南大双创嵌入式应用大作业报告

侯为栋 软件学院 191250045

实验目的

嵌入式系统（Embedded System），是一种嵌入机械或电气系统内部、具有专一功能和实时计算性能的计算机系统。

嵌入式计算机是面向应用、面向产品的、具有特定用途的计算机。它们没有刻意被制造成计算机的形态，往往以产品本身的形式展现在用户面前。

本次实验使用树莓派作为开发平台，AlphaBot 作为开发载体，Python 作为开发语言，引导学生学习体会嵌入式系统开发的基本技能。

实验原理

树莓派结构简单、体积小、耗电低，却拥有与普通计算机几乎相同的功能和性能，可以很方便地植入到各种应用系统中。此类单板计算机是典型的嵌入式系统的基础。

AlphaBot2 智能车开发套件，包含一个基板 AlphaBot2-Base 和一个适配板（AlphaBot2-Ar、AlphaBot2-Pi、AlphaBot2-PiZero 三者之一）。其结构稳定，集成度高，不用复杂的组装以及繁琐的接线，有助于快速学习嵌入式系统开发。

Python 有着大量支持嵌入式系统开发的第三方库，例如 RPi.GPIO 等。以 Python 作为开发语言，可以避免直接与底层交互，将主要精力放在高层抽象逻辑上。

本次实验使用 Python 的第三方库 RPi.GPIO 和 rpi_ws281x，实现小车的红外控制、led控制、蜂鸣器控制、超声波自动避障。

实验内容

源码见 github 仓库 pithon。

基本介绍

基于 Python 实现的智能小车系统。

小车开机自动启动系统，或者通过进入根目录运行：

```
python3 app.py
```

手动启动。

小车使用红外遥控器进行控制。

红外遥控器如下图所示：

自左向右，自上而下编号，将遥控器按钮编码为 0-20。

例如，第一排第三个编号为 2，第二排第三个编号为 5。

小车开机自动启动，进入 set_param 模式。

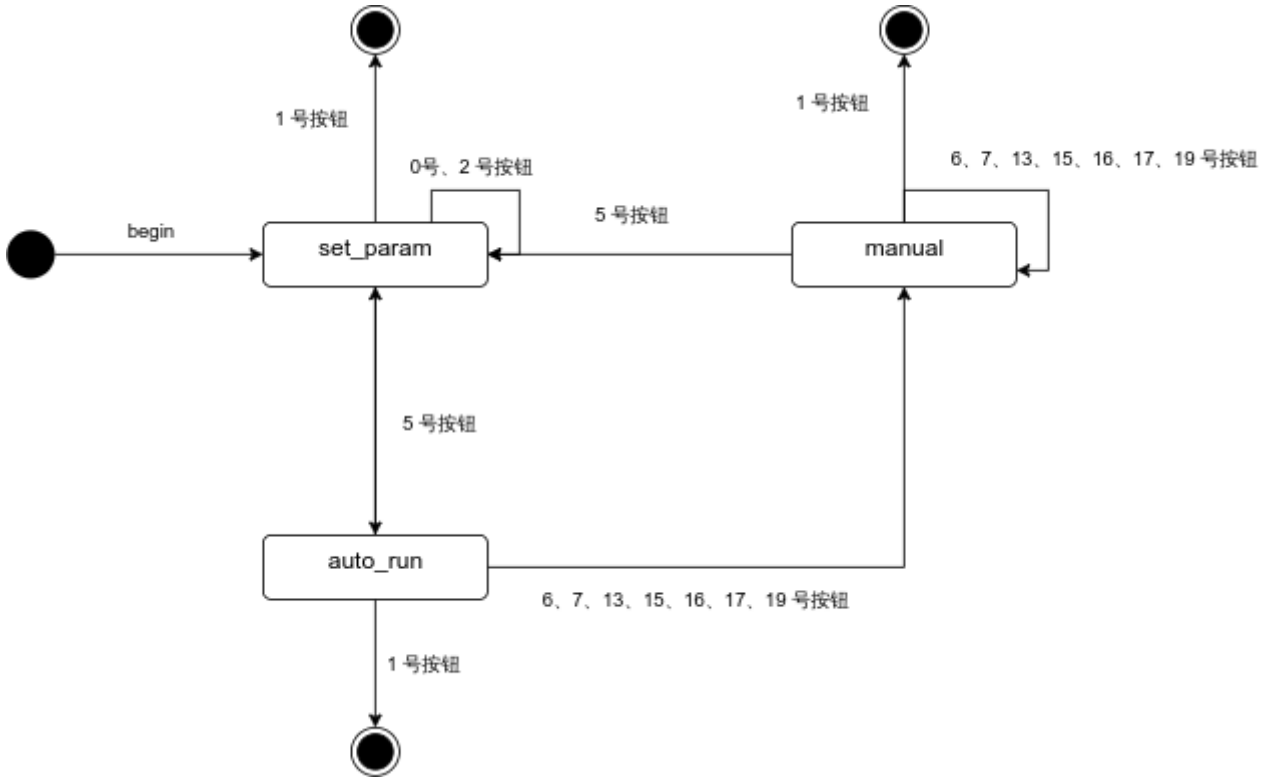
set_param 模式下，小车不移动。按 0 号按钮可以短暂启动 LED，按 2 号按纽可以启动蜂鸣器，按 5 号按钮进入 auto_run 模式。

auto_run 模式下，小车自动移动，并使用超声波测距来自动避障。按 5 号按钮进入 set_param 模式，按 6、7、13、15、16、17、19 号按钮进入 manual 模式。

manual 模式下，小车由遥控器手动控制。其中，6、7 号分别为加速、减速，13、15、16、17、19 号分别为前进、左转、停止、右转、后退。按 5 号按钮进入 set_param 模式。

在三种模式下，按 1 号按钮都会直接退出系统。

具体的状态图如下：



具体实现

系统入口为`app.py`。

`app.py`的任务是：初始化 GPIO channels，创建小车实例，开始监听，并在小车系统退出后清除 GPIO 设置。

```python
if __name__ == "__main__":
    init_gpio()
    car = Pithon()
    car.start()
    cleanup()
```

`init_gpio()`职责是在系统启动时**唯一**初始化 GPIO channels，源码在`util/gpio.py`中。

`Pithon`为对应小车的实体类，具体实现在`util/pithon.py`中。

```python
class Pithon:

    def __init__(self):
        # drive
        self.drive = AlphaBot2()
        # state
        self.state = State.set_param
        # pwm
        self.PWM = 50
        # buzzer
        self.beep = False
```

`self.drive`为掌控小车驱动的实体类，其实现在`lib/alphabot.py`中，提供了前进、后退、停止、左转、右转、加速、减速的驱动实现。

`self.state`为枚举类`State`的实例，具体实现在`util/state.py`中。一共三个属性`set_param`、`auto_run`、`manual`，分别对应三种状态。

`self.PWM`为小车的速度。

`self.beep`为小车蜂鸣器的开关。

该实例的启动函数为`self.start()`。职责为创建一条监听线程监听红外线信号。

```
def start(self):
    t = Thread(target=self.listen_wrapper)
    t.start()
    t.join()
```

`self.listen()`负责监听、处理红外信号。辅助函数为`get_key()`，实现在`util/infrared.py`中，有红外信号则返回信号，没有则返回`None`。

当没有红外信号时，小车保持当前状态。

```
def listen(self):
    key = get_key()
    if key is None:
        self.exec(0)
        return 0
```

```
def exec(self, key):
    if self.state == State.set_param:
        return
    if key == 0:
        if self.state == State.auto_run:
            dis = distance()
            print(dis)
            if dis <= 20:
                self.drive.right()
            else:
                self.drive.forward()
            return
```

如上图所示，如果是在`auto_run`模式下，小车会持续通过超声波获取距离，而后选择避障与否。如果实在`manual`模式下，会不作处理即保持当前运行状态。

超声波测距的实现在`util/ultrasonic.py`中。

若有红外信号，则根据当前状态进行有选择性的处理。

```
        print("key num: " + str(key))
        if key == 70:
            return 1
        if self.state == State.set_param:
            if key == 69:
                led()
                pass
            if key == 71:
                if self.beep is False:
                    beep_on()
                    self.beep = True
                else:
                    beep_off()
                    self.beep = False
            if key == 67:
                self.state = State.auto_run
                self.PWM = 15
                self.drive.setPWMA(self.PWM)
                self.drive.setPWMB(self.PWM)
        elif self.state == State.auto_run:
            if key == 67:
                self.state = State.set_param
                self.drive.stop()
            else:
                self.state = State.manual
                self.exec(key)
        else:
            print("manual")
            if key == 67:
                self.state = State.set_param
                self.drive.stop()
            else:
                self.exec(key)
        return 0
```

如上图所示，1 号按钮使函数返回 1，系统退出。

在`set_param`模式下，小车可以接收 0、1、2、5 号按钮的指令，进行相应的处理。

其中 0 号按钮控制 led，实现在`util/led.py`中，2 号按钮控制蜂鸣器，实现在`util/buzzer.py`中。

在`auto_run`模式下，只有 1、5 号按钮会改变小车状态。

在`manual`模式下，5 号按钮会改变小车状态，其余按钮在`self.exec()`函数中进行有选择性的处理。

```
    # 前进 后退 左转 右转 停止 加速 减速
    last = self.state
    self.state = State.manual
    if key == 0x18:      You, 11 hours ago
        self.drive.forward()
    elif key == 0x08:
        self.drive.left()
    elif key == 0x1c:
        self.drive.stop()
    elif key == 0x5a:
        self.drive.right()
    elif key == 0x52:
        self.drive.backward()
    elif key == 0x15:
        # speed up
        if self.PWM + 10 < 101:
            self.PWM += 10
            self.drive.setPWMA(self.PWM)
            self.drive.setPWMB(self.PWM)
            print(self.PWM)
    elif key == 0x07:
        # slow down
        if self.PWM - 10 > -1:
            self.PWM = self.PWM - 10
            self.drive.setPWMA(self.PWM)
            self.drive.setPWMB(self.PWM)
            print(self.PWM)
    else:
        self.state = last
```

更详细实现请自行阅读源码。

结果

小车能够较成功运行。

在`set_param`模式下，小车能够自由控制 led 和蜂鸣器，而在`auto_run`模式下，小车检测到障碍物过近时会自动转向，在`manual`模式下，小车能够被遥控器自由控制。

三种模式之间的切换较为灵敏。

运行结果见附录视频。

结论

本次实验较成功通过 Python 实践了嵌入式系统编程，较好地完成了实验目的。

附录

演示视频在bilibili上。