北京郵電大學

Beijing University of Posts and Telecommunications

Webots 机器人仿真实验

实验报告

课 程 名 称：	智能交互机器人实验
姓　　名：	叶淳瑜
学　　号：	2019212686
学　　院：	计算机学院
专　　业：	智能科学与技术
班　　级：	2019211316

二〇二二年 四月

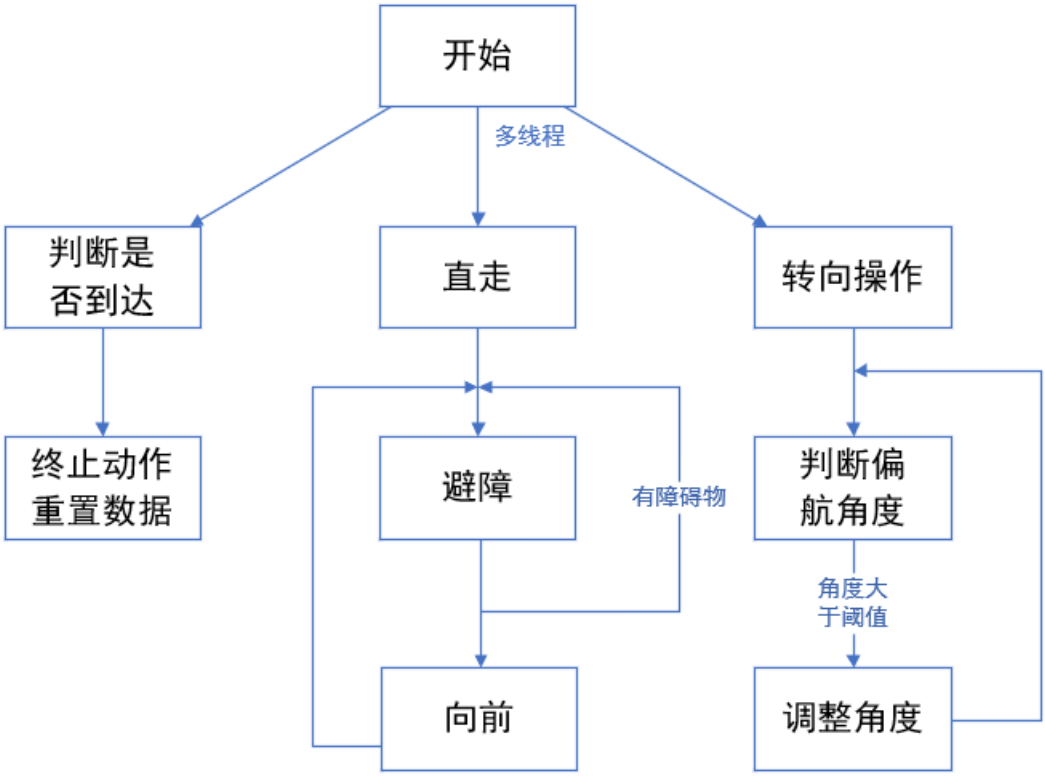
Webots仿真机器人控制

一. 功能描述与概要设计

首先使用Webots搭建展览馆场景，使用**Nao机器人**完成了**展品导览与避障**功能，用户可以通过键盘输入展品的序号，机器人就能指引用户前往目的地，并且用户可以在**中途更换**目的地，到达目的地后，机器人会用手指向展品。

机器人通过预设的场馆地图首先使用A*算法计算大致路线，然后在导览过程中使用动态调整的方法不断修正角度以及避开障碍物。

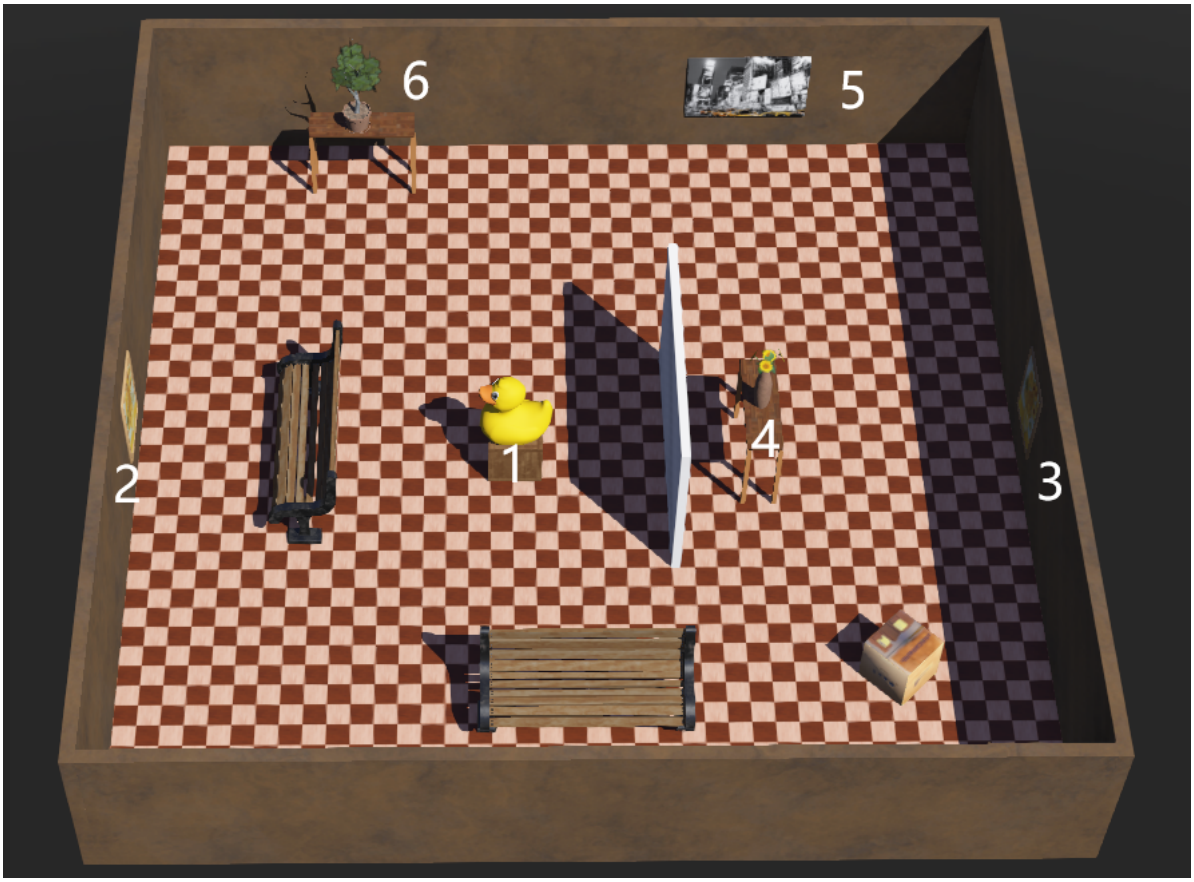
Nao机器人控制程序的流程图大致如下。



二. 详细设计

2.1 环境布局

仿真环境布局如下，共有3个实物展品，3幅平面展品。并设计了长凳、纸箱等障碍物。



2.2 数据结构

```
exhibitsGPS = [[0, 0.73], [0, 5], [0, -5],
               [0, -2], [-5, -2.2], [-4.35, 2.58]]  # GPS of exhibits
exhibitsPos = [[19, 0], [19, 15], [19, 39],
               [19, 27], [0, 9], [2, 28]]  # coordinate of exhibits for A*
search
posMap = []
```

2.3 关键函数

`findAndEnableDevices`、`loadMotionFiles`、`startMotion`等函数为官方demo代码中的函数，作用分别是加载设备、预设动作、执行动作，我在`startMotion`增加了执行动作之后的睡眠时间，使得机器人的执行更加稳定，不会摔倒。

`run`函数是机器人主体函数，逻辑十分简单。如果用户中途更改目的地，那么直接修改`is_arrived`就能停止前一个动作。

```
def run(self):
    print("Ready to go!")
    self.handWave.setLoop(True)
    self.handWave.play()

    # wait until a key is pressed
    prekey = None
```

```
    while robot.step(self.timeStep) != -1:  # retrun -1 when Webots terminates
the controller
        key = self.keyboard.getKey()
        if key > 0:
            break

    self.handWave.setLoop(False)

    while True:
        key = self.keyboard.getKey()

        if 48 < key < 56:
            exhibit = key - 49
            if prekey != key and not self.is_arrived:
                self.is_arrived = True
                self.currentlyPlaying = None
                print(f"Changing the destination to exhibit {exhibit + 1}...")
                time.sleep(3)
            else:
                print(f"Going to exhibit {exhibit + 1}...")

            if prekey == key:
                continue

            self.go(exhibit)
            prekey = key

        if robot.step(self.timeStep) == -1:
            break
```

`path_search`使用A*算法求解最佳路径。

```
def path_search(self, exhibit):
    curPos = self.gps.getValues()
    start_x = -curPos[2] / 0.25 + 20
    start_y = curPos[0]/0.25 + 20
    path = astar(self.posMap, (start_x,start_y),
tuple(self.exhibitsPos[exhibit]))
    return path
```

`go`函数，创建三个线程执行重要的三个功能函数，path的格式是[tuple, tuple...]

```
def go(self, exhibit):
    self.is_arrived = False

    Thread(target=self.goFoward).start()
    Thread(target=self.isArrived, args=(exhibit,)).start()
    Thread(target=self.turn, args=(exhibit,)).start()
```

`isArrived`函数用来判断机器人是否到达某个展品。

```
def isArrived(self, exhibit):
    while True:
        pos = self.gps.getValues()

        if (pos[0] - self.exhibitsPos[exhibit][0]) ** 2 + (pos[2] -
self.exhibitsPos[exhibit][1]) ** 2 < 1:
            if self.currentlyPlaying:
                self.currentlyPlaying.stop()

            print(f"Arrived at exhibit {exhibit + 1}")
            self.handWave.setLoop(True)
            self.handWave.play()
            self.handWave.setLoop(False)
            return
```

`goFoward` 函数实现前进以及避障功能。

```
def goFoward(self):
    while not self.is_arrived:
        if self.currentlyPlaying is None:
            while self.us[0].getValue() < 0.7:
                self.startMotion(self.sideStepRight)
            while self.us[1].getValue() < 0.7:
                self.startMotion(self.sideStepLeft)

            self.startMotion(self.forwards)
```

`turn` 和 `getAngle` 函数不断检测机器人与展品目的地之间的角度，一旦角度大于一个阈值就执行修正转向操作。

```
def turn(self, exhibit):
    while not self.is_arrived:
        angle = self.getAngle(exhibit)
        # print(angle)

        # modify the heading direction
        if abs(angle) > 0.5:
            if angle > 0:
                self.startMotion(self.turnLeft40)
            else:
                self.startMotion(self.turnRight40)

def getAngle(self, exhibit):
    """
    get angle between heading direction and destination
    :return: angle < 0 when exhibit on the left
    """
    curPos = self.gps.getValues()
    rpy = self.inertialUnit.getRollPitchYaw()
    yawAngle = -rpy[2]  # get yaw angle
    dest = self.exhibitsGPS[exhibit]
```

```
    deltaz = dest[1] - curPos[2]
    deltax = dest[0] - curPos[0]
    posAngle = math.atan(deltaz / deltax)
    # print(deltaz, deltax)

    if deltaz < 0 and deltax < 0:
        posAngle -= math.pi
    elif deltaz > 0 and deltax < 0:
        posAngle += math.pi

    # print(f"posangle:{posAngle},yawangle:{yawAngle}")
    angle = yawAngle - posAngle
    return angle
```

`pointFinger`函数通过控制Nao机器人的部件从而实现指向展品的动作。

```
def pointFinger(self, mode):
    self.LElbowRoll.setPosition(0)
    self.RElbowRoll.setPosition(0)
    if mode == 'point':
        self.RShoulderPitch.setPosition(0)
        for i in range(3, 6):
            self.rphalanx[i].setPosition(1)
    elif mode == 'reset':
        self.RShoulderPitch.setPosition(1.5)
        for i in range(3, 6):
            self.rphalanx[i].setPosition(0)
```

三. 使用方法

打开webots加载NAO_controller控制器，之后进行地图的加载，并提示按下1~7前往不同展品。

```
Map loaded.
Press 1 ~ 7 to start the Nao Robot.
Ready to go!
```

在键盘上按下数字键1~7即可前往指定目标地点，同时会在控制台上打印目标点。到达后，会在控制台打印到达信息，并且Nao机器人会挥手示意。在机器人前往某展品途中，按下其他展品的按钮，可以中断当前行程，开始前往新目的地。

四. 结果分析

4.1 实验结果

在Webots仿真平台上设计的NAO机器人可以较好地完成寻路任务，使用A*静态规划算法加上导览途中的动态避障算法。用户可以中途修改目的地，机器人到达后能够指向展品。总的来说完成了实验要求。

静态规划算法加上动态避障算法是本产品最大的创新点，它可以灵活实现寻路与避障功能。并且使用了多线程，能够一边前进，一边修正方向，一边检测是否到达。更多的，多线程实现一些细节，例如虽然每次Motion转动角度和前进距离是固定的，这就会导致有时我们只用走一点距离或转一点角度，但是必须完整执行Motion，多线程能够改善这一点，它不断检测目标从而及时停止。

4.2 遇到的问题及改进

当然，开发过程中也有不少问题，例如在多线程开发，本来想使用信号量机制实现互斥操作，但是发现并不能达到目标，因为执行动作的命令语句的确是互斥的，但是动作进行的过程中不是，这会导致多个命令同时操控机器人，于是会出现动作错乱的情况（机器人摔跤、执行奇怪的动作、不听掌控），最终大部分互斥访问都是使用的传统判断语句。

还有调节指向展品动作的过程是痛苦的，我不断在官网上摆弄Nao机器人，但是总是实现不了最好的“指向”姿势。最终我决定修改官方提供的一些不需要的Motion的动作细节，最终实现动作的良好执行。

还有一点是，由于A*算法是静态路径规划算法，如果地图中出现了新的未录入的障碍物，那A*算法就会失效，并且前期地图的录入要花费大量时间。对于新的障碍物，可以使用声纳进行检测，然后更新地图，从事保证A*算法的可用性。

五. 经验总结

官方文档：一定要仔细阅读官方文档，许多错误都能在官方文档中找出，Robot类的属性与方法也都可以找到官方的注释，并且提供的 `nao_demo_python` 文件提供了很大的帮助。

小心谨慎：操作机器人的逻辑程序十分复杂，有一些先后关系、互斥关系、资源共享都是需要特别注意的，必须要细心谨慎。

善用工具：使用Pycharm开发减少了不少麻烦，使用Webots IDE没有的debug功能排除了不少顽固bug。