1. Experiment preparation

After running this program, the robotic arm will automatically reach the recognition position. Place block in the gray area on the map. After the robotic arm continues to correctly identify 10 times, it will automatically grab and place this block in the area of the corresponding color on map.

Path: dofbot_ws/src/dofbot_color_sorting/Catch_game.ipynb

2. About code

●Import header file

```
#!/usr/bin/env python
# coding: utf-8
import cv2 as cv
import threading
from time import sleep
from dofbot_config import *
import ipywidgets as widgets
from IPython.display import display
from color_sorting import color_sorting
```

●Create an instance, initialize parameters

```
import Arm_Lib
Arm = Arm_Lib.Arm_Device()
joints_0 = [90, 135, 0, 45, 90, 0]
Arm.Arm_serial_servo_write6_array(joints_0, 1000)
```

●Create an instance, initialize parameters

```
# Create an instance
sorting = color_sorting()
# Initialization mode
model = 'General'
# Color HSV threshold
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))}
# HSV parameter path
HSV_path="/home/jetson/dofbot_ws/src/dofbot_color_sorting/HSV_config.txt"
# Read HSV configuration file, update HSV value
try: read_HSV(HSV_path,color_hsv)
except Exception: print("Read HSV_config Error!!!")
```

●Create control

```
# Create control layout
button_layout    = widgets.Layout(width='200px', height='70px', align_self='center')
# Output print
output = widgets.Output()
# Exit button
exit_button = widgets.Button(description='Exit', button_style='danger', layout=button_layout)
# Image widgets
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
controls_box = widgets.VBox([imgbox, exit_button], layout=widgets.Layout(align_self='center'))
# ['auto', 'flex-start', 'flex-end', 'center', 'baseline', 'stretch', 'inherit', 'initial', 'unset']
```

●Control button

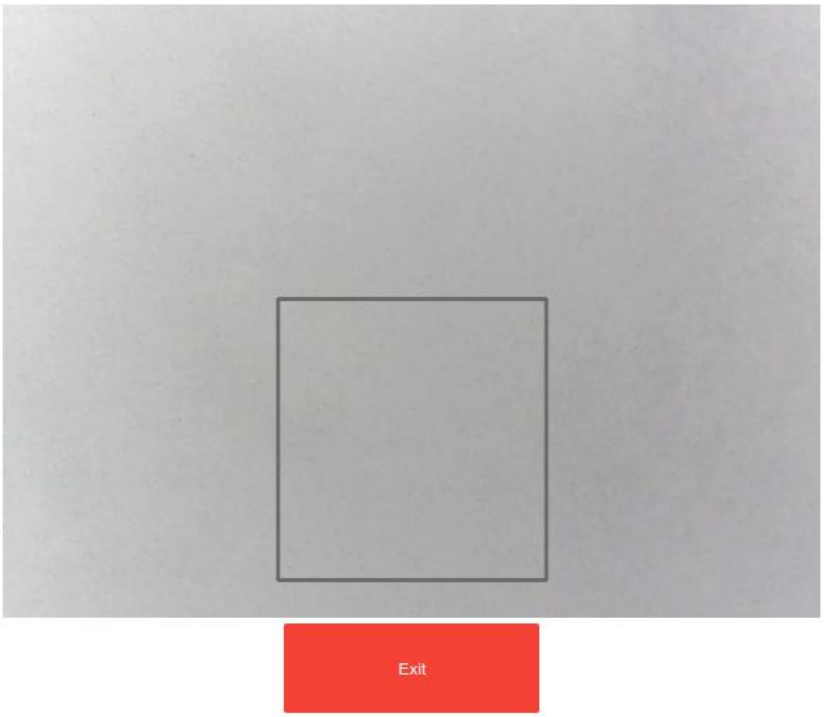
```
def exit_button_Callback(value):
    global model
    model = 'Exit'
#       with output: print(model)
exit_button.on_click(exit_button_Callback)
```

●Main process

```
def camera():
    # Open camera
    capture = cv.VideoCapture(0)
    while capture.isOpened():
        try:
            _, img = capture.read()
            img = cv.resize(img, (640, 480))
            img = sorting.Sorting_grap(img, color_hsv)
            if model == 'Exit':
                cv.destroyAllWindows()
                capture.release()
                break
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except KeyboardInterrupt:capture.release()
```

●Start

```
display(controls_box,output)
threading.Thread(target=camera, ).start()
```



2. About library code

●HSV filters out the target color

```
# Convert the image to HSV
HSV_img = cv.cvtColor(mask, cv.COLOR_BGR2HSV)
# Filter out the elements located between the two arrays
img = cv.inRange(HSV_img, lowerb, upperb)

mask[img == 0] = [0, 0, 0]
```

●Morphological transformation

```
kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))

dst_img = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
```

●Find contours in binary images

```
dst_img = cv.cvtColor(dst_img, cv.COLOR_RGB2GRAY)

ret, binary = cv.threshold(dst_img, 10, 255, cv.THRESH_BINARY)

contours, heriachy = cv.findContours(binary, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

●Calculate outline border

```
for i, cnt in enumerate(contours):
        x, y, w, h = cv.boundingRect(cnt)
```

```
area = cv.contourArea(cnt)
```

Obtain the target and drive the robotic arm to grab block.
The process is as follows:
Raise the robotic arm -> open the clip -> move to the block position -> close the clip -> lift -> move to the target position -> open the clip -> reset the robotic arm.