

# 手势识别

## 1. 学习目标





















本次课程我们主要学习使用树莓派和手势识别模块实现手势识别功能。

## 2. 课前准备



手势识别模块采用的是 I2C 通讯，将模块的 SDA，SCL 分别连接树莓派板子的 SDA 和 SCL 引脚。VCC 和 GND 分别连接树莓派的 5V 和 GND。树莓派需要开启 I2C 服务。

### 引脚定义

Raspberry Pi GPIO Header + PoE Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1, I²C)		DC Power 5v	04
05	GPIO03 (SCL1, I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

01	TR01		TR00	02
03	TR03		TR02	04

树莓派 I2C 开启后终端输入 `lmusb`，查看 I2C 是否成功启动

```
videobuf2_dma_contig    20480    1 bcm2835_codec
videobuf2_vmalloc       16384    1 bcm2835_v4l2
videobuf2_memops        16384    2 videobuf2_dma_contig,videobuf2_vmal
videobuf2_v4l2          24576    3 bcm2835_codec,bcm2835_v4l2,v4l2_mem
videobuf2_common        45056    4 bcm2835_codec,bcm2835_v4l2,v4l2_mem
videodev                200704    6 bcm2835_codec,v4l2_common,videobuf2
media                   36864    2 videodev,v4l2_mem2mem
argmon_mem              16384    0
uio_pdrv_genirq         16384    0
uio                     20480    1 uio_pdrv_genirq
fixed                   16384    0
i2c_dev                 16384    0
i2c_bcm2708             16384    0
snd_bcm2835             24576    2
snd_pcm                 102400    1 snd_bcm2835
snd_timer               32768    1 snd_pcm
snd                     73728    7 snd_timer,snd_bcm2835,snd_pcm
ip_tables               24576    0
x_tables                32768    1 ip_tables
ipv6                    450560    26
pi@raspberrypi:~/speech $
```

下载 `i2c-tools`，该软件可以对硬件设备的使用情况、故障进行监控

终端输入 `sudo apt-get install i2c-tools`

通过终端输入 `i2cdetect -y -a 1` 查看是否有检测到模块的地址：`0x73`

### 3. 程序

本次课程的程序请参考：`PAJ7620U2.py`

定义模块的设备地址和寄存器地址便于后续操作

```

#I2C地址
PAJ7620U2_I2C_ADDRESS = 0x73
#寄存器Bank选择
PAJ_BANK_SELECT = 0xEF #Bank0== 0x00,Bank1== 0x01
#寄存器Bank 0
PAJ_SUSPEND = 0x03 #I2C suspend命令(写= 0x01进入挂起状态)。I2C唤醒命令是奴隶ID唤醒。参考主题“I2C总线定时特性和协议”
PAJ_INT_FLAG1_MASK = 0x41 #手势检测中断标志掩码
PAJ_INT_FLAG2_MASK = 0x42 #手势/PS检测中断标志掩码
PAJ_INT_FLAG1 = 0x43 #手势检测中断标志
PAJ_INT_FLAG2 = 0x44 #手势/PS检测中断标志
PAJ_STATE = 0x45 #手势检测状态指示灯(仅在手势检测模式下起作用)
PAJ_PS_HIGH_THRESHOLD = 0x69 #PS迟滞高阈值(仅在接近检测模式下起作用)
PAJ_PS_LOW_THRESHOLD = 0x6A #PS迟滞低阈值(仅在接近检测模式下起作用)
PAJ_PS_APPROACH_STATE = 0x6B #PS趋近状态, 趋近= 1, (8位PS数据>= PS高门限), 不趋近= 0, (8位PS数据<= PS低门限)(仅在接近检测模式下有效)
PAJ_PS_DATA = 0x6C #PS 8位数据(仅在手势检测模式下有效)
PAJ_OBJ_BRIGHTNESS = 0x80 #物体亮度(最大255)
PAJ_OBJ_SIZE_L = 0x81 #对象大小(低8位)
PAJ_OBJ_SIZE_H = 0x82 #对象大小(高8位)
#寄存器Bank 1
PAJ_PS_GAIN = 0x44 #PS增益设置(仅在接近检测模式下有效)
PAJ_IDLE_S1_STEP_L = 0x67 #空闲S1步长, 用于设置S1, 响应系数(低8位)
PAJ_IDLE_S1_STEP_H = 0x68 #空闲S1步长, 用于设置S1, 响应系数(高8位)
PAJ_IDLE_S2_STEP_L = 0x69 #空闲S2步长, 用于设置S2, 响应因子(低8位)
PAJ_IDLE_S2_STEP_H = 0x6A #空闲S2步长, 用于设置S2, 响应因子(高8位)
PAJ_OPTOS1_TIME_L = 0x6B #OPTOS1 Step, 用于将操作状态的OPTOS1时间设置为待机1状态(低8位)
PAJ_OPTOS2_TIME_H = 0x6C #OPTOS1 Step, 用于将OPTOS1运行状态时间设置为待机1 stateHigh 8位)
PAJ_S1TOS2_TIME_L = 0x6D #S1toS2步, 用于将待机1状态的S1toS2时间设置为待机2状态(低8位)
PAJ_S1TOS2_TIME_H = 0x6E #S1toS2步, 用于将待机1状态的S1toS2时间设置为待机2状态高8位)
PAJ_EN = 0x72 #启用/禁用PAJ7620U2
#手势检测中断标志
PAJ_RIGHT = 0x01
PAJ_LEFT = 0x02
PAJ_UP = 0x04
PAJ_DOWN = 0x08
PAJ_FORWARD = 0x10
PAJ_BACKWARD = 0x20
PAJ_CLOCKWISE = 0x40
PAJ_COUNT_CLOCKWISE = 0x80
PAJ_WAVE = 0x100

```

定义初始化阵列、寄存器数组、手势寄存器地址。

```

#启动初始化阵列
Init_Register_Array = (
    (0xEF,0x00),
    (0x37,0x07),
    (0x38,0x17),
    (0x39,0x06),
    (0x41,0x00),
    (0x42,0x00),
    (0x46,0x2D),
    (0x47,0x0F),
    (0x48,0x3C),|
    (0x49,0x00),
    (0x4A,0x1E),|
    (0x4C,0x20),|
    (0x51,0x10),
    (0x5E,0x10),
    (0x60,0x27),
    (0x80,0x42),

```

```
#寄存器初始化数组
```

```
Init_PS_Array = (  
    (0xEF, 0x00),  
    (0x41, 0x00),  
    (0x42, 0x00),  
    (0x48, 0x3C),  
    (0x49, 0x00),  
    (0x51, 0x13),  
    (0x83, 0x20),  
    (0x84, 0x20),  
    (0x85, 0x00),  
    (0x86, 0x10),  
    (0x87, 0x00),  
    (0x88, 0x05),  
    (0x89, 0x18),  
    (0x8A, 0x10),  
    (0x9F, 0xF8),  
);
```

```
#手势寄存器初始化数组
```

```
Init_Gesture_Array = (  
    (0xEF, 0x00),  
    (0x41, 0x00),  
    (0x42, 0x00),  
    (0xEF, 0x00),  
    (0x48, 0x3C),  
    (0x49, 0x00),  
    (0x51, 0x10),  
    (0x83, 0x20),  
    (0x9F, 0xF9),  
    (0xEF, 0x01),  
    (0x01, 0x1E),  
    (0x02, 0x0F),  
);
```

通过 I2C 将启动初始化数组的值和手势寄存器初始化数组的值写入对应的寄存器来启动和初始化手势识别模块。



```
def __init__(self, address=PAJ7620U2_I2C_ADDRESS):
    self._address = address
    self._bus = smbus.SMBus(1)
    time.sleep(0.5)
    if self._read_byte(0x00) == 0x20:
        print("\nGesture Sensor OK\n")
        for num in range(len(Init_Register_Array)):
            self._write_byte(Init_Register_Array[num][0], Init_Register_Array[num][1])
    else:
        print("\nGesture Sensor Error\n")
    self._write_byte(PAJ_BANK_SELECT, 0)
    for num in range(len(Init_Gesture_Array)):
        self._write_byte(Init_Gesture_Array[num][0], Init_Gesture_Array[num][1])
```

手势识别函数，通过读取手势识别存放寄存器的值判断当前识别到的手势并打印出相应的手势名。

```
def check_gesture(self):
    Gesture_Data=self._read_u16(PAJ_INT_FLAG1)
    if Gesture_Data == PAJ_UP:
        print("Up\r\n")
    elif Gesture_Data == PAJ_DOWN:
        print("Down\r\n")
    elif Gesture_Data == PAJ_LEFT:
        print("Left\r\n")
    elif Gesture_Data == PAJ_RIGHT:
        print("Right\r\n")
    elif Gesture_Data == PAJ_FORWARD:
        print("Forward\r\n")
    elif Gesture_Data == PAJ_BACKWARD:
        print("Backward\r\n")
    elif Gesture_Data == PAJ_CLOCKWISE:
        print("Clockwise\r\n")
    elif Gesture_Data == PAJ_COUNT_CLOCKWISE:
        print("AntiClockwise\r\n")
    elif Gesture_Data == PAJ_WAVE:
        print("Wave\r\n")
    return Gesture_Data
```

初始化成功后，通过循环调用手势识别函数来对当前手势进行判断。

```
if __name__ == '__main__':  
    import time  
  
    print("\nGesture Sensor Test Program ...\n")  
  
    paj7620u2=PAJ7620U2()  
  
    while True:  
        time.sleep(0.05)  
        paj7620u2.check_gesture()
```

#### 4. 运行程序

终端输入 `python3 PAJ7620U2.py` 运行程序。

#### 5. 实验现象

程序运行以后，模块初始化成功则打印“**Gesture Sensor OK**”，否则打印“**Gesture Sensor Error**”。如果初始化失败可以重新运行下程序。初始化成功后，开始对手势识别的值进行判断，不同的手势会通过串口打印出不同的动作名。将手势识别模块放正，将手掌展开面向模块，从模块的正前方从左往右划过，则打印“**Left**”。从模块的正前方从右往左划过，则打印“**Right**”。从模块的正前方从下往上划过，则打印“**Up**”。从模块的正前方从上往下划过，则打印“**Down**”。从模块的正前方从后往前靠近，则打印“**Forward**”。从模块的正前方从前往后远离，则打印“**Backward**”。握拳伸出两 three 根手指指向模块的正前方，然后顺时针绕一小会,则打印“**Clockwise**”。握拳伸出两 three 根手指指向模块的正前方，然后逆时针绕一小会,则打印“**AntiClockwise**”。在模块的正前方挥手一会，则打印“**Wave**”。