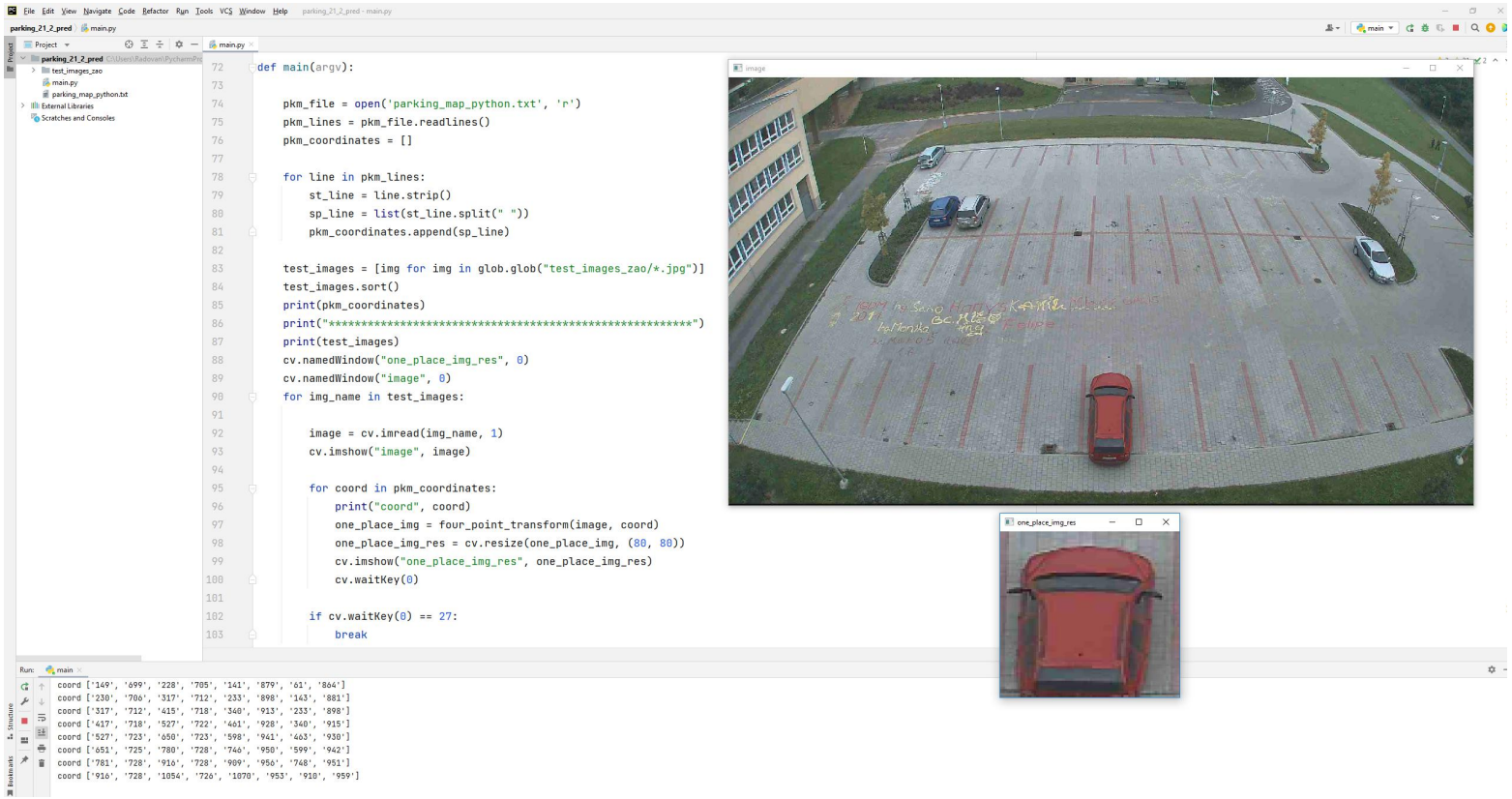
Parking Template

http://mrl.cs.vsb.cz/data/vyuka/zao/parking/parking_zao_template.zip



Parking Template

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)
```


https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

Parking Template

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)
```


https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

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)
    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)

            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)
```

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

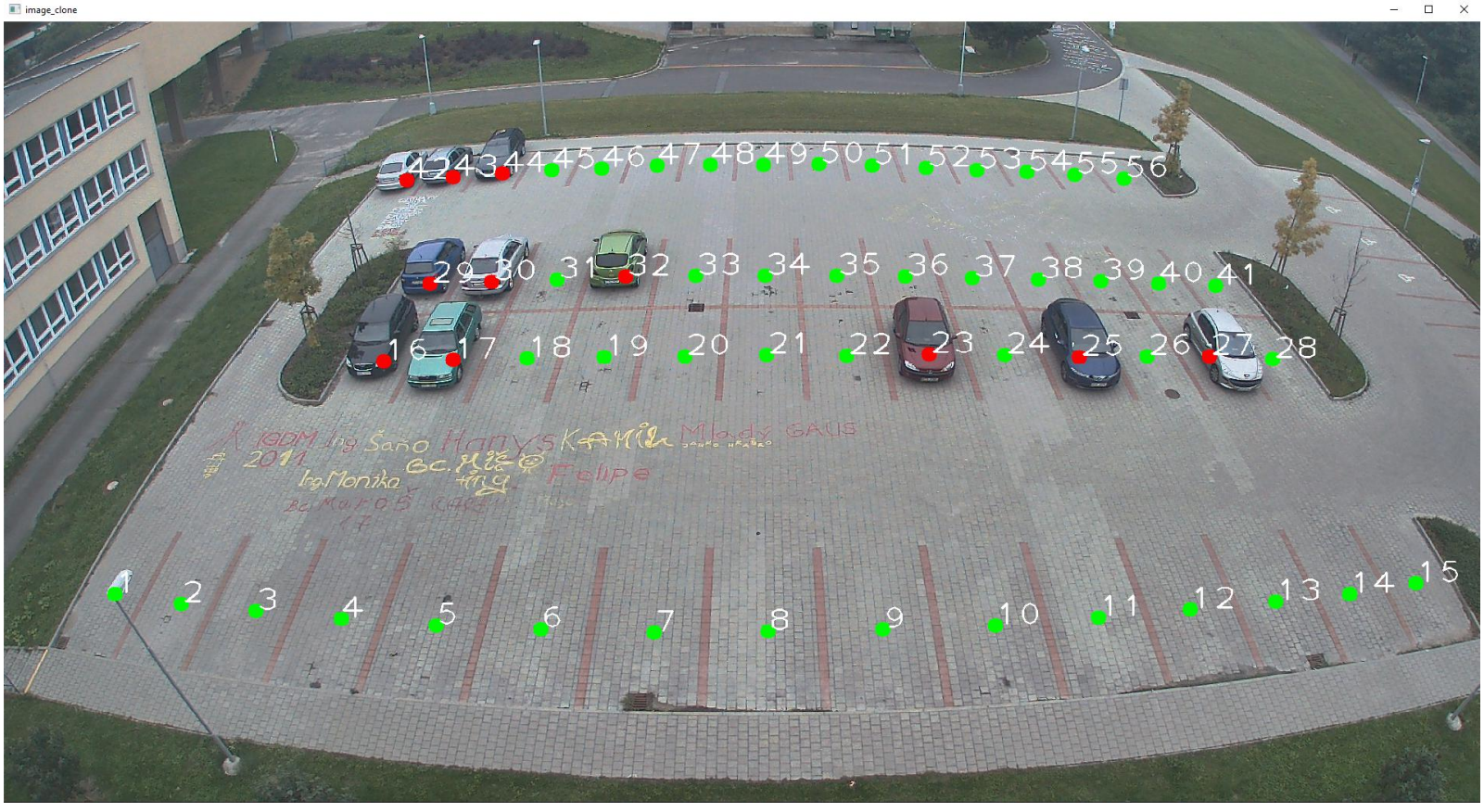
What type of template?

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

Template Matching

Parking Template

Result



Parking Template

- **TASK – parking lot occupation detection**

  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

- 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.

https://stackoverflow.com/questions/58158129/understanding-and-evaluating-template-matching-methods