6、Object detection and action

The Raspberry Pi motherboard series does not currently support this tutorial.

6.1、Introduction to gameplay

Object detection and action gameplay are functions implemented by combining the detectNet module of jetson-inference and the preset action group of the Muto robot. detectNet is mainly responsible for detecting whether the object of the model is included in the image. If so, it uses a box to frame the object. , and at the same time, the name of the detected object is passed to the robot for judgment. Eight types of objects are selected in the program, corresponding to the eight preset actions of the Muto robot. If the corresponding object is detected in the image, the Muto robot will perform the corresponding action.

The object detection function is more suitable for practical applications than the image classification function. The reason is that in most of the pictures seen in daily life, there is not only one object, but usually multiple categories or multiple objects. Image classification is based on the image. , it can only tell whether an object is contained in the image, but cannot confirm the location and size of the object; object detection takes the object as the unit and can detect multiple objects in the image, as well as the location, size and other information of the object.

6.2、Core content analysis

Initialize the Muto robot and set the initial angle of the camera gimbal. The default is S1=90 and S2=60. The angle of the camera gimbal can be modified according to the actual viewing angle.

```
g_bot = Muto()
g_bot.Gimbal_1_2(90, 60)
```

Import the detectNet module from jetson-inference, and import the image input and output modules from jetson_utils.

```
from jetson_inference import detectNet
from jetson_utils import videoSource, videoOutput, Log
```

Configure input parameters：

The input source defaults to USB camera device /dev/video0, and the USB camera device number can be modified according to the device number found in the system.

The output source defaults to the graphical desktop.

The network network model defaults to ssd-mobilenet-v2.

Overlays default to lines, labels, and configurations.

The default detection threshold is 0.8, which can be modified according to the actual recognition effect. The recommended range is 0.5-1.0.

```
parser = argparse.ArgumentParser(description="Locate objects in a live camera stream using an object detection DNN.",
                                 formatter_class=argparse.RawTextHelpFormatter,
                                 epilog=detectNet.Usage() + videoSource.Usage() + videoOutput.Usage() + Log.Usage())

parser.add_argument("input", type=str, default="/dev/video0",
                    nargs='?', help="URI of the input stream")
parser.add_argument("output", type=str, default="",
                    nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="ssd-mobilenet-v2",
                    help="pre-trained model to load (see below for options)")
parser.add_argument("--overlay", type=str, default="line,labels,conf",
                    help="detection overlay flags (e.g. --overlay=box,labels,conf)\nvalid combinations are:  'line', 'box', 'labels', 'conf', 'none'")
parser.add_argument("--threshold", type=float, default=0.8,
                    help="minimum detection threshold to use")

is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else [""]

try:
    args = parser.parse_known_args()[0]
except:
    print("")
    parser.print_help()
    sys.exit(0)

print("THRESHOLD:", args.threshold)
# load the object detection network
net = detectNet(args.network, sys.argv, args.threshold)

# create video sources and outputs
input = videoSource(args.input, argv=sys.argv)
output = videoOutput(args.output, argv=sys.argv+is_headless)
```

Read the camera image and then transmit it to detectNet to calculate and output the detection result.

```
img = input.Capture()
detections = net.Detect(img, overlay=args.overlay)
```

Traverse the detection results, take out the ClassID of the detection object in the result, and pass it to the robot for detection. In order to reduce the risk of false triggering, the number of consecutive detections is increased to 3 before the object is considered detected.

```
index = 0
for detection in detections:
    index = index + 1
    # print(detection)
    # print("ClassID:", detection.ClassID)
    if last_id == detection.ClassID:
        detect_count = detect_count + 1
    else:
        if index == len(detections):
            detect_count = 0
            last_id = detection.ClassID

    if detect_count > 3:
        detect_count = 0
        # print("Detect ID:", detection.ClassID)
        control_robot(detection.ClassID)
```

The Muto robot performs different actions based on the ClassID of the detected object. Due to the limited actions of the robot, it can only select eight objects to perform actions. The traffic light object corresponds to the stretch action, the bicycle object corresponds to the greeting action, the car corresponds to the fearful shrinking action, the motorcycle corresponds to the warm-up squatting action, the airplane corresponds to the spinning action, the bus corresponds to the waving and saying no action, and the train corresponds to the curling up action. The boat corresponds to the forward movement.

```
def control_robot(class_id):
    global g_control_count
    if g_control_count > 0:
        return
    g_control_count = 20
    print("class_id:", class_id)
    if CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_traffic_light]:
        g_bot.action(1) # stretch
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_bicycle]:
        g_bot.action(2) # say hello
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_car]:
        g_bot.action(3) # fear of retreating
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_motorcycle]:
        g_bot.action(4) # warm up
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_airplane]:
        g_bot.action(5) # circle in place
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_bus]:
        g_bot.action(6) # say no
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_train]:
        g_bot.action(7) # curl up
    elif CLASS_TYPE.get(class_id) == CLASS_TYPE[ID_boat]:
        g_bot.action(8) # stride forward
    else:
        g_control_count = 0
```

detectNet uses the ssd-mobilenet-v2 network model to detect 90 types of objects. For detailed categories, please refer to the ssd_coco_labels.txt file in the ~/jetson-inference/data/networks directory. For simplicity and convenience, only the names of the first 8 objects are listed here.

```
ID_persion = 1
ID_bicycle = 2
ID_car = 3
ID_motorcycle = 4
ID_airplane = 5
ID_bus = 6
ID_train = 7
ID_truck = 8
ID_boat = 9
ID_traffic_light = 10
```

```
CLASS_TYPE = {
    ID_persion : "person",
    ID_bicycle : "bicycle",
    ID_car : "car",
    ID_motorcycle : "motorcycle",
    ID_airplane : "airplane",
    ID_bus : "bus",
    ID_train : "train",
    ID_truck : "truck",
    ID_boat : "boat",
    ID_traffic_light : "traffic light"
}
```

6.3、Program execution and operation

Note that the following commands need to be run in the system desktop terminal. You can connect the display or use VNC to remotely log in to the desktop to operate.

Open the terminal on the system desktop and run the following command：

```
python3 ~/muto/Samples/Deep_Learning/detectnet_action.py
```

At this time, the program starts running, and a camera screen pops up on the desktop. If the object or object image corresponding to the above actions is placed within the camera collection range, the Muto robot will frame the object and perform corresponding actions.

If you need to exit the program, press Ctrl+C in the terminal interface to exit the program.