Roban机器人实验报告

学号 2020212097　姓名 佟欣阳

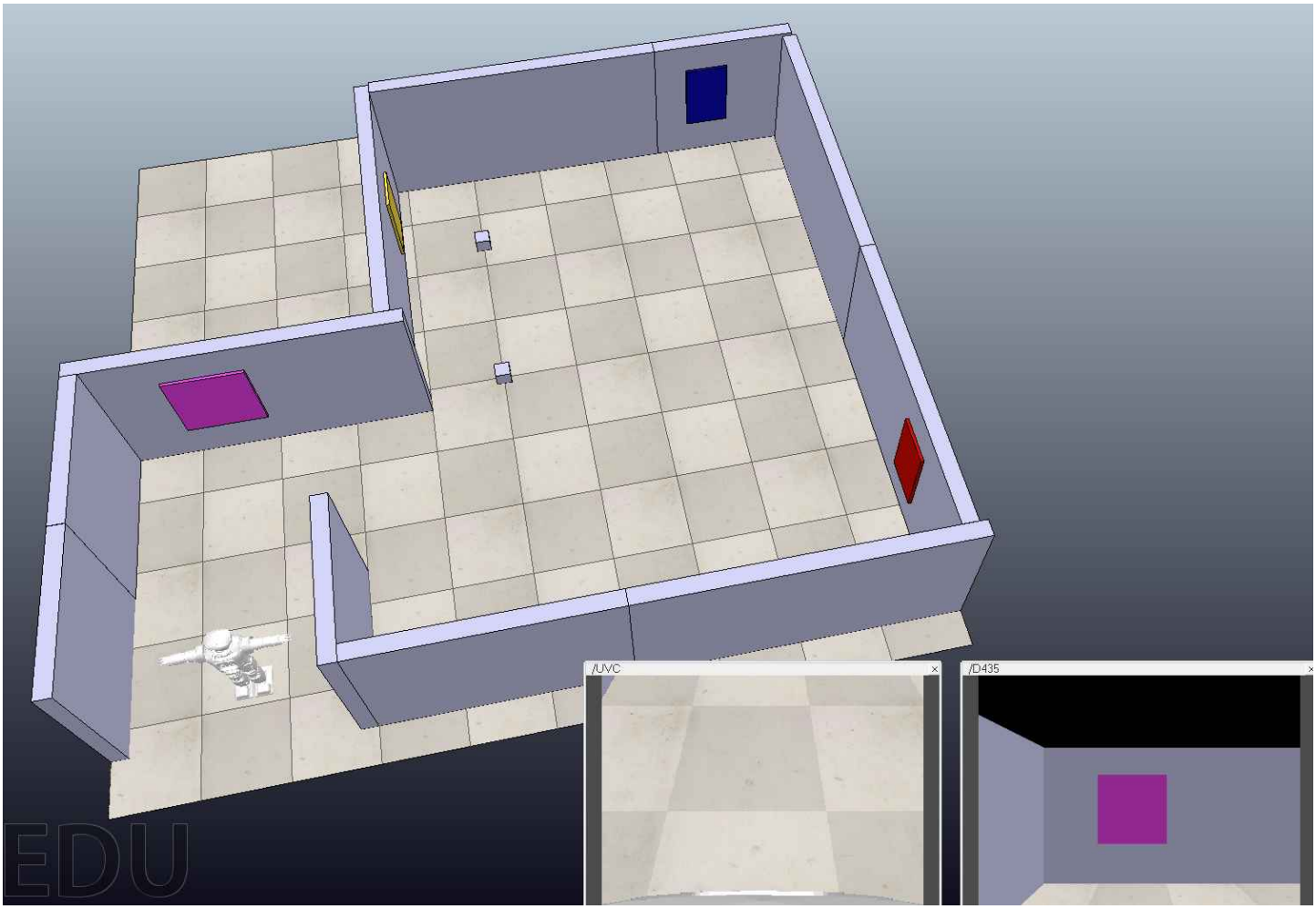
一、实验目的

掌握coppeliasim等仿真环境的使用方法，了解ROS操作系统，通过程序读取机器人传感器数据，通过程序控制机器人。使用Webots或CoppeliaSim搭建博物馆/展览馆场景，藏品/展品数量4张以上，实物展品2个以上，障碍物若干。地图细节不限。

二、模块总览

1. Bodyhub实现行走
2. 定点自动导航移动pathtracking
3. 手部动作ikmodule
4. 键盘输入控制
5. 避障算法
6. 视觉识别

地图展示：



三、实现方法

3.1 Bodyhub实现行走

3.2 定点自动导航移动pathtracking

```
def path_tracking(self, path_point, pos_wait_mode=1,ignore_angle=False,pos_exit_high_acc=True,wait_time=0.4)
    '''
    pos_wait_mode=1:等待机器人站稳后才进行定位,精度高但慢
    ignore_angle模式:用于回程运动,忽略标记点的方向
    wait_time:等待机器人定位稳定的时间
    pos_exit_high_acc:确保终点精度高,调节时间会增加
    '''
    step_len = [0, 0, 0]
    rot_adjust = False
    path_marker_index, marker_num = 0, len(path_point)
    while not rospy.is_shutdown():
        rospy.wait_for_message('/requestGaitCommand', Bool, 100)

        if self.pose_update == False:
            print ('location not updated!')
            time.sleep(0.5)
            continue
        for i in range(3):
            step_len[i] = path_point[path_marker_index][i] - self.current_pose[i]

        if (pos_wait_mode == 1):
            time.sleep(wait_time)
```

slam生成中间路径点，输入给path_tracking函数，从而实现自动走到某个坐标点位置。

注意：需要更改public.py和Talos.lua，增加仿真中角度方向yaw的数据发布。

```
def pose_callback(self, msg):
    roban_torsoPR = msg.data
    x, y = roban_torsoPR[0], roban_torsoPR[1]
    yaw=roban_torsoPR[9]
    # _, _, yaw = self.toRPY(msg.pose.pose)
    old_pos=self.current_pose
    self.current_pose = [x, y, yaw * 180.0 / math.pi]
    if self.__debug:
        print ('pose:', self.current_pose)
    if ((old_pos[0]-self.current_pose[0])**2+(old_pos[1]-self.current_pose[1])**
    or abs(old_pos[2]-self.current_pose[2])>50:
        return#滤掉高频跳点
    self.pose_update = True
```

3.3 手部动作ikmodule

调用body_motion,使用封装机器人逆解相关操作，主要使用 toInitPose，body_motion 和 reset 方法。

```
class actions(IkModuleSim):
    def __init__(self):
        # super(actions, self).__init__()
        IkModuleSim.__init__(self)
    def arms(self):
        print("in actions")
        self.body_motion([C.RArm_z, C.RArm_y], [0.05, -0.05], 50)
        self.reset()
```

封装类见**ikmodulesim.py**

```
class IkModuleSim(object):
    #机器人位姿初始化，在动作执行前调用
    #初始化成功 返回 True ,否则，返回 False
    def body_motion(self, body_pose, value, count=100):
    # 机器人末端动作执行，给定目标位姿 body_pose，value 通过逆解得到舵机角度，控制机器人
    Args:
        - body_pose: CtrlType, or [CtrlType]
```

```
                    表示末端位置，如左/右脚(Lfoot/Rfoot)，身体中心(Torso)
        - value: list, or two dim list. RPY, xyz.
                    表示给相应末端的位姿值，[r, p, y, x, y, z]
        - count: int. times of division
                  插值次数，缺省为 100
```

从ik_lib中导入 IkModuleSim 和 CtrlType， 然后新建一个类，继承自 IkModuleSim, 然后调用上述方法，完成自定义动作。

```
if action.toInitPoses():
    action.body_motion([C.RArm_z, C.RArm_y], [0.05, -0.05], 50)
    action.body_motion([C.RArm_z, C.RArm_x], [0.11, 0.04], 50)
    action.body_motion([C.RArm_x, C.RArm_y], [0.09, 0.03], 10)
action.reset()
```

3.4 键盘输入控制

参考key_tele.py和key_cmd.py，部分代码如下：

```
def keyboard_museum(self):
    while not rospy.is_shutdown():
        cmd = self.getch('key:')
        if cmd == 'w':
            self.walking_n_steps(self.step_len[0], 0.0, 0.0, self.step_num)
        elif cmd == 's':
            self.walking_n_steps(-self.step_len[0]*0.8, 0.0, 0.0, self.step_num)
        elif cmd == 'a':
            self.walking_n_steps(0, self.step_len[1], 0.0, self.step_num)
        elif cmd == 'd':
            self.walking_n_steps(0, -self.step_len[1], 0.0, self.step_num)
        elif cmd == 'z':
            self.walking_n_steps(0.0, 0.0, self.step_len[2], self.step_num)
        elif cmd == 'c':
            self.walking_n_steps(0.0, 0.0, -self.step_len[2], self.step_num)
        elif cmd == 'x':
            self.walking_n_steps(0.0, 0.0, 0.0, self.step_num)
        elif cmd == 'q':
            return
```

捕捉键盘输入，通过不同按键实现左转、右转、直走等功能。

添加代码，使key_cmd.py成为主程序控制状态机，输入数字1到5代表目标展品，机器人接收到命令后会向指定标号展品位置自动移动，部分代码如下所示：

```
elif cmd== '1':
    print("heading position 1!")
    Poses.A_pos(1)
    time.sleep(0.1)
    action.reset()
    if action.toInitPoses():
        action.body_motion([C.RArm_z, C.RArm_y], [0.05, -0.05], 50)
        action.body_motion([C.RArm_z, C.RArm_x], [0.11, 0.04], 50)
        action.body_motion([C.RArm_x, C.RArm_y], [0.09, 0.03], 10)
    action.reset()
    time.sleep(0.5)
```

3.5 避障算法

参考roban_avoidance.py里的代码，在前方无障碍物时加快向前行进的速度，遇到障碍物后进行避障，部分代码如下所示：

```
def move(mean_distance):
    print(mean_distance)
    if mean_distance < 150:
        print("离得太近了，我识别不到了。")
        return
    elif mean_distance > 1000:
        print("前方障碍物比较远，我可以走的快一些")
        slow_walk("forward", 4)
    elif mean_distance > 500:
        print("我准备往前走了。")
        slow_walk("forward", 1)
    elif mean_distance < 250:
        print("有点近，我需要后退一下")
        slow_walk("backward", 1)
    else:
        print("前方有障碍物，我准备右转30度")
        slow_walk("rotation", angle=-30)
    WaitForWalkingDone()
```

订阅深度摄像头的话题，根据输入的距离信息控制机器人的运动方向。

3.6 视觉识别

颜色识别主要在Box_color_detecter类中实现，从 sim/d435/imgae_raw主题获取摄像头数据，通过话题消息读取颜色识别结果。

获取摄像头图像后经过腐蚀膨胀等操作后找出图片中目标hsv空间的范围。在`Task_clear_obstruction.py`中找到目标颜色名称，并最终输出在终端。

部分代码如下：

```
class Task_clear_obstruction(PublicNode):
    def __init__(self,nodename=NODE_NAME,control_id=CONTROL_ID):
        super(Task_clear_obstruction, self).__init__(nodename, control_id)

    def debug(self):
        self.colordetecter=Box_color_detecter(debug=True)

    def detect_color_box(self):
        pos=self.colordetecter.get_box_pos()
        print(pos)
        if pos and TARGET_BOX_COLOR in pos:
            for color in pos.keys():
                if color !=TARGET_BOX_COLOR :
                    return "left" if pos[color][0]>pos[TARGET_BOX_COLOR][0] else

        rospy.logerr("UNKNOW BOX COLOR!")
        return
```

四、执行顺序

👍
1. Roscore
2. 开vrep
3. 开 sudo bash bodyhub.sh ，开 ik_module
4. 开key_tele.py，需要tele卡死一次即可
5. 开key_cmd.py，调用public里的path_tracking（path_tracking融进key_cmd了）

- 修改Talos.lua

```
function torsoPosRotPublish()
```

```
    bodyMat = sim.getObjectMatrix(bodyTorso, -1)
    bodyQ = sim.getQuaternionFromMatrix(bodyMat)
    bodyEuler = sim.getEulerAnglesFromMatrix(bodyMat)
    bodyPos = sim.getObjectPosition(bodyTorso,-1)
    bodyPosRot = {bodyPos[1], bodyPos[2], bodyPos[3], bodyQ[4], bodyQ[1], bodyQ[
    bodyQ[3],bodyEuler[1],bodyEuler[2],bodyEuler[3]}
    simROS.publish(torsoPRPub, {data = bodyPosRot}) --torso position rotation
end
```

- 修改全部controlid
- 注释ikmodule.py中的import pyaudio

五．注意事项及报错解决

5.1 仿真内深度相机格式错误

- 原因：仿真内深度相机默认给的格式为16UC1，但是建图要求输入的图片格式为32FC1。
- 解决方法：

调用
\2022\normal_sim_game\ai_innovative_roban_sim\scripts\sim_image_convert_slam_form.py

将16UC1格式改为为32FC1

5.2 currentControlId 报错

- 原因：程序要求在一次程序运行中，bodyhub中调用的controlid相同。但是在官方例程中，ik_module里reset是给的controlid＝6，一般程序里最开始的control_id是2，sim_ikmodule里初始化controlid给的是4，导致出现报错“IkModule is busy with controlID”
- 解决方式：

或者搜索“IkModule is busy with controlID”

15行uint8_t currentControlId = 6, bodyhubControlId = 0;

改全部controlid为6

5.3 仿真环境超时

由于仿真环境中，消息和服务发布的频率与电脑处理效率有关。可能存在卡顿情况导致节点消息发布丢包，从而引发机器人抖动或者程序超时现象。

解决方式：

1. 将话题等待的超时判定设长

```
rospy.wait_for_service('MediumSize/BodyHub/armMode',100)
```

2. 尽量避免长时间运行仿真

5.4 避障例程存在问题

目前避障的策略为判断是否前面有物体，如果有物体，则旋转一定角度来躲避物体。实际上这样的避障算法十分不灵敏，容易使机器人撞到物体或者无法及时避开障碍物，且需要大量的手动调参。

使用SLAM方法可以从根本上解决避障问题。

六、心得体会

在机器人课设中，我学习了ROS操作系统控制机器人移动的基本原理和方法，通过实践巩固了所学知识，并锻炼了自己的能力。在这个过程中，我深刻认识到了机器人控制的重要性以及ROS在机器人控制中的优势。同时，通过这段经历，我深刻地认识到了官方代码文档的重要性。一个清晰、详细的文档可以帮助广大开发者更快地掌握机器人开发框架的使用方法，从而更加高效地开发应用程序。

通过ROS，我们可以方便地对机器人进行控制、通信和数据处理，使得机器人的开发更加高效、快速。然而，在机器人课设中，我也遇到了一些困难和挑战。尤其是在仿真环境下，由于环境复杂性和不稳定性的影响，机器人的移动轨迹往往不够理想，这给机器人的控制和调试带来了很大的挑战。因此，我深刻认识到了实物机器人的重要性。只有在实物机器人上进行测试，才能更真实地感受到机器人的运动状态和环境变化对机器人的影响，更好地掌握机器人的控制方法和调试技巧，从而更好地完成机器人的任务。

另外一个问题就是自己的电脑跑仿真很慢，需要每周到机房才能进行实验，这导致了我的进度不快，影响了实验的完成。建图模块实现了一部分，但由于实验环境缺失难以部署，因此和同伴分享了相关的方法，协助他们完成了八叉树的建图与导航。

在未来的学习中，我将继续深入学习ROS操作系统，不断探索机器人控制的方法和技巧，同时也会积极争取机会，尽可能地进行实物机器人测试，不断提高自己的实践能力和创新能力，为未来的机器人开发工作做好充分的准备。