Color stacking

1. Functional description

The color stacking function is based on the color sorting function, and the position of the clamping placement is modified to the stacking method of building blocks. The function operation is similar.

The color recognition function uses HSV color recognition. The path where the HSV color calibration file is saved is ~/dofbot_ws/src/dofbot_color_identify/scripts/HSV_config.txt. If the color recognition is not accurate enough, please recalibrate the HSV value of the building block color according to the [Visual Basic Course] -> [Color Calibration] course. After the calibration operation is completed, it will be automatically saved to the HSV_config file. Rerun the program without additional code modification.

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

The path where the file for the robot position calibration is saved is ~/dofbot_pro/dofbot_color_stacking/scripts/XYT_config.txt.

Code path:

```
~/dofbot_pro/dofbot_color_stacking/scripts/Color_Stacking.ipynb
```

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 dofbot_utils.dofbot_config import *
from stacking_target import stacking_GetTarget
from dofbot_utils.fps import FPS
```

- Create an instance and initialize parameters

```
target      = stacking_GetTarget()
calibration = Arm_Calibration()
num = 0
dp = []
xy = [90, 106]
msg   = {}
threshold = 120
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))}
# XYT参数路径  XYT Parameter path
HSV_path="/home/jetson/dofbot_pro/dofbot_color_identify/scripts/HSV_config.txt"
XYT_path="/home/jetson/dofbot_pro/dofbot_color_stacking/scripts/XYT_config.txt"
try: read_HSV(HSV_path,color_hsv)
except Exception: print("No HSV_config file!!!")
try: xy, threshold = read_XYT(XYT_path)
except Exception: print("No XYT_config file!!!")
```

```
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)
# Select grab color 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)
# Target detection and capture 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)
# 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_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)
```

- Switching 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, cv.CAP_V4L2)
    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("---",color_hsv['green'])
                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 Exception as e:
            print("program end")
            print(e)
            capture.release()
```

- Start

```
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
```

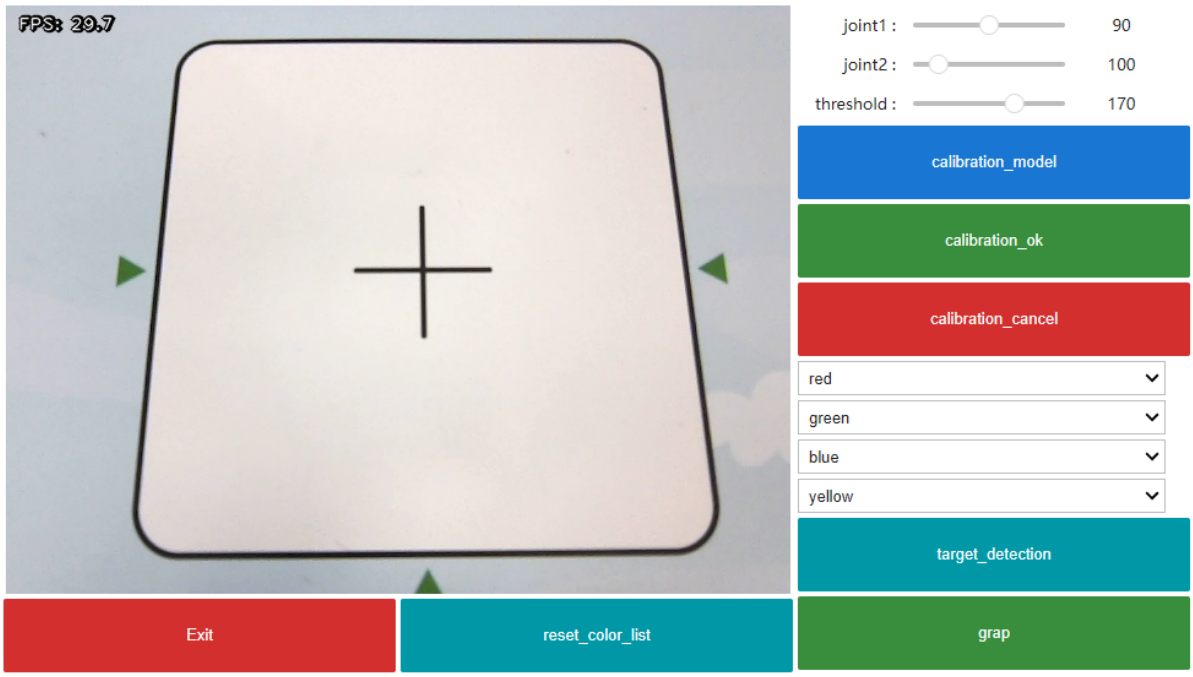
Start the program

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

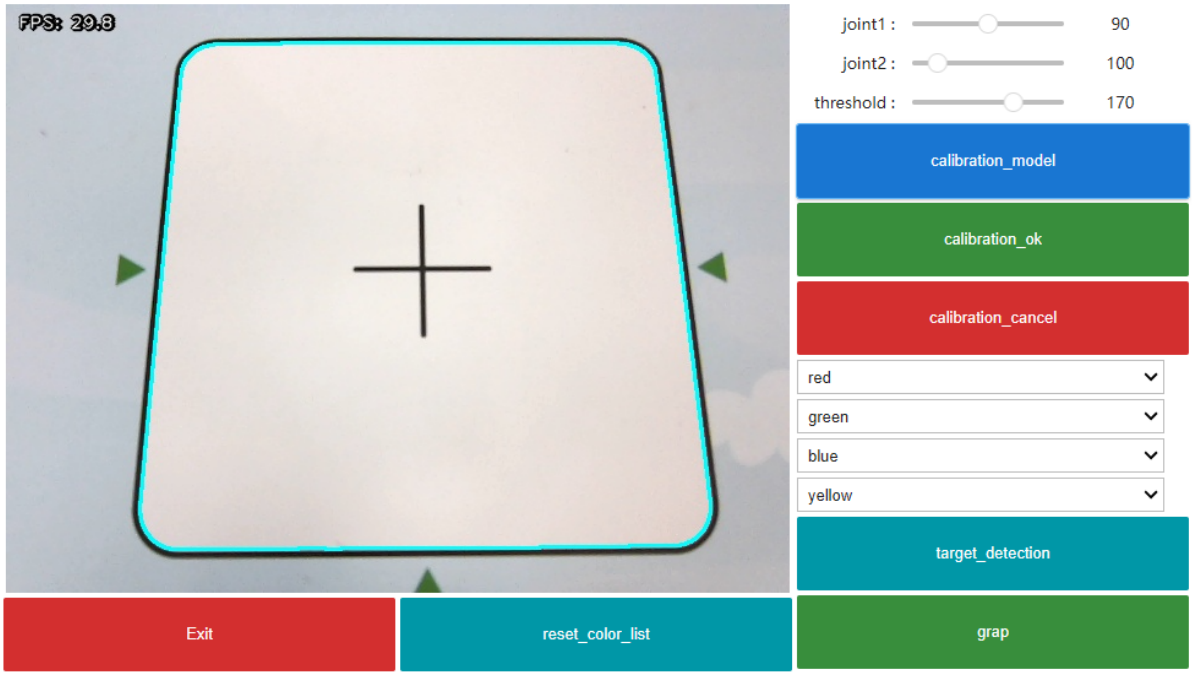
Then click Run All Commands.

4. Experimental operation and effect

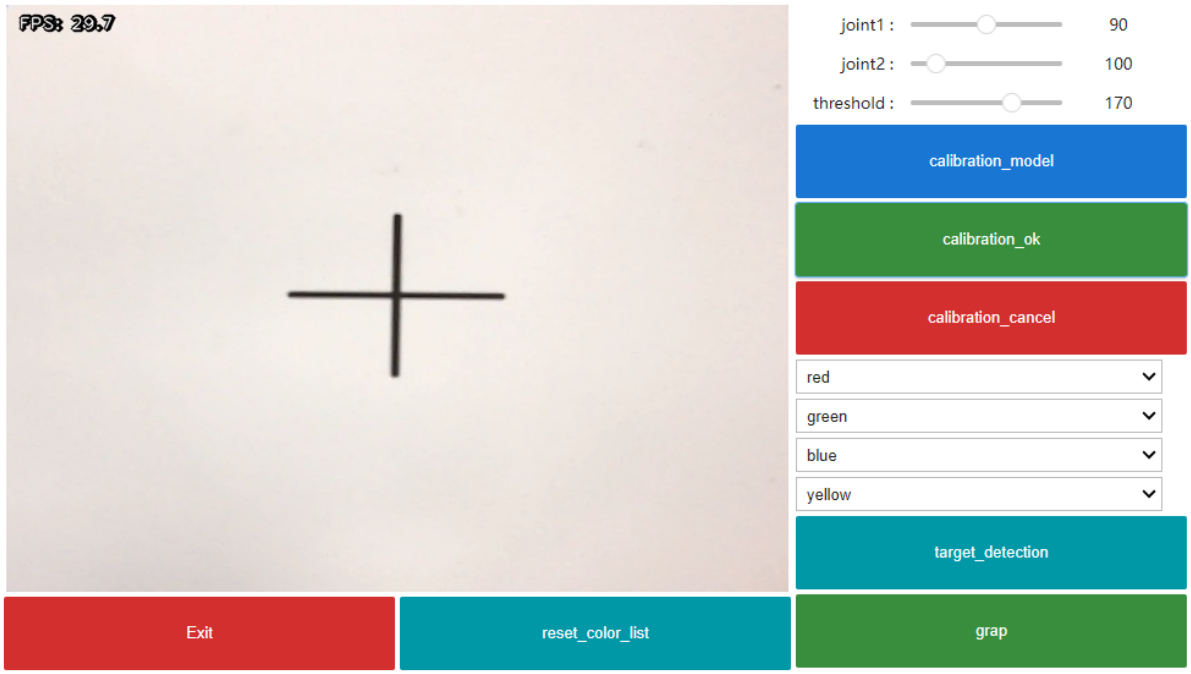
After the program is run, the jupyterlab webpage will display the control, the camera screen on the left, and the functions of the related buttons on the right.



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

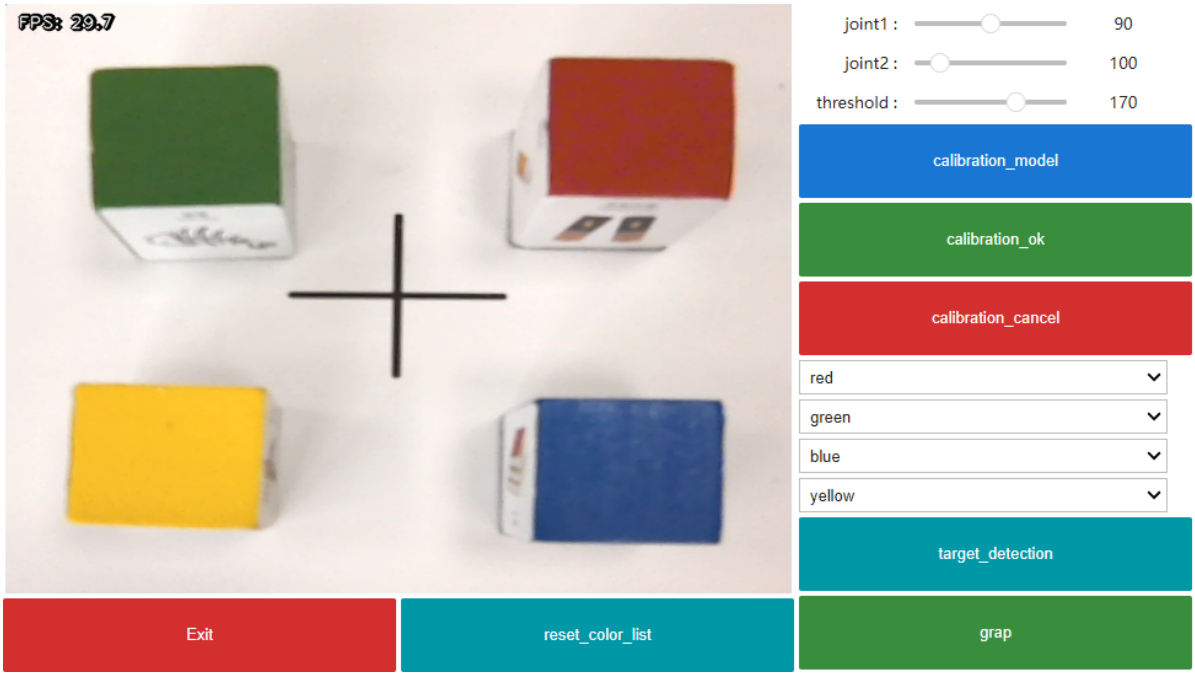


Click [calibration_ok] to calibrate OK, and the camera screen will switch to the recognition area perspective.

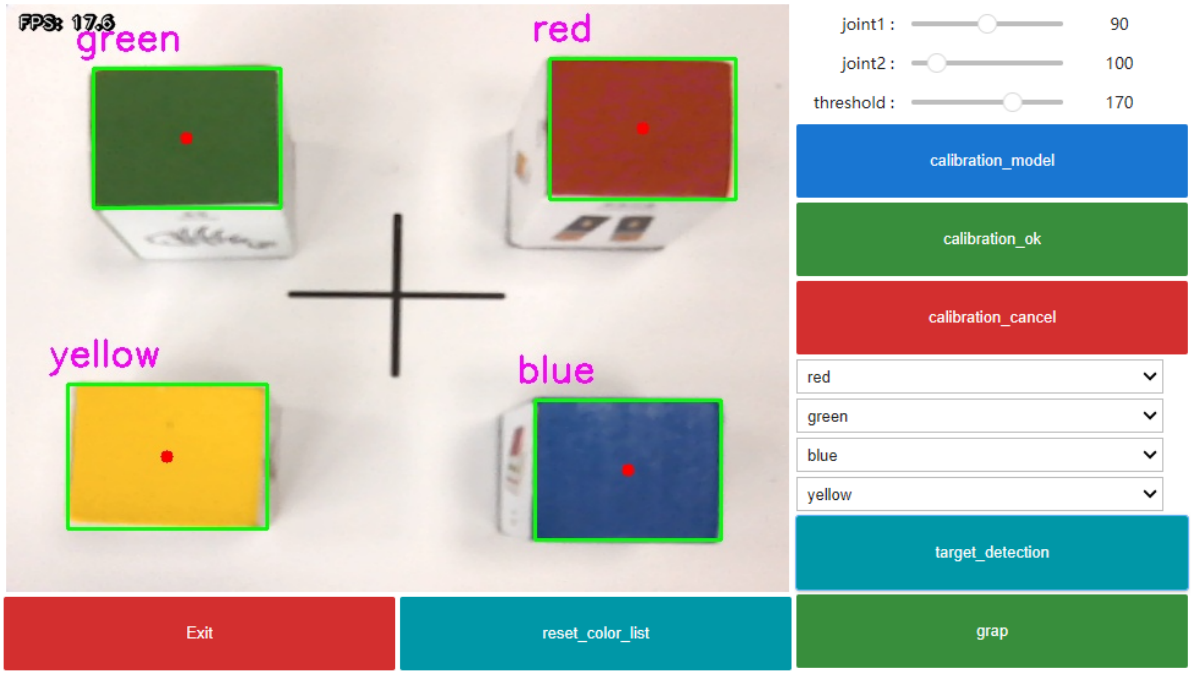


Place the block in the recognition area, with the colored side facing up and placed right in the recognition area, with the bottom edge parallel to the bottom edge of the recognition area as much as possible. Then select the color order on the right. The default selection is red, green, blue, and yellow. You can also click [reset_color_list] to reset all color options before making a selection.

Note: Due to the height of the blocks, try to place them in the middle area when placing them, otherwise the color of the blocks may not be fully recognized and errors may occur.



Then click [target_detection] to start color recognition. If the color recognition is inaccurate, calibrate the color before rerunning the program.



Then click the [grap] button to start sorting. The system will stack the identified colors in order into the green area.

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