

# Parking Template

[http://mrl.cs.vsb.cz/data/vyuka/zao/parking/parking\\_zao\\_template.zip](http://mrl.cs.vsb.cz/data/vyuka/zao/parking/parking_zao_template.zip)

The screenshot displays a Python IDE with a project named 'parking\_21\_2\_gred'. The main.py file contains the following code:

```

72 def main(argv):
73
74     pkm_file = open('parking_map_python.txt', 'r')
75     pkm_lines = pkm_file.readlines()
76     pkm_coordinates = []
77
78     for line in pkm_lines:
79         st_line = line.strip()
80         sp_line = list(st_line.split(" "))
81         pkm_coordinates.append(sp_line)
82
83     test_images = [img for img in glob.glob("test_images_zao/*.jpg")]
84     test_images.sort()
85     print(pkm_coordinates)
86     print("*****")
87     print(test_images)
88     cv.namedWindow("one_place_img_res", 0)
89     cv.namedWindow("image", 0)
90     for img_name in test_images:
91
92         image = cv.imread(img_name, 1)
93         cv.imshow("image", image)
94
95         for coord in pkm_coordinates:
96             print("coord", coord)
97             one_place_img = four_point_transform(image, coord)
98             one_place_img_res = cv.resize(one_place_img, (80, 80))
99             cv.imshow("one_place_img_res", one_place_img_res)
100             cv.waitKey(0)
101
102     if cv.waitKey(0) == 27:
103         break

```

The IDE's Run console shows the output of the script, listing the coordinates for each parking spot:

```

Run: main
coord ['149', '699', '228', '705', '141', '879', '61', '864']
coord ['230', '706', '317', '712', '233', '898', '143', '881']
coord ['317', '712', '415', '718', '340', '913', '233', '898']
coord ['417', '718', '527', '722', '461', '928', '340', '915']
coord ['527', '723', '650', '723', '589', '941', '463', '930']
coord ['651', '725', '780', '728', '745', '950', '599', '942']
coord ['781', '728', '910', '728', '909', '950', '748', '951']
coord ['916', '728', '1054', '726', '1070', '953', '910', '959']

```

The main window displays an aerial view of a parking lot with a red car in the foreground. A smaller window titled 'one\_place\_img\_res' shows a zoomed-in view of the red car, demonstrating the four-point transform used for object detection.

## Drawing Line

To draw a line, you need to pass starting and ending coordinates of line. We will create a black image and draw a blue line on it from top-left to bottom-right corners.

```
import numpy as np
import cv2 as cv

# Create a black image
img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
cv.line(img,(0,0),(511,511),(255,0,0),5)
```

## Drawing Rectangle

To draw a rectangle, you need top-left corner and bottom-right corner of rectangle. This time we will draw a green rectangle at the top-right corner of image.

```
cv.rectangle(img,(384,0),(510,128),(0,255,0),3)
```

## Drawing Circle

To draw a circle, you need its center coordinates and radius. We will draw a circle inside the rectangle drawn above.

```
cv.circle(img,(447,63), 63, (0,0,255), -1)
```

## Adding Text to Images:

To put texts in images, you need specify following things.

- Text data that you want to write
- Position coordinates of where you want put it (i.e. bottom-left corner where data starts).
- Font type (Check `cv.putText()` docs for supported fonts)
- Font Scale (specifies the size of font)
- regular things like color, thickness, `lineType` etc. For better look, `lineType = cv.LINE_AA` is recommended.

We will write **OpenCV** on our image in white color.

```
font = cv.FONT_HERSHEY_SIMPLEX  
cv.putText(img, 'OpenCV', (10, 500), font, 4, (255, 255, 255), 2, cv.LINE_AA)
```

# Parking Template

```
def main(argv):
    pkm_file = open('parking_map_python.txt', 'r')
    pkm_lines = pkm_file.readlines()
    pkm_coordinates = []

    for line in pkm_lines:
        st_line = line.strip()
        sp_line = list(st_line.split(" "))
        pkm_coordinates.append(sp_line)

    test_images = [img for img in glob.glob("test_images_zao/*.jpg")]
    test_images.sort()
    temp = cv.imread("template.png")
    size = (80, 80)
    temp = cv.resize(temp, size)
    cv.namedWindow("image_clone", 0)
    cv.namedWindow("one_place_img", 0)
    cv.namedWindow("temp", 0)
    cv.namedWindow("source", 0)
```

Prepare/Load:  
+ template  
+ testing images  
+ parking coordinates

What type of template?

```
n_park = 0
font = cv.FONT_HERSHEY_PLAIN
for img_name in test_images:
    image = cv.imread(img_name)
    image_clone = image.copy()
    cv.imshow("image", image)

    for coord in pkm_coordinates:
        n_park+=1
        print("coord", coord)

        one_place_img = four_point_transform(image, coord)
        one_place_img = cv.resize(one_place_img, size)
```

Reading coordinates and obtain  
one parking place e.g. image  
80x80, 120x120, ...

Template Matching

```
cv.putText(image_clone, str(n_park), (int(center[0]), int(center[1])), font, 3, (255, 255, 255), 2)
cv.imshow("image_clone", image_clone)
n_park = 0
cv.waitKey(0)
```

# Parking Template

## Result



# Parking Template

- **TASK – parking lot occupation detection**

[http://mrl.cs.vsb.cz/data/vyuka/zao/parking/parking\\_zao\\_template.zip](http://mrl.cs.vsb.cz/data/vyuka/zao/parking/parking_zao_template.zip)

- create visualization of the results (which place is occupied and which is free)
- experiments with the template matching methods from OpenCV (try several templates etc.)
- read .txt file with ground truth data of each parking place image (in test\_images\_zao folder) and calculate accuracy and F-core: [https://en.wikipedia.org/wiki/Evaluation\\_of\\_binary\\_classifiers](https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers)
- try to create custom code for template matching: own version of TM\_CCOEFF\_NORMED and TM\_SQDIFF\_NORMED and compare it to the functions in the OpenCV.