3、Color following

3、Color following
3.1 Introduction to gameplay
3.2 Core content analysis
3.3 operation

3.1 Introduction to gameplay

This course mainly identifies colors and controls the function of the robot to follow the color movement. After turning on the start switch, the robot will adjust its position to keep the color object in the center of the screen as much as possible.

3.2 Core content analysis

Color_HSV defines the HSV value of a color. Since the color of an object has color differences and light effects, if the recognition of a certain color is inaccurate, you can adjust the course based on the color HSV value. After adjusting for the best effect, record the data and update the HSV value of the corresponding color.

```
Color_HSV = {
    'Red': ([0, 43, 46], [10, 255, 255]),
    'Green': ([35, 43, 46], [77, 255, 255]),
    'Blue': ([100, 43, 46], [124, 255, 255]),
    'Yellow': ([26, 43, 46], [34, 255, 255]),
    'Custom': ([0, 0, 0], [180, 255, 255]),
}
```

Follow red is selected by default.

```
color_lower = np.array(Color_HSV["Red"][0])
color_upper = np.array(Color_HSV["Red"][1])
```

The following color can be modified via the button. Each button press updates the following color HSV value.

```
def on_button_clicked(b):
    global color_lower, color_upper, g_stop_program
    ALL_Uncheck()
    b.icon = 'check'
    with output:
        print("Button clicked:", b.description)
    if b.description == Name_widgets['Close_Camera'][g_ENABLE_CHINESE]:
        #  Stop the thread and release the camera
        g_stop_program = True
        time.sleep(.1)
        g_camera.release()
```

```
        b.icon = 'uncheck'
        Button_Start.icon = 'uncheck'
    elif b.description == Name_widgets['Red'][g_ENABLE_CHINESE]:
        color_lower = np.array(np.array(Color_HSV["Red"][0]))
        color_upper = np.array(np.array(Color_HSV["Red"][1]))
    elif b.description == Name_widgets['Green'][g_ENABLE_CHINESE]:
        color_lower = np.array(np.array(Color_HSV["Green"][0]))
        color_upper = np.array(np.array(Color_HSV["Green"][1]))
    elif b.description == Name_widgets['Blue'][g_ENABLE_CHINESE]:
        color_lower = np.array(np.array(Color_HSV["Blue"][0]))
        color_upper = np.array(np.array(Color_HSV["Blue"][1]))
    elif b.description == Name_widgets['Yellow'][g_ENABLE_CHINESE]:
        color_lower = np.array(np.array(Color_HSV["Yellow"][0]))
        color_upper = np.array(np.array(Color_HSV["Yellow"][1]))
```

In order for the robot to better follow colored objects, the angle of the camera platform needs to be adjusted. The specific viewing angle can be adjusted according to the actual effect.

```
from MutoLib import Muto
g_bot = Muto()
g_bot.Gimbal_1_2(90, 30)
```

Relevant information is obtained based on the camera analysis, and the robot movement is controlled based on the recognized color xy coordinates and radius size.

```
def robot_control(x, y, radius):
    step = 15
    limit_radius = 100
    limit_xy = 100
    if radius < limit_radius*0.9:
        # 前进方向 forward
        if abs(x) < limit_xy or -limit_xy < y:
            g_bot.forward(step)
        elif x > 0 :
            g_bot.right(step)
        else:
            g_bot.left(step)
    elif radius < limit_radius*1.1:
        # 左右方向 left right
        if -limit_xy*0.15 < abs(x) < limit_xy*0.15:
            g_bot.stop()
        elif x > 0 :
            g_bot.right(step)
        else:
            g_bot.left(step)
    else:
        # 后退方向 back
        if abs(x) < limit_xy:
            g_bot.back(step)
        elif x > 0 :
            g_bot.right(step)
        else:
            g_bot.left(step)
```

Read camera images, analyze images to obtain relevant information, and control robot movement.

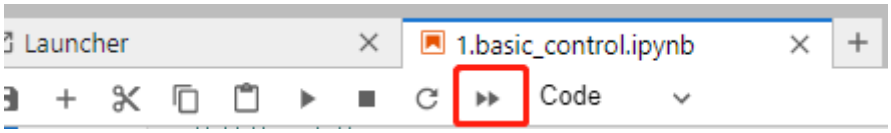
```
def task_processing():
    global color_lower, color_upper
    global g_stop_program, g_start_function
    t_start = time.time()
    m_fps = 0
    fps = 0
    while g_camera.isOpened():
        if g_stop_program:
            break
        ret, frame = g_camera.read()

        # 根据HSV值处理图像 The image is processed according to the HSV value
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, color_lower, color_upper)

        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
        if len(cnts) > 0:
            cnt = max (cnts, key = cv2.contourArea)
            (color_x, color_y), color_radius = cv2.minEnclosingCircle(cnt)
            if color_radius > 10:
                # 将检测到的颜色用圆形线圈标记出来 Mark the detected colors with
circular coils
                cv2.circle(frame, (int(color_x), int(color_y)), int(1),
(0,255,0), 2)
                cv2.circle(frame, (int(color_x), int(color_y)),
int(color_radius), (255,0,255), 2)
            if g_start_function:
                robot_control(int(color_x-img_width/2), int(color_y-
img_height/2), int(color_radius))
                # time.sleep(.01)
```

3.3 operation

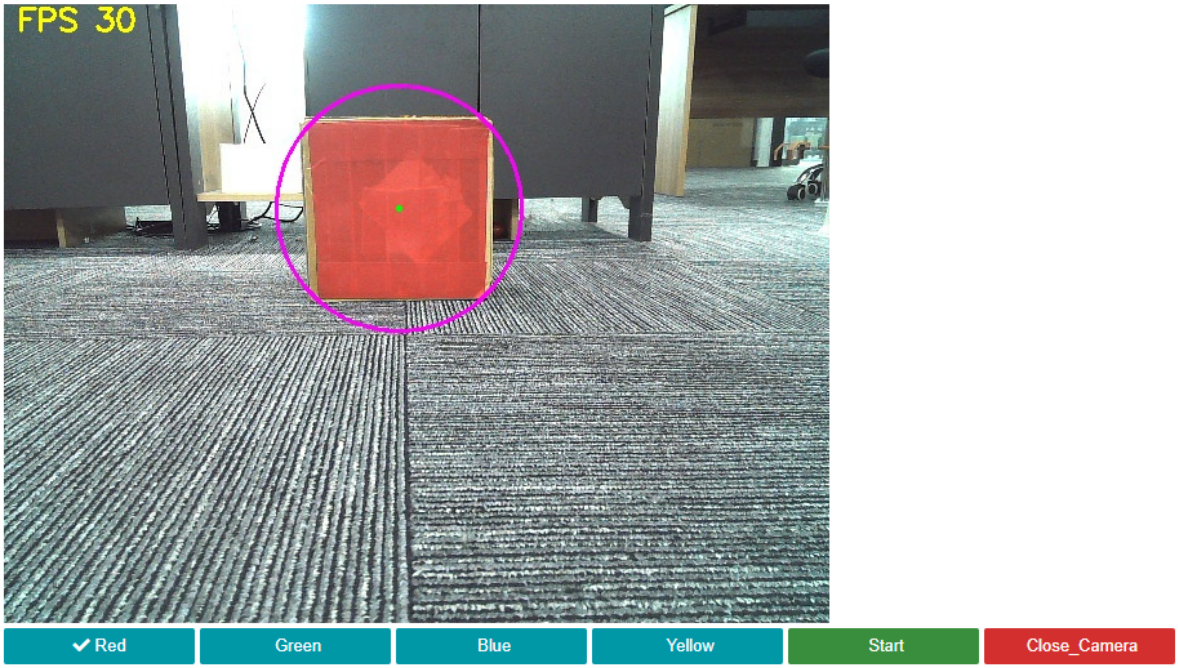
Open the jupyterLab client and find the code path:

```
muto/Samples/AI_Samples/03_color_following/color_following.ipynb
```

Click to run all cells, and then scroll to the bottom to see the generated controls.



The camera will track red objects. If you need to track other objects, click the button below to switch. Click the Start button and the robot will start to move. Keep the color object in the middle of the camera screen. If you move the color object, the robot will move accordingly. When the robot thinks that the color object is in the middle area of vision, it will automatically stop.



Finally click the Close_Camera button to close the camera.