ECE4310/CIE6042 Programming for Robotics

Project 2: Warehouse Robot

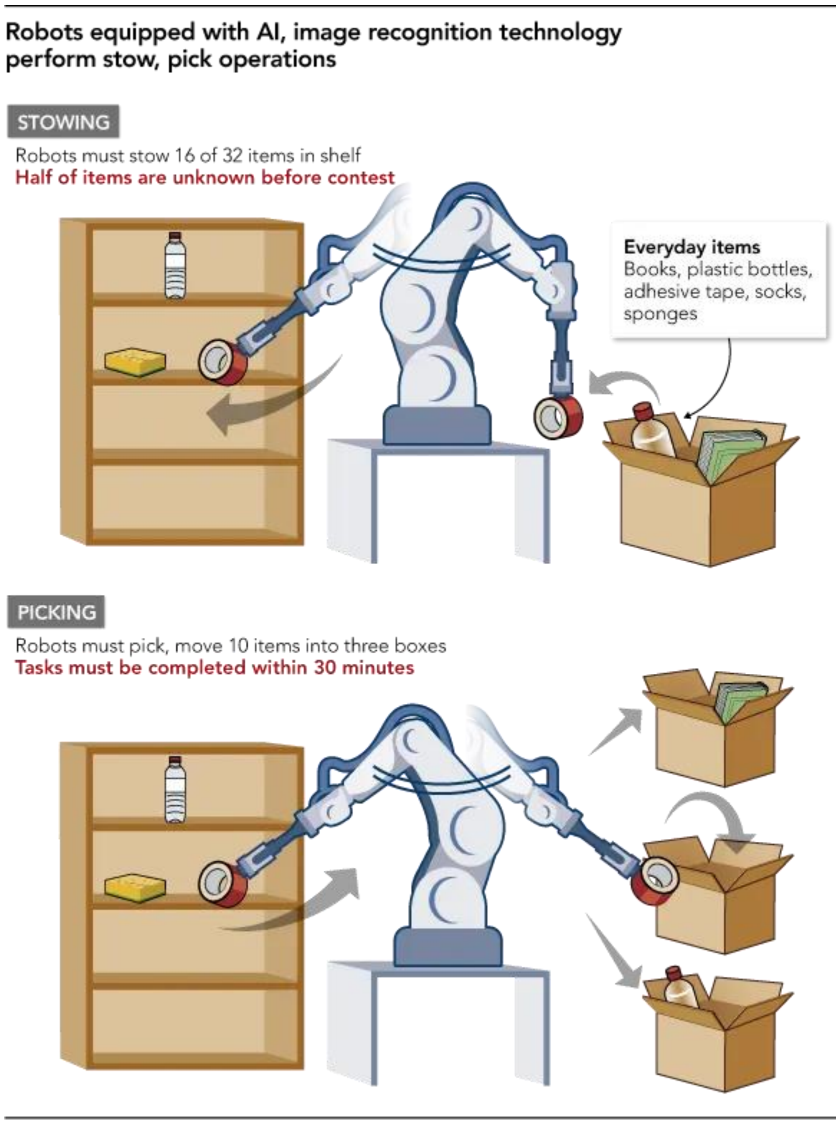
School of Science and Engineering

The Chinese University of Hong Kong, Shenzhen

2021-2022 Term 2

Motivation:

1. To integrate what you have learned and practiced in the previous labs to build a service robot.


Robots equipped with AI, image recognition technology perform stow, pick operations
STOWING
Robots must stow 16 of 32 items in shelf
Half of items are unknown before contest
Everyday items
Books, plastic bottles, adhesive tape, socks, sponges
PICKING
Robots must pick, move 10 items into three boxes
Tasks must be completed within 30 minutes

(Source: https://asia.nikkei.com/Business/Technology/Warehouse-robots-getting-better-at-sorting-things-out)

Fig. 1.

1. Intelligent Sorting System Development

1. The principle of eye calibration

The hand-eye calibration can be divided into: Eye-in-Hand (eye in the hand) and Eye-to-Hand (eye outside the hand), as shown in Figure 1.

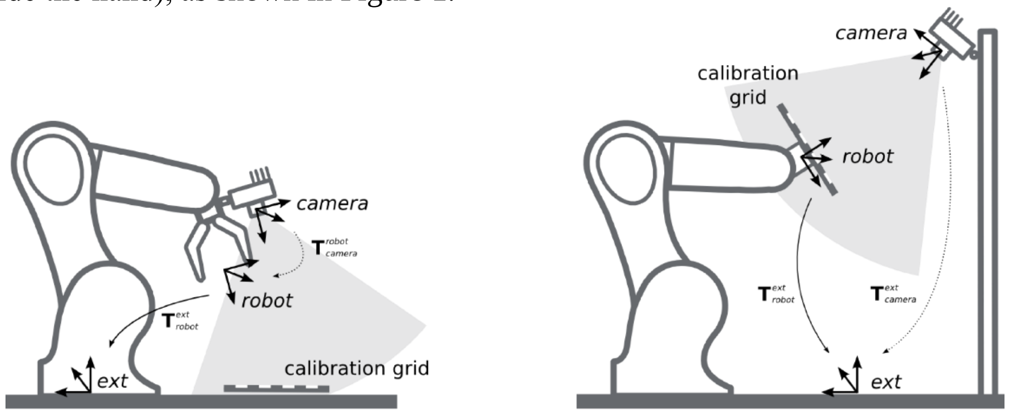


Fig. 2. Hand-eye calibration.

For Eye-in-Hand, the pose of the camera coordinate system relative to the robot world coordinate system is always changing, while the pose relative to the end effector is constant. For Eye-to-Hand, it is opposite. Therefore, for the former, the hand-eye calibration is obtained from the camera's pose relative to the robot's end effector coordinate system; for the latter, the hand-eye calibration is obtained from the eye's pose relative to the robot's base.

1.2 Color recognition

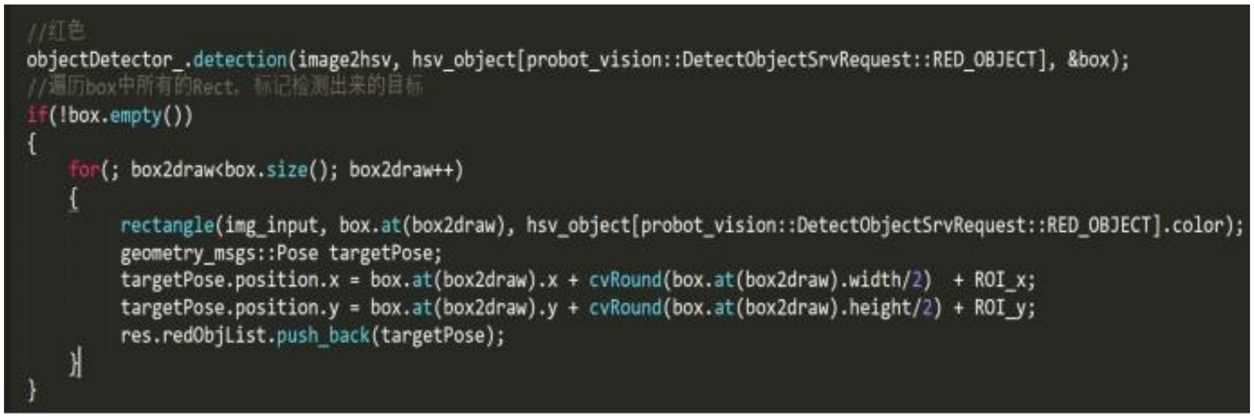
First, the three attributes of color are hue, lightness, and color. Although RGB is direct, there is no direct connection between R, G and B values and the three attributes of color. HSV (Hue, Saturation, Value) is a more appropriate to describe color, which is based on the intuitive properties of color created by A. R. Smith in 1978. The H parameter represents the color information, i.e., the position of the spectral color in which it is located. The purity S is a proportional value ranging from 0 to 1, which is expressed as a ratio between the purity of the selected color and the maximum purity of that color. v indicates the brightness of the color, ranging from 0 to 1.

Color is greatly influenced by the external environment, especially the light source. For example, the red color under warm light and the red color under cold light will be different in the camera picture. How to avoid the error caused by the external environment? The answer is to perform a threshold adjustment in the current environment.

Note: There is no direct correlation between HSV itself and light intensity.

The files under ***probot_vision/src*** are the core code used for color recognition in this project. The function of object_detector.cpp is to adjust the threshold parameter and find the color threshold in the current environment. The comments in the code are very detailed, so please understand the code against the comments. The ***main_object_detector.cpp*** focuses on "recognizing different colors", firstly, it goes through the pre-processing of image cropping, gray value normalization, color space transformation, and then traverses the target detection frame and marks the detected targets.

Here is the code to detect the red discs, and it's similar to detect other colors.



```
//红色
objectDetector_.detection(image2hsv, hsv_object[probot_vision::DetectObjectSrvRequest::RED_OBJECT], &box);
//遍历box中所有的Rect, 标记检测出来的目标
if(!box.empty())
{
    for(; box2draw<box.size(); box2draw++)
    {
        rectangle(img_input, box.at(box2draw), hsv_object[probot_vision::DetectObjectSrvRequest::RED_OBJECT].color);
        geometry_msgs::Pose targetPose;
        targetPose.position.x = box.at(box2draw).x + cvRound(box.at(box2draw).width/2)  + ROI_x;
        targetPose.position.y = box.at(box2draw).y + cvRound(box.at(box2draw).height/2) + ROI_y;
        res.redObjList.push_back(targetPose);
    }
}
```

Fig. 3. Core code for color recognition.

1.3 Visual recognition parameter adjustment

Vision sorting uses HSV color model for target object recognition, which reduces the influence of light on recognition and does not require color threshold resetting in general, but different cameras and complex environments can still have an impact, so when changing hardware or installation locations, please follow the steps below to adjust recognition parameters before vision calibration.

(1) Attach the camera to the end of the robot arm and connect it to the computer.

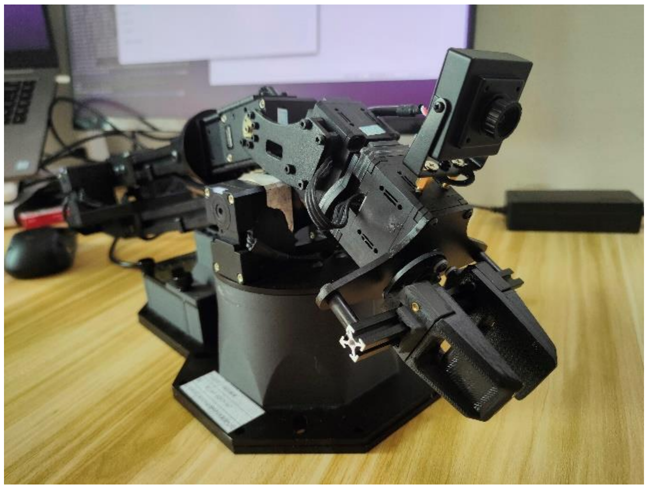


Fig. 4. Camera installation

(2) Start the robot arm program with the following command for the real robot.

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s use_actual:=true dof:=6
```

Or you can start the Gazebo simulation environment with following command:

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s
use_gazebo:=true dof:=6 world_name:=warehouse.world
```

After successful startup, move the robot arm to the position shown below to ensure the camera can see the target object.

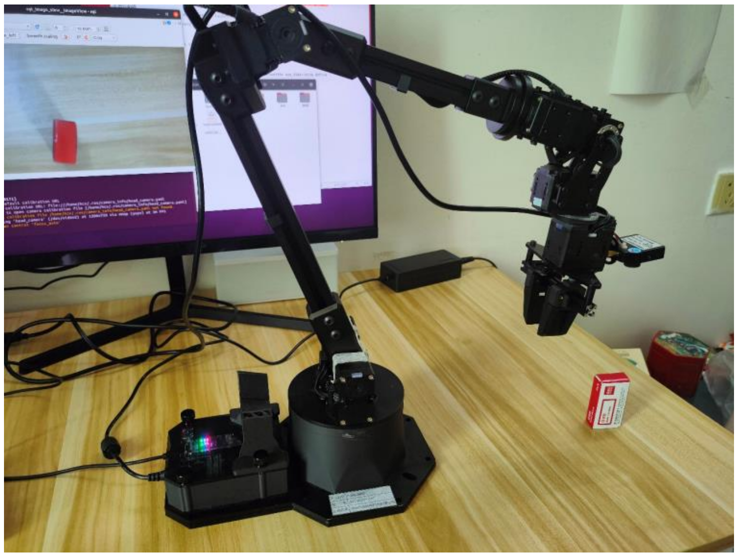


Fig. 5. Controlling the movement of the robot arm.

Restart a terminal and start the camera node using the following command for the real robot.

```
$ roslaunch object_color_detector usb_cam.launch
```

If you are using Gazebo simulation, the camera simulation has already started with Gazebo.

(3) After starting the ***rqt_image_view*** tool and configuring the subscription ***/usb_cam/image_raw*** (real robot) or ***/wx250s/camera/image_raw*** (simulation), you can see the live image of the camera and make sure that the area to be identified is all within the camera range.

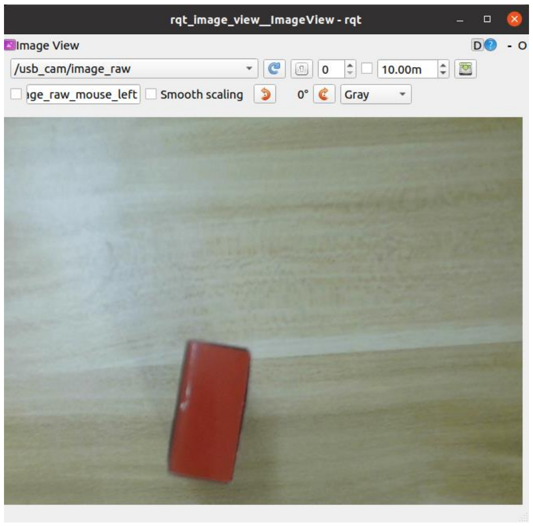
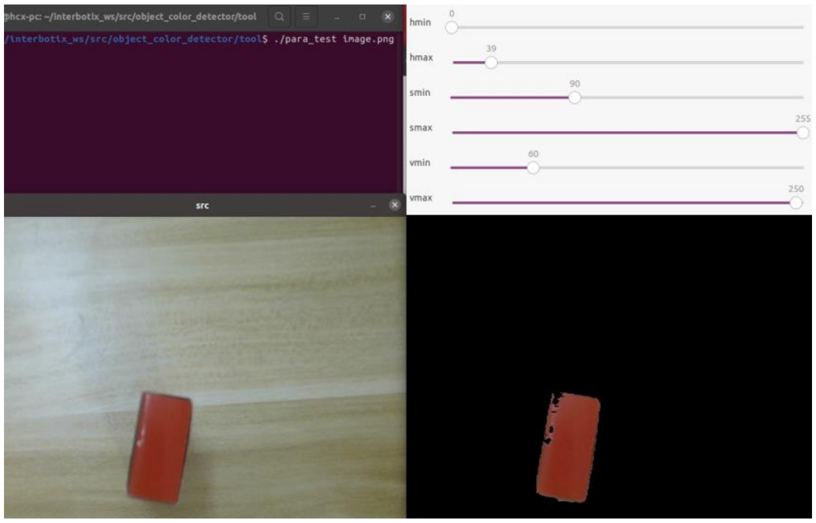


Fig. 6. Live camera image.

(4) Save the image containing the target object in ***rqt_image_view*** and place it under the path ***object_color_detector/tool***, and open the terminal in this path and open the parameter debugging tool with the following command, and enter the image name according to the actual name.

```
$. /para_test image.png
```

(5) Complete the adjustment of the ***hmin*** and ***hmax*** parameters of the target object color in the debugging tool to ensure that only targets of the same color will be recognized.

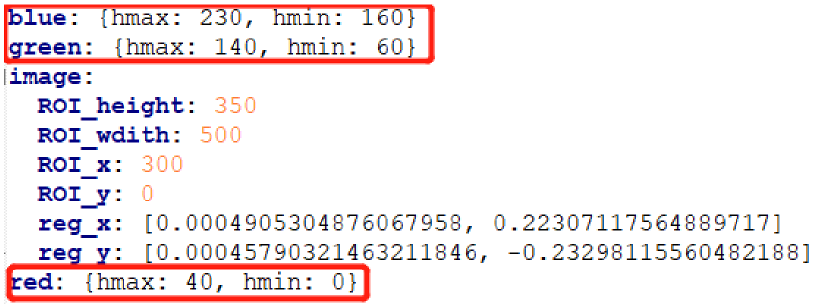


```
@hcx-pc: ~/interbotix_ws/src/object_color_detector/tool
/interbotix_ws/src/object_color_detector/tool$ ./para_test image.png
```

hmin 0
hmax 39
smin 90
smax 255
vmin 60
vmax 250

Fig. 7. Debugging parameters.

(6) Fill the above parameters into the configuration file ***object_color_detector/config/vision_config.yaml***, pay attention to the corresponding color, where the default provides red, blue, green color recognition parameters.



```
blue: {hmax: 230, hmin: 160}
green: {hmax: 140, hmin: 60}
image:
  ROI_height: 350
  ROI_wdith: 500
  ROI_x: 300
  ROI_y: 0
  reg_x: [0.0004905304876067958, 0.22307117564889717]
  reg_y: [0.00045790321463211846, -0.23298115560482188]
red: {hmax: 40, hmin: 0}
```

Fig. 8. Red, green and blue recognition parameters

(7) **Close the previous camera node**, reopen a new terminal and start the procedure for object color recognition using the following command for real camera.

```
$ roslaunch object_color_detector object_detect_hsv.launch
```

Or run the following command for Gazebo camera.

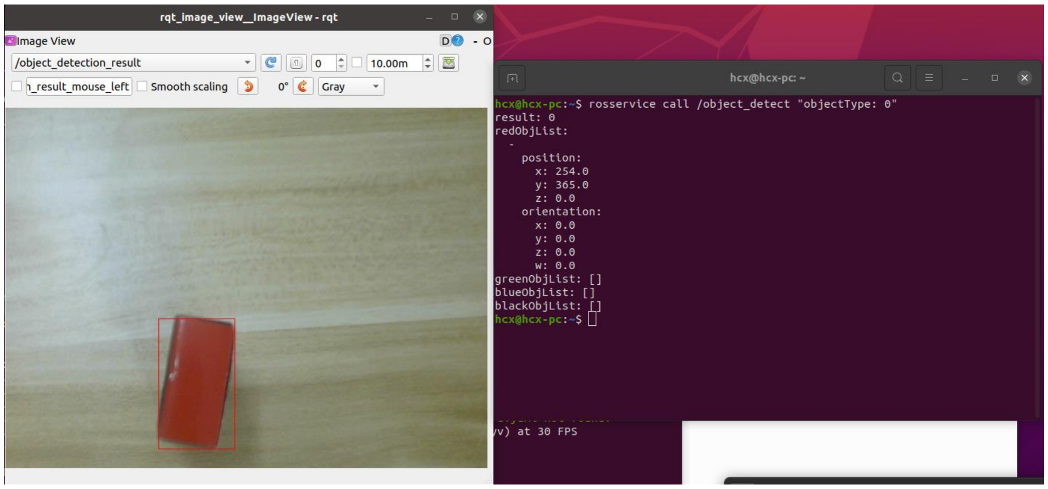
```
$ roslaunch object_color_detector object_detect_hsv.launch use_gazebo:=true
```

After running successfully, modify the ***rqt_image_view*** subscription to ***/object_detect_result*** topic to facilitate subsequent viewing of the dynamic image recognition results.

(8) Open a new terminal and invoke the image recognition function using the following command.

```
$ rosservice call /object_detect "objectType: 0"
```

(9) After successful image recognition, the recognition result will be displayed in the ***rqt_image_view***.



```
hcx@hcx-pc:~$ rosservice call /object_detect "objectType: 0"
result: 0
redObjList:
  -
    position:
      x: 254.0
      y: 365.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
greenObjList: []
blueObjList: []
blackObjList: []
hcx@hcx-pc:~$
```

```
v) at 30 FPS
```

Fig. 9. Recognition result display.

(10) If the image is not recognized successfully, please turn off the program and follow steps (1) to (9) again until the recognition is successful.

1.4 Robotic arm eye-in-hand calibration

The robotic arm automatic calibration algorithm is trained using linear regression with five data, which enables the transformation from the pixel points of the image recognition results to the robot spatial coordinate system.

```
$ sudo apt install python3-sklearn
```

The procedure of the visual calibration is as follows.

(1) Connect the camera to the computer and ensure that the adjustment of the image recognition parameters has been completed.

(2) Start the real robot arm.

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s use_actual:=true dof:=6
```

Or the Gazebo simulation.

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s \
use_gazebo:=true dof:=6 world_name:=warehouse.world
```

Restart a terminal and use the following command to start the visual recognition function for the real robot.

```
$ roslaunch object_color_detector object_detect_hsv.launch
```

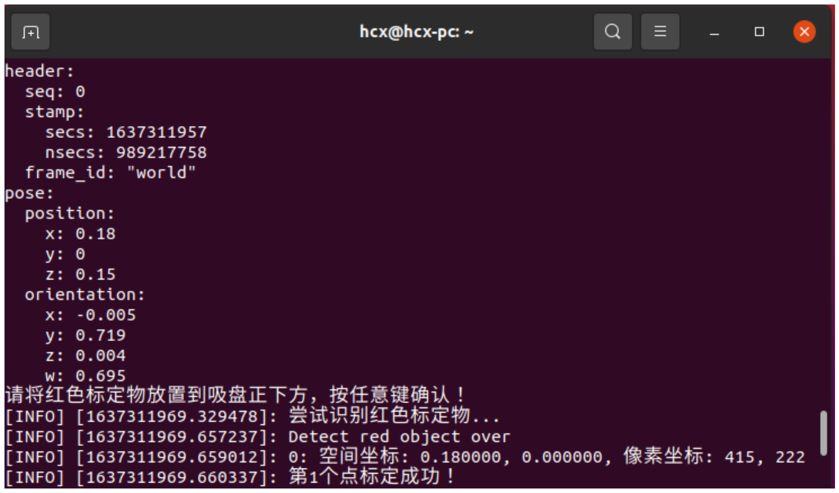
Or for the Gazebo simulation.

```
$ roslaunch object_color_detector object_detect_hsv.launch use_gazebo:=true
```

(3) Reopen a terminal and use the following command to start the automatic visual calibration function, **paying attention to the movement of the robot to avoid accidental injury**.

```
$ roslaunch object_color_detector camera_calibration_hsv.launch __ns:=wx250s
```

(4) After calibration is initiated, the robot will move to the designated position, follow the prompts in the terminal, place the red calibration object directly below the robot's gripper, and then click Enter in the terminal to calibrate the next point.



```
header:
  seq: 0
  stamp:
    secs: 1637311957
    nsecs: 989217758
  frame_id: "world"
pose:
  position:
    x: 0.18
    y: 0
    z: 0.15
  orientation:
    x: -0.005
    y: 0.719
    z: 0.004
    w: 0.695
请将红色标定物放置到吸盘正下方，按任意键确认！
[INFO] [1637311969.329478]: 尝试识别红色标定物...
[INFO] [1637311969.657237]: Detect red object over
[INFO] [1637311969.659012]: 0: 空间坐标: 0.180000, 0.000000, 像素坐标: 415, 222
[INFO] [1637311969.660337]: 第1个点标定成功！
```

Fig. 10. Calibration process.

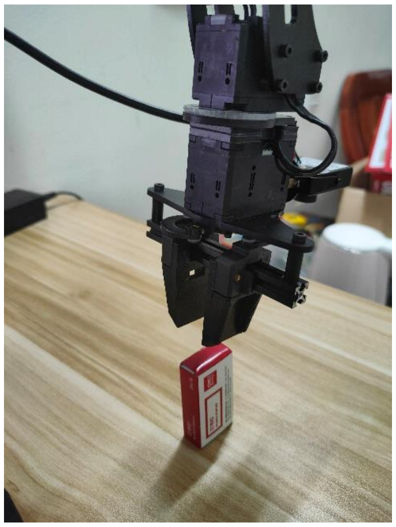
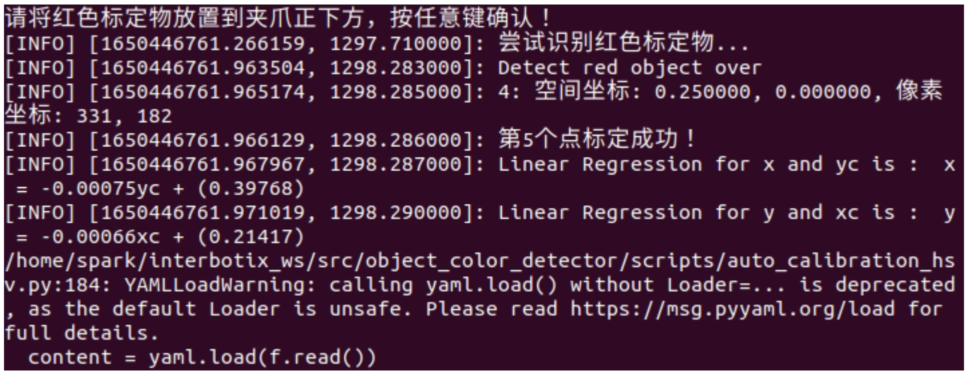


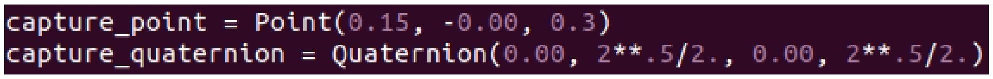
Fig. 11. Place the red calibration object.

(5) After all points are calibrated, the parameters are automatically saved to the configuration file. When the end of robot arm is located at the calibration pose (defined at ***object_color_detector/scripts/auto_calibration_hsv.py*** line: 39), the estimated position (x, y) of objects can be calculated through $x = k_1 yc + b_1, y = k_2 xc + b_2$, where yc, xc are the pixel coordinate of the object in image and k_1, k_2, b_1, b_2 are linear regression results.



```
请将红色标定物放置到夹爪正下方，按任意键确认！
[INFO] [1650446761.266159, 1297.710000]: 尝试识别红色标定物...
[INFO] [1650446761.963504, 1298.283000]: Detect red object over
[INFO] [1650446761.965174, 1298.285000]: 4: 空间坐标: 0.250000, 0.000000, 像素
坐标: 331, 182
[INFO] [1650446761.966129, 1298.286000]: 第5个点标定成功！
[INFO] [1650446761.967967, 1298.287000]: Linear Regression for x and yc is :  x
 = -0.00075yc + (0.39768)
[INFO] [1650446761.971019, 1298.290000]: Linear Regression for y and xc is :  y
 = -0.00066xc + (0.21417)
/home/spark/interbotix_ws/src/object_color_detector/scripts/auto_calibration_hs
v.py:184: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated
, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for
full details.
  content = yaml.load(f.read())
```

Fig. 12. Calibration completed.



```
capture_point = Point(0.15, -0.00, 0.3)
capture_quaternion = Quaternion(0.00, 2**.5/2., 0.00, 2**.5/2.)
```

Fig. 13. Calibration pose.

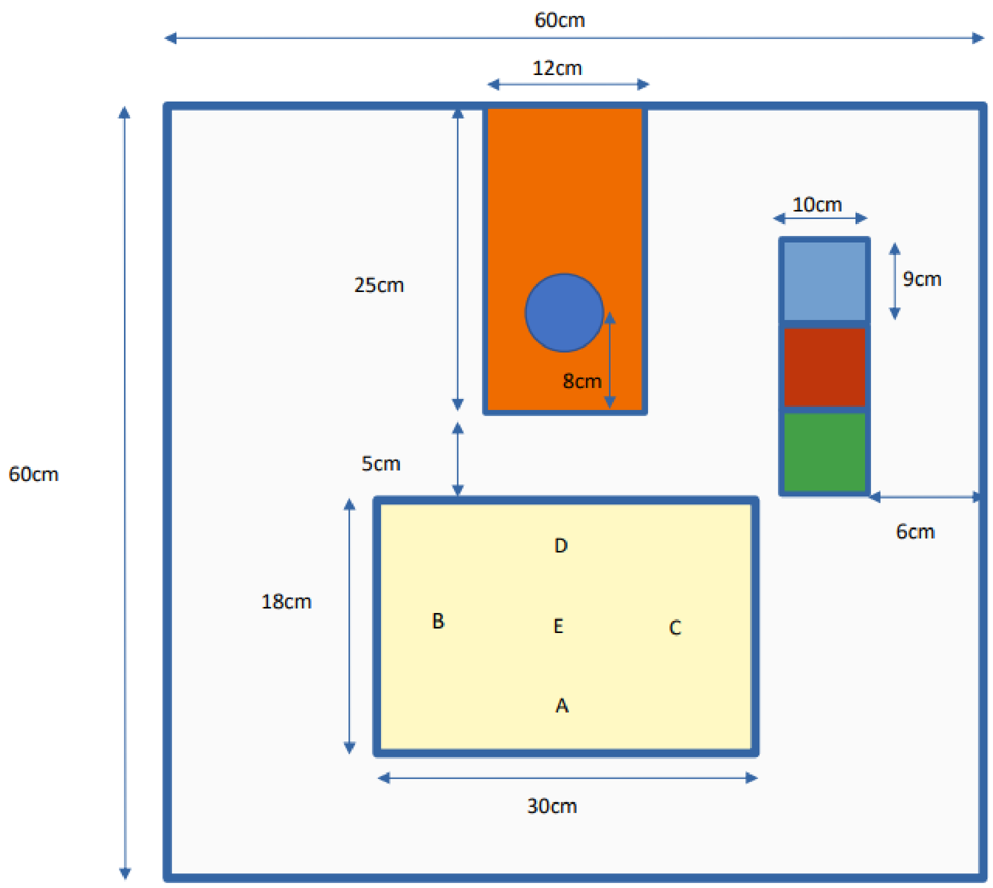


Fig. 14. Intelligent Sorting System.

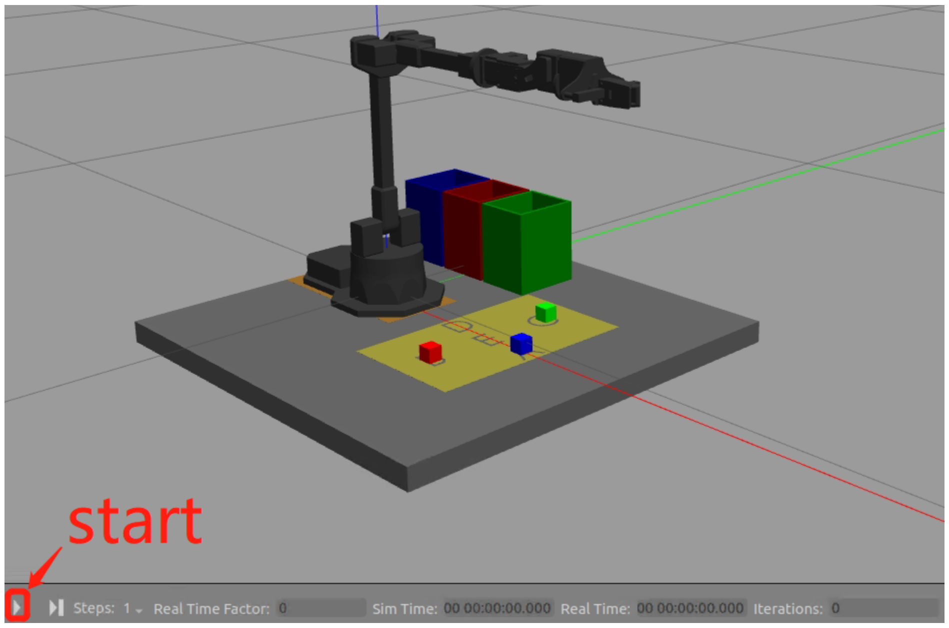


Fig. 15. Gazebo Simulation Environment.

Tasks:

1. The intelligent sorting system is shown in Fig. 14. above, with an overall area of 60cm×60cm square, where the orange part is the robot arm of size 12cm×25cm; the yellow part is the object recognition area of size 18cm×30cm; the blue area, red area and green area are the placement areas of size 9cm×10cm. the grasping objects are small sponge cubes with blue, red, or green color, the length of the cube is 2cm.
2. Firstly, three cubes of different colors (blue, red, and green) are placed in the recognition area, respectively at point A point B and C. After starting visual recognition, the recognized square position is obtained and the robot arm pick the cubes within the field of view.
3. Sort and place the cubes according to their color in the placement area.
4. After successfully picking up the three cubes, manually place the blue, red, and green cube in turn to the E point, the robot arm can again recognize the cube and successfully move the cube to the designated area.
5. Manually place the blue, red, and green cube cubes at point B, point C and point D respectively at once, and finish the task above again.

Hints:

You may first follow the procedure below to check if the provided codes can progress well.

1. Download and build the source code.
2. Start the MoveIt group with the real robot driver.

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s
use_actual:=true dof:=6
```

Or the Gazebo simulation.

```
$ roslaunch interbotix_moveit interbotix_moveit.launch robot_name:=wx250s \

use_gazebo:=true dof:=6 world_name:=warehouse.world
```

Remember to click the start button in Gazebo as shown in Fig. 15.

3. Start the usb camera driver and the object color detection program.

```
$ roslaunch object_color_detector object_detect_hsv.launch
```

Or only the object color detection program that subscribe to Gazebo camera.

```
$ roslaunch object_color_detector object_detect_hsv.launch use_gazebo:=true
```

4. Try to find the ROS interface of object detection (you can refer to ***object_color_detector/src/main_object_detector.cpp***) and how to calculate the object position from the detection results (1.4-5). Write your program for the vision sorting system.
5. Please refer to 1.3 or 1.4 if the object detection parameters and the eye-in-hand calibration parameter is not accurate enough.
6. The pick and place process of robot arm can be referred to Lab 6.
7. The table, the placement areas, and the cubes can be regarded as inflated obstacles during the robot arm motion planning, so that the robot arm will not collide with these objects.
8. In Gazebo simulation, you can randomly reset the position of cubes with the following command for convenience.

```
$ rosrun interbotix_gazebo reset_cube_position.py
```

Submission:

[Report] In the report, draw the flowchart of the robot's behavior, and explain the nodes and messages used in your code.

[Code] Provide sufficient files and codes with adequate comments that we can replicate your result.

[Demo Video] Provide a video to show the delivery performance of the robot.

[Deadline] 9:00 am, May 26, 2022

Gradings:

- Report, Code, and Demo Video [50%]
- New methods [10%]: Bonus will be given if any effective library have been used which is not covered in the Labs.
- Presentation [20%]: Make a 5 minutes presentation in the class to explain how you achieve the task. (May 26, 2022 10:30am-11:30am)
- Competition [20%]: To show the effectiveness and the adaptiveness of your methods. We will reimplement your methods in the same environment but place the cube randomly in the object recognition area. Under same maximum joint speed and acceleration, the faster the task is completed, the higher the score. (May 26, 2022 11:30am-12:30pm)