Program Analysis

1. File Path

The program file is stored in:

/home/ubuntu/armpi_pro/src/visual_processing/scripts/visual_processing_node.py

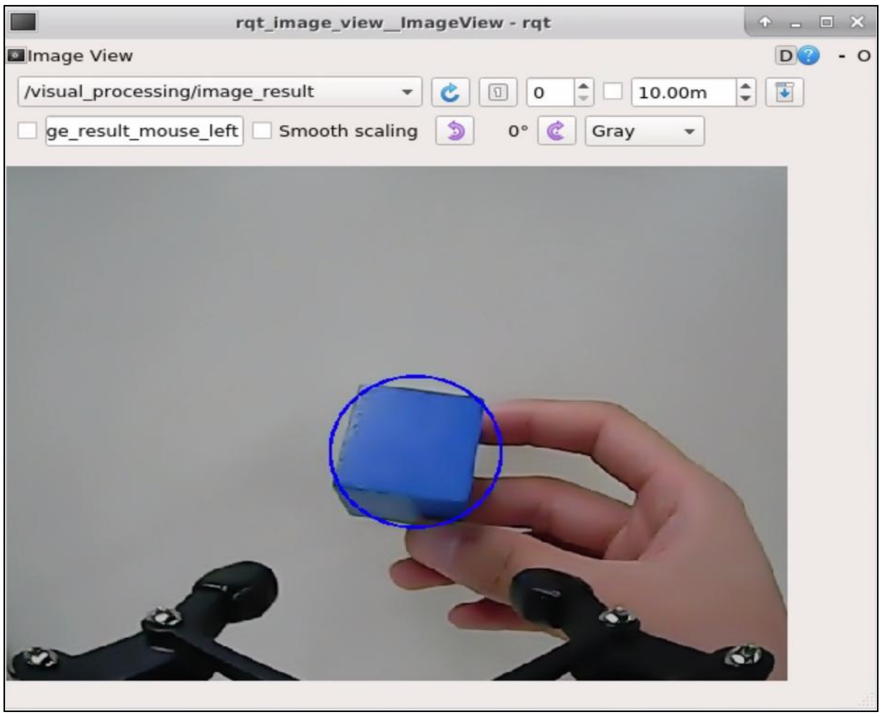
（image processing）

/home/ubuntu/armpi_pro/src/object_tracking/scripts/object_tracking_node.py

（motion control）

2. Program Performance

After the game starts, place the blue block within the detected range of camera. The target color will be framed with a circle in rqt tool once the target is detected. At this point, move the block slowly, and then robotic arm will rotate towards the direction of the block, and the car will move towards the block.


rqt_image_view__ImageView - rqt
Image View
/visual_processing/image_result
0
10.00m
ge_result_mouse_left
Smooth scaling
0°
Gray

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

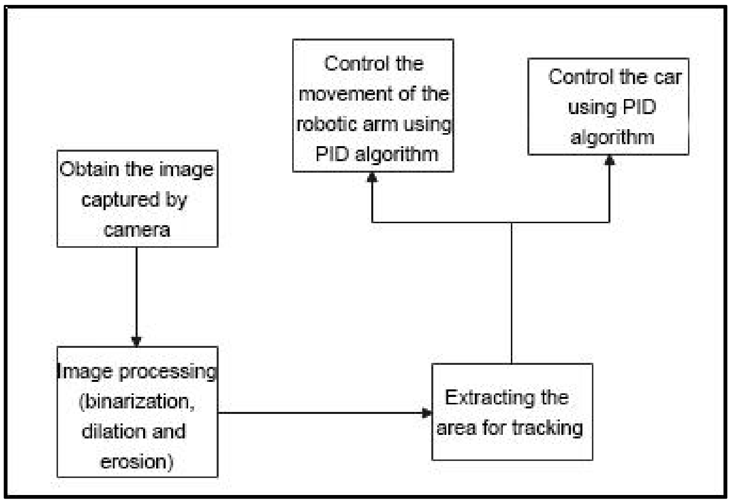
Imported Module	Function
import sys	The sys module of Python is imported to access to system-related functionalities and variables.
import cv2	The OpenCV library of Python is imported to perform image processing and computer vision-related functions.
import time	The time module of Python is imported 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

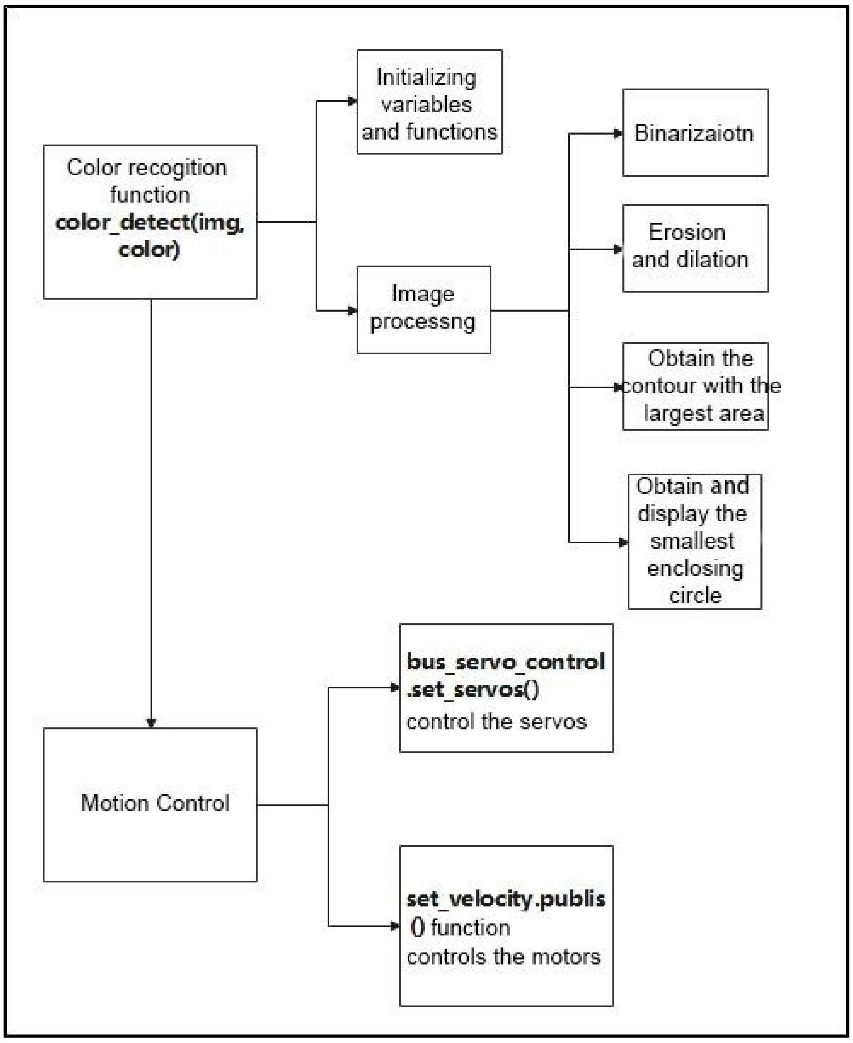
	the kinematics module to perform conversion of inverse kinematics.

3.2 Program Logic


Obtain the image captured by camera
Image processing (binarization, dilation and erosion)
Extracting the area for tracking
Control the movement of the robotic arm using PID algorithm
Control the car using PID algorithm

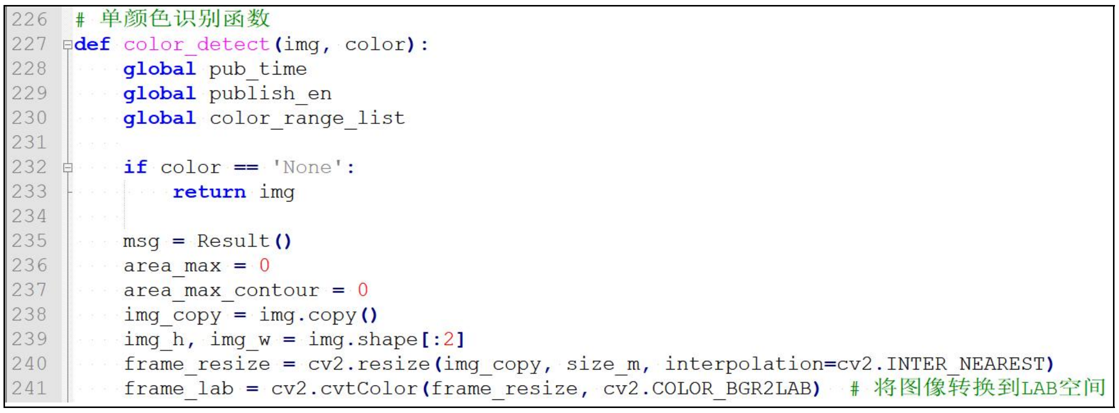
Obtain image information through the camera, and then perform binarization on image. To reduce the interference and get a smooth image, it is necessary to perform dilation and erosion on the image. Next, obtain the contour with a largest area and the minimum circumscribed circle for the target to get an area for tracking. Then control the robotic arm to rotate to the target using PID algorithm, and the vehicle will move towards the target.

3.3 Code Analysis


Color recogition function
color_detect(img, color)
Initializing variables and functions
Image processng
Binarizaiotn
Erosion and dilation
Obtain the contour with the largest area
Obtain and display the smallest enclosing circle
Motion Control
bus_servo_control .set_servos()
control the servos
set_velocity.publis () function
controls the motors

From the above flow diagram, the program is mainly used for the color recognition and the motion control.

3.3.1 Image Processing



```
# 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 operation 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 make a smoother image, it is necessary to perform dilation and erosion operations on image.

```
            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 takes 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 with the largest area

After processing the above image, it is necessary to obtain the contour of the target. The findContours() function from the 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. Here uses an example of the code "**contours = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[-2]**".

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.

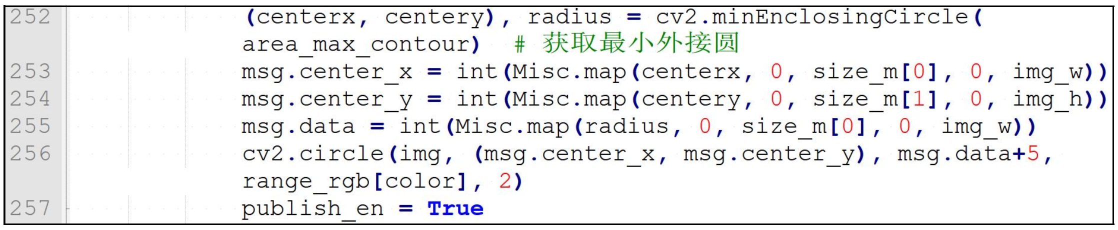
Find the maximum contour from the obtained contours. To avoid interference, set a minimum value. Only when the area is greater than this minimum value, the target contour will take effect. The minimum value here is "50".

```
        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 on the live feed image

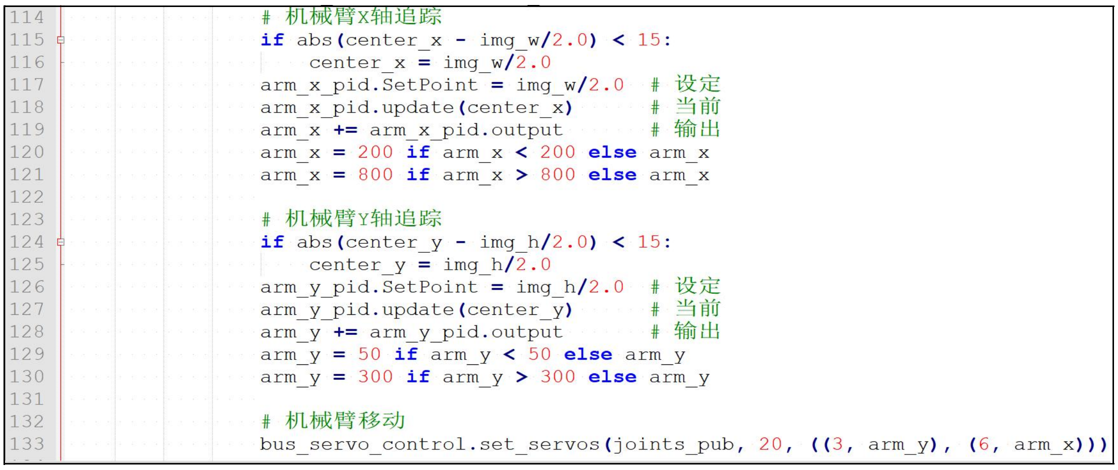
The **minEnclosingCircle()** function from the cv2 library is utilized to obtain the minimum enclosing circle and the coordinates of its center for the target contour. The obtained circle is then displayed in the feedback image using the circle() function.



```python
            (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 Motion Control

By invoking the bus_servo_control.set_servos() function to control the servos on robotic arm to allow the robotic arm to move with the target within the recognition range.



```python
                # 机械臂X轴追踪
                if abs(center_x - img_w/2.0) < 15:
                    center_x = img_w/2.0
                arm_x_pid.SetPoint = img_w/2.0  # 设定
                arm_x_pid.update(center_x)      # 当前
                arm_x += arm_x_pid.output       # 输出
                arm_x = 200 if arm_x < 200 else arm_x
                arm_x = 800 if arm_x > 800 else arm_x

                # 机械臂Y轴追踪
                if abs(center_y - img_h/2.0) < 15:
                    center_y = img_h/2.0
                arm_y_pid.SetPoint = img_h/2.0  # 设定
                arm_y_pid.update(center_y)      # 当前
                arm_y += arm_y_pid.output       # 输出
                arm_y = 50 if arm_y < 50 else arm_y
                arm_y = 300 if arm_y > 300 else arm_y

                # 机械臂移动
                bus_servo_control.set_servos(joints_pub, 20, ((3, arm_y), (6, arm_x)))
```

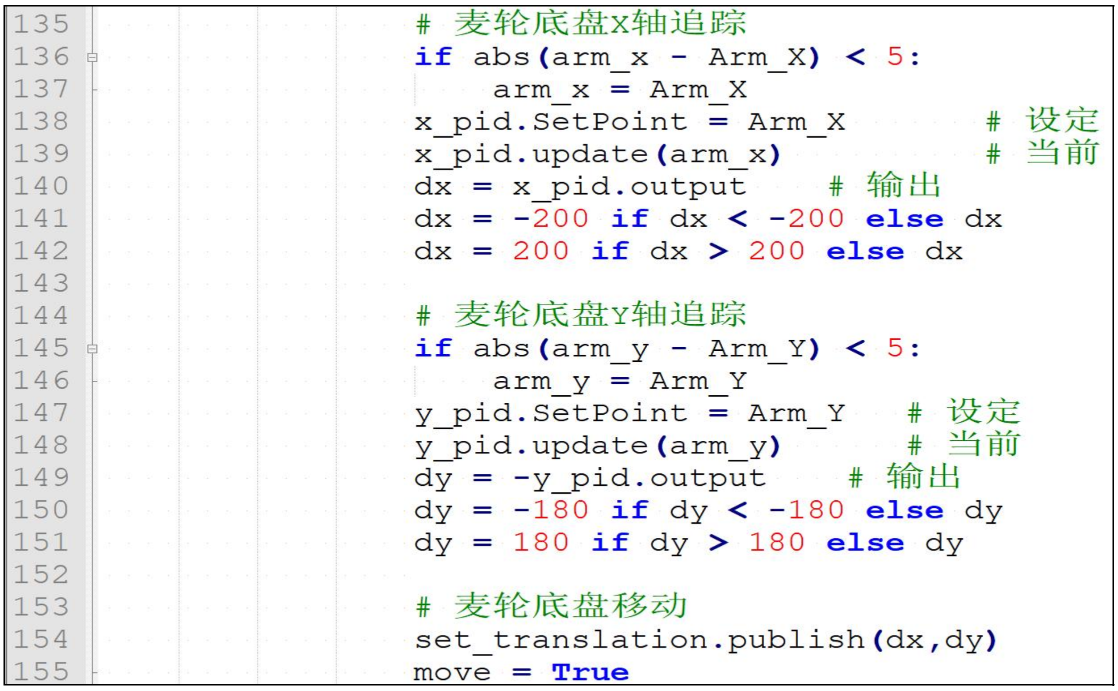
Take the code "**bus_servo_control.set_servos(joints_pub, 20, ((3, arm_y), (6, arm_x)))**" as example and the meaning of parameters in parentheses are as follow:

The first parameter "**joints_pub**" is to publish the message of the servo control node.

The second parameter "**20**" is the running time.

The third parameter is "**((3, arm_y), (6, arm_x)**". "**3**" is the servo number, "**arm_y']**" is the servo angle.

Lastly, by invoking the **set_velocity.publis()** function, the motors on ArmPi Pro is controlled to drive the mecanum wheels to achieve the performance of tracking.



```
                # 麦轮底盘x轴追踪
                if abs(arm_x - Arm_X) < 5:
                    arm_x = Arm_X
                x_pid.SetPoint = Arm_X       # 设定
                x_pid.update(arm_x)          # 当前
                dx = x_pid.output    # 输出
                dx = -200 if dx < -200 else dx
                dx = 200 if dx > 200 else dx

                # 麦轮底盘Y轴追踪
                if abs(arm_y - Arm_Y) < 5:
                    arm_y = Arm_Y
                y_pid.SetPoint = Arm_Y   # 设定
                y_pid.update(arm_y)      # 当前
                dy = -y_pid.output    # 输出
                dy = -180 if dy < -180 else dy
                dy = 180 if dy > 180 else dy

                # 麦轮底盘移动
                set_translation.publish(dx,dy)
                move = True
```

Here will use a example of the motor control “**set_translation.publish(dx,dy)**”, and the meaning of parameters in parentheses are as follow:

The first parameter “**dx**” is the movement distance of car in x-axis. Its range is from -200 to 200.

The second parameter “**dy**” is the movement distance of car in y-axis. Its range is from -180 to 180.