Garbage sorting

1. Functional description

The garbage sorting function is based on the single garbage sorting function, adding the function of identifying multiple garbage blocks and sorting multiple garbage blocks.

Note: Before starting the program, please follow the [Assembly and Assembly Tutorial] -> [Install Map] tutorial and install the map correctly before operating.

The path for saving the file of the robot arm position calibration is ~/dofbot_ws/src/dofbot_garbage_yolov5/XYT_config.txt.

2. Code block design

- Import header file

```
import cv2 as cv
import threading
from time import sleep
import ipywidgets as widgets
from IPython.display import display
from identify_target import identify_GetTarget
from dofbot_utils.dofbot_config import *
from dofbot_utils.fps import FPS
```

- Create an instance and initialize parameters

```
target = identify_GetTarget()
calibration = Arm_Calibration()
num = 0
dp = []
xy = [90, 106]
msg = {}
threshold = 116
debug_pos = False
model = "General"
color_list = {'1': 'red', '2': 'green', '3': 'blue', '4': 'yellow'}
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_path="/home/jetson/dofbot_ws/src/dofbot_color_identify/scripts/HSV_config.tx
t"
# XYT参数路径  XYT Parameter path
XYT_path="/home/jetson/dofbot_ws/src/dofbot_color_sorting/scripts/XYT_config.txt
"
try: read_HSV(HSV_path,color_hsv)
except Exception: print("Read HSV_config Error !!!")
try: xy, threshold = read_XYT(XYT_path)
except Exception: print("Read XYT_config Error !!!")
```

```
import Arm_Lib
arm = Arm_Lib.Arm_Device()
joints_0 = [xy[0], xy[1], 0, 0, 90, 30]
arm.Arm_serial_servo_write6_array(joints_0, 1000)
fps = FPS()
```

- Create controls

```
button_layout      = widgets.Layout(width='320px', height='60px',
align_self='center')
output = widgets.Output()
# 调整滑杆 Adjust the slider
joint1_slider      = widgets.IntSlider(description='joint1 :'   ,    value=xy[0]
    , min=70 , max=110, step=1, orientation='horizontal')
joint2_slider      = widgets.IntSlider(description='joint2 :'   ,    value=xy[1]
    , min=90, max=150, step=1, orientation='horizontal')
threshold_slider   = widgets.IntSlider(description='threshold :',
value=threshold , min=0  , max=255, step=1, orientation='horizontal')

# 进入标定模式  Enter calibration mode
calibration_model  = widgets.Button(description='calibration_model',
button_style='primary', layout=button_layout)
calibration_ok     = widgets.Button(description='calibration_ok',
button_style='success', layout=button_layout)
calibration_cancel = widgets.Button(description='calibration_cancel',
button_style='danger', layout=button_layout)
# 选择抓取颜色   Select grab color
color_list_one     = widgets.Dropdown(options=['red', 'green', 'blue', 'yellow',
'none'], value='red', disabled=False)
color_list_two     = widgets.Dropdown(options=['red', 'green', 'blue', 'yellow',
'none'], value='green', disabled=False)
color_list_three   = widgets.Dropdown(options=['red', 'green', 'blue', 'yellow',
'none'], value='blue', disabled=False)
color_list_four    = widgets.Dropdown(options=['red', 'green', 'blue', 'yellow',
'none'], value='yellow', disabled=False)
# 目标检测抓取  Target detection and capture
target_detection   = widgets.Button(description='target_detection',
button_style='info', layout=button_layout)
reset_color_list   = widgets.Button(description='reset_color_list',
button_style='info', layout=button_layout)
grap = widgets.Button(description='grap', button_style='success',
layout=button_layout)
# 退出  exit
exit_button = widgets.Button(description='Exit', button_style='danger',
layout=button_layout)
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
color_down = widgets.HBox([exit_button, reset_color_list],
layout=widgets.Layout(align_self='center'));
color_img = widgets.VBox([imgbox, color_down],
layout=widgets.Layout(align_self='center'));
color_identify = widgets.VBox(
    [joint1_slider, joint2_slider, threshold_slider, calibration_model,
calibration_ok, calibration_cancel,
     color_list_one, color_list_two, color_list_three, color_list_four,
target_detection, grap],
```

```
    layout=widgets.Layout(align_self='center'));
controls_box = widgets.HBox([color_img, color_identify],
layout=widgets.Layout(align_self='center'))
```

- Calibration callback

```
def calibration_model_Callback(value):
    global model
    model = 'Calibration'
    with output: print(model)
def calibration_OK_Callback(value):
    global model
    model = 'calibration_OK'
    with output: print(model)
def calibration_cancel_Callback(value):
    global model
    model = 'calibration_Cancel'
    with output: print(model)
calibration_model.on_click(calibration_model_Callback)
calibration_ok.on_click(calibration_OK_Callback)
calibration_cancel.on_click(calibration_cancel_Callback)
```

- Color selection sequence

```
# 选择颜色  Select color
def color_list_one_Callback(value):
    global model,color_list
    model="General"
    if not isinstance(value['new'],str):return
    if value['new'] == "none":
        if '1' in color_list:del color_list['1']
    elif value['new'] == "red":
        color_list['1'] = "red"
    elif value['new']== "green":
        color_list['1'] = "green"
    elif value['new'] == "blue":
        color_list['1'] = "blue"
    elif value['new'] == "yellow":
        color_list['1'] = "yellow"
    with output:
        print("color_list_three_Callback clicked.",color_list)
def color_list_two_Callback(value):
    global model,color_list
    model="General"
    if not isinstance(value['new'],str):return
    if value['new'] == "none":
        if '2' in color_list:del color_list['2']
    elif value['new'] == "red":
        color_list['2'] = "red"
    elif value['new'] == "green":
        color_list['2'] = "green"
    elif value['new'] == "blue":
        color_list['2'] = "blue"
    elif value['new'] == "yellow":
        color_list['2'] = "yellow"
    with output:
```

```
        print("color_list_three_Callback clicked.",color_list)
def color_list_three_Callback(value):
    global model,color_list
    model="General"
    if not isinstance(value['new'],str):return
    if value['new'] == "none":
        if '3' in color_list:del color_list['3']
    elif value['new'] == "red":
        color_list['3'] = "red"
    elif value['new'] == "green":
        color_list['3'] = "green"
    elif value['new'] == "blue":
        color_list['3'] = "blue"
    elif value['new'] == "yellow":
        color_list['3'] = "yellow"
    with output:
        print("color_list_three_Callback clicked.",color_list)
def color_list_four_Callback(value):
    global model,color_list
    model="General"
    if not isinstance(value['new'],str):return
    if value['new'] == "none":
        if '4' in color_list:del color_list['4']
    elif value['new'] == "red":
        color_list['4'] = "red"
    elif value['new'] == "green":
        color_list['4'] = "green"
    elif value['new'] == "blue":
        color_list['4'] = "blue"
    elif value['new'] == "yellow":
        color_list['4'] = "yellow"
    with output:
        print("color_list_four_Callback clicked.",color_list)
color_list_one.observe(color_list_one_Callback)
color_list_two.observe(color_list_two_Callback)
color_list_three.observe(color_list_three_Callback)
color_list_four.observe(color_list_four_Callback)
```

- Switch Mode

```
def target_detection_Callback(value):
    global model, debug_pos
    model = 'Detection'
    debug_pos = True
    with output: print(model)
def reset_color_list_Callback(value):
    global model
    model = 'Reset_list'
    with output: print(model)
def grap_Callback(value):
    global model
    model = 'Grap'
    with output: print(model)
def exit_button_Callback(value):
    global model
    model = 'Exit'
    with output: print(model)
```

```
target_detection.on_click(target_detection_Callback)
reset_color_list.on_click(reset_color_list_Callback)
grap.on_click(grap_Callback)
exit_button.on_click(exit_button_Callback)
```

- Main program

```
def camera():
    global color_hsv,model,dp,msg,color_list,debug_pos
    # 打开摄像头 Open camera
    capture = cv.VideoCapture(0)
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    index=1
    # Be executed in loop when the camera is opened normally
    # 当摄像头正常打开的情况下循环执行
    while capture.isOpened():
        try:
            _, img = capture.read()
            fps.update_fps()
            xy=[joint1_slider.value,joint2_slider.value]
            if model == 'Calibration':
                _, img =
calibration.calibration_map(img,xy,threshold_slider.value)
            if model == 'calibration_OK':
                try: write_XYT(XYT_path,xy, threshold_slider.value)
                except Exception: print("File XYT_config Error !!!")
                dp, img =
calibration.calibration_map(img,xy,threshold_slider.value)
                model="General"
            if len(dp) != 0: img = calibration.Perspective_transform(dp, img)
            if model == 'calibration_Cancel':
                dp = []
                msg= {}
                model="General"
            if len(dp)!= 0 and len(color_list)!= 0 and model == 'Detection':
                img, msg = target.select_color(img, color_hsv, color_list)
                # print("Detection msg:", msg)
                if debug_pos:
                    debug_pos = False
                    print("detect msg:", msg)
            if model=="Reset_list":
                msg={}
                color_list = {}
                color_list_one.value = 'none'
                color_list_two.value = 'none'
                color_list_three.value = 'none'
                color_list_four.value = 'none'
                model="General"
            if len(msg)!= 0 and model == 'Grap':
                print("grasp msg:", msg)
                threading.Thread(target=target.target_run, args=
(msg,xy)).start()
                msg={}
                model="Detection"
            if model == 'Exit':
                capture.release()
```

```
                break
            index+=1
            fps.show_fps(img)
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except KeyboardInterrupt:capture.release()
```

- Start up

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

3. Start the program

Start the ROS node service

Open the system terminal and enter the following command. If it is already started, you don't need to start it again.

```
sudo systemctl start yahboom_arm.service
```

Start the program

Open the jupyterlab webpage and find the corresponding .ipynb program file.

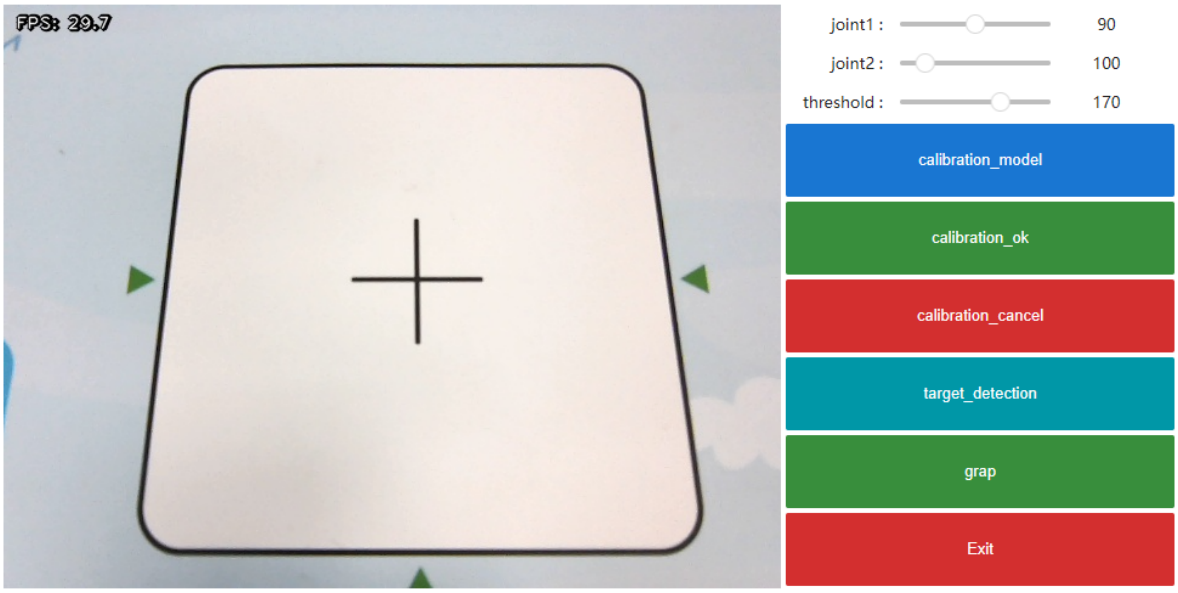
Code path:

```
~/dofbot_ws/src/dofbot_garbage_yolov5/Garbage_Sorting.ipynb
```

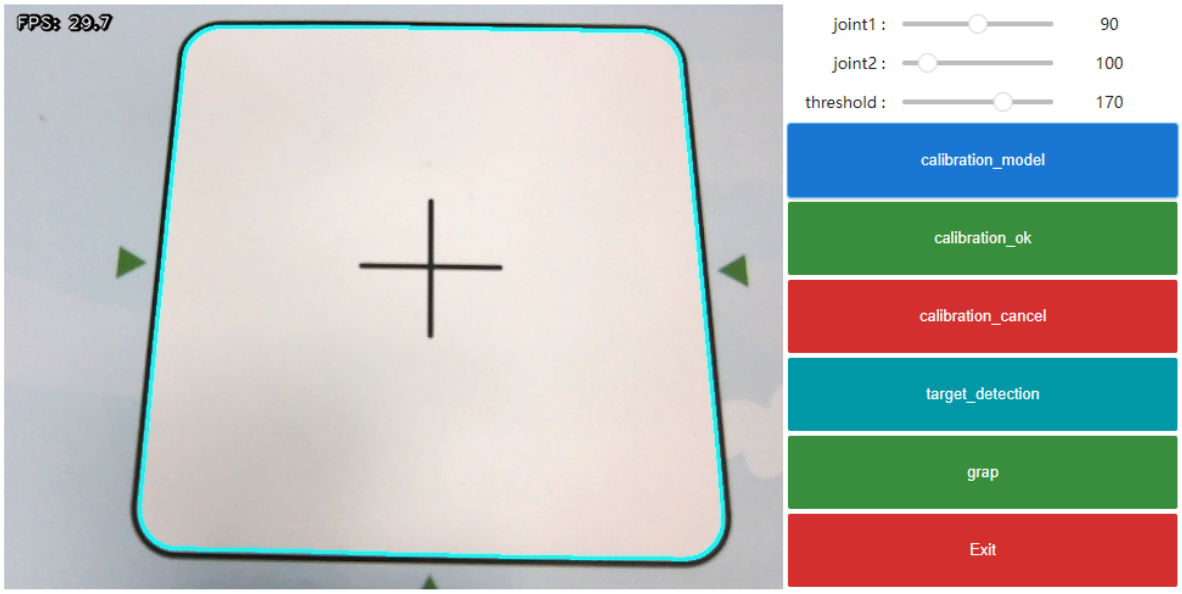
Then click Run all commands.

4. Experimental effect

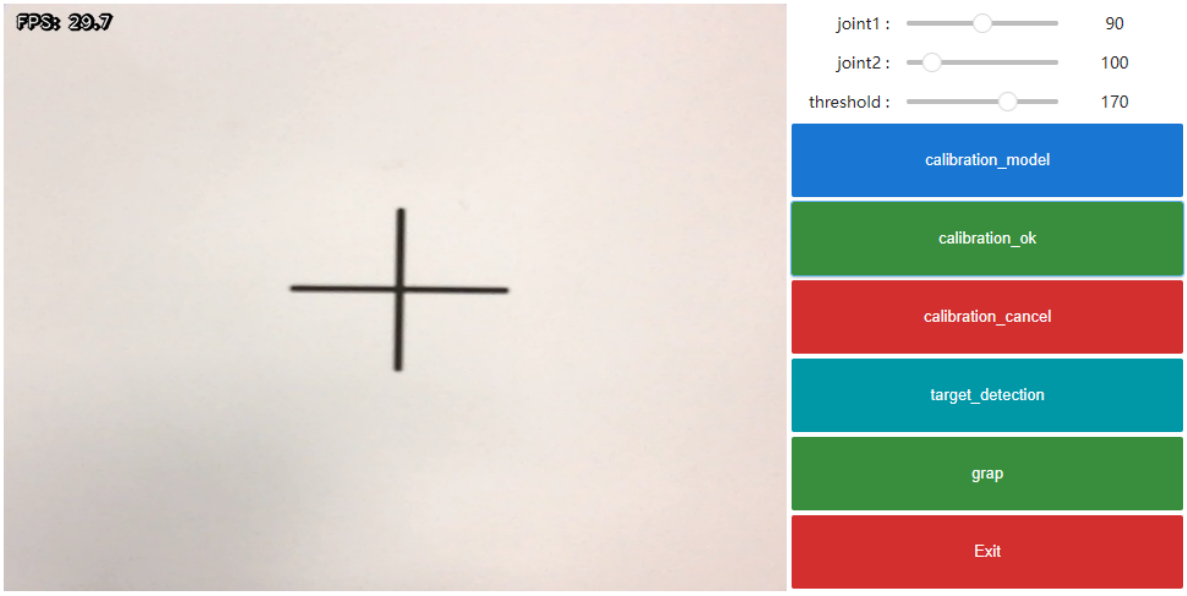
After the program runs, the jupyterlab webpage will display the control, the camera screen on the left, and the functions of the related buttons on the right.



Click [calibration_model] to enter the calibration mode, and adjust the upper robot joint slider and threshold slider to make the displayed blue line overlap with the black line of the recognition area.



Click [calibration_ok] to calibrate OK, and the camera screen will switch to the recognition area perspective.

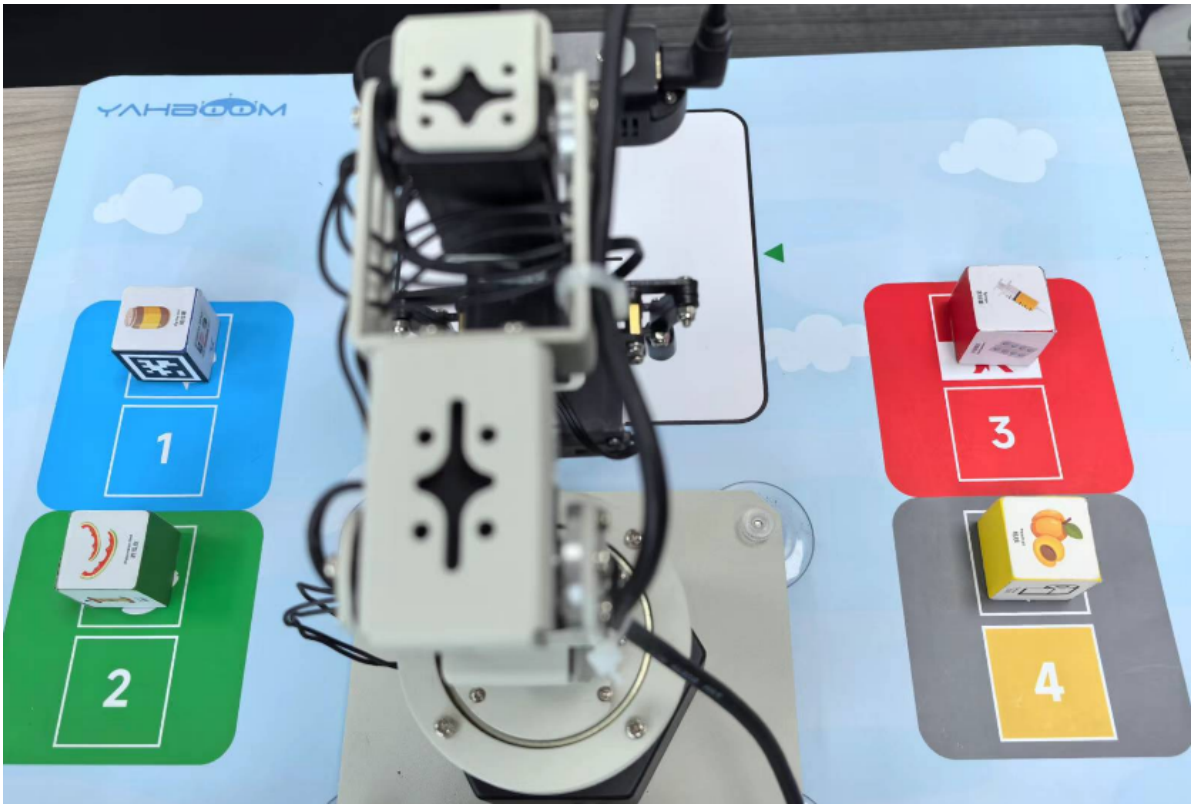


Place the building block in the recognition area, with the side with the garbage sticker facing up, the text direction consistent with the camera direction, and the bottom edge parallel to the bottom edge of the recognition area as much as possible.

Note: Due to the height of the building block, try to place it in the middle area when placing it, otherwise the camera will not be able to see the garbage sticker completely and cannot recognize it correctly.

Then click [target_detection] to start color recognition. If the color recognition is inaccurate, please calibrate the color first and then run the program again.

Then click the [grap] button to start sorting. The system will sort the identified garbage into the corresponding garbage category area in order.



If you need to exit the program, please click the [Exit] button.

Since garbage identification requires loading model files and takes up a lot of memory space, [Exit] only ends the function and does not close the programs related to the model files. You need to close the current kernels before you can close the model file programs. Click [Kernel]->[Shut Down All Kernels] in the menu bar.