6.5 Object follow

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP function, execute the following command:

```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

In Section 6.3.2, we have mastered the color recognition, and then we will implement the object following of the car through color recognition.

6.5.1 Implementation of motion algorithm

1. Calculate the center coordinates of the object:

```
(color_x,color_y),color_radius=cv2.minEnclosingCircle(cnt)
center_x = (150 - color_x)/150
```

2. The advanced optimization version adds algorithms:

Because the follow motion based on the basic version does not seem to be very satisfying, and it is not very smooth when following, we added a new algorithm to control the following process:

- **Following speed PID adjustment algorithm**: This will make Jetbotmini faster when we are far away from Jetbotmini, and slower when we are farther away, until it stops at a certain distance from the target , Does this sound smarter?
- **Steering gain PID adjustment algorithm**: When Jetbotmini deviates greatly from the corresponding tracking target, the greater the steering gain will speed up the direction calibration. When the steering gain is smaller, the steering gain will not be too large to make the movement look smoother.

```
#Follow speed PID regulation
follow_speed_pid.SystemOutput = 90000 * (color_radius/200)
follow_speed_pid.SetStepSignal(10000)
follow_speed_pid.SetInertiaTime(0.2, 0.1)

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

6.5.2 Object Follow-Basic Edition

First of all, we still need to import the relevant package, then create the camera instance that needs to be used, and create the color recognition array. Here we use red as the color to follow. If you need to modify it, you can use the code in section 6.3.2 as an example to modify.

Import related packages and create camera instances

```
from jetbotmini import Camera
from jetbotmini import bgr8_to_jpeg
camera = Camera.instance(width=300, height=300)
```

Create an array that stores HSV gamut color classification data

```
import numpy as np
global color_lower
color_lower=np.array([156,43,46])
global color_upperv
color_upper = np.array([180, 255, 255])
```

Set to recognize red array data

```
color_lower=np.array([0,43,46])
color_upper = np.array([10, 255, 255])
```

Then we create widgets to control the robot speed and turning gain, control the turning speed of the robot according to the distance between the target object and the center of the robot's field of view, and create a robot instance that drives the motor.

```
import torch
import torchvision
import torch.nn.functional as F
import cv2
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
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)
```

Create a robot instance that drives the motor.

```
from jetbotmini import Robot

robot = Robot()
```

Then, let us display all the control widgets, and connect the network execution function to the camera update, get the position and size of the recognized color through cv2.minEnclosingCircle(cnt), and convert it to the motor speed and send it to Jetbotmini to realize the color follow.

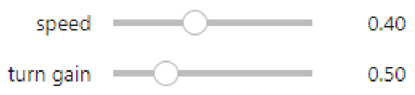
```
from jetbotmini import bgr8_to_jpeg

image_widget = widgets.Image(format='jpeg', width=300, height=300)
speed_widget = widgets.FloatSlider(value=0.4, min=0.0, max=1.0, description='speed')
turn_gain_widget = widgets.FloatSlider(value=0.5, min=0.0, max=2.0, description='turn gain')
center_x = 0
display(widgets.VBox([
    widgets.HBox([image_widget]),
    speed_widget,
    turn_gain_widget
]))
width = int(image_widget.width)
height = int(image_widget.height)
def execute(change):
    frame = camera.value
    frame = cv2.resize(frame, (300, 300))
    frame_=cv2.GaussianBlur(frame,(5,5),0)
    hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    mask=cv2.inRange(hsv,color_lower,color_upper)
    mask=cv2.erode(mask,None,iterations=2)
    mask=cv2.dilate(mask,None,iterations=2)
    mask=cv2.GaussianBlur(mask,(3,3),0)
    cnts=cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
    # Control jetbotmini to follow the set object
    if len(cnts)>0:
        cnt = max (cnts,key=cv2.contourArea)
        (color_x,color_y),color_radius=cv2.minEnclosingCircle(cnt)
        if color_radius > 10:
            # Mark the detected color
            cv2.circle(frame,(int(color_x),int(color_y)),int(color_radius),(255,0,255),2)
            center_x = (150 - color_x)/150
            robot.set_motors(
                float(speed_widget.value - turn_gain_widget.value * center_x),
                float(speed_widget.value + turn_gain_widget.value * center_x)
            )
    # If no target is detected, stop
    else:
        pass
        robot.stop()
    # Update image display to widget
    image_widget.value = bgr8_to_jpeg(frame)
execute({'new': camera.value})
```

Finally, the execution function is connected to each camera frame update, and Jetbotmini will make the corresponding action when it recognizes the red color.

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

We can adjust the speed of Jetbotmini through the slider.



```
speed      0.40
turn gain  0.50
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/13.Object_following-basic_version/Object_following-basic_version.ipynb

6.5.3 Object follow-advanced optimization version

In order to make Jetbotmini smarter and more natural in movement, we have added a new algorithm to the machine movement algorithm.

- Following speed PID adjustment algorithm
- Steering gain PID adjustment algorithm

The difference between it and the basic version is:

Increase the import of the PID driver module, create an instance of the PID controller, 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, 0.05)
global turn_gain_pid
turn_gain_pid = PID.PositionalPID(0.15, 0, 0.05)
```

Also key in the corresponding PID adjustment in the motion control to calculate the best speed value.

```python
# Target detected
if len(cnts)>0:
    cnt = max (cnts,key=cv2.contourArea)
    (color_x,color_y),color_radius=cv2.minEnclosingCircle(cnt)
    if color_radius > 10:
        # Mark the detected color
        cv2.circle(frame,(int(color_x),int(color_y)),int(color_radius),(255,0,255),2)
        # move robot forward and steer proportional target's x-distance from center
        center = (150 - color_x)/150

        #Follow speed PID regulation
        follow_speed_pid.SystemOutput = 90000 * (color_radius/200)
        follow_speed_pid.SetStepSignal(10000)
        follow_speed_pid.SetInertiaTime(0.2, 0.1)

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

        #Limit the steering gain to the valid range
        target_value_turn_gain = 0.2 + 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 valid driving range
        target_value_speed = 0.8 + follow_speed_pid.SystemOutput / 90000
        target_value_speedl = target_value_speed - target_value_turn_gain * center
        target_value_speedr = target_value_speed + target_value_turn_gain * center
        if target_value_speedl<0.3:
            target_value_speedl=0
        elif target_value_speedl>1:
            target_value_speedl = 1
        if target_value_speedr<0.3:
            target_value_speedr=0
        elif target_value_speedr>1:
            target_value_speedr = 1

        robot.set_motors(target_value_speedl, target_value_speedr)
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/14.Object_following-Advanced_Optimization/Object_following-Advanced_Optimization.ipynb