

手势运动手套中文说明书



Github <https://github.com/keywish/Gesture-MotionTracking>

修订版历史

Date	Version	Description	Author
2018-5-25	V. 1. 0	Create	Jason. Yang
2018-7-4	V. 1. 1	Review	Ken. Chen
2018-11-2	v. 1. 2	Optimize the document	Abbott. Chen
2019-4-8	v. 1. 3	修改 NRF24L01+和 UnoR3 引脚连接冲突	Ken. chen

目录

前期准备.....	5
1. 器件清单介绍.....	5
2. NANO 处理器介绍	6
2.1 电源.....	6
2.2 存储器.....	6
2.3 输入输出.....	6
2.4 通信接口.....	7
2.5 下载程序.....	7
2.6 注意要点.....	7
3. 无线通讯模块介绍（JDY-16）	7
3.1 无线通讯模块配置.....	8
3.2 操作步骤.....	8
3.4 Nano 和 JDY-16 的连接方式:	11
4. NRF24L01+无线模块介绍	11
4.1 模块特点.....	12
4.2 实验目的.....	13
4.3 本次实验所需元器件.....	13
4.4 程序原理.....	15
5. MPU6050 说明介绍	17
5.1 模块特点.....	17
5.2 模块原理图.....	18
5.3 Arduino NANO 与 MPU6050 的通信	18
5.3.2 从 MPU-6050 读出数据	19
5.3.4 MPU6050 的数据格式	20
5.4.1 实验一 读取加速度计	21
5.4.2 实验二 陀螺仪数据读取	26
5.5 运动数据分析	31
5.5.1 加速度模型.....	31
5.5.2 Roll-pitch-yaw 模型与姿态计算	32
5.5.3 yaw 角的问题	34
5.6 数据处理与实现.....	34
5.6.1 实验三 imu_kalman 得到 Roll 和 Pitch.....	35

5.6.2 实验代码.....	35
6、运动追踪控制原理.....	41
6.1 赛车方向坐标角模型.....	41
7. 通讯协议.....	50
8. 综合实验：.....	52
8.1 Nrf24L01 无线控制.....	52
8.1.1 运动手套控制 Hummer-bot 智能车	52
8.1.2 运动控制 Beetle-bot 智能车	53
8.2 蓝牙通讯方式控制.....	53
8.2.1 运动手套控制 Aurora-Racing.....	54
8.2.2 运动手套控制 Panther-tank	55
8.2.3 运动手套控制 balance car.....	56

前期准备

1. 器件清单介绍

为完成，需要准备以下套件：

MPU6050 模块	1
JDY-16 蓝牙模块	1
NRF24L01+模块	2
10uf 电容	2
Arduino NANO 主控板	1
mini 面包板	2
手套	1
公对母杜邦线	若干
电池	1



图 1：器件清单图

2.NANO 处理器介绍

Arduino Nano 微处理器是 ATmega328(Nano3.0)，带 USB-mini 接口，同时具有 14 路数字输入/输出（其中 6 路可作为 PWM 输出），8 路模拟输入，一个 16MHz 晶体振荡器，一个 mini-B USB 口，一个 ICSP header 和一个复位按钮。

- ◆ 处理器 ATmega328
- ◆ 工作电压 5V
- ◆ 输入电压（推荐） 7-12V
- ◆ 输入电压（范围） 6-20V
- ◆ 数字 IO 脚 14 (其中 6 路作为 PWM 输出)
- ◆ 模拟输入脚 6
- ◆ IO 脚直流电流 40mA
- ◆ Flash Memory 16 or 32KB （其中 2KB 用于 bootloader）
- ◆ SRAM 1KB or 2KB
- ◆ EEPROM 0.5 KB or 1KB （ATmega328）
- ◆ CH340 USB 转串口芯片
- ◆ 工作时钟 16 MHz

2.1 电源

Arduino Nano 供电方式：mini-B USB 接口供电和外部 vin 接 7~12V 外部直流电源

2.2 存储器

ATmega168/ATmega328 包括了片上 16KB/32KB Flash，其中 2KB 用于 Bootloader。同时还有 1KB/2KB SRAM 和 0.5KB/1KB EEPROM。

2.3 输入输出

- ◆ 14 路数字输入输出：工作电压为 5V，每一路能输出和接入最大电流为 40mA。每一路配置了 20-50K 欧姆内部上拉电阻（默认不连接）。除此之外，有些引脚有特定的功能。
- ◆ 串口信号 RX（0 号）、TX（1 号）：提供 TTL 电压水平的串口接收信号，与 FT232RL 的相应引脚相连。
- ◆ 外部中断（2 号和 3 号）：触发中断引脚，可设成上升沿、下降沿或同时触发。
- ◆ 脉冲宽度调制 PWM（3、5、6、9、10、11）：提供 6 路 8 位 PWM 输出。
- ◆ SPI（10(SS), 11(MOSI), 12(MISO), 13(SCK))：SPI 通信接口。
- ◆ LED（13 号）：Arduino 专门用于测试 LED 的保留接口，输出为高时点亮 LED，反之输出为低时 LED 熄灭。

- ◆ 6路模拟输入 A0 到 A5：每一路具有 10 位的分辨率（即输入有 1024 个不同值），默认输入信号范围为 0 到 5V，可以通过 AREF 调整输入上限。除此之外，有些引脚有特定功能。
- ◆ TWI 接口（SDA A4 和 SCL A5）：支持通信接口（兼容 I2C 总线）。
- ◆ AREF：模拟输入信号的参考电压。
- ◆ Reset：信号为低时复位单片机芯片。

2.4 通信接口

串口：ATmega328 内置的 UART 可以通过数字口 0（RX）和 1（TX）与外部实现串口通信。

2.5 下载程序

Arduino Nano 上的 MCU 已经预置了 bootloader 程序，因此可以通过 Arduino 软件直接下载程序。也可以直接通过 Nano 上 ICSP header 直接下载程序到 MCU。

2.6 注意要点

Arduino Nano 提供了自动复位设计，可以通过主机复位。这样通过 Arduino 软件下在程序到 Nano 中软件可以自动复位，不需要在复位按钮。

3. 无线通讯模块介绍（JDY-16）

本设计采用 JDY-16 主从一体的蓝牙模块，该蓝牙模块有以下特点：

- ◆ BLE 高速透明传输支持 8K 字节速率通信；
- ◆ 发送和接收无字节限制的数据，支持 115200 波特率连续发送并接收数据；
- ◆ 支持 3 种工作模式（请参阅 AT + STARTEN 指令功能的说明）；
- ◆ 支持（串口，IO，APP）睡眠唤醒；
- ◆ 支持微信 Airsync，微信 applet 和 APP 通讯；
- ◆ 支持 4 通道 IO 端口控制和高精度 RTC 时钟；
- ◆ 支持 PWM 功能（可以通过 UART，IIC，APP 等进行控制）；
- ◆ 支持 UART 和 IIC 通讯模式，默认为 UART 通讯；
- ◆ iBeacon 模式（支持微信摇一摇协议与苹果 iBeacon 协议）和主机透传模式（应用模块间数据透传，主机与从机通信）。

3.1 无线通讯模块配置

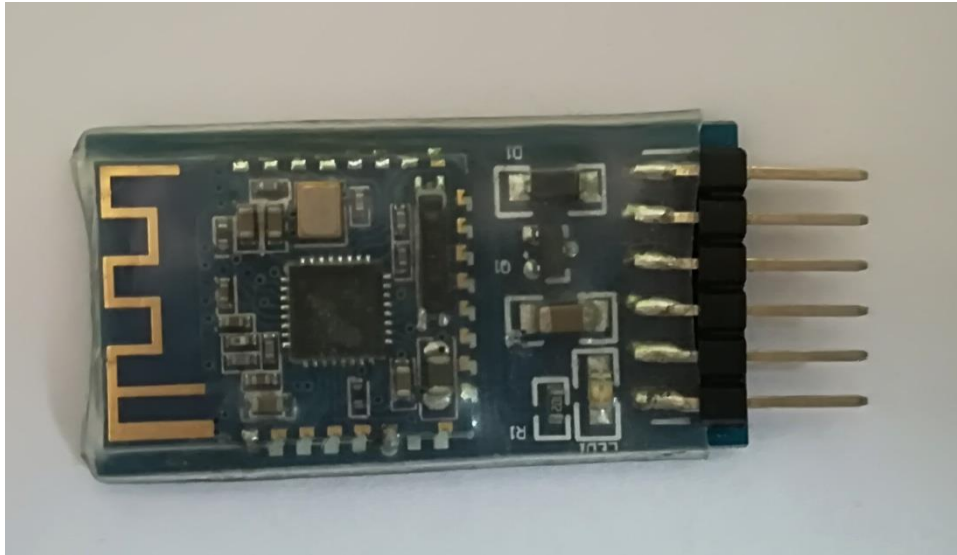


图 2：JDY-16 模块实物图

(1) Nano 和 JDY-16 的连接线

JDY-16 Module	Arduino NANO
VCC	5V
GND	GND
RXD	D2
TXD	D3
STAT	NC
PWRC	NC

下载程序 MotionTrack\JDY-16\AT_CMD\ AT_CMD.ino

(2) 配置要求：

实现两个 JDY-16 蓝牙模块的主从绑定。

3.2 操作步骤

1.用杜邦线连接好 Nano 和 JDY-16 模块。

2.进入 AT 指令模式

将下载器与电脑连接，并打开串口助手，设置波特率 9600，数据位 8 位，停止位 1 位，无校验位。

测试通讯：

发送：AT

返回：OK



图 3：发送 AT 指令图

3.发送：AT+HOSTEN1\r\n ----->设置蓝牙为主模式

返回：OK

返回：OK



图 4：设置主机模式

4. 扫描周边 JDY-16 蓝牙：

1.从机发送：AT+SCAN\r\n ----->查询从机自身地址

返回： OK

+DEV:1=3CA5090A160F,-62,JDY-16

+STOP:SCAN

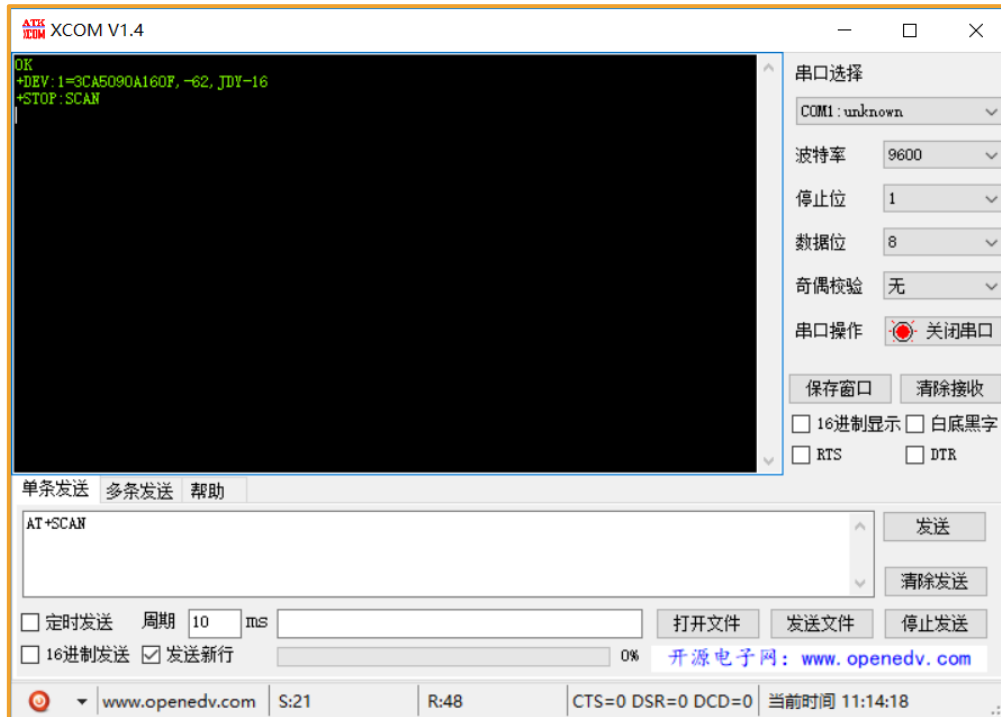


图 5： 查询周边蓝牙设备

5.连接蓝牙

主机发送：AT+CONN3CA5090A160F\r\n ----->主机绑定从机地址

返回：OK

接上电源，两个蓝牙就可以相互连接了，通过主机蓝牙发送数据，从机蓝牙就可接收到同样的数据。或者你也可以直接烧录准备好的主从配置程序，来设置主从机蓝牙，之后主机烧录连接程序，自动连接。(注意目前只能自动连接 JDY-16 的蓝牙模块)

3.4 Nano 和 JDY-16 的连接方式:

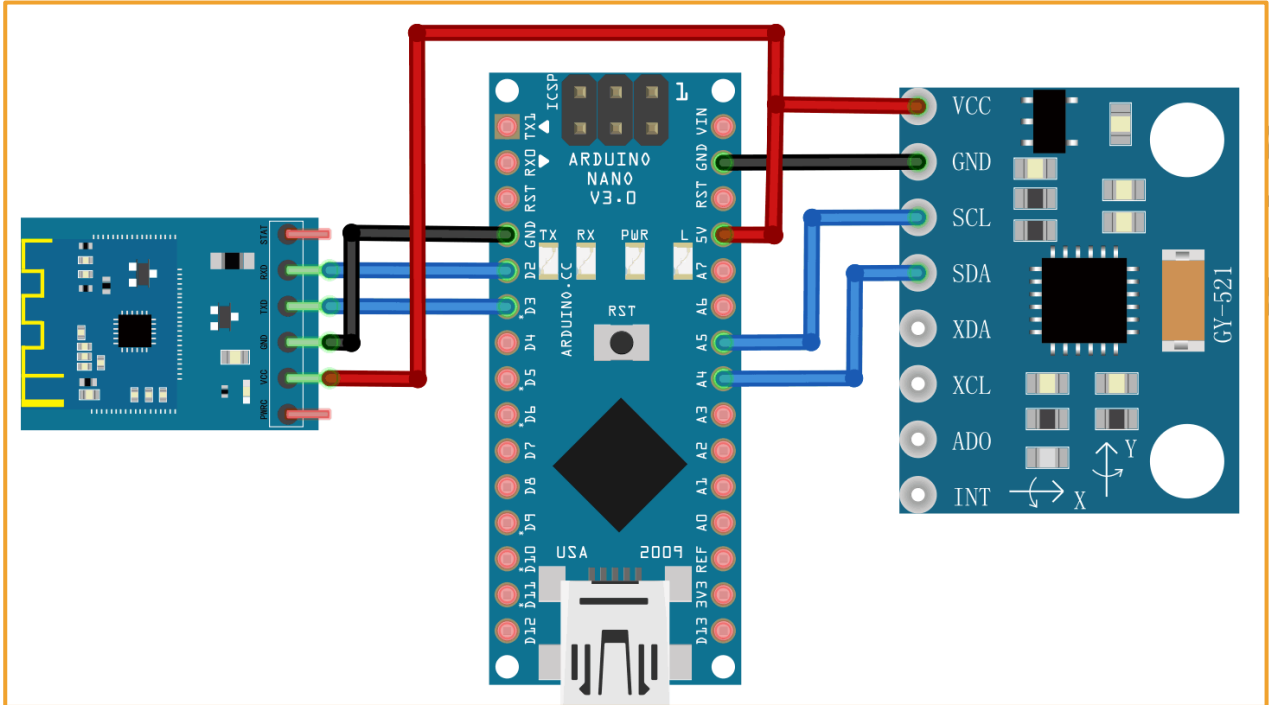


图 6: Nano 和 JDY-16 连接图

4、NRF24L01+无线模块介绍

nRF24L01+模块是 Nordic 公司基于 nRF24L01 芯片开发成的 2.4G 无线通讯模块。采用 FSK 调制，内部集成 Nordic 自己的 Enhanced Short Burst 协议。可以实现点对点或是 1 对 6 的无线通信。无线通信速度最高可达到 2M (bps)，NRF24L01 有收发模式，配置模式，空闲模式，关机模式四个工作模式。本次实验使用的实物图 7 左边。为了 Nrf24L01 收据的接收稳定建议 VCC 和 GUD 之间连接 10uf 电容如下图右边

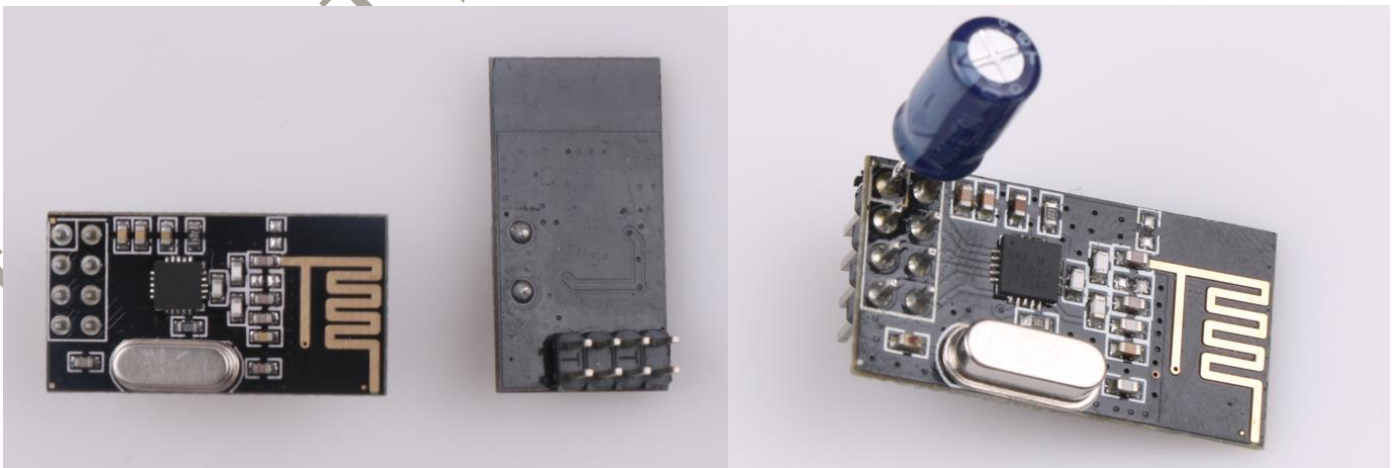


图 7: Nrf24l01+实物图和焊接图

4.1 模块特点

- ◆ 2.4GHz 体积小巧，15x29mm（包括天线）
- ◆ 支持六路通道的数据接收
- ◆ 低工作电压： 1.9~3.6V 低电压工作
- ◆ 数据传输速率支持 1Mbps、2Mbps
- ◆ 低功耗设计，接收时工作电流 12.3mA，0dBm 功率发射时 11.3mA，掉电模式时仅为 900nA
- ◆ 自动重发功能，自动检测和重发丢失的数据包，重发时间及重发次数可软件控制
- ◆ 自动应答功能，在收到有效数据后，模块自动发送应答信号
- ◆ 内置硬件 CRC 检错和点对多点通信地址控制

NRF24L01 芯片详细参数请参考《nRF24L01 Datasheet.pdf》

引脚信息

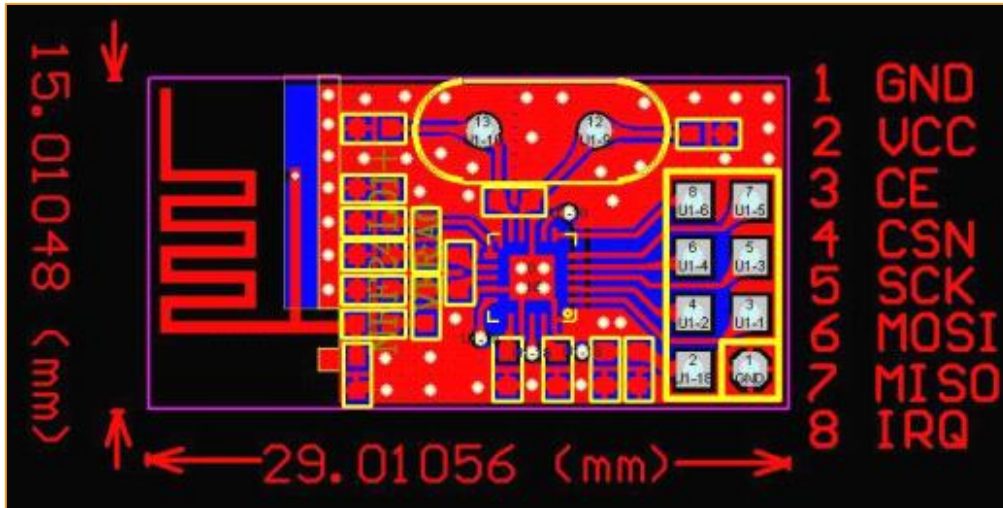


图 8: Nrf24L01 引脚信息图

管脚	符号	功能	方向
1	GND	电源地	
2	+5V	电源	
3	CE	工作模式控制线	IN
4	CSN	芯片片选信号, 低电平工作	IN
5	SCK	SPI 时钟	IN
6	MOSI	SPI 输入	IN
7	MISO	SPI 输出	OUT
8	IPQ	中断输出	OUT+

4.2 实验目的

1. 了解 nRF24L01+模块如何和 arduino 连接。
2. 如何使用 arduino 和 nRF24L01+模块完成接收发送数据。

4.3 本次实验所需元器件

- ◆ Arduino UNO R3 主板*1
- ◆ Arduino NANO 主板*1
- ◆ nRF24L01 模块*2
- ◆ 导线若干

4.4 本次实验原理图

本次实验连接示意图

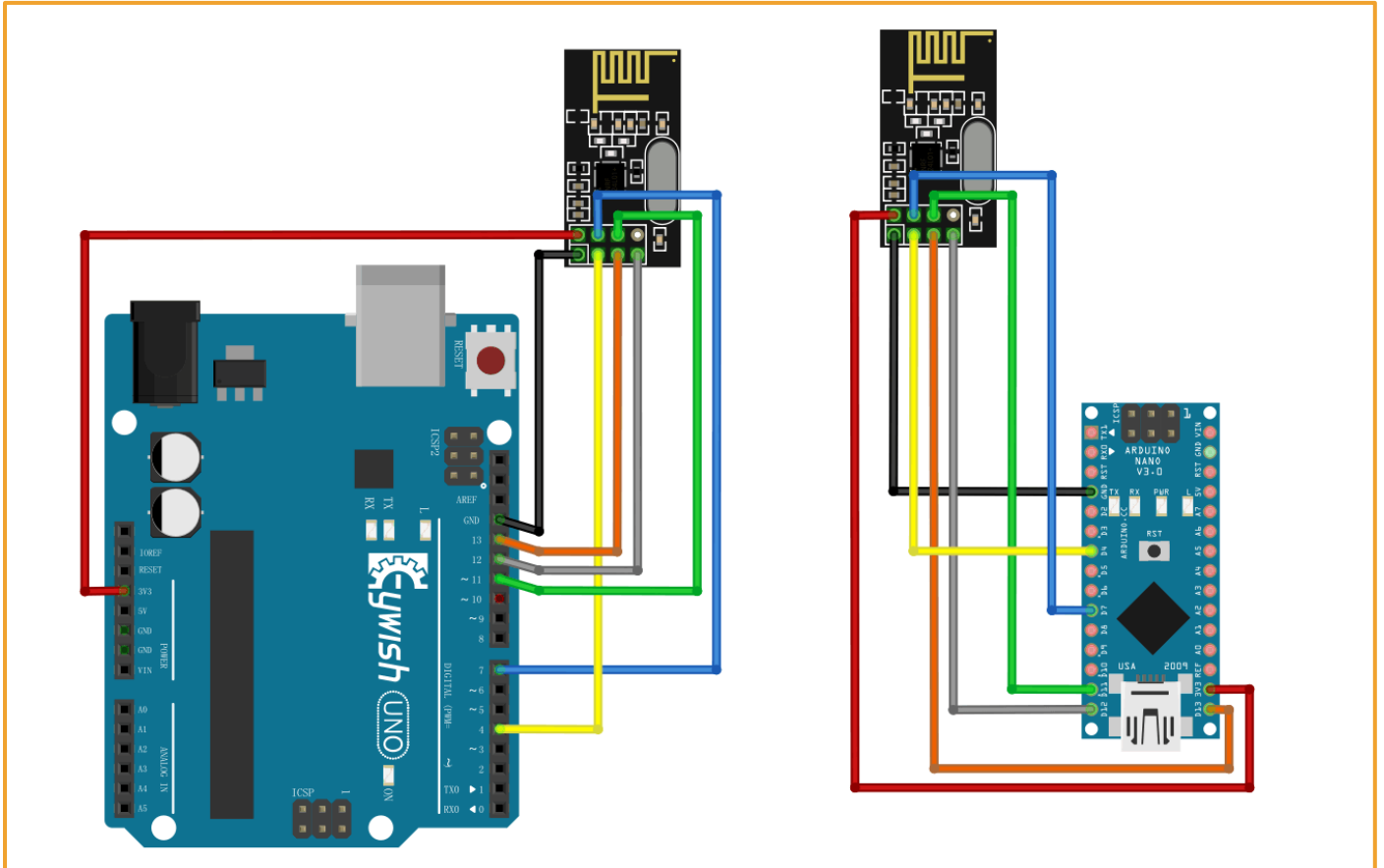


图 9: Nano 和 Nrf24L01 连接图

Arduino 和 NRF24L01 接线方式

arduino Nano	nRF24L01
+3.3V	VCC
GND	GND
7pin	4pin CSN
4pin	3pin CE
11pin	6pin MOSI
12pin	7pin MISO
13pin	5pin SCK

arduino Uno	nRF24L01
+3.3V	VCC
GND	GND
7	4pin CSN

4	3pin CE
11pin	6pin MOSI
12pin	7pin MISO
13pin	5pin SCK

4.4 程序原理

发射流程

- 1、首先将 nRF24L01 配置为发射模式，
- 2、接着把接收端的地址 TX_ADDR 和要发送的数据 TX_PLD 按照时序由 SPI 口写入 nRF24L01 缓存区。
- 3、Arduino 将 CE 配置为高电平并保持至少 10 μ s，延迟 130 μ s 后发射数据；若自动应答开启，那么 nRF24L01 在发射数据后立即进入接收模式，接收应答信号。如果收到应答，则认为此次通信成功。
- 4、NRF24L01 会自动将 TX_DS 置高，同时 TX_PLD 从发送堆栈中清除；若未收到应答，则自动重新发射该数据，若重发次数 (ARC_CNT) 达到上限，MAX_RT 置高，TX_PLD 不会被清除；MAX_RT 或 TX_DS 置高时，IRQ 变低触发 MCU 中断。最后发射成功时，若 CE 为低，则 nRF24L01 进入待机模式
- 5、若发送堆栈中有数据且 CE 为高，则进入下一次发射；若发送堆栈中无数据且 CE 为高，则进入待机模式 2。

接收数据流程

- 1、nRF24L01 接收数据时，首先将 nRF24L01 配置为接收模式，
- 2、接着延迟 130 μ s 进入接收状态等待数据的到来。当接收方检测到有效的地址和 CRC 时，就将数据包存储在接收堆栈中，同时中断标志位 RX_DR 置高，IRQ 变低，以便通知 MCU 去取数据。
- 3、若此时自动应答开启，接收方则同时进入发射状态回传应答信号。最后接收成功时，若 CE 变低，则 nRF24L01 进入空闲模式 1。

nRF24L01 收发测试程序见 “**Gesture-MotionTracking\nRF24l01+\program**”

5、MPU6050 说明介绍

MPU6050 是 InvenSense 公司推出全球首例集成 3 轴陀螺仪与 3 轴加速器的 6 轴运动处理组件，可以通过第二个 I2C 端口连接其他磁力传感器、或其他传感器的数字运动处理(DMP: Digital Motion Processor)硬件加速引擎，主要由 I2C 端口以单一数据流的形式，向主控端 MCU 输出完整的 9 轴融合演算技术。

MPU6050 芯片内自带了一个数据处理子模块 DMP，已经内置了硬件滤波算法，在许多应用中使用 DMP 输出的数据已经能够很好地满足要求，不需要我们软件再做滤波。本课程将以通过 Arduino 读取 DMP 作为输出数据为基础，制造一个完整的运动手套。

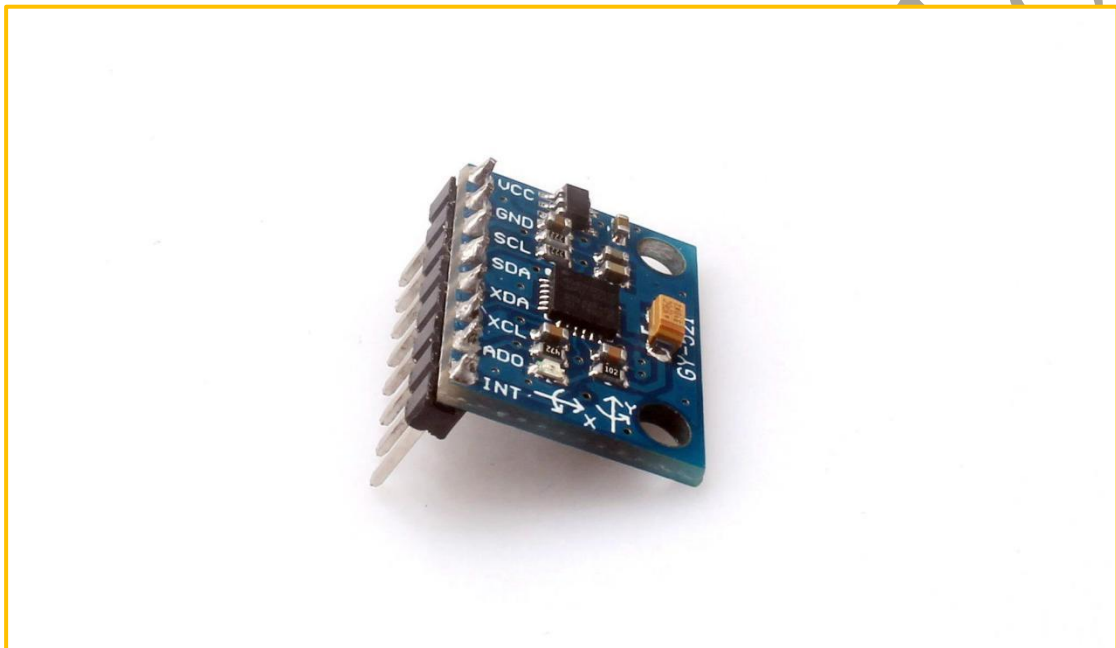


图 10: mpu6050 模块实物图

5.1 模块特点

- ◆ 以数字输出 6 轴或 9 轴的旋转矩阵、四元数(quaternion)、欧拉角格式(Euler Angle forma)的融合演算数据。
- ◆ 具有 131 LSBs/°/sec 敏感度与全格感测范围为 ± 250 、 ± 500 、 ± 1000 与 ± 2000 °/sec 的 3 轴角速度感测器(陀螺仪)。
- ◆ 可程式控制，且程式控制范围为 $\pm 2g$ 、 $\pm 4g$ 、 $\pm 8g$ 和 $\pm 16g$ 的 3 轴加速器。
- ◆ 数字运动处理(DMP: Digital Motion Processing)引擎可减少复杂的融合演算数据、感测器同步化、姿势感应等的负荷。
- ◆ 运动处理数据库支持 Android、Linux 与 Windows。
- ◆ 内建运作时间偏差与磁力感测器校正演算技术，免除了客户须另外进行校正的需求。
- ◆ 以数位输出的温度传感器。
- ◆ VDD 供电电压为 $2.5V \pm 5\%$ 、 $3.0V \pm 5\%$ 、 $3.3V \pm 5\%$ ；VDDIO 为 $1.8V \pm 5\%$ 。

5.2 模块原理图

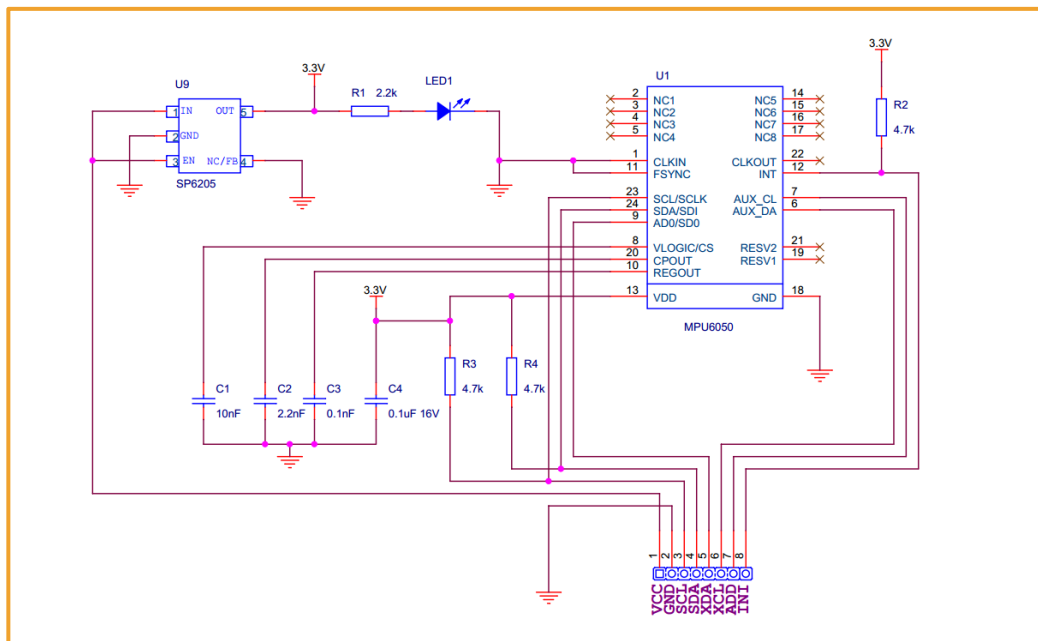


图 11: mpu6050 模块原理图

5.3 Arduino NANO 与 MPU6050 的通信

5.3.1 电路连接

集成 MPU6050 模块的数据界面用的是 I2C 总线协议，因此我们需要 Wire 程序库的帮助来实现 NANO 与 MPU6050 之间的通信。而 NANO 板对应连线如下图：

MPU6050 Module	Arduino NANO
VCC	5V
GND	GND
SCL	A5
SDA	A4
XDA	NC
XCL	NC
ADD	NC
INT	NC/GND

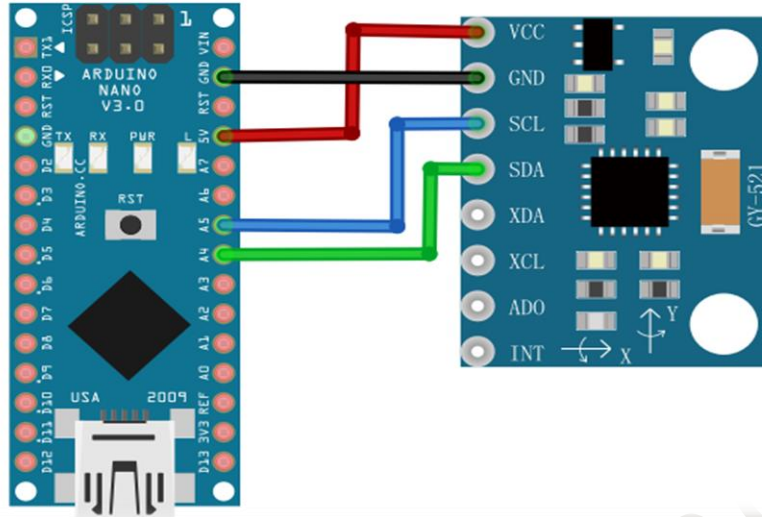


图 11: Nano 和 mpu6050 连接示意图

MPU6050 的数据写入和读出均通过其芯片内部的寄存器实现，而这些寄存器的地址都是 1 个字节，即 8 位的寻址空间。详情请参考《RM-MPU-6000A.pdf》1.1 将数据写入 MPU-6050。

在每次向器件写入数据前要先打开 Wire 的传输模式，并指定器件的总线地址，MPU6050 的总线地址是 0x68（AD0 引脚为高电平时地址为 0x69）。然后写入一个字节的寄存器起始地址，再写入任意长度的数据。这些数据将被连续地写入到指定的起始地址中，超过当前寄存器长度的将写入到后面地址的寄存器中。写入完成后关闭 Wire 的传输模式。下面的示例代码是向 MPU6050 的 0x6B 寄存器写入一个字节 0。

```
Wire.beginTransmission(0x68); //开启 MPU6050 的传输
Wire.write(0x6B); //指定寄存器地址
Wire.write(0); //写入一个字节的数
Wire.endTransmission(true); //结束传输，true 表示释放总线
```

5.3.2 从 MPU-6050 读出数据

读出和写入一样，要先打开 Wire 的传输模式，然后写一个字节的寄存器起始地址。接下来将指定地址的数据读到 Wire 库的缓存中，并关闭传输模式，最后从缓存中读取数据。下面的示例代码是从 MPU6050 的 0x3B 寄存器开始读取 2 个字节的数据：

```
Wire.beginTransmission(0x68); //开启 MPU6050 的传输
Wire.write(0x3B); //指定寄存器地址
Wire.requestFrom(0x68, 2, true); //将输出读到缓存
Wire.endTransmission(true); //关闭传输模式
int val = Wire.read() << 8 | Wire.read(); //两个字节组成一个 16 位整数
```

5.3.3 具体实现

通常应当在 `setup` 函数中对 Wire 库进行初始化：

```
Wire.begin();
```

在对 MPU6050 进行各项操作前，必须启动该器件，向它的 0x6B 写入一个字节即可启动。通常也是在 `setup` 函数完成，代码见 2.4.1 节。

5.3.4 MPU6050 的数据格式

我们感兴趣的数据位于 0x3B 到 0x48 这 14 个字节的寄存器中。这些数据会被动态更新，更新频率最高可达 1000HZ。下面列出相关寄存器的地址，数据的名称。注意，每个数据都是 2 个字节。

0x3B：加速度计的 X 轴分量 ACC_X

0x3D：加速度计的 Y 轴分量 ACC_Y

0x3F：加速度计的 Z 轴分量 ACC_Z

0x41：当前温度 TEMP

0x43：绕 X 轴旋转的角速度 GYR_X

0x45：绕 Y 轴旋转的角速度 GYR_Y

0x47：绕 Z 轴旋转的角速度 GYR_Z

MPU6050 芯片的坐标系定义：将芯片表面朝向自己，并将表面文字转至正确角度，此时，以芯片内部中心为原点，水平向右的为 X 轴，竖直向上的为 Y 轴，指向自己的为 Z 轴。见下图：

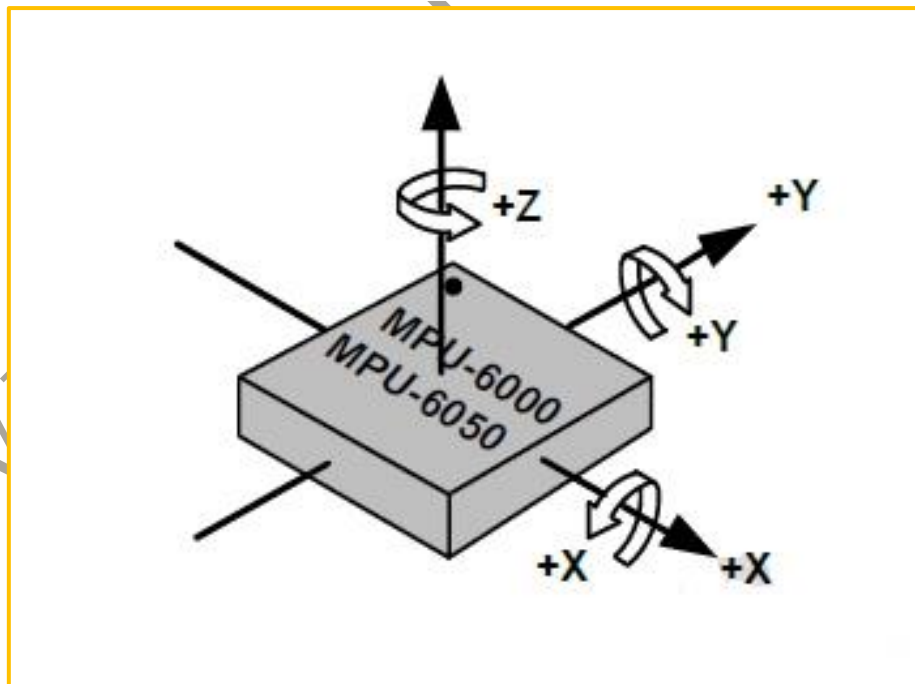


图 12： mpu6050 旋转和角速度关系图

我们只关心加速度计和角速度计数据的含义，下面我们通过两个实验来熟悉 mpu6050 使用。

5.4.1 实验一 读取加速度计

加速度计的三轴分量 ACC_X、ACC_Y 和 ACC_Z 均为 16 位有符号整数，分别表示器件在三个轴向上的加速度。取负值时加速度沿坐标轴负向，取正值时沿正向。

三个加速度分量均以重力加速度 g 的倍数为单位，能够表示的加速度范围，即倍率可以统一设定，有 4 个可选倍率：2g、4g、8g、16g。以 ACC_X 为例，若倍率设定为 2g（默认），则意味着 ACC_X 取最小值-32768 时，当前加速度为沿 X 轴正方向 2 倍的重力加速度，以此类推。显然，倍率越低精度越好，倍率越高表示的范围越大，这要根据具体的应用来设定。

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 2g$	16384LSB/g
1	$\pm 4g$	8192LSB/g
2	$\pm 8g$	4096LSB/g
3	$\pm 16g$	2048LSB/g

三轴加速度计和模块的转动方向关系如下图：

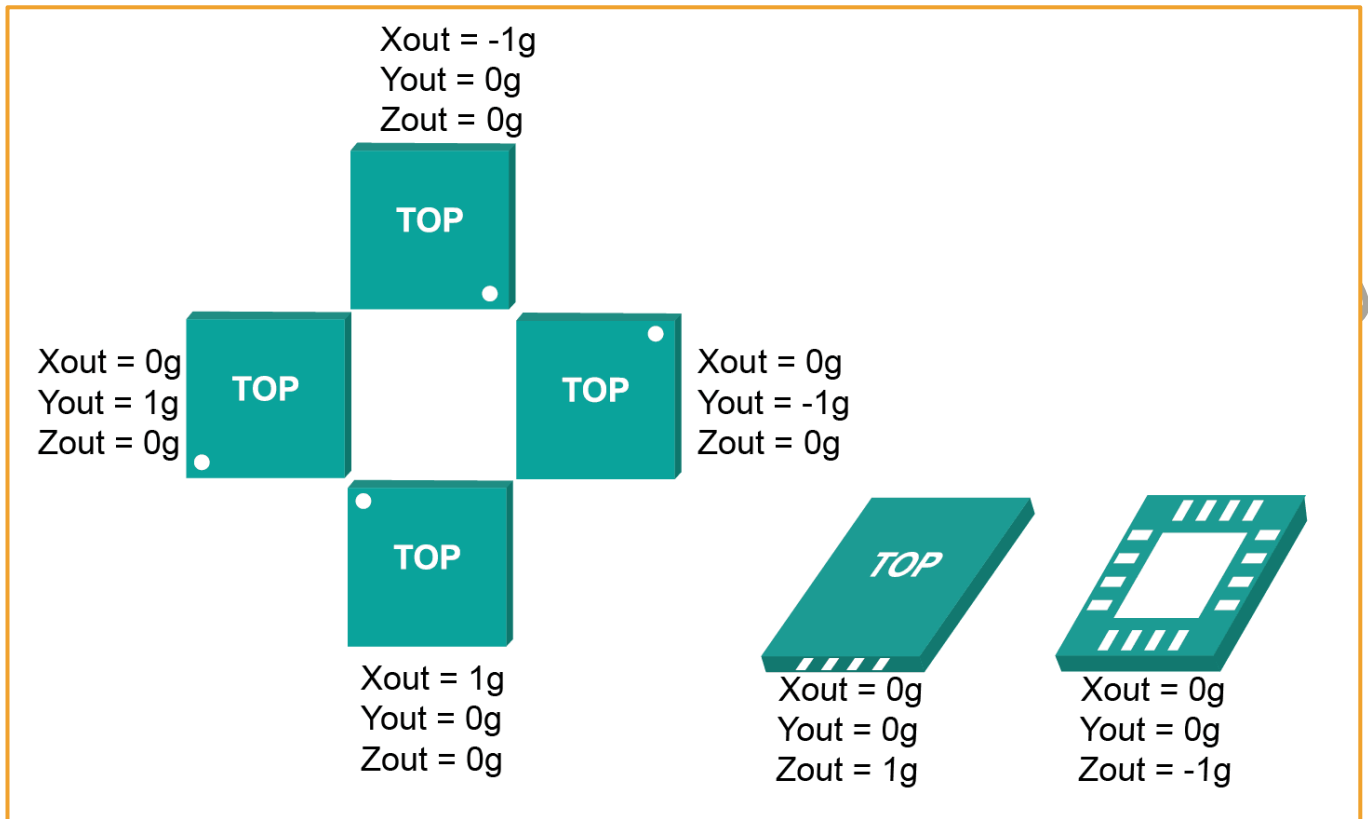


图 13: mpu6050 模块转动和加速度方向图

MPU6050 读取的数据是有波动的，所以需要做校验，即在芯片处理静止状态时，这个读数理论上讲应当为 0。但它往往会存在偏移量，比如我们以 10ms 的间隔读取了 200 个值，再取平均值，这个值我们称之为零偏移量。每次的读数都减去零偏移量就可以得到校准后的读数了。由于 ACC_X 和 ACC_Y 的理论值应为 0，这两个读数偏移量可用统计均值的方式校准。ACC_Z 则需要多一步处理，即在统计偏移量的过程中，每次读数都要减去 Z 轴的重力加速度 g，如果加速的倍率为 2g 时那么要减去 16384，再进行统计均值校准。一般校准可以在每次启动系统时进行，那么你应当在准确度和启动时间之间做一个权衡。

实验目的

通过转动 mpu6050 来观察加速度计三个轴的输出数据关系。

实验代码

代码位置: MotionTrack\Lesson\mpu6050_accel\mpu6050_accel.ino

```
#include "Wire.h"
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h
files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"
// class default I2C address is 0x68
// AD0 low = 0x68 (default for InvenSense evaluation board) AD0 high = 0x69
```

Keywish Technology

```
#define LED_PIN 13
MPU6050 accelgyro;

struct RAW_type
{
    uint8_t x;
    uint8_t y;
    uint8_t z;
};

int16_t ax, ay, az;
int16_t gx, gy, gz;
struct RAW_type accel_zero_offsent;
char str[512];
bool blinkState = false ;
float AcceRatio = 16384.0;
float accx, accy, accz;

void setup() {
    int i ;
    int32_t ax_zero = 0, ay_zero = 0, az_zero = 0 ;
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();
    Serial.begin(115200);
    // initialize device
    Serial.println("Initializing I2C devices...");
    accelgyro.initialize();
    delay(500) ;
    accelgyro.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");
```

```
for( i = 0 ; i < 200 ; i++)
{
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    ax_zero += ax ;
    ay_zero += ay ;
    az_zero += az ;
}
accel_zero_offsent.x = ax_zero/200 ;
accel_zero_offsent.y = ay_zero/200 ;
accel_zero_offsent.z = az_zero/200 ;
Serial.print(accel_zero_offsent.x); Serial.print("\t");
Serial.print(accel_zero_offsent.y); Serial.print("\t");
Serial.print(accel_zero_offsent.z); Serial.print("\n");
pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // read raw accel/gyro measurements from device
    delay(1000);
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    sprintf(str,"%d,%d,%d\n",ax-accel_zero_offsent.x,
ay-accel_zero_offsent.y ,az-accel_zero_offsent.z);
    Serial.print(str);
    accx = (float)( ax-accel_zero_offsent.x )/AcceRatio;
    accy = (float)( ay-accel_zero_offsent.y )/AcceRatio ;
    accz = (float)( az-accel_zero_offsent.z )/AcceRatio ;
    Serial.print(accx);Serial.print("g\t");
    Serial.print(accy);Serial.print("g\t");
    Serial.print(accz);Serial.print("g\n");

    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}
```

1. 绕 X 轴旋转 90 度

当绕 X 轴旋转 90 度时，Y 轴慢慢向上，Z 轴慢慢向下。当完全达到 90 度时，由于 Y 轴与重力方向刚好相反，所以 Y 轴的输出是 $1g(1g=9.8m/s^2)$ ，而 Z 轴的值从原来的 1 逐渐减小为 0。

2. 回到初始位置，并反向旋转 90 度

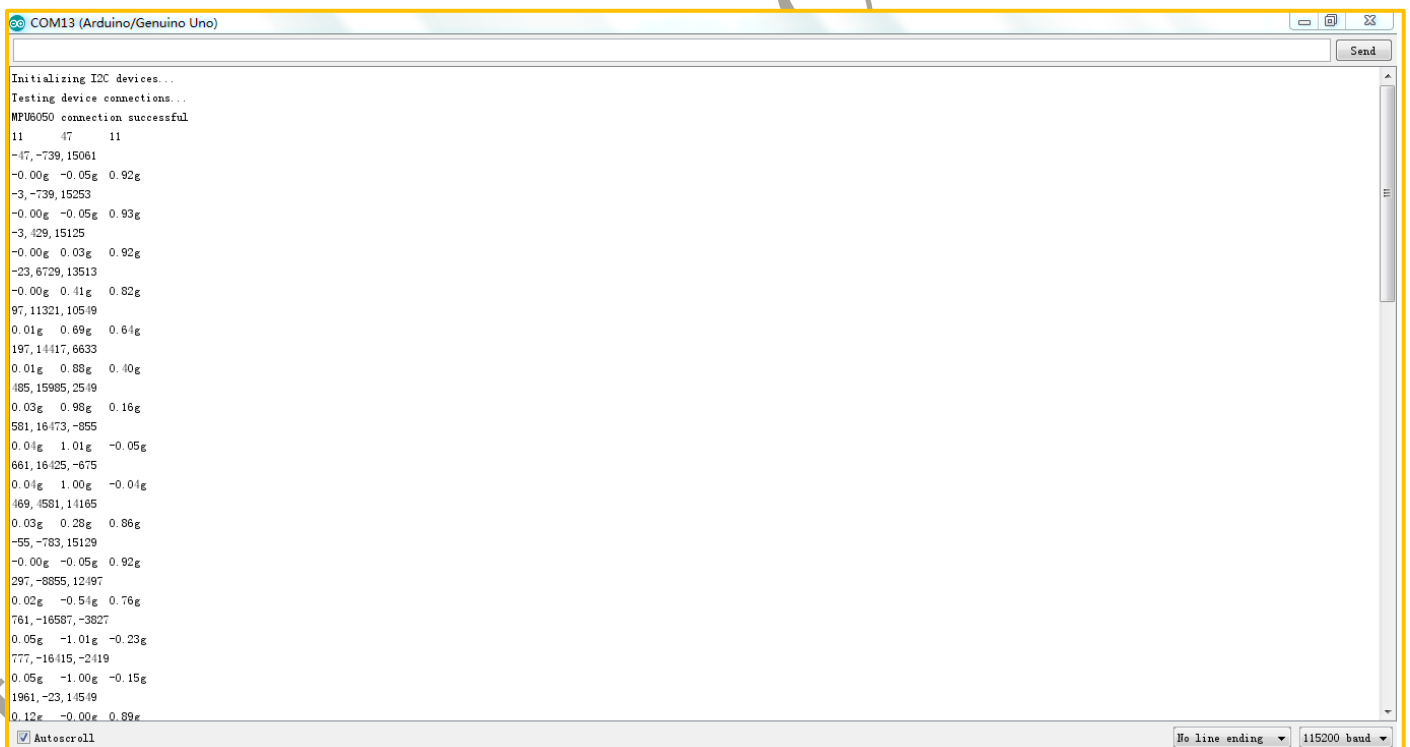
当回到初始位置时，Y 轴数据慢慢减小为 0，而 Z 轴数据慢慢上升为 1。然后逆向旋转 90 度，Y 轴慢慢减小，直至为 -1，因为此时 Y 轴方向与重力方向一致了，而反应出来的加速度值应该是负值。而 Z 轴慢慢减小为 0。

3. 回到初始位置

解释如下：然后从逆向 90 度回到初始位置。此时 Y 轴和 Z 轴的数据慢慢恢复到初始值，Y 轴为 0，而 Z 轴为 1。

分析完了 X 轴的旋转，其实 Y 轴的旋转也是类似的，就不说了。现在说说 Z 轴的，因为绕 Z 轴旋转时，相当于左右摆动 90 度，此时 Z 轴的输出始终是 1，而 X 轴和 Y 轴由于是正交于重力轴的，所以输出值都是 0，当然这都是在相对静止的条件下的值。如果是将这个设备安装在车辆上，当车左右转弯时，X 轴和 Y 轴的读数可不一定是 0。

实验结果



5.4.2 实验二 陀螺仪数据读取

绕 X、Y 和 Z 三个座标轴旋转的角速度分量 GYR_X、GYR_Y 和 GYR_Z 均为 1 位有符号整数。从原点向旋转轴方向看去，取正值时为顺时针旋转，取负值时为逆时针旋转。

三个角速度分量均以“度/秒”为单位，能够表示的角速度范围，即倍率可统一设定，有 4 个可

选倍率：250 度/秒、500 度/秒、1000 度/秒、2000 度/秒。以 GYR_X 为例，若倍率设定为 250 度/秒，则意味着 GYR 取正最大值 32768 时，当前角速度为顺时针 250 度/秒；若设定为 500 度/秒，取 32768 时表示当前角速度为顺时针 500 度/秒。显然，倍率越低精度越好，倍率越高表示的范围越大。

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 250^{\circ}/s$	131LSB/ $^{\circ}/s$
1	$\pm 500^{\circ}/s$	65.5LSB/ $^{\circ}/s$
2	$\pm 1000^{\circ}/s$	32.8LSB/ $^{\circ}/s$
3	$\pm 2000^{\circ}/s$	16.4LSB/ $^{\circ}/s$

程序位置“MotionTrack\Lesson\mpu6050_gryo\ mpu6050_gryo.ino”

实验代码

```
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h
files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"
#define LED_PIN 13

MPU6050 accelgyro;

struct RAW_type
{
    uint8_t x;
    uint8_t y;
    uint8_t z;
};

int16_t ax, ay, az;
int16_t gx, gy, gz;
struct RAW_type accel_zero_offsent ,gyro_zero_offsent;

bool blinkState = false;
char str[512];
```

```
float pi = 3.1415926;
float AcceRatio = 16384.0;
float GyroRatio = 131.0;
float Rad = 57.3 ; //180.0/pi;
float gyro_x, gyro_y, gyro_z;

void setup() {
    int i ;
    int32_t ax_zero = 0, ay_zero = 0, az_zero = 0, gx_zero = 0, gy_zero = 0, gz_zero = 0 ;
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    Serial.begin(115200);

    // initialize device
    // Serial.println("Initializing I2C devices...");
    accelgyro.initialize();
    delay(500) ;
    accelgyro.setFullScaleGyroRange(MPU6050_GYRO_FS_250);

    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");
    for( i = 0 ; i < 200 ; i++)
    {
        accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        gx_zero += gx ;
        gy_zero += gy ;
        gz_zero += gz ;
    }
    gyro_zero_offsent.x = gx_zero/200 ;
    gyro_zero_offsent.y = gy_zero/200 ;
```

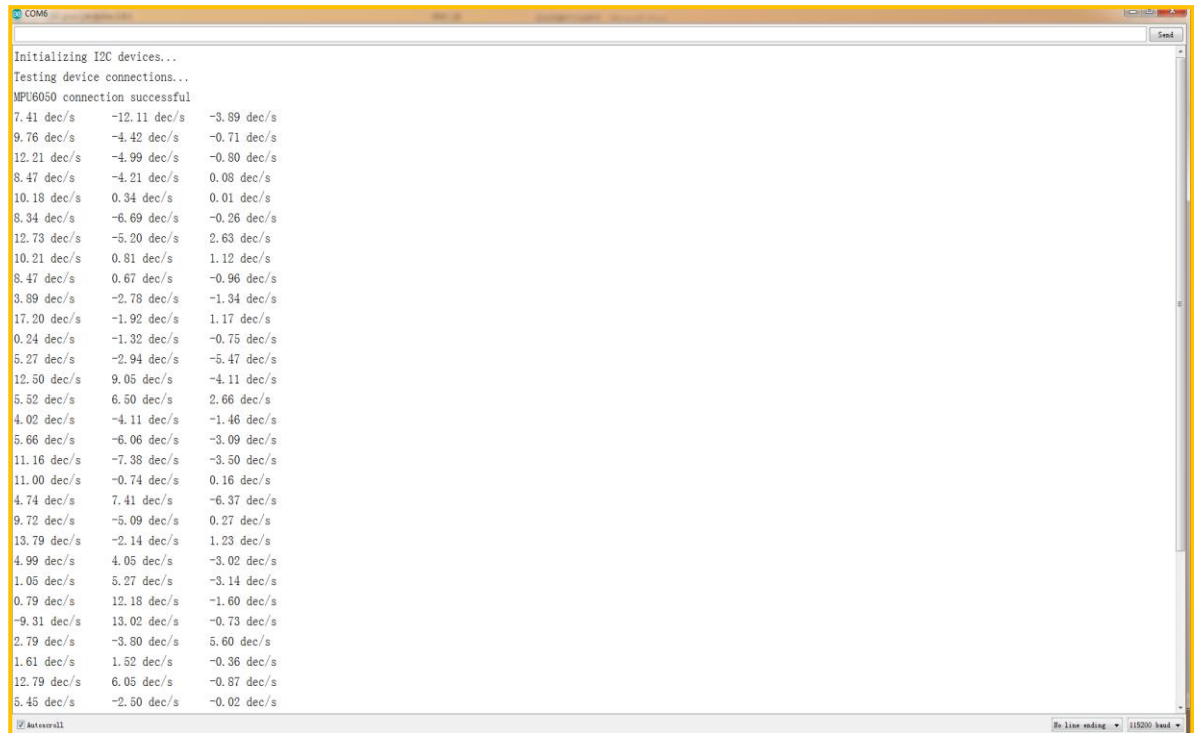
```
gyro_zero_offsent.z = gz_zero/200;
pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    //sprintf(str,"%d,%d,%d\n",
gx-gyro_zero_offsent.x ,gy-gyro_zero_offsent.y,
gz-gyro_zero_offsent.z);
    //Serial.print(str);
    gyrox = (float) (gx-gyro_zero_offsent.x)/AcceRatio;
    gyroy = (float) (gy-gyro_zero_offsent.y)/AcceRatio ;
    gyroz = (float) (gz-gyro_zero_offsent.z)/AcceRatio ;
    Serial.print(gyrox);Serial.print("g\t");
    Serial.print(gyroy);Serial.print("g\t");
    Serial.print(gyro);Serial.print("g\n");

    delay(100);
    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}
```

当我们沿着 x 轴正方向旋转，是我们看到打印的 gyrox 数据为正，否则为负数。



5.5 运动数据分析

在读取加速度计和角速度计的数据并换算为物理值后，根据不同的应用，数据有不同的解译方式。本章将以飞行器运动模型为例，根据加速度和角速度来算出当前的飞行姿态。

5.5.1 加速度模型

我们可以把加速度计想象为一个正立方体盒子里放着一个球，这个球被弹簧固定在立方体的中心。当盒子运动时，根据假想球的位置即可算出当前加速度的值。见下图：

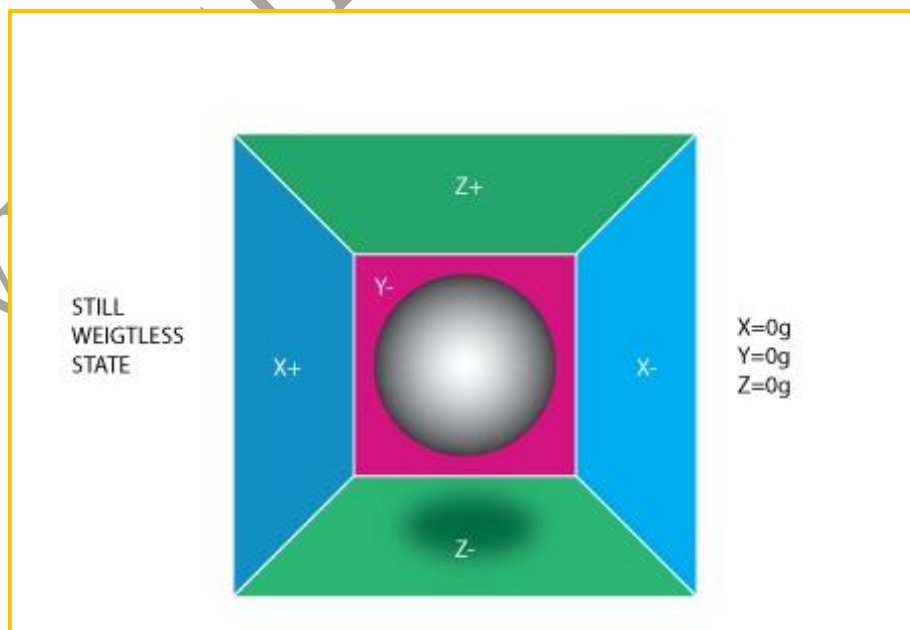


图 14：失重状态加速度值

如果我们给盒子施加一个水平向左的力，那么显然盒子就会有一个向左的加速度，此时盒内的假想球会因为惯性作用贴向盒内的右侧面。如下图所示：

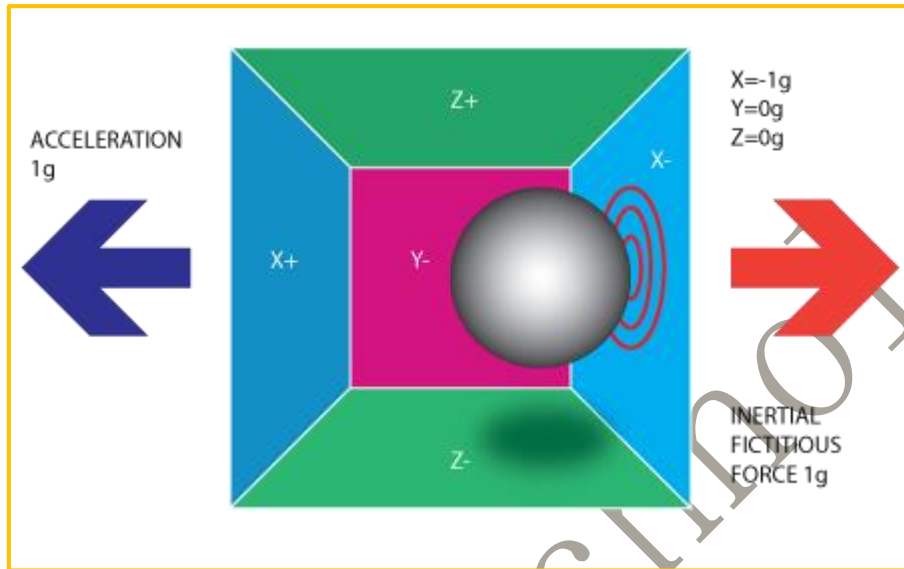


图 15：物体向右运动时加速度

为了保证数据的物理意义，MPU6050 的加速度计是以假想球在三轴上坐标值的相反数作为三个轴的加速度值。根据以上分析，当我们把 MPU6050 芯片水平放于地方，芯片表面朝向天空，此时由于受到地球重力的作用，假想球的位置偏向 Z 轴的负向，因此 Z 轴的加速度读数应为正，且在理想情况下应为 g。注意，此加速度的物理意义并不是重力加速度，而是自身运动的加速度，可以这样理解：正因为其自身运动的加速度与重力加速度大小相等方向相反，芯片才能保持静止。

5.5.2 Roll-pitch-yaw 模型与姿态计算

表示飞行器当前飞行姿态的一个通用模型就是建立下图所示坐标系，并用 Roll 表示绕 X 轴的旋转，Pitch 表示绕 Y 轴的旋转，Yaw 表示绕 Z 轴的旋转。

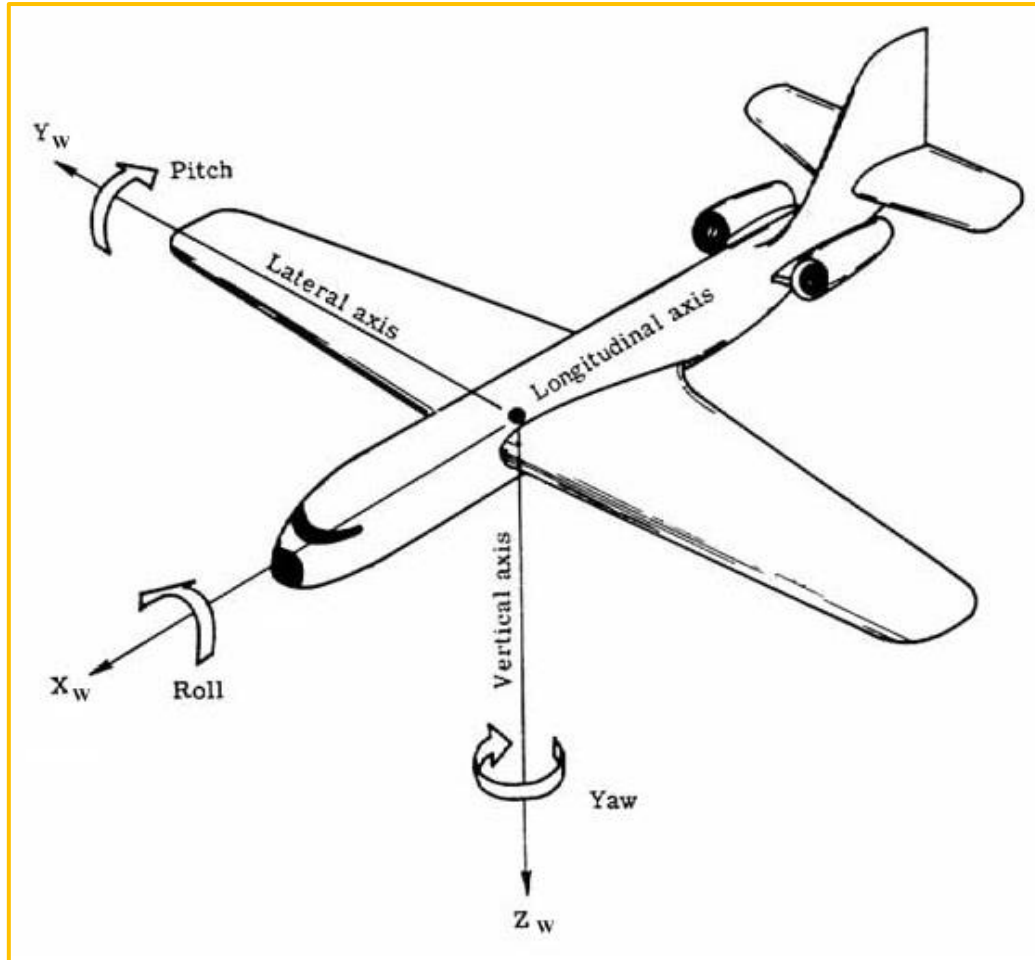


图 15: Roll-pitch-yaw 模型

由于 MPU6050 可以获取三个轴向上的加速度，而地球重力则是长期存在且永远竖直向下，因此我们可以根据重力加速度相对于芯片的指向为参考算得当前姿态。为方便起见，我们让芯片正面朝下固定在上图飞机上，且坐标系与飞机的坐标系完全重合，以三个轴向上的加速度为分量，可构成加速度向量 $a(x,y,z)$ 。假设当前芯片处于匀速直线运动状态，那么 a 应垂直于地面上向，即指向 Z 轴负方向，模长为 $|a|=g=\sqrt{x^2+y^2+z^2}$ （与重力加速度大小相等，方向相反，见 3.1 节）。若芯片（坐标系）发生旋转，由于加速度向量 a 仍然竖直向上，所以 Z 轴负方向将不再与 a 重合。见下图。

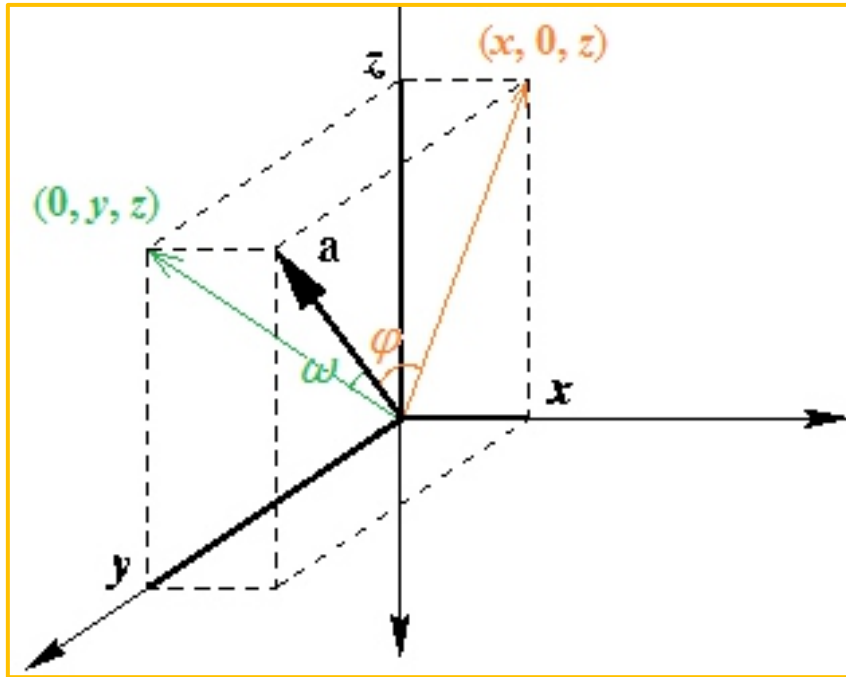


图 16: 姿态角计算模型

为了方便表示，上图坐标系的 Z 轴正方向（机腹以及芯片正面）向下，X 轴正方向（飞机前进方向）向右。此时芯片的 Roll 角 ϕ （黄色）为加速度向量与其在 XZ 平面上投影(x,0,z)的夹角，Pitch 角 ω （绿色）与其在 YZ 平面上投影(0,y,z)的夹角。求两个向量的夹角可用点乘公式： $a \cdot b = |a| \cdot |b| \cdot \cos \theta$ ，简单推导可得： $\phi = \cos^{-1}(\sqrt{x^2 + z^2}/g)$ 以及 $\omega = \cos^{-1}(\sqrt{y^2 + z^2}/g)$

注意，因为 arccos 函数只能返回正值角度，因此还需要根据不同情况来取角度的正负值。当 y 轴为正时，Roll 角要取负值，当 x 轴为负时，Pitch 角要取负值。

5.5.3 yaw 角的问题

因为没有参考量，所以无法求出当前的 Yaw 角的绝对角度，只能得到 Yaw 的变化量，也就是角速度 GYR_Z。当然，我们可以通过对 GYR_Z 积分的方法来推算当前 Yaw 角（以初始值为准），但由于测量精度的问题，推算值会发生漂移，一段时间后就完全失去意义了。然而在大多数应用中，比如无人机，只需要获得 GRY_Z 就可以了。

如果必须要获得绝对的 Yaw 角，那么应当选用 MPU9250 这款九轴运动跟踪芯片，它可以提供额外的三轴罗盘数据，这样我们就可以根据地球磁场方向来计算 Yaw 角了，具体方法此处不再赘述。

5.6 数据处理与实现

MPU6050 芯片提供的数据夹杂有较严重的噪音，在芯片处理静止状态时数据摆动都可能超过 2%。除了噪音，各项数据还会有偏移的现象，也就是说数据并不是围绕静止工作点摆动，因此要先对数据偏移进行校准，再通过滤波算法消除噪音。对于夹杂了大量噪音的数据，卡尔曼滤波器的效果

无疑是最好的。本文不想考虑算法细节，大家可以参考

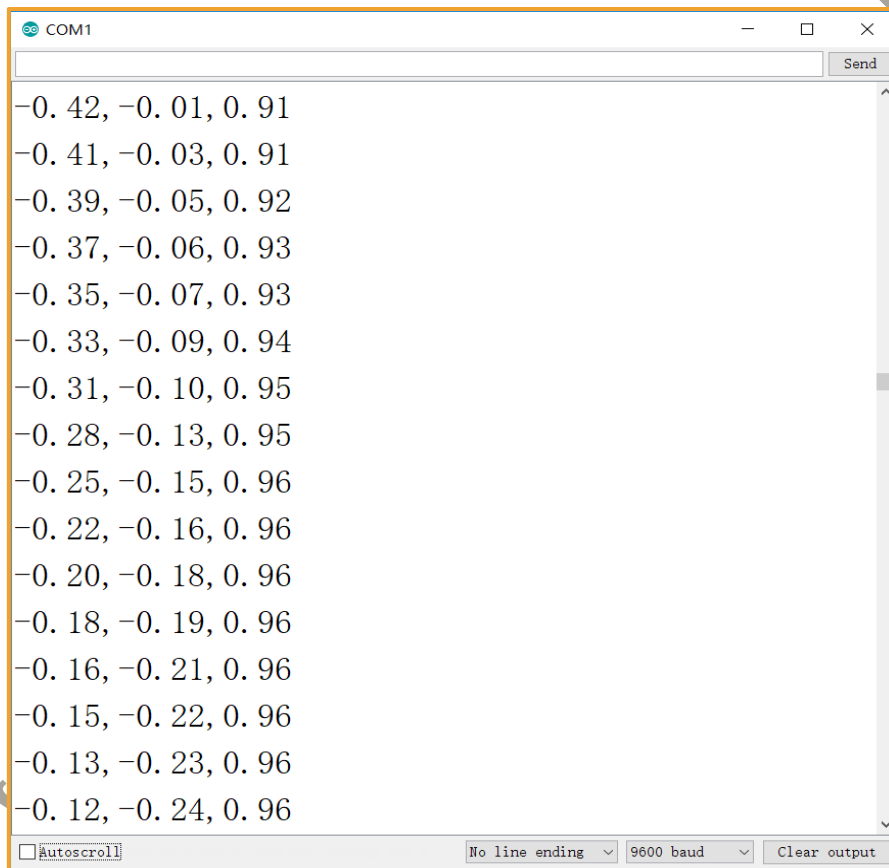
http://www.starlino.com/imu_kalman_arduino.html 可以直接使用 Arduino 的 Klamen Filter 库完成。

5.6.1 实验三 imu_kalman 得到 Roll 和 Pitch

通过读取 mpu6050 实时校验过后的加速度 ACCEL_X, ACCEL_Y, ACCEL_Z 数据和陀螺仪数据 GYRO_X, GYRO_Y, GYRO_Z 数据，经过滤波算法计算出 Roll 角和 Pitch 角传输给 processing 程序实时显示 mpu6050 的 3D 运动状态。

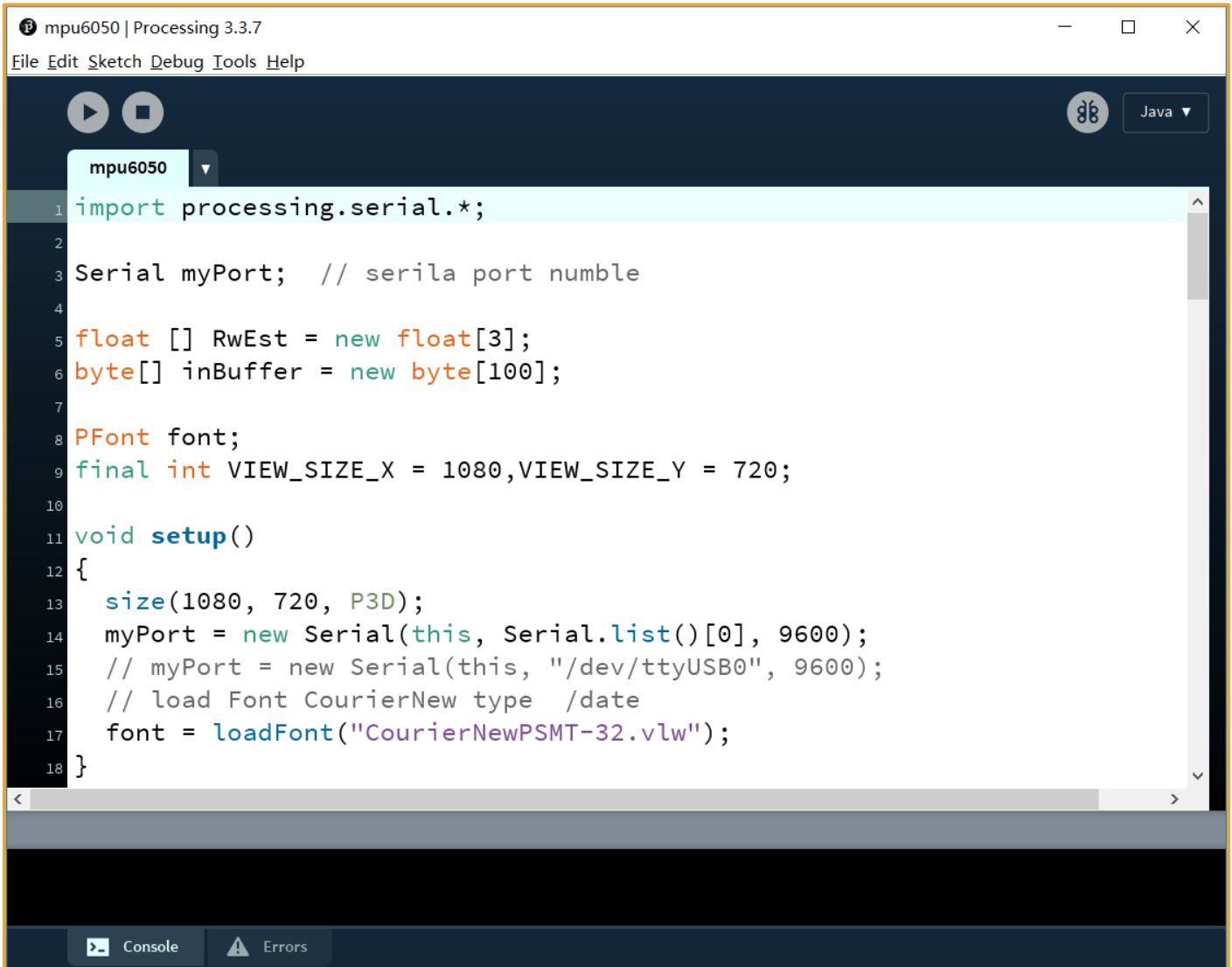
5.6.2 实验代码

Arduino 实验代码 “**MotionTrack\Lesson\mpu6050\mpu6050.ino**” 可以得到 roll 和 pitch。
运行结果如下



```
COM1
-0.42, -0.01, 0.91
-0.41, -0.03, 0.91
-0.39, -0.05, 0.92
-0.37, -0.06, 0.93
-0.35, -0.07, 0.93
-0.33, -0.09, 0.94
-0.31, -0.10, 0.95
-0.28, -0.13, 0.95
-0.25, -0.15, 0.96
-0.22, -0.16, 0.96
-0.20, -0.18, 0.96
-0.18, -0.19, 0.96
-0.16, -0.21, 0.96
-0.15, -0.22, 0.96
-0.13, -0.23, 0.96
-0.12, -0.24, 0.96
```

上传程序到 Arduino NANO 主控板之后，用 Processing 软件(下载地址 <https://www.processing.org>) 打开 Processing 程序 “**MotionTrack\Processing_demo\mpu6050\mpu6050.pde**”。注意这里的 “[]” 里的数字不是 Arduino NANO 的端口号，而是通信端口的序号，你需要打开计算机的设备管理器来查看这个序号，比如我的显示为 COM1，而我们 Processing 用的串口是从下标 0 开始。所以我把 Processing 主程序中 “[]” 里面的值修改成 0 就行。修改完成之后，请点击 Run Sketch 运行 Processing。



```
mpu6050 | Processing 3.3.7
File Edit Sketch Debug Tools Help

mpu6050
1 import processing.serial.*;
2
3 Serial myPort; // serial port number
4
5 float [] RwEst = new float[3];
6 byte[] inBuffer = new byte[100];
7
8 PFont font;
9 final int VIEW_SIZE_X = 1080, VIEW_SIZE_Y = 720;
10
11 void setup()
12 {
13   size(1080, 720, P3D);
14   myPort = new Serial(this, Serial.list()[0], 9600);
15   // myPort = new Serial(this, "/dev/ttyUSB0", 9600);
16   // load Font CourierNew type /date
17   font = loadFont("CourierNewPSMT-32.vlw");
18 }
```

```
import processing.serial.*;

Serial myPort; // serial port number

float [] RwEst = new float[3];
byte[] inBuffer = new byte[100];

PFont font;
final int VIEW_SIZE_X = 1080, VIEW_SIZE_Y = 720;

void setup()
{
    size(1080, 720, P3D);
    myPort = new Serial(this, Serial.list()[0], 9600);
    // myPort = new Serial(this, "/dev/ttyUSB0", 9600);
    // load Font CourierNew type /date
    font = loadFont("CourierNewPSMT-32.vlw");
}

void readSensors() {
    if (myPort.available() > 0) {
        if (myPort.readBytesUntil('\n', inBuffer) > 0) {
            String inputString = new String(inBuffer);
            String [] inputStringArr = split(inputString, ',');
            RwEst[0] = float(inputStringArr[0]);
            RwEst[1] = float(inputStringArr[1]);
            RwEst[2] = float(inputStringArr[2]);
        }
    }
}
```

```
void buildBoxShape() {  
    //box(60, 10, 40);  
    noStroke();  
    beginShape(QUADS);  
  
    //Z+  
    fill(#00ff00);  
    vertex(-30, -5, 20);  
    vertex(30, -5, 20);  
    vertex(30, 5, 20);  
    vertex(-30, 5, 20);  
  
    //Z-  
    fill(#0000ff);  
    vertex(-30, -5, -20);  
    vertex(30, -5, -20);  
    vertex(30, 5, -20);  
    vertex(-30, 5, -20);  
  
    //X-  
    fill(#ff0000);  
    vertex(-30, -5, -20);  
    vertex(-30, -5, 20);  
    vertex(-30, 5, 20);  
    vertex(-30, 5, -20);  
  
    //X+  
    fill(#ffff00);  
    vertex(30, -5, -20);  
    vertex(30, -5, 20);  
    vertex(30, 5, 20);  
    vertex(30, 5, -20);  
  
    //Y-  
    fill(#ff00ff);  
    vertex(-30, -5, -20);  
    vertex(30, -5, -20);  
    vertex(30, -5, 20);  
    vertex(-30, -5, 20);  
}
```

```
//Y+
fill(#00ffff);
vertex(-30, 5, -20);
vertex(30, 5, -20);
vertex(30, 5, 20);
vertex(-30, 5, 20);

endShape();
}

void drawCube() {
  pushMatrix();
  // normalize3DVec(RwEst);
  translate(300, 450, 0);
  scale(4, 4, 4);
  rotateX(HALF_PI * -RwEst[1]);
  //rotateY(HALF_PI * -0.5);
  rotateZ(HALF_PI * -RwEst[0]);
  buildBoxShape();
  popMatrix();
}

void draw() {
  // getInclination();
  readSensors();
  background(#214565);
  fill(#ffffff);
  textFont(font, 20);
  text("RwEst :\n" + RwEst[0] + "\n" + RwEst[1] + "\n" + RwEst[2], 220, 180);
  // display axes
  pushMatrix();
  translate(450, 250, 0);
  stroke(#ffffff);
  // scale(100, 100, 100);
  line(0, 0, 0, 100, 0, 0);
  line(0, 0, 0, 0, -100, 0);
  line(0, 0, 0, 0, 0, 100);
  line(0, 0, 0, -RwEst[0], RwEst[1], RwEst[2]);
  popMatrix();
  drawCube();
}
```


我们可以看到我们运行结果如下：

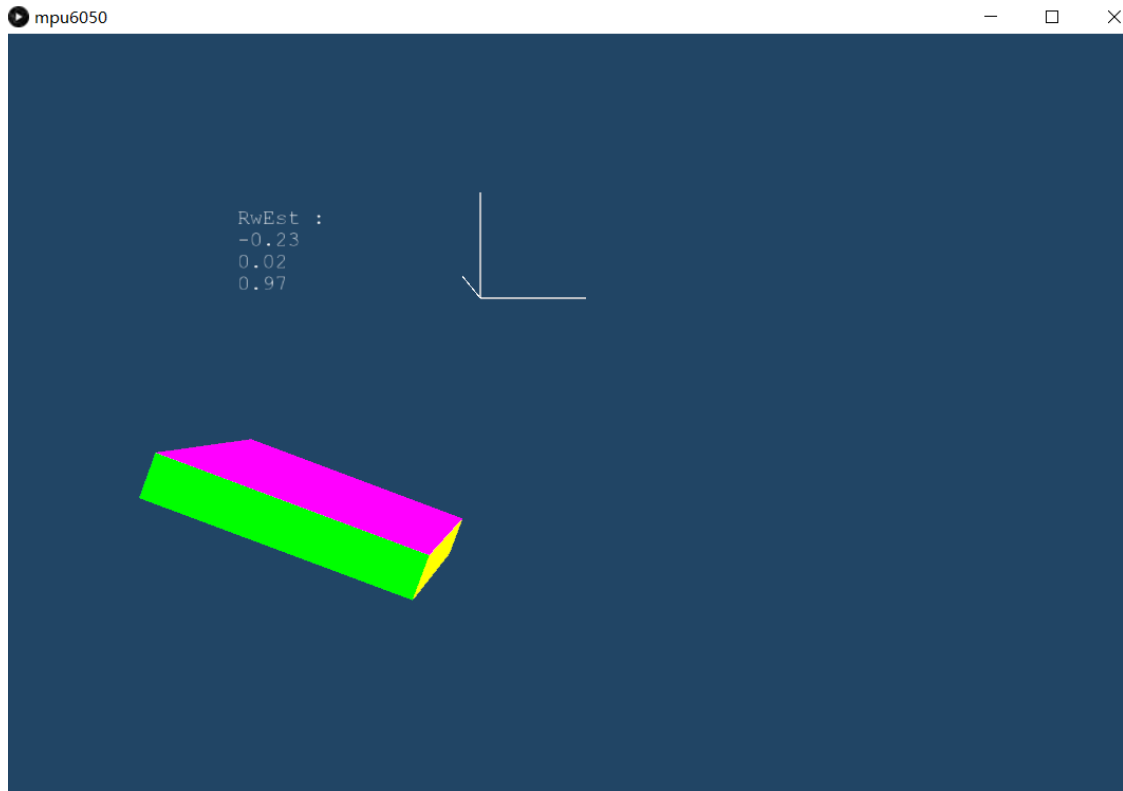


图 17: Processing 演示效果图

6、运动追踪控制原理

通过前面对 mpu6050 数据采集得到 Roll 角和 Pitch 我们建立如下对应关系，
控制下位机运动物体我们是通过 mpu6050 得到速度和方向就可以控制任何下位机物体。

通过读取 mpu6050 实时校验过后的加速度 ACCEL_X, ACCEL_Y, ACCEL_Z 数据和陀螺仪数据 GYRO_X, GYRO_Y, GYRO_Z 数据传输给四元数处理函数，程序实时显示 mpu6050 的 Roll 角和 Pitch 角状态。

6.1 赛车方向坐标角模型

为了方便控制赛车我们现在建立如下的赛车运动方向用坐标角表示的模型

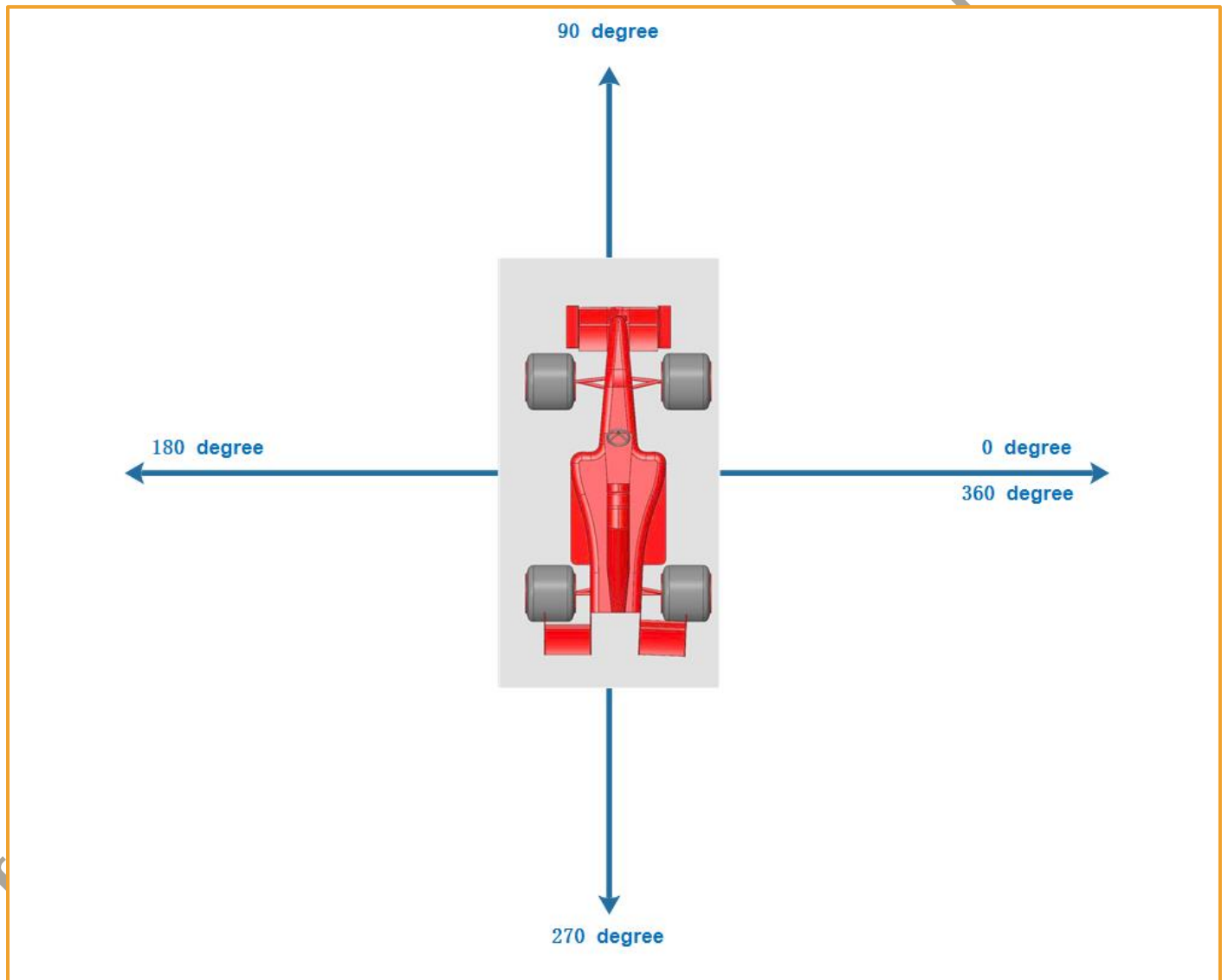


图 18：赛车方向角度坐标

如上图，我们将小车正前方定义为 90 度，正后方为 270 度，正右方 180 度，正左方 0/360 度。
0~90 代表右前方向 90~180 代表左前 180~270 代表左后，270~360 代表右后。



深圳启汇科技有限公司
Shenzhen Keywish Technology Complay Limited

Room 416 building A, huafeng business center, republican industrial road,
xixiang, bao 'an district, Shenzhen, GuangDong, China.

Keywish Technology

同样我们将 mpu6050 安装到运动手套上我们建立如下的运动模型

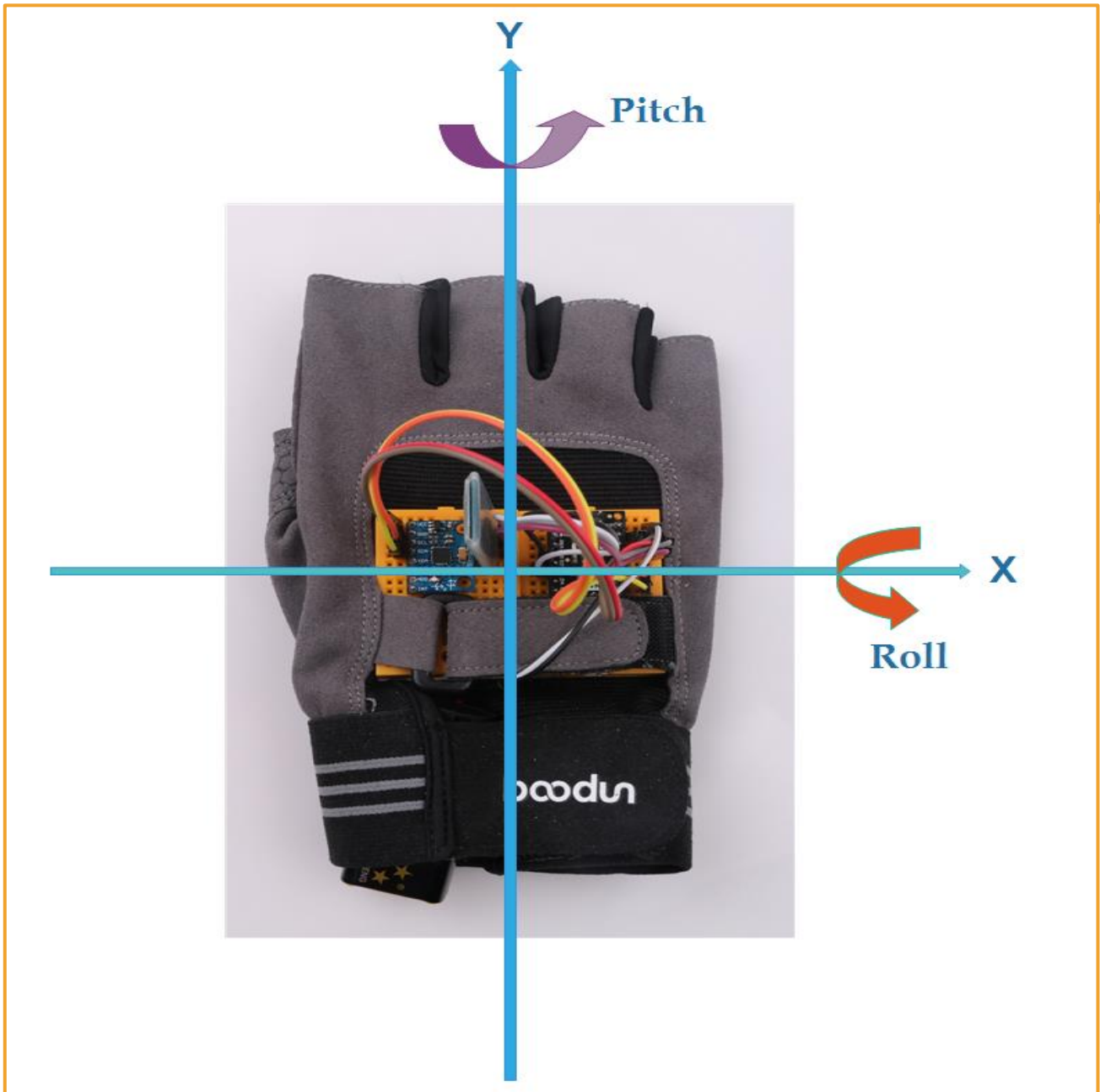


图 19: 运动手套的方向坐标

通过 5.5.2 Roll-pitch-yaw 数据模型，运动手套围绕图示 Y 旋转（左右转动手背）我们得到 Pitch 角
围绕 X 旋转（前后翻转）我们得到 Roll 角。

现在我们对坐标已经对姿态角和赛车的角坐标有所了解，现在具体到场景来描述。

当运动手套向右前方向倾斜时，如下图所示，如果按照坐标系就是向 0~90 度区域倾斜。

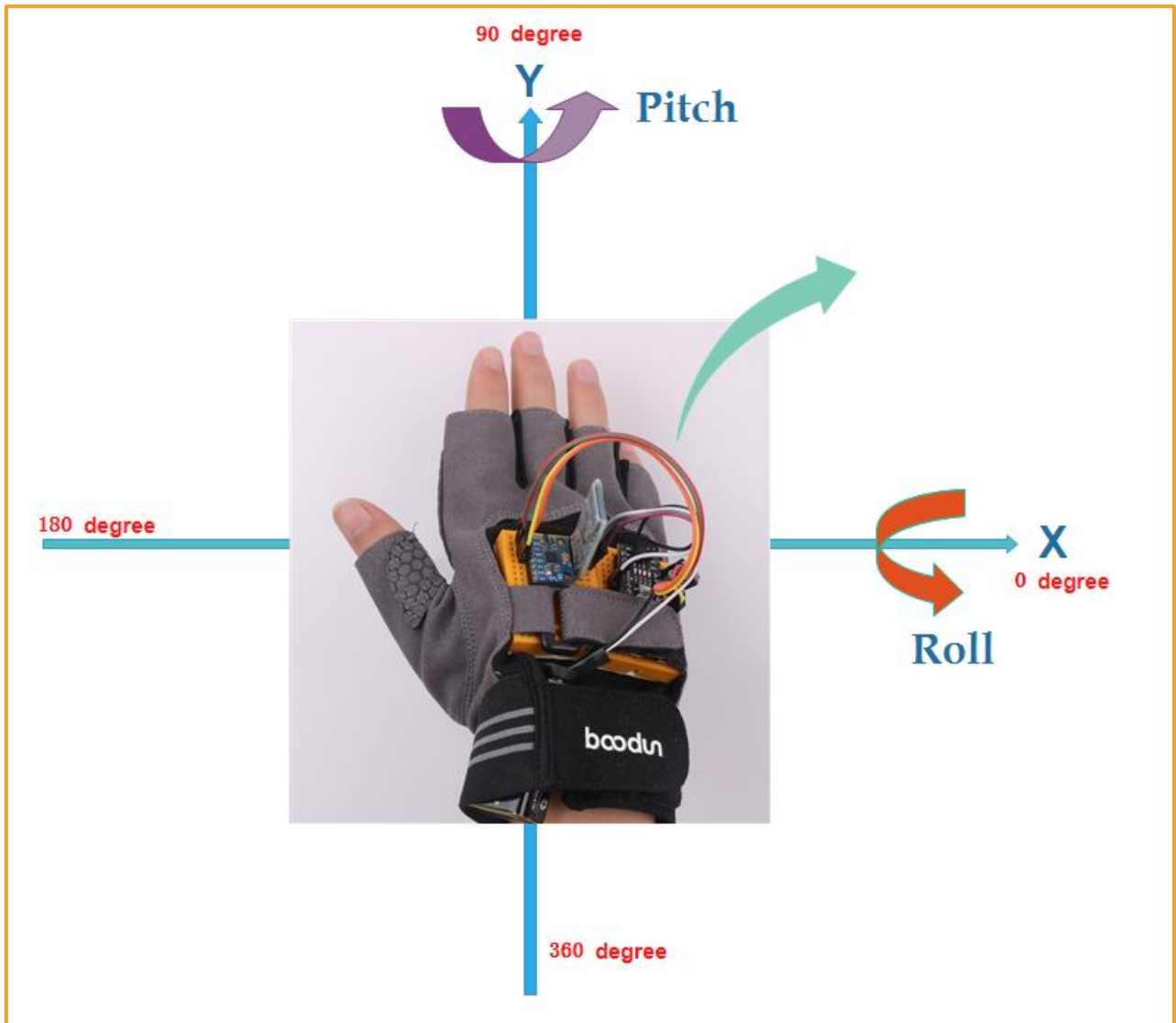


图 20: 运动手套右前方向倾斜示意图

对应下车的转角如下所示

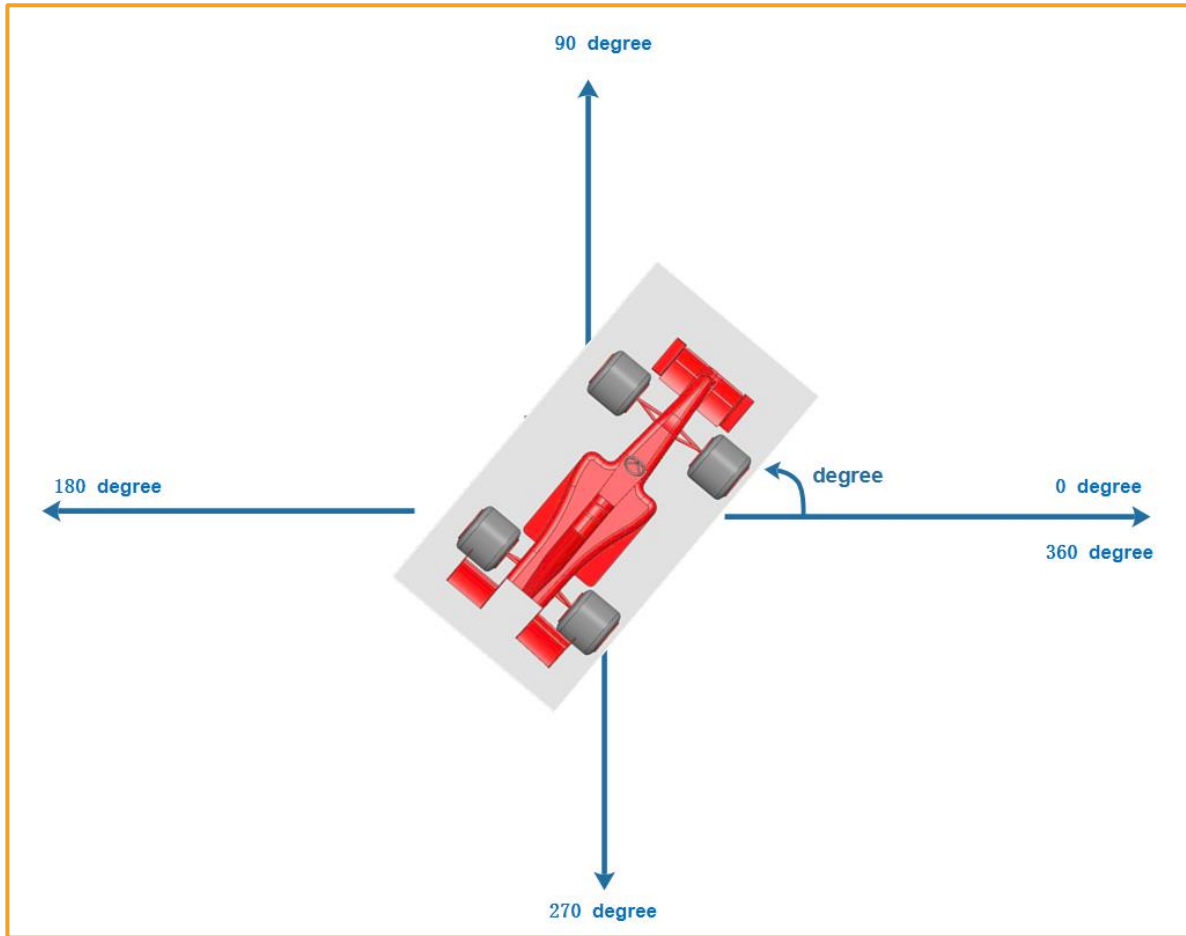


图 21：赛车右前方运动示意图

通过运动手套的姿态，我们需要计算出下位机小车运动方向角度 degree 。于是我们建立如下立体模型

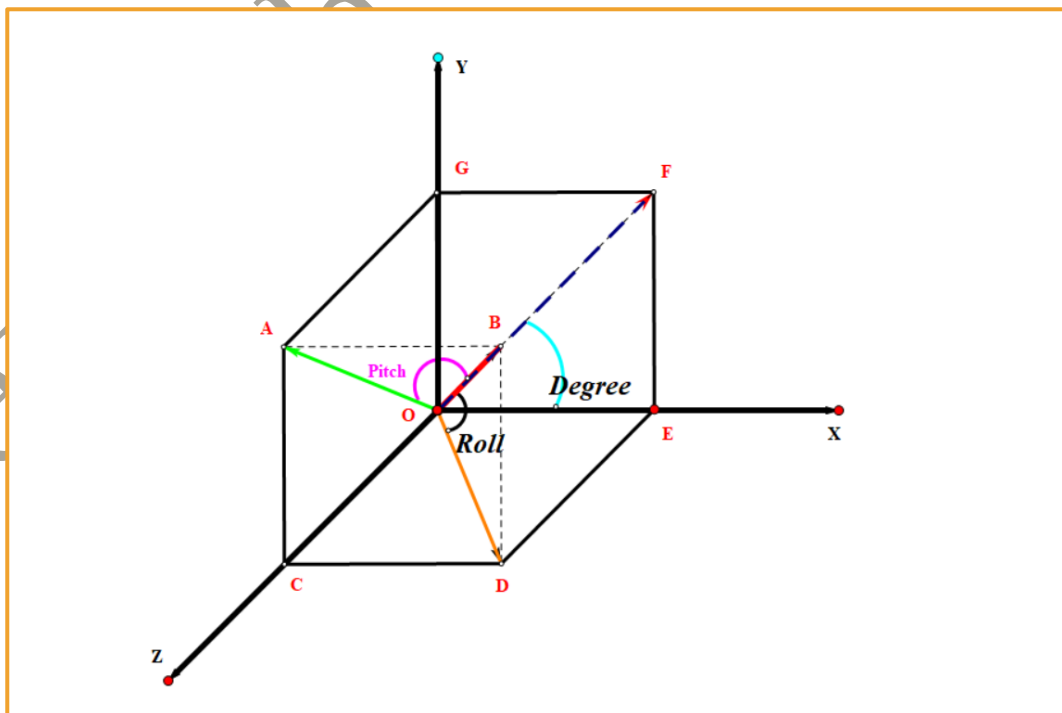


图 22：根据运动手套建立的立体模型图

为了方便表示，上图坐标系的 Z 轴正方向（运动手套正面）向上，X 轴正方向（运动手套右边）向右。初始状态，运动手套和 XY 平面水平。

此时运动手套倾斜的方向 OB 表示时，Roll 角 $\angle EOD$ （黑色）为加速度向量 OB 与其在 XZ 平面上投影(x,0,z)的夹角，Pitch 角 $\angle BOA$ （紫色）与其在 YZ 平面上投影(0,y,z)的夹角。蓝色为 degree 为运动手套投射在 XY 平面上投影(x,o,y)的夹角。我们用这个角度控制下位机赛车的转向角。通过前面的 5.6.1 实验我们已经得到 Roll 和 Pitch 我们现在来通过下面公式计算 degree.

图中： $BA \perp OA$ $AB = OB \times \sin(\text{Pitch})$

$BD \perp OD$ $BD = OB \times \sin(\text{Roll})$

$$\tan(\text{degree}) = \frac{FE}{OE} = \frac{BD}{AB} = \frac{\sin(\text{Roll})}{\sin(\text{Pitch})} \approx \frac{\text{Roll}}{\text{Pitch}}$$

$$\text{degree} = \arctan\left(\frac{\text{Roll}}{\text{Pitch}}\right)$$

上面我们已经得到了转向余弦角度，我们转换成坐标度数还需要 $2\pi = 360^\circ$ 我们需要再乘以系数 $\frac{180}{\pi} = 57.3$

上面的分析只是当像右上倾斜的时候的角度分析。同理当运动手套向左前倾斜的时候

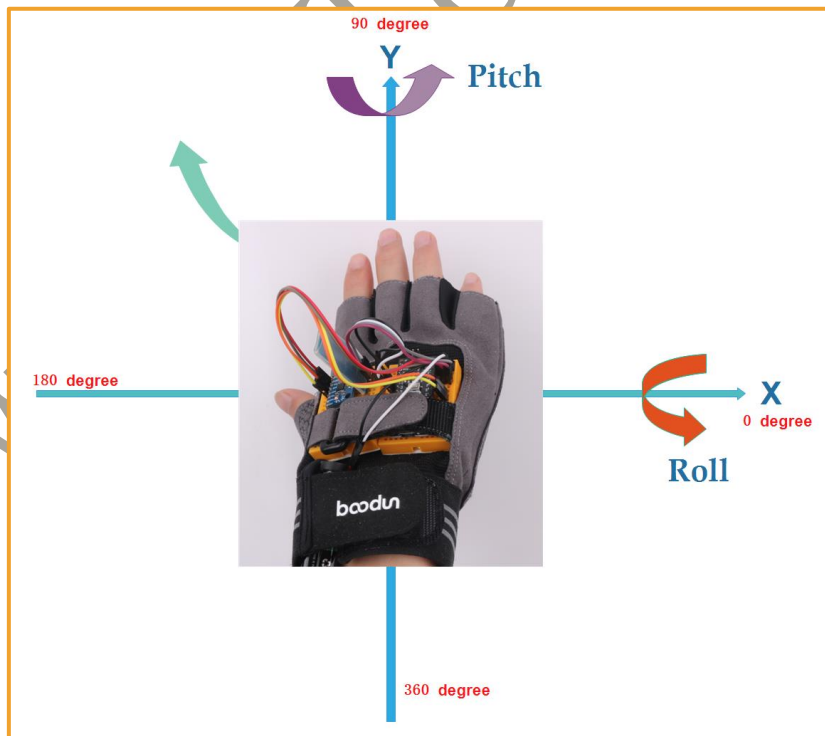


图 23：运动手套左前方向倾斜示意图

$$\text{degree} = \arctan\left(-\frac{\text{Pitch}}{\text{Roll}}\right) * 57.3 + 90^\circ$$

同理当运动手套向左后倾斜的时候

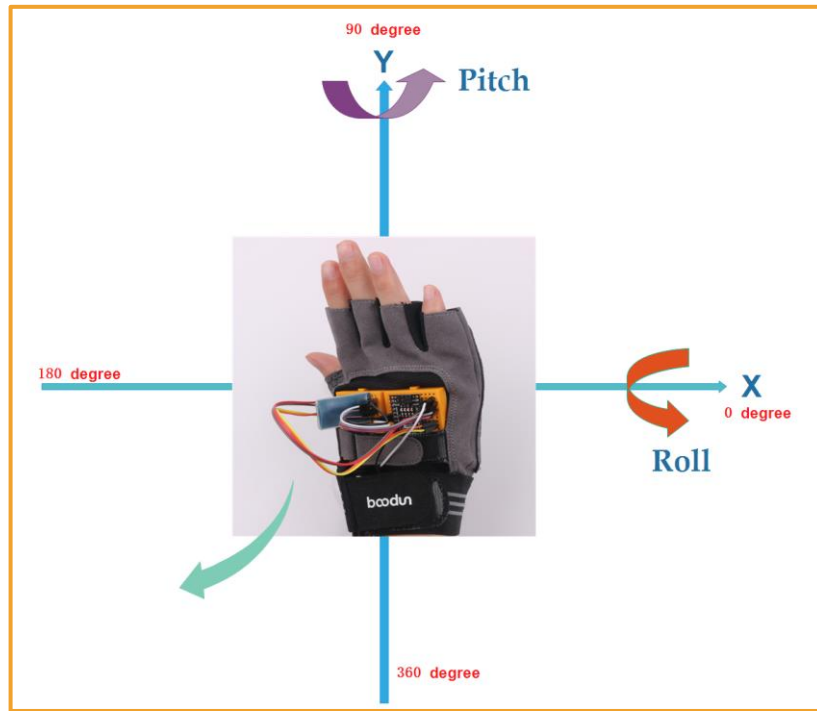


图 24：运动手套左后方向倾斜示意图

$$\text{degree} = \arctan\left(\frac{\text{Roll}}{\text{Pitch}}\right) * 57.3 + 180^\circ$$

同理当运动手套向右下倾斜的时候

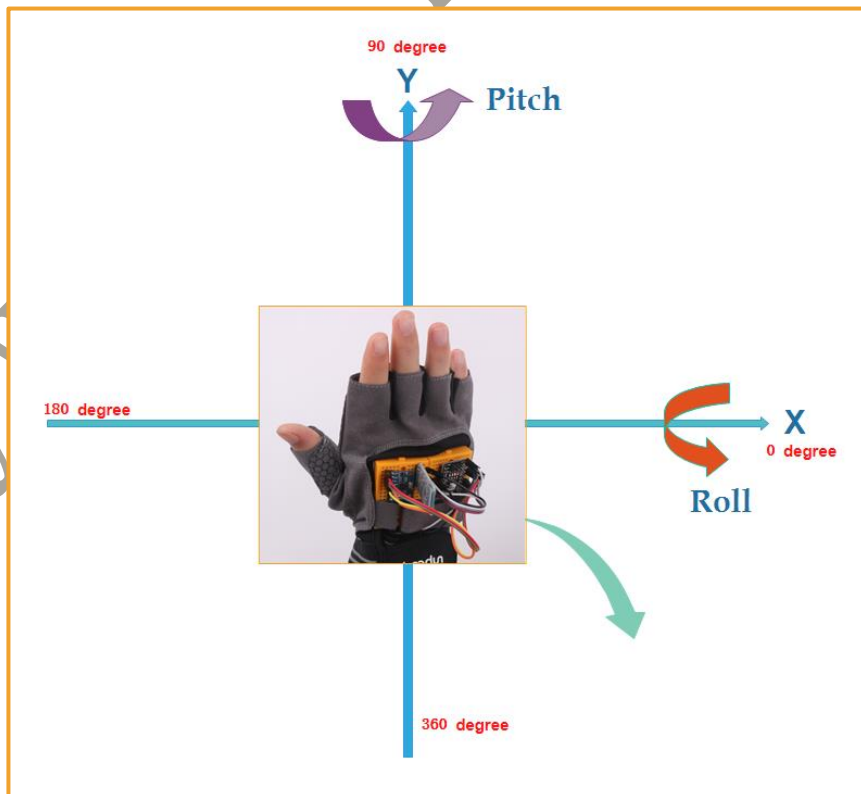
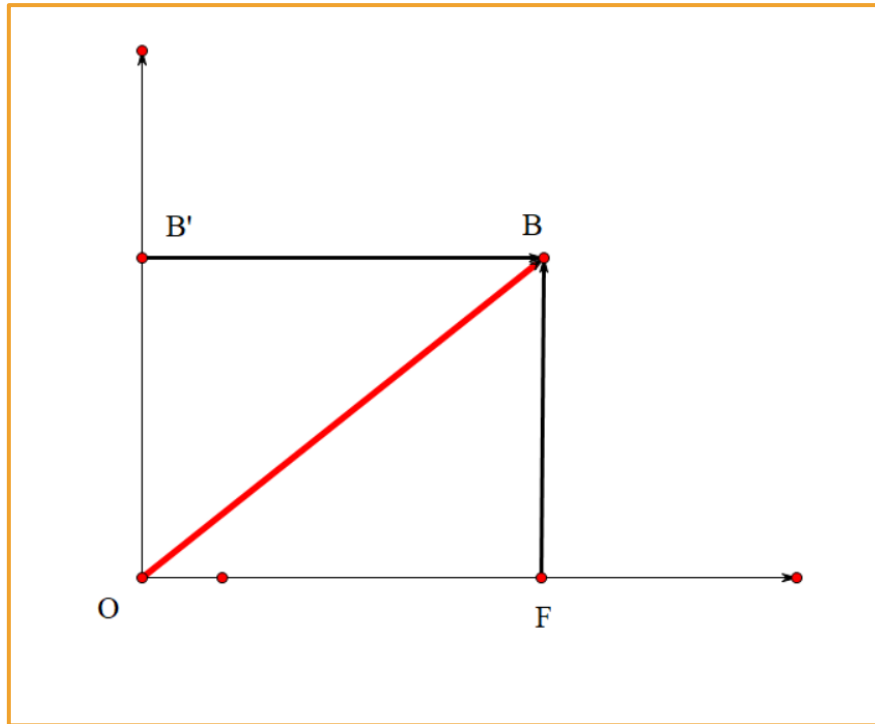


图 25: 运动手套右后方向倾斜示意图

$$\text{degree} = \arctan\left(-\frac{\text{Pitch}}{\text{Roll}}\right) * 57.3 + 270^\circ$$

现在我们来探讨一下如何控制赛车的速度

从上图我们可以看到 $\angle BOF$ 为运动手套在 mpu6050 和平面 XY 平面倾斜角度，我们用如下平面来表示



$\angle B'OB$ 为运动手套相对初始状态倾斜的角度，用这个倾斜程度来控制赛车的速度。

$$\angle B'OB = \angle OBF = \arcsin\left(\frac{OF}{OB}\right) = \arcsin\left(\frac{\sqrt{OE^2 + EF^2}}{OB}\right)$$

$$EF = BD \quad OE = AB$$

$$\angle OBF = \arcsin(\sqrt{\sin^2(\text{Roll}) + \sin^2(\text{Pitch})}) \approx \arcsin(\sqrt{\text{Roll}^2 + \text{Pitch}^2})$$

上面得到的是弧度角同样我们需要再乘以系数 $\frac{180}{\pi} = 57.3$ ，这个角度的范围为 $(0^\circ \sim 90^\circ)$ 我们直接通过这个角度来作为赛车的控制速度。

这样我们得到方向和速度的控制算法如下

```
void HandInclination(void)
{
    static int count = 0;
    static int SendSpeed = 0, SendDegree = 90;
    count++;
    speed = CalculateSpeed(roll, pitch);
    if ((-0.2 <= pitch) && (pitch <= 0.2) && (-0.2 <= roll) && (roll <= 0.2)) {
        speed = 0;
        SendDegree += 90;
    } else if (pitch < 0 && roll < 0) {
        degree = atan(roll/pitch)*Rad;
        SendDegree += ((unsigned int)(degree/10))*10;
    } else if (pitch > 0 && roll < 0) {
        degree = atan(-pitch/roll)*Rad + 90;
        SendDegree += ((unsigned int)(degree/10))*10;
    } else if (pitch > 0 && roll > 0) {
        degree = atan(roll/pitch)*Rad + 180;
        SendDegree += ((unsigned int)(degree/10))*10;
    } else if (pitch < 0 && roll > 0) {
        degree = atan(-pitch/roll)*Rad + 270;
        SendDegree += ((unsigned int)(degree/10))*10;
    } else {
        SendDegree += 90;
    }
    SendSpeed = (unsigned int)(speed/20)*20;

    if (degree < 30 || degree > 330) {
        SendDegree = 0;
    }
    if (count >= 3) {
        count = 0;
        Send_Direction(SendDegree/3);
        delay(5);
        Send_Speed(SendSpeed);
        SendDegree = 0;
        SendSpeed = 0;
    }
}
```

7. 通讯协议

无论是使用蓝牙还是无线控制赛车，我们需要设计如下通信协议，运动手套和 Arduino 之间实现完美的交互。主要流程是：Android 端识别到控制命令后并将命令打包成对应的数据包，然后发送给蓝牙模块（JDY-16），JDY-16 接收到数据后通过串口传输给 Arduino，接着 Arduino 对数据进行解析，然后执行相应的动作。其中上位机端（Android）发送的数据格式如下，主要包含 8 个字段：

协议首部	数据长度	设备类型	设备地址	功能码	控制数据	校验和	协议尾部
------	------	------	------	-----	------	-----	------

在上面 8 个字段中：

我们用一个结构体来表示

```
typedef struct
{
    unsigned char start_code;    // 8bit 0xAA
    unsigned char len;
    unsigned char type;
    unsigned char addr;
    unsigned short int function; // 16 bit
    unsigned char *data;        // n bit
    unsigned short int sum;      // check sum
    unsigned char end_code;      // 8bit 0x55
}ST_protocol;
```

“协议首部”就是数据包的开始部分，比如说统一指定为 0xAA。

“数据长度”除数据起始和结束码的有效数据长度。

“指令类型”表示从机设备的类型。

“设备地址”这个为控制设置的地址。

“功能码”需要控制的设备功能类型，我们目前支持的功能类型如下

```
typedef enum
{
    E_BATTERY = 1,
    E_LED = 2,
    E_BUZZER = 3,
    E_INFO = 4,
    E_ROBOT_CONTROL = 5,
    E_ROBOT_CONTROL_SPEED = 6,
    E_TEMPERATURE = 7,
    E_IR_TRACKING = 8,
    E_ULTRASONIC = 9,
    E_VERSION = 10,
    E_UPGRADE = 11,
} E_CONTROL_FUNC ;
```

“数据”对于小车的具体控制数值，比如速度，角度。

“校验和”是“控制指令”各个数据位进行异或计算的结果。

“协议尾部”就是数据包的结束部分，当收到这个数据时，表示这个数据包已经发送结束了，准备接收下一个数据包，这里我们统一指定为 0x55。

例如：一个完整的数据包可以是这样的“AA 07 01 01 06 50 00 5F 55”，其中：

“07” Transmission Data Length 7 bytes

“06”是“设备类型”就是指定是哪一种设备，如电机、LED、蜂鸣器等；这里的 06 是指传输“速度”，05 是指传输“方向”。

“50（或 0050）”是控制数据，其中 0x50 是十六进制，转换为二进制为 80，如果“设备类型”传输的 06，那么这里的数据就是速度值，即速度为 80。如果传输 05 时就为控制方向，即 80° 方向（向前）。

“005F”是校验和，即 $0x07+0x01+0x01+0x06+0x50=0x5F$ 。

“55”为协议尾部，表示数据传输结束。

8. 综合实验:

8.1 Nrf24L01 无线控制

8.1.1 运动手套控制 Hummer-bot 智能车

按照上个章节的连接线路进行连接，然后给 Arduino NANO 接上 9v 干电池，然后下载好程序。上电后 Nrf24L01 模块发送数据到 Hummer-Bot Nrf24L01 上，然后通过调整手套的位置去控制 Hummer-Bot 多功能小车，这样，一个运动手套控制小车就诞生了！

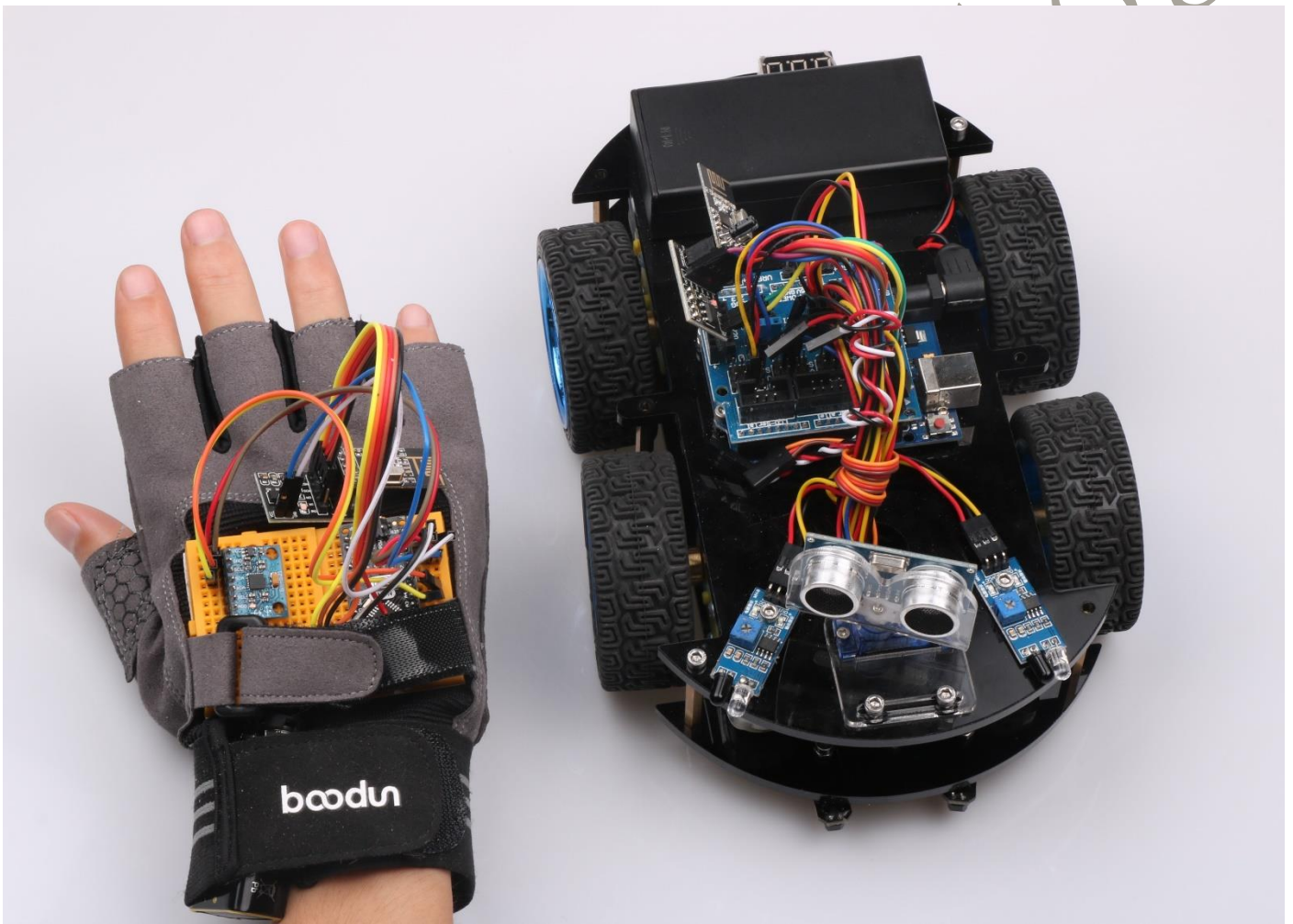


图 26: 手势控制 hummer-bot

下位机的程序链接: <https://github.com/keywish/keywish-hummer-bot>

Hummer-Bot 多功能小车购买链接: Buy on Amazon: <https://www.amazon.com/dp/B078WM15DK>

控制演示视频

8.1.2 运动控制 Beetle-bot 智能车

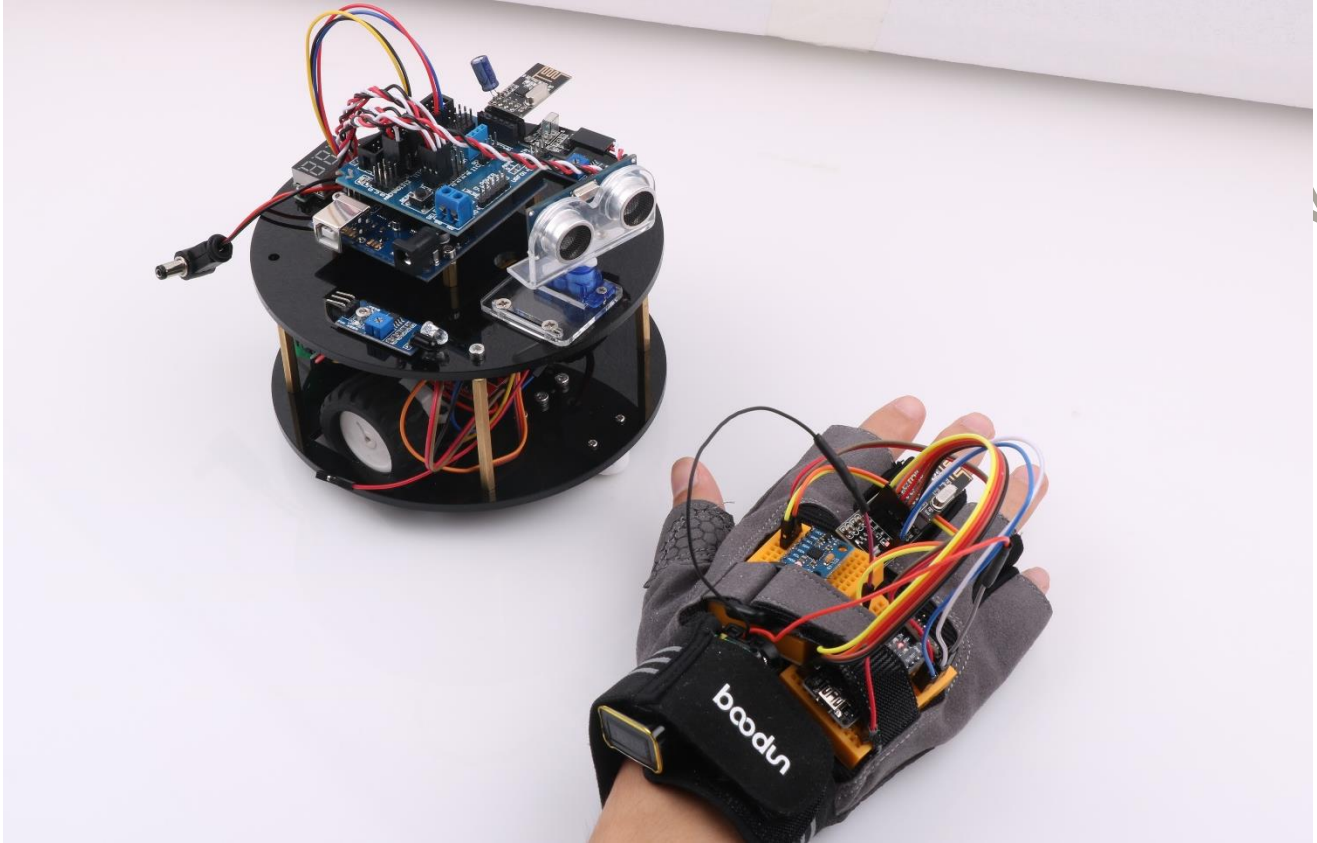


图 27：手势控制 Beetle-bot

Beetle-bot 的资料 <https://github.com/keywish/keywish-beetle-bot>

8.2 蓝牙通讯方式控制

按照蓝牙的连接线进行连接，然后给 Arduino NANO 接上 9v 干电池，然后下载程序 (MotionTrack\Lesson\MotionTrack_Bluetooth\ MotionTrack_Bluetooth.ino)。注意需要修改下位机蓝牙的 mac 地址，上电后运动手套自动连接赛车的蓝牙模块。

8.2.1 运动手套控制 Aurora-Racing



图 28: 手势控制 Aurora-Racing

产品资料: <https://github.com/keywish/Aurora-Racing>

制视频见: [gitee\MotionTrack\video\Control_RacingCar.mp4](#)

购买链接: <https://www.amazon.com/dp/B07DDG2YQQ>

8.2.2 运动手套控制 Panther-tank

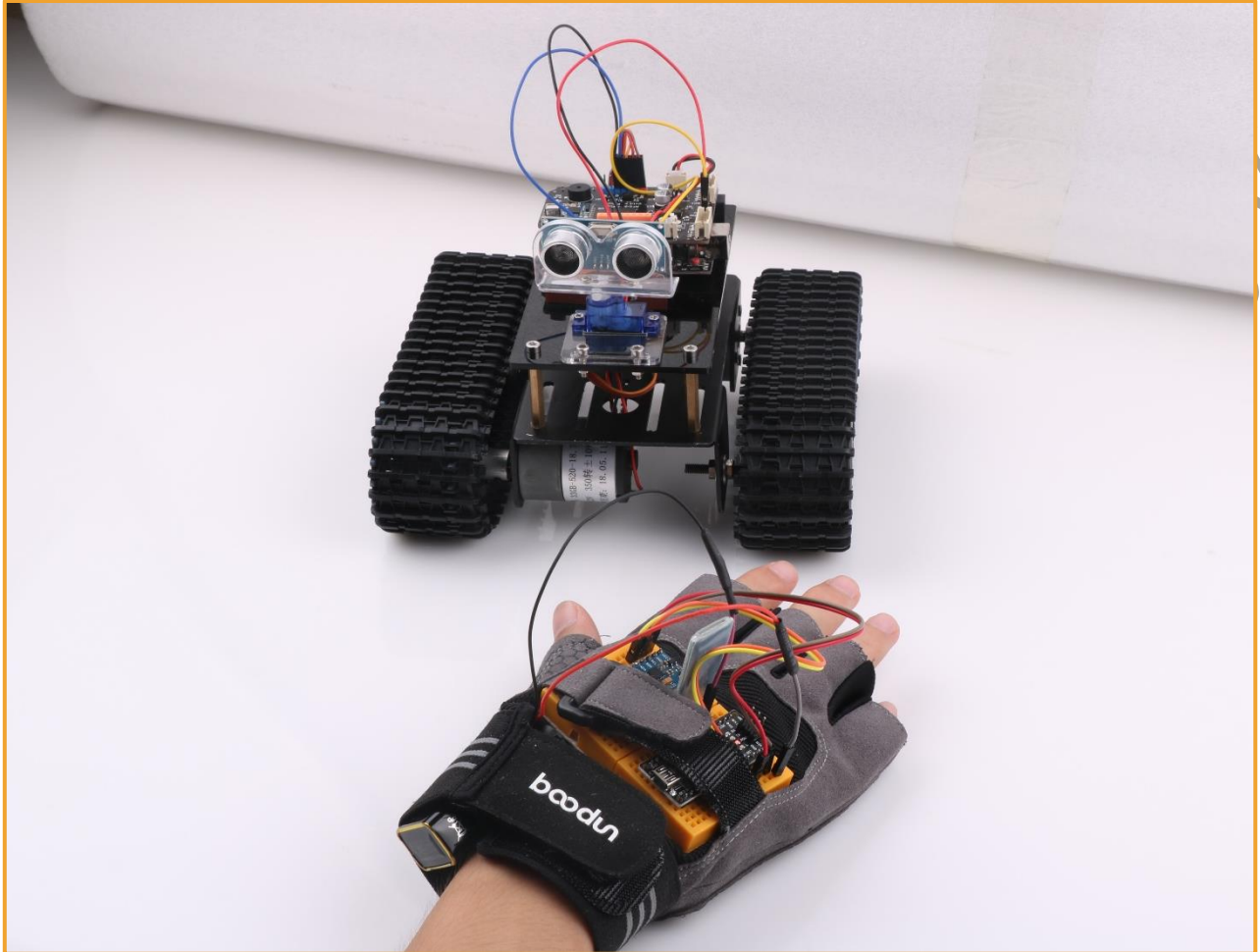


图 26: 手势控制 Pather-tank

Panther-tank 制作资料请参考 <https://github.com/keywish/keywish-panther-tank>

8.2.3 运动手套控制 balance car

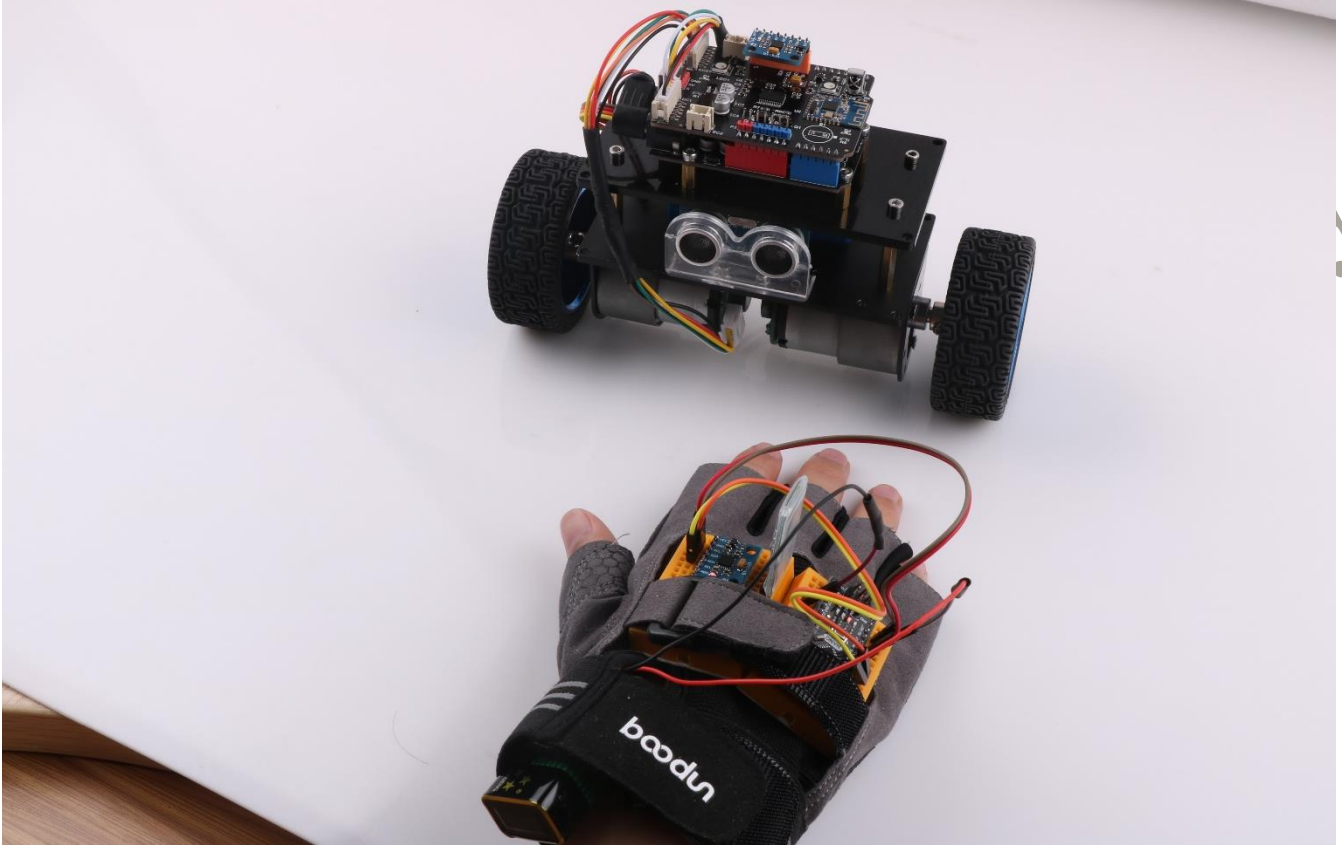


图 26: 手势控制 Mini balance-car

Balance 产品资料请参考 <https://github.com/keywish/mini-balance-car>