VRTRIX
无 远 弗 届

北京无远弗届（VRTRIX™）数据手套操作手册

Date	Modified by	Comments
2018-10-22	Guo	Init Version.
2019-01-28	Guo	Add Motion Capturing Integration Part
2019-03-10	Guo	Add Software Manual

简介

Introduction

VRTRIX™ 数据手套通过遍布全手的高性能 9 轴 MEMS 惯性传感器实时采集各指头关节运动数据，并通过反向动力学还原骨骼运动，可以在虚拟现实的场景中实现对真实手部运动的重现，并进行精细的手部运动还原和交互。每只手套上分布有 6 个传感器，双手共 12 个，可以实时高精度低延迟输出全手所有关节的运动姿态。

VRTRIX™ 惯性传感器模块采用九轴传感器（3 轴陀螺仪，3 轴加速度计，3 轴磁力计），精确高效的数据融合算法保证传感器以每秒 400Hz 的频率输出精确的姿态四元数，同时保证数据延迟低于 5ms。

VRTRIX™ 数据手套还搭载无线传输功能，双手传感器数据可以通过手背上的无线发射模块实时发送给 pc 并进行渲染。无线传输采用 2.4GHz 专有协议，安全高效延迟不超过 10ms。同时，系统进行了低功耗设计，数据手套不间断使用情况下的电池续航时间可以达到 16 小时以上。

特性

Features

- VRTRIX™ 数据手套内置 12 个高性能 9 轴 MEMS 惯性传感器，实时精确解算 3DOF 手指动态姿态数据。
- VRTRIX™ 数据手套采用自主研发的精确高效的数据融合算法，保证传感器以每秒 400Hz 的频率输出精确的姿态四元数且数据延迟低于 5ms。
- VRTRIX™ 数据手套采用 2.4GHz 专有协议与主机 PC 互相连接，安全高效延迟不超过 10ms。
- VRTRIX™ 数据手套采用精美的工业设计，不只是一套简单的虚拟现实外设，而是与游戏中的任意虚拟物体进行精准的交互，创造了更强的沉浸感。
- VRTRIX™ 数据手套采用 1200mAh 大容量电池，且严格控制功耗，休眠模式和正常工作模式自动切换，可达到 30 小时以上的续航，满足虚拟现实线下体验店的需求。
- VRTRIX™ 完美兼容 HTC Vive 等主流虚拟现实头盔，同时兼容 SteamVR 平台且免费提供开发 SDK，便于内容开发和制作。

参数

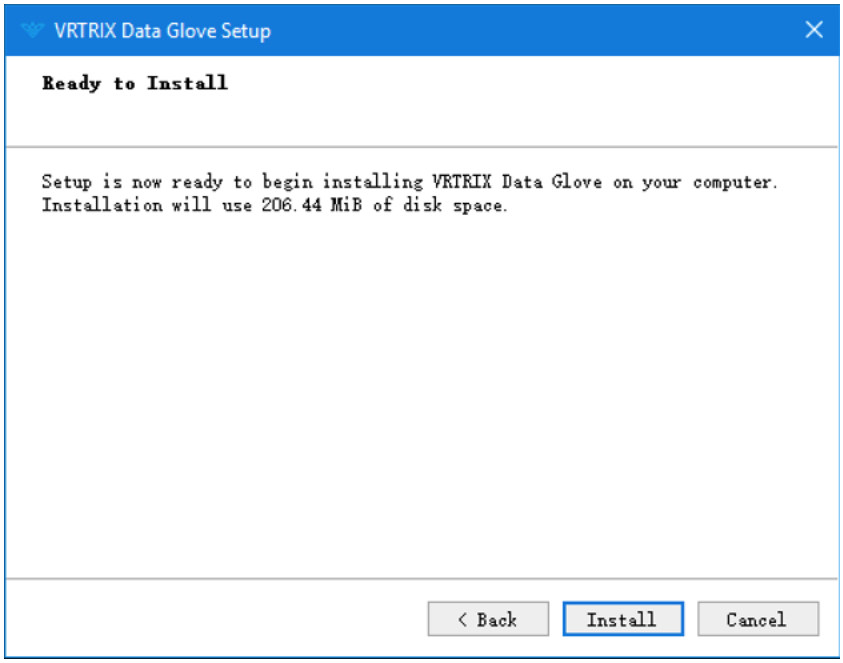
Specifications

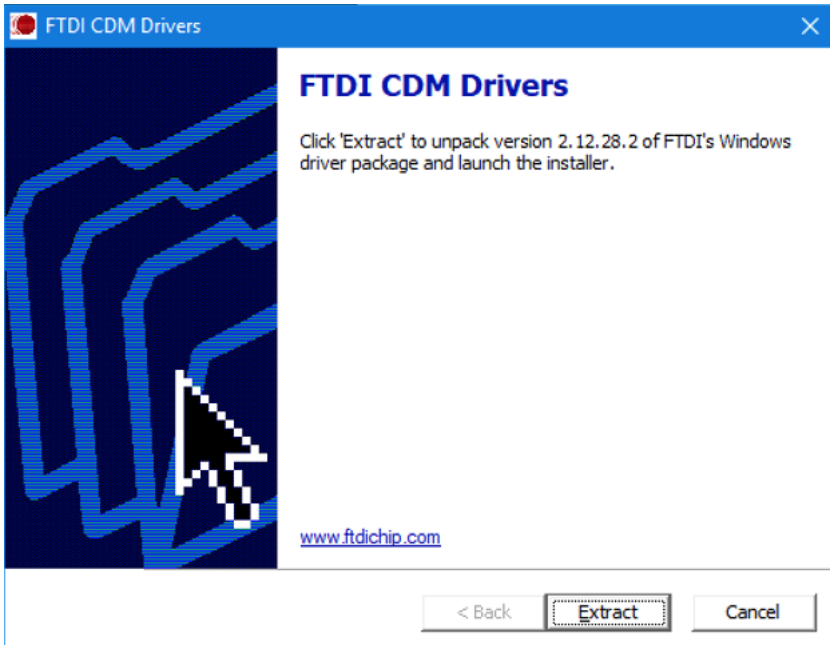
运动传感器数目	左右手各 6 个，共计 12 个
数据输出精确度（空间位置）	< 1mm
数据输出准确度（姿态角）	Yaw 轴< 2°，Pitch 轴 < 1°，Roll 轴 < 1°
数据输出精确度（姿态角）	< 1°
数据传输协议	2.4GHz 专有协议
数据输出最高支持频率	单手 200Hz
数据延迟	<10ms (无线连接)
陀螺仪数据输出频率（ODR）	1000Hz
加速度计输出频率（ODR）	800Hz
磁力计输出频率（ODR）	200Hz
陀螺仪数据输出量程（FS）	+-2000°/s
加速度计输出量程（FS）	+-16g
磁力计输出量程（FS）	+-5000uT
陀螺仪输出敏感度（Sensitivity）	16.4 LSB°/s
加速度计输出敏感度（Sensitivity）	2048 LSB/g
磁力计输出敏感度（Sensitivity）	0.15 uT/LSB
系统电压	3.3V
系统功耗	VR 工作模式峰值 < 80mA 节能模式 < 10mA
充电电流	1A
充电电压	5V
可充电锂电池容量	单手 1200mAh
续航时间	> 30 小时
使用温度	5°C 到 60°C
尺寸	8mm*10mm*2mm

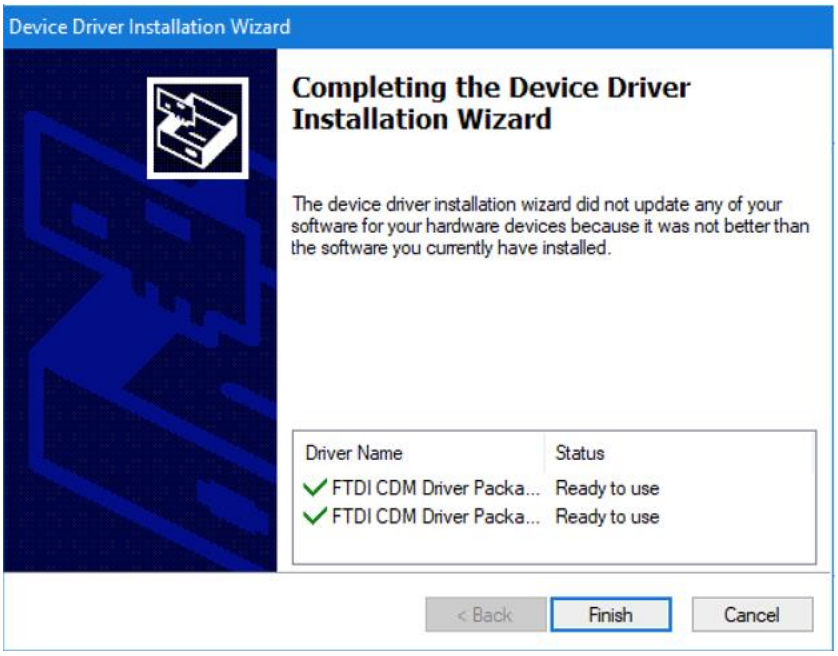
基本操作

Basics

1. **安装驱动及客户端软件**：安装软件之前，将两个 usb 接收器插入 pc，然后双击安装 VRTRIXGloveInstaller.exe，该安装程序会自动安装所需驱动，在 win10 下安装过程中会请求管理员权限，安装完成后会在桌面上生成客户端软件快捷方式。



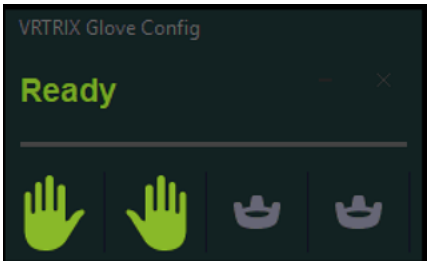




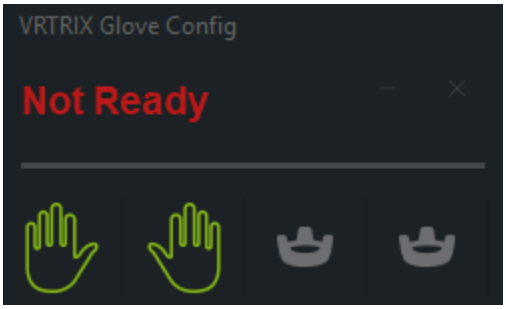
驱动安装完成后，会识别出两个新设备，并显示 Ready to use.

2. **开启/关闭设备：**短按手套壳体正面按钮即可开启设备，注意开机后手套硬件大约有 3-4s 初始化的过程，请等待初始化完毕后进行下一步操作，初始化完毕后手套壳体上方 led 灯会开始以 1s 一次的频率慢闪。如果要关闭设备，那么长按手套壳体正面的按钮，随即 led 灯会开始以 1s 五次的频率快闪，此时立刻松开按钮，设备关闭。
注意：此步骤当准备关闭设备时，看到 led 灯开始快闪后请立刻松开按钮，否则 3s 后将会进入配对模式，详见第 4 条。

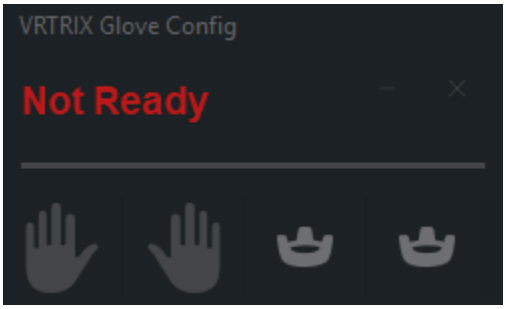
3. **检查手套状态：**双击打开桌面上生成的快捷方式（VRTRIXGloveConfigTool），如果两只手套都正常运行且连接上 usb 接收器，则图标显示常亮，状态提示 Ready。接下来设置串口配置，右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，点击 Reset Latency Timer 完成配置。此时可以跳过第 4 步，开始执行 demo 测试。



如果手套已经打开但是未和接收器进行配对（此时应该进行配对，请阅读步骤 4）或者手套处于关机状态，状态显示为下图：

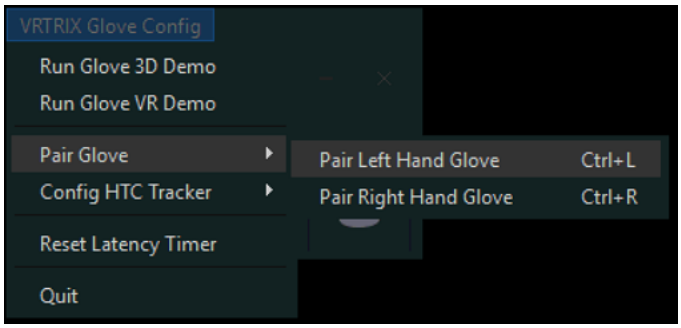


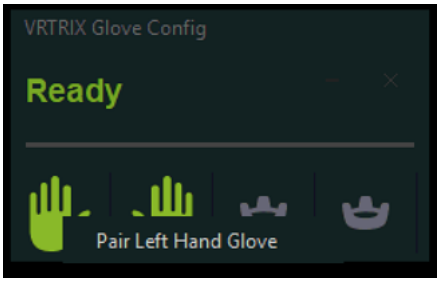
如果未找到接收器，状态显示为下图：



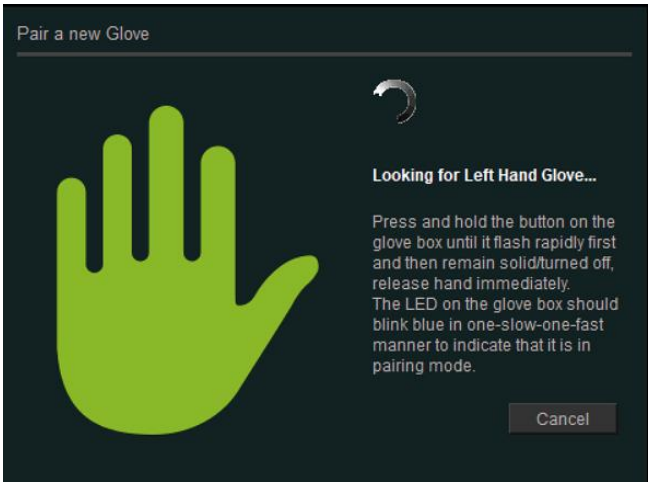
4. 配对设备（可选）：

手套出厂前预先和盒中的接收器进行了配对，所以可跳过此步骤。如果由于其他原因配对信息丢失可以使用软件对手套进行配对。右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，或者在希望配对的手套图标（左手或者右手）上点击右键，选择 Pair Left Hand Glove/Pair Right Hand Glove 来配对对应的手套。

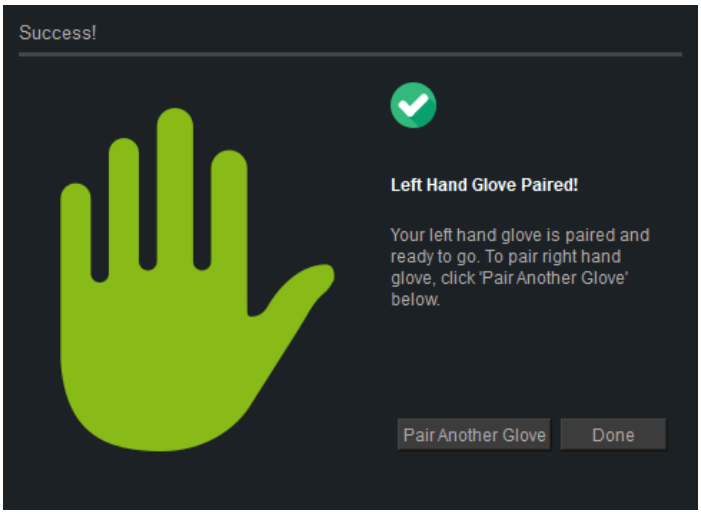




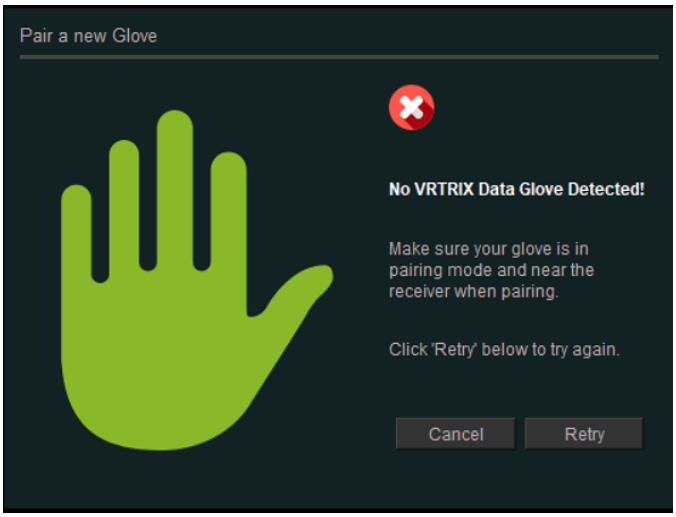
当配对流程开始后，会出现如下对话框：



然后确认设备正常开启并已经初始化完成，然后长按手套壳体正面按钮，随机 led 灯会开始以 1s 五次的频率快闪，继续保持按钮按下状态，直至 led 不再闪烁，保持常亮或者熄灭，此时松开按钮，led 灯会以一快一慢的频率闪烁，此时手套进入配对模式，配对成功对话框显示如下：此后可以选择配对另外一只手套，或者结束配对。



如果超过 30s，软件还未检测到附近有手套进入配对模式，则配对超时失败，可以选择重试或取消配对，此时显示如下界面：

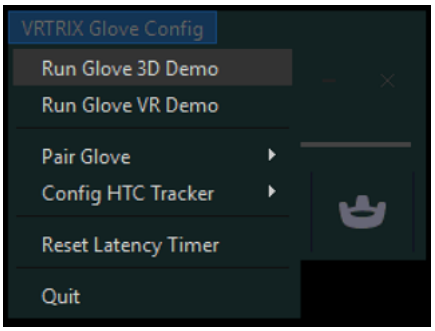


注意 1：有时由于信号干扰等原因，手套进入配对模式后软件无法成功搜索进行配对，可以点击取消配对或重试。

注意 2 ：如果配对成功后手套自动关机，是正常现象，重新正常开机即可。

5. demo 测试：

- 先测试手套 3D Demo，右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，选择 Run Glove 3D Demo 可直接开始运行 demo。



点击 UI 左上角的 connect 即可和手套进行连接，连接后如果手部运动正常，则代表手套硬件正常。

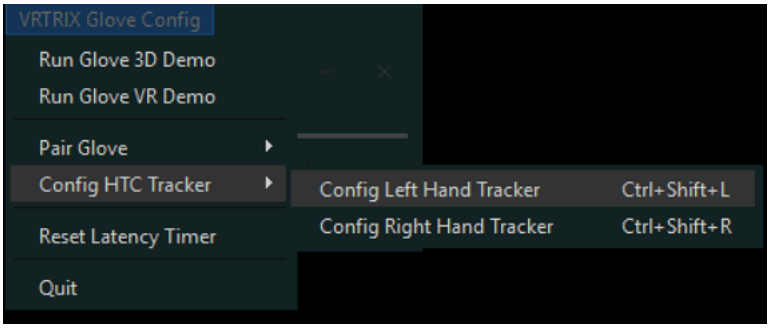
注意：正常连接后，手套壳体上的蓝色 led 灯应当从慢闪变为常亮，如果点击 connect 后发现手部模型不动，且手套上蓝色 led 灯依然在慢闪，则说明手套和接收器没有正常配对，那么需要根据步骤 4 的说明重新配对。

- 后续可以继续测试 VR 场景 Demo, 该场景需和 HTC VIVE 头盔和 Tracker 配合使用，否则无法使用。右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，选择 Run Glove VR Demo 可直接开始运行 demo。

注意 1：该场景使用的 tracker 硬件需改写配置文件才能识别手套，如果 tracker 从本公司直接购买，则无需担心，tracker 已经预先为您配置好，如果自行购买的 tracker 则需要使用软件按下述步骤对 tracker 进行配置

I. 首先开启 tracker 电源，如果 tracker 未与 HTC VIVE 头盔进行过配对，先在 SteamVR 中配对 tracker，配对成功后软件中 tracker 图标将会被点亮，如下图所示：

II. 右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，或者在希望配置的 tracker 图标上点击右键，选择 Config Left Hand Tracker/ Config Right Hand Tracker.



注意 2： 该 SDK 中的 VR 场景 默认使用 HTC VIVE 的硬件平台构建，如果有使用其他定位系统搭配手套的开发需求，请直接联系本公司，针对您的需求进行后期的技术对接。

6. BVH 文件录制：

右键点击软件空白区域，或者左键点击左上角 VRTRIX Glove Config，选择 Start Recording 开始录制，录制过程中状态栏会提示录制进行中，此时无法进行其他操作（例如配对，运行 demo 等），需先结束录制，结束录制后选择 bvh 文件储存文件夹，确认后完成录制。

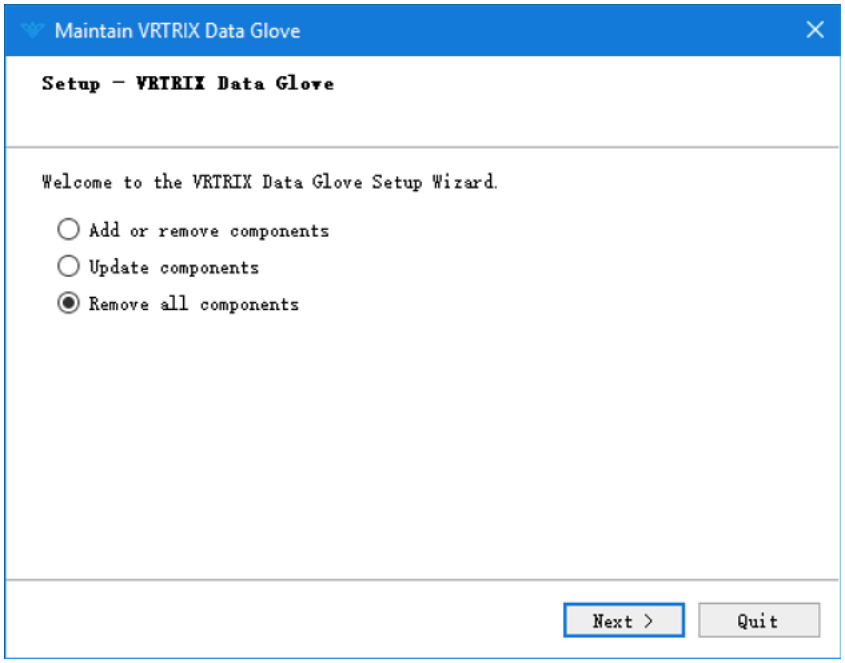
注意 1： 该 bvh 文件欧拉角顺序默认为 Z-Y-X，长度单位为毫米，导入时需选择正确的顺序和尺度。

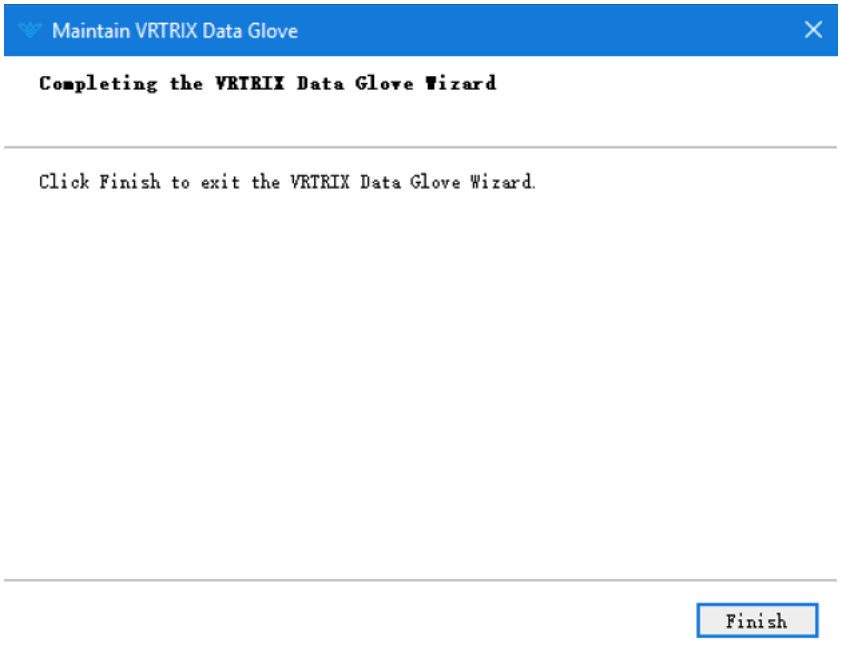
***7.* 设备指示灯：**

所有灯熄灭：设备关闭状态
蓝灯慢闪（1Hz）： 正常待机状态
蓝灯快闪（5Hz）： 关机/进入配对模式的前置状态
蓝灯一快一慢闪烁: 配对模式
蓝灯常亮：正常运行状态
橙灯慢闪：电量低（10%）
橙灯快闪：电量极低（5%）
橙灯常亮：正在充电（只有手套开机状态下才会显示）
绿灯常亮： 电量充满

8. 卸载软件

打开"控制面板”，选择“卸载程序”，选择 VRTRIX Data Glove 并右击打开“卸载/更改”选项，按提示继续操作。


Maintain VRTRIX Data Glove
Setup – VRTRIX Data Glove
Welcome to the VRTRIX Data Glove Setup Wizard.
Add or remove components
Update components
Remove all components
Next >
Quit


Maintain VRTRIX Data Glove
Completing the VRTRIX Data Glove Wizard
Click Finish to exit the VRTRIX Data Glove Wizard.
Finish

全身动捕设备对接
Integrate with Motion Capturing

全身动捕设备往往 sdk 中自带一个全身模型，由于不同动捕设备的人物模型没有统一标准（注意：即便都是 T-Pose 也不是此处所讲的统一标准，此处标准的意思是人物模型骨骼的局部坐标系朝向），所以在接入手套时需要一个额外的模型对齐步骤。该步骤的原理是通过两个四元数变换（左右手各一个）将这个模型手部的局部坐标系和手套硬件传感器的坐标系进行统一，对于一个确定的模型，有且只有一种对应的四元数变换，确定下来以后只要填写到插件中去，之后就不需要修改了。

另外由于全身动捕设备往往包含腕关节的节点，和手套中手背的传感器节点表示同一个骨骼的旋转，所以为了更好的效果，往往将这两种硬件设备在该节点的输出数据进行对齐，原理就是在每一帧获取动捕腕关节的姿态数据，和手套腕关节数据进行四元数差值运算，得到两个姿态的 offset，将这个 offset 动态的作用于手套所有输出关节上，这样就可以对齐动捕设备和手套的腕关节了。

以上两点涉及到对 sdk 原有代码的一些调整，我们给出了一个针对全身动捕设备接入的 c#脚本（DriverMyHand.cs），该脚本是对 sdk 中脚本 VRTRIXGloveSimpleDataRead.cs 的一个调整和补充，下面的文档将会大致说明一下该脚本的几个主要函数，如果有任何其他问题，请咨询本公司技术支持人员。

1. 动态计算手腕关节姿态（由动捕设备得到）和手背姿态（由数据手套得到）之间的四元数差值函数。主要用于获取上述段落中描述的动捕设备手腕关节和手套手背之间的旋转 offset

```
//用于计算右手腕关节姿态（由动捕设备得到）和右手手背姿态（由数据手套得到）之间的四元数差值，该方法为动态调用，即每一帧都会调用该计算。
//适用于：当动捕设备有腕关节/手背节点时
private Quaternion CalculateDynamicRHandOffset()
{
    Quaternion qr = new Quaternion(0f, 0f, 0f, 1f);
    string BoneNameForR_hand = VRTRIXUtilities.GetBoneName((int)VRTRIXBones.R_Hand);
    GameObject R_hand = GameObject.Find(BoneNameForR_hand);
    //MapToVRTRIX_BoneName: 此函数用于将任意的手部骨骼模型中关节名称转化为VRTRIX数据手套可识别的关节名称。
    //GameObject R_hand = MyHandsMapToVrtrixHand.UniqueStance.MapToVRTRIX_BoneName(BoneNameForR_hand);

    //计算场景中角色右手腕在unity世界坐标系下的旋转与手套的右手腕在手套追踪系统中世界坐标系下右手腕的旋转之间的角度差值，意在匹配两个坐标系的方向；
    return  R_hand.transform.rotation * Quaternion.Inverse(RH.GetReceivedRotation(VRTRIXBones.R_Hand) * qr);
}

//用于计算左手腕关节姿态（由动捕设备得到）和左手手背姿态（由数据手套得到）之间的四元数差值，该方法为动态调用，即每一帧都会调用该计算。
//适用于：当动捕设备有腕关节/手背节点时
private Quaternion CalculateDynamicLHandOffset()
{
    Quaternion ql = new Quaternion(0f, 0f, 0f, 1f);
    string BoneNameForL_hand = VRTRIXUtilities.GetBoneName((int)VRTRIXBones.L_Hand);
    GameObject L_hand = GameObject.Find(BoneNameForL_hand);
    //MapToVRTRIX_BoneName: 此函数用于将任意的手部骨骼模型中关节名称转化为VRTRIX数据手套可识别的关节名称。
    //GameObject L_hand = MyHandsMapToVrtrixHand.UniqueStance.MapToVRTRIX_BoneName(BoneNameForL_hand);

    //计算场景中角色左手腕在unity世界坐标系下的旋转与手套的左手腕在手套追踪系统中世界坐标系下左手腕的旋转之间的角度差值，意在匹配两个坐标系的方向；
    return L_hand.transform.rotation * Quaternion.Inverse(LH.GetReceivedRotation(VRTRIXBones.L_Hand) *ql);
}
```

2. 静态计算初始化姿态（由用户指定）和手背姿态（由数据手套得到）之间的四元数差值函数。该函数主要用于当动捕设备没有腕关节/手背节点或者只单独使用手套，无其他定位硬件设备时。由于无法实时获得手腕关节的姿态，只能在初始化（例如 t-pose 校准时），调用一次该函数获得 offset，此后就使用该 offset 进行关节赋值。

```
//用于计算初始化物体的姿态和手背姿态（由数据手套得到）之间的四元数差值，该方法为静态调用，即只在初始化的时候调用一次，之后所有帧均使用同一个四元数。
//适用于：当动捕设备没有腕关节/手背节点或者只单独使用手套，无其他定位硬件设备时。
private static Quaternion CalculateStaticOffset(GameObject objectToAlign, VRTRIXDataWrapper glove, HANDTYPE type)
{
    Quaternion qr = new Quaternion(0f, 0f, 0f, 1f);
    Quaternion ql = new Quaternion(0f, 0f, 0f, 1f);
    if(type == HANDTYPE.RIGHT_HAND){
        return  objectToAlign.transform.rotation * Quaternion.Inverse(glove.GetReceivedRotation(VRTRIXBones.R_Hand) * qr);
    }
    else if(type == HANDTYPE.LEFT_HAND){
        return objectToAlign.transform.rotation * Quaternion.Inverse(glove.GetReceivedRotation(VRTRIXBones.L_Hand) *ql);
    }
    else{
        return new Quaternion(0f, 0f, 0f, 1f);
    }
}
```

3. 手部模型赋值函数，该函数负责计算得出最终赋予手部骨骼模型的旋转四元数，该四元数由数据手套原始数据经两次四元数旋转变换得到，第一次旋转变换用于对齐手部模型坐标系和手套硬件传感器坐标系的不统一，第二次旋转变换用于对齐动捕设备腕部数据和手套手背数据不统一。

```
//手部关节赋值函数，每一帧都会调用，通过从数据手套硬件获取当前姿态，进一步进行处理，然后给模型赋值。
//重要参数：
//      ql:该参数指的是模型左手坐标系和左手手套传感器硬件坐标系之间的四元数偏差，往往这个偏差为绕x/y/z轴旋转+/-90，+-/180度，需要根据具体情况而定。
//      qr:该参数指的是模型右手坐标系和右手手套传感器硬件坐标系之间的四元数偏差，往往这个偏差为绕x/y/z轴旋转+/-90，+-/180度，需要根据具体情况而定。
private void SetRotation(VRTRIXBones bone, Quaternion rotation, bool valid, HANDTYPE type)
{
    string bone_name = VRTRIXUtilities.GetBoneName((int)bone);
    GameObject obj = GameObject.Find(bone_name);
    //GameObject obj = MyHandsMapToVrtrixHand.UniqueStance.MapToVRTRIX_BoneName(bone_name);

    Quaternion ql = new Quaternion(0f, 0f, 0f, 1f);
    Quaternion qr = new Quaternion(0f, 0f, 0f, 1f);
    if (obj != null)
    {
        if (!float.IsNaN(rotation.x) && !float.IsNaN(rotation.y) && !float.IsNaN(rotation.z) && !float.IsNaN(rotation.w))
        {
            if (valid)
            {
                if (type == HANDTYPE.LEFT_HAND)
                {
                    //该语句的含义为由手套得到的raw data先经过ql的旋转，再经过计算得到的当前该帧下手背和动捕设备手腕的旋转差值的作用之后得到的最终旋转，
                    //该最终的旋转就会直接赋值给骨骼模型。
                    obj.transform.rotation = CalculateDynamicLHandOffset() * rotation * ql;
                    //obj.transform.rotation = rotation * ql;
                }
                else if (type == HANDTYPE.RIGHT_HAND)
                {
                    //该语句的含义为由手套得到的raw data先经过ql的旋转，再经过计算得到的当前该帧下手背和动捕设备手腕的旋转差值的作用之后得到的最终旋转，
                    //该最终的旋转就会直接赋值给骨骼模型。
                    obj.transform.rotation = CalculateDynamicRHandOffset() * rotation * qr;
                    //obj.transform.rotation = rotation * qr;
                }
            }
        }
    }
}
```