DEBUG to see
5 more info.
6
7 def randcolor():
8     return [random.randint(0, 256) for i in range(3)]
```

```

## 7. Main function

```

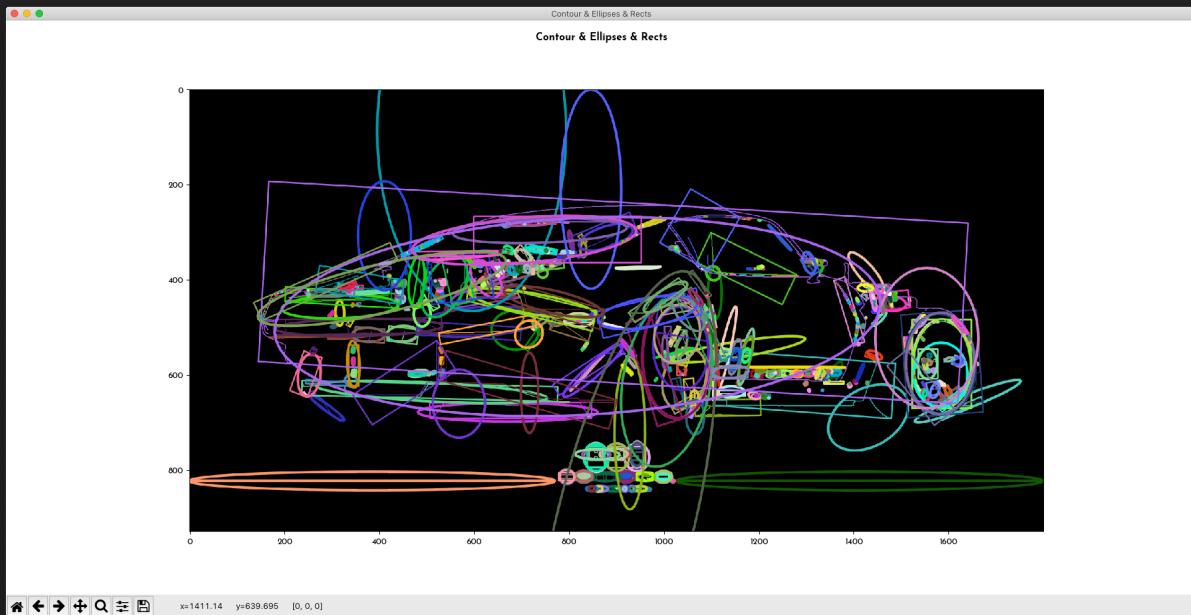
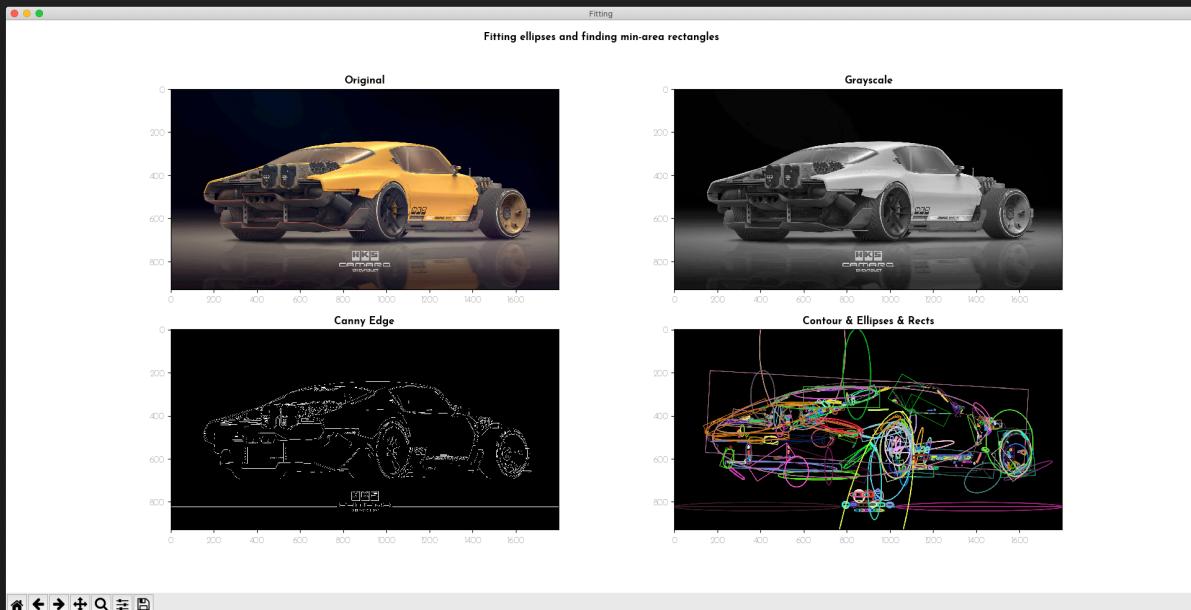
``` python
1 def main():
2     if len(sys.argv) > 1:
3         imgName = sys.argv[1]
4     else:
5         imgName = "camaro.jpg"
6
7     if not os.path.exists(imgName):
8         log.error(f"Cannot find the file specified: {imgName}")
9         log.info(f"This program fits ellipses to an image,
10 usage:\npython hw2.py <image name>")
11     return 1
12
13     img, rgb, gray, edge = getImg(imgName)
14     contours, rects, ellipses = getFeatures(edge)
15     drawing = render(img.shape, contours, rects, ellipses)
16
17     # plt.figure(figsize=(16, 9))
18     plt.figure("Fitting")
19     plt.suptitle("Fitting ellipses and finding min-area
20 rectangles", fontweight="bold")
21     plt.subplot(221)
22     plt.title("Original", fontweight="bold")
23     plt.imshow(rgb)
24     plt.subplot(222)
25     plt.title("Grayscale", fontweight="bold")
26     plt.imshow(gray, cmap="gray")
27     plt.subplot(223)
28     plt.title("Canny Edge", fontweight="bold")
29     plt.imshow(edge, cmap="gray")
30     plt.subplot(224)
31     plt.title("Contour & Ellipses & Rects", fontweight="bold")
32     plt.imshow(drawing)
33     plt.show()
```

```

## ## Homework Results

A screenshot of a Python development environment (VS Code) and a separate visualization window. The code editor shows the Python script `hw2.py` which processes an image of a yellow Camaro. The script uses OpenCV and Matplotlib to find contours, fit ellipses, and draw rectangles around them. The visualization window displays the original image, a grayscale version, and a processed version where the car's features are highlighted with red outlines and green rectangles. A terminal window at the bottom shows the command used to run the script and the resulting image file path.

```
hw2.py -- CompVision
You, seconds ago > hw2.py > ...
You, seconds ago > hw2.py > ...
#python
1 #!/usr/bin/python
2 import matplotlib.pyplot as plt
3 import matplotlib as mpl
4 import sys # for commandline arguments
5 import cv2
6 import random
7 import logging
8 import coloredlogs
9
10 log = logging.getLogger(__name__)
11 coloredlogs.install(level='INFO') # Change this to DEBUG to see more info.
12
13 def randcolor():
14     return [random.randint(0, 256) for i in range(3)]
15
16 You, seconds ago > Most of them
17 def randcolor():
18     return [random.randint(0, 256) for i in range(3)]
19
20
21 def getImg(imgName):
22     img = cv2.imread(imgName)
23     rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
24     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25     # img = cv2.imread(imgName, cv2.IMREAD_GRAYSCALE) # assuming the first user defined arg as img name
26     edge = cv2.Canny(gray, 100, 200) # using Canny to get edge
27     return img, rgb, gray, edge
28
29
30 def getFeatures(edge):
31     contours, _ = cv2.findContours(edge, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
32     rects = [None]*len(contours)
33     ellipses = [None]*len(contours)
34     for index, contour in enumerate(contours):
35         rectt(index) = cv2.boundingRect(contour)
36         if contour.shape[0] > 5:
37             ellipses[index] = cv2.fitEllipse(contour)
38     return contours, rects, ellipses
39
40
41 def render(shape, contours, rects, ellipses):
42     # plt.figure()
43     drawing = np.zeros(shape, dtype='uint8')
44     log.info("Drawing on shape: {shape}")
45     for index, contour in enumerate(contours):
46         color = randcolor()
47         log.info("Color: {color}")
48         cv2.drawContours(drawing, contours, index, color, 1, cv2.LINE_AA)
49
50
51
52
53
54
55
56
57
58
59
59
```



## ## Thoughts

Python forever...

It's just muuuuch easier to implement things and see how they work out, especially with the recent development of `cv2` API.

With `cv2` we get rid of `CvArr` or `CvMat`, and we embrace `numpy.ndarray`, which is intuitive and easy to work on.

And it's much easier to *borrow* other packages' functionalities too. For example, it's not gonna be as easy to plot figures and stuff using C++, where `matplotlib` doesn't exist.

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)



But I personally assume this is mainly a result of a too detailed image.

## ## Appendix

Full source code:

```
``` python
1  #!/python
2  import matplotlib.pyplot as plt
3  import matplotlib as mpl
4  import os
5  import sys # for commandline arguments
6  import cv2
7  import random # random color generator
8  import logging
9  import coloredlogs
10 import numpy as np
11
```

```
12 # Setting up font for matplotlib
13 mpl.rcParams["font", family=["Josefin Sans", "Trebuchet MS", "Inconsolata"],
   weight="medium")
14
15 # Setting up logger for the project
16 log = logging.getLogger(__name__)
17 coloredlogs.install(level='INFO') # Change this to DEBUG to see more
   info.
18
19
20 def randcolor():
21     """
22     Generate a random color, as list
23     """
24     return [random.randint(0, 256) for i in range(3)]
25
26
27 def getImg(imgName):
28     """
29     Load image by name (relative or absolute path)
30     Convert BGR to RGB/Grayscale
31     Get edge using Canny
32     """
33     img = cv2.imread(imgName)
34     rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
35     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
36     edge = cv2.Canny(gray, 100, 200) # using Canny to get edge
37     return img, rgb, gray, edge
38
39
40 def getFeatures(edge):
41     """
42     Detect contours in an image
43     Get minimum area box / best fit ellilipse of contours
44     """
45     contours, _ = cv2.findContours(edge, cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)
46     rects = [None]*len(contours) # [None, None, None, ... ]
47     ellipses = [None]*len(contours) # [None, None, None, ... ]
48
49     # All detected contour will have a min area box
50     # But only those with more than 5 points will have a ellipse
51     for index, contour in enumerate(contours):
52         rects[index] = cv2.minAreaRect(contour)
53         if contour.shape[0] > 5:
54             ellipses[index] = cv2.fitEllipse(contour)
55     return contours, rects, ellipses
56
57
58 def render(shape, contours, rects, ellipses):
59     """
60     Render the contours, rectangles and ellipses to an image
61     for later display
62     """
```

```

63     # Specifying the data type so that matplotlib won't treat this as a
64     # floating point image
65     drawing = np.zeros(shape, dtype='uint8')
66     log.info(f"Drawing on shape: {shape} with type {drawing.dtype}")
67     for index, contour in enumerate(contours):
68         color = randcolor()
69         log.debug(f"Getting random color: {color}")
70         cv2.drawContours(drawing, contours, index, color, 1,
71                           cv2.LINE_AA)
72
73         # All detected contour will have a min area box
74         # But only those with more than 5 points will have a ellipse
75         if contour.shape[0] > 5:
76             cv2.ellipse(drawing, ellipses[index], color, 3,
77                         cv2.LINE_AA)
78
79         box = cv2.boxPoints(rects[index])
80         box = box.astype(int)
81         cv2.drawContours(drawing, [box], 0, color, 2, cv2.LINE_AA)
82
83     return drawing
84
85
86 def main():
87     """
88     The main logic for this homework
89     """
90
91     # help message
92     log.info(f"This program fits ellipses to an image, usage:\npython
93     hw2.py <image name>")
94     # User might forget to give any commandline argument
95     if len(sys.argv) > 1:
96         imgName = sys.argv[1]
97     else:
98         imgName = "camaro.jpg"
99     # Or he/she might mistype the file name
100    if not os.path.exists(imgName):
101        log.error(f"Cannot find the file specified: {imgName}")
102        return 1
103
104    img, rgb, gray, edge = getImg(imgName)
105    contours, rects, ellipses = getFeatures(edge)
106    drawing = render(img.shape, contours, rects, ellipses)
107
108    # Display all four images in the same figure window
109    plt.figure("Fitting")
110    plt.suptitle("Fitting ellipses and finding min-area rectangles",
111                 fontweight="bold")
112    plt.subplot(221)
113    plt.title("Original", fontweight="bold")
114    plt.imshow(rgb)
115    plt.subplot(222)
116    plt.title("Grayscale", fontweight="bold")
117    plt.imshow(gray, cmap="gray")

```

```
113     plt.subplot(223)
114     plt.title("Canny Edge", fontweight="bold")
115     plt.imshow(edge, cmap="gray")
116     plt.subplot(224)
117     plt.title("Contour & Ellipses & Rects", fontweight="bold")
118     plt.imshow(drawing)
119     plt.show()
120
121     # Display the detected contours rectangles and ellipses separately
122     for examination:
123         plt.figure("Contour & Ellipses & Rects")
124         plt.suptitle("Contour & Ellipses & Rects", fontweight="bold")
125         plt.imshow(drawing)
126         plt.show()
127
128 # with the #!/python at the beginning of this file
129 # you can ommit the python when executing this program
130 if __name__ == "__main__":
131     main()
132
```

```