第六周实验进度

王一飞 张国康 2022-03-31

Arduino RC Receiver与中断实验

实验预习

1 RC信号控制原理

RC即遥控（Remote Control)。它被广泛用于我们生产生活的各个领域中。一般来说，RC可以指多种信号所进行的遥控，但本实验中所说的RC指通过遥控器进行2.4GHz无线电通讯的遥控。

完整的RC控制需要一个遥控器（Controller）和一个接收器（Receiver）。图1，2是典型的七通道遥控器与接收器，本实验中使用Microzone MC6C型遥控器和Microzone MC7RB型接收器。下面仅讨论前4个通道（摇杆）的信号处理。

图1 RC遥控器

图2 RC接收器

通道是RC的一个重要概念，不同的通道对应遥控器上不同的操纵杆，其信号则可以从接收器的不同引脚测出。不同通道的信号相互独立，互不影响，可以实现对不同部件的分别控制。前4个通道与操纵杆的对应关系如图3所示.

图3 前4通道对应操纵杆

在使用前，遥控器和接收器需要进行对频（即配对）。配对后，RC接收器将遥控器的控制信号转换为一段脉冲，脉冲长度代表控制信号的大小。对于本实验使用的接收器和遥控器，脉冲长度的范围是$1000-2000\mu s$，频率是$250Hz$。通过读取这段脉冲的长度，就可以直到此时控制器上的信号大小。

2 Arduino中断控制

Arduino本身并不提供多任务处理功能，只能执行一个`loop`循环函数。但实际应用中时常需要检测一个不能预测出现时间的信号，若不等待该信号，则它很可能被忽略；但若等待该信号，又将使整个`loop`函数原地踏步而不能继续工作。为了解决这个问题，Arduino提供了外部中断功能。

简单的来说，Arduino的中断功能需要指定一个中断引脚，一个中断函数和某种信号变化。当中断引脚上的信号发生指定变化时，Arduino就会从正在执行的位置立即停下来，转而执行中断函数，完成后再回到刚才的位置继续工作。具体地讲，就是需要在`setup`函数中添加以下语句。

```
attachInterrupt(digitalPinToInterrupt(interruptPin), isr, mode);
```

其中`interruptPin`就是中断引脚的编号，`isr`就是中断函数的名称（即Interrupt Service Routine），`mode`则是信号变化的模式，具体来说，信号变化可以是以下几种：

- `LOW` 当中断引脚处于低电平时触发
- `CHANGE` 当中断引脚电平改变时触发
- `RISING`当中断引脚从低电平变为高电平（上升沿）时触发
- `FALLING`当中断引脚从高电平变为低电平（下降沿）时触发 对基于ARM芯片的Arduino（Due,Zero 和 MKR1000），还可以使用`HIGH` 表示当中断所在 Pin 口处于高电平时触发

对于一般的使用，这种中断方式已经足够。但对于Arduino Uno而言，只有2个引脚可以使用上述的中断功能进行控制，即D2，D3。对于更多引脚的中断，它以无能为力。因此，需要使用更高级的中断方式，即ATMEL Mega328P 所提供的引脚变化中断（Pin Change Interrupt）。利用这种方式，Arduino上的所有数字和模拟 I/O 口都可以使用中断。

为使用这种中断，首先需要了解 Mega328p中的两种寄存器：PCICR (Pin Change Interrupt Control Register)和PCMSK (Pin Change Enable Mask Register)。Mega328p的详细寄存器信息可以在技术手册中找到。在PCICR中，Arduino Uno的引脚被分为三组，分别是PCIE0, PCIE1, PCIE2 (PCIE, Pin Change Interrupt Enable Bit)，它们在PCICR中的结构和对应引脚如表1所示。

Bit	7	6	5	4	3	2	1	0
PICE						PCIE2	PCIE1	PCIE0
引脚						D0-D7	A0-A5	D8-D13

表1 PCICR结构

当PCICR的后三位中某一位被设为1时，这一组引脚就开启了中断功能。进一步，要明确其中这一组中的哪个引脚，则需要修改PCMSK。这3组引脚分别对应3个PCMSK，即PCMSK0，PCMSK1，PCMSK2。以PCMSK0为例，其结构如表2所示。

Bit	7	6	5	4	3	2	1	0
PCINT编号			PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
引脚			D13	D12	D11	D10	D9	D8

表2 PCMSK0结构

要启用特定的引脚进行中断，只需要将对应的位设为1即可。这其中PCINT就是中断引脚的编号方式。PCINT编号与IDE中编号的关系见图4。

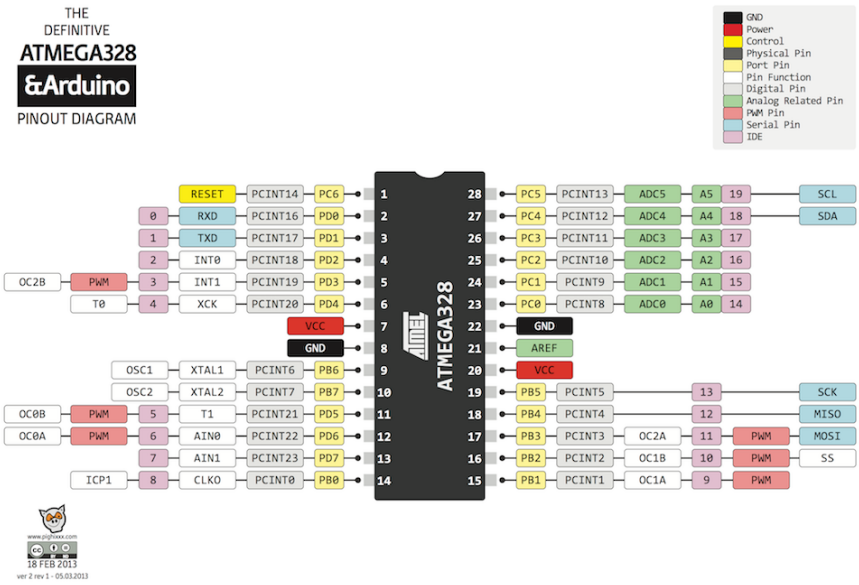


图4 Mega328p引脚图

以D8引脚为例，操作中，要启用其作为 Pin Change Interrupt引脚，需要在`setup`函数中添加以下语句。

```
PCICR |= 0b00000001;
PCMSK0 |= 0b00000001;
```

一种更方便的方法是使用

```
PCICR |= (1 << PCIE0);
PCMSK0 |= (1 << PCINT0);
```

若要设置PCMSK0中的其它引脚，只需要用第二行代码重复设定PCMSK0即可。

进一步，每一组PCIE引脚只能激活同一个中断函数。同样以D8（属于PCIE0）为例，其中断函数写法为

```
ISR(PCINT0_vect){
    // do something
}
```

其中`PCINT0_vect`代表这个中断函数接收的是PCIE0的中断信号，对其他引脚组，把0改成1或2即可。

实验过程

1 RC信号接收

尝试不用中断获取RC信号。定义变量`last_v`来存储上一次测量时的引脚信号（高或低），`start_time`存储本次脉冲信号上升沿的时刻，`now_time`存储当前时间，`pulseWid`存储脉冲长度。当引脚信号变化时，若此时电压为高且上一次电压信号为低，则记录上升沿时刻；若此时信号为低且上一次电压信号为高，则计算脉冲长度。如图5所示。

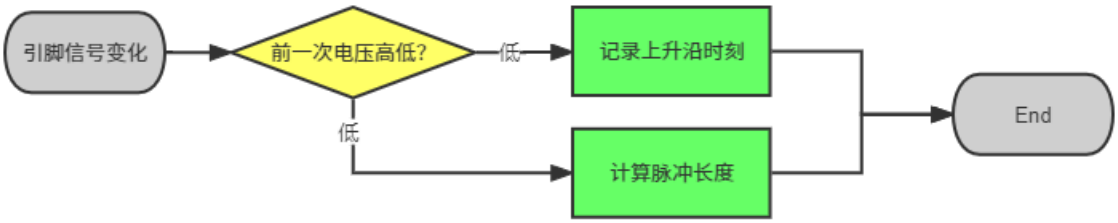


图5 程序流程图

具体实现，`loop`函数中的代码如下（`rc-signal.ino`）

```
void loop(){
    if( digitalRead(rcPin) != last_v){  // if signal change
        now_time = micros();  // record previous time
        if(last_v == 0){
            start_time = now_time;  // record the pulse start time
            last_v = 1;
        }
        else{
            pulseWid = now_time - start_time;  // calculate the pulse width
            last_v = 0;
            Serial.println(pulseWid);
        }
    }
}
```

打开串口绘图器，摇动操纵杆（右侧操纵杆上下移动），观察到如下波形（数值表示脉冲长度）

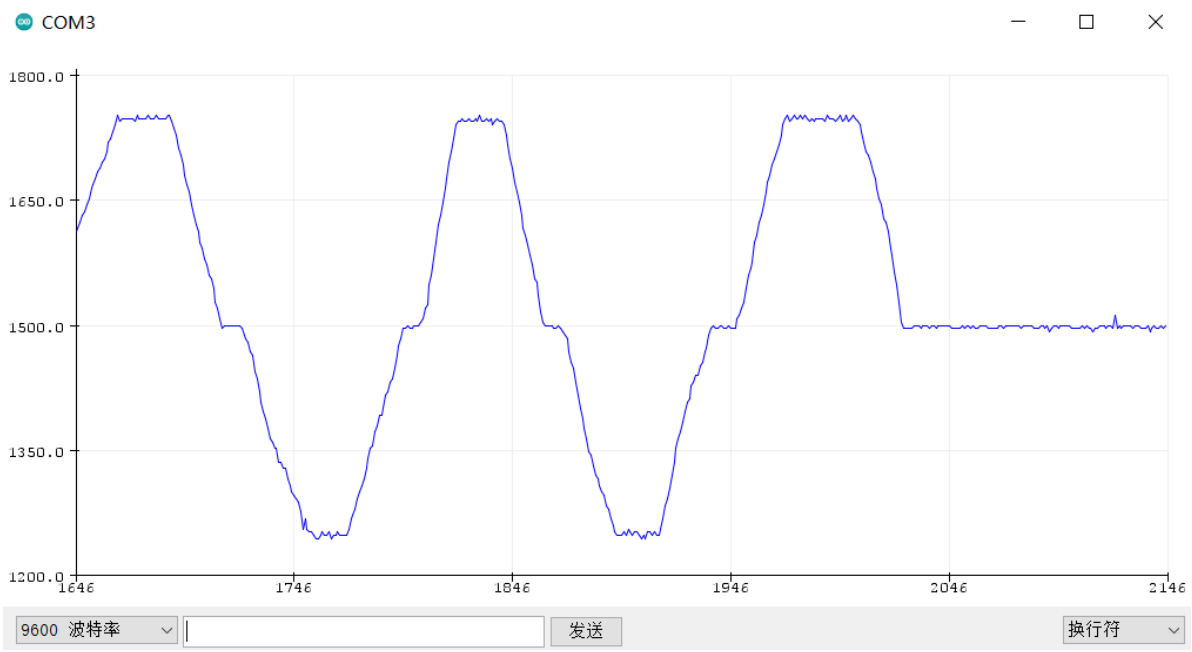


图6 RC信号波形

观察到信号范围在1250-1750之间而非1000-2000，这是因为打开了遥控器左上角的小舵开关。进一步观察到在操纵杆保持不动时，信号有小的涨落，在下面的实验中，我们将讨论这个涨落的来源和规避它的方法。总得来看，信号稳定性还是较好的。

2 RC信号控制舵机

接下来，尝试用RC信号控制舵机的角度。在`loop`函数的末尾加上以下命令(`rc-servo.ino`)

```
pulseWid_filter = pulseWid / 8;  // try to filter the signal
myservo.write(map(pulseWid_filter,156,218,0,180));  // write the angle
```

第一行代码是为了尽可能减小信号涨落对舵机的影响。观察串口绘图器

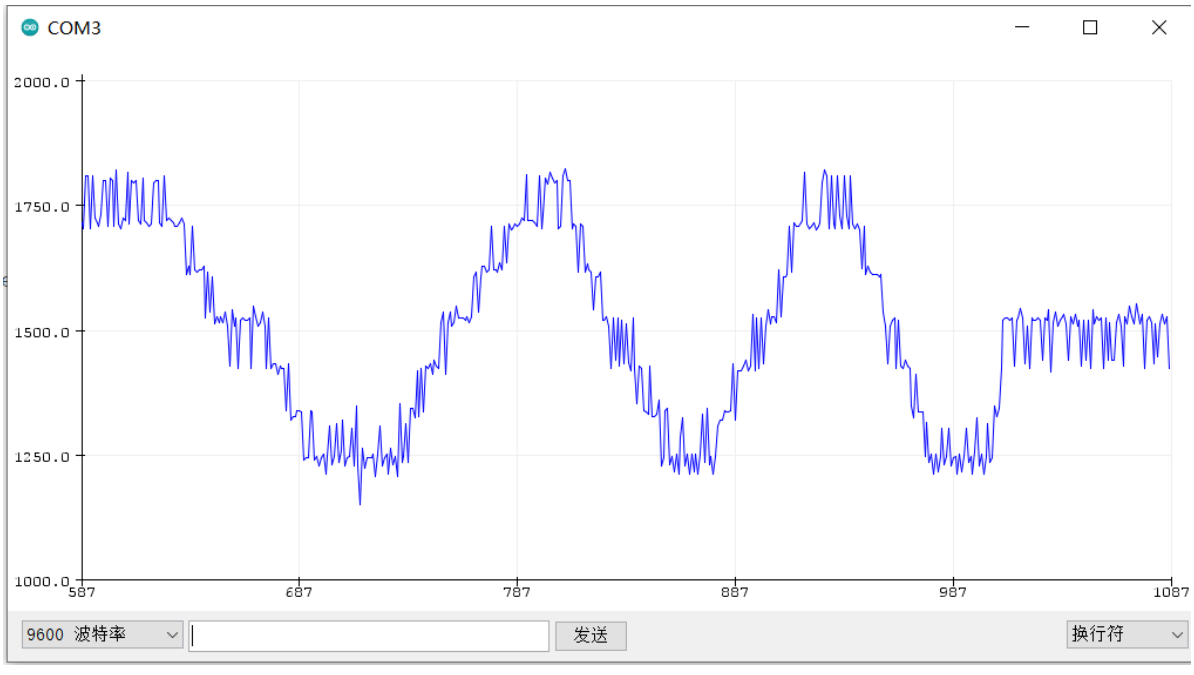


图7 控制舵机时RC信号

可以看到信号涨落比前一个实验大大加剧了。实验现象也证实了这一点，在操纵杆保持不动时，舵机也不断抖动。这使得控制舵机变得异常困难。分析问题的原因。不难想到，因此检测信号的操作与驱动舵机的操作在同一个循环中执行，而操作舵机需要消耗比较长的时间，因此当程序运行到检测信号的步骤时，脉冲可能早已开始或结束，这就会使得测得的脉冲比实际上更短或更长，造成较大的涨落。解决这个问题的方法就是使用中断进行控制。

3 RC信号中断控制舵机

我们按照实验预习中讲过的方法设置中断引脚为D8。原来程序中对引脚信号变化的判断因为中断的存在可以略去。但为了实现对多引脚信号的读取，我们仍然保留这个判断。并将其修改为以下的形式

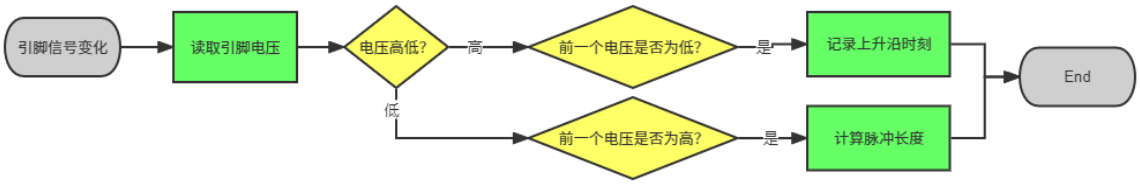


图8 中断函数流程图

ISR的具体写法为（`rc-servo-interrupt.ino`）

```
ISR(PCINT0_vect){  // Interrupt Service Routine for PCIE0
  now_time = micros();  // record previous time
  if( PINB & 0b00000001 ){  // if D8 is 1
      if(last_v == 0){      // and changed from 0 to 1 (rising edge)
          start_time = now_time;  // record the pulse start time
          last_v = 1;
    }
  }
  else if(last_v == 1){  // if D8 is 0 and changed from 1 to 0 (falling edge)
      pulseWid = now_time - start_time;  // calculate the pulse width
      last_v = 0;
  }
}
```

这其中`PINB & 0b00000001`等价于`digitalRead(8) == 1`，但前者由于直接调用Mega328p的寄存器，速度更快，利于对信号的捕捉。

`loop`中的命令如下

```
void loop(){
    myservo.write(map(pulseWid,1248,1748,0,180));  // write servo angle
    Serial.println(pulseWid);
    delay(100);
}
```

这样得到的的信号如图8。

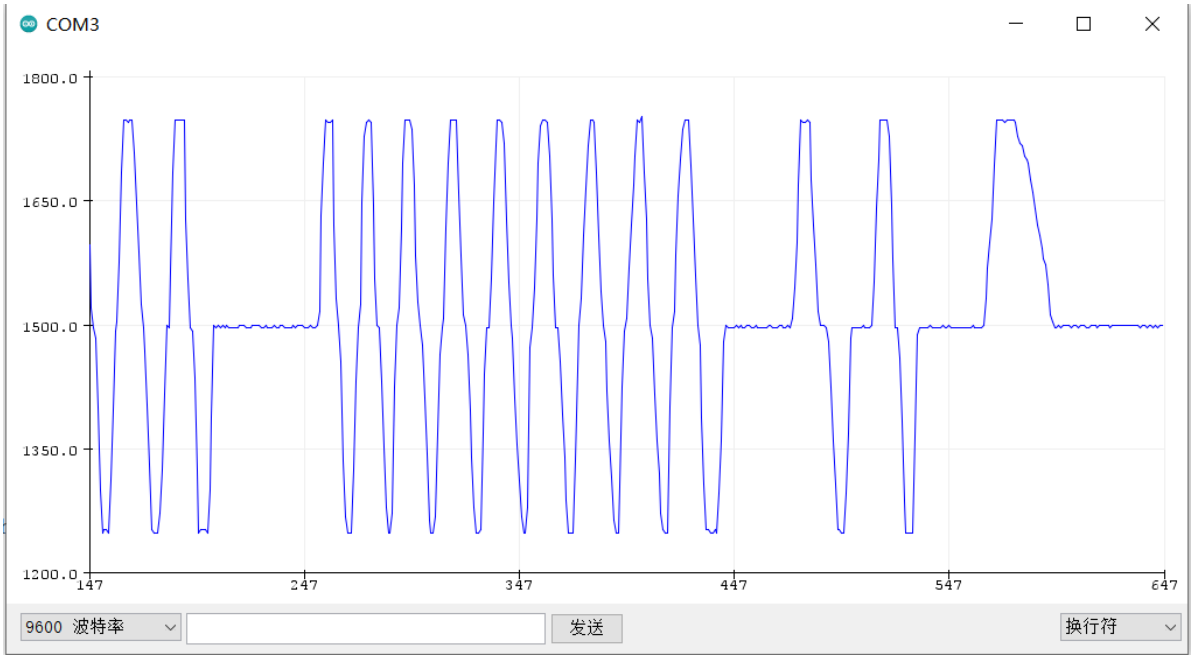


图9 中断控制RC信号图

可以发现信号的涨落大大减小，稳定性有了明显提升。在操纵杆保持不动时，舵机也不运动，控制效果令人满意。

多通道信号测量的方法与单通道类似，源代码为`rc-multi-channel.ino`，串口监视器中可以同时观察到两路信号的变化，如图10所示。

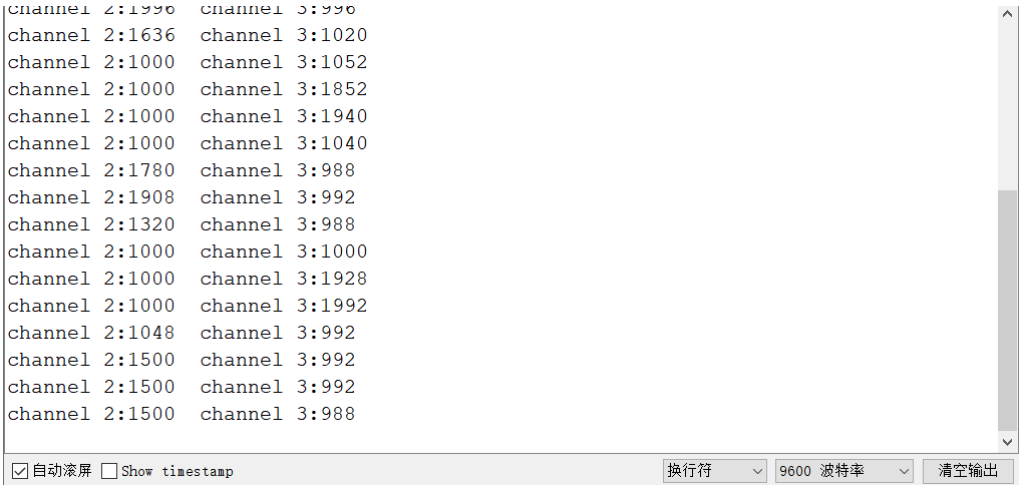


图10 RC信号双通道接收

实验结论

RC信号可以通过读取接收器的脉冲长度来转化为数值。在`loop`函数比较复杂，耗时较多的情况下，为了保证RC控制的有效性，必须采用中断。

ESC控制

关于ESC控制电机与电流方向的原理，在背景调研文档中已有介绍。下面主要说明如何通过Arduino控制ESC从而对直流无刷电机进行调速。

本项目使用 Simonk 30A ESC，它有一对电源正负极接口，5V、信号和GND接口，以及三相输出接口，如图11。

图11 ESC接口

对ESC的控制类似控制LED的亮度，即通过脉冲宽度调制输出信号，脉冲长度代表信号大小，ESC把脉冲长度转换为电机的转速。但由于Arduino 默认的PWM脉冲频率为500 Hz，而此ESC所能适配的也仅30-500Hz。其次，由于ESC的最大信号对应$2000\mu s$，在500Hz下无法分辨$\geq 2000\mu s$的信号，因此会影响精度。所以采用手动控制脉冲宽度的方式，在250Hz的频率下工作。

手动控制脉冲宽度的方法为：先将ESC信号引脚设为高，并记录这个上升沿对应的时刻；接着不断循环获取当前时间，计算已经发出的脉冲宽度；直到已发出的脉冲宽度大于信号长度，将ESC控制引脚调低。流程图如图12所示。

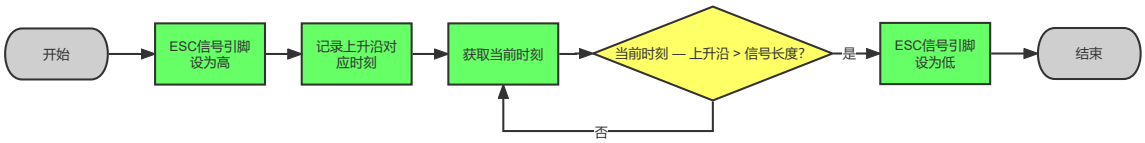


图12 ESC控制程序流程

要同时控制连接在4，5，6，7引脚上的ESC，转化为代码（来自`YMFC-AL_esc_calibrate.ino`）

```
zero_timer = micros();  // remember rising edge
PORTD |= B11110000;  // set 4,5,6,7 to high

timer_channel_1 = esc_1 + zero_timer;
timer_channel_2 = esc_2 + zero_timer;
timer_channel_3 = esc_3 + zero_timer;
timer_channel_4 = esc_4 + zero_timer;   // calculate the falling edge

while(PORTD >= 16){  // loop until all pins are set to low
    esc_loop_timer = micros();  // check the current time.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;  // set the pin to
low if  get to falling edge
}
```

MPU 6050控制

MPU6050是一款功能强大，方便易用的角速度和加速度传感器。它能够测量沿三个轴的加速度与绕三个轴旋转的角速度。MPU 6050的轴如图13所示

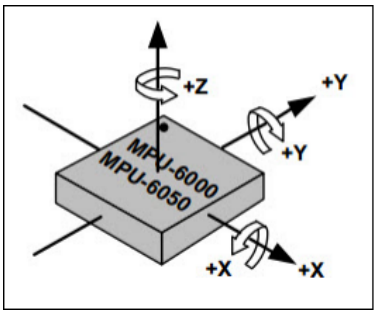


图13 MPU 6050轴

使用Arduino读取MPU 6050数据需要通过I2C协议，从SDA和SCL引脚读取。这一系列操作可以比较方便地通过`Wire.h`库实现。

对传感器的初始化流程如下：

- 向`0x6B`位置（PWR_MGMT_1 寄存器）写入`0x00`以启动传感器
- 向`0x1B`位置（GYRO_CONFIG 寄存器）写入以设定角速度量程。本项目中，写入`0x08`，对应量程$\pm 500^\circ/\mathrm{s}$，对应 65.5个最小有效位$/(^\circ/\mathrm{s})$
- 向`0x1C`（ACCEL_CONFIG 寄存器）写入以设定加速度量程。本项目中，写入`0x10`，对应量程$\pm 8g$

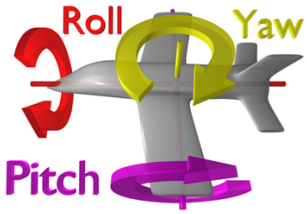
读取数据时。利用

```
Wire.beginTransmission(gyro_address);
Wire.write(0x3B);
Wire.endTransmission();
Wire.requestFrom(gyro_address,14);
```

可以获取从`0x3B`（ACCEL_XOUT[15:8] 寄存器）往后14个寄存器里的字节。每次通过`Wire.read()`可以读取一个字节。值得注意的是，由于加速度和角速度是以16位的形式存储的，在读取时需要先读取上八位，再与下八位合并，即`Wire.read() << 8 | Wire.read();`。具体的寄存器地址可以参考技术手册。

PID控制

在本项目中，使用PID控制的部分较为简单。其核心在于平衡当前时刻的飞机的运动与操纵杆的信号，如果飞机当前已有某个方向的运动，则向该方向机动的控制信号会被削弱，反之，机动信号会被增强。RC遥控器三个摇杆的信号分别对应3个姿态轴，即Pitch，Roll，Yaw，操纵杆信号经过一定处理，转换为角速度信号，即`pid_set_point`，并从MPU6050中读取各轴的角速度`gyro_input`，对这两个信号做差得到`pid_error`。与`pid_error`成比例的信号即P信号`pid_p`，对其求和的信号即I信号`pid_i`，对其与上一次得到的`pid_error`做差的信号即D信号`pid_d`。它们合成各个轴的`pid_output`。需要注意的是，对于`pid_i`和`pid_output`需要设置一个信号上限，防止其过大，影响飞行。



计算pid输出信号具体步骤如下， 分4步， `xxx` 表示pitch, roll 或 yaw。

- step 1 计算姿态差别 `pid_error_temp`：当前角速度 `gyro_xxx_input` 减去操纵杆输入 `pid_xxx_setpoint`
- step 2 计算积分信号 `pid_i_mem_xxx`：用积分系数 `pid_i_gain_xxx` 乘当前姿态差别 `pid_error_temp`，加到上次的积分信号上；若信号超过最大幅度 `pid_max_xxx`，则将其限制为最大幅度
- step 3 计算pid输出信号 `pid_output_xxx`：输出信号等于比例信号 `pid_p_gain_xxx * pid_error_temp`，加积分信号 `pid_i_mem_xxx`，加微分信号 `pid_d_gain_xxx * (pid_error_temp - pid_last_xxx_d_error)`
- step 4 重置微分中的上次姿态差别 `pid_last_roll_d_error`：将当前姿态差别 `pid_error_temp` 赋值给上次姿态差别