6.6 Autopilot

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
```

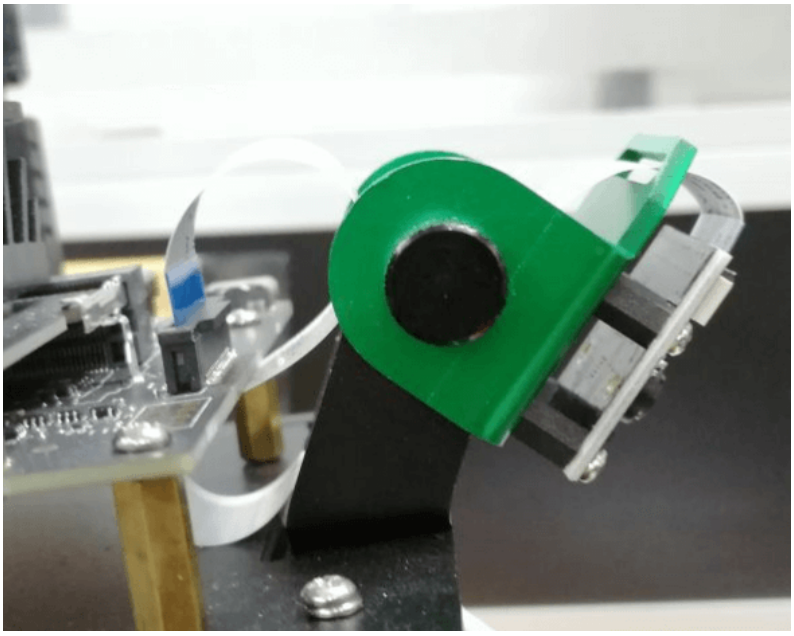
If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

```
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
```

To use the camera in this chapter, you need to manually twist the camera to face the line, otherwise it will not be able to drive automatically.



6.6.1 Autopilot-Basic Edition

In this section, we realize autonomous driving through color recognition.
As with the color follow, first we first import the relevant package, then create the camera instance that needs to be used, and create the color recognition array.

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([156,43,46])
color_upper = np.array([180, 255, 255])
```

Set to recognize yellow array data

```
color_lower=np.array([16,43,46])
color_upper = np.array([34, 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
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

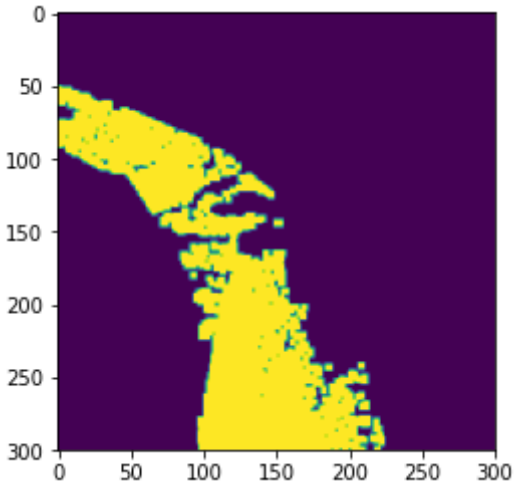
```
from jetbotmini import Robot

robot = Robot()
```

Here, the pictures processed by the camera are displayed in a loop of 10 pictures. We can adjust the corresponding value through this recognition result. The adjustment method can refer to section 6.3.2.

```
from  matplotlib import pyplot as plt
%matplotlib inline
from IPython import display

for i in range(10):
    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)
    plt.imshow(mask)
    plt.show()
    display.clear_output(wait=True)
```



Finally, let Jetbotmini make corresponding actions according to the processing results. The method here is the same as the color follow.

```
from jetbotmini import bgr8_to_jpeg
from IPython.display import display
image_widget = widgets.Image(format='jpeg', width=300, height=300)
speed_widget = widgets.FloatSlider(value=0.5, min=0.0, max=1.0, description='speed')
turn_gain_widget = widgets.FloatSlider(value=0.3, 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]
    # If so, 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})
camera.unobserve_all()
camera.observe(execute, names='value')
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/15.Autopilot-basic_version/Autopilot-basic_version.ipynb

6.6.2 Autonomous driving-advanced optimization version

Autopilot after advanced optimization adds PID adjustment algorithm to the motion 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 drive module, create PID controller instance and initialize corresponding control variables

```
import PID

global turn_gain
turn_gain = 1.7
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.

```
# 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

        #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 = 0.15 + 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_speedl = speed_widget.value - target_value_turn_gain * center
        target_value_speedr = speed_widget.value + 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/16.Autopilot-Advanced_Optimization/Autopilot-Advanced_Optimization.ipynb