Lab3

0. 封面

- **实验题目**：实验2 小车多颜色智能搬运
- **院系**：软件学院
- **设计者**：

学号	姓名
202100300063	李彦浩
202100300340	黄幸兒
202100300078	李世会
202100161049	徐芃恺

- **指导教师**：李新
- **实验时间**：2023-12-5(18:00-20:00)

1. 实验内容

1.1. 实验目标

实现小车来回往返三次，每次根据颜色追踪夹取指定的物块，并将物块放到指定位置，三个物块放在同一位置，实现物块的码垛

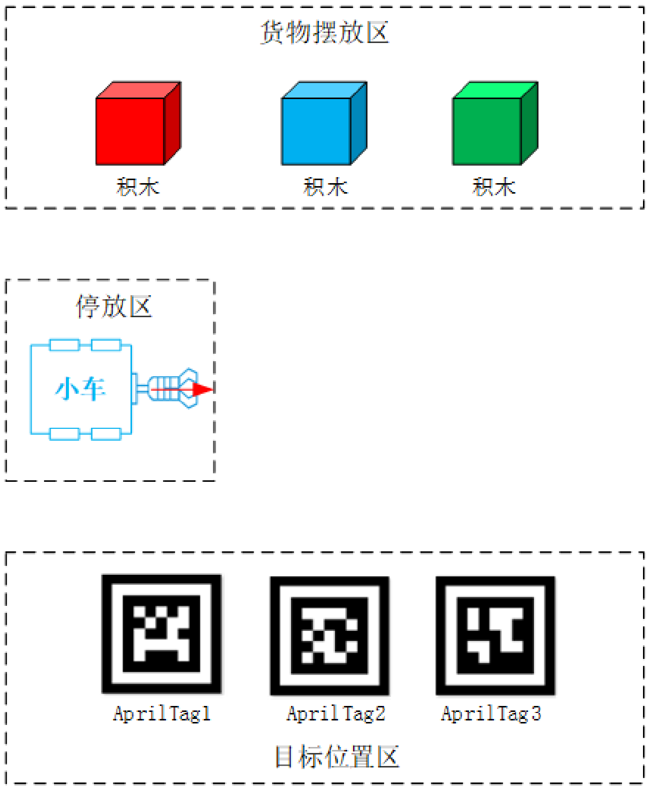


图 实验场地示意图

1.2. 实验要求

1. 验证小车的导航和路径规划算法是否正确工作，能否成功将小车引导到目标位置。
2. 验证小车的机械臂能否正确地夹取积木，并且在搬运过程中不会掉落。
3. 验证小车能否正确识别目标放置区的位置和形状，并将积木放置到指定区域。
4. 验证小车的重复执行任务的能力，能否按照要求重复执行任务。
5. 验证小车的健壮性，能否在电量不足或其他异常情况下仍然保持正常运行

2. 需求分析

2.1. 功能性需求分析

1. 需要实现启动小车时，小车从停放区行驶到货物摆放区
2. 需要实现在货物摆放区夹取积木
3. 需要实现小车夹取积木后，搬运到目标放置区，并放置到指定区域（若区域内已有木块，则叠放在其上方）
4. 需要实现将需求2、3重复三次执行

2.2. 非功能性需求分析

1. 代码可扩展性强，可以根据需求扩展相关功能
2. 代码健壮性强，可以在小车电量不足时也保证夹取的成功率以及行驶的准确度。

3. 代码可维护性强，耦合度低，便于维护
4. 代码复用性高，控制小车的不同行为动作的代码按模块进行封装，用到时直接调用

3. 实验设计

本次实验内容承接上次实验，请参照上次实验报告阅读。

3.1. 总体流程控制

简单来说，就是将实验二的内容重复三次。每次识别的色块颜色均不同。

对于不同的颜色，可以用一个颜色数组来标识，每轮循环取其中的一个颜色：

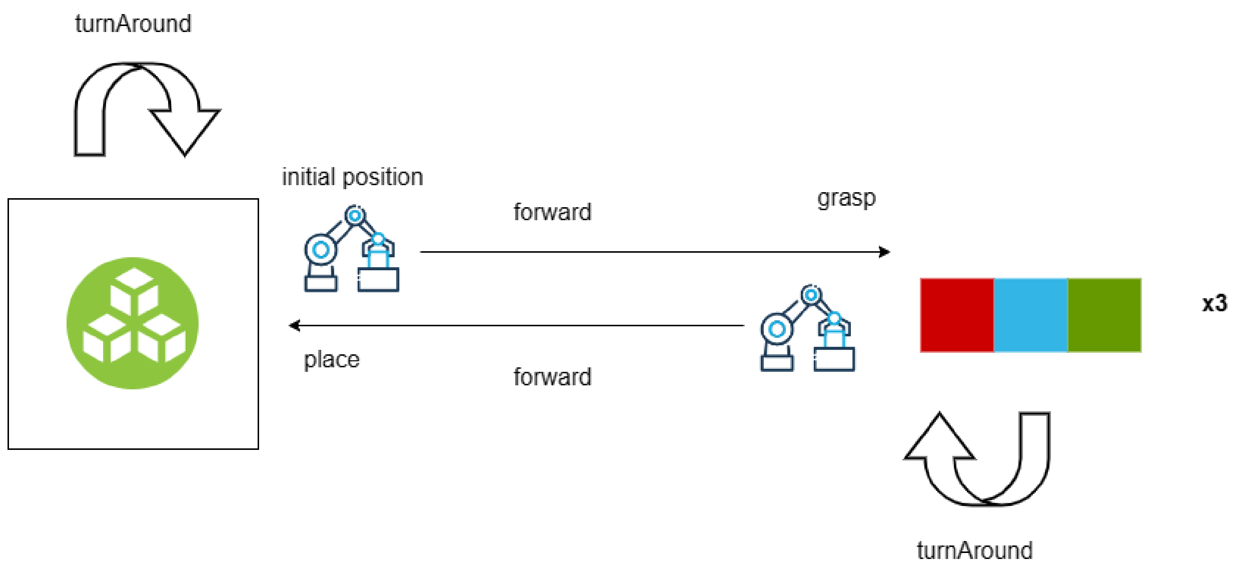
```
target_colors = ['red', "blue", "green"]  # 3 turns
cur_turn = 0
```

其中cur_turn代表当前一轮识别的色块的颜色。

这里要注意的是，实验二中夹取并放置一个色块后就结束了任务，而实验三要求小车还要返回货物放置区域，因此最终放置色块到目标区域后，要多加一次转身。

```
steps = ['forward', 'grasp', 'turnAround', 'forward', 'place', 'turnAround']  # each has 6 steps
cur_step = 0
```

流程图如下：



3.2. turnAround部分实现的调整

在本次实验中，我们有两处turnAround，在后一次转身时，需要将目标颜色在数组中的索引加1，而前一次转身不需要。因此turnAround部分的实现需要调整。

```
elif steps[cur_step] == 'turnAround':
      # -0.55 * 2 = -0.1 = 90deg, thus -1.1 * 2 = -2.2 = 180 deg
      set_velocity.publish(0.0, 90.0, -0.55)
      #rospy.sleep(0)
      rospy.sleep(4.07)
      # stop
      set_velocity.publish(0.0, 0.0, 0.0)
      rospy.sleep(1.0)
      # next step / next turn
      if cur_step < len(steps)-1:
          cur_step += 1
      else:
          cur_turn += 1
          cur_step = 0
```

这里判断当前一步是否是当前轮次的最后一步，如果是，跳到下一个目标颜色，重新开始；否则继续接下来的步骤。

3.2. place部分实现的调整

除了turnAround和place部分需要较大的调整之外，其余部分可以直接使用上一次实验的代码，因为在需求和流程上，这两部分的实现逻辑都是极为相似的。

而place之所以要修正，是因为我们在实验二中基于标签参照物来放置物块。这本身其实是没有问题的。但实验三中，我们组尽可能地想要实现码垛的功能。而由于三次往返中，每次计算得到的标签坐标都是不一样的（甚至有很大的偏差），因此就无法实现把物块垒起来的功能。

因此在这里我们反而选择实验二中抛弃的固定位置放置的模式。我们的思路是：只要三次往返的动作足够精准（如走直线/180°转身等），固定位置放置就可以避免复杂的参数计算，直接准确的放置。

其实我们也不是没有想过更加“智能”的方式。在最初，我们组尝试在放置时，追踪上一次放置的色块，比如第一次放置的是红色的，第二次放置的是蓝色的，那么第二次放置蓝色色块时将追踪红色色块，这样就能将蓝色色块放置在红色色块上，比第二种方式智能，比第一种方式精准。

但在调试时，我们发现一个不可调解的结构性矛盾。当机械爪夹着物块时，物块会挡住摄像头！当机械臂完全移动到码垛上后，物块会完全挡住摄像头，这样就完全挡住了上一次放置的色块。机械臂会移动开，以便寻找上一次放置的色块。这个过程将循环往复，永无宁日。

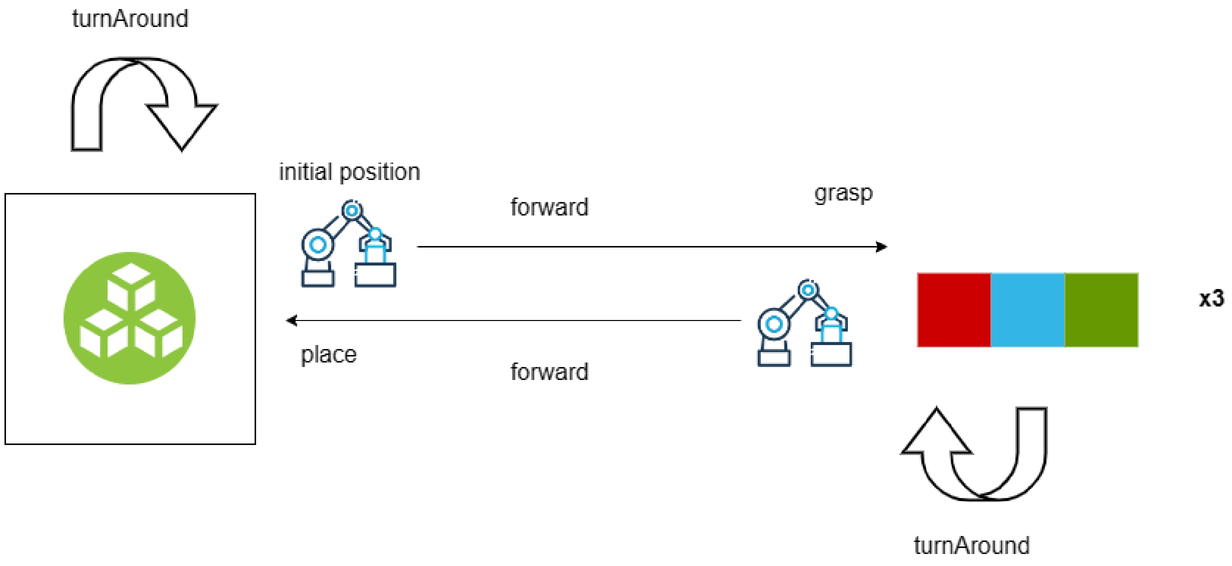
因此，在权衡利弊后，我们组最终选择了第二种方式。

```
elif steps[cur_step] == 'place':
    if flag:
        x_dis = 500
        y_dis = 0.15
        flag = False
        #if cur_turn != 0:
            #visual_running('color', target_colors[cur_turn - 1])  # last block's color

    #if cur_turn == 0:
    target = ik.setPitchRanges((0, round(0.2, 4), 0), -180, -180, 0)
    if target:
        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)))
        rospy.sleep(0.02)
    num += 1
    rospy.loginfo("num:%d",num)
    if num == 4:
        num = 0
        offset_y = Misc.map(target[2], -180, -150, -0.03, 0.03)
        arm_move = True
    if arm_move:
        rospy.loginfo("ready to place april tag")
        angle_pul = Misc.map(angle, 0, 80, 500, 800)
        bus_servo_control.set_servos(joints_pub, 500, ((2, angle_pul),))
        rospy.sleep(0.5)
        target = ik.setPitchRanges((0, round(0.2, 4), -0.08+0.03*cur_turn), -180, -180, 0)  # arm down
        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']), (6, x_dis)))
        rospy.sleep(1.1)
        buzzer_pub.publish(0.1)
        bus_servo_control.set_servos(joints_pub, 500, ((1, 120),))  # open claw
        rospy.sleep(0.5)
        # arm reset
        bus_servo_control.set_servos(joints_pub, 1500,
                                     ((1, 120), (2, 500), (3, 80), (4, 825), (5, 625),
                                      (6, 500)))  # arm up
        rospy.sleep(1.5)
        arm_move = False
        cur_step += 1
    # next step
```

这里我们直接让机械臂前伸4次，每次都是一个很小的增量，然后放置物块。

4. 流程图



5. 完整工程

主要代码如下，详见于附录：

```
#!/usr/bin/python3
# coding=utf8
# Date:2022/05/30import sys
import sys
import cv2
import time
import math
import rospy
import numpy as np
from threading import RLock, Timer, Thread

from std_srvs.srv import *
from std_msgs.msg import *
from sensor_msgs.msg import Image

from sensor.msg import Led
from chassis_control.msg import *
from visual_processing.msg import Result
from visual_processing.srv import SetParam
from intelligent_transport.srv import SetTarget
from hiwonder_servo_msgs.msg import MultiRawIdPosDur

from armpi_pro import PID
from armpi_pro import Misc
from armpi_pro import bus_servo_control
from kinematics import ik_transform

lock = RLock()
ik = ik_transform.ArmIK()

target_colors = ['red', "blue", "green"]  # 3 turns
cur_turn = 0
steps = ['forward', 'grasp', 'turnAround', 'forward', 'place', 'turnAround']  # each has 6 steps
cur_step = 0
arm_move = False
target_color = 'None'  # deprecated in Lab3
__isRunning = False

# color for LED
range_rgb = {
    'red': (0, 0, 255),
    'blue': (255, 0, 0),
    'green': (0, 255, 0),
}

x_dis = 500  # 6th steering engine initial angle
y_dis = 0.15  # a magic number for ik solution
color_center_x = 0
color_center_y = 0
apriltag_center_x = 0
apriltag_center_y = 0
centreX = 320
centreY = 410
offset_y = 0
object_angle = 0
angle = 0
tag_id = 0
flag = False  # whether it is ok for next step

# cube tracking
color_x_pid = PID.PID(P=0.06, I=0, D=0)
color_y_pid = PID.PID(P=0.00003, I=0, D=0)
apriltag_x_pid = PID.PID(P=0.06, I=0, D=0)  # pidåå§å
apriltag_y_pid = PID.PID(P=0.00003, I=0, D=0)
result_sub = None
heartbeat_timer = None

num = 0


def run(msg):
    global lock
    global arm_move
    global steps, cur_step
    global color_center_x, color_center_y
    global apriltag_center_x, apriltag_center_y, object_angle, tag_id

    # rospy.loginfo("center_x = %f,center_y = %f",center_x,center_y)
    # rospy.loginfo("cur_step: %d",cur_step)
    with lock:
        if steps[cur_step] == 'grasp' and not arm_move:
            color_center_x = msg.center_x
            color_center_y = msg.center_y
        elif steps[cur_step] == 'place' and not arm_move:
            apriltag_center_x = msg.center_x
            apriltag_center_y = msg.center_y
            object_angle = msg.angle
            tag_id = msg.data


def move():
    global cur_step, steps
    global target_colors, cur_turn
    global x_dis, y_dis
    global arm_move, flag
    global offset_y, object_angle, angle
    global __isRunning
    global target_color
    global num
    global color_center_x, color_center_y
    global apriltag_center_x, apriltag_center_y, tag_id

    while __isRunning:
        # 3 turns
        while cur_turn < len(target_colors):
            # rospy.loginfo("cur_step: %d",cur_step)
            if steps[cur_step] == 'forward':
                set_velocity.publish(85.0, 90.0, 0.0)
                rospy.sleep(10.45+0.3*cur_turn)
                #rospy.sleep(0)
                # stop
                set_velocity.publish(0, 0, 0)
                # next step
                flag = True
                cur_step += 1
            elif steps[cur_step] == 'grasp':
                if flag:
                    x_dis = 500
                    y_dis = 0.15
                    flag = False
                    visual_running('color', target_colors[cur_turn])
                    rospy.loginfo("current target_color:%s",target_colors[cur_turn])

                diff_x = abs(color_center_x - centreX)
                diff_y = abs(color_center_y - centreY)

                #rospy.loginfo("color block pos:x %d,y %d",color_center_x,color_center_y)

                if diff_x < 10:
                    color_x_pid.SetPoint = color_center_x
                else:
                    color_x_pid.SetPoint = centreX
                color_x_pid.update(color_center_x)
                dx = color_x_pid.output
                #rospy.loginfo("dx: %f", dx)
                x_dis += int(dx)
                x_dis = 200 if x_dis < 200 else x_dis
                x_dis = 800 if x_dis > 800 else x_dis

                if diff_y < 10:
                    color_y_pid.SetPoint = color_center_y
                else:
                    color_y_pid.SetPoint = centreY
                color_y_pid.update(color_center_y)
                dy = color_y_pid.output
                #rospy.loginfo("dy: %f", dy)
                y_dis += dy
                y_dis = 0.12 if y_dis < 0.12 else y_dis
                y_dis = 0.28 if y_dis > 0.28 else y_dis
                target = ik.setPitchRanges((0, round(y_dis, 4), 0.0), -180, -180, 0)

                if target:
                    servo_data = target[1]
                    #rospy.loginfo("servo_data[3]: %d,servo_data[4]: %d,servo_data[5]: %d,x_dis: %d ",
                                  #servo_data['servo3'],
                                  #servo_data['servo4'], servo_data['servo5'], x_dis)
                    bus_servo_control.set_servos(joints_pub, 200, ((3, servo_data['servo3']), (4, servo_data['servo4']),
                                                                  (5, servo_data['servo5']), (6, x_dis)))
                    rospy.sleep(0.3)
                if abs(dx) < 2 and abs(dy) < 0.003:
                    num += 1
                    if num == 10:
                        num = 0
                        # rospy.loginfo("ready to grasp")
                        offset_y = Misc.map(target[2], -180, -150, -0.03, 0.03)
                        arm_move = True  # ready to grasp
                else:
                    num = 0

                if arm_move:
                    buzzer_pub.publish(0.1)
                    bus_servo_control.set_servos(joints_pub, 500, ((1, 20),))  # open claw
                    rospy.sleep(0.5)
                    target = ik.setPitchRanges((0-0.07, round(y_dis, 4), -0.08), -180, -180, 0)  # arm down
                    if target:
                        servo_data = target[1]
                        #rospy.loginfo("servo_data[3]: %d,servo_data[4]: %d,servo_d ata[5]: %d,x_dis: %d",
                                      #servo_data['servo3'], servo_data['servo4'], servo_data['servo5'], x_dis)
                        bus_servo_control.set_servos(joints_pub, 1000,
                                                     ((3, servo_data['servo3']), (4, servo_data['servo4']),
                                                      (5, servo_data['servo5']), (6, x_dis)))
                    rospy.sleep(1.5)
                    bus_servo_control.set_servos(joints_pub, 500, ((1, 450),))  # close claw
                    rospy.sleep(0.8)
                    bus_servo_control.set_servos(joints_pub, 1500,
                                                 ((1, 450), (2, 500), (3, 80), (4, 825), (5, 625), (6, 500)))  # æºæ¢°èæ¬èµ·æ¥
                    rospy.sleep(1.5)
                    arm_move = False
                    # next step
                    cur_step += 1
            elif steps[cur_step] == 'turnAround':
                # -0.55 * 2 = -0.1 = 90deg, thus -1.1 * 2 = -2.2 = 180 deg
                set_velocity.publish(0.0, 90.0, -0.55)
                #rospy.sleep(0)
                rospy.sleep(4.07)
                # stop
                set_velocity.publish(0.0, 0.0, 0.0)
                rospy.sleep(1.0)
                # next step / next turn
                if cur_step < len(steps)-1:
                    cur_step += 1
                else:
                    cur_turn += 1
                    cur_step = 0
            elif steps[cur_step] == 'place':
                if flag:
                    x_dis = 500
                    y_dis = 0.15
                    flag = False
                    #if cur_turn != 0:
                        #visual_running('color', target_colors[cur_turn - 1])  # last block's color

                #if cur_turn == 0:
                target = ik.setPitchRanges((0, round(0.2, 4), 0), -180, -180, 0)
                if target:
                    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)))
                    rospy.sleep(0.02)
                num += 1
                rospy.loginfo("num:%d",num)
                if num == 4:
                    num = 0
                    offset_y = Misc.map(target[2], -180, -150, -0.03, 0.03)
                    arm_move = True
                if arm_move:
                    rospy.loginfo("ready to place april tag")
                    angle_pul = Misc.map(angle, 0, 80, 500, 800)
                    bus_servo_control.set_servos(joints_pub, 500, ((2, angle_pul),))
                    rospy.sleep(0.5)
                    target = ik.setPitchRanges((0, round(0.2, 4), -0.08+0.03*cur_turn), -180, -180, 0)  # arm down
                    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']), (6, x_dis)))
                    rospy.sleep(1.1)
                    buzzer_pub.publish(0.1)
                    bus_servo_control.set_servos(joints_pub, 500, ((1, 120),))  # open claw
                    rospy.sleep(0.5)
                    # arm reset
                    bus_servo_control.set_servos(joints_pub, 1500,
                                                 ((1, 120), (2, 500), (3, 80), (4, 825), (5, 625),
                                                  (6, 500)))  # arm up
                    rospy.sleep(1.5)
                    arm_move = False
                    cur_step += 1
                # next step

    reset()


def init():
    rospy.loginfo('Lab3 Init')
    initMove()
    reset()


def initMove(delay=True):
    with lock:
        bus_servo_control.set_servos(joints_pub, 1500, ((1, 75), (2, 500), (3, 80), (4, 825), (5, 625), (6, 500)))
    if delay:
        rospy.sleep(2)


def reset():
    global x_dis, y_dis
    global cur_step
    global target_color
    global arm_move, flag
    global color_center_x, color_center_y
    global apriltag_center_x, apriltag_center_y

    with lock:
        cur_step = 0
        target_color = 'None'
        arm_move = False
        flag = False
        color_x_pid.clear()
        color_y_pid.clear()
        apriltag_x_pid.clear()
        apriltag_y_pid.clear()
        off_rgb()
        x_dis = 500
        y_dis = 0.15
        color_center_x = 0
        color_center_y = 0
        apriltag_center_x = 0
        apriltag_center_y = 0
        set_velocity.publish(0, 90, 0)


def off_rgb():
    led = Led()
    led.index = 0
    led.rgb.r = 0
    led.rgb.g = 0
    led.rgb.b = 0
    rgb_pub.publish(led)
    led.index = 1
    rgb_pub.publish(led)


def set_rgb(color):
    global lock
    with lock:
        led = Led()
        led.index = 0
        led.rgb.r = range_rgb[color][2]
        led.rgb.g = range_rgb[color][1]
        led.rgb.b = range_rgb[color][0]
        rgb_pub.publish(led)
        rospy.sleep(0.05)
        led.index = 1
        rgb_pub.publish(led)
        rospy.sleep(0.05)


def set_running(msg):
    if msg.data:
        start_running()
    else:
        stop_running()

    return [True, 'set_running']


def set_target(msg):
    global lock
    global target_color

    rospy.loginfo('%s', msg)
    with lock:
        target_color = msg.data
        led = Led()
        led.index = 0
        led.rgb.r = range_rgb[target_color][2]
        led.rgb.g = range_rgb[target_color][1]
        led.rgb.b = range_rgb[target_color][0]
        rgb_pub.publish(led)
        led.index = 1
        rgb_pub.publish(led)
        rospy.sleep(0.1)
    return [True, 'set_target']


def stop_running():
    global lock
    global __isRunning

    rospy.loginfo('stop running Lab3')
    with lock:
        __isRunning = False
        reset()
        initMove(delay=False)
        # set_velocity.publish(0, 0, 0)
        rospy.ServiceProxy('/visual_processing/set_running', SetParam)()


def start_running():
    global lock
    global __isRunning

    rospy.loginfo('start running Lab3')
    with lock:
        # init()
        __isRunning = True
        rospy.sleep(0.1)
        th = Thread(target=move)
        th.setDaemon(True)
        th.start()


# enter service
def enter_func(msg):
    global lock
    global result_sub

    rospy.loginfo('enter object tracking')
    init()
    with lock:
        if result_sub is None:
            # wake up visual_processing_node
            rospy.ServiceProxy('/visual_processing/enter', Trigger)()
            # subscribe the result from the detection module of visual_processing_node
            result_sub = rospy.Subscriber('/visual_processing/result', Result, run)

    return [True, 'enter']


# exit service
def exit_func(msg):
    global lock
    global result_sub
    global __isRunning
    global heartbeat_timer

    rospy.loginfo('exit Lab3')
    with lock:
        __isRunning = False
        rospy.ServiceProxy('/visual_processing/exit', Trigger)()
        reset()
        try:
            if result_sub is not None:
                result_sub.unregister()
                result_sub = None
            if heartbeat_timer is not None:
                heartbeat_timer.cancel()
                heartbeat_timer = None
        except BaseException as e:
            rospy.loginfo('%s', e)

    return [True, 'exit']


# heartbeat connection service
def heartbeat_srv_cb(msg):
    global heartbeat_timer

    if isinstance(heartbeat_timer, Timer):
        heartbeat_timer.cancel()
    if msg.data:
        heartbeat_timer = Timer(5, rospy.ServiceProxy('/Lab3/exit', Trigger))
        heartbeat_timer.start()
    rsp = SetBoolResponse()
    rsp.success = msg.data

    return rsp


if __name__ == '__main__':
    # init node
    rospy.init_node('Lab3', log_level=rospy.DEBUG)
    # Steering Engine
    joints_pub = rospy.Publisher('/servo_controllers/port_id_1/multi_id_pos_dur', MultiRawIdPosDur, queue_size=1)
    # ServiceProxy
    visual_running = rospy.ServiceProxy('/visual_processing/set_running', SetParam)
    # Services
    enter_srv = rospy.Service('/Lab3/enter', Trigger, enter_func)
    running_srv = rospy.Service('/Lab3/set_running', SetBool, set_running)
    set_target_srv = rospy.Service('/Lab3/set_target', SetTarget, set_target)
    exit_srv = rospy.Service('/Lab3/exit', Trigger, exit_func)
    heartbeat_srv = rospy.Service('/Lab3/heartbeat', SetBool, heartbeat_srv_cb)
    # Publishers
    # Chassis Control
    set_velocity = rospy.Publisher('/chassis_control/set_velocity', SetVelocity, queue_size=1)
    set_translation = rospy.Publisher('/chassis_control/set_translation', SetTranslation, queue_size=1)
    # Buzzer Control
    buzzer_pub = rospy.Publisher('/sensor/buzzer', Float32, queue_size=1)
    # RGB LED Control
    rgb_pub = rospy.Publisher('/sensor/rgb_led', Led, queue_size=1)
    rospy.sleep(0.5)

    try:
        rospy.spin()
    except KeyboardInterrupt:
        rospy.loginfo('Shutting down')
```

6. 如何复现我们的成果

仅提供命令行方式。请注意，小车与物块摆放的参数请参照2023年秋季学期SDU软件学院实验楼501的实验环境。如果不同，请自行调整代码中的参数。

6.1. 启动服务节点

```
rosrun Lab3 Lab3_node.py
```

6.2. 进入服务

```
rosservice call /Lab3/enter "{}"
```

6.3. 开始运行

```
rosservice call /Lab3/set_running "data: true"
```