

5. Opencv Application

5. Opencv Application

5.1. Overview

5.2, QR code

5.2.1, Introduction to QR code

5.2.2, QR code structure

5.2.3. Characteristics of QR Codes

5.2.4, QR code creation and recognition

5.3 Human Posture Estimation

5.3.1 Overview

5.3.2 Principle

5.3.3 Startup

5.4, Object Detection

5.4.1, Model structure

5.4.2, Code Analysis

5.4.3. Startup

5.1. Overview

OpenCV is a cross-platform computer vision and machine learning software library released under the BSD license (open source), which can run on Linux, Windows, Android and MacOS operating systems. [1] It is lightweight and efficient - consisting of a series of C functions and a small number of C++ classes, while providing interfaces for languages such as Python, Ruby, and MATLAB, and implementing many common algorithms in image processing and computer vision.

Please do this step before running the program








```
sudo supervisorctl stop ChassisServer #Stop the self-starting chassis service
#Perform this step more for the indoor version of NAVROBO-astra_pro2 camera
/home/yahboom/YBAMR-COBOT-EDU-00001/start/OBColorViewer #Release color stream
video100
#[info][722318][Pipeline.cpp:251] Start streams done!
#[info][722318][Pipeline.cpp:234] Pipeline start done!
#[warning][722318][Pipeline.cpp:327] wait for frame timeout, you can try to
increase the wait time! current timeout=100
```

5.2, QR code

5.2.1, Introduction to QR code

QR code is a type of two-dimensional barcode, QR The abbreviation of "Quick Response" in English means quick response. It comes from the inventor's hope that QR code can allow its content to be decoded quickly. QR code not only has large information capacity, high reliability and low cost, but also can represent a variety of text information such as Chinese characters and images, has strong confidentiality and anti-counterfeiting properties and is very convenient to use. More importantly, the QR code technology is open source.

5.2.2, QR code structure

Image	Analysis
	Positioning markings indicate the direction of the QR code.
	Alignment markings If the QR code is large, these additional elements help with positioning.
	Timing pattern Through these lines, the scanner can identify how large the matrix is.
	Version information This specifies the version number of the QR code being used. There are currently 40 different versions of QR codes. Versions used in the sales industry are usually 1-7.
	Format information The format mode contains information about error tolerance and data mask modes and makes scanning the code easier.
	Data and error correction keys These modes store the actual data.
	Quiet zone This area is very important for the scanner, and its function is to separate itself from the surroundings.

5.2.3. Characteristics of QR Codes

The data values in QR codes contain repeated information (redundant values). Therefore, even if up to 30% of the QR code structure is destroyed, it does not affect the readability of the QR code. The storage space of the QR code is up to 7089 bits or 4296 characters, including punctuation and special characters, which can be written into the QR code. In addition to numbers and characters, words and phrases (such as URLs) can also be encoded. As more data is added to the QR code, the code size increases and the code structure becomes more complex.

5.2.4, QR code creation and recognition

Source code path: /home/yahboom/YBAMR-COBOT-EDU-00001/src/yahboom_navrobo_other/yahboom_navrobo_visual/simple_qrcode

Installation

```
python3 -m pip install qrcode pyzbar
sudo apt-get install libzbar-dev
```

- Create

Create a qrcode object

```
'''
Parameter meaning:
    version: An integer from 1 to 40 that controls the size of the QR code (the
    minimum value is 1, which is a 12x12 matrix).
    If you want the program to determine it automatically, set the value to None
    and use the fit parameter.
    error_correction: Controls the error correction function of the QR code. The
    following 4 constants are available.
    ERROR_CORRECT_L: About 7% or less errors can be corrected.
    ERROR_CORRECT_M (default): About 15% or less errors can be corrected.
    ERROR_CORRECT_H: About 30% or less errors can be corrected.
    box_size: Controls the number of pixels contained in each small grid in the
    QR code.
    border: Controls the number of grids contained in the border (the distance
    between the QR code and the image boundary) (the default is 4, which is the
    minimum value specified by the relevant standards)
'''

qr = qrcode.QRCode( version=1,
error_correction=qrcode.constants.ERROR_CORRECT_H, box_size=5, border=4,)
```

QR code to add logo

```
# If the logo address exists, add the logo image
my_file = Path(logo_path)
if my_file.is_file(): img = add_logo(img, logo_path)
```

Note: When using Chinese, you need to add Chinese characters

Just use python3 + py file to execute, then enter the content to be generated and press Enter to confirm.

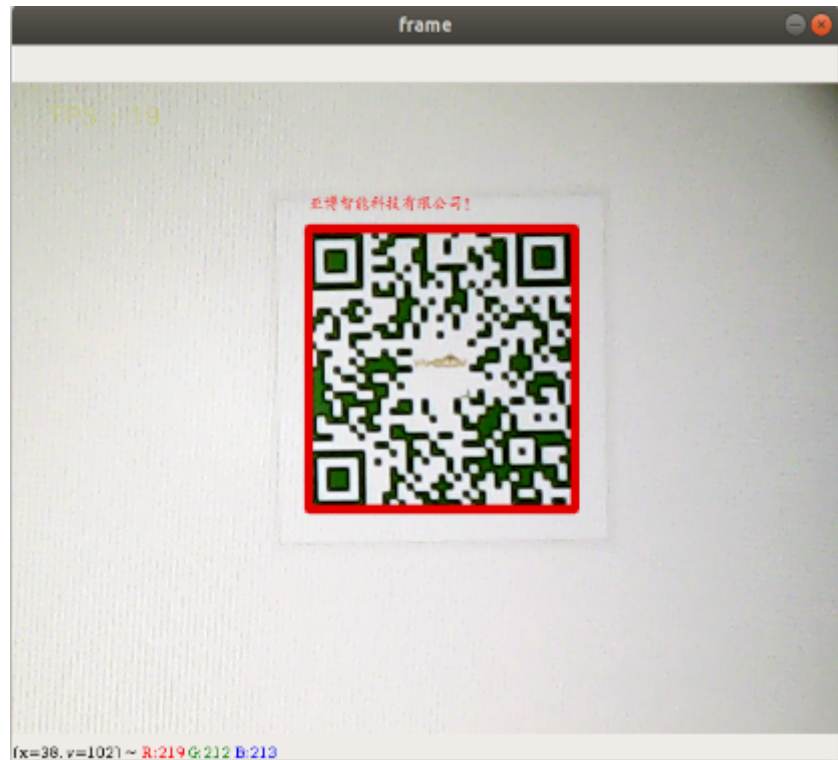


- Identification

```
def decodeDisplay(image, font_path):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    # You need to convert the output Chinese characters into Unicode encoding
    # first
    barcodes = pyzbar.decode(gray)
    for barcode in barcodes:
        # Extract the position of the bounding box of the QR code
        (x, y, w, h) = barcode.rect
        # Draw the bounding box of the barcode in the image
        cv.rectangle(image, (x, y), (x + w, y + h), (225, 0, 0), 5)
        encoding = 'UTF-8'
        # To draw it, you need to convert it into a string first
        barcodeData = barcode.data.decode(encoding)
        barcodeType = barcode.type
        # Draw the data and type on the image
        piling = Image.fromarray(image)
        # Create a brush
        draw = ImageDraw.Draw(piling)
        # Parameter 1: font file path, parameter 2: font size
        fontStyle = ImageFont.truetype(font_path, size=12, encoding=encoding)
        # Parameter 1: print coordinates, parameter 2: text, parameter 3: font
        # color, parameter 4: font
        draw.text((x, y - 25), str(barcode.data, encoding), fill=(255, 0, 0),
        font=fontStyle)
        # Convert PIL image to cv2 image
        image = cv.cvtColor(np.array(piling), cv.COLOR_RGB2BGR)
        # Print barcode data and barcode type to the terminal
        print("[INFO] Found {} barcode: {}".format(barcodeType, barcodeData))
    return image
```

- Effect demonstration

Just use python3 + py file to execute



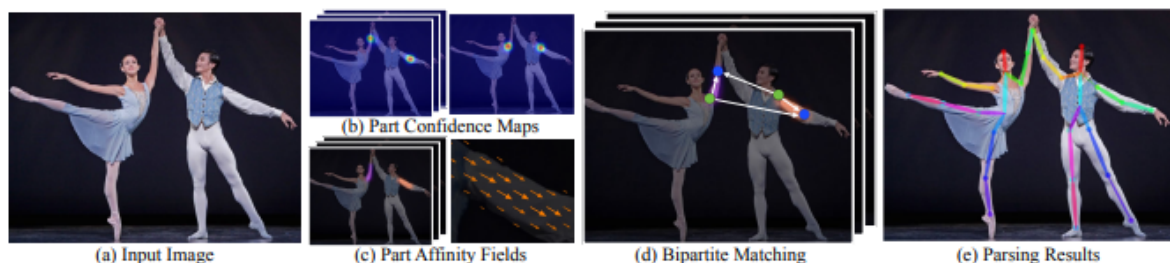
5.3 Human Posture Estimation

Source code path: /home/yahboom/YBAMR-COBOT-EDU-00001/src/yahboom_navrobo_other/yahboom_navrobo_visual/detection

5.3.1 Overview

Human Posture Estimation is to estimate the human posture by correctly connecting the key points of the human body detected in the image. The key points of the human body usually correspond to the joints with a certain degree of freedom on the human body, such as the neck, shoulder, elbow, wrist, waist, knee, ankle, etc., as shown in the figure below.

5.3.2 Principle



Input an image, extract features through the convolutional network, get a set of feature maps, and then split into two branches, using the CNN network to extract Part Confidence Maps and Part Affinity Fields respectively;

After obtaining these two pieces of information, we use Bipartite Matching in graph theory to find Part Association, connect the joints of the same person, and because of the vector nature of PAF itself, the generated bipartite matching is very correct, and finally merged into a person's overall skeleton;

Finally, based on PAFs, find Multi-Person Parsing—>Convert the Multi-person parsing problem into a graphs problem—>Hungarian Algorithm

(The Hungarian algorithm is the most common algorithm for partial graph matching. The core of the algorithm is to find augmenting paths. It is an algorithm that uses augmenting paths to find the maximum matching of bipartite graphs.)

5.3.3 Startup

```
cd /home/yahboom/YBAMR-COBOT-EDU-00001/src/yahboom_navrobo_other/yahboom_navrobo_visual/detection
python target_detection.py
```

After clicking the image frame, use the [f] key on the keyboard to switch target detection.

```
if action == ord('f') or action == ord('F'):state = not state # Function Switch
```

Input picture

Output image

5.4, Object Detection

The main problem solved in this section is how to use the dnn module in OpenCV to import a trained object detection network. However, there are requirements for the version of opencv.

At present, there are three main methods for object detection using deep learning:

- Faster R-CNNs
- You Only Look Once (YOLO)
- Single Shot Detectors (SSDs)

Faster R-CNNs is the most commonly heard neural network based on deep learning. However, this method is technically difficult to understand (especially for deep learning novices), difficult to implement, and difficult to train.

In addition, even if the "Faster" method is used to implement R-CNNs (here R stands for Region Proposal), the algorithm is still relatively slow, about 7FPS.

If we pursue speed, we can turn to YOLO, because it is very fast, reaching 40-90 FPS on TianXGPU, and the fastest version may reach 155 FPS. But the problem with YOLO is that its accuracy needs to be improved.

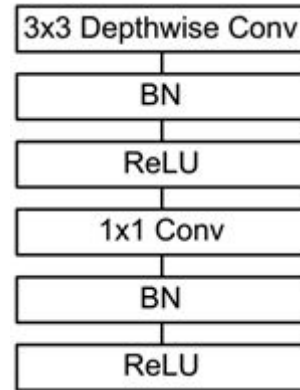
SSDs was originally developed by Google and can be said to be a balance between the above two. Compared with Faster R-CNNs, its algorithm is more direct. Compared with YOLO, it is more accurate.

5.4.1, Model structure

The main work of MobileNet is to replace the previous standard convolutions with depthwise separable convolutions to solve the computational efficiency and parameter quantity problems of convolutional networks. The MobileNets model is based on depthwise separable convolutions, which can decompose standard convolutions into a depthwise convolution and a point convolution (1×1 convolution kernel). **Deep convolution applies each convolution kernel to each channel, while 1×1 convolution is used to combine the outputs of channel convolutions.**

Batch Normalization (BN) is added to the basic components of MobileNet, that is, at each SGD (stochastic gradient descent), the standardization process is performed so that the result (each dimension of the output signal) has a mean of 0 and a variance of 1. Generally, when the convergence speed of the neural network training is very slow, or the gradient explodes and other conditions that cannot be trained, BN can be tried to solve the problem. In addition, BN can also be added in general use to speed up the training speed and improve the model accuracy.

In addition, the model also uses the ReLU activation function, so the basic structure of the depthwise separable convolution is shown in the figure below:



The MobileNets network is composed of many depthwise separable convolutions shown in the above figure. Its specific network structure is shown in the figure below:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5×	Conv dw / s1 $3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	Conv / s1 $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s1	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

5.4.2, Code Analysis

List of recognizable objects

```
[person, bicycle, car, motorcycle, airplane, bus, train,
truck, boat, traffic light, fire hydrant, street sign,
stop sign, parking meter, bench, bird, cat, dog, horse,
sheep, cow, elephant, bear, zebra, giraffe, hat, backpack,
umbrella, shoe, eye glasses, handbag, tie, suitcase,
frisbee, skis, snowboard, sports ball, kite, baseball bat,
baseball glove, skateboard, surfboard, tennis racket,
bottle, plate, wine glass, cup, fork, knife, spoon, bowl,
banana, apple, sandwich, orange, broccoli, carrot, hot dog,
pizza, donut, cake, chair, couch, potted plant, bed, mirror,
dining table, window, desk, toilet, door, tv, laptop, mouse,
remote, keyboard, cell phone, microwave, oven, toaster,
sink, refrigerator, blender, book, clock, vase, scissors,
teddy bear, hair drier, toothbrush]
```

Load the category [object_detection_coco.txt], import the model [frozen_inference_graph.pb], and specify the deep learning framework [TensorFlow]

```
# Load COCO class name
with open('object_detection_coco.txt', 'r') as f: class_names =
f.read().split('\n')
# Display different colors for different targets
COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))
# Load DNN image model
model = cv.dnn.readNet(model='frozen_inference_graph.pb',
config='ssd_mobilenet_v2_coco.txt', framework='TensorFlow')
```

Import the image, extract the height and width, calculate the 300x300 pixel blob, and pass this blob into the neural network

```
def Target_Detection(image):
    image_height, image_width, _ = image.shape
    # Create a blob from an image
    blob = cv.dnn.blobFromImage(image=image, size=(300, 300), mean=(104, 117,
123), swapRB=True)
    model.setInput(blob)
    output = model.forward()
    # Iterate through each detection
    for detection in output[0, 0, :, :]:
        # Extract the confidence of the detection
        confidence = detection[2]
        # Draw bounding box only if detection confidence is above a certain
threshold, skip otherwise
        if confidence > .4:
            # Get class ID
            class_id = detection[1]
            # Map class ID to class
            class_name = class_names[int(class_id) - 1]
            color = COLORS[int(class_id)]
            # Get bounding box coordinates
            box_x = detection[3] * image_width
            box_y = detection[4] * image_height
```



```

# Get bounding box width and height
box_width = detection[5] * image_width
box_height = detection[6] * image_height
# Draw a rectangle around each detected object
cv.rectangle(image, (int(box_x), int(box_y)), (int(box_width),
int(box_height)), color, thickness=2)
# Write class name text on detected objects
cv.putText(image, class_name, (int(box_x), int(box_y - 5)),
cv.FONT_HERSHEY_SIMPLEX, 1, color, 2)
return image

```

5.4.3. Startup

```

cd /home/yahboom/YBAMR-COBOT-EDU-
00001/src/yahboom_navrobo_other/yahboom_navrobo_visual/detection
python target_detection.py

```

After clicking the image frame, use the keyboard [f] key to switch human posture estimation.

```

if action == ord('f') or action == ord('F'): state = not state # Function switch

```

