

Garbage Sorting(Orin)

1. Function Description

The garbage sorting function is based on the single garbage sorting function, adding the ability to recognize multiple garbage building blocks and sort multiple garbage building blocks.

Note: Before starting the program, please follow the [A2. Assembly Course] -> [Install the Map] tutorial to correctly install the map before proceeding with operations.

The robotic arm position calibration file is saved at
~/dofbot_pro/dofbot_garbage_yolov11/XYT_config.txt.

2. Code Block Design

- Import header files

```
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 instances, initialize parameters

```
target = garbage_identify()
calibration = Arm_Calibration()
arm = Arm_Lib.Arm_Device()
num=0
dp = []
msg = {}
xy = [90, 105]
debug_pos = False
threshold = 118
model = "General"
# XYT参数路径 XYT Parameter path
XYT_path="/home/jetson/dofbot_pro/dofbot_garbage_yolov11/XYT_config.txt"
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 widgets

```

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)

# Target detection and capture
target_detection   = widgets.Button(description='target_detection',
                                    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(alignment='center'))
garbage_identify = widgets.VBox([
    [joint1_slider, joint2_slider, threshold_slider, calibration_model,
     calibration_ok, calibration_cancel, target_detection, grap, exit_button],
    layout=widgets.Layout(alignment='center'));
controls_box = widgets.HBox([imgbox, garbage_identify],
                           layout=widgets.Layout(alignment='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)

```

- Mode switching

```

def target_detection_callback(value):
    global model, debug_pos
    model = 'Detection'
    with output: print(model)
    debug_pos = True
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)
grap.on_click(grap_callback)
exit_button.on_click(exit_button_callback)

```

- Main program

```

def camera():
    global model, dp, msg, debug_pos
    # Open camera
    capture = cv.VideoCapture(0, cv.CAP_V4L2)
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    m_fps = 0
    t_start = time.time()
    fps = FPS()
    # Be executed in loop when the camera is opened normally
    while capture.isOpened():
        try:
            _, img = capture.read()
            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 model == 'Detection':
                img, msg = target.garbage_run(img)
                if debug_pos:
                    debug_pos = False
                    print("detect msg", msg)
            if len(msg)!= 0 and model == 'Grap':
                print("grap msg", msg)
                threading.Thread(target=target.garbage_grap, args=(msg,
xy,)).start()
                msg={}

```

```

        model="Detection"
if model == 'Exit':
    capture.release()
    break
fps.update_fps()
fps.show_fps(img)

    imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
except KeyboardInterrupt:
    capture.release()
except:
    capture.release()

```

- Startup

```

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

```

3. Start the Program

Start ROS Node Service

Open the system terminal and enter the following command. If it's already running, there's no need to start it again.

```
ros2 run dofbot_pro_info kinematics_dofbot
```

Start the Program

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

Code path:

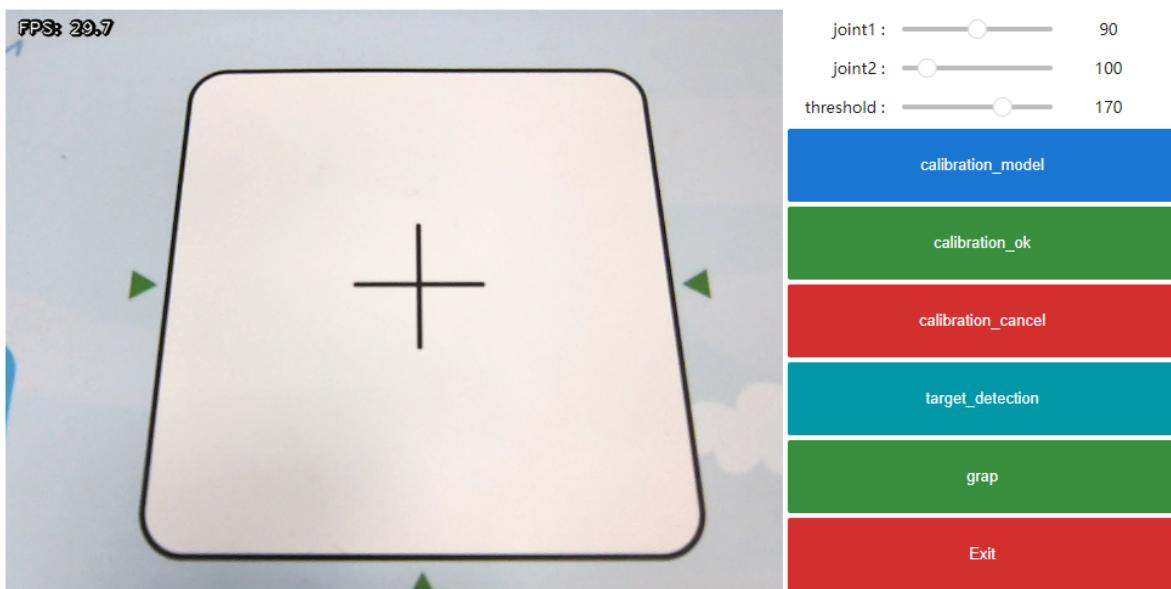
```
~/dofbot_pro/dofbot_garbage_yolov11/Garbage_Sorting.ipynb
```

Then click the run all command.

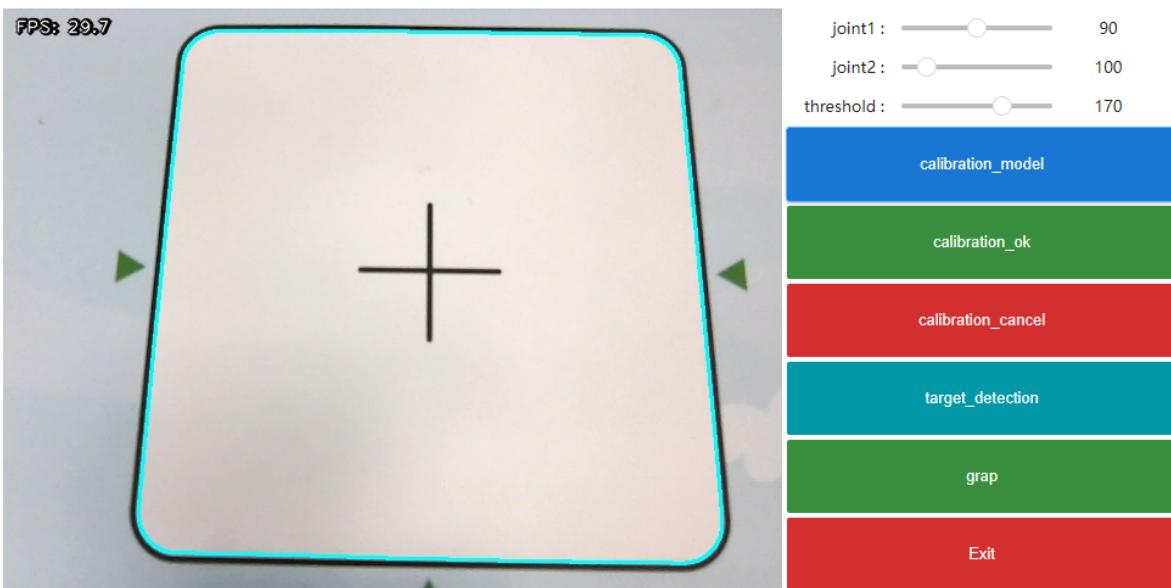


4. Experimental Effects

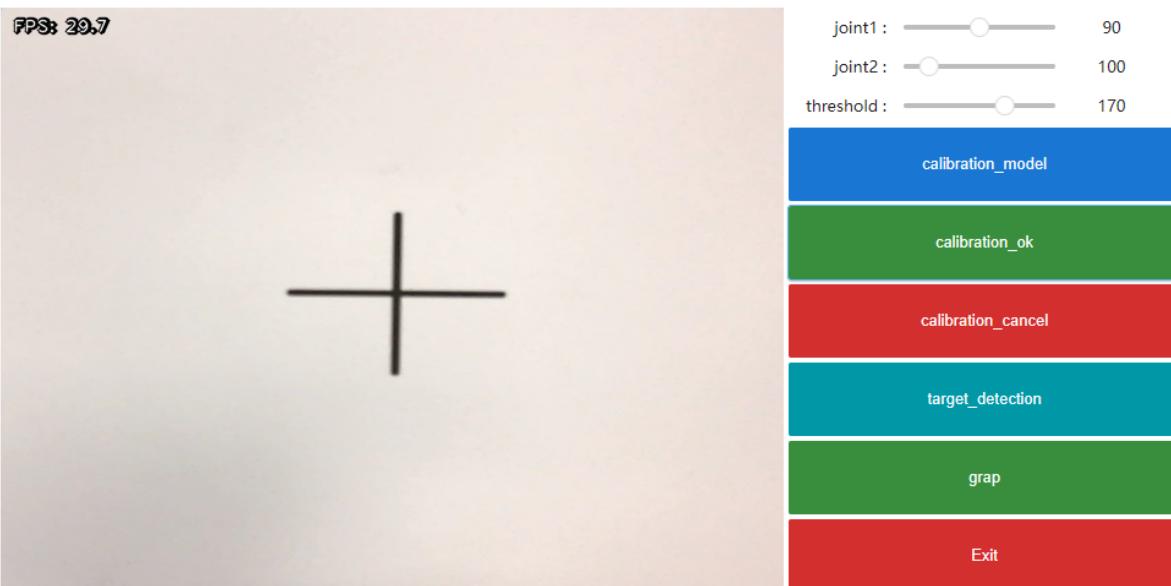
After the program runs, the jupyterlab webpage will display controls, with the camera display on the left and related button functions on the right.



Click [calibration_model] to enter calibration mode. Adjust the robotic arm joint sliders and threshold slider above to make the displayed blue line overlap with the black line of the recognition area.



Then click [calibration_ok] for calibration OK. At this time, the camera view switches to the recognition area perspective.

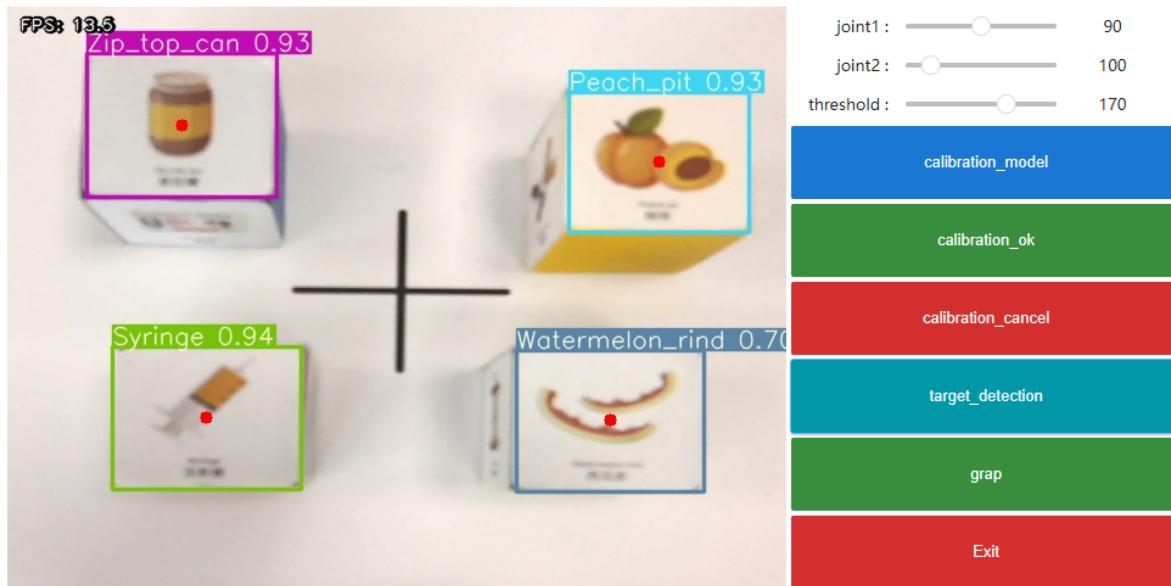


Place the building blocks in the recognition area with the garbage sticker side facing up, keeping the text direction consistent with the camera direction, and try to keep the bottom edge parallel to the bottom edge of the recognition area.

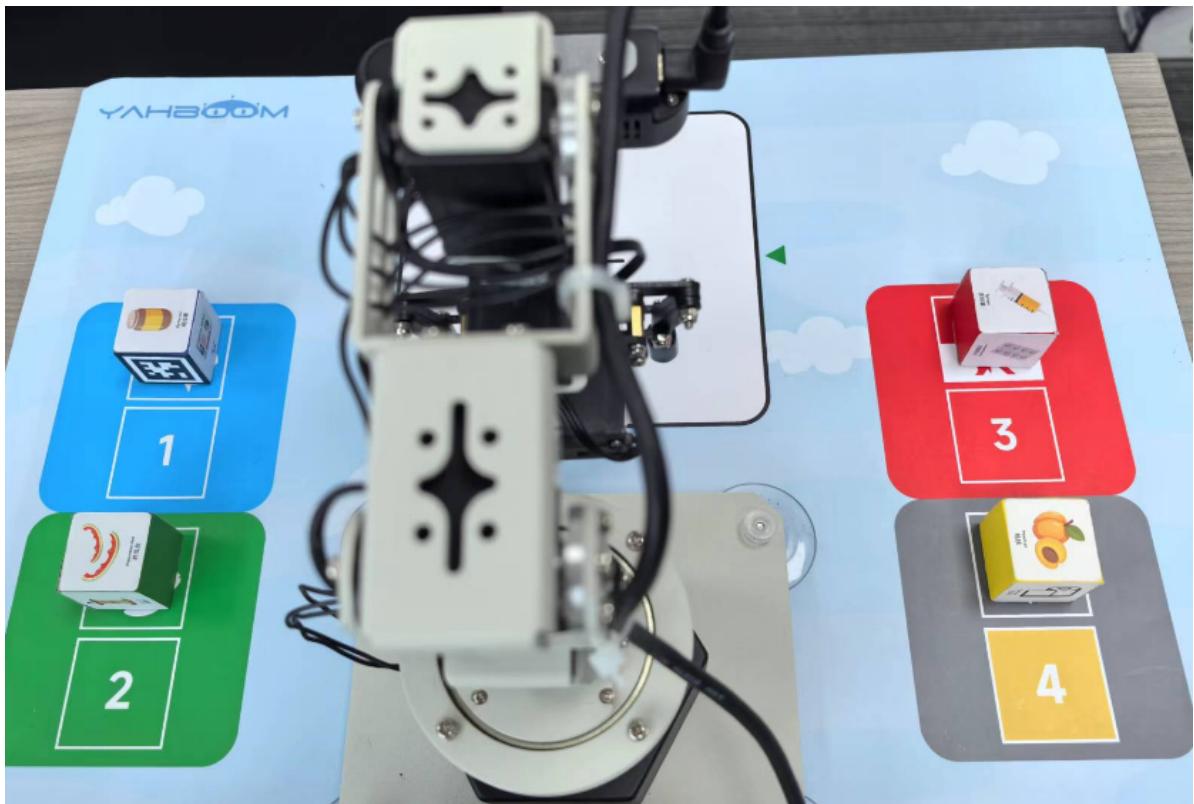
Note: Due to the height of the building blocks, try to place them in the central area when placing, otherwise the camera may not capture the complete garbage sticker, making correct recognition impossible.



Then click [target_detection] to start color recognition. If color recognition is not accurate, please calibrate the colors first and then restart the program.



Then click the [grap] button to start sorting. The system will sort the recognized garbage sequentially into the corresponding garbage category areas.



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

Since garbage recognition requires loading model files and occupies a large amount of memory space, [Exit] only ends the functionality and does not close the model file-related programs. You need to close the current kernels to close the model file programs. Click [Kernel] -> [Shut Down All Kernels] in the menu bar sequentially.

