2.Color recognition grab blocks

1.1 Experimental placement

The experimental placement is as follows: yellow building blocks are placed in the yellow area, red building blocks are placed in the red area, green building blocks are placed in the green area, and blue building blocks are placed in the blue area.

1.2 Code content

Code path：/home/dofbot/Dofbot/6.AI_Visual/3.color_grab.ipynb

The following code content needs to be executed according to the actual step. It cannot be run all at once. Running the last unit will directly exit the thread.

```
#bgr8 to jpeg format
import   enum
import   cv2

def   bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg',   value)[1])
```

```
#Import related modules
import   threading
import   time
from   Arm_Lib import Arm_Device

#   Create robot arm object
Arm   = Arm_Device()
time.sleep(.1)
```

```
#Camera component display
import   traitlets
import   ipywidgets.widgets as widgets
import   time

#   Thread function operation library
import   threading
import   inspect
import   ctypes

origin_widget   = widgets.Image(format='jpeg', width=320, height=240)
mask_widget   = widgets.Image(format='jpeg',width=320, height=240)
result_widget   = widgets.Image(format='jpeg',width=320, height=240)

#   Create a horizontal box container to place image widgets next to each other
image_container   = widgets.HBox([origin_widget, mask_widget, result_widget])
#   image_container = widgets.Image(format='jpeg', width=600, height=500)
display(image_container)
```

```
#Thread related functions
def   _async_raise(tid, exctype):
    """raises the exception,   performs cleanup if needed"""

    tid = ctypes.c_long(tid)
    if not inspect.isclass(exctype):
        exctype = type(exctype)
    res =   ctypes.pythonapi.PyThreadState_SetAsyncExc(tid,
ctypes.py_object(exctype))
    if res == 0:
        raise ValueError("invalid thread   id")
    elif res != 1:
        # If the value of res is greater than 1, exception handling is required
          ctypes.pythonapi.PyThreadState_SetAsyncExc(tid, None)

def   stop_thread(thread):
    _async_raise(thread.ident, SystemExit)
```

```
def   get_color(img):
    H = []
    color_name={}
    img = cv2.resize(img, (640, 480), )
    # Convert color image to HSV
    HSV = cv2.cvtColor(img,   cv2.COLOR_BGR2HSV)
    # Draw a rectangular frame
```

```
    cv2.rectangle(img, (280, 180), (360,   260), (0, 255, 0), 2)
    # Take out the H, S, and V values of each row and column in turn and put them
into the container.
    for i in range(280, 360):
        for j in range(180, 260):   H.append(HSV[j, i][0])
    # Calculate the maximum and minimum values of H, S, and V respectively.
    H_min = min(H);H_max = max(H)
#     print(H_min,H_max)
   # judge color

    if H_min >= 0 and H_max
    elif H_min >= 26 and H_max <= 34:   color_name['name'] = 'yellow'
    elif H_min >= 35 and H_max <= 78:   color_name['name'] = 'green'
    elif H_min >= 100 and H_max <= 124:   color_name['name'] = 'blue'
    return img, color_name
```

```
#   Define variable parameters at different locations
look_at   = [90, 164, 18, 0, 90, 90]
p_top   = [90, 80, 50, 50, 270]
p_Yellow   = [65, 22, 64, 56, 270]
p_Red   = [118, 19, 66, 56, 270]

p_Green   = [136, 66, 20, 29, 270]
p_Blue   = [44, 66, 20, 28, 270]
p_gray   = [90, 48, 35, 30, 270]
```

```
#  Define the mobile robot arm function to simultaneously control the movement of
servos No. 1-6, p=[S1, S2, S3, S4, S5, S6]

def   arm_move_6(p, s_time = 500):
    for i in range(6):
        id = i + 1
        Arm.Arm_serial_servo_write(id, p[i],   s_time)

        time.sleep(.01)

    time.sleep(s_time/1000)

#   Define the mobile robot arm function to simultaneously control the movement
of servos No. 1-5, p=[S1, S2, S3, S4, S5]
def   arm_move(p, s_time = 500):
    for i in range(5):
        id = i + 1
        if id == 5:
            time.sleep(.1)
            Arm.Arm_serial_servo_write(id,   p[i], int(s_time*1.2))
        elif id == 1 :
            Arm.Arm_serial_servo_write(id,   p[i], int(3*s_time/4))
        else:
            Arm.Arm_serial_servo_write(id,   p[i], int(s_time))
        time.sleep(.01)
    time.sleep(s_time/1000)
```

```
#   Define the function of clamping building blocks, enable=1: clamp, =0:
release
def   arm_clamp_block(enable):
    if enable == 0:
        Arm.Arm_serial_servo_write(6, 60,   400)
    else:
        Arm.Arm_serial_servo_write(6, 130,   400)
    time.sleep(.5)
```

```
arm_move_6(look_at,   1000)
time.sleep(1)
```

```
global   g_state_arm
g_state_arm   = 0

def   ctrl_arm_move(index):
    arm_clamp_block(0)

    if index == 1:
        print("yellow")
        number_action(index)
        put_down_block()

    elif index == 2:
        print("red")
        number_action(index)
        put_down_block()

    elif index == 3:
        print("green")
        number_action(index)
        put_down_block()

    elif index == 4:
        print("blue")
        number_action(index)
        put_down_block()

    global g_state_arm

    g_state_arm = 0
```

```
def   start_move_arm(index):

    # Open the robot arm control thread
    global g_state_arm
    if g_state_arm == 0:
        closeTid = threading.Thread(target =   ctrl_arm_move, args = [index])
        closeTid.setDaemon(True)
        closeTid.start()
        g_state_arm = 1
```

```
#   main process
```

```
import   cv2

import   numpy as np

import   ipywidgets.widgets as widgets

cap   = cv2.VideoCapture(0)

cap.set(3,   640)

cap.set(4,   480)

cap.set(5,   30)  #Set frame rate

cap.set(cv2.CAP_PROP_FOURCC,   cv2.VideoWriter.fourcc('M', 'J', 'P', 'G'))

#   image.set(cv2.CAP_PROP_BRIGHTNESS, 40) #Set brightness -64 - 64  0.0

#   image.set(cv2.CAP_PROP_CONTRAST, 50)     #Set contrast -64 - 64  2.0

#   image.set(cv2.CAP_PROP_EXPOSURE, 156)    #Set exposure value 1.0 - 5000
156.0

#   Red is selected by default, and the program will automatically switch colors
based on the color detected in the box.

#   red range

color_lower   = np.array([0, 43, 46])

color_upper   = np.array([10, 255, 255])

#   #green range

#   color_lower = np.array([35, 43, 46])

#   color_upper = np.array([77, 255, 255])

#   #blue range

#   color_lower=np.array([100, 43, 46])

#   color_upper = np.array([124, 255, 255])

#   #yellow range

#   color_lower = np.array([26, 43, 46])

#   color_upper = np.array([34, 255, 255])

#   #orange range

#   color_lower = np.array([11, 43, 46])

#   color_upper = np.array([25, 255, 255])
```

```
def   Color_Recongnize():
    Arm.Arm_Buzzer_On(1)
    s_time = 300
    Arm.Arm_serial_servo_write(4, 10, s_time)
    time.sleep(s_time/1000)
    Arm.Arm_serial_servo_write(4, 0, s_time)
    time.sleep(s_time/1000)
    Arm.Arm_serial_servo_write(4, 10, s_time)
    time.sleep(s_time/1000)
    Arm.Arm_serial_servo_write(4, 0, s_time)
    time.sleep(s_time/1000)
    while(1):
        # get a frame and show Obtain video frames and convert them into HSV format. Use cvtColor() to convert BGR format into HSV format. The parameter is cv2.COLOR_BGR2HSV.
        ret, frame = cap.read()
        frame, color_name = get_color(frame)
        if len(color_name)==1:
            global color_lower
            global color_upper
#             print ("color_name :",   color_name)
#             print ("name :",   color_name['name'])
            if color_name['name'] ==   'yellow':
                color_lower = np.array([26,   43, 46])
                color_upper = np.array([34,   255, 255])
                start_move_arm(1)
            elif color_name['name'] == 'red':
                color_lower = np.array([0,   43, 46])
```

```
                color_upper = np.array([10,   255, 255])

                start_move_arm(2)

            elif  color_name['name'] == 'green':

                color_lower = np.array([35,   43, 46])

                color_upper = np.array([77,   255, 255])

                start_move_arm(3)

            elif color_name['name'] ==   'blue':

                color_lower=np.array([100,   43, 46])

                color_upper = np.array([124,   255, 255])

                start_move_arm(4)

        origin_widget.value =   bgr8_to_jpeg(frame)

        #cv2.imshow('Capture', frame)

        # change to hsv model

        hsv = cv2.cvtColor(frame,   cv2.COLOR_BGR2HSV)

        # get mask Use the inRange() function and the upper and lower bounds of
the blue range in the HSV model to obtain the mask. The blue part of the original
video in the mask will be made white and the other parts black.
        mask = cv2.inRange(hsv, color_lower,   color_upper)
        #cv2.imshow('Mask', mask)
        mask_widget.value =   bgr8_to_jpeg(mask)

        # detect blue Perform a bitwise AND operation on the mask on the
original video frame, and the white in the mask will be replaced with the real
image:
        res = cv2.bitwise_and(frame, frame,   mask=mask)
        #cv2.imshow('Result', res)
        result_widget.value =   bgr8_to_jpeg(res)

        #       if cv2.waitKey(1) & 0xFF == ord('q'):
        #         break
        time.sleep(0.01)
    cap.release()
    #cv2.destroyAllWindows()
```

```
#Start process
thread1   = threading.Thread(target=Color_Recongnize)
thread1.setDaemon(True)
thread1.start()
```

```
#End the process. This code needs to be executed only when it ends.
stop_thread(thread1)
```

You can place the building blocks in front of the camera, and the camera will detect the color of the building blocks.Then put the building blocks into the area of the corresponding color, and the robot arm will pick up the building blocks in the corresponding area to the middle area based on the detected color.Or first put the building blocks into the corresponding color area, then find cards in four colors: yellow, red, green, and blue and put them in front of the camera.The robotic arm will grab the corresponding building blocks according to the detected color and place them in the middle area.

Note: Each time you grab a building block, you need to remove the building blocks in the middle area, otherwise it will block the next time you place the building blocks.