Program Analysis for Color Tracking

1. File Path

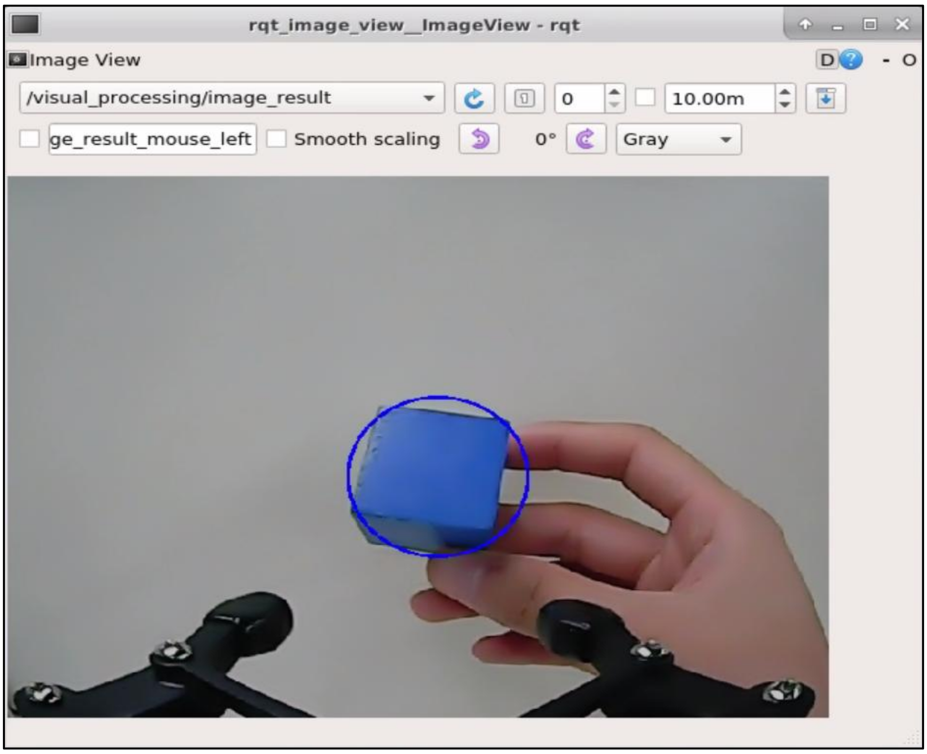
The program file is stored in:

/home/ubuntu/armpi_pro/src/visual_processing/scripts/visual_processing_node.py (Image processing)

/home/ubuntu/armpi_pro/src/color_tracking/scripts/color_tracking_node.py （tacking control）

2. Program Performance

After the program is started, move the blue tube within the camera's view range. It can be viewed that the recognized target color is framed in rqt tool. At this time, slowly move the the tube with your hand, and then robotic arm will move with the target color.



3. Program Analysis

Note: please back up the initial program before making any modifications. It is prohibited editing the source code files directly to prevent making changes in an incorrect manner that could lead to robot malfunctions, rendering them irreparable.

3.1 Import Parameter Module

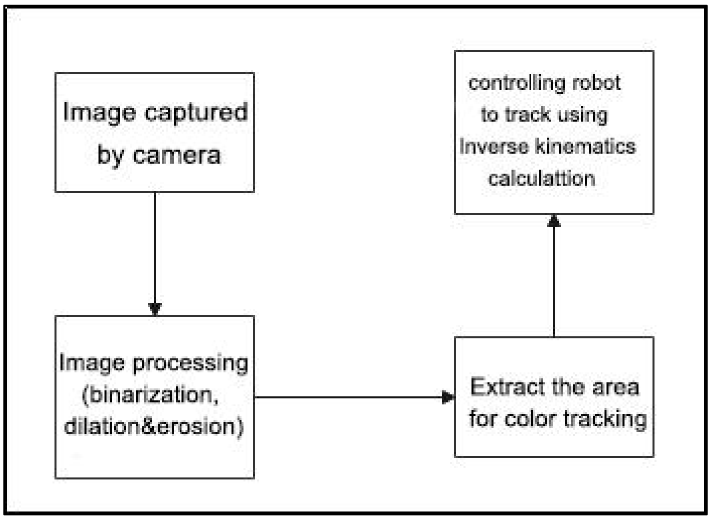
Imported Module	Function
import sys	The sys module of Python is imported to access to system-related functionalities and variables.
import cv2	The OpenCV library of Python is imported to perform image processing and computer vision-related functions.
import time	The time module of Python is imported to perform time-related functionalities, such as delay operations.
import math	The math module of Python is imported to perform mathematical operations and functions.
import rospy	The Python library rosy is imported for communication and interaction with ROS.
import numpy as np	The NumPy library is imported and is renamed as np for performing array and matrix operations.

from armpi_pro import Misc	The Misc module is imported from arm_pi_pro package to handle the recognized rectangular data.
from armpi_pro import apriltag	The apriltag module is imported from arm_pi_pro package to perform Apriltag recognition and processing.
from threading import RLock, Timer	The “RLock” class and “Timer” class is imported from the threading module of Python for thread-related operations.
from std_srvs.srv import *	All service message types are imported from the std_srvs in ROS for defining and using standard service messages.
from std_msgs.msg import *	All message types are imported form the std_msgs package in ROS for defining and using standard messages.
from sensor_msgs.msg import Image	The image message type is imported from the sensor_msgs packages for processing image data.
from visual_processing.msg import Result	The Result message type is imported from the visual_processing package for the message of image processing results.
from visual_processing.srv import SetParam	The SetParam service type is imported from the visual_processing packages for using customs service related to parameter

	settings.
from sensor.msg import Led	The Led message type is imported form the sensor.msg module for controlling or representing the LED status on a sensor.
from chassis_control.msg import *	All message types are imported from the chassis_control.msg module, which indicated that all message types defined in this module is imported to perform the chassis control.
from visual_patrol.srv import SetTarget	The SetTarget service type is imported from the visual_patrol.srv module is used to set a target for line following.
from hiwonder_servo_msgs.msg import MultiRawIdPosDur	The MultiRawIdPosDur message type is imported from the hiwonder_servo_msgs.msg module for controlling servos.
from armpi_pro import PID	The PID class is imported from the armpi_pro module to perform PID algorithm.
from armpi_pro import bus_servo_control	The bus_servo_control module is imported from the armpi_pro module, including the functions and methods related to the servo control.
from kinematics import ik_transform	The ik_transform function is imported from

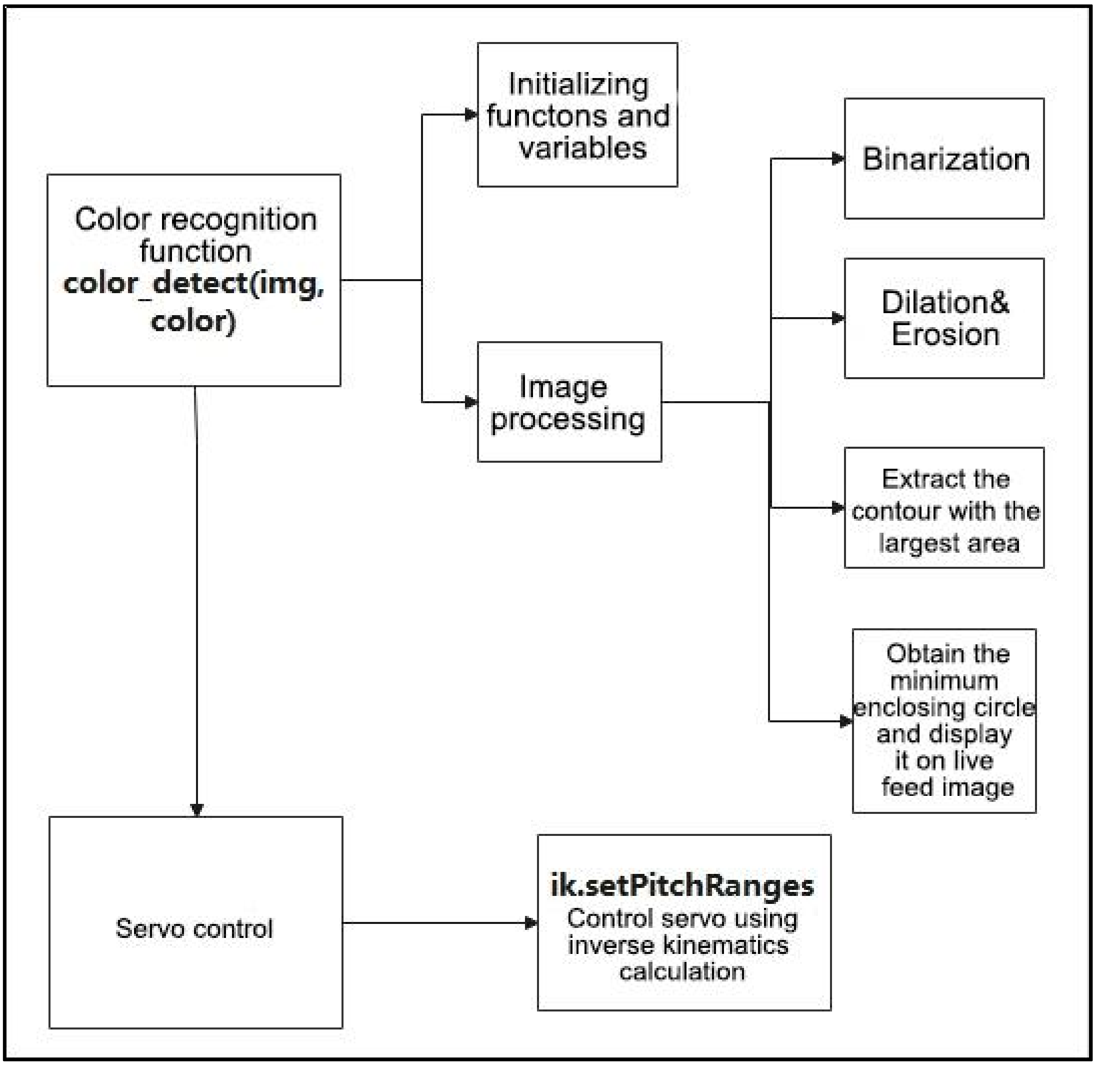
	the kinematics module to perform conversion of inverse kinematics.

3.2 Program Logic



Obtaining the image information through the camera, and then perform image processing, e.i, binarization. To reduce interference and create smoother images, erosion and dilation processes are applied. Then, the largest area contour and the minimum enclosing rectangle of the target are extracted to obtain an area for color tracking. Lastly, the angle of the robotic arm is calculated using inverse kinematics, controlling servo to perform color tracking.

3.3 Code Analysis



From the above flow diagram, the program is mainly used for color recognition and servo control. The following content is analyzed based on the above flow diagram.

3.3.1 Image Processing

Initializing functions and variables

```
# 单颜色识别函数
def color_detect(img, color):
    global pub_time
    global publish_en
    global color_range_list

    if color == 'None':
        return img

    msg = Result()
    area_max = 0
    area_max_contour = 0
    img_copy = img.copy()
    img_h, img_w = img.shape[:2]
    frame_resize = cv2.resize(img_copy, size_m, interpolation=cv2.INTER_NEAREST)
    frame_lab = cv2.cvtColor(frame_resize, cv2.COLOR_BGR2LAB)  # 将图像转换到LAB空间
```

Binarization

Using the inRange() function from the cv2 library to perform binarization on image.

```
        frame_mask = cv2.inRange(frame_lab, tuple(color_range['min']),
        tuple(color_range['max']))  # 对原图像和掩模进行位运算
```

The first parameter “frame_lab” is the input image.

The second parameter “tuple(color_range['min'])” is the lower limit of threshold.

The third parameter “tuple(color_range['max'])” is the upper lower of threshold.

Dilation and Erosion

To reduce interference and create smoother images, erosion and dilation processes are applied

```
        eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.
        MORPH_RECT, (2, 2)))          # 腐蚀
        dilated = cv2.dilate(eroded, cv2.getStructuringElement(cv2.
        MORPH_RECT, (2, 2)))            # 膨胀
```

erode() function is applied to erode image. Here uses an example of the code “eroded = cv2.erode(frame_mask, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))”. The meaning of parameters in parentheses are as follow:

The first parameter “frame_mask” is the input image.

The second parameter “cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))” is the structural elements and kernel that determines the nature of operation. The first parameter in parentheses is the shape of kernel and the second parameter is the size of kernel.

dilate() function is applied to dilate image. The meaning of parameters in parentheses is the same as the parameters of erode() function.

Obtain the contour of the maximum area

After processing the above image, obtain the contour of the recognition target. The findContours() function in cv2 library is involved in this process.

```
        contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]  # 找出轮廓
```

The erode() function is applied to erode. Take code “contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]” as example.

The first parameter “dilated” is the input image.

The second parameter “cv2.RETR_EXTERNAL” is the contour retrieval mode.

The third parameter “cv2.CHAIN_APPROX_NONE)[-2]” is the approximate method of contour.

Find the maximum contour from the obtained contours. To avoid interference, set a minimum value. Only when the area is greater than this minimum value, the target contour will take effect. The minimum value here is “50”.

```
        area_max_contour, area_max = getAreaMaxContour(contours)  # 找出最大轮廓

        if area_max > 200:  # 有找到最大面积
```

Obtain the minimum enclosing circle and display it on the live feed image

Using the minEnclosingCircle() function from the cv2 library to obtain the minimum bounding circle and its center coordinates for the target contour.

Then, utilize the circle() function to display the enclosing circle on the live feed image.

```
            (centerx, centery), radius = cv2.minEnclosingCircle(
            area_max_contour)  # 获取最小外接圆
            msg.center_x = int(Misc.map(centerx, 0, size_m[0], 0, img_w))
            msg.center_y = int(Misc.map(centery, 0, size_m[1], 0, img_h))
            msg.data = int(Misc.map(radius, 0, size_m[0], 0, img_w))
            cv2.circle(img, (msg.center_x, msg.center_y), msg.data+5,
            range_rgb[color], 2)
            publish_en = True
```

3.3.2 Tracking Control

Take the center coordinates X and Y as the set value.

Perform an inverse kinematic calculation using the X and Y coordinates of the image center as set values and the X and Y coordinates of the currently detected target as input values to determine the target position.

```
                # z轴追踪
                z_pid.SetPoint = img_h / 2.0  # 设定
                z_pid.update(center_y)        # 当前
                dy = z_pid.output             # 输出
                z_dis += dy

                z_dis = 0.22 if z_dis > 0.22 else z_dis
                z_dis = 0.17 if z_dis < 0.17 else z_dis

                target = ik.setPitchRanges((0, round(y_dis, 4), round(
                z_dis, 4)), -90, -85, -95) # 逆运动学求解
                if target:
                    # 发布舵机控制节点消息,移动机械臂
                    servo_data = target[1]
                    bus_servo_control.set_servos(joints_pub, 20, (
                        (3, servo_data['servo3']), (4, servo_data[
                        'servo4']), (5, servo_data['servo5']), (6, x_dis
                        )))
```

The inverse kinematics takes an example of the code “ik.setPitchRanges((0, round(y_dis, 4), round(z_dis, 4)), -90, -85, -95)”, where the meaning of the parameters in parentheses are as follow:

In the first parameter “(0, round(y_dis, 4), round(z_dis, 4)”, “0” represents the position on x axis; “round(y_dis, 4)” represents the position on Y axis; “round(z_dis, 4)” represents the position on Z axis.

The second parameter “**-90**” represents the pitch angle.

The third parameter “-80” represents the range of the pitch angle.

The fourth parameter “-90” represents the range of the pitch angle.

The servo control takes an example of “bus_servo_control.set_servos(joints_pub, 20, ((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)))”, where the meaning of the parameters in parentheses are as follow:

The first parameter, "joints_pub," publishes messages to control the servo.

The second parameter "20" represents the runtime duration.

The third parameter, "((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis))," consists of tuples where:

"3" is the servo number.

"servo_data['servo3']" is the angle of the servo.

Similarly, "(4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)" follow the same pattern.