Label code sorting

1. Functional description

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

The path for saving the robot arm position calibration file is ~/dofbot_pro/dofbot_apriltag/scripts/XYT_config.txt.

Code path:

```
~/dofbot_pro/dofbot_apriltag/scripts/Apriltag_Sorting.ipynb
```

2. Code block design

- Import header files

```
import cv2 as cv
import threading
import time
import ipywidgets as widgets
from IPython.display import display

from apriltag_identify import ApriltagIdentify
from apriltag_grasp import Apriltag_Grasp
from dofbot_utils.fps import FPS
from dofbot_utils.dofbot_config import *
```

- Create an instance and initialize parameters

```
apriltag_Identify = ApriltagIdentify()
target = Apriltag_Grasp()
calibration = Arm_Calibration()
num = 0
dp = []
xy = [90, 106]
msg = {}
threshold=120
debug_pos = False
model = "General"

# XYT parameter path XYT Parameter path
XYT_path="/home/jetson/dofbot_pro/dofbot_apriltag/scripts/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 controls

```
button_layout = widgets.Layout(width='320px', height='60px', align_self='center')
output = widgets.Output()
# 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')

# Enter calibration mode 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 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)
# exit 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_identify = widgets.VBox(
 [joint1_slider, joint2_slider, threshold_slider, calibration_model,
calibration_ok, calibration_cancel,
 target_detection, grap, exit_button],
 layout=widgets.Layout(align_self='center'));
controls_box = widgets.HBox([imgbox, 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)
```

- Switching Mode

```
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 color_hsv,model,dp,msg,debug_pos
    # 打开摄像头 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)
    # 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 model == 'Detection':
                img, msg = apriltag_Identify.getApriltagPosMsg(img)
                # print("Detection msg:", msg)
                if debug_pos:
                    debug_pos = False
                    print("detect msg:", msg)
            if len(msg)!= 0 and model == 'Grap':
                print("grasp msg:", msg)
                threading.Thread(target=target.target_sorting, args=
(msg,xy)).start()
                msg={}
                model="Detection"
            if model == 'Exit':
                capture.release()
                break
            fps.show_fps(img)
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except Exception as e:
            print("program end")
            print(e)
            capture.release()
```

- 启动

```
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.

```
ros2 run dofbot_pro_info kinemarics_dofbot
```

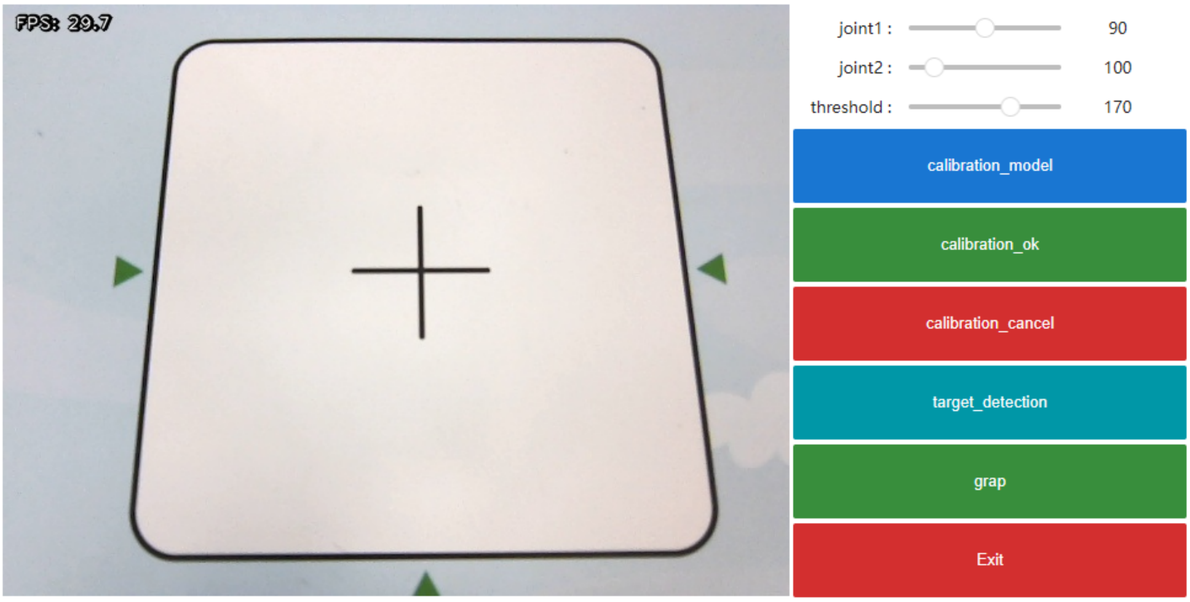
Start the program

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

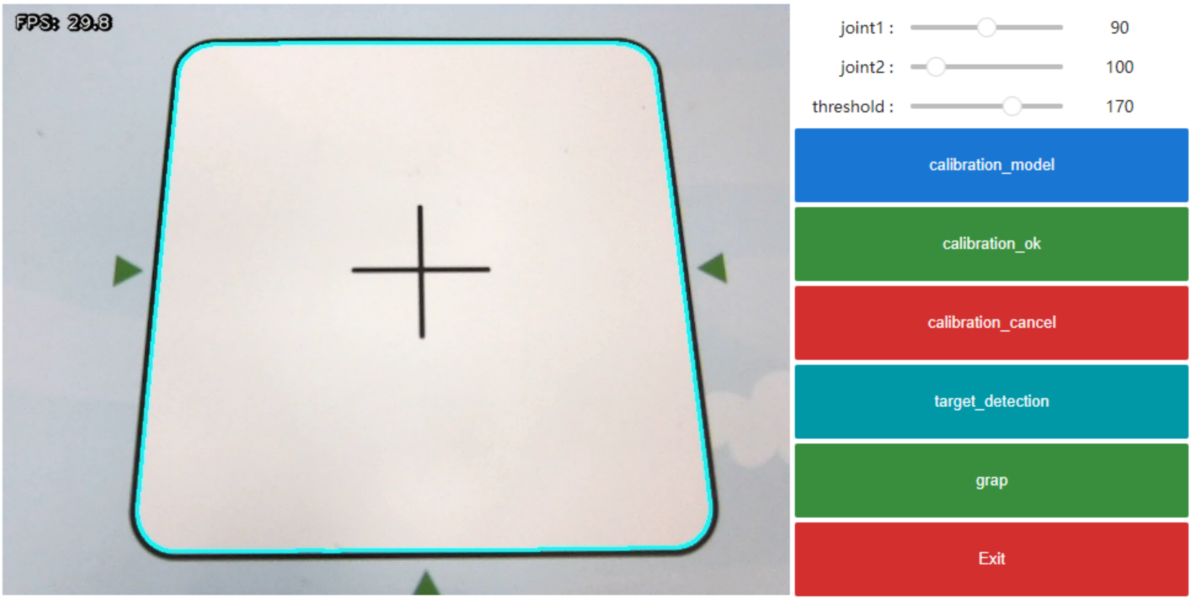
Then click Run All Commands.

4. Experimental operation and effect

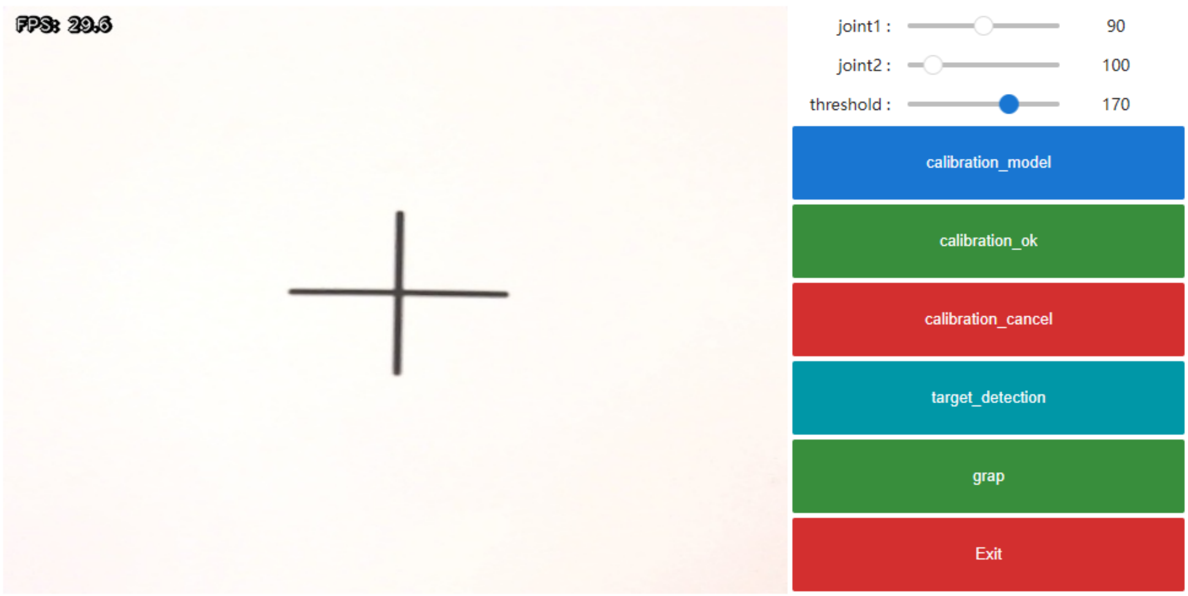
After the program is running, the jupyterlab webpage will display the control, the camera screen on the left, and the functions of the related buttons on the right.



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.

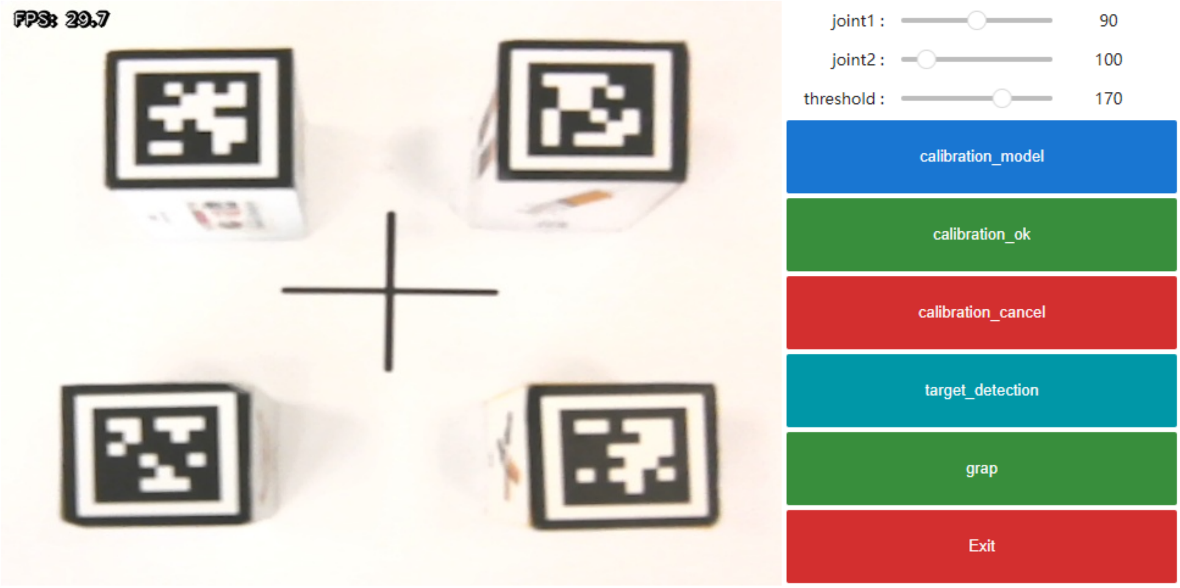


Click [calibration_ok] to calibrate OK, and the camera screen will switch to the recognition area perspective.

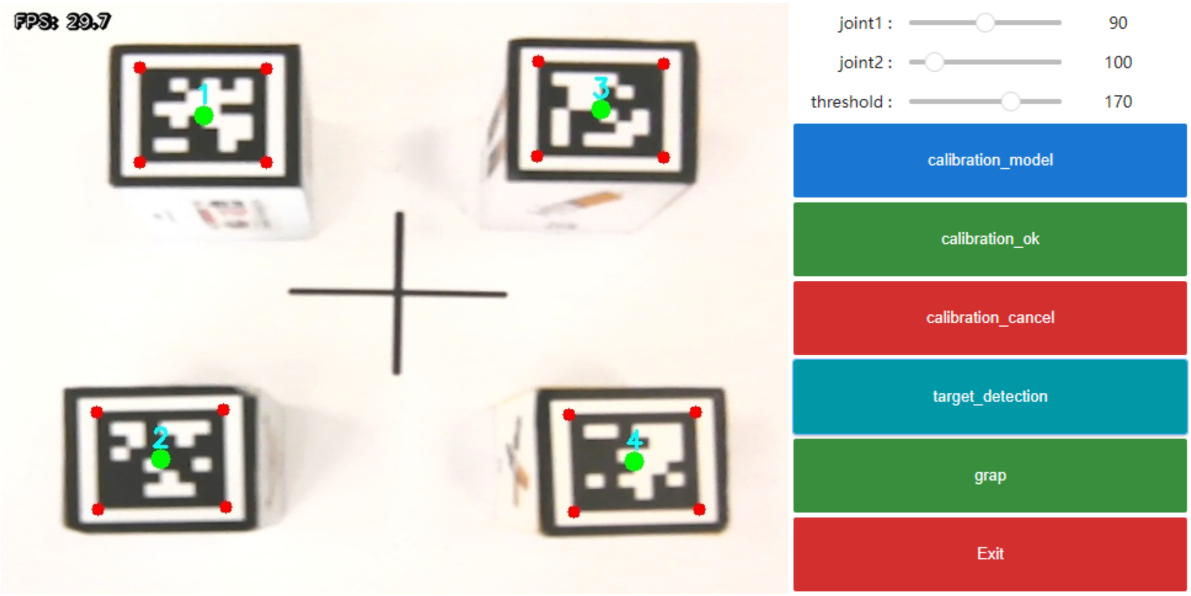


Place the building block in the recognition area, with the label code facing up, and the position is placed in the recognition area, and the bottom edge is kept parallel to the bottom edge of the recognition area as much as possible.

Note: Due to the height of the building blocks, try to place them in the middle area when placing them, otherwise the building block label code may not be recognized.



Then click [target_detection] to start identifying the label code.



Then click the [grap] button to start sorting. The system will sort the identified label codes into the digital area in the order of ID numbers.

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