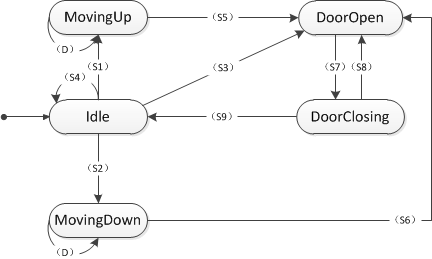
三层电梯课程设计报告



经分析，三层电梯一共有如上图的五个状态，分别为空闲状态（idle）、上升状态（movingup）、下降状态（movingdown）、开门状态（dooropen）、关门状态（doorclosing）。

## 一、Idle （电梯停止在某楼层，门是关闭的，处于静止状态，等待相关事件的发生，从而转换到下一个状态。）

### 1.从空闲状态到上升状态的条件为

**目标楼层大于所在楼层，操作为开电机上（SetMotorPower(1)），并跳到上升状态；**

### 2.从空闲状态到下降状态的条件为

**目标楼层小于所在楼层，操作为开电机下（SetMotorPower(-1)），并跳到下降状态；**

### 3.从空闲状态到开门状态的条件为

**1. 门外上升按钮开，执行操作为关闭门外上升按钮，开门，跳入Dooropen；**

**2. 门外下降按钮开，执行操作为关闭门外下降按钮，开门，跳入Dooropen；**

**3. 门内开门按钮开，执行操作为关闭门内开门按钮，开门，跳入Dooropen。**

### 4.从空闲到关门状态的条件为

**门内管门按钮开，执行操作为关闭门内关门按钮，跳入Doorclosing。**

## 二、Movingup

### 1.当坐标楼层与实际楼层差值小于0.01时

**关闭电机，开门，关闭门外开关门灯，关闭本层楼内本楼层按钮灯，跳入Dooropen。**

### 2.不是1的情况时

**即为上升过程中，此时检测门内开关门按钮，如果有开启情况，立即关闭门内按钮。**

## 三、Movingdown

### 1.当坐标楼层与实际楼层差值小于0.01时

**关闭电机，开门，关闭门外开关门灯，关闭本层楼内本楼层按钮灯，跳入Dooropen。**

### 2.不是1的情况时

**即为下降过程中，此时检测门内开关门按钮，如果有开启情况，立即关闭门内按钮。**

## 四、Dooropen

### 1.门内关门灯开启时

**关闭门内关门按钮，关门，跳到Doorclosing。**

### 2.判断门开启一定时间无操作时：2s（IsDoorOpen）

**关门，跳到Doorclosing。**

### 3.门内开门灯亮时

**关闭门内关门灯。**

## 五、Doorclosing

### 1. 门内开门灯开启时

**关闭门内开门按钮，开门，跳到Dooropen。**

### 2.门内关门灯亮时

**关闭门内关门灯**

### 3.门的下部红外线探测到物体时

**开门，跳到Dooropen。**

### 4.若门关闭一定时间后:10s

**熄灭门内开门按钮，跳到Idle.**

# 函数说明

GetCallLight(int floor, bool up)（获取电梯门外Up/Down呼叫按钮灯）

SetCallLight(int floor, bool up, bool s)（设置电梯门外Up/Down呼叫按钮灯）

GetOpenDoorLight（获取电梯门内开关门按钮灯）

SetOpenDoorLight(bool s)（设置电梯门内开关门按钮灯）（门内关门同理）

IsDoorOpen(int floor) IsDoorClosed(int floor)（获取开关门情况）

SetDoor(int floor, bool open)（设置开关门）

SetMotorPower(double power) （设置电机功率，power=1,上升；-1，下降；0，停止）

IsBeamBroken()（是否红外探测到遮挡物）

下页附主程序

↓↓↓↓↓↓↓↓↓

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "ElevatorLib.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Idle状态，电梯停止在某楼层，门是关闭的，处于静止状态，等待相关事件的发生，从而转换到下一个状态。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void StateIdle(int \*state)

{

int floor; bool up;

floor = IdleWhatFloorToGoTo(&up);

if (floor > 0 && up) {

SetMotorPower(1);

\*state = MovingUp;

}

if (floor > 0 && !up){

SetMotorPower(-1);

\*state = MovingDown;

}

if (GetCallLight(GetNearestFloor(), 1))

{

SetCallLight(GetNearestFloor(), 1, false);

SetDoor(GetNearestFloor(),true);

\*state = DoorOpen;

}

if (GetCallLight(GetNearestFloor(), 0))

{

SetCallLight(GetNearestFloor(), 0, false);

SetDoor(GetNearestFloor(), true);

\*state = DoorOpen;

}

if (GetOpenDoorLight()){

SetOpenDoorLight(false);

SetDoor(GetNearestFloor(), true);

\*state = DoorOpen;

}

if (GetCloseDoorLight()){

SetCloseDoorLight(false);

\*state = DoorClosing;

}

}

void StateMovingUp(int \*state)

{

int floor; bool up;

floor = GoingUpToFloor();

if (fabs(GetFloor() - floor) < Lib\_FloorTolerance)

{

SetMotorPower(0);

SetDoor(GetNearestFloor(), true);

SetMotorPower(0);

SetCallLight(GetNearestFloor(), false, false);

SetCallLight(GetNearestFloor(), true, false);

SetPanelFloorLight(GetNearestFloor(), false);

\*state = DoorOpen;

}

else {

if (GetOpenDoorLight() == true) { SetOpenDoorLight(false); }

if (GetCloseDoorLight() == true) { SetCloseDoorLight(false); }

}

}

void StateMovingDown(int \*state)

{

int floor; bool up;

floor = GoingDownToFloor();

if (fabs(GetFloor() - floor) < Lib\_FloorTolerance)

{

SetMotorPower(0);

SetCallLight(GetNearestFloor(), true, false);

SetCallLight(GetNearestFloor(), false, false);

SetDoor(GetNearestFloor(), true);

SetPanelFloorLight(GetNearestFloor(), false);

\*state = DoorOpen;

}

else {

if (GetOpenDoorLight() == true) { SetOpenDoorLight(false); }

if (GetCloseDoorLight() == true) { SetCloseDoorLight(false); }

}

}

void StateDoorOpen(int \*state)

{

if (GetCloseDoorLight()){

SetDoor(GetNearestFloor(),false);

SetCloseDoorLight(false);

\*state = DoorClosing;

}

if (IsDoorOpen(GetNearestFloor())) {

SetDoor(GetNearestFloor(), false);

\*state = DoorClosing;

}

if (GetOpenDoorLight()) {

SetOpenDoorLight(false);

}

}

void StateDoorClosing(int \*state)

{

if (GetOpenDoorLight()){

SetOpenDoorLight(false);

\*state = DoorOpen;

}

if (GetCloseDoorLight()){

SetCloseDoorLight(false);

}

if (IsBeamBroken()){

SetDoor(GetNearestFloor(), true);

\*state = DoorOpen;

}

if (IsDoorClosed(GetNearestFloor())){

SetOpenDoorLight(false);

\*state = Idle;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 状态机，每隔一定时间(如，100ms)被调用一次，采集系统的运行状态

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main\_control(int \*state)

{

if(IsElevatorRunning()) // 仿真正在运行

{

switch(\*state)

{

case Idle:

// Idle状态，一定时间无动作，自动到一楼

if(GetNearestFloor() !=1 ) {

AutoTo1Floor();

}

StateIdle(state);

break;

case MovingUp:

CancelTo1Floor(); // 其它状态，取消自动到一楼

StateMovingUp(state);

break;

case MovingDown:

CancelTo1Floor();

StateMovingDown(state);

break;

case DoorOpen:

CancelTo1Floor();

StateDoorOpen(state);

break;

case DoorClosing:

CancelTo1Floor();

StateDoorClosing(state);

break;

default:

printf("没有这种状态!!!\n");

}

}

}