4 Constructing Basic Shapes in OpenCV

OpenCV基本图形

CV一个基本功能是绘制基本形状，提供了画直线、圆、矩形、椭圆等形状的函数。在计算机视觉项目

中，常通过绘制形状来修改图像，例如开发人脸检测算法，要在计算机图像中绘制矩形框住检测到的人脸。如开发人脸识别算法，要绘制矩形突出显示检测到的人脸，用文本显示检测到人脸的身份。编写调试信息，如显示检测到的面孔数量或处理时间，以便查看面孔检测算法的性能。在本章中介绍OpenCV库基本和高级图形绘制。

将讨论下列主题：

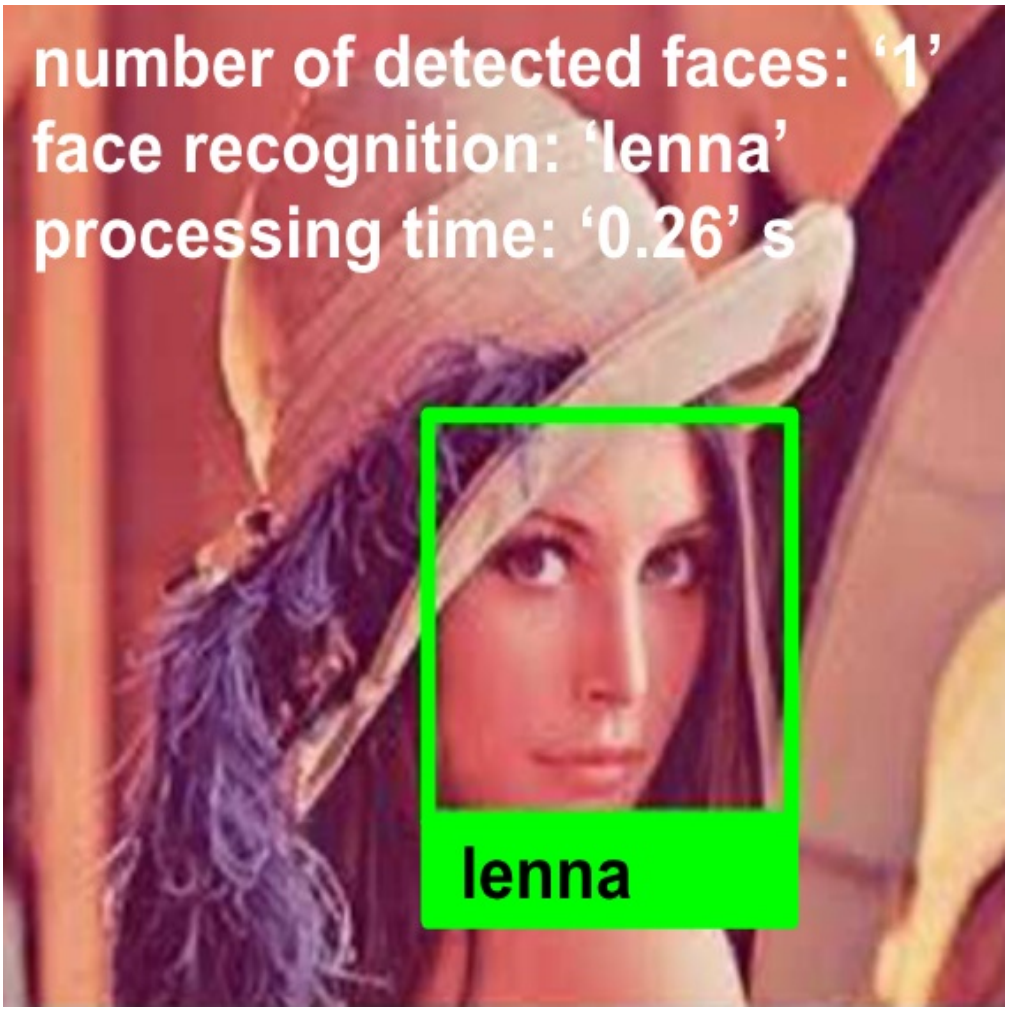
- OpenCV绘图介绍
- 在OpenCV中绘制基本形状，线、矩形和圆
- 基本形状(2)-剪贴线和带箭头的线、椭圆和折线
- 绘制文本
- 鼠标事件动态绘图
- 高级绘图

OpenCV绘图介绍

OpenCV提供了很多绘制基本形状的函数。常见图形包括线条、矩形和圆形。使用OpenCV可以绘制更多图形。正如前面提到，在图像上绘制图形常见于：

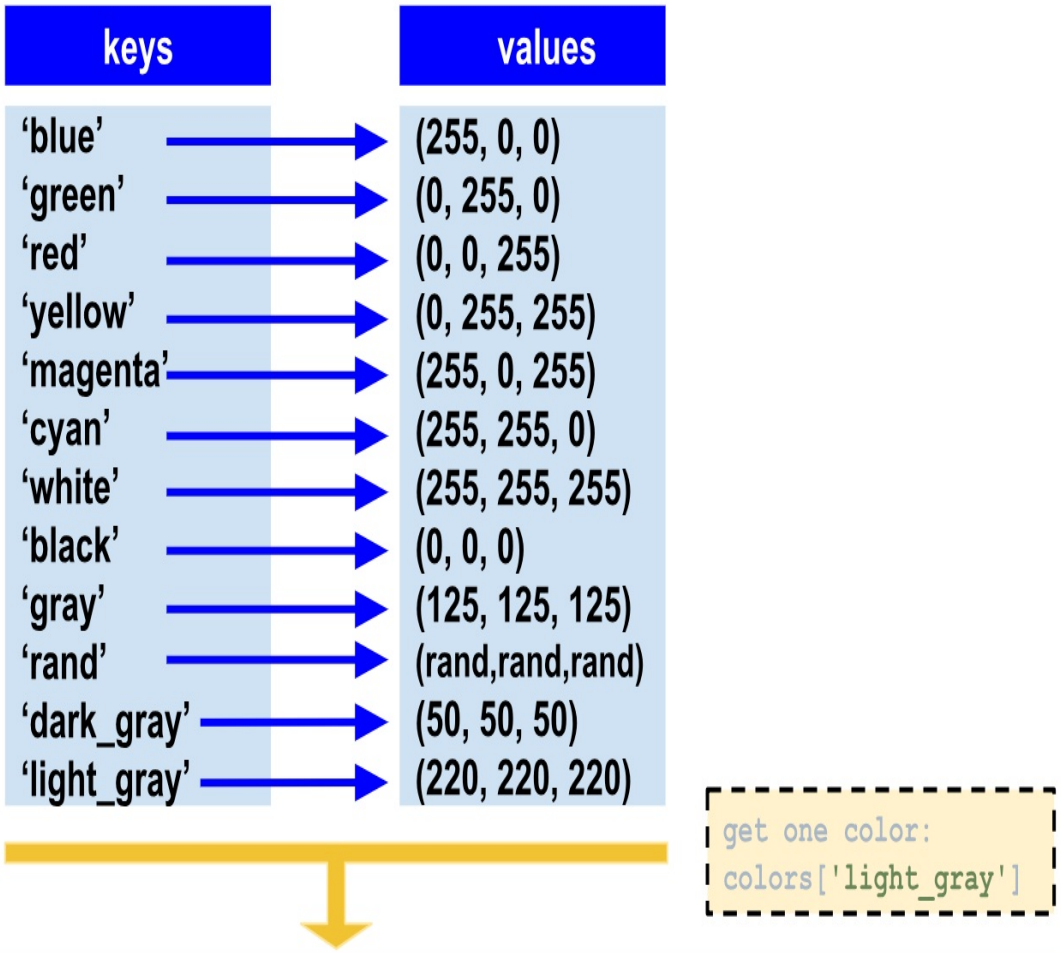
- 显示算法的中间结果
- 显示算法的最终结果
- 显示调试信息

在下一个屏幕截图中，可看到修改后的图像，其中包含两种算法(人脸检测和人脸识别)的一些信息。通过这种方式，可处理目录中的所有图像，看到算法在哪里检测到错误的面孔(假阳性，误报)，甚至是面孔缺失(假阴性，漏报)



误报是指结果应该是一种情形，而实际出现的并不是，例如应该是椅子，却归类为脸。漏报是指结果应该出现，实际却缺失，例如没有检测到人脸。

这一章介绍如何画一些基本形状和不同颜色的文字。将展示在很多例程使用的两个基本函数。第一个函数是构建一个颜色字典，定义要使用的主要颜色。以下截图可看到它是如何工作的




```
colors = {'blue': (255, 0, 0), 'green': (0, 255, 0), 'red': (0, 0, 255), 'yellow': (0, 255,
255), 'magenta': (255, 0, 255), 'cyan': (255, 255, 0), 'white':  (255, 255, 255), 'black':
(0, 0, 0), 'gray': (125, 125, 125), 'rand': np.random.randint(0,high=256,size=(3,)).tolist(),
'dark_gray': (50, 50, 50), 'light_gray': (220, 220, 220)}
```

这字典只是用来训练和练习的。对于其他目的，可以使用其他选项。创建constant.py文件定义颜色。每种颜色都由一个常数定义

```
"""
Common colors triplets (BGR space) to use in Op

BLUE = (255, 0, 0)
GREEN = (0, 255, 0)
RED = (0, 0, 255)
YELLOW = (0, 255, 255)
MAGENTA = (255, 0, 255)
CYAN = (255, 255, 0)
DARK_GRAY = (50, 50, 50)
...
```

下面代码，使用这些常量

```
```python
import  constant

#  Getting red color:
<p class="mume-header " id="getting-red-color">

print("red: '{}'".format(constant.RED))
```

常量通常用大写字母指定，例如BLUE，单词之间用下划线指定，例如，DARK_GRAY。
```

由于使用Matplotlib绘制图形，建了show_with_matplotlib()函数，它有两个参数。第一个是要显示的图像，第二个是要绘制的图形的标题。函数第一步是将BGR图像转换为RGB，因为使用Matplotlib显示彩色图像。函数的第二步用Matplotlib功能显示图像。testing_colors.py将这些片段组合。在这个脚本中，绘制线条，每一条都用字典的颜色表示。

创建字典的代码

```
#   Dictionary  containing  some  colors
<p class="mume-header " id="dictionary-containi

colors = {'blue': (255, 0, 0), 'green': (0, 255
```

可看到一些预定义的颜色包括在这个字典中:蓝色、绿色、红色、黄色、品红、青色、白色、黑色、灰色、随机的灰色、暗灰和浅灰色。如果想用特定的颜色(例如，品红)，操作如下

```
colors['magenta']
```

或者，可用(255, 0, 255)来获得洋红色。使用字典比三列数字更容易些，不需要记住RGB颜色空间的叠加属性(蓝色(255, 0, 0)和红色(0, 0, 255)相加得品红(255, 0, 255))。可用constant.py来执行此功能。

如果不知道这些数字是什么或表示什么，阅读第2章OpenCV图像基础，介绍了这些概念。

大多数示例中使用colors函数和show_with_matplotlib()函数，testing_colors.py脚本了解如何使用这两个函数。执行后的屏幕截图

例程中创建了大小为500 × 500的图像，有3个通道(因需要一个彩色图像)和一个uint8类型(8位无符号整数)。黑色背景。

```
#  We  create the canvas  to draw: 400 x 400 pi
<p class="mume-header " id="we-create-the-canva

#  We set background to black using np.zeros()
<p class="mume-header " id="we-set-background-t

image = np.zeros((500, 500, 3), dtype="uint8")
```

本例中，希望将背景设置为浅灰色，而不是黑色。如果要更改背景，可以执行以下操作

```
#  If you want another background color, you ca
<p class="mume-header " id="if-you-want-another

image[:]  =  colors['light_gray']
```

接下来添加绘制线条的功能，线条都用字典颜色表示。应该注意的是，在下一节将看到如何创建基本形状，如果不理解创建线条的代码，也不要担心

```
#  We  draw all the colors to test the dictiona
<p class="mume-header " id="we-draw-all-the-col

#  We  draw  some  lines,  each  one  in  a  co
<p class="mume-header " id="we-draw-some-lines-

for key in colors:
cv2.line(image, (0, separation), (500, separati
separation  +=  40
```

最后，用创建的show_with_matplotlib()函数绘制图像

```
#   Show  image:
<p class="mume-header " id="show-image"></p>

show_with_matplotlib(image,  'Dictionary  with
```

show_with_matplotlib()的两个参数是要绘制的图像和要显示的标题，现在准备使用OpenCV和Python画基本形状。

图形绘制

本节将看到如何使用OpenCV功能绘制图形。包括基本形状和更高级的形状。

基本形状线条、长方形和圆形

在下一个示例中将看到如何在OpenCV中绘制基本形状。这些基本形状包括线条、矩形和圆，它们是最常见和最简单的图形。第一步是创建一个图像，图形在此绘制。包含3个通道(以显示BGR图像)和uint8类型(8位无符号整数)的400×400图像

```
#  We  create the canvas  to draw: 400 x 400 pi
<p class="mume-header " id="we-create-the-canva

#  We set the background to black using np.zero
<p class="mume-header " id="we-set-the-backgrou

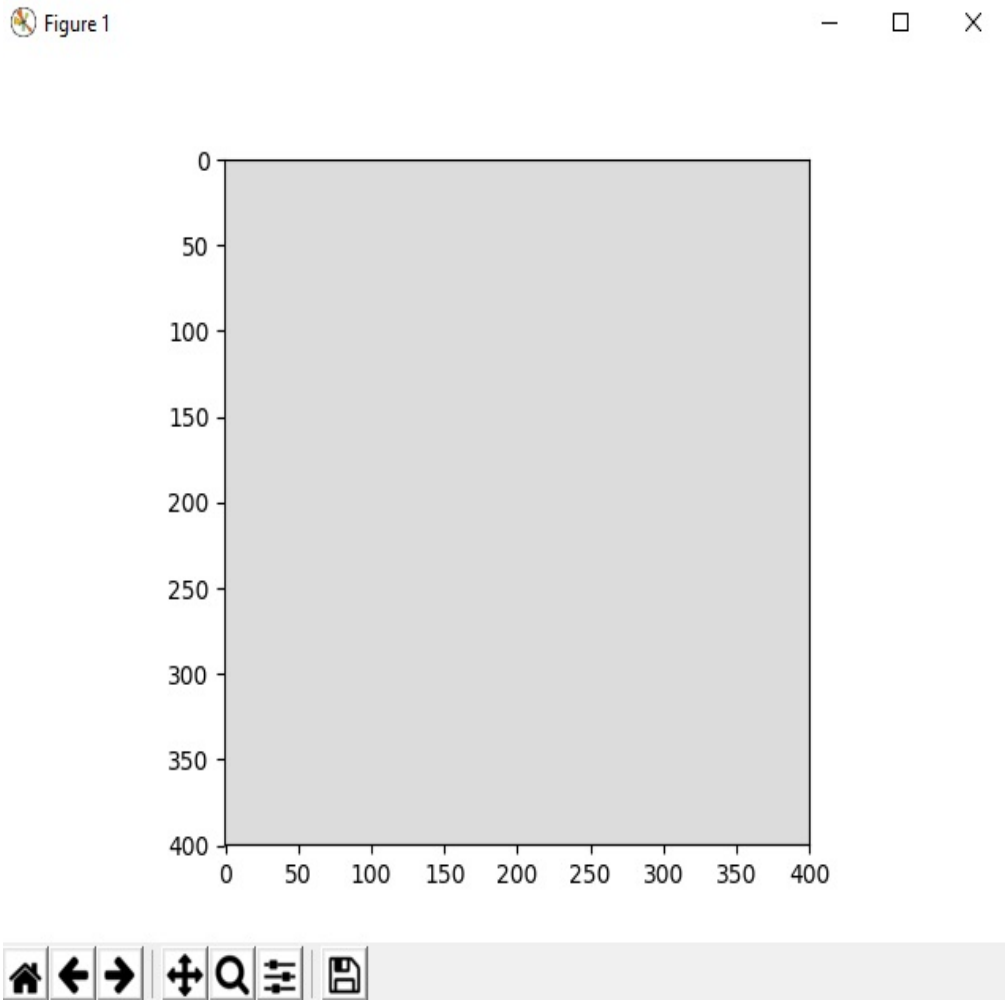
image = np.zeros((400, 400, 3), dtype="uint8")
```

用颜色字典将背景设置为浅灰色

```
#  If you want another background color, you ca
<p class="mume-header " id="if-you-want-another

image[:]  =  colors['light_gray']
```

此画布(或图像)的屏幕截图



现在绘制基本形状。应该注意的是，OpenCV提供的大多数绘图函数都有公共参数。为了简单起见，简要介绍这些参数

- img 要绘制图形的图像
- color:用来绘制图形的颜色(BGR三色)
- thickness:如果该值为正，则为形状轮廓的厚度。否则，将绘制填充形状.
- lineType:形状边线类型。OpenCV提供三种类型的行:
    - cv2.LINE_4: This means four-connected lines四连线
    - cv2.LINE_8: This means eight-connected lines八连线
    - cv2.LINE_AA: This means an anti-aliased line反锯齿线
      shift:表示与定义形状点的坐标相关的小数位数。

与上述参数相关的是cv2.LINE_AA选项为lineType生成更好的绘图(例如，在绘制文本时)，但绘制速度较慢。八连通线和四连通线都是非反锯齿线，都是用布列舍纳姆算法绘制的。对于抗锯齿的线型，采

用高斯滤波算法。此外，移位参数是必要的，因为许多绘图函数无法处理亚像素精度。为了简单起见，在例子中将用整数坐标。因此，这个值设置为0(shift = 0)，为更全面地理解，还将提供一个如何使用shift参数的示例。

> 本节所有示例，都创建了画布来绘制所有形状。这个画布是$400x400$像素的图像，背景是浅灰色。请参见前面的屏幕截图。

画线

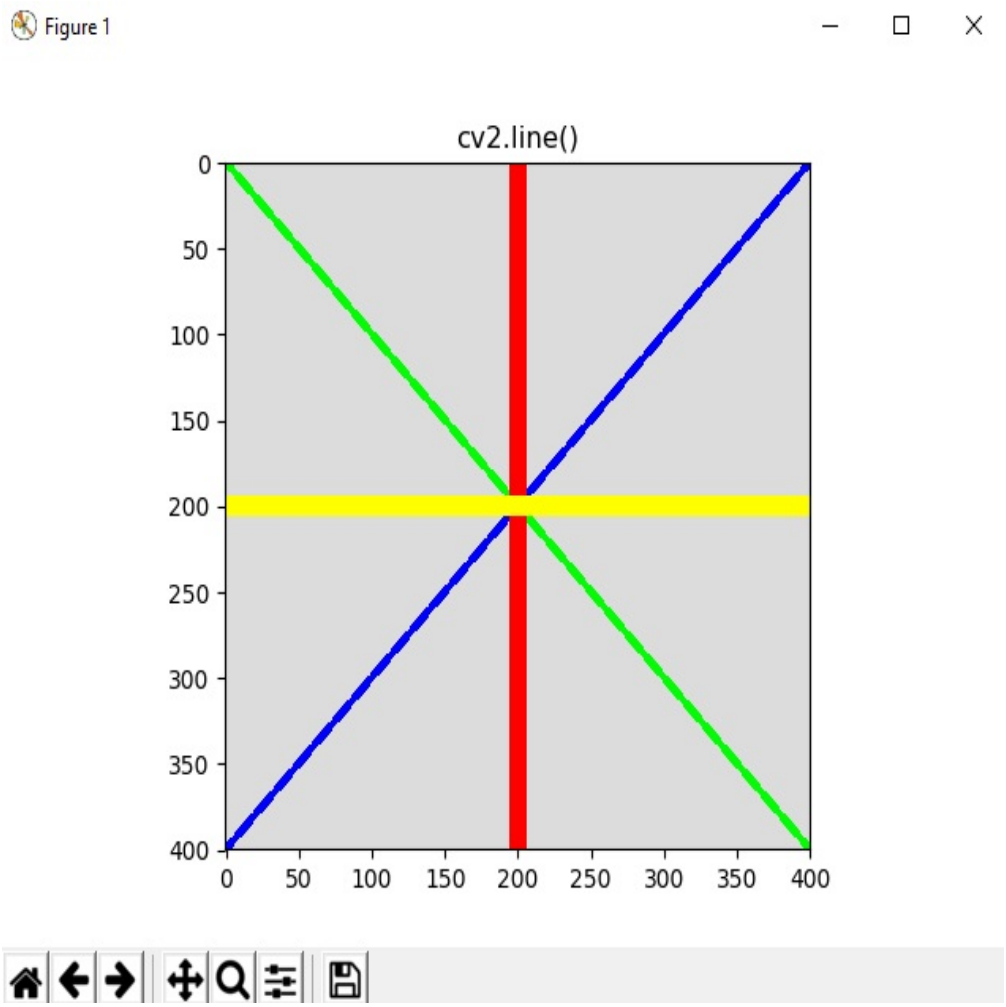
第一个函数是cv2.line()。函数原型

```
img = line(img, pt1, pt2, color, thicknes
```

函数在连接pt1和pt2的img图像上画一条线

```
cv2.line(image, (0, 0), (400, 400), colors['gre
cv2.line(image, (0, 400), (400, 0), colors['blu
cv2.line(image, (200, 0), (200, 400), colors['r
cv2.line(image, (0, 200), (400, 200), colors['y
```

调用show_with_matplotlib(image, 'cv2.line()')函数。屏幕截图：



画矩形

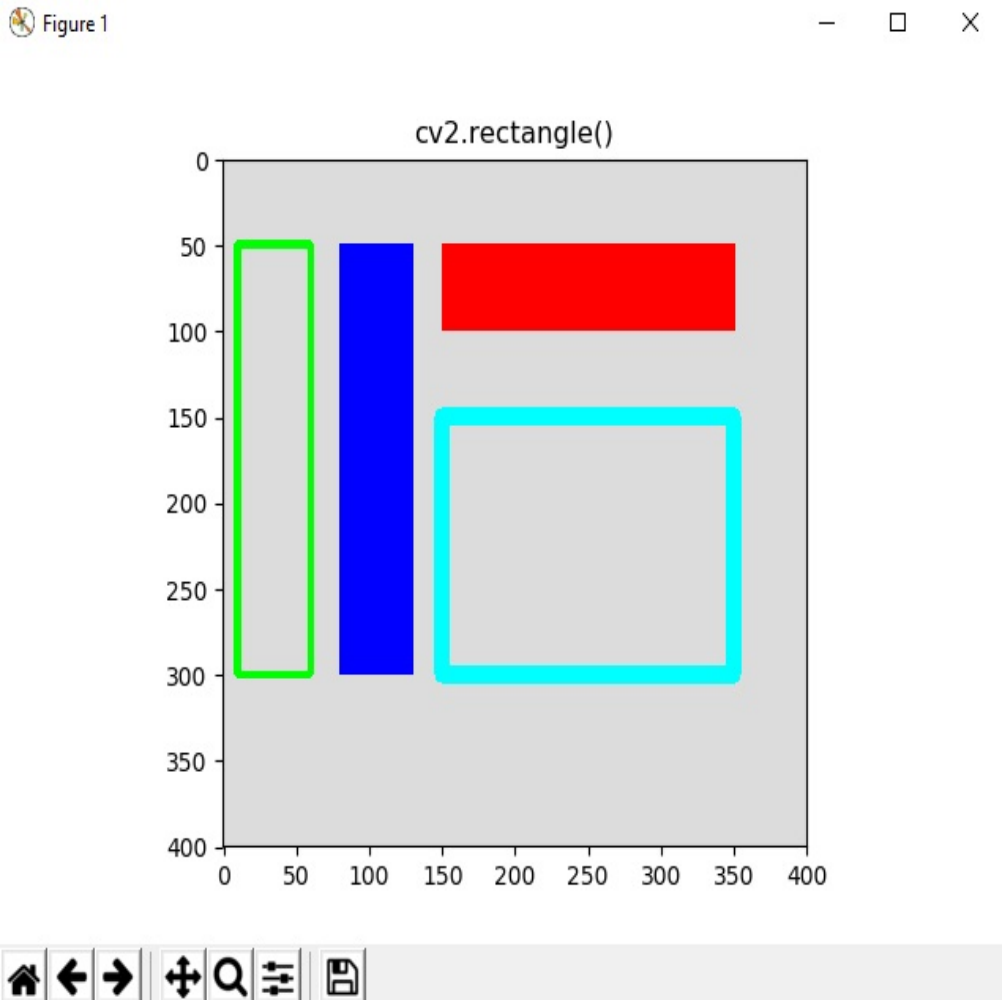
cv2.rectangle()函数原型

```
img = rectangle(img, pt1, pt2, color, thickness
```

这个函数画两个相对的角pt1和pt2的矩形

```
cv2.rectangle(image, (10, 50), (60, 300), color
cv2.rectangle(image, (80, 50), (130, 300), colo
cv2.rectangle(image, (150, 50), (350, 100), col
cv2.rectangle(image,  (150,  150),  (350,  300)
```

在绘制这些矩形之后，我们调用show_with_matplotlib(image, 'cv2.rectangle()')函数。结果在下一个屏幕截图中显示



请记住，厚度参数的负值(例如-1)表示绘制填充形状。

画圆

circle()函数签名如下

```
img = circle(img, center, radius, color,
```

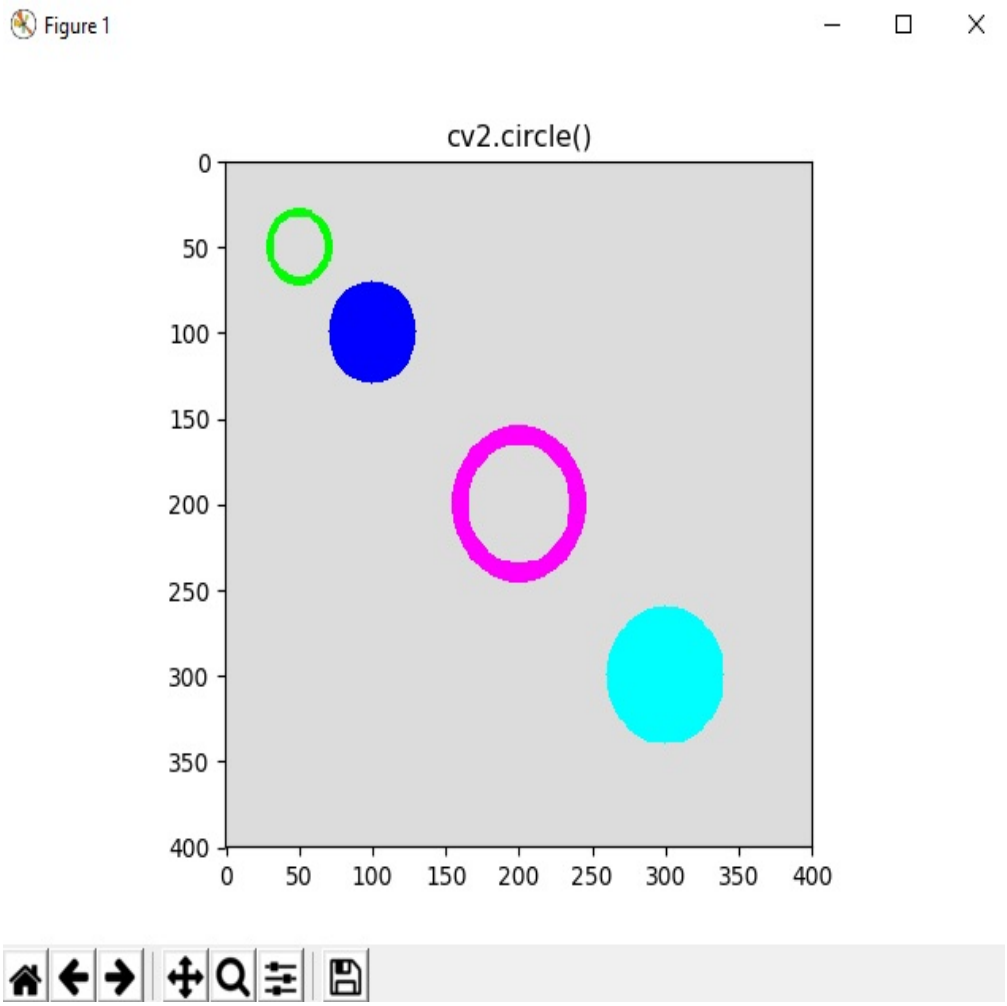
以圆心位置、半径绘圆。代码如：

```
cv2.circle(image, (50, 50), 20, colors['green']
cv2.circle(image, (100, 100), 30, colors['blue'
cv2.circle(image, (200, 200), 40, colors['magen
cv2.circle(image, (300, 300), 40, colors['cyan'
```

绘制这些圆后，调用

```
show_with_matplotlib(image, 'cv2.circle()')
```

函数，见屏幕截图



示例完整代码在basic_drawing.py。

了解高级图形

在本节中将看到如何绘制剪辑线、带箭头的线、椭圆和折线。这些形状不像前一节的图形容易绘制，但容易理解。第一步是创建绘制图形的图像，3个通道(显示BGR图像)和uint8类型(8位无符号整数)的300×300的图像

```
#  We  create the canvas  to draw: 300 x 300 pi
<p class="mume-header " id="we-create-the-canva

#  We set the background to black using np.zero
<p class="mume-header " id="we-set-the-backgrou

image = np.zeros((300, 300, 3), dtype="uint8")
```

使用颜色字典将背景设置为浅灰色

```
#  If you want another background color, you ca
<p class="mume-header " id="if-you-want-another

image[:]  =  colors['light_gray']
```

这时可开始绘制新图形。

画剪辑线

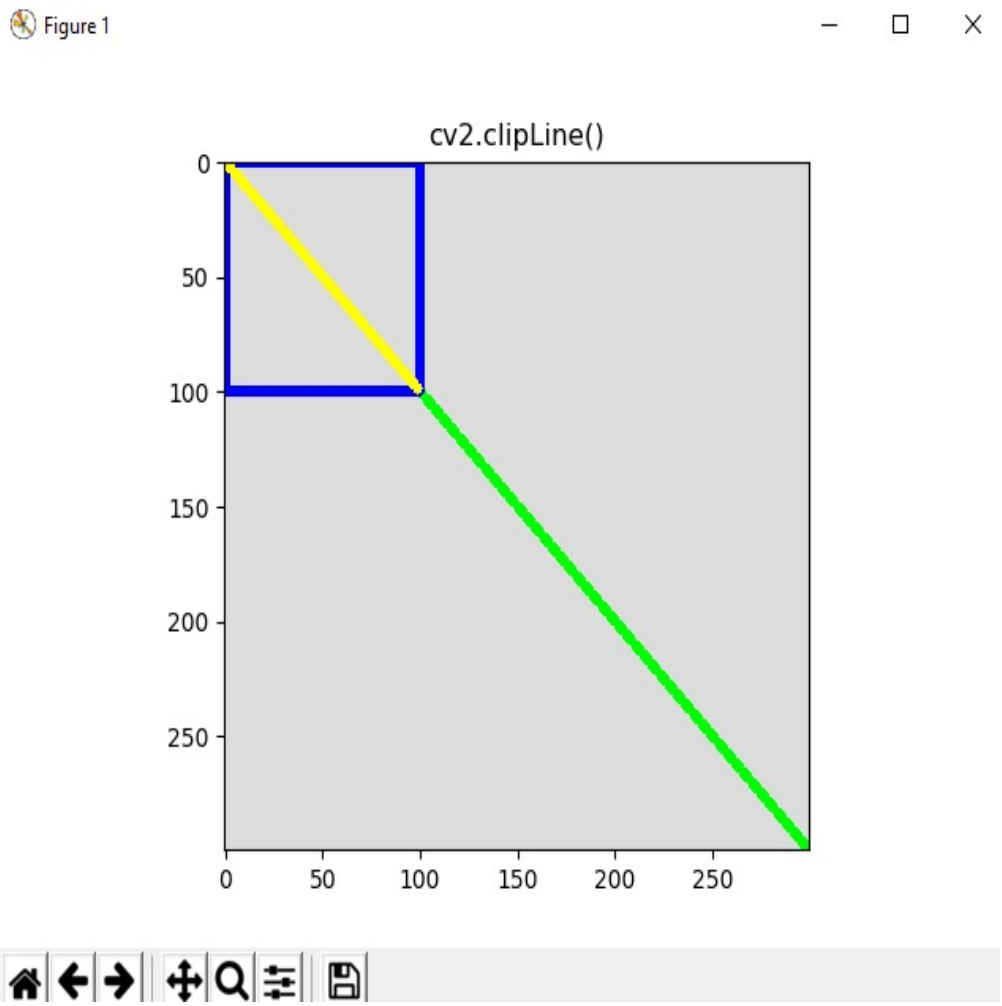
cv2.clipLine()函数的签名如下

```
retval,  pt1,  pt2  =  clipLine(imgRect,  pt1,
```

cv2.clipLine()函数返回矩形内的段，由pt1和pt2输出点，该函数列矩形内的线段。在这个意义上，如果两个原始的pt1和pt2点都在矩形外，则retval为假。否则，如两个pt1或pt2点在矩形内，函数返回True。在下一段代码中可以更清楚地看到这一点

```
cv2.line(image, (0, 0), (300, 300), colors['gre
cv2.rectangle(image, (0, 0), (100, 100), colors
ret, p1, p2 = cv2.clipLine((0, 0, 100, 100), (0
if ret:
    cv2.line(image, p1, p2, colors['yellow'], 3
```

以下屏幕截图，执行代码结果图



可以看到，由p1和p2点定义的线段以黄色显示，将原始线段剪切到矩形上。因矩形内至少有一个点，ret为真，由pt1和pt2定义的线段绘黄色。

画箭头

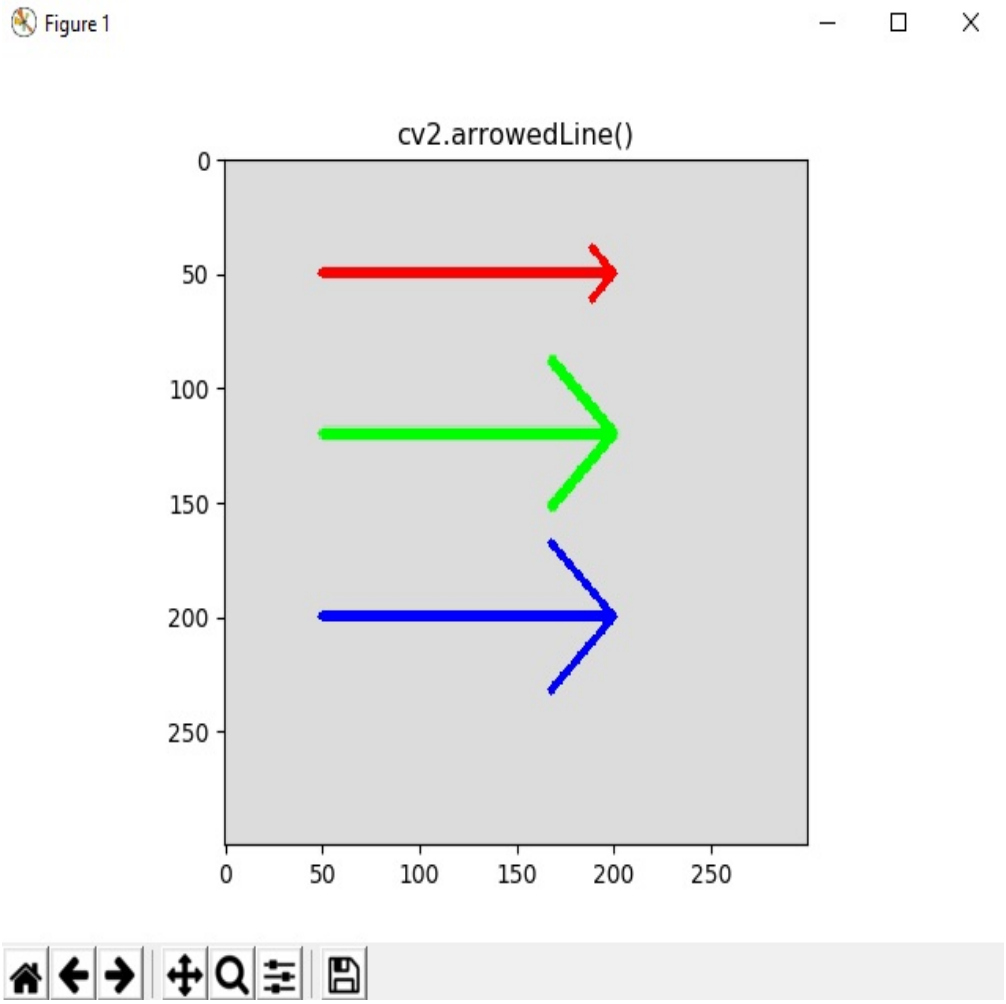
函数签名如下

```
cv.arrowedLine(img, pt1, pt2, color, thickness=
```

函数允许创建一个箭头，从定义的第一个点pt1指向定义的第二个点pt2。箭头长度可通过tipLength参数来控制，tipLength参数是根据线段长度(pt1和pt2之间的距离)定义

```
cv2.arrowedLine(image, (50, 50), (200, 50), col
cv2.arrowedLine(image, (50, 120), (200, 120), c
cv2.arrowedLine(image, (50, 200), (200, 200), c
```

可以看到定义了三个箭头。请看屏幕截图，箭头绘制的地方。查看cv2.LINE_AA(也可以写成16)和8(也可以写成cv2.LINE_8)



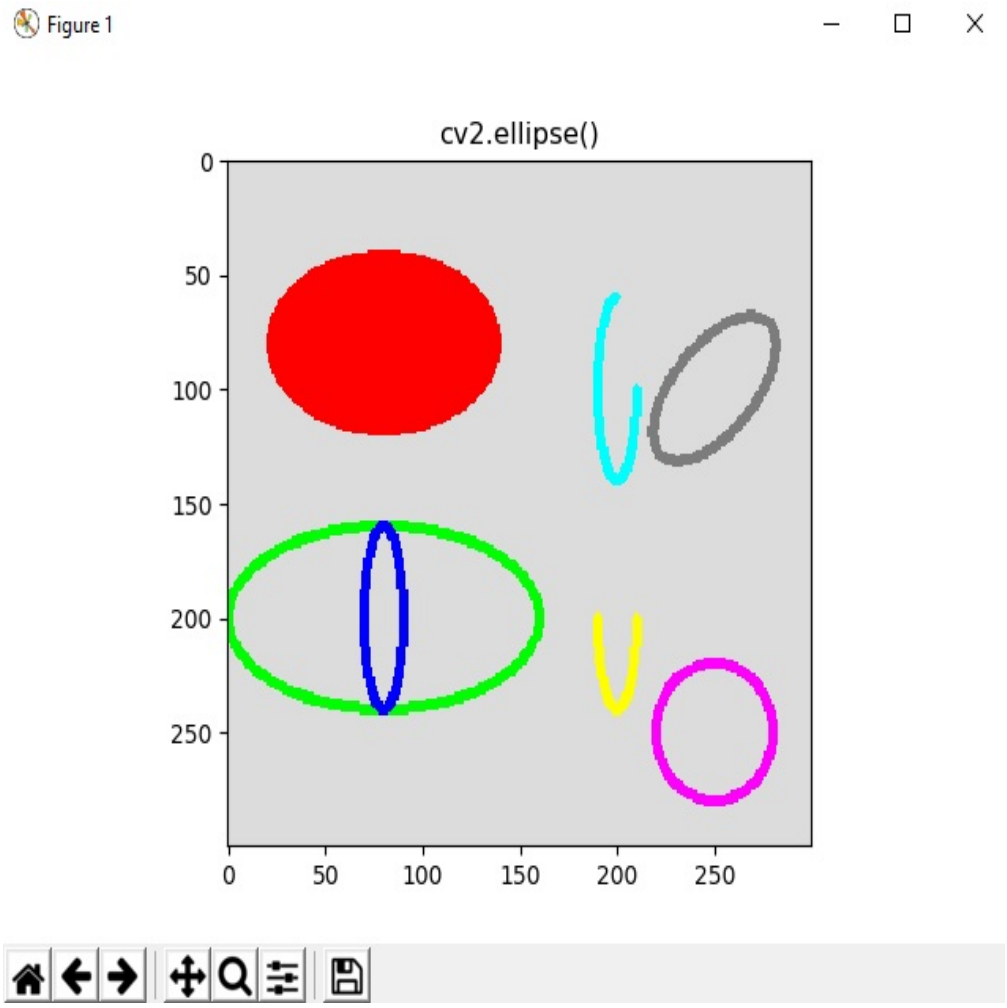
> 本例中将两个enum(例如，cv2.LINE_AA)或直接将值(例如，8)与lineType参数相结合。但这不是好主意，因为它可能会让你迷惑。应该在所有代码中建立和维护一个标准。

画椭圆

函数签名如下

```
cv2.ellipse(img, center, axes, angle, startAngl
```

函数允许创建不同类型的椭圆。以度为单位的角度参数允许旋转椭圆。坐标轴参数控制椭圆的大小，椭圆的大小相当于坐标大小的一半。如果需要完整的椭圆，则startAngle = 0, endAngle = 360。否则，应将参数调整到所需的椭圆弧(以度为单位)。通过对坐标轴参数传递相同的值，可绘制圆



```
cv2.ellipse(image, (80, 80), (60, 40), 0, 0, 36
cv2.ellipse(image, (80, 200), (80, 40), 0, 0, 3
cv2.ellipse(image, (80, 200), (10, 40), 0, 0, 3
cv2.ellipse(image, (200, 200), (10, 40), 0, 0,
cv2.ellipse(image, (200, 100), (10, 40), 0, 0,
cv2.ellipse(image, (250, 250), (30, 30), 0, 0,
cv2.ellipse(image, (250, 100), (20, 40), 45, 0,
```

截图可看到椭圆

画多边形

The signature for this function is as follows:

这个函数的签名如下

```
cv2.polylines(img, pts, isClosed, color, thickn
```

这个函数允许你创建多边形曲线。这里的关键参数是pts，其中应该提供定义多边形曲线的数组。这个参数的形状应该是(number_vertex, 1,2)，所以常

用的方法是使用np来定义它。数组来创建(np. int32类型的)坐标，然后将其重新塑形以匹配前面提到的形状。例如，要创建一个三角形，代码如下所示

```
#  These  points define a triangle
<p class="mume-header " id="these-points-define

pts = np.array([[250, 5], [220, 80], [280, 80]]
#  Reshape to shape (number_vertex, 1, 2)
<p class="mume-header " id="reshape-to-shape-nu

pts  =  pts.reshape((-1,  1,  2))
#  Print the shapes: this line is not necessary
<p class="mume-header " id="print-the-shapes-th

print("shape of pts '{}'".format(pts.shape))
#  this gives: shape of pts '(3, 1, 2)'
<p class="mume-header " id="this-gives-shape-of
```

另一个重要的参数是isClosed。如果该参数为真，则该多边形将被封闭绘制。否则，第一个和最后一个顶点之间的线段将不会被绘制出来，从而形成一

个开放的多边形。完整的解释，为了画一个封闭的三角形，下面给出代码

```
#  These  points define a triangle
<p class="mume-header " id="these-points-define

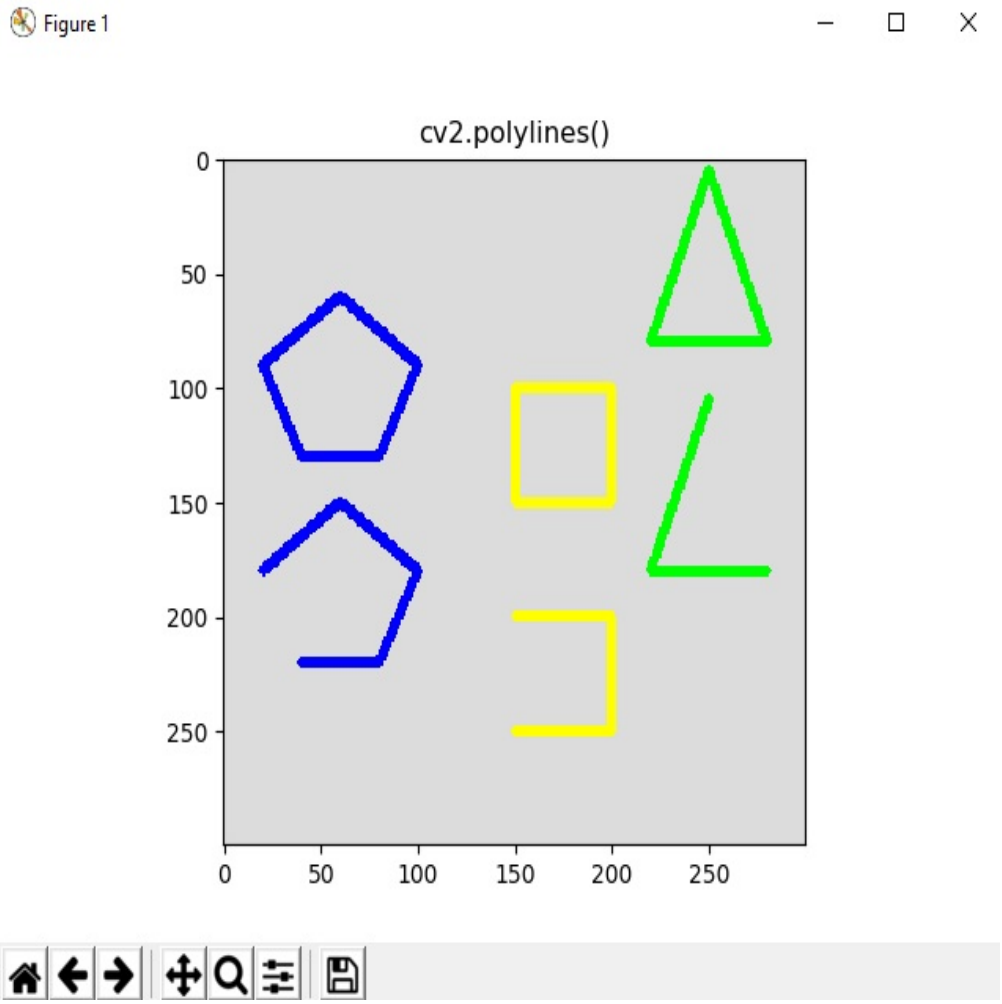
pts = np.array([[250, 5], [220, 80], [280, 80]]
#  Reshape to shape (number_vertex, 1, 2)
<p class="mume-header " id="reshape-to-shape-nu

pts  =  pts.reshape((-1,  1,  2))
#  Print the shapes: this line is not necessary
<p class="mume-header " id="print-the-shapes-th

print("shape of pts '{}'".format(pts.shape))
#  Draw this poligon with True option
<p class="mume-header " id="draw-this-poligon-w

cv2.polylines(image,  [pts],  True,  colors['gr
```

以同样的方式，我们对五边形和长方形进行了编码，在下一个截图中可以看到



要查看本节的完整代码，可以查看basic_drawing_2.py脚本

绘图函数中的移位参数

以前的一些函数(具有移位参数的函数)可以在与像素坐标相关的亚像素精度下工作。为了解决这个问题，您应该将坐标作为定点数传递，定点数被编码为整数。

定点数表示为整数(小数点左边)和小数部分(小数点右边)保留特定的(固定的)位数(位)。

因此，移位参数允许您指定小数位数的数目(小数点右侧)。最后，实点坐标计算如下

例如，这段代码画了两个半径为300的圆。其中一个使用shift = 2的值来提供亚像素精度。在这种情况下，应该将原点和半径同时乘以4 (2 shift= 2)

```
shift = 2
factor = 2 ** shift
print("factor: '{}'".format(factor))
cv2.circle(image, (int(round(299.99 * factor)),
```

如果shift = 3，则该因子的值为8 (2shift = 3)，依此类推。乘以2的幂相当于将整数二进制表示的位向左移动1。这样就可以画出浮点坐标。为了总结这一点，我们还可以为cv2.circle()创建一个包装器函数，它可以使用shift参数属性处理浮动坐标draw_float_circle()。下面显示这个示例的关键代

码。完整的代码是在shift_parameter.py脚本中定义的

```
def draw_float_circle(img, center, radius, colo

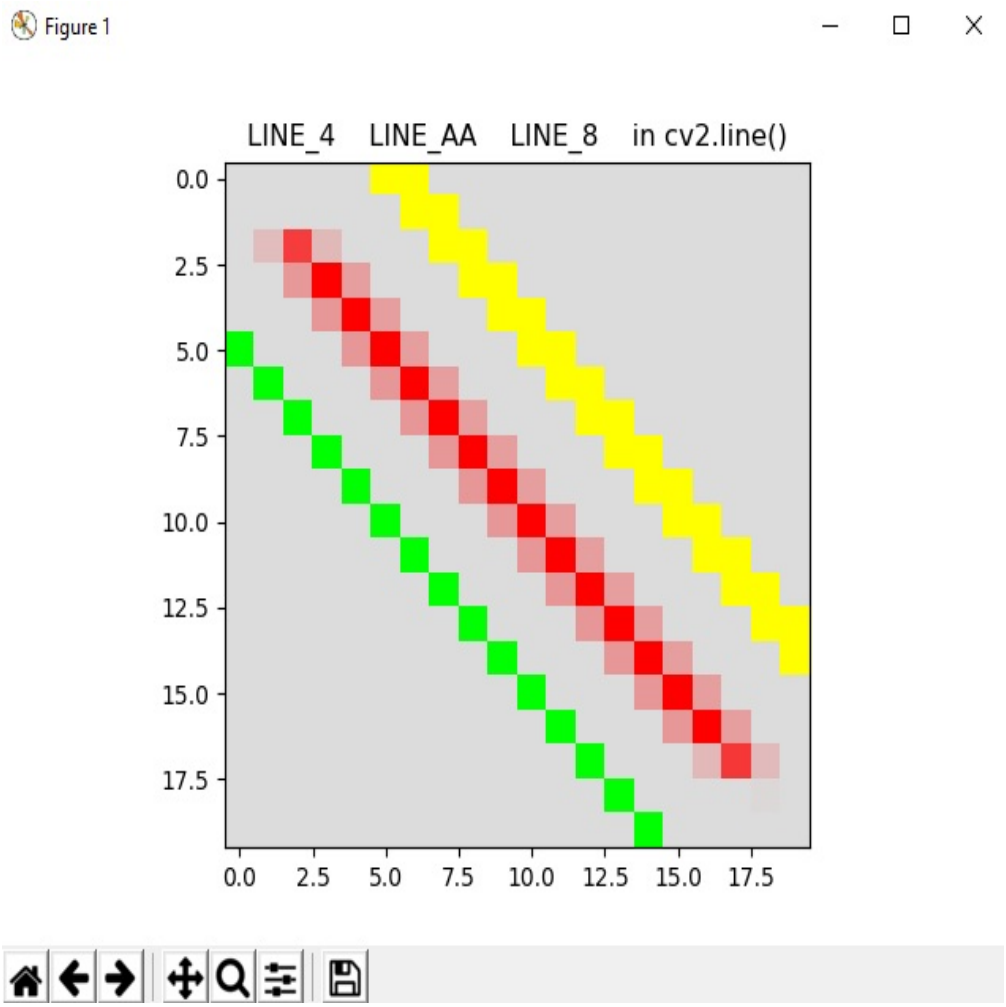
"""
factor = 2 ** shift
center = (int(round(center[0] * factor)), int(r
cv2.circle(img, center, radius, color, thicknes

draw_float_circle(image, (299, 299), 300, color
draw_float_circle(image, (299.9, 299.9), 300, c
draw_float_circle(image, (299.99, 299.99), 300,
draw_float_circle(image, (299.999, 299.999), 30
```

绘图函数中的lineType参数

另一个常见的参数是lineType，可接受三个不同的值。之前经讨论过这三种类型之间的区别。参看屏

幕截图，绘制了三条具有相同厚度和倾角的线:黄色= cv2.LINE_4，红色= cv2.LINE_AA，绿色= cv2.LINE_8。示例完整代码见basic_line_types.py



在前面的屏幕截图中，可清楚看到在用三种不同线条类型绘制直线时的差异。

写文本

OpenCV可在图像中渲染文本。本节介绍如何使用cv2.putText()函数来绘制文本。此外，将看到所有可用的字体，一些与绘制文本相关的OpenCV函数。

画文本

cv2.putText()函数签名

```
img = cv.putText( img, text, org, fontFace, fon
```

此函数使用fontFace和fontScale比例的字体类型从org坐标绘制所文本字符串，如果bottomLeftOrigin = False，为左上角，否则为左下角。通过这个示例，可看到的参数lineType采用OpenCV中三个不同的值。cv2.LINE_4 cv2.LINE_8，cv2.LINE_AA。通过这种方式，可以在绘制这些类型时看到差异。记住cv2.LINE_AA提供了更好的质量,为一种抗锯齿的线条类型，但绘制速度比其他两种类型慢。下面给

出绘制文本的关键代码，完整代码见

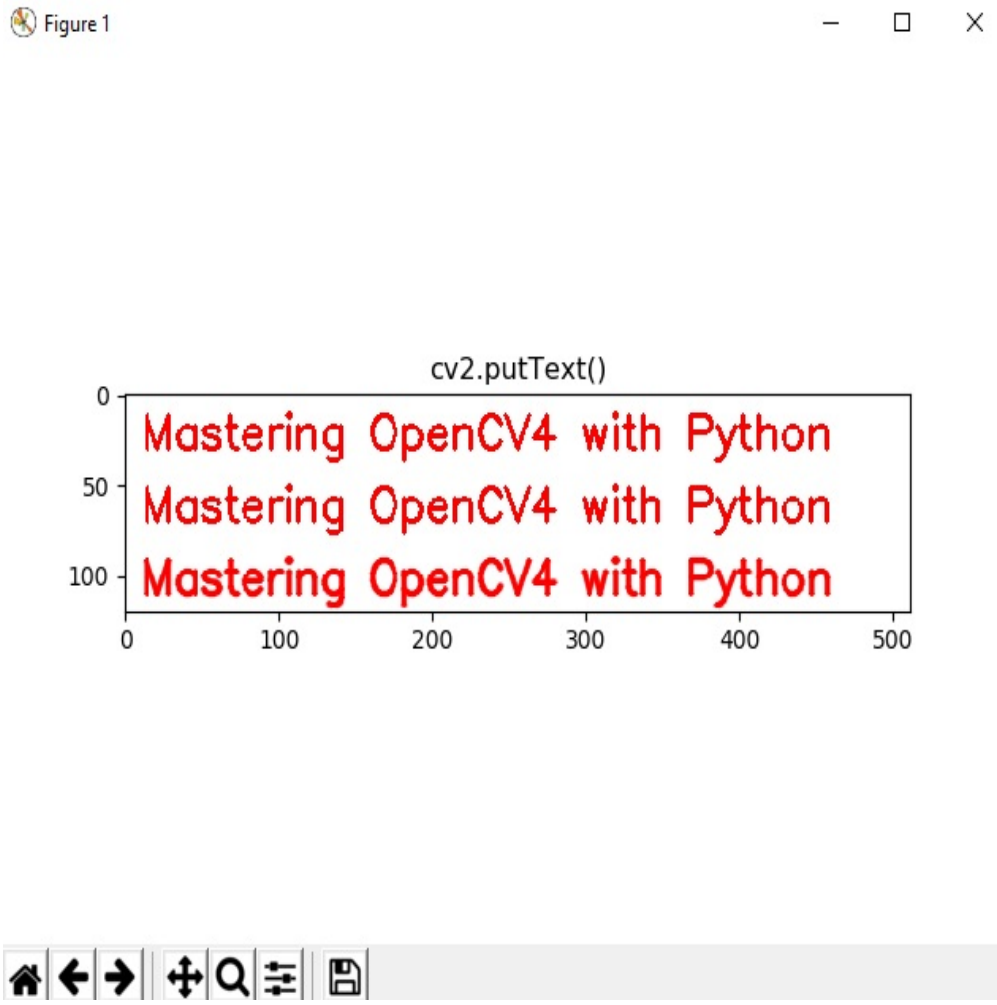
text_drawing.py

```
  #   We  draw  some  text on  the  image:
  <p class="mume-header " id="we-draw-some-text-o

  cv2.putText(image, 'Mastering OpenCV4 with Pyth
              cv2.LINE_4)
  cv2.putText(image, 'Mastering OpenCV4 with Pyth
              cv2.LINE_8)
  cv2.putText(image, 'Mastering OpenCV4 with Pyth
              cv2.LINE_AA)
  #   Show  image:
  <p class="mume-header " id="show-image-1"></p>

  show_with_matplotlib(image,  'cv2.putText()')
```

屏幕截图：



在本例中，背景颜色设置为白色。执行以下操作

```
image.fill(255)
```

OpenCV所有字体

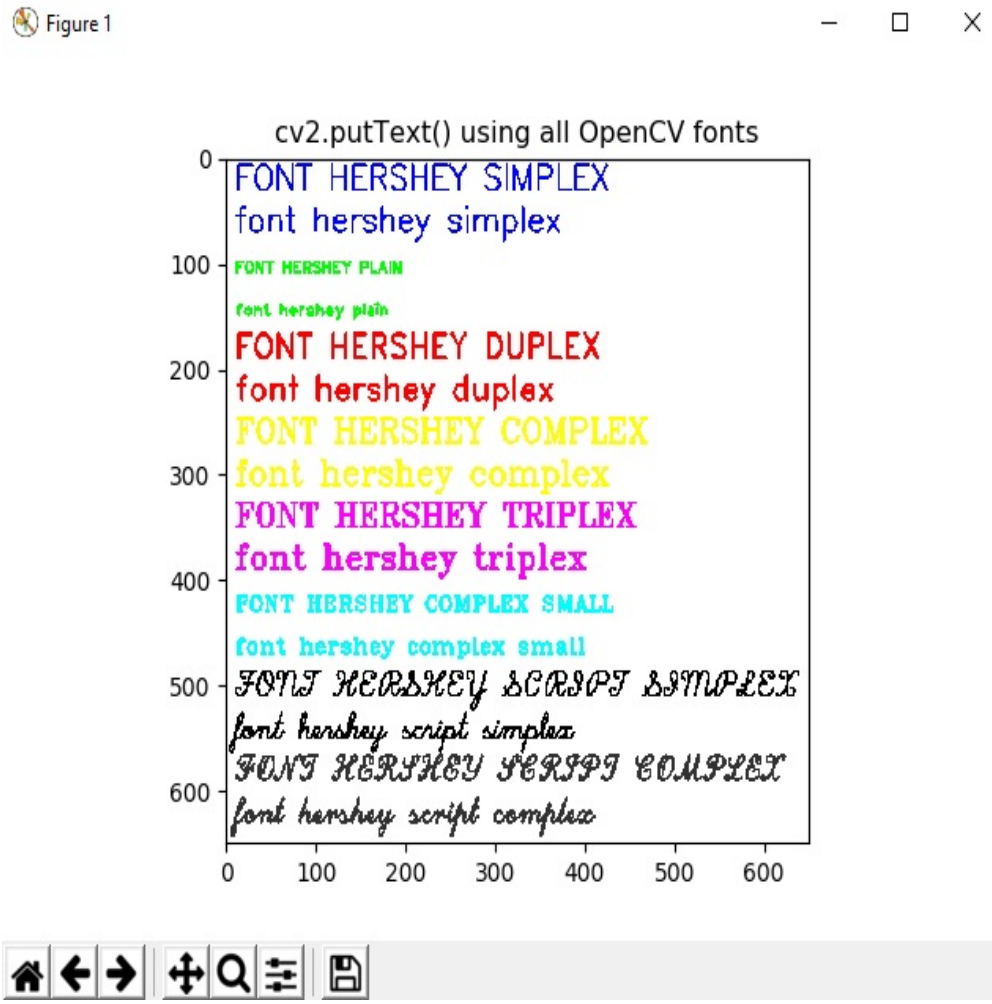
OpenCV中所有可用的字体如下

```
FONT_HERSHEY_SIMPLEX = 0
FONT_HERSHEY_PLAIN = 1
FONT_HERSHEY_DUPLEX = 2
FONT_HERSHEY_COMPLEX = 3
FONT_HERSHEY_TRIPLEX = 4
FONT_HERSHEY_COMPLEX_SMALL = 5
FONT_HERSHEY_SCRIPT_SIMPLEX = 6
FONT_HERSHEY_SCRIPT_COMPLEX =  7
```

与此相关，我们编写了text_drawing_font.py脚本，它将绘制所有可用的字体。由于所有这些字体都在(0-7)范围内，我们可以迭代并调用cv2.putText()函数，改变颜色、字体和org参数。我们还绘制了这些字体的小写和大写版本。执行此功能的关键代码如下

```
position = (10, 30) for i in range(0, 8):
cv2.putText(image, fonts[i], position, i, 1.1,
cv2.putText(image, fonts[i].lower(), position,
```

生成的屏幕截图如下所示



在前面的屏幕截图中，可看到OpenCV中所有小写和大写字体。可以使用此屏幕截图作为参考，以便轻松检查想在项目中使用的字体。

更多与文本相关的函数

OpenCV提供了更多与文本绘图相关的函数。需要注意的是，这些函数不是用来绘制文本的，但可以用来补充前面提到的cv2.putText()函数，注释在下

面。的第一个函数是
cv2.getFontScaleFromHeight()，函数签名

```
retval = cv2.getFontScaleFromHeight(fontFace,
```

这个函数返回字体比例（fontScale)，是cv2.putText()函数中使用的一个参数，用于实现以像素为单位的高度，同时考虑字体类型(fontFace)和厚度。

第二个函数是cv2.getTextSize()

```
retval, baseLine = cv2.getTextSize(text, fontFa
```

此函数可用于根据要绘制的参数文本、字体类型(fontFace)、比例和厚度获取文本大小(宽度和高度)。函数返回大小和基线，应于基线相对于文本底部的y坐标。下一段代码展示的关键点。完整的代码见text_drawing_bounding_box.py

```
#  Assign parameters to be used in the drawing
<p class="mume-header " id="assign-parameters-t

font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 2.5
thickness = 5
text = 'abcdefghijklmnopqrstuvwxyz'
circle_radius = 10

#  We get the size of the text:
<p class="mume-header " id="we-get-the-size-of-

ret, baseline = cv2.getTextSize(text, font, fon

#  We get the text width and text height from r
<p class="mume-header " id="we-get-the-text-wid

text_width, text_height = ret

#  We center the text in the image:
<p class="mume-header " id="we-center-the-text-

text_x = int(round((image.shape[1] - text_width
text_y = int(round((image.shape[0] + text_heigh
```

```
#  Draw this point for reference:
<p class="mume-header " id="draw-this-point-for

cv2.circle(image, (text_x, text_y), circle_radi

#  Draw the rectangle (bounding box of the text
<p class="mume-header " id="draw-the-rectangle-

cv2.rectangle(image, (text_x, text_y + baseline
              colors['blue'], thickness)

#  Draw the circles defining the rectangle:
<p class="mume-header " id="draw-the-circles-de

cv2.circle(image, (text_x, text_y + baseline),
cv2.circle(image, (text_x + text_width - thickn

#  Draw the baseline line:
<p class="mume-header " id="draw-the-baseline-l

cv2.line(image, (text_x, text_y + int(round(thi

#  Write the text centered in the image:
<p class="mume-header " id="write-the-text-cent
```

```
cv2.putText(image, text, (text_x, text_y), font
```

示例输出见截屏

注意红色、青色和绿色三个小点是如何绘制的，黄色基线是如何显示的。

动态绘图与鼠标事件

在本节中将学习如何使用鼠标事件执行动态绘图。我们将会看到一些例子，它们的复杂度是递增的。

画动态图形

下一示例介绍如何使用OpenCV处理鼠标事件。cv2.setMouseCallback()函数执行这个功能。此方法的签名如下

```
cv2.setMouseCallback(windowName, onMouse, par
```

此函数为名为windowName的窗口建立鼠标处理程序。onMouse函数是回调函数，执行鼠标事件时调用它，例如双击、左键向下、左键向上等。可选参数param用于向回调函数传递附加信息。

第一步是创建回调函数

```
#  This is the mouse callback function:
<p class="mume-header " id="this-is-the-mouse-c

def draw_circle(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        print("event: EVENT_LBUTTONDBLCLK")
        cv2.circle(image, (x, y), 10, colors['m

    if event == cv2.EVENT_MOUSEMOVE:
        print("event: EVENT_MOUSEMOVE")

    if event == cv2.EVENT_LBUTTONUP:
        print("event: EVENT_LBUTTONUP")

    if event == cv2.EVENT_LBUTTONDOWN:
        print("event: EVENT_LBUTTONDOWN")
```python

draw_circle()函数是响应每个鼠标事件，接收坐标(x, y

还打印了一些消息以查看其他事件，但不执行任何其他操作

下一步是创建一个命名窗口。在本例中，名为Image mouse

```python
```

```
#  We  create a named window where the mouse ca
<p class="mume-header " id="we-create-a-named-w

cv2.namedWindow('Image mouse')
```

最后，鼠标回调函数设置为之前创建的函数，或激活，

```
#   We  set  the  mouse  callback  function  to
<p class="mume-header " id="we-set-the-mouse-ca

cv2.setMouseCallback('Image  mouse',  draw_circ
```

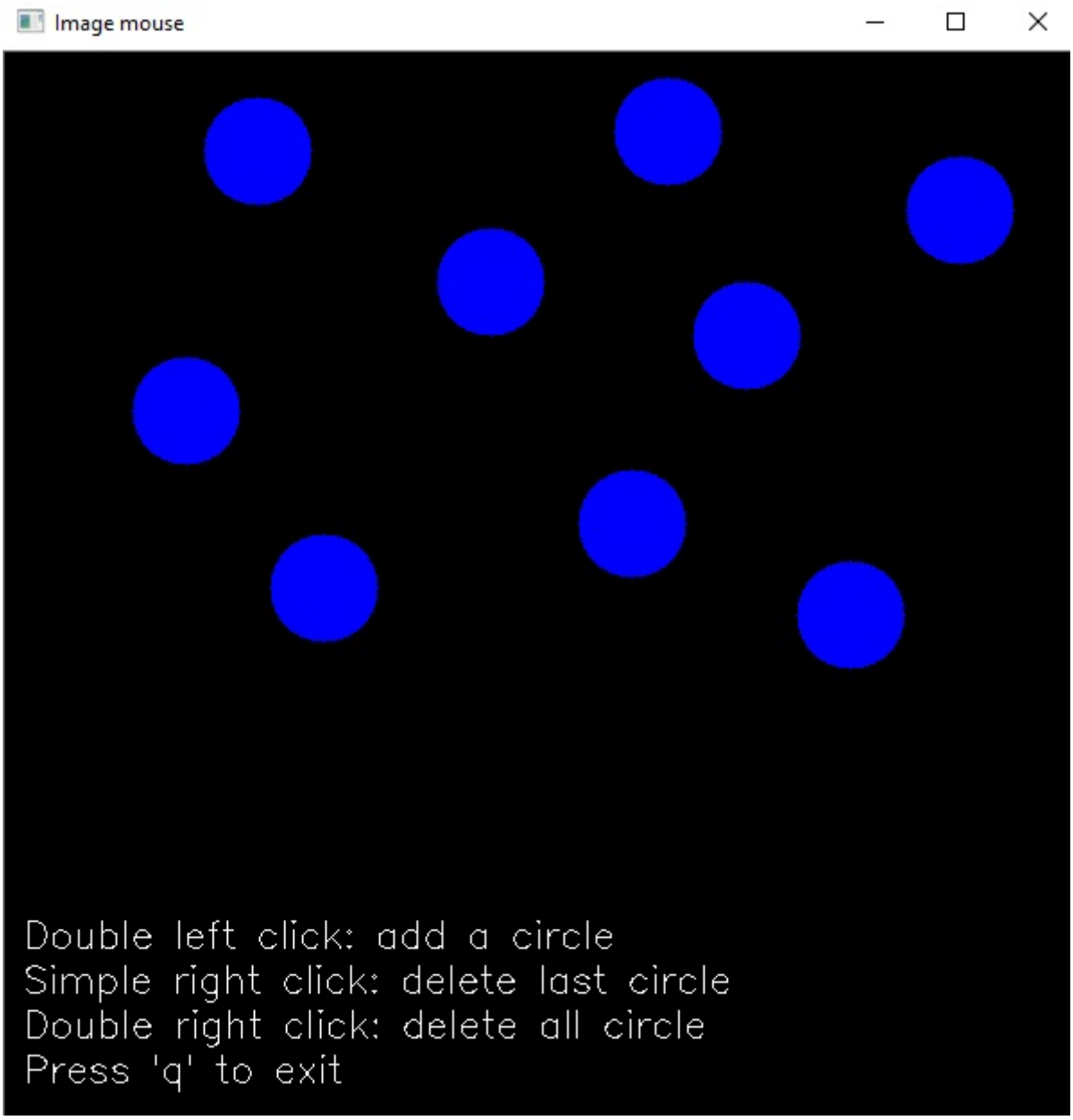
总之，当执行左双击时，在双击(x，y)位置的中心绘制一个填充洋红色的圆。完整代码见

mouse_draw.py

绘制文本和形状

在本例中鼠标事件和绘图文本结合，画出文本显示如何使用鼠标事件来执行特定的操作。为了更好地理

解这个例子，在屏幕截图可看到文本



可以执行以下操作

双击左键添加一个圆，单击右键删除最后添加的圆，双击右键删除所有圆。

为完成此功能，创建了一个名为circles的列表，管理用户生成的圆，可选择当前圆。还用渲染文本创

建备份图像。当鼠标事件产生时，从循环列表中添加或删除图形。在绘图时，只从列表中绘制当前的圆。例如，当用户右键单击时，最后添加的圆从列表中删除。完整代码见mouse_drawing_circles_and_text.py。

事件处理与Matplotlib

前面示例可看到，没有用Matplotlib显示图像。Matplotlib可以处理事件处理和拾取，可用Matplotlib功能捕获鼠标事件。可通过Matplotlib连接更多事件(https://matplotlib.org/users/event_handling.html)。例，在连接鼠标时，有以下事件button_press_event、button_release_event、motion_notify_event和scroll_event。

一个简单示例，在鼠标单击button_press_event事件时画一个圆

```
#  'button_press_event' is a MouseEvent where a
<p class="mume-header " id="button_press_event-

#  When this event happens the function 'click_
<p class="mume-header " id="when-this-event-hap
```

我们还必须为button_press_event事件定义事件监听器

```
#  We define the event listener for the 'button
<p class="mume-header " id="we-define-the-event

def click_mouse_event(event):
    # (event.xdata, event.ydata) contain the fl
    cv2.circle(image, (int(round(event.xdata)),
    # Call 'update_image()' method to update th
    update_img_with_matplotlib()


当鼠标单击时，绘制一个蓝色圆。编写了update_img_wit

#  高级绘图
<p class="mume-header " id="高级绘图"></p>


本节中将看到如何组合函数来在OpenCV中绘制基本形状，例

analog_clock_values.py
analog_clock_opencv.py

analog_clock_opencv.py使用cv.line()、cv.circle()

```python
# Coordinates to define the origin for the hour
```
```

```
hours_orig = np.array(
    [(620, 320), (580, 470), (470, 580), (320,
     (469, 60), (579, 169)])

# Coordinates to define the destiny for the hou
hours_dest = np.array(
    [(600, 320), (563, 460), (460, 562), (320,
     (459, 77), (562, 179)])
```

这些数组绘制小时标记，每小时的时钟线，起点和刻度。标记是这样画的

```
# We draw the hour markings:
for i in range(0, 12):
    cv2.line(image, array_to_tuple(hours_orig[i
```

另外，画出模拟时钟形状相对应的一个大圆

```
cv2.circle(image, (320, 320), 310, colors['
```

用Python绘制包含主控OpenCV 4的矩形，绘制时钟

```
cv2.rectangle(image, (150, 175), (490, 270), co
cv2.putText(image, "Mastering OpenCV 4", (15
cv2.putText(image, "with Python", (210, 250), 1
```

在图像中绘制静态信息后将其复制到image_original图像中

```
image_original = image.copy()
```

为了绘制动态信息，需要执行几个步骤

1. 从当前时间中获取小时、分钟和秒

```
# Get current date:
date_time_now = datetime.datetime.now()
# Get current time from the date:
time_now = date_time_now.time()
# Get current hour-minute-second from the time:
hour = math.fmod(time_now.hour, 12)
minute = time_now.minute
second = time_now.second
```

2. 将小时、分钟和秒转换为角度

```
# Get the hour, minute and second angles:
second_angle = math.fmod(second * 6 + 270, 360)
minute_angle = math.fmod(minute * 6 + 270, 360)
hour_angle = math.fmod((hour * 30) + (minute /
```

3. 画出与小时、分钟、秒钟相对应的线

```
# Draw the lines corresponding to the hour, min
    second_x = round(320 + 310 * math.cos(secon
    second_y = round(320 + 310 * math.sin(secon
    cv2.line(image, (320, 320), (second_x, seco

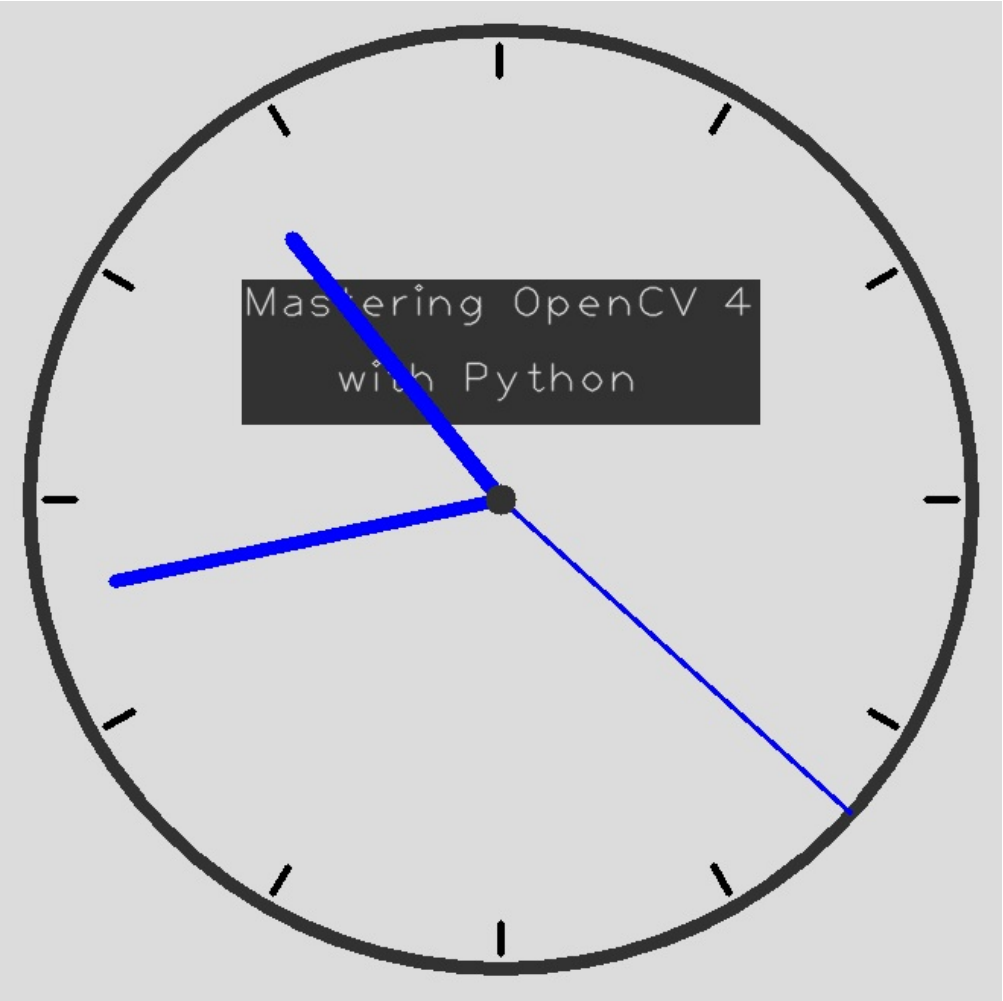
    minute_x = round(320 + 260 * math.cos(minut
    minute_y = round(320 + 260 * math.sin(minut
    cv2.line(image, (320, 320), (minute_x, minu

    hour_x = round(320 + 220 * math.cos(hour_an
    hour_y = round(320 + 220 * math.sin(hour_an
    cv2.line(image, (320, 320), (hour_x, hour_y
```

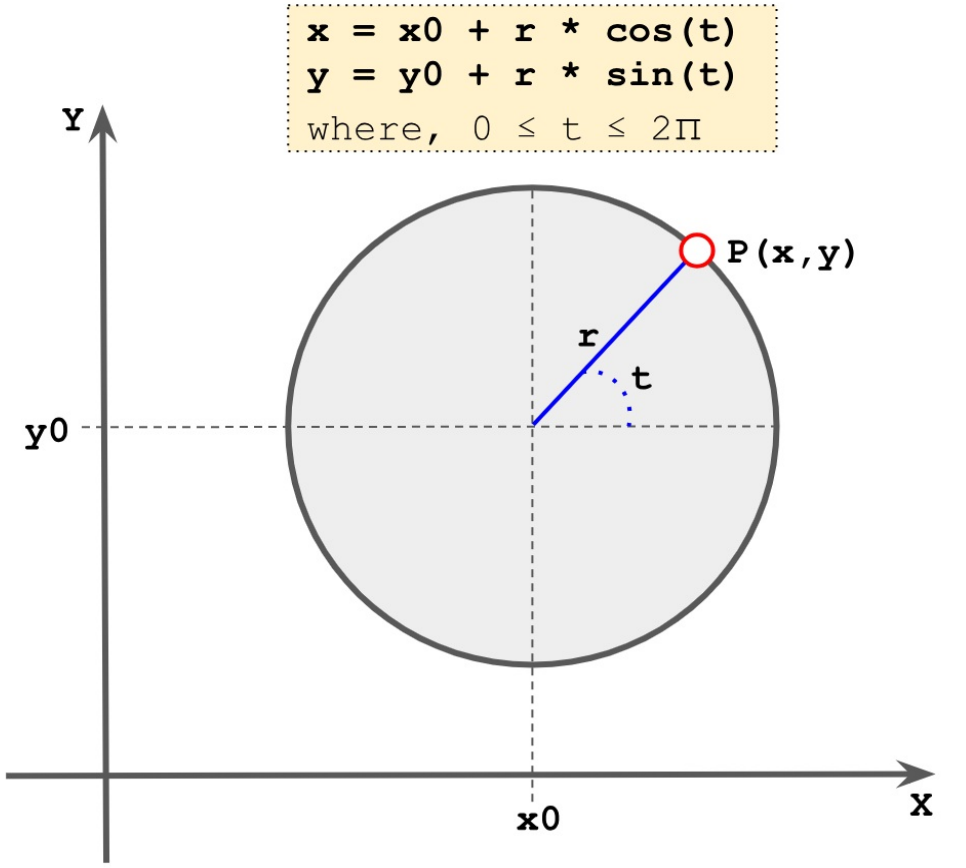
4. 最后，画三针连接点的一个小圆

```
cv2.circle(image, (320, 320), 10, colors['dark_
```

在下一个屏幕截图中，你可以看到模拟时钟的样子



analog_clock_values.py脚本计算hours_orig和hours_dest数组的固定坐标。为了计算小时的(x, y)坐标，使用圆参数方程，如下面的截图所示



The parametric equations of a translated circle with center (x0, y0) and radius r

按照前面截图公式计算了12个点P(x，y)每隔30度的坐标，并从0度开始，

0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300和330，并有两个不同的半径。就定下小时段的坐标。计算这些坐标的代码如下

```
radius = 300
center = (320, 320)

for x in (0, 30, 60, 90, 120, 150, 180, 210, 24
    x_coordinate = center[0] + radius * math.co
    y_coordinate = center[1] + radius * math.si
    print("x: {} y: {}".format(round(x_coordina

print("..............")

for x in (0, 30, 60, 90, 120, 150, 180, 210, 24
    x_coordinate = center[0] + (radius - 20) *
    y_coordinate = center[1] + (radius - 20) *
    print("x: {} y: {}".format(round(x_coordina
```

完整代码见analog_clock_values.py。应该注意的是，我们可以在另一个脚本中包含计算这些坐标的代码，但这是一个很好的练习。

小结

在本章中，回顾了OpenCV与绘制形状和文本相关的功能。已经了解如何绘制基本形状(直线、矩形和圆形)，以及更高级的形状(剪辑线、箭头、椭圆和多边形)。如何绘制文本，以及如何在OpenCV库中绘制所有字体。此外还介绍了如何捕获鼠标事件并执行特定的操作(例如，鼠标事件的所在点(x，y)绘制点)。最后，绘制了一个模拟时钟，运用前面介绍的所有概念。

下一章将看到有关图像处理技术的主要概念。处理基本的图像转，例如，平移、旋转、调整大小、翻转和裁剪。另一个关键点是如何对图像执行基本的算术运算，比如按位操作(AND，OR，XOR，AND NOT)。最后将介绍主要的颜色空间和颜色映射。

练习

1. 绘制填充形状，例如，圆形或矩形，应配置哪个参数？

2. 绘制反锯齿的线型，应配置哪个参数?
3. 绘制一条从(0,0)到(512,512)的对角线。
4. 用所需的参数绘制“Hello OpenCV”。
5. 用12个点画一个多边形，形状为圆形。
6. 用鼠标事件和Matplotlib事件在双击时绘制矩形。
7. 用lenna.png图像作为背景来绘制一个简单的meme generator