QR Code Detector in InnoBioDiv Project

Teng Tian
last edited: 01.02.2024

Background: In the previous approach, the detection of the flower pot used in the project is based on the Hough Circle Algorithm. This method can be very inaccurate under various circumstances, for example, if the circle shape of the flower pot is not completely shown in the captured image. Nevertheless, we have to deal with multiple circles if there are more than one flower pots in the image. **Advantages of QR Code:** QR code, due to its high contrast and monochrome color, can be very easily recognized by computer vision. It gives information not only about the position but also about the scale of pixels in the image (assuming the length of the QR code is given). **Goal of this attempt:** Since the flower pots are placed in standard tracks, the relative distance of the pots to the track is given. So there must be an easier and more efficient way to determine the positions of the pots from the captured image. And there must also be a way to detect multiple pots on one track.

In [ ]:
```
import cv2 as cv #OpenCV
import numpy as np
from matplotlib import pyplot as plt
```

About QR-Code detector in OpenCV

Due to the mounting position of the camera on the FarmBot, only one QR code will be captured at a time according to the current experiment setting in the greenhouse. OpenCV provides a QRdetector class that includes all functions for detecting and decoding QR codes. The Python code is shown here:

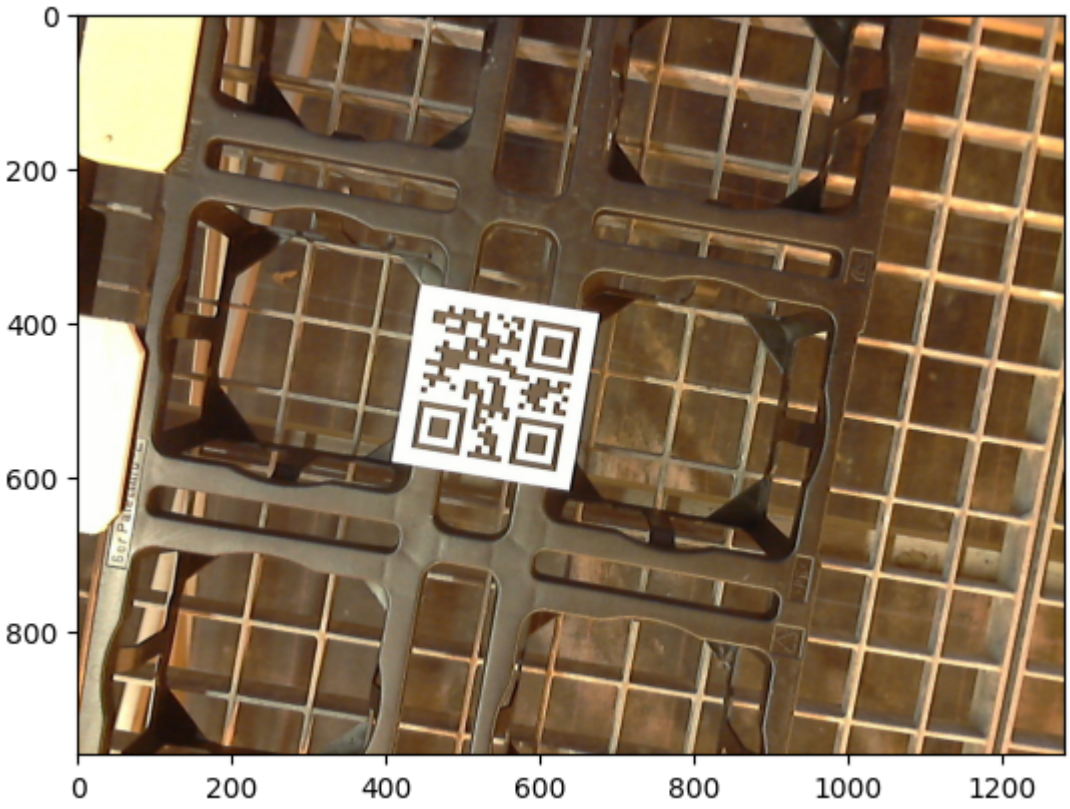
```
retval, points, straight_qrcode =
QRdetector.detectAndDecode(image)
```

The function returns three values:

- ***retval***: will be TRUE if a QR Code was successfully detected in the image.
- ***point***: returns a list of the coordinates of the four corners. The corners are always sorted clockwise. The upper left corner of the displayed QR Code is always the second element of the list.
- ***straight_qrcode***: returns the stored data of the QR code as a string, here '0xE3CC6E0G'. This data can be changed to any possible content that a version 1 QR code can hold, such as 'test_plant_01'.

Problem 1: Lighting conditions in the greenhouse

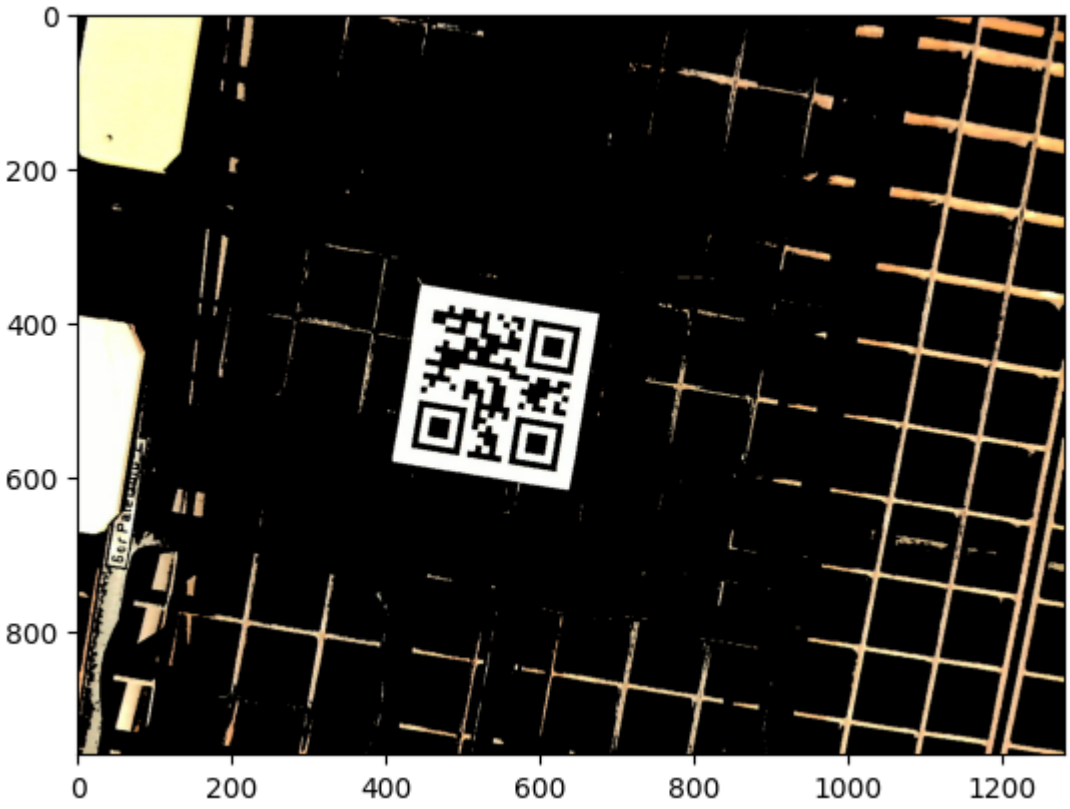
In [ ]:
```
# read an image
image=cv.imread(r'../data/Farmbox_QR/6.jpeg')
plt.figure()
plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
plt.show()
```



As shown in the image, the lighting conditions in the greenhouse cause a red tint to all the pixels, which needs to be corrected.

In [ ]:
```
# Convert the image to HSV color space
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
# Define the color range for filtering the QR code area
H_MAX=179
H_MIN=0
S_MAX=255
S_MIN=0
V_MAX=255
V_MIN=200
# Set the filter range
lower_color = (H_MIN, S_MIN, V_MIN)
upper_color = (H_MAX, S_MAX, V_MAX)
# Create a mask to filter the specified color range
mask = cv.inRange(hsv_image, lower_color, upper_color)
# Apply the mask to the original image
result = cv.bitwise_and(image, image, mask=mask)

plt.figure()
plt.imshow(cv.cvtColor(result, cv.COLOR_BGR2RGB))
plt.show()
```



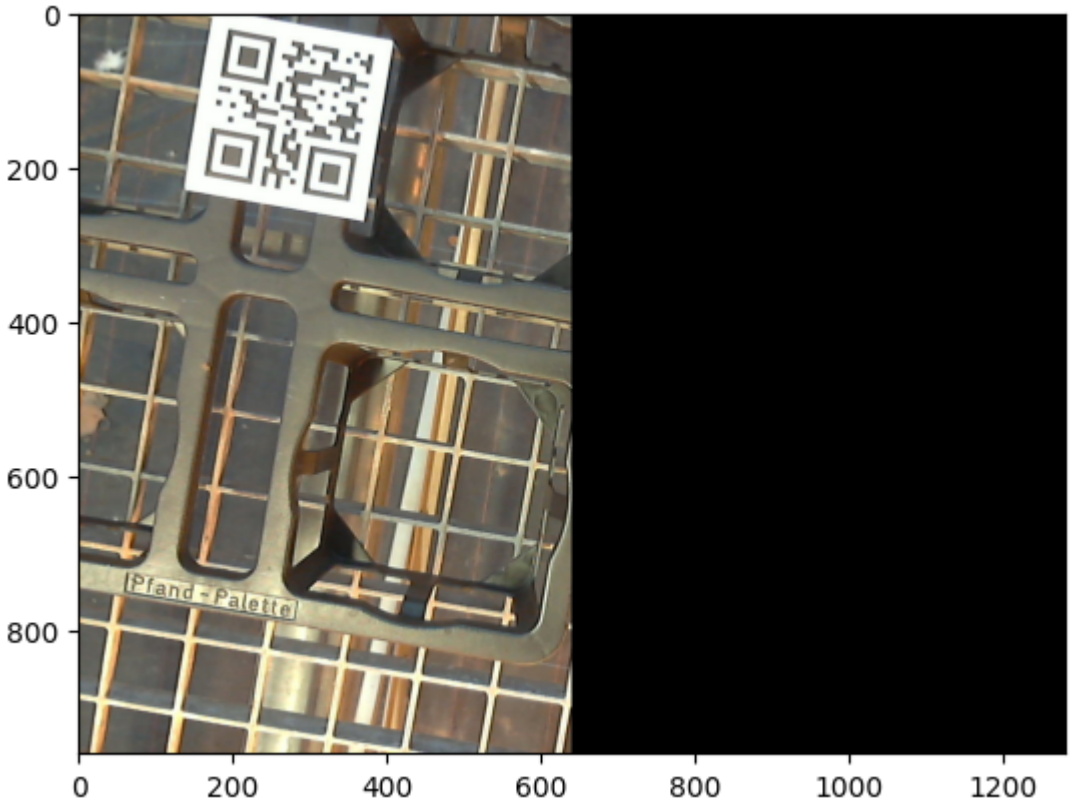
After filtering, the color of the QR code becomes normal and the distracting background is reduced to a minimum.

Problem 2: Off-centered QR code capture

The QR code detection provided by OpenCV sometimes misses the QR code if the QR code is near the edge of the image. This can be solved or at least improved by splitting the image into different areas, what I tried here is to cover half of the image with black color, or in mathematical terms: change the value of the pixel to zero.

In [ ]:
```
# Get the dimensions of the image
height, width, channels = image.shape
# Define the width of the black region you want to add (you can adjust this valu
black_region_width = int(width/2)
# Create a black image with the same height as the original image and the specif
black_region = np.zeros((height, black_region_width, channels), dtype=np.uint8)
# Concatenate the black image with the left side of the original image
result_image = np.concatenate((image[:, :-black_region_width, :], black_region),

plt.figure()
plt.imshow(cv.cvtColor(result_image, cv.COLOR_BGR2RGB))
plt.show()
```



Here is the part of the program that detects the QR code and calculates the positions of all pots on a track.

In [ ]:
```
# create detector object
LENGTH_OF_LABEL = 5.5 #cm
DIA_OF_POT = 5 #cm
LENGTH_A = 7.5 #cm
LENGTH_B = 13.5 #cm
QRdetector = cv.QRCodeDetector() #create a detector object


# make a copy of the origin image
image_display = image.copy()
result_image = result.copy()
# Get the dimensions of the image
height, width, channels = image_display.shape
# Define the width of the black region you want to add (you can adjust this valu
black_region_width = int(width/2)
```

```
for i in range(0,3):
        retval, points, straight_qrcode = QRdetector.detectAndDecode(result_imag
        if retval:

                centerQR = (points[0].sum(axis=0)/4).astype(int)
                scale_cm2pix = cv.arcLength(points[0], closed=True)/(4*LENGTH_OF
                dia_pot_pix = scale_cm2pix * DIA_OF_POT
                length_a_pix = scale_cm2pix * LENGTH_A
                length_b_pix = scale_cm2pix * LENGTH_B

                vector = points[0][0] - points[0][1]
                angle_rad = np.arctan2(vector[1], vector[0])
                angle_deg = np.degrees(angle_rad)
                #Pot left to QR
                center_x_L = centerQR[0] - length_a_pix * np.cos(angle_rad)
                center_y_L = centerQR[1] - length_a_pix * np.sin(angle_rad)
                #Pot right to QR
                center_x_R = centerQR[0] + length_a_pix * np.cos(angle_rad)
                center_y_R = centerQR[1] + length_a_pix * np.sin(angle_rad)
                #Pot top-left tp QR
                center_x_TL = center_x_L + length_b_pix * np.sin(angle_rad)
                center_y_TL = center_y_L - length_b_pix * np.cos(angle_rad)
                #Pot top-right tp QR
                center_x_TR = center_x_R + length_b_pix * np.sin(angle_rad)
                center_y_TR = center_y_R - length_b_pix * np.cos(angle_rad)
                #Pot bottom-left tp QR
                center_x_BL = center_x_L - length_b_pix * np.sin(angle_rad)
                center_y_BL = center_y_L + length_b_pix * np.cos(angle_rad)
                #Pot bootm-right tp QR
                center_x_BR = center_x_R - length_b_pix * np.sin(angle_rad)
                center_y_BR = center_y_R + length_b_pix * np.cos(angle_rad)

                cv.circle(img=image_display, center=(int(center_x_L),int(center_
                cv.circle(img=image_display, center=(int(center_x_R),int(center_
                cv.circle(img=image_display, center=(int(center_x_TL),int(center
                cv.circle(img=image_display, center=(int(center_x_TR),int(center
                cv.circle(img=image_display, center=(int(center_x_BL),int(center
                cv.circle(img=image_display, center=(int(center_x_BR),int(center


                cv.putText(image_display, 'L', (int(center_x_L),int(center_y_L))
                cv.putText(image_display, 'R', (int(center_x_R),int(center_y_R))
                cv.putText(image_display, 'TL', (int(center_x_TL),int(center_y_T
                cv.putText(image_display, 'TR', (int(center_x_TR),int(center_y_T
                cv.putText(image_display, 'BL', (int(center_x_BL),int(center_y_B
                cv.putText(image_display, 'BR', (int(center_x_BR),int(center_y_B

                cv.polylines(image_display, points.astype(int),isClosed=True,col
                cv.circle(img=image_display, center=centerQR, radius=4, color=(2

                cv.circle(img=image_display, center=points[0][0].astype(int), ra
                cv.circle(img=image_display, center=points[0][1].astype(int), ra
                str_length_QR=str(int(scale_cm2pix))+' px/cm'
                cv.putText(image_display, str_length_QR, (0,30), cv.FONT_HERSHEY

                plt.figure()
                plt.imshow(cv.cvtColor(image_display, cv.COLOR_BGR2RGB))
                plt.show()

                break # break for-loop
```

```
        else:
                if i==0:
                        # Concatenate the black image with the left side of the
                        result_image=result.copy()
                        result_image = np.concatenate((result[:, :-black_region_
                if i==1:
                        # Concatenate the black image with the right side of the
                        result_image=result.copy()
                        result = np.concatenate((black_region, result[:, :-black
                if i==2:
                        print('No QR-code found!')
```