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/visual_processing/scripts/intelligent_transport_node.py

（Function achievement）

2. Program Performance

After the game starts, hold the block within the detected range of camera. When the block is recognized by ArmPi Pro, it will grip it and keep following line. Then place the block with different colors to the corresponding position.

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

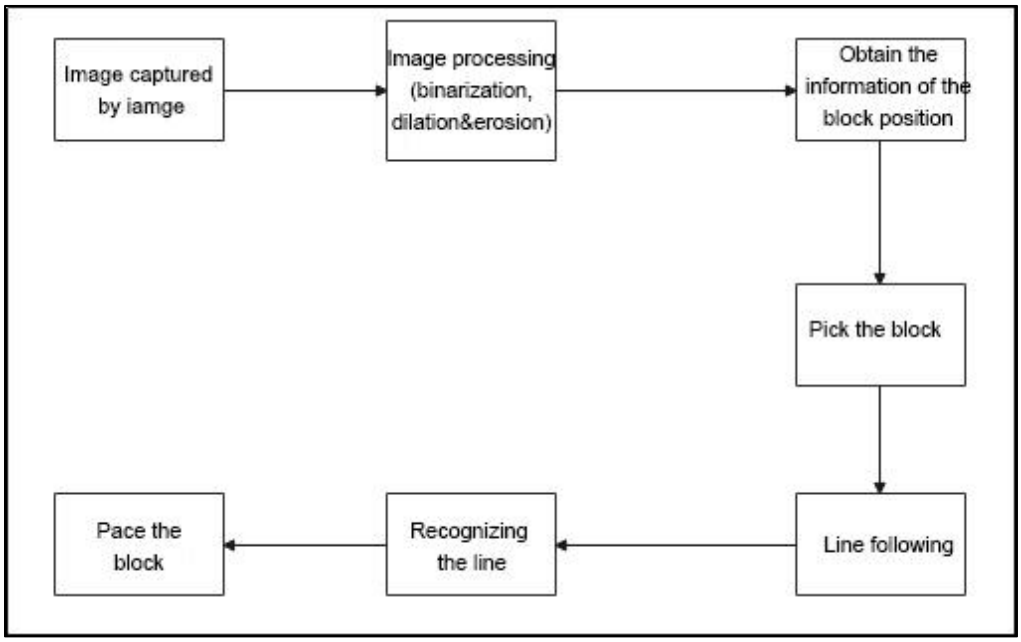
Imported Module	Function
import sys	The sys module of Python is imported to

	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

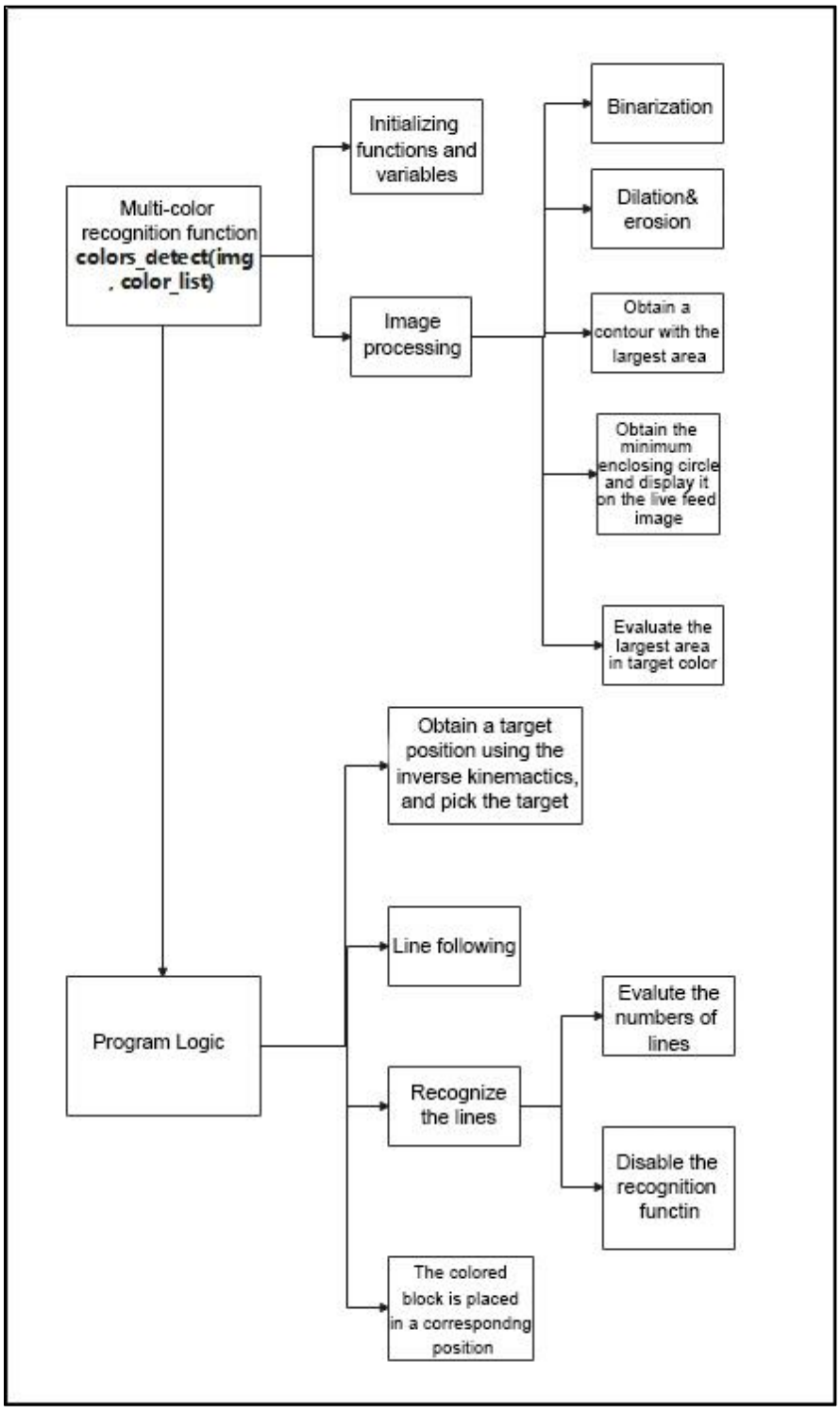
	chassis control.
from visual_patrol.srv import SetTarget	The SetTarget service type is imported from the visual_patrol.srv module is used to set a target for line following.
from hiwonder_servo_msgs.msg import MultiRawIdPosDur	The MultiRawIdPosDur message type is imported from the hiwonder_servo_msgs.msg 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



By capturing image information through a camera, followed by image processing. To mitigate interference and enhance image quality, the image is further subjected to erosion and dilation operations. Subsequently, the position of the colored block is determined, and based on this information, the block is grasped. Using a pre-arranged map, a line-following procedure is executed. Upon identifying a corresponding horizontal line based on the color of the grasped block, the motion halts, and the block is placed at a designated location.

3.3 Code Analysis



From the above diagram flow, the program is mainly used for multi-colors recognition and function realization.

3.3.1 Image Processing

Initializing Functions and Variables

```
# 多颜色识别函数
def colors_detect(img, color_list):
    global pub_time
    global publish_en
    global color_range_list

    if color_list == 'RGB' or color_list == 'rgb':
        color_list = ('red','green','blue')
    else:
        return img

    msg = Result()
    msg.data = 0
    color_num = 0
    max_area = 0
    color_area_max = None
    areaMaxContour_max = 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

Use 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
```

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 “**frame_mask**” is the input image.

The second parameter “**cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))**” 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.

```
299                 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”.

```
300             areaMaxContour, area_max = getAreaMaxContour(contours)
                                                    # 找出最大轮廓
301             if areaMaxContour is not None:
302                 if area_max > max_area:#找最大面积
```

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.

```
308             (centerx, centery), radius = cv2.minEnclosingCircle(
                areaMaxContour_max)  # 获取最小外接圆
309             msg.center_x = int(Misc.map(centerx, 0, size_m[0], 0, img_w))
310             msg.center_y = int(Misc.map(centery, 0, size_m[1], 0, img_h))
311             radius = int(Misc.map(radius, 0, size_m[0], 0, img_w))
312             cv2.circle(img, (msg.center_x, msg.center_y), radius+5, range_rgb
                [color_area_max], 2)
```

Determine the color block with the largest area.

```
314             if color_area_max == 'red':  #红色最大
315                 msg.data = 1
316             elif color_area_max == 'green':  #绿色最大
317                 msg.data = 2
318             elif color_area_max == 'blue':  #蓝色最大
319                 msg.data = 3
```

3.3.2 Function Realization

Grip the block

The position of the target on x, y and z axes is obtained after processing image. Then the target position is calculated using inverse kinematics. Lastly, robot will perform the gripping action.

```
319                 # 机械臂追踪移动到色块上方
320                 target = ik.setPitchRanges((0, round(y_dis, 4
                    ), 0.03), -180, -180, 0)
321                 if target:
322                     servo_data = target[1]
323                     bus_servo_control.set_servos(joints_pub,
                        20,((3, servo_data['servo3']),
324                          (4, servo_data['servo4']),(5,
                           servo_data['servo5']), (6, x_dis)))
325
326                 if dx < 2 and dy < 0.003 and not stable: #
                    等待机械臂稳定停在色块上方
327                     num += 1
328                     if num == 10:
329                         stable = True  # 设置可以夹取
330                         num = 0
331                 else:
332                     num = 0
333
334                 if stable: #控制机械臂进行夹取
335                     offset_y = Misc.map(target[2], -180, -150,
                         -0.03, 0.03)
336                     set_rgb(detect_color)          # 设置rgb灯颜色
337                     target_color = detect_color # 暂存目标颜色
```

The inverse kinematics takes “**ik.setPitchRanges((0, round(y_dis + offset_y, 4), -0.08), -180, -180, 0)**” as example and the meaning of parameters in parentheses are as follow:

The first parameter is “**(0, round(y_dis + offset_y, 4)**”. “**0**” is the position of the target on x-axis. “**round(y_dis, 4)**” is the position of the target on y-axis. “**round(z_dis, 4)**” is the position of the target on z-axis.

The second parameter “**-180**” is the angle of x-axis.

The third parameter “**-180**” is the range of the pitch angle.

The fourth parameter “**0**” is the range of pitch angle.

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

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

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

The third parameter is "**((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']), (6, x_dis)**". Among them, "**3**" is the servo number. "**servo_data['servo3']**" and the rest of parameters are the servo angle.

Line Following

After gripping the block, the car will follow the line. Firstly, judge if there is the line within the detected range. The code is shown in the following figure:

```
206                elif line_width > 0:
```

Then the current x-coordinate of line subtracts the value of ideal center point. Get the yaw rate by PID mapping to adjust the speed of motor.

```
208                if abs(line_center_x - img_w/2) < 30:
209                    line_center_x = img_w/2
210                line_x_pid.SetPoint = img_w/2        # 设定
211                line_x_pid.update(line_center_x)     # 当前
212                dx = round(line_x_pid.output, 2)     # 输出
213                dx = 0.8 if dx > 0.8 else dx
214                dx = -0.8 if dx < -0.8 else dx
215
216                set_velocity.publish(100, 90, dx) # 控制底盘
217                chassis_move = True
```

Take the code "**set_velocity.publish(100, 90, dx)**" as example:

The first parameter "**100**" is the linear velocity.

The second parameter "**90**" is the angular velocity.

The third parameter "**dx**" is the yaw rate. The larger the yaw rate, the faster the rotation speed of car.

Recognize the placement line

1) Determine the number of lines

In the process of identifying the color of block, we set the corresponding numbers of recognized lines for placement position of different blocks, as the figure shown below:

```
194        position = {'red':1, 'green':2, 'blue':3, 'None':-1}
```

If the recognized color is red, the robot will run the the code for transporting the block to the first placement line when it identifies single line.

If the recognized color is green, the robot will transport the block to the second placement line when only two lines are recognized.

In the process of following line, the car will keep detecting the placement line. If the following condition is satisfied, which means the line is recognized.

```
220                    if line_width > 100 and block_clamp:
```

Then determine the position of placement line.

```
226                    if transversae_num == position[target_color]:
```

The width of the line is obtained by the following function.

```
147            if detect_step == 'line':
148                line_center_x = center_x
149                line_center_y = center_y
150                line_width = data
```

2) Stop recognizing

After all the lines are recognized completely, the recognition function will be stopped to prevent the interference from repeat recognition of the same placement line.

```
221                    if (time.time()-transversae_time) > 1:
222                        transversae_num += 1
223                        print(transversae_num)
224                        transversae_time = time.time()
```

Place the block

When the numbers of recognized placement lines is equivalent to the numbers of placement lines corresponding to the placement position of the target block,

the car will stop in the corresponding position and the robotic arm will be controlled to place the block to the corresponding position.

```
235                 elif place_en:
236                     if time.time() >= place_delay: #
                        延时停下来，把色块放到横线旁边
237                         rospy.sleep(0.1)
238                         set_velocity.publish(0, 0, 0)
239                         target = ik.setPitchRanges((-0.24, 0.00, -0.04), -180,
                            -180, 0) #机械臂移动到色块放置位置
240                         if target:
241                             servo_data = target[1]
242                             bus_servo_control.set_servos(joints_pub, 1200, ((6,
                                 servo_data['servo6']),))
243                             rospy.sleep(1)
244                             bus_servo_control.set_servos(joints_pub, 1500, ((3,
                                 servo_data['servo3']), (4, servo_data['servo4']),
                                (5, servo_data['servo5'])))
245                         rospy.sleep(1.8)
246
247                         bus_servo_control.set_servos(joints_pub, 500, ((1, 150
                            ),))  #张开机械爪
248                         rospy.sleep(0.8)
```

In the code shown in the figure above, the inverse kinematics is used to set the movement of robotic arm. Take code "**target = ik.setPitchRanges((-0.24, 0.00, -0.04), -180, -180, 0)**" as example:

The first parameter "**(-0.24, 0.00, -0.04)**" is the coordinate value (x,y,and z axes) of the end of robotic arm.

The second parameter "-180" is the pitch angle value of the end of robotic arm.

The third and fourth parameter "-180" and "0" is the range of the pitch angle.

Due the limitation of the detected range of camera, when the car has not moved to the corresponding position and the lines is no longer in the detected range. Therefore, it is necessary to add a delay, so that the car can keep moving when the line is not recognized.

```
228                             if transversae_num == 1:
229                                 place_delay = time.time() + 1.1 #
                                    设置延时停下来时间
230                             elif transversae_num == 2:
231                                 place_delay = time.time() + 1.1
232                             elif transversae_num == 3:
233                                 place_delay = time.time() + 1.2
```

Then the car continues following the line and return to the initial position, and starts the next round of recognizing and sorting.

```
                    move_time = time.time() + (11.5 - transversae_num) # 设置放置色块后要巡线的时间，让机器人回到初始位置

                    # 变量重置
                    place_en = False
                    block_clamp = False
                    target_color = 'None'
                    set_rgb('black')
                    transversae_num = 0

                if not block_clamp and time.time() >= move_time: # 放置色块后机器人巡线回到初始位置
                    rospy.sleep(0.1)
                    set_velocity.publish(0, 0, 0)
                    detect_step = 'color'
```