5. OpenCV application

- 5. OpenCV application
  - 5.1. Overview
  - 5.2. QR code
    - 5.2.1 Introduction of QR code
    - 5.2.2 The structure of QR code
    - 5.2.3. Features of QR code
    - 5.2.4. QR code creation and recognition
  - 5.3. Human Pose Estimation
    - 5.3.1. Overview
    - 5.3.2. Principle
    - 5.3.3. Start
  - 5.4, target detection
    - 5.4.1. Model structure
    - 5.4.2. code analysis
    - 5.4.3. Start

5.1. Overview

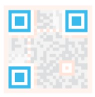
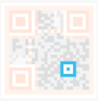
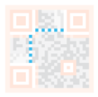
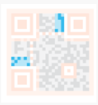
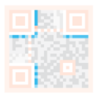
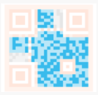
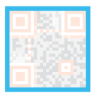
OpenCV is a cross-platform computer vision and machine learning software library released under the BSD license (open source) and can run on Linux, Windows, Android and MacOS operating systems. [1] It is lightweight and efficient - it consists of a series of C functions and a small number of C++ classes,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

5.2. QR code

5.2.1 Introduction of QR code

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

5.2.2 The structure of QR code

picture	Parse
	**Positioning** markings indicate the direction of the QR code.
	**Alignment** markings If the QR code is large, these additional elements help with positioning.
	**pattern** With these lines, the scanner can identify how big the matrix is.
	**Version information** (Version information) here specifies the version number of the QR code in use. There are currently 40 different version numbers of the QR code. Version numbers for the sales industry are usually 1-7.
	**Format** information Format patterns contain information about fault tolerance and data mask patterns and make scanning codes easier.
	**Data** and error correction keys These modes hold the actual data.
	**Quiet** zone This zone is very important for the scanner, its role is to separate itself from the surrounding.

5.2.3. Features of QR code

The data values in the QR code contain duplicate information (redundant values). Therefore, even up to 30% of the QR code structure is destroyed without affecting the readability of the QR code. The storage space of the QR code is up to 7089 bits or 4296 characters, including punctuation marks and special characters, 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 path: ~/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode

Install

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

- create

Create qrcode object

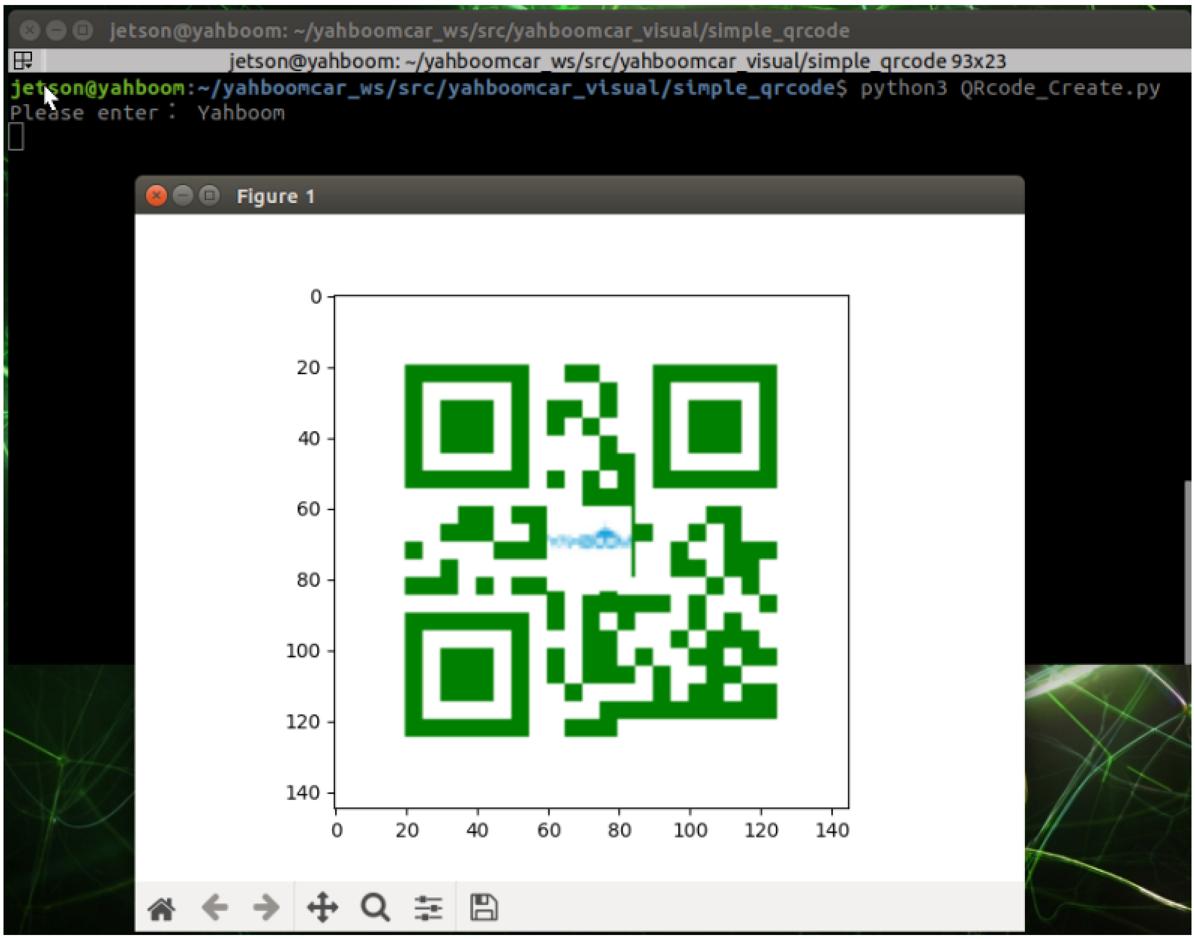
```
    '''
    参数含义:
    version: 值为1~40的整数, 控制二维码的大小 (最小值是1, 是个12×12的矩阵) 。
             如果想让程序自动确定, 将值设置为 None 并使用 fit 参数即可。
    error_correction: 控制二维码的错误纠正功能。可取值下列4个常量。
    ERROR_CORRECT_L: 大约7%或更少的错误能被纠正。
    ERROR_CORRECT_M (默认) : 大约15%或更少的错误能被纠正。
    ROR_CORRECT_H: 大约30%或更少的错误能被纠正。
    box_size: 控制二维码中每个小格子包含的像素数。
    border: 控制边框 (二维码与图片边界的距离) 包含的格子数 (默认为4, 是相关标准规定的最小值)
    '''
    qr = qrcode.QRCode( version=1,
error_correction=qrcode.constants.ERROR_CORRECT_H, box_size=5, border=4,)
```

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



```
jetson@yahboom:~/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode$ python3 QRcode_Create.py
Please enter： Yahboom
```

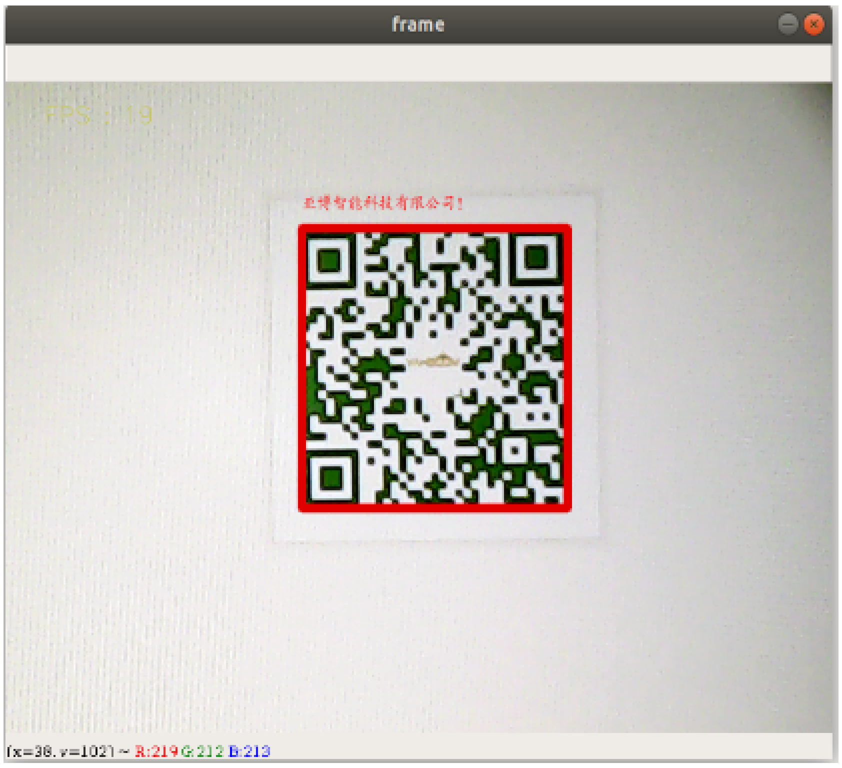
jetson@yahboom: ~/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode
jetson@yahboom: ~/yahboomcar_ws/src/yahboomcar_visual/simple_qrcode 93x23
Figure 1
0
20
40
60
80
100
120
140

- Identify

```
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 to a string first
        barcodeData  =  barcode . data . decode ( encoding )
        barcodeType  =  barcode . type
        # Plot the data and type on the image
        pilimg  =  Image . fromarray ( image )
        # create brush
        draw  =  ImageDraw . Draw ( pilimg )
        # 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 )
        # PIL image to cv2 image
        image  =  cv . cvtColor ( np . array ( pilimg ),  cv . COLOR_RGB2BGR )
        # Print barcode data and barcode type to terminal
        print ( "[INFO] Found {} barcode: {}" . format ( barcodeType ,
 barcodeData ))
    return  image
```

- Effect demonstration

```
Just use python3 + py file to execute it
```

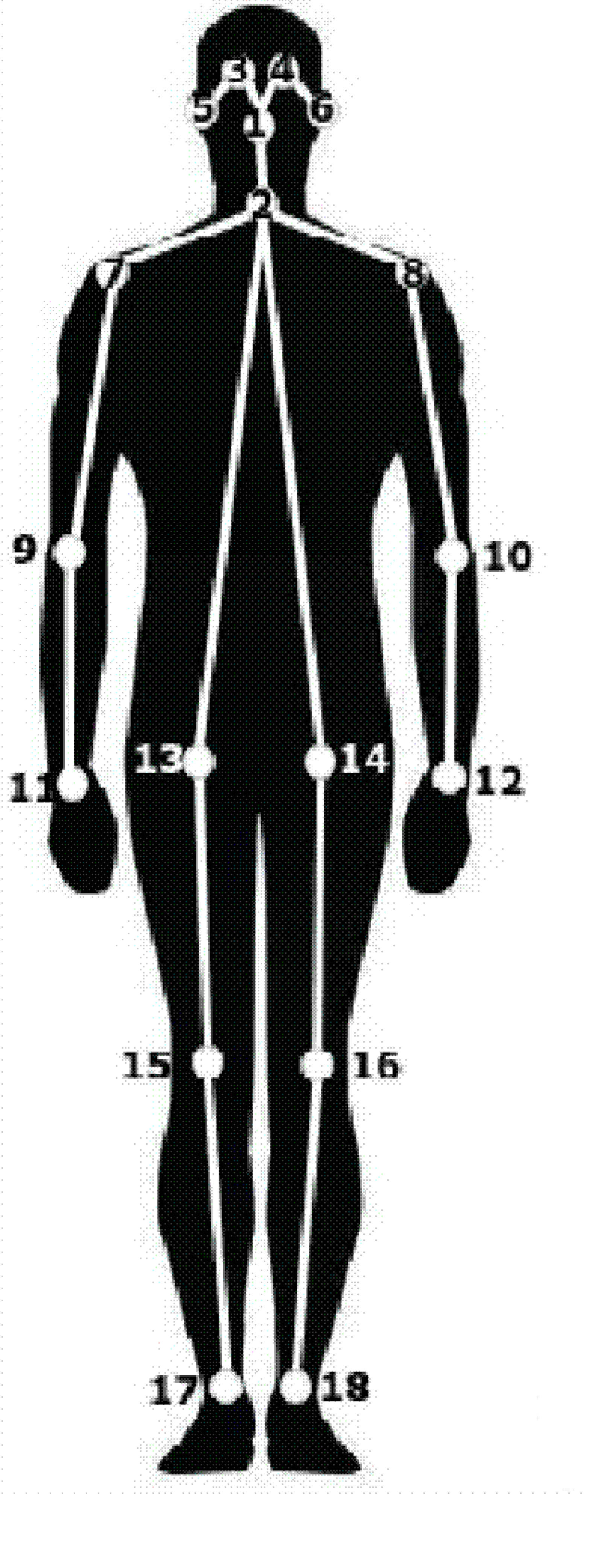


5.3. Human Pose Estimation

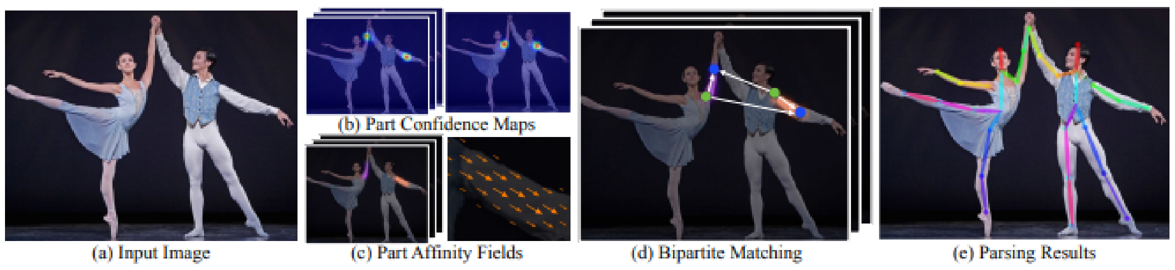
Source path: ~/yahboomcar_ws/src/yahboomcar_visual/detection

5.3.1. Overview

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


3
4
5
6
1
2
7
8
9
10
13
14
11
12
15
16
17
18

5.3.2. Principle



Input an image, extract features through a convolutional network, and obtain a set of feature maps, which are then divided into two forks, and use the CNN network to extract Part Confidence Maps and Part Affinity Fields respectively; after obtaining these two information, we use the graph theory in Bipartite Matching (even matching) Find the Part Association, connect the joint points of the same person, due to the vector nature of the PAF itself, the resulting bipartite matching is very correct, and finally merged into the overall skeleton of a person; Finally, based on PAFs, Multi- Person Parsing—>Convert the Multi-person parsing problem into a graphs problem—>Hungarian Algorithm (Hungarian Algorithm) (The Hungarian algorithm is the most common algorithm for partial graph matching. The core of the algorithm is to find an augmentation path. An algorithm for finding the maximum matching of a bipartite graph with a wide path.)

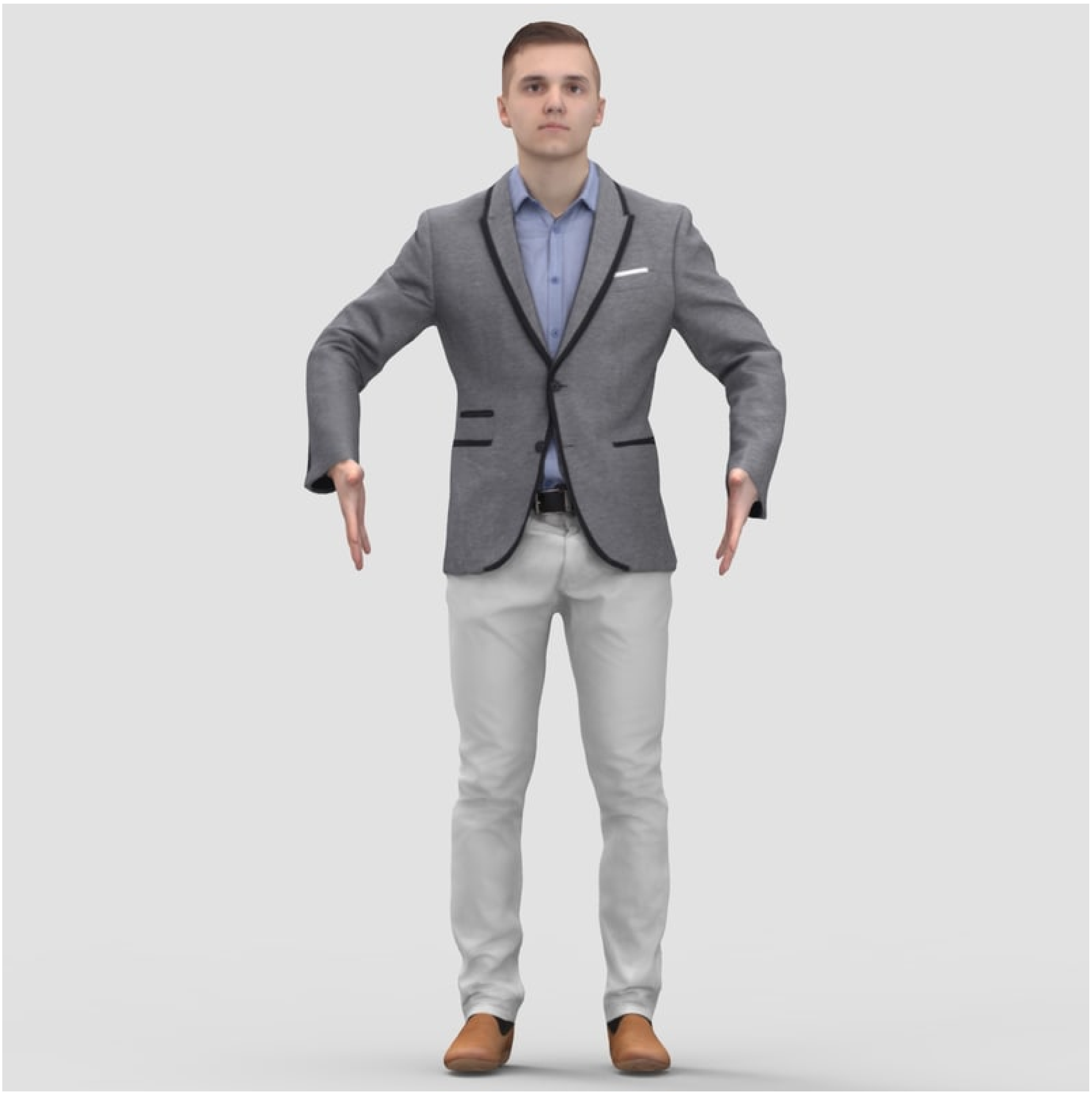
5.3.3. Start

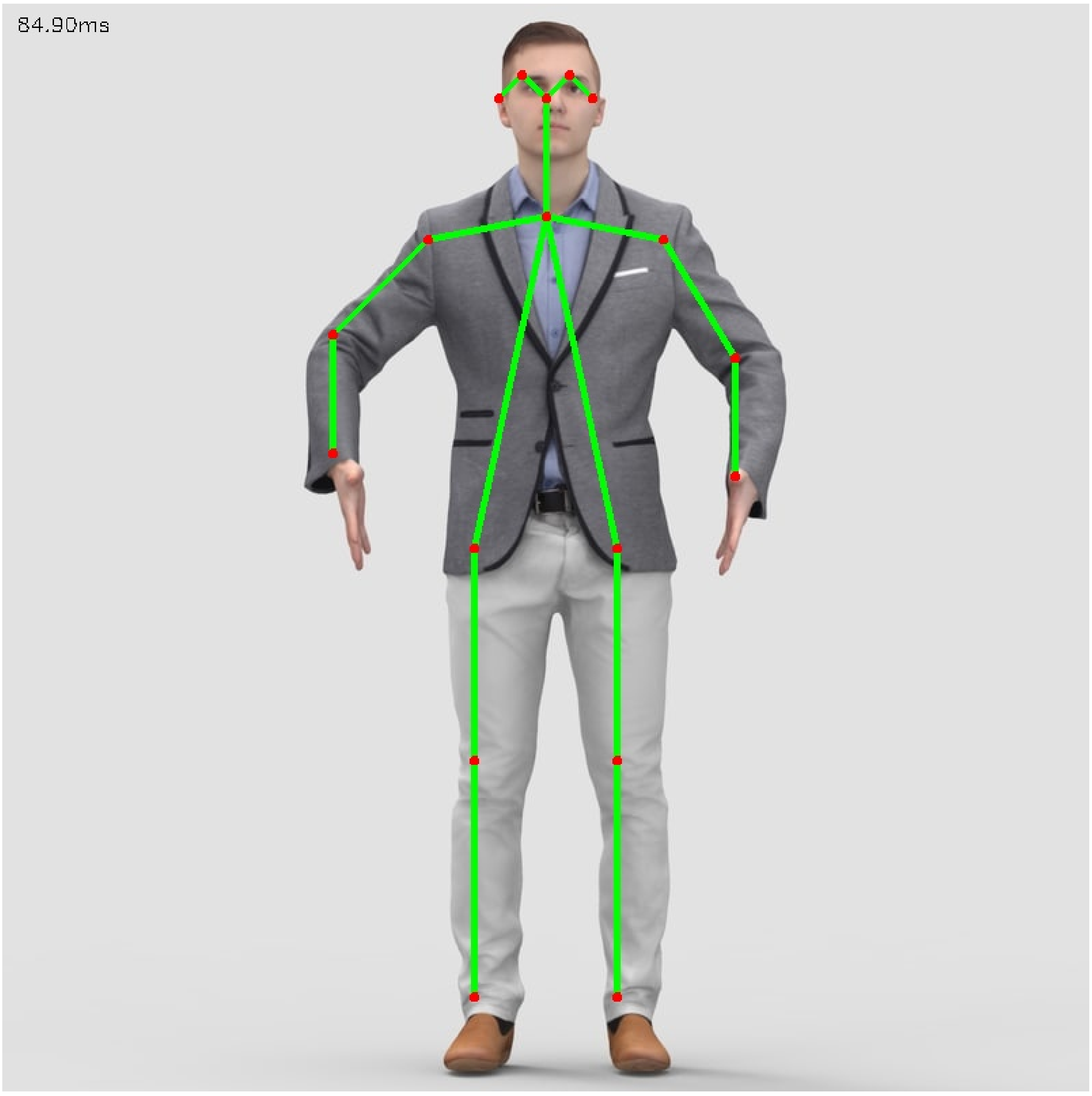
```
cd ~/yahboomcar_ws/src/yahboomcar_visual/detection
python target_detection.py
```

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

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

input image


output image



5.4, target detection

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

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

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

Faster R-CNNs are the most commonly heard of deep learning based neural networks. However, this approach is technically difficult to understand (especially for deep learning newbies), difficult to implement, and difficult to train.

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

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

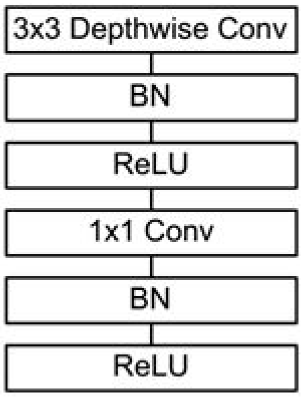
SSDs were originally developed by Google and can be said to be a balance between the above two. Compared to Faster R-CNNs, its algorithm is more straightforward. Compared with YOLO, it is more accurate.

5.4.1. Model structure

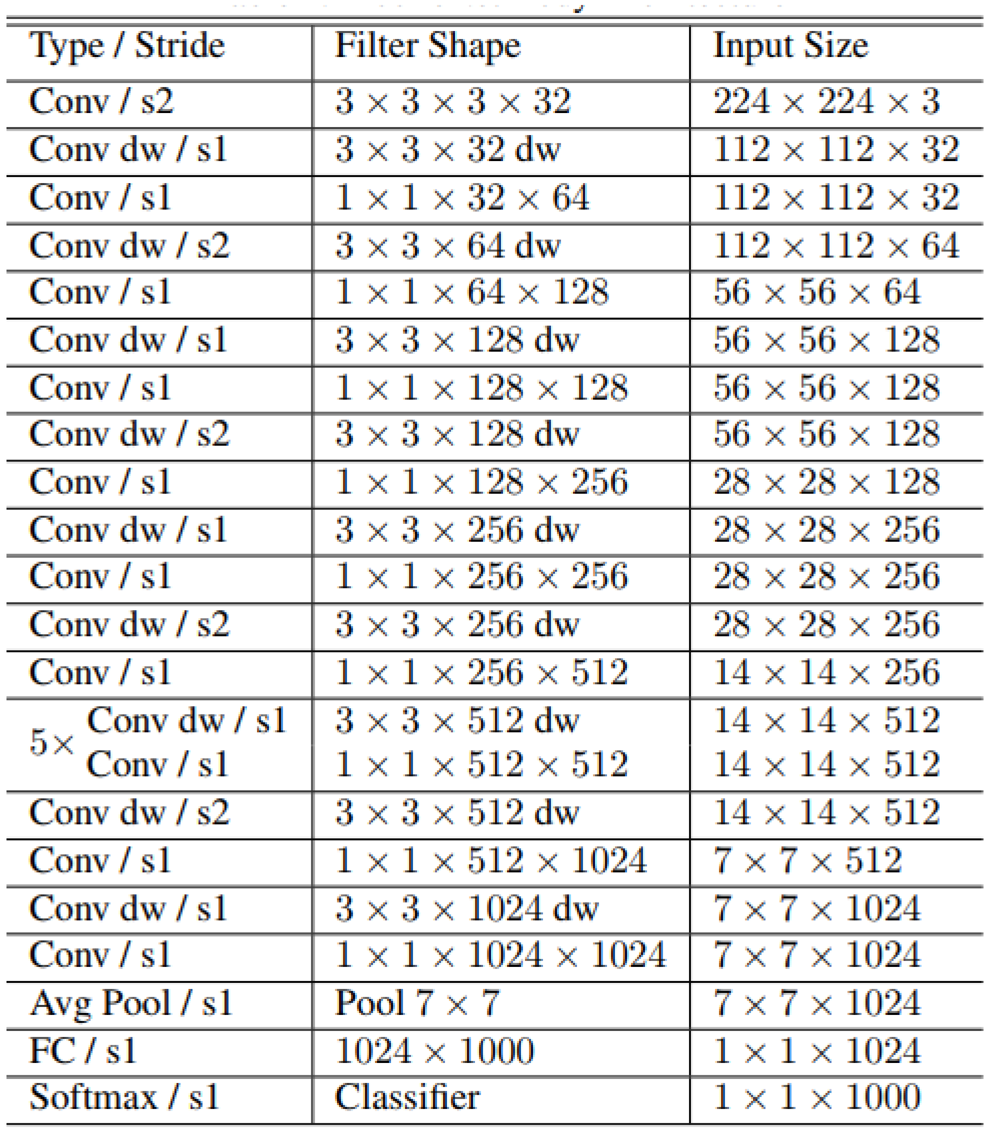
The main work of MobileNet is to replace the past standard convolutions (standard convolutions) with depthwise sparable convolutions (depth-level separable convolutions) to solve the problems of computational efficiency and parameter size of convolutional networks. The MobileNets model is based on depthwise sparable convolutions (depth-level separable convolutions), which can decompose standard convolutions into a depthwise convolution and a point convolution (1 × 1 convolution kernel). **Depthwise convolution applies each 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 mean of the result (each dimension of the output signal) is 0 and the variance is 1. Generally, you can try BN to solve the problem that the convergence speed is very slow or the gradient explosion cannot be trained during the training of the neural network. In addition, in general use cases, BN can also be added 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 following figure:



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



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	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ 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 recognized 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]

```
with open('object_detection_coco.txt', 'r') as f: class_names =
f.read().split('\n')
COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))
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

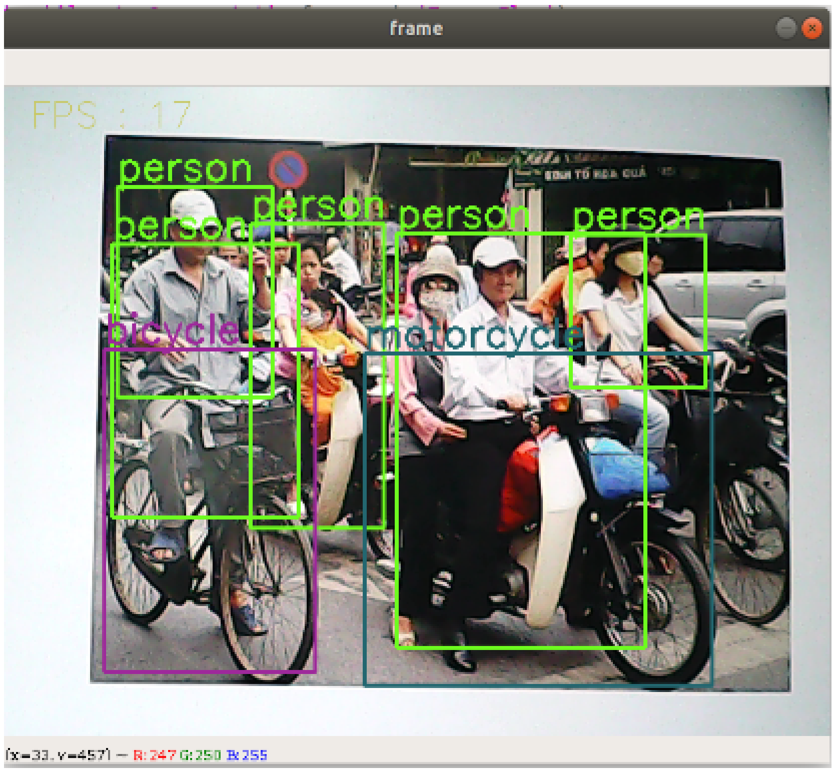
```
def  Target_Detection ( image ):
    image_height ,  image_width ,  _  =  image . shape
    # create blob from image
    blob  =  cv . dnn . blobFromImage ( image = image ,  size =( 300 ,  300 ),
 mean =( 104 ,  117 ,  123 ),  swapRB = True )
    model . setInput ( blob )
    output  =  model . forward ()
    # iterate over 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 the 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 the width and height of the bounding box
            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 the class name text on the detected object
            cv . putText ( image ,  class_name , ( int ( box_x ),  int ( box_y
 -  5 )),  cv . FONT_HERSHEY_SIMPLEX ,  1 ,  color ,  2 )
    return  image
```

5.4.3. Start

```
cd ~/yahboomcar_ws/src/yahboomcar_visual/detection
python target_detection.py
```

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

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


frame
FPS : 17
person
person
person
person
person
bicycle
motorcycle
(x=33, y=457) ~ R:247 G:250 B:255