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/intelligent_palletizer/scripts/intelligent_palletizer_node.py (stacking control)

2. Program Performance

After the game starts, ArmPi Pro will recognize the block tag within the detected range. Then the robotic arm will grip and stack the block at the tacking area.

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

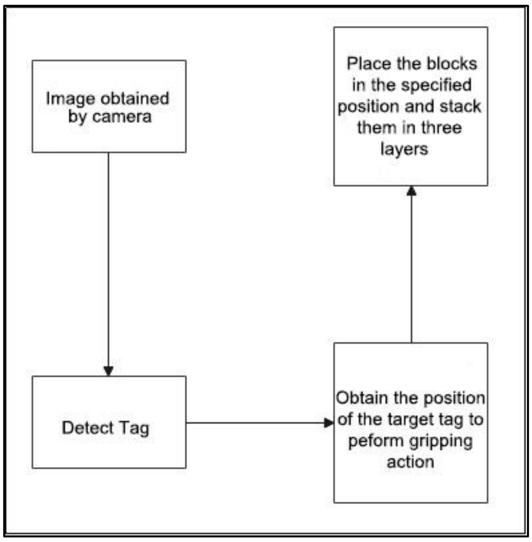
Imported Module	Function
import sys	The sys module of Python is imported to access to system-related functionalities and

	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

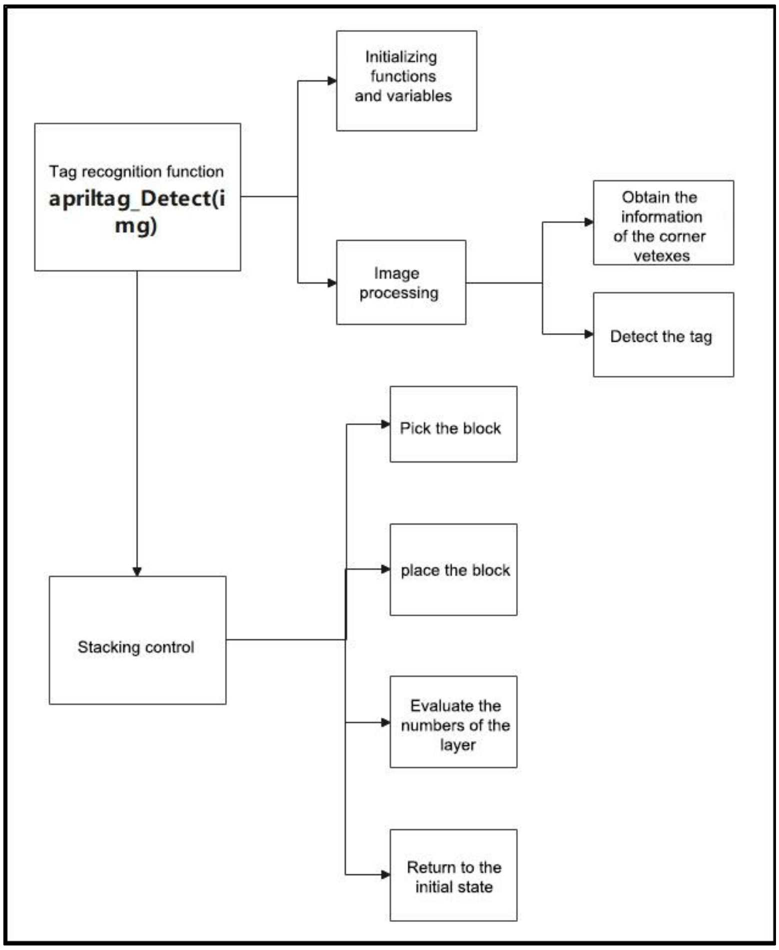
	chassis control.
from visual_patrol.srv import SetTarget	The SetTarget service type is imported from the visual_patrol.srv module is used to set a target for line following.
from hiwonder_servo_msgs.msg import MultiRawIdPosDur	The MultiRawIdPosDur message type is imported from the hiwonder_servo_msgs.msg 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 using a camera to capture image information, the process starts by detecting labels. The position of the labeled wooden block is then determined through inverse kinematics for gripping. Subsequently, the wooden block is placed in a designated location, and layers are stacked, with a maximum of three layers.

3.3 Code Analysis



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

3.3.1 Image Processing

Initializing Functions and Variables

```
# 检测apriltag函数
detector = apriltag.Detector(searchpath=apriltag._get_demo_searchpath())
def apriltag_Detect(img):
    global pub_time
    global publish_en
    
    msg = Result()
    img_copy = img.copy()
    img_h, img_w = img.shape[:2]
    frame_resize = cv2.resize(img_copy, size_m, interpolation=cv2.
    INTER_NEAREST)
    gray = cv2.cvtColor(frame_resize, cv2.COLOR_BGR2GRAY)
    detections = detector.detect(gray, return_image=False)
```

Obtain the information of corner point.

Obtain the four corner points through np.rint() function.

```
    if len(detections) != 0:
        for i, detection in enumerate(detections):
            corners = np.rint(detection.corners)  # 获取四个角点
            for i in range(4):
                corners[i][0] = int(Misc.map(corners[i][0], 0, size_m[0],
                 0, img_w))
                corners[i][1] = int(Misc.map(corners[i][1], 0, size_m[1],
                 0, img_h))
```

Detect Tag

1) After getting the corner point information of tag, recognize tag by calling drawContours() function in cv2 library.

```
                cv2.drawContours(img, [np.array(corners, np.int)], -1, (0,
                255, 255), 2)
```

The meaning of parameter in parentheses is as follow:

The first parameter “img” is the input image.

The second parameter “[np.array(corners, np.int)]” is the contour which is list in Python.

The third parameter “-1” is the index of contour. The value here represents all contour in drawing contour list.

The fourth parameter “(0, 255, 255)” is the contour color. The value sequence is B, G, and R. The color here is yellow.

The fifth parameter “2” is the width of contour.

Obtain the tag model (tag_family) and ID (tag_id).

```
            tag_family = str(detection.tag_family, encoding='utf-8')  # 获取tag_family
            tag_id = int(detection.tag_id)      # 获取tag_id
```

By calling putText() function in cv2 library, print the tag ID and type on the live feed image.

```
            cv2.putText(img, str(tag_id), (object_center_x - 10, object_center_y + 10), cv2.FONT_HERSHEY_SIMPLEX, 1, [0, 255, 255], 2)
```

The meaning of parameters in parentheses is as follow:

The first parameter “**img**” is the input image.

The second parameter “**str(tag_id)**” is the displayed content.

The third parameter “**(object_center_x - 10, object_center_y + 10)**” is the display position.

The fourth parameter “**cv2.FONT_HERSHEY_SIMPLEX**” is the font type.

The fifth parameter “**1**” is the size of font.

The sixth parameter “**[0, 255, 255]**” is the font color and its sequence is B, G, R. The value here is yellow.

The seventh parameter “**2**” is the thickness of font.

3.3.2 Control Action

```
# 初始位置
def initMove(delay=True):
    with lock:
        target = ik.setPitchRanges((0, 0.15, 0.0), -180, -180, 0) # 逆运动学求解
        if target:
            servo_data = target[1]
            bus_servo_control.set_servos(joints_pub, 1800, ((1, 200), (2, 500), (3, servo_data['servo3']), (4, servo_data['servo4']),
(5, servo_data['servo5']),(6, servo_data['servo6'])))
    if delay:
        rospy.sleep(2)

def turn_off_rgb():
    led = Led()
    led.index = 0
    led.rgb.r = 0
    led.rgb.g = 0
    led.rgb.b = 0
    rgb_pub.publish(led)
```

Pick the Block

1) By determining whether the coordinate of tag block is change, then we can determine if the black is stable. If it meets the conditions, robotic arm will grip the block.

```
    while __isRunning:
        if steadier and object_center_x > 0 and object_center_y > 0:
            # 木块已经放稳，进行追踪夹取
```

2) Robotic arm moves to above the block using the inverse kinematics.

```
            # 机械臂追踪移动到木块上方
            target = ik.setPitchRanges((0, round(y_dis, 4), 0.0), -180, -180, 0)
```

The analysis of code above is as follow:

The first parameter “0” is the position in x-axis.

The second parameter “**round(y_dis, 4), 0.0)**” is the position in y-axis.

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

The fourth and fifth parameter “-180”, “0” is the range of pitch angle.

3) When the determination conditions are met, robotic arm will stop above the

block and adjust the angle of robotic arm.

```
if abs(dx) < 3 and abs(dy) < 0.003 and not stack_en: # 等待机械臂稳定停在木块上方
    count_ += 1
    if count_ == 10:
        count_ = 0
        stack_en = True
        angle = object_angle % 90
        print(angle)
        offset_y = Misc.map(target[2], -180, -150, -0.01, 0.02) # 设置位置补偿
```

4) Then the robotic arm is controlled to grip and raise the block through bus_servo_control.set_servos() function.

```
bus_servo_control.set_servos(joints_pub, 500, ((1, 450),)) # 闭合机械爪
rospy.sleep(0.8)

bus_servo_control.set_servos(joints_pub, 1500, ((1, 450), (2, 500), (3, 80), (4, 825), (5, 625), (6, 500))) # 机械臂抬起来
rospy.sleep(1.5)
```

Place the Block

Using the inverse kinematics to control the robotic arm to transport and put down the block.

```
target = ik.setPitchRanges(place_coord[stack_num], -180, -180, 0) # 机械臂移动到色块放置位置
if target:
    servo_data = target[1]
    bus_servo_control.set_servos(joints_pub, 1000, ((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']))) # 再放下了
rospy.sleep(1)
```

Take “target = ik.setPitchRanges(place_coord[stack_num], -180, -180, 0)” as example. Among them, the first parameter “place_coord[stack_num]” represent s the coordinate position of tag block. The following image is the position information of corresponding ID.

```
place_coord = {1:(0.18, 0.0, -0.09),
               2:(0.18, 0.0, -0.05),
               3:(0.18, 0.0, -0.02)}
```

The second parameter “-180” is the pitch angle.

The third and fourth parameters “-180” and “0” are the range of the pitch angle.

Controlling each servo by bus_servo_control.set_servos（） and let gripper put down and release the block.

```
            if target:
                servo_data = target[1]
                bus_servo_control.set_servos(joints_pub, 1000, ((3, servo_data['servo3']), (4, servo_data['servo4']), (5, servo_data['servo5']))) # 再放下了
            rospy.sleep(1)

            bus_servo_control.set_servos(joints_pub, 500, ((1, 150),)) #张开机械爪
            rospy.sleep(0.8)
```

Evaluate the layers

When stacking action is executed three times, it will starts from scratch.

```
            if stack_num >= 3:
                stack_num = 0
```

Restore to the Initial Status

Robotic arm returns to the initial posture through inverse kinematics.

```
            #机械臂复位
            target = ik.setPitchRanges((0, 0.15, 0.0), -180, -180, 0)
            if target:
                servo_data = target[1]
                bus_servo_control.set_servos(joints_pub, 1000, ((1, 200), (2, 500), (3, servo_data['servo3']),
                                                                (4, servo_data['servo4']), (5, servo_data['servo5'])))
                rospy.sleep(1)
                bus_servo_control.set_servos(joints_pub, 1500, ((6, servo_data['servo6']),))
                rospy.sleep(1.5)

            start_en = True
            reset()  # 变量重置
```