5.5 Object follow

1. Introduction to COCO dataset pre-training model

What is COCO?

COCO is a large-scale object detection, segmentation and caption data set. COCO has several characteristics:

- Object segmentation
- Recognition in the background
- Super pixel division
- 330K image (> 200K label)
- 1.5 million object instances
- 80 object categories
- 91 things category
- 5 subtitles per picture
- 250,000 people have key points

To achieve object tracking, we used a pre-trained neural network trained on the COCO dataset (http://cocodataset.org, which requires scientific access to the web) to detect 90 different common objects.

For more details, please refer to: **[Jetbot-AI Car] --> [Annex] --> [COCO_data.txt]**

2. Implementation of motion algorithm

Shared processing algorithm:

Detection_center(detection)

Calculating the center x, y coordinate algorithm of the object, code as shown below.

```
def detection_center(detection):
    """Calculate the center x, y coordinates of the object"""
    bbox = detection['bbox']
    center_x = (bbox[0] + bbox[2]) / 2.0 - 0.5
    center_y = (bbox[1] + bbox[3]) / 2.0 - 0.5
    return (center_x, center_y)
```

Norm(vec)

Calculate the length algorithm of a two-dimensional vector, code as shown below.

```
def norm(vec):
    """Calculate the length of a two-dimensional vector"""
    return np.sqrt(vec[0]**2 + vec[1]**2)
```

Closest_detection(detections)

Find the detection algorithm closest to the center of the image, as shown below.

```
def closest_detection(detections):
    """Find the detection closest to the center of the image"""
    closest_detection = None
    for det in detections:
        center = detection_center(det)
        if closest_detection is None:
            closest_detection = det
        elif norm(detection_center(det)) < norm(detection_center(closest_detection)):
            closest_detection = det
    return closest_detection
```

①Advanced Optimization Edition Add Algorithm:
Because the basic version of the follow-up movement does not seem to be very satisfying, and then not very smooth when following, we add a new algorithm to control the following process.

②Follow the speed PID Adjustment Algorithm: This will make the Jetbot faster when we are far away from the Jetbot. When the distance is closer, the Jetbot will be slower until it reaches a certain distance from the target.

③Steering Gain PID Adjustment Algorithm: When the Jetbot deviates significantly from the direction of travel of the corresponding tracking target, the steering gain will be larger to speed up the direction calibration. When it is smaller, the steering gain will not be too large, making the motion look smoother.

```
#Follow speed PID adjustment
follow_speed_pid.SystemOutput = 90000 * (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
if label_widget.value == 1:
    follow_speed_pid.SetStepSignal(30000)
elif label_widget.value == 53 or label_widget.value == 55: # 53:Apple 55:orange
    follow_speed_pid.SetStepSignal(10000)
else:
    follow_speed_pid.SetStepSignal(10000)
follow_speed_pid.SetInertiaTime(0.2, 0.1)
```

④Camera vertical angle PID Adjustment Algorithm: When tracking non-human small object objects, the object may be lost in the vertical direction, so add a vertical camera angle here, let Jetbot automatically capture the field of view to the target object.

```
if label_widget.value != 1:
    #Vertical angle camera PID adjustment
    object_yservo_pid.SystemOutput =bbox[1]*300+150*(bbox[3] - bbox[1])
    # print('Y-axis object center position value:')
    # print(object_yservo_pid.SystemOutput)
    object_yservo_pid.SetStepSignal(150)
    object_yservo_pid.SetInertiaTime(0.2, 0.1)
    #Limit the vertical angle camera angle to the specified range
    target_value_object_yservo = int(2048 + object_yservo_pid.SystemOutput)
    servo_device.Servo_serial_control(2, target_value_object_yservo)
```

3. Object following

We need to import the ``ObjectDetector`` class, which uses our pre-trained

SSD engine. There are many other classes (you can check [this file] (https://github.com/tensorflow/models/blob/master/research/object_detection/data/mscoco_complete_label_map.pbt)
Get a complete list of class indexes).
This model comes from [TensorFlow Object Detection API] (https://github.com/tensorflow/models/tree/master/research/object_detection) The API also provides utilities for object detector training for custom tasks. Once the model is trained, we use the NVIDIA TensorRT to optimize the Jetson Nano. This makes the network very fast and can control Jetbot in real time.

First, we need to import the "ObjectDetector" class, which uses our pre-trained SSD engine.

```
from jetbot import ObjectDetector

model = ObjectDetector('ssd_mobilenet_v2_coco.engine')
```


Internally, the 'ObjectDetector' class uses the TensorRT Python API to execute the engine we provide. It is also responsible for pre-processing the input of the neural network and parsing the detected objects.
Currently, it only works with engines created with the ``jetbotssd_tensorrt`` package. This package has utilities for converting models from the TensorFlow object detection API to the optimized TensorRT engine. Next, let's initialize the camera. Our detector needs 300x300 pixels of input, so we will set this parameter when creating the camera.
(Note: The resolution must be 300 * 300, otherwise the objects in the model will not be recognized.)

```
from jetbot import Camera

camera = Camera.instance(width=300, height=300)
```


Next, let's use some camera input to execute our network. By default, the ``ObjectDetector`` class expects the camera to generate a format of ``bgr8``. However, if the input format is different, you can override the default preprocessor function.
If there are any COCO objects in the camera's field of view, they should now be stored in the ``detections`` variable, and we print them out with the code shown below or with a text widget:

```
detections = model(camera.value)

print(detections)
```



```
from IPython.display import display
import ipywidgets.widgets as widgets

detections_widget = widgets.Textarea()
detections_widget.value = str(detections)
display(detections_widget)
```

Print out the first object detected in the first image:

```
image_number = 0
object_number = 0

print(detections[image_number][object_number])
```

Control the robot to follow the center object

Now we want the robot to follow the object of the specified class. To do this, we need to do the following:

1. Detect objects that match the specified class
2. Select the object closest to the center of the camera's field of view. This is the target object.
3. Guide the robot to move to the target object, otherwise it will drift

We will also create widgets that control the target object label, robot speed and cornering gain, and control the speed of the robot's cornering based on the distance between the target object and the center of the robot's field of view.

```
import torch
import torchvision
import torch.nn.functional as F
import cv2
import numpy as np

mean = 255.0 * np.array([0.485, 0.456, 0.406])
stdev = 255.0 * np.array([0.229, 0.224, 0.225])

normalize = torchvision.transforms.Normalize(mean, stdev)

def preprocess(camera_value):
    global device, normalize
    x = camera_value
    #Image zoomed to 224,224 versus 224,244 obstacle avoidance model
    x = cv2.resize(x, (224, 224))
    x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    x = x.transpose((2, 0, 1))
    x = torch.from_numpy(x).float()
    x = normalize(x)
    x = x.to(device)
    x = x[None, ...]
    return x
```

Create a robot instance of the drive motor, code as shown below:

```
from jetbot import Robot

robot = Robot()
```

Finally, we need to display the widgets for all controls and connect the network execution function to the camera update. We set the tracked object by the value of label_widget.

For more details, please refer to: **[Jetbot-AI Car] --> [Annex] --> [COCO_data.txt]**

```
from jetbot import bgr8_to_jpeg

image_widget = widgets.Image(format='jpeg', width=300, height=300)
label_widget = widgets.IntText(value=1, description='tracked label')
speed_widget = widgets.FloatSlider(value=0.4, min=0.0, max=1.0, description='speed')
turn_gain_widget = widgets.FloatSlider(value=0.8, min=0.0, max=2.0, description='turn gain')

display(widgets.VBox([
    widgets.HBox([image_widget]),
    label_widget,
    speed_widget,
    turn_gain_widget
]))

width = int(image_widget.width)
height = int(image_widget.height)

def detection_center(detection):
    """Calculate the center x, y coordinates of the object"""
    bbox = detection['bbox']
    center_x = (bbox[0] + bbox[2]) / 2.0 - 0.5
    center_y = (bbox[1] + bbox[3]) / 2.0 - 0.5
    return (center_x, center_y)

def norm(vec):
    """Calculate the length of a two-dimensional vector"""
    return np.sqrt(vec[0]**2 + vec[1]**2)

def closest_detection(detections):
    """Find the detection closest to the center of the image"""
    closest_detection = None
    for det in detections:
        center = detection_center(det)
        if closest_detection is None:
            closest_detection = det
        elif norm(detection_center(det)) < norm(detection_center(closest_detection)):
            closest_detection = det
    return closest_detection
```

```
def execute(change):
    image = change['new']
    # Calculate all detected objects
    detections = model(image)

    # Draw all tests on the image
    for det in detections[0]:
        bbox = det['bbox']
        cv2.rectangle(image, (int(width * bbox[0]), int(height * bbox[1])), (int(width * bbox[2]), int(height * bbox[3])), (255, 0, 0), 2)

    # Select to match the detection of the selected class label
    matching_detections = [d for d in detections[0] if d['label'] == int(label_widget.value)]

    # Let the detection be closest to the center of the field and draw it
    det = closest_detection(matching_detections)
    if det is not None:
        bbox = det['bbox']
        cv2.rectangle(image, (int(width * bbox[0]), int(height * bbox[1])), (int(width * bbox[2]), int(height * bbox[3])), (0, 255, 0), 5)

    # If don't detected object,jetbot car will keep davance
    if det is None:
        pass
        robot.forward(float(speed_widget.value))

    # control Jetbot follow object
    else:
        # Move the robot forward and control the x distance between the proportional target and the center
        center = detection_center(det)
        robot.set_motors(
            float(speed_widget.value + turn_gain_widget.value * center[0]),
            float(speed_widget.value - turn_gain_widget.value * center[0])
        )

    # Update image display to widget
    image_widget.value = bgr8_to_jpeg(image)

execute({'new': camera.value})
```

Call the following code block to connect the execution function to each camera frame to update:

```
camera.unobserve_all()
camera.observe(execute, names='value')
```

If the robot is not blocked, you can see that the blue box surrounds the detected object and the target object (the object that the robot follows) will be displayed in green.
When the target is discovered, the robot should turn to the target.
You can call the following code block to manually disconnect the camera and stop the robot.

```
import time
camera.unobserve_all()
time.sleep(1.0)
robot.stop()
```

The corresponding complete source code is located:
/home/jetbot/Notebook/14.Object follow-Basic/Object follow-Basic.ipynb

4. Object follow with Automatic avoid

We use the 【5.4 Automatic avoid】 training obstacle avoidance model and the object follow-up of this course, because there is no conflict in loading them at the same time, the processing speed has no effect, so we can put these two functions Used in combination, the difference from the basic version of the code is:

Load into the obstacle avoidance model:

```
collision_model = torchvision.models.alexnet(pretrained=False)
collision_model.classifier[6] = torch.nn.Linear(collision_model.classifier[6].in_features, 2)
collision_model.load_state_dict(torch.load('../collision_avoidance/best_model.pth'))
device = torch.device('cuda')
collision_model = collision_model.to(device)
```

Execute the conflict model before executing the following code to determine whether to block. If it is blocked, turn left, then return directly skips the execution of the following following code to start the next loop.

```
# Execute a conflict model to determine if it is blocking
collision_output = collision_model(preprocess(image)).detach().cpu()
prob_blocked = float(F.softmax(collision_output.flatten(), dim=0)[0])
blocked_widget.value = prob_blocked

# If blocked turn left
if prob_blocked > 0.5:
    robot.left(0.3)
    image_widget.value = bgr8_to_jpeg(image)
    return
```

The corresponding complete source code is located:
/home/jetbot/Notebook/15.Object follow-Avoid/Object follow-Avoid.ipynb

5. Optimized object following

Advanced optimized object tracking, we have added a new program:
Following speed PID Adjustment Algorithm
Steering gain PID Adjustment Algorithm
Camera vertical angle PID Adjustment Algorithm

The difference between it and the basic version is:
Add the import PID driver module, create a PID controller instance, and initialize the corresponding control variable cell code:

```
import PID

global follow_speed
follow_speed = 0.5
global turn_gain
turn_gain = 1.7
global follow_speed_pid, follow_speed_pid_model
follow_speed_pid_model = 1
# follow_speed_pid = PID.PositionalPID(3, 0, 0)
follow_speed_pid = PID.PositionalPID(1.5, 0, 0)
global turn_gain_pid
turn_gain_pid = PID.PositionalPID(1, 0, 0)
global object_yservo_pid
object_yservo_pid = PID.PositionalPID(3, 0.5, 0)
```


If the set tracking object is detected, the following three PID adjustment algorithms are added. Because when people follow people, the person is big enough for the Jetbot object, so there is no adjustment in the vertical angle direction of the gimbal. Others, we recommend adjusting the camera angle to the elevation angle when following the human body, expanding the scope of the Jetbot's field of view, following The effect will be better.

```
if det is None:
    # robot.forward(float(follow_speed))
    robot.stop()
# otherwsie steer towards target
else:
    # move robot forward and steer proportional target's x-distance from center
    center = detection_center(det)

    #Follow speed PID adjustment
    follow_speed_pid.SystemOutput = 90000 * (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
    if label_widget.value == 1:
        follow_speed_pid.SetStepSignal(30000)
    elif label_widget.value == 53 or label_widget.value == 55: # 53:Apple 55:orange
        follow_speed_pid.SetStepSignal(10000)
    else:
        follow_speed_pid.SetStepSignal(10000)
    follow_speed_pid.SetInertiaTime(0.2, 0.1)

    #Steering gain PID adjustment
    turn_gain_pid.SystemOutput = center[0]
    turn_gain_pid.SetStepSignal(0)
    turn_gain_pid.SetInertiaTime(0.2, 0.1)

    #Limit steering gain to the effective range
    if label_widget.value == 1:
        target_value_turn_gain = 0.6 + abs(turn_gain_pid.SystemOutput)
    elif label_widget.value == 53 or label_widget.value == 55:
        target_value_turn_gain = 0.5 + abs(turn_gain_pid.SystemOutput)
    else:
        target_value_turn_gain = 0.5 + abs(turn_gain_pid.SystemOutput)
    if target_value_turn_gain < 0:
        target_value_turn_gain = 0
    elif target_value_turn_gain > 2:
        target_value_turn_gain = 2
```



```
        #Keep the output motor speed within the effective driving range
        target_value_speed = 0.46 + follow_speed_pid.SystemOutput / 90000
        target_value_speedl = target_value_speed + target_value_turn_gain * center[0]
        target_value_speedr = target_value_speed - target_value_turn_gain * center[0]
        if target_value_speedl<0.5:
            target_value_speedl=0
        elif target_value_speedl>1:
            target_value_speedl = 1
        if target_value_speedr<0.5:
            target_value_speedr=0
        elif target_value_speedr>1:
            target_value_speedr = 1

        robot.set_motors(target_value_speedl, target_value_speedr)

        if label_widget.value != 1:
            #Vertical angle camera PID adjustment
            object_yservo_pid.SystemOutput =bbox[1]*300+150*(bbox[3] - bbox[1])
            # print('Y-axis object center position value:')
            # print(object_yservo_pid.SystemOutput)
            object_yservo_pid.SetStepSignal(150)
            object_yservo_pid.SetInertiaTime(0.2, 0.1)
            #Limit the vertical angle camera angle to the specified range
            target_value_object_yservo = int(2048 + object_yservo_pid.SystemOutput)
            servo_device.Servo_serial_control(2, target_value_object_yservo)
    # Update image display to widget
    image_widget.value = bgr8_to_jpeg(image)

execute({'new': camera.value})
```


The corresponding complete source code is located: /home/jetbot/Notebook/16.Object follow-Optimized/Object follow-Optimized.ipynb