计 算 机 应 用 技 术

实
验
报
告

姓名：杨敬轩

学号：SZ160310217

班级：机械二班

一、实验项目

1、组装以 Arduino 为主控的智能小车，完成小车的简单控制操作
2、学习与掌握典型的最短路径规划算法，实现给定地图下的最短路径搜索
3、调试小车的运动控制，完成给定地图下起点到终点的自主运动。

二、实验原理

1、路径规划算法原理（给出算法的流程图与简要说明）

(1) 简要说明：

A*算法是一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近，最终搜索速度越快。公式表示为：

$$f(n) = g(n) + h(n) \tag{1}$$

其中：$f(n)$——从初始状态经由状态 n 到目标状态的代价估计；

$g(n)$——在状态空间中从初始状态到状态 n 的实际代价；

$h(n)$——从状态 n 到目标状态的最佳路径的估计代价。

保证找到最短路径（最优解的）条件，关键在于估价函数$f(n)$的选取（或者说$h(n)$的选取）。以$d(n)$表达状态 n 到目标状态的距离，那么$h(n)$的选取大致有如下三种情况：

- 如果$h(n) < d(n)$，这种情况下，搜索的点数多，搜索范围大，效率低，但能得到最优解。
- 如果$h(n) = d(n)$，即距离估计$h(n)$等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。
- 如果$h(n) > d(n)$，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

A*算法的过程如下：

1. 把起点加入 open list；
2. 重复如下过程：
   - 遍历 open list，查找 F 值最小的节点，把它作为当前要处理的节点，然后移到 close list 中；
   - 对当前方格的 8 个相邻方格一一进行检查，如果它是不可抵达的或者它在 close list 中，忽略它。否则，做如下操作：
     - 如果它不在 open list 中，把它加入 open list，并且把当前方格设置为它的父节点；
     - 如果它已经在 open list 中，检查这条路径 (即经由当前方格到达它那里) 是否更近。如果更近，把它的父节点设置为当前方格，并重新计算它的 G 和 F 值。如果 open list 是按 F 值排序的话，改变后可能需要重新排序；
   - 遇到下面情况停止搜索：
     - 把终点加入到了 open list 中，此时路径已经找到了；
     - 查找终点失败，并且 open list 是空的，此时没有路径；
3. 从终点开始，每个方格沿着父节点移动直至起点，形成路径。

(2) A*算法流程图如图 1 所示。

2、算法的程序实现与结果（给出算法的源代码，给出已知地图下的路径搜索运行结果）

(1) 算法的 Python 源代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 20 23:59:42 2018

@author: Administrator
"""

import numpy
from heapq import heappush,heappop


def heuristic_cost_estimate(neighbor, goal):
    x = neighbor[0] - goal[0]
    y = neighbor[1] - goal[1]
    return abs(x) + abs(y)


def dist_between(a, b):
    return (b[0] - a[0]) ** 2 + (b[1] - a[1]) ** 2


def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    return path



# astar function returns a list of points (shortest path)
def astar(array, start, goal):

    directions = [(0,1),(0,-1),(1,0),(-1,0)] # 4 个方向

    close_set = set()
    came_from = {}
    gscore = {start:0}
    fscore = {start:heuristic_cost_estimate(start, goal)}

    openSet = []
heappush(openSet, (fscore[start], start))
                                    # 往堆中插入一条新的值

    # while openSet is not empty
    while openSet:
        # current := the node in openSet having the lowest
fScore value
        current = heappop(openSet)[1]
                                  # 从堆中弹出 fscore 最小的节点

        if current == goal:
            return reconstruct_path(came_from, current)

```

```
        close_set.add(current)

        for i, j in directions:
                        # 对当前节点的 8 个相邻节点一一进行检查
            neighbor = current[0] + i, current[1] + j

            ## 判断节点是否在地图范围内，并判断是否为障碍物
            if 0 <= neighbor[0] < array.shape[0]:
                if 0 <= neighbor[1] < array.shape[1]:
                    if array[neighbor[0]][neighbor[1]] == 1:
                                                    # 1 为障碍物
                        continue
                else:
                    # array bound y walls
                    continue
            else:
                # array bound x walls
                continue

            # Ignore the neighbor which is already evaluated.
            if neighbor in close_set:
                continue

            #  The distance from start to a neighbor via
current
            tentative_gScore = gscore[current] +
dist_between(current, neighbor)

            if  neighbor not in [i[1] for i in openSet]:
# Discover a new node
                heappush(openSet, (fscore.get(neighbor,
numpy.inf), neighbor))
            elif tentative_gScore >= gscore.get(neighbor,
numpy.inf):   # This is not a better path.
                continue

            # This path is the best until now. Record it!
            came_from[neighbor] = current
            gscore[neighbor] = tentative_gScore
            fscore[neighbor] = tentative_gScore +
heuristic_cost_estimate(neighbor, goal)

    return False

if   name   == "  main  ":
    """nmap = numpy.array([
        [0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,1,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0],
        [1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0],
        [1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0],
        [1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0],
        [1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0],
```

```
            [1,1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0]])"""#原始数据

    nmap = numpy.array([
        [0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0],
        [0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0],
        [0,0,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0],
        [1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0,0],
        [1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0],
        [1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0],
        [1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0],
        [1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0],
        [1,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0],
        [1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0],
        [1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0],
        [1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0]])#扩充障碍

    path = astar(nmap, (14,0), (0,18))

    for i in range(len(path)):
        nmap[path[i]] = 100
```

(2) 代码的运行结果如图 2 所示，其中黄色区域为初始设置的障碍物，粉色区域为防止小车碰撞障碍物扩充的虚拟障碍物，绿色区域为小车的行驶路径。

0	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	100	0
0	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	100	0
0	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	100	0
1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	100	0
1	1	1	1	0	100	100	100	100	100	100	100	100	1	1	1	1	1	100	0
1	1	1	1	0	100	1	1	1	1	1	1	100	100	100	100	100	100	100	0
1	1	1	1	0	100	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	100	1	1	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	0	100	1	1	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	100	1	1	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	100	0	0	0	0	0	0	1	1	1	1	1	0	0	0
1	1	1	1	1	100	0	0	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	100	0	0	1	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	100	0	0	1	1	1	1	1	0	0	0	0	0	0	0
100	100	100	100	100	100	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0

图 2　A*算法运行结果

三、实验实现

1、小车运动控制策略（给出小车运动控制的流程图与简要说明）

(1) 说明：首先设置硬件接口：左电机前进(IN1)接数字输入 6，左电机后退(IN2)接数字输入 9，右电机前进(IN3)接数字输入 10，右电机前进(IN4)接数字输入 11。此设置与默认设置不同，主要是考虑到每个电机的前进后退控制均需使用 PWM。小车前进则左右电机均前进，左转则左电机后退右电机前进，右转则左电机前进右电机后退，停止则左右电机均停止。运动控制的流程即根据实际地图进行路径规划，然后编写相应的程序。

(2) 流程图如图 3 所示。

2、小车的程序实现与结果（给出小车运动控制的源代码，并说明程序中各函数功能，以及参数的选择依据）

(1) 小车运动控制的源代码如下：

```
//============================================================
//  智能小车路径规划
//============================================================
//#include <Servo.h>
int Left_motor_go=6;       //左电机前进(IN1)
int Left_motor_back=9;     //左电机后退(IN2)
int Right_motor_go=10;     //右电机前进(IN3)
int Right_motor_back=11;   //右电机后退(IN4)

int T1 = 136;//left1
int T2 = 136;//right2
int T3 = 145;//left3

void setup()
{
  //初始化电机驱动 IO 为输出方式
  pinMode(Left_motor_go,OUTPUT);    // PIN 6 (PWM)
  pinMode(Left_motor_back,OUTPUT); // PIN 9 (PWM)
  pinMode(Right_motor_go,OUTPUT);   // PIN 10 (PWM)
  pinMode(Right_motor_back,OUTPUT);// PIN 11 (PWM)
}

void brake(int time)          //刹车，停车
{
  digitalWrite(Right_motor_go,LOW);
  digitalWrite(Right_motor_back,LOW);
  digitalWrite(Left_motor_go,LOW);
  digitalWrite(Left_motor_back,LOW);
  delay(time * 100);//执行时间，可以调整
}

void run (int time)      // 前进
{
  digitalWrite(Right_motor_go,HIGH);  // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,162);
                               //PWM 比例 0~255 调速，左右轮差异略增减
  analogWrite(Right_motor_back,0);
  digitalWrite(Left_motor_go,LOW);  // 左电机前进
  digitalWrite(Left_motor_back,HIGH);
  analogWrite(Left_motor_go,0);
                               //PWM 比例 0~255 调速，左右轮差异略增减
  analogWrite(Left_motor_back,175);
  delay(time * 100);    //执行时间，可以调整
}

void spin_left(int time)          //左转(左轮后退，右轮前进)
{
  digitalWrite(Right_motor_go,HIGH);     // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,T1);
  analogWrite(Right_motor_back,0);     //PWM 比例 0~255 调速
  digitalWrite(Left_motor_go,HIGH);    //左轮后退
  digitalWrite(Left_motor_back,LOW);
```

```
  analogWrite(Left_motor_go,T1);
  analogWrite(Left_motor_back,0);//PWM 比例 0~255 调速
  delay(time * 100);              //执行时间，可以调整
}

void spin_left2(int time)                //左转(左轮后退，右轮前进)
{
  digitalWrite(Right_motor_go,HIGH);    // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,T3);
  analogWrite(Right_motor_back,0);     //PWM 比例 0~255 调速
  digitalWrite(Left_motor_go,HIGH);    //左轮后退
  digitalWrite(Left_motor_back,LOW);
  analogWrite(Left_motor_go,T3);
  analogWrite(Left_motor_back,0);//PWM 比例 0~255 调速
  delay(time * 100);              //执行时间，可以调整
}

void spin_right(int time)               //右转(右轮后退，左轮前进)
{
  digitalWrite(Right_motor_go,LOW);    //右电机后退
  digitalWrite(Right_motor_back,HIGH);
  analogWrite(Right_motor_go,0);
  analogWrite(Right_motor_back,T2);//PWM 比例 0~255 调速
  digitalWrite(Left_motor_go,LOW); //左电机前进
  digitalWrite(Left_motor_back,HIGH);
  analogWrite(Left_motor_go,0);
  analogWrite(Left_motor_back,T2);//PWM 比例 0~255 调速
  delay(time * 100);               //执行时间，可以调整
}
void back(int time)           //后退
{
  digitalWrite(Right_motor_go,LOW);  //右轮后退
  digitalWrite(Right_motor_back,HIGH);
  analogWrite(Right_motor_go,0);
  analogWrite(Right_motor_back,100); //PWM 比例 0~255 调速
  digitalWrite(Left_motor_go,HIGH);  //左轮后退
  digitalWrite(Left_motor_back,LOW);
  analogWrite(Left_motor_go,100);
  analogWrite(Left_motor_back,0);//PWM 比例 0~255 调速
  delay(time * 100);              //执行时间，可以调整
}

void loop()
{
  int i;
  delay(1000);       //延时 1s 后启动
  run(8);            //前进 45
  spin_left(3);      //左转 90
  run(18);           //前进 100
  spin_right(3);     //右转 90
  run(25);           //前进 125
  spin_left2(3);     //左转 90
  run(12);           //前进 60
  brake(1000);
}
```

(2) 函数功能及参数选择依据：

- ❖ setup()函数：初始化电机驱动 IO 为输出方式；
- ❖ brake()函数：刹车、停止；
- ❖ run()函数：前进，由于左轮和右轮电机的灵敏度不同，所以 PWM 参数设置不同，参数选择方式为多次尝试累次优化；
- ❖ spin_left()函数：原地旋转，参数选择方式为多次尝试累次优化；
- ❖ spin_left2()函数：原地旋转，因为小车行走之后尾部的万向轮方向会偏转，所以原来设置的 spin_left()函数的参数不能够使小车精准旋转 90 度，故另设置此函数单独设置参数，参数选择方式仍为多次尝试累次优化；
- ❖ spin_right()函数：原地旋转，参数选择方式为多次尝试累次优化；
- ❖ back()函数：后退，本次实验并未用到此函数，故参数并未修改；
- ❖ loop()函数：主函数，根据实际路线编写函数，通过多次测试设置各个子函数的运行时间。

四、实验总结与建议

1、实验总结（总结实验实施过程，说明实验过程中遇到的问题与解决方案）

(1) 实施过程：

智能小车实验，首先组装小车，在这个过程中了解了电动机、舵机、Arduino 板等硬件设备；然后对小车进行调试，主要是熟悉小车的功能，并调试小车实现前进，直角转弯等基本操作；第三节课则进行 A*算法的学习和编写；最后，根据规划好的路线进行编程，多次测试修改参数，使小车按照地图顺利前行。

(2) 问题与解决方案：

- ➢ 上传程序出错

  解决方案：重启电脑或者换用其他电脑
- ➢ 设置好参数后，过一段时间小车无法精准旋转 90 度

  解决方案：电池电量低，及时充电即可
- ➢ 小车行进一段距离后旋转 90 度需要的参数与开始不同

  解决方案：增加子函数，单独控制此处旋转的参数

2、实验建议（对本课程的意见与建议）

首先，智能小车实验使我们对硬件接口，电气控制，编写程序等有了更深入的认识，相关技能也越来越熟练，是一门具有很强实践价值的课程。然而，这次实验最终实现的成果是不断尝试合适的参数得来的，很多传感器等可以利用的器件并未使用到，如果可以增加实验项目，将各种传感器加到本实验里，同学们的收获会更多。