

linespread = plain)

西南民族大学

Bachelor of Engineering Thesis

智能手表的非接触式备择交互模式的设计

Designing Alternative Contact-free Control Modalities for Smart Watches

Academic Unit	<u>Computer Science and Technology</u>
Name	<u>Changkun Ou</u>
Student Number	<u>201231102123</u>
Class	<u>2012</u>
Major	<u>Computer Science and Technology</u>
Advisor	<u>Yaxi Chen</u>
Co-Advisor	<u>Andreas Butz (University of Munich)</u>

April 14, 2016

Contents

摘要

本文以现有智能手表产品的代表 Apple Watch 为例，对 Apple Watch 上的交互模式的优缺点进行了全面的分析，并据此给出了一套非接触式的备择设计。其配合了 Haptic Engine 对用户的直观震动反馈，完成了对基础点按、滑动、Digital Crown、Force Touch 等系列原生交互的非接触式手势交互设计，此设计将嵌套两步逻辑的十种原生交互简化为了单步逻辑上的八种备择交互，且消除了接触式交互的依赖，解决了现有交互中对双手依赖的缺陷，并同时说明了给出的备择设计的交互完备性。最后，本文对设计的硬件结构、交互方式和系统架构的现有缺陷进行了讨论，并从中得到的启示，给出了可行的解决思路。

关键词 智能手表；非接触式；手势交互；备择设计

Abstract

In this thesis, we study in Apple Watch as the representative of smart watches, and analysed the advantages and disadvantages of interaction pattern in Apple Watch with watchOS 2. According to this, we introducing a contact-free alternative design for smart watch interaction.

This design, by combine the Haptic feedback, convert the basic tap, swipe, Digital Crown, Force Touch and etc. native watch interaction to a contact-free gesture interaction. It simplified ten native interaction with two-step logic to only eight alternative interaction with single-step logic, eliminated contact-required interaction method, solved the bimanual interaction within smart watches, and explained the completeness of this alternative design.

Finally, we discussed the weakness of current hardware structure, interaction design and system architecture. Trough this, we concluded few implications for the future.

Keywords Smart Watches; Contact-free; Gesture Interaction; Alternative Design

1 Introduction

可穿戴设备真正兴起于二十一世纪一零年代初期, 伴随计算机芯片技术三十几年的快速发展, 终于在民用电子消费品市场上出现了以手表、手环为代表的几个主要表现形式。对于手环而言, 仅仅只采集用户佩戴时的使用数据, 本质上而言几乎和用户不存在直接交互。而对于智能手表来说, 尽管搭载了大量简化、合适设计的操作系统, 但依然存在的性能上的不足, 这便给穿戴式设备的软件设计带来了巨大的挑战和机遇 [?].

1.1 HCI Research on Wearable Devices

从上个世纪九十年代中后期起, 可穿戴设备的概念开始逐渐进入人们的视野。

一个可穿戴设备可以从简单的输入设备 (如手环) 到复杂的具备操作系统和通信功能的设备 (如 Apple Watch、Android Wear 设备)。可穿戴设备出现的初衷就是为了向人们提供比智能手机更亲密的、一个计算设备。而穿戴式计算本质上就依托于这样一种小型的、可穿戴的计算机。『实时在线』是穿戴式计算的显著特点, 可穿戴设备穿戴于用户的某个部位, 时刻感知着用户环境以及观察用户的行为。

穿戴式计算的目标是提供一个类似个人助理的服务, 例如便捷的互联网访问、事件通知、信息采集等等, 这种基于穿戴式计算的普适计算技术是集多通道交互 (Multi-Modal Interaction)、上下文感知交互模型 (Context-Interactive Model) 和增强现实 (Augmented Reality) 等于一身的, 人机交互领域所期望的终极任务 [?, ?].

然而受到当下科学水平、市场推广的限制, 这些研究依然还非常基础, 尤其当人们回过头来思考为什么要在自己身上增加一个设备时, 可穿戴设备的存在意义几乎为零, 可替代性非常高。即便如此, 对于可穿戴设备的人机交互相关研究一直在继续, 并且走在电子消费品的前面, 尽管离实现这些设计还很遥远, 甚至可能永远不会被应用, 但这些研究的方法以及它们产生的概念, 都有着其存在的让人们继续思考、反思的价值 [?], 并一步步推动着这个行业的进步。

1.2 选题意义

在智能手机的发展已经略显疲态的今天, 而可穿戴设备却显得力不从心。无论从其产品的可接受度、易用性还是价格, 智能手表存在的意义还不够显著, 智能手表的使用与它本身所处的情景很有关系 [?].

因此, 在对智能手表的交互进行设计前, 要考虑以下几点:

1. 如何增加用户对手表使用的黏性;
2. 如何将交互做得足够易用;
3. 如何平衡一个交互设计的功能可见性和系统复杂性。

本文考虑了目前智能手表市场为代表的产品——Apple Watch, 结合其操作系统自身特点, 设计另一种无接触的、可以释放双手依赖的交互模式。在这种交互模式下, 用户在手表上执行交互的目标得到放大。手表的应用情景能够被得到放大, 交互逻辑的复杂度得到简化, 间接增加了智能手表的存在意义。

1.3 Related Work

智能手表中的界面是与人进行交互中最直观的部分 [?], 关于界面的研究已经较为的成熟和深入 [?, ?], 其中为代表的 Apple Watch 是在所有智能手表中对界面最简洁、优秀的, 其发展出来的『人机交互设计则例』[?] 已经成为交互设计中的标准指南。

然而在执行交互输入的部分却依然混乱不堪, 市面上所有的智能手表上的交互都依赖两只手来完成, 考虑到手表需要抬腕才能观看的自身属性, 如果我们要对现有的设计进行重新设计, 并解除对双手接触式交互的依赖, 对于交互模式的设计便落在了佩戴手腕的那只手上, 这也就无法绕开对手势技术的探讨。

General Interaction of Gesture

手势技术在人机交互的研究中一直经久不衰, 手势可以按空间形式分为平面手势和空间手势。对于平面手势而言, 已经有较为成熟的 $S1$ [?] 和 S_n [?] 算法, 这些方法将连续的手势在等时间间隔内进行重新采样, 然后利用二范数对两个不同的手势序列进行对比; 而对于空间手势而言, 最大的困难就是如何确定手势的开始和结束, 但是实际上由于手势的特殊性我们避开这些问题, 通过识别特定手势、对手势状态机进行建模依然能够进行相关的应用 [?, ?, ?, ?, ?]。幸运的是, 与手表的交互在抬起腕臂后, 由手表输出给用户的全部主要信息都是通过表盘, 这时被确定在佩戴手表的手臂的移动不能幅度过高, 因此这种交互也和传统的空间手势并不完全相同。

文 [?] 详细研究并定义一套复杂地、不同地捏合手势来完成不同的单手、无需用户进行视觉观察的交互任务。虽然该文实现的效果与本文的目的有一部分相同, 但可惜的是, 这些复杂的捏合手势需要将整个手表的交互逻辑重新设计, 这于情于理都不够现实。

非接触式的交互设计同样有相关的研究, 文 [?] 使用 Google 眼镜上的摄像头应用视觉方法来检测用户的手势, 进而完成非触摸式的交互, 然而文中只是对这一技术只适合有佩戴眼镜需求的用户, 不具备普遍适用的价值。

而文 [?] 甚至考虑了利用所佩戴手表的手臂的旋转方式来制造不同的交互。

在这些研究中, 现有的手表交互虽然得到了扩展, 将手势相关的成熟研究应用到智能手表上, 但这些研究并没有认识到手表上的交互方式就存在不够便捷的根本问题。

Extension Interaction of Wearable Devices

一些研究给出了不同的思路, 他们把局限于智能手表上的交互扩展到了手表的外围 [?, ?]、手表的表带 [?]、甚至用户的皮肤上 [?]。这些扩展式的交互虽然打破了屏幕大小的限制, 但是并没有考虑到即便是没有了输入式交互的范围限制, 依然需要双手来执行这些操作, 依然是一种较为疲态、不够便捷的交互形式。

[?] 在手表的周围增加了多个无线传感器, 通过对手表屏幕上方的不同动作进行识别从而增加对手表交互的方式。然而这种方法仍依赖两只手进行完成, 并且完全没有考虑过这种方法在单手交互上的扩展性。

在文 [?] 中设计了一个增强交互的方式, 通过对佩戴手表的手部的手势进行识别来执行不同类型的交互。虽然这种设计考虑了手势在智能手表交互上的应用, 但是这种方式不但没有简化在小屏幕设备上的交互逻辑, 反而进一步增加了用户对执行这种隐式交互的学习成本。

对于扩展式的交互还不仅局限于手表上, 文 [?] 则将手表作为一个手机的位于身体手部位置的扩展辅助传感器, 文 [?] 考虑了智能手表的空间位置, 本质上将智能手表扩展往一种穿戴式的体感交互平台

发展 [?], 这些研究进一步增加了对手表状态的识别进而增强在智能手机上交互的多样性。

综上所述, 尽管对手表现有交互模式的研究覆盖面很广, 但现有的这些关于智能手表的研究中普遍回避了、且没有考虑下述的问题:

- 与现有智能手表的交互模式共存的兼容性较低, 无法通用;
- 没有针对一个系列的产品进行全面可行的应用分析, 给出的设计适用面很窄, 通常一个方法只能完成一件事情, 而其设计本身又不具备功能可见性;
- 给出的设计普遍只适用于双手交互, 对增加用户在产品上的黏性没有明显帮助;
- 在设计上没有考虑过交互模式与用户界面之间的关系, 交互的逻辑性不强, 用户的学习成本高。

2 Interaction Technique on Smart Watches

在 Apple Watch 中，由于触摸屏的存在，大部分手表中的交互方式沿袭自带有触摸屏幕的智能手机[?]。为了对手表中的交互方式进行非接触式备择设计，我们必须分析并且明确在 Apple Watch 中现存的交互方式及其优缺点。

2.1 Traditional Technique

传统交互上苹果公司认为在 Apple Watch 的小屏幕上实施超过两个接触点的点按行为是严重影响用户交互的，因此 Apple Watch 上的触摸屏没有沿用多点触控的方案。并且，在屏幕的普通操作中，只有基础点按和四个方向上的滑动，如图??所示¹。



Figure 2.1: **传统交互**：触摸屏上的单点触摸交互能够制造点按与滑动交互，而滑动交互本身又可以看做一系列连续的点按交互。此外，滑动交互仅涉及上下左右四个方向。

Tap

屏幕上的普通单点触控与智能手机上的方式并没有明显差异。这种点按方式在人机交互形式上早已被大众所接受，此种方式的接受成本也是最低最自然的。

然而将此种交互应用在智能手表上时则极大的降低了其自身的可用性，根据触摸屏上的 FFitts' 定律[?]：

$$T = a + b \log_2 \left(\frac{A}{\sqrt{2\pi e(\sigma^2 - \sigma_a^2)}} + 1 \right) \quad (2.1)$$

其中 σ 是触摸点分布的标准差， σ_a 是输入手指的绝对精度。A 为开始点到目标中心的距离。

根据 FFitts' 定律我们可以看到，一方面，在屏幕大小及其有限的屏幕下将另一只手的手指一动到手表屏幕上会使得 A 的值很大，而且正由于屏幕大小的限制， σ 的值不会较大甚至比手机触摸屏上的标准差还小，而对于同一目标而言 σ_a 的值又不存在变化。因此 T 值会明显变大，即手表屏幕上的点按交互可用性并不高。

Swipe

屏幕上点按位置的连续变化形成了滑动式的交互。Apple Watch 在处理目标点按时的的低可用性问题时应用了滑动手势。例如让可交互的元素尽可能的通过滑动来进行处理。

¹ 图片来源：<https://developer.apple.com/watch/human-interface-guidelines/>

水平方向的滑动可以用于对手表界面的多个水平视图进行切换、返回上级视图等，垂直反向上的滑动则可以在竖直方向上滚动当前视图。注意，垂直方向的滑动所表达的功能并非是 Digital Crown 所具备功能的子集，因为在处于 watchOS 表盘界面时，上下滑动的功能会被表达为向下滑动呼出通知栏和向下滑动呼出 Glance 界面。

2.2 Typical Technique

Apple Watch 在传统触摸屏交互的基础上引入了三个全新的交互硬件，分别是 Digital Crown、Force Touch 和 Haptic Engine，如图??所示²。

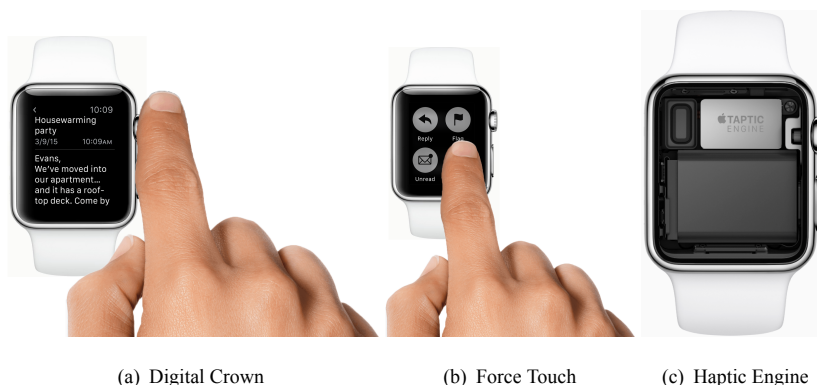


Figure 2.2: 特有交互：Digital Crown、Force Touch 和 Haptic Engine 是 Apple Watch 不同于其他智能手表产品的特殊点。

Digital Crown

Digital Crown 是苹果公司在 Apple Watch 上推出的一个全新的交互技术，苹果公司在推出此项交互的技术时，将其对比了在人机交互历史中的两个革命性的交互技术：鼠标和触摸屏，这意味着苹果公司认为，Digital Crown 是一项在手表上的革命性交互方式。

这种交互方式利用了传统手表的时钟旋钮在功能可见性上的不足。在传统手表中，时钟旋钮在普通状态下不具备任何功能，只有当旋钮被从里向外拉出时，才具备调节时间的功能。这一装置在大部分时间里都不能发挥自身的作用，是一个典型的需求驱动型设计，并没有仔细考虑过其自身的存在方式，只是习惯性沿用。

而在 Apple Watch 上，信息呈现的方式以流式进行纵向展示，所有呈现的内容被限制在一个宽度固定、纵向可伸缩的屏幕区域里。这时，Digital Crown 便能发挥其旋钮的功能。当产生旋转时，内容在竖直方向上进行移动，从而呈现更多的内容，如图??；并且，在交互情景发生变化时，Digital Crown 能够表达出不同的交互指令，例如在影片播放界面时，Digital Crown 的旋转能够调节播放音乐的音量。

Force Touch

Force Touch 这项交互技术是首次在民用消费品中出现。在学术界中，对触摸的感知被研究了多年，[?, ?, ?] 等文献研究了触觉感知如何在触摸屏上进行增强，包括感觉反馈、触摸面积的测量、触摸力度和精度等等，而 Force Touch 就是触摸力度的实际体现。

Force Touch 一共将触摸行为分为了两个等级，第一触摸等级就是传统意义上的触摸行为，手指轻

²图片修改自：<https://developer.apple.com/watch/human-interface-guidelines/>

触屏幕时即可被感知；第二触摸等级就是 Force Touch，这时需要用户将触摸屏幕的力度提升到一个级别后，系统才会进行响应进一步处理交互，如图??。然而，这是一个典型的不具备功能可见性的交互形式，只有当操作被执行后，才能进行后续的交互。

Haptic Engine

Haptic Engine 是一个内置的震动部件，如图??。通过这个部件对手腕传达的震动信息，可以进一步扩大用户与手表进行交互时的反馈。例如，当用户成功实施 Force Touch 后 Haptic Engine 能够给予用户不同层级的反馈，让用户亲自感受到 Force Touch 的实施成功；Haptic Engine 能够在不同用户之间分享彼此的心跳频率时自行调节振动的幅度和速度，达到真实的模拟心跳，进而显著提升 Apple Watch 的用户体验。当我们对现有交互进行备择设计时，无论交互方式如何更改，都应该向用户提供震动反馈来提醒用户操作是否成功。换句话说，Haptic Engine 所提供的震动反馈是进行非接触式备择设计的关键所在。

2.3 Others

除了以上的基础交互手段外，在 Apple Watch 上还有两个最高优先级的交互方式，如图??所示³。。



Figure 2.3: **其他交互**：侧面按钮的存在提高将一项功能提升到了最高的呼出级别，而 Siri 是 Apple Watch 上唯一的文字输入入口。

Apple Watch 侧面的两个按钮实际上是 Apple Watch 里所有交互中优先级最高的交互方式，这是因为无论用户在 watchOS 中所处的位置在哪儿，都能够通过这两个方法执行固定的交互指令（注意，单次点按 Digital Crown 并不能够作为最高优先级的交互，因为单次点按 Digital Crown 后用户所处的位置可能不唯一：可能会处于表盘，也可能处于 App 总界面）。

Side Button

苹果公司可能注意到了在 Apple Watch 上对应用操作的延时复杂性和不便，沿用了手机上电源键的设计，在已经存在 Digital Crown 按键的基础上依然添加了另一个实体按键，如图??。

该颗改建存在两个不同的交互内容，当按下该按键一次时，按键会呼出常用联系人列表，方便通过 Apple Watch 拨打电话及发送讯息等；当连续按下改建两次时，如果当前 Apple Watch 绑定 Apple Pay 银

³ 图片来源：<https://developer.apple.com/watch/human-interface-guidelines/>

行卡时，能够呼出默认银行卡进行 Apple Pay 支付。

Voice Control

尽管对于穿戴式设备的输入方式有很多设计，但是在 watchOS 上是没有哪怕是屏幕上的键盘来制造文本输入。因此，Siri 成了唯一一个能够向 watchOS 输入文本的接口，如图??。

watchOS 上的 Siri 有两种呼出方式，一种是在 Apple Watch 处于唤醒状态下时说出『Hey Siri』；另一种则是长按 Digital Crown。

然而，无论是其中的哪种方式，最后的文字输入效果都不理想。这是因为 Siri 依赖自然语言的识别处理，同时还需要一个配对的手机进行网络访问。这其中只要任何一个环节出问题便不能完成文字的输入。

值得一提的是，在上文提到的所有的交互方式中，只有语音控制是非接触式的交互设计。

3 Alternative Technique Design

上一章中我们分析了 Apple Watch 上的交互方式，除了语音交互外，这些方式都要求用户使用双手来完成整个交互。但实际场景中，如果用户双手空闲，则用户完全可以使用手机完成需要完成的事情，况且在屏幕适中的手机上完成交互也只需一只手。因此，这种交互模式从本质上就是一种不利于增加用户粘性的交互方式。考虑使用手表时的抬碗姿势，最自然的交互自然就是在抬碗时仅用一只手便能完成全部的交互行为。

3.1 Interaction Pattern

Tap and View Switch

点按与视图切换的操作一共涉及五个操作：普通点按、Force Touch 点按、切换到左视图、切换到右视图、切换到上一视图。

单手指的点击由两个手指的捏合操作来完成，拇指和其他四个手指的组合恰好能完成全部的点按交互，拇指和食指点按来替代普通点击事件，拇指和中指、无名指的捏合实现视图的切换，拇指和小指的捏合则实现返回上级视图的功能。

此外，拇指和食指点按还能对 Force Touch 进行仿真，方法详见??一节。

Swipe and Slider Control

滑动与连续调节其实是由 Digital Crown 这一种交互手段在两种不同场景下表达，在一般视图下滑动时能够对视图内容进行纵向调节，而当交互目标选择为滑动器 (Slider Bar) 时，能够对其值进行连续调整。

我们可以在单手上利用双指的相对滑动来实现这一交互。它是指由拇指在食指上的滑动，在视图内通过上下快速滑动来实现交互对象的选取，通过闭合时的慢速滑动来调节纵向的内容展示，如果选取的交互对象是滑杆，那么这种滑动还能够调节滑动器的值。

Force Touch Simulation

一项开源项目 Forcify [?] 是一个针对 Web 端触摸事件的通用框架，将任何 Web 应用里的点击事件作为 3D Touch¹ 进行处理，对不具备 Force Touch 功能设备采用触摸事件延时处理的方法进行模拟。然而，其处理事件的延时时间需要开发者自行定义，且触发 Force Touch 的力度是线性函数。为此，我们应对这个方法进行改进。

首先，对屏幕上的触摸事件划分为两个阶段，第一个阶段的触摸事件处理为普通触摸的触摸事件，第二个阶段的触摸事件处理为 Force Touch，并使用 DELAY 表示触发 Force Touch 的时间延时，DURATION 表示 Force Touch 从最小值到最大值的持续时间。下面考察这两个常量的取值。

在 AugmentedTouch [?] 项目中，我们实施了一个用户调研并发布了一个包含 16 名用户、使用 4 种不同的手姿、共实施 61440 次屏幕点击的数据集。在这个数据集中，每次的屏幕点击均记录了用户的手指在屏幕上的停留时间，图??展示了这 61440 次点击的停留时间的分布。

¹Apple 对 Force Touch 技术在 iOS 设备上重新命名。

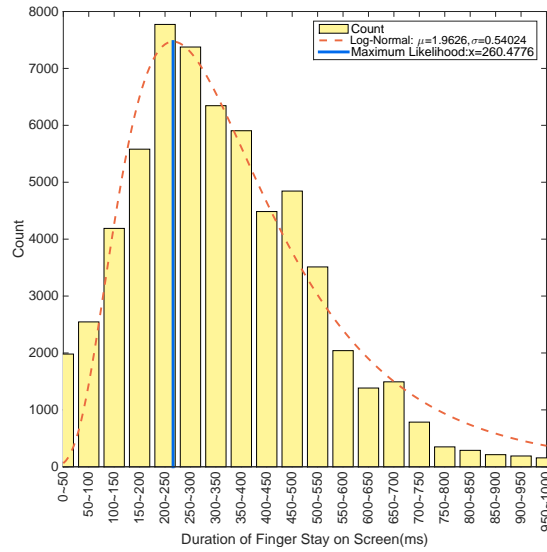


Figure 3.1: 统计结果

从图?? 的分布上我们可以看出，手指在屏幕上的停留时间主要集中在 260 ms 前后，呈对数正态分布。对此，我们不妨将手指在屏幕上的平均停留时间设为 250 ms，故 DELAY 的值为 250ms；另外，设 Force Touch 总持续时间为普通触摸事件的五倍，故 DURATION 的值为 1000ms；这时触发 Force Touch 操作所需要的总时间为 1.25 秒。对 Force Touch 而言，考虑人类手指的按压力度变化，我们设其力度的变化率线性变化，这时能保证整条力度变化曲线为光滑曲线。

综上所述，记在一次按压中的按压时间为 t_{press} ，则 Force Touch 可以使用公式??进行模拟：

$$v_F = \begin{cases} 0 & \text{if } t_{\text{press}} < \text{DELAY} \\ \left(\frac{t_{\text{press}} - \text{DELAY}}{\text{DURATION}} \right)^2 & \text{if } 0 < t_{\text{press}} - \text{DELAY} < \text{DURATION} \\ 1 & \text{Otherwise} \end{cases} \quad (3.1)$$

其中， v_F 表示模拟的压力值，DELAY = 250，DURATION = 1000，两者单位为毫秒 (ms)，图像如图??所示。

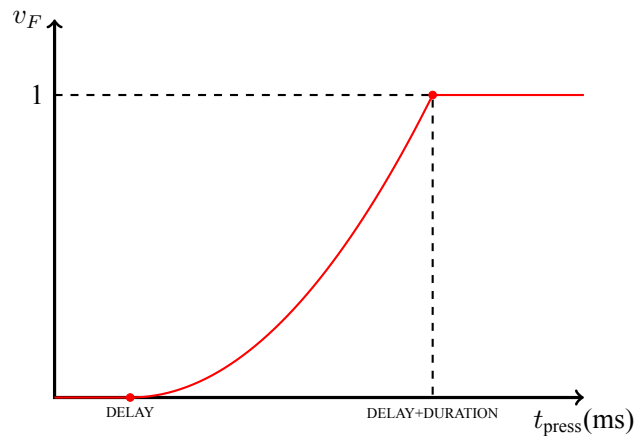


Figure 3.2: Force Touch 仿真函数的图像

Haptic Feedback Design

在 watchOS 中²，根据开发文档[?]显示 Haptic Engine 一共能表达代码??所示的几种类型。

```

1 public enum WKHapticType : Int {
2     case Notification
3     case DirectionUp
4     case DirectionDown
5     case Success
6     case Failure
7     case Retry
8     case Start
9     case Stop
10    case Click
11 }

```

Code 3.1: Haptic 反馈类型

Success、Failure、Retry 三个 Haptic 类型均为对原生交互的操作执行结果的反馈，于是我们将 Success 设置为备择交互中执行完操作且成功后的反馈，Failure 则对应为失败。在原生交互时，Retry 只用于在输入密码时失败后的特殊情况，不妨将此种情况的反馈依然用 Failure 表示，先保留此反馈类型，以便用于其他交互。

Notification 用于在休眠状态下的通知提醒，不对其进行修改；DirectionUp 和 DirectionDown 是原生交互中 Digital Crown 旋转到顶端和底端时的反馈提醒，对此我们同样继承此反馈设计，并扩展到数值调节中，当调整到数值最大值时，执行 DirectionUp 反馈，反之执行 DirectionDown 反馈。

Start、Stop、Click 和 Retry 四个反馈类型恰好可以对拇指与食指、中指、无名指捏合点击与 Force Touch 的四个不同操作进行对应，即 Click 表示普通点击事件，Stop 表示 ForceTouch 操作，Start 表示切换到下一视图，Retry 恰好用于切换到上一视图的提示反馈。

3.2 The Interaction Completeless

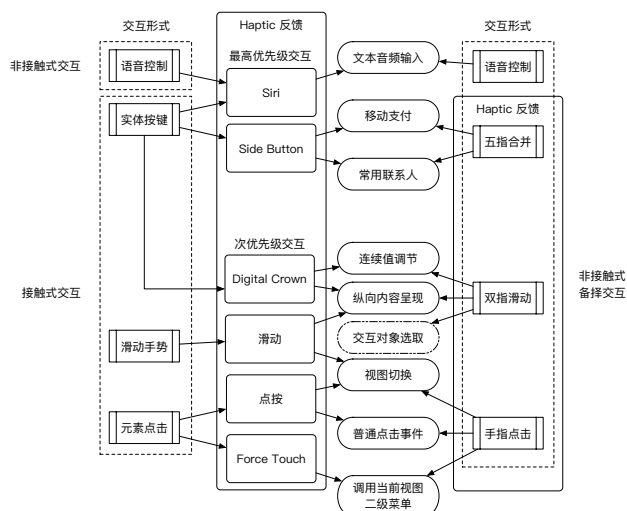


Figure 3.3: 非接触式备择设计总览

²本文写成时的 watchOS 版本为 2.2。

本章中我们全面地对 Apple Watch 上的交互进行了重新设计，简化了元素与元素之间的交互逻辑，作为小结，如图??。在此套备择设计中，我们将原有的两层交互逻辑中十种不同交互逻辑通过简化为单层交互逻辑的八种，并且消除了接触式交互这一限制。

从图??中我们容易看出，备择设计在交互上是完备的，即从另一个角度实现了原交互的全部功能。

4 System Design

4.1 Related Frameworks

LeapMotion and LeapJS

LeapMotion [?] 是一个硬件输入设备及其自身软件的一个总称，其硬件中内置的两颗深度摄像头和一个红外探测器可以对视野内的手检测，其软件部分利用内置的骨骼模型对检测的手部进行重新建模，进而完成手行为的高精度¹识别。

LeapMotion SDK 有两种风格的 API 可以用于获取 LeapMotion 提供的手部数据：本地接口及 WebSocket 接口，WebSocket 接口便其提供了浏览器环境中的 JavaScript 接口。LeapMotion 的 WebSocket 服务遵循 RFC6455²，运行在连接 LeapMotion 硬件的桌面端的 6437 端口。

而 LeapJS 就是 LeapMotion 控制器的客户端 JavaScript 框架。使用 LeapJS 可以让 LeapMotion 与网页前端进行通信，此框架可以用来处理接受 LeapMotion JSON 消息。但随着 NodeJS 的存在，LeapJS 也能够运行在服务端，因此我们也能在服务端使用 JavaScript 处理 LeapMotion 消息。

Leap API 以 Frame 为对象，当可被检测的手出现在 LeapMotion 视野内时，深度摄像头捕捉到的每个 Frame 都能够访问到一个 Hand 对象。Leap 通过对手部的重新建模，将当前帧的手部信息提供给开发者，例如当前手的指向、当前手在 LeapMotion 坐标系统下的位置。借助 LeapMotion 能够省去开发者的手部检测和识别的部分基础工作，更多的关注于手势算法的研究 [?, ?, ?, ?, ?]。

另外，空间手势已经得到较成熟简单应用，例如文 [?] 基于 LeapMotion 实现了 TV 上的控制。

watchOS and WatchConnectivity

watchOS 是运行在 Apple Watch 上的操作系统。watchOS 刚推出时，第三方 App 是作为 iOS 的应用扩展存在，手表端只负责对代码的执行结果进行展示，所有的代码都在 iOS 端执行。随着 watchOS 2 的发布，现在第三方 App 能够通过 WatchConnectivity 框架在 watchOS 和 iOS 之间进行数据通信。

使用 WatchConnectivity 框架执行的通信必须经过 watchOS 和 iOS 系统级的中转。这些通信在 iOS 端和 watchOS 端开启一个会话，因此若需要使用 WatchConnectivity，就必须要实现 WCSSessionDelegate 代理协议。

另外，在 WatchConnectivity 中存在两种不同的通信模式：后台模式和交互式消息模式。在后台模式中，操作系统会将一些非及时消息发送到等待消息队列，并在 App 被唤醒时进行处理；而在交互式消息模式中，由于通信必须经过系统级的中转，因此 watchOS 和 iOS 之间的及时通信，仅当检查到 WCSSession.defaultSession().reachable 为真时，才能够进行通信。值得一提的时，在交互式消息中，即便 iOS 端程序尚未启动，在 Apple Watch 中依然能够从后台将 iOS 应用唤醒。

¹LeapMotion 的官方宣称的识别精度为 0.01mm，文献 [?, ?] 分析表明 LeapMotion 的实际精度在 0.2 mm 左右。

²<http://tools.ietf.org/html/rfc6455>

4.2 Architecture Design

Communication Structure

watchOS 从 2.0 开始从 iOS App Extension 中剥离开来，将 Watch App 部分全部移至 watchOS 端，这时这部分代码在手表端具备了可执行的权限，因此将 watchOS 从 1.0 中的单向接收 iOS 端的系统级的通信，转变为第三方 App 执行管理 [?], 如图 ?? 所示。这使其与外界及时通信成为了可能。

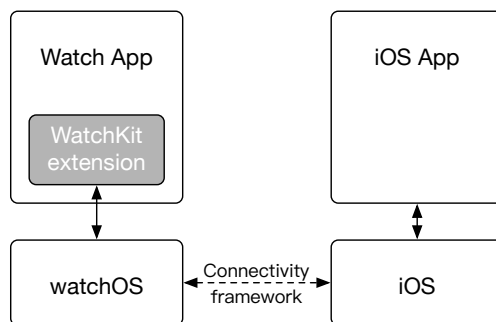


Figure 4.1: Watch App、WatchKit 扩展和 iOS App 之间的联系

然而，即便如此在 watchOS 上的网络访问能力依然十分有限，在 watchOS 2 中，Apple Watch 只能在和与其配对的 iPhone 失去连接，且同时处于已保存的 Wi-Fi 网络覆盖范围内时，才能独立使用 NSURLSession 访问网络，条件十分苛刻。

鉴于以上考虑，本文对从服务端到客户端的通信架构设计如图 ?? 所示。

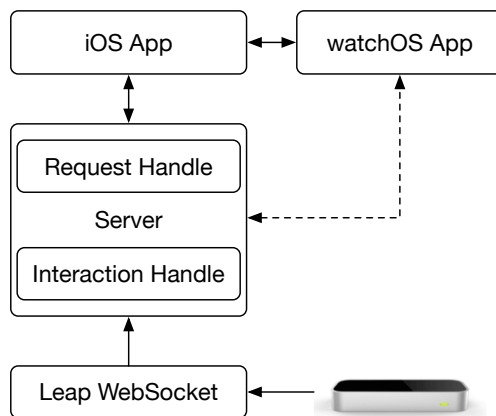


Figure 4.2: 通信架构: watchOS 不直接与服务器进行通信，而是将 iOS 端作为与服务器通信的桥梁

其中，watchOS 将 iOS 端作为与服务器通信的桥梁，处理性能及其有限的 watchOS 端仅负责对通信内容的呈献，性能稍强的 iOS 端对服务端消息进行筛选与加工，而服务端则对 LeapMotion 原始数据进行分析，并封装其分析结果后与 iOS 端进行通信。

Client Structure

客户端包含 iOS 端和 watchOS 端两个部分，其架构设计如图 ?? 所示。

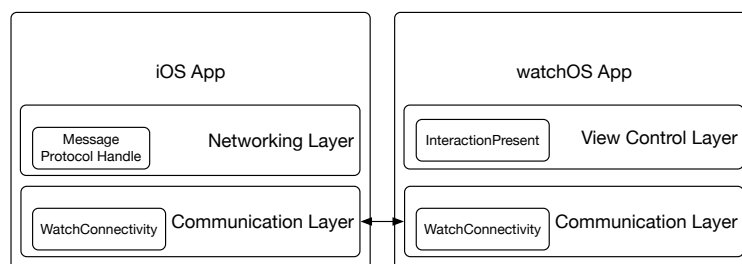


Figure 4.3: **客户端架构**: iOS 端 App 的网络层对服务器消息进行进一步加工处理, 并通过通信层与 watchOS 进行通信, watchOS 端 App 收到消息后通过视图控制层响应 UI 元素的交互。

Server Structure

服务端的设计如图 ?? 所示, 其组件的核心是交互处理层, 该层负责对原始的交互数据进行加工处理为将要实施的交互消息, 然后将加工后的消息按消息协议封装好进而通过请求层分发给相应的请求对象。

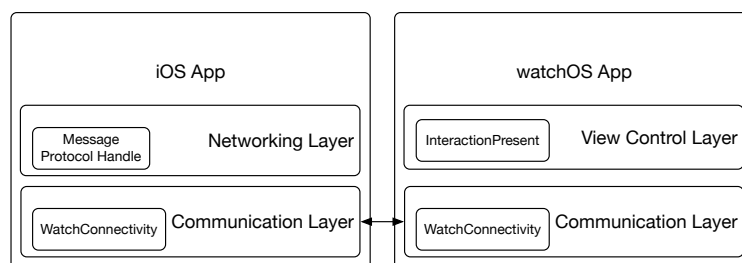


Figure 4.4: **服务端架构**: 交互处理层对原始数据进行加工处理, 完成分析后立即将消息交给请求层并对消息进行分发。

4.3 Communication Protocol

我们需要设计在 watchOS 和 iOS 之间、iOS 与服务端之间设计相关的交互通信协议。

根据图??所示, 用户与手表端进行交互时共涉及五指合并、双指滑动、手指点击三个基本操作, 其中双指滑动仅拇指和食指之间的滑动。而对于交互本身的表达, 在图??中被设计为由 iOS 端进行表达。因此, 对于服务端与 iOS 端之间的通信数据字段设计如下:

返回值字段	字段类型	字段说明
pinchIndex	int	执行捏合操作的手指, 从食指到小指分别为 1 至 4, -1 表示当前没有手指进行捏合
pinchStrength	double	拇指与食指捏合的程度, 为从 0 至 1 连续变化的浮点数
grabStrength	double	五只手指合并的程度, 为从 0 至 1 连续变化的浮点数
forceValue	double	触发 Force Touch 后手指按压力度的模拟值

Table 4.1: 交互字段说明

4.4 Demonstrate

尽管我们对交互方式进行了完备的设计, 但由于 Apple Watch 开发上的限制, 我们无法对系统级的视图进行操作, 因此本文对备择交互一共给出了以下的五个不同效果的演示。

- 第一个演示程序展示了手表点按交互的备择交互设计方案，如图所示；
- 第二个演示程序展示了手指在手表屏幕上滑动的备择交互设计方案，如图所示；
- 第三个演示程序展示了对 Apple Watch 所特有的 Digital Crown 的备择交互设计，如图所示；
- 第四个演示程序展示了对 Apple Watch 所特有的 Force Touch 的备择交互设计方案，如图所示；
- 第五个演示程序展示了一个非接触式交互的游戏案例。

以上五个演示程序的演示视频可以在 YouTube³ 链接中查看，全部相关源代码、环境搭建与演示效果重现的方法可以在 GitHub⁴ 中查看。

³https://www.youtube.com/playlist?list=PLwUqqMt5en7c2QaQ_DkuvZm9dGTz6RjRM

⁴<https://github.com/changkun/BachelorThesis>

5 Implementation Detail

5.1 Development Environment

无论是手表端的还是服务端，都存在框架依赖，因此环境搭建不可避免。在本项目中，服务端使用 NodeJS 进行编码，限于篇幅，对 NodeJS 相关的基本环境，如 Node 本体，NPM 包管理等常见工具的配置在文中略去，这里主要介绍运行本平台最重要第三方 Leap Motion 环境；而在手表端中，虽然我们不依赖其他第三方框架，但由于 watchOS 自身的限制 [?] 在 watchOS 2 中我们需要使用 WatchConnectivity 框架与 iOS 应用本体进行数据通信。由于 iOS 系统本身限制（iOS 9 及以上）强制要求应用必须与 HTTPS 服务器进行通信，因此这里介绍在 iOS 9 中与 HTTP 服务器通信的配置方法。

Configuration of Local LeapMotion

LeapMotion 提供了在 Mac OS X 中的开发环境，并且提供了各种不同的开发语言，根据上文的讨论，我们需要在服务端配置 LeapMotion 本体以及 LeapJS。

1. Installation

首先，在应用程序中引入 Leap SDK 需要在相应桌面端安装 LeapMotion 宿主程序，进而才能使用 LeapMotion 相关的 API。

其次，在 Node App 的 package.json 中添加 LeapJS 依赖：

```
"dependencies": {  
  "leapjs": "^0.6.4"  
}
```

再使用 `npm install` 安装 LeapJS。

2. Configuration

受到 LeapMotion 自身的限制 [?], WebSocket 服务并非默认的向非本地访问开放，因此需要将 Leap 配置启用非本地客户端连接。

这需要对 LeapMotion 的配置文件进行修改。在修改配置之前，需要关闭 LeapMotion 的相关服务。在 Mac 中，使用下面的命令关闭 LeapMotion 的守护进程：

```
sudo launchctl unload /Library/LaunchDaemons/com.leapmotion.leapd.plist
```

接下来我们需要修改 LeapMotion 的配置文件，根据 LeapMotion 的官方文档显示，Leap 包含两个不同的配置，其中控制面板配置的优先级最高，因此我们需要下面这个目录下：

```
$HOME/Library/Application\ Support/Leap\ Motion
```

找到 config.json 的修改配置，编辑 config.json 文件，并在 configuration 字段中的任意位置添加一条：

```
"websockets_allow_remote": true
```

最终得到：

```
"configuration": {  
  "websockets_allow_remote": true,
```

```

"background_app_mode": 2,
"images_mode": 2,
"interaction_box_auto": true,
"power_saving_adapter": true,
"robust_mode_enabled": false,
"tracking_tool_enabled": true
}

```

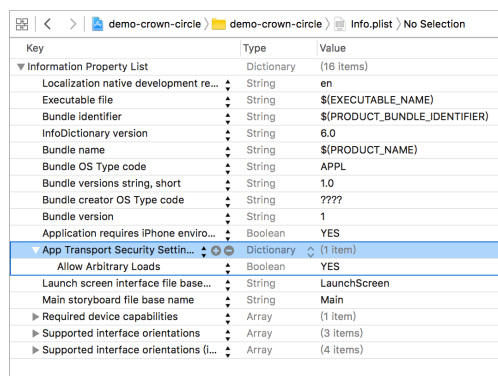
保存退出，重新启动 LeapMotion 服务，便完成了 LeapMotion 的相关配置：

```
sudo launchctl load /Library/LaunchDaemons/com.leapmotion.leapd.plist
```

Configuration of watchOS Networking

从 iOS 9 开始，iOS 对网络访问引入了 App Transport Security (ATS) 特性，这使得在默认状态下 iOS 应用无法发起非安全的网络请求 (如 HTTP)。需要进行下列两个步骤配置项目，结果如图??所示：

1. 在 Info.plist 中添加 NSAppTransportSecurity 类型 Dictionary；
2. 在 NSAppTransportSecurity 下添加 NSAllowsArbitraryLoads 类型 Boolean, 值设为 YES



Key	Type	Value
Information Property List	Dictionary	(16 items)
Localization native development re...	String	en
Executable file	String	\$(EXECUTABLE_NAME)
Bundle Identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
App Transport Security Settin...	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (...)	Array	(4 items)

Figure 5.1: 配置客户端 info.plist 文件

5.2 Server-Side Core

实现一个端口为 10086 的 HTTP 服务器是第一步，如代码??。

```

1 var http = require('http');
2 function handler (req, res) {
3     res.writeHead(200);
4 }
5 http.createServer(handler).listen(10086);
6 console.log("Server running at http://localhost:10086")

```

Code 5.1: 在 10086 端口创建 HTTP 服务器

下面我们来关注服务端中对 LeapMotion 手势的关键处理。

Tap and Force Touch Recognition

LeapMotion 软件本身会对所识别手部进行重新建模，并提供其骨骼模型的全部数据，因此也包括对手指位置。而对于手指捏合的识别，可以转化为对两只手指指尖位置的判断，故可对除拇指外的其他四根手指中进行遍历，如代码?? 所示。

```

1 pinchIndex = function findPinchingFinger(hand, closest) {
2     var pincher;
3     for(var f = 1; f < 5; f++) {
4         current = hand.fingers[f];
5         distance = leap.vec3.distance(
6             hand.thumb.tipPosition,
7             current.tipPosition
8         );
9         if(current != hand.thumb && distance < closest) {
10             closest = distance;
11             pincher = current;
12         }
13     }
14     return pincher;
15 }

```

Code 5.2: 手指点按识别

对于 Force Touch 的模拟在??一节中已经详细描述过方法，其核心实现如代码??所示。

```

1 forceValue = function( ... ) {
2     .... 【待补充】
3 }

```

Code 5.3: Force Touch 仿真

Two Fingers Slipe

对于双指之间滑动的识别看似困难，但实际上我们可以将其转化为两指之间的捏合度。其实 Leap-Motion 本身提供了对拇指和食指捏合度参数，但由于其结果为一般情况下的捏合度，即在任何收拾下都存在捏合度，而对于我们来说需要控制交互，构建双指滑动的触发事件，因此我们有必要重新实现这个功能。

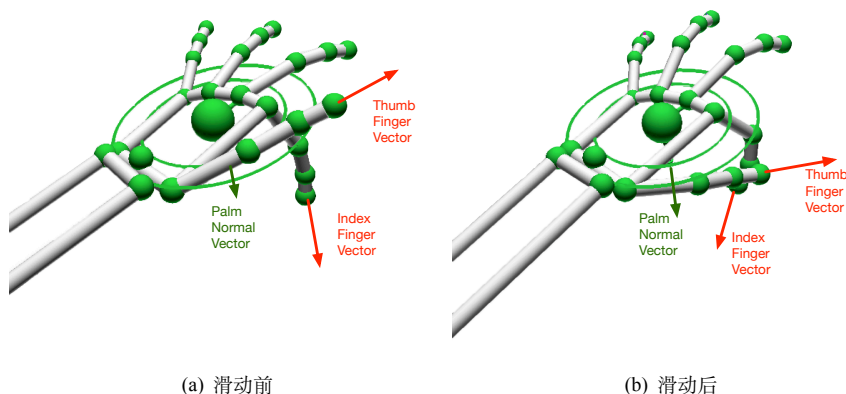


Figure 5.2: **双指滑动**: 双指滑动可以通过手指和手掌的指向向量设计触发事件

双指滑动手势如图??所示, 当手呈现如图??所示手势时, 食指的方向向量和手掌的法向量几乎平行, 拇指的方向向量与手掌的法向量几乎垂直, 故我们可以利用这个特点设计手势的触发事件。如代码??:

```
1 pinchStrength = function findPinchStrength(hand, closest) {
2     ... .. 【待补充】
3 }
```

Code 5.4: 手指点按识别

Full Fingers Grab

五指的合并不需要重新实现, LeapMotion 本身就提供了握拳程度的参数 grabStrength, 我们可以直接从 Frame 对象中获取, 如代码??所示。

```
1 grabStrength = function getGrabStrength(frame) {
2     return frame.hands[0].grabStrength;
3 }
```

Code 5.5: 五指合并识别

5.3 Client-Side Core

手表端编码的两个重要关键就是通信层和视图控制层的编码, 通信层又包含 iOS 端和服务端之间通信以及 iOS 端和 watchOS 端间通信, 我们使用 Swift¹来完成相关编码 [?, ?]。

View Controller Core

【待补充】

Communication Layer Core

与服务端的通信通过 URL 请求服务端的手势数据, 其结果为 JSON 格式数据。在代码??中, completeFlag 保证了向服务器请求对象时仅在上一次请求完成后才能进行, 而 Gesture 类的构造函数通过从

¹本文写成时的 Swift 版本为 2.2。

URL 中解析来的 JSON 数据字典构造 Gesture 对象模型。

```

1 func connectServer() {
2     // member variable
3     if completeFlag == 0 {
4         return
5     }
6     task = session.dataTaskWithURL(url, completionHandler: { (data, res, error) ->
Void in
7         if let e = error {
8             print("dataTaskWithURL fail: \(e.debugDescription)")
9             return
10        }
11        if let d = data {
12            print("\(NSString(data: d, encoding: NSUTF8StringEncoding))")
13
14            if let jsonObj = try? NSJSONSerialization.JSONObjectWithData(d, options:
NSJSONReadingOptions.AllowFragments) as? NSDictionary {
15                self.Gesture = Gesture(fromDictionary: jsonObj!)
16            }
17            self.completeFlag = 1
18        }
19    })
20    task!.resume()
21 }
22

```

Code 5.6: iOS 端与服务端通信

在 iOS 端和 watchOS 端之间通信需要利用 WatchConnectivity 框架，代码包含 iOS 端发送以及 watchOS 端接受，如代码?? 和 ??所示。

```

1 func updateMessage() {
2     if WCSSession.defaultSession().reachable {
3         let content:[String:String] = ["x":PinchFinger.text!, "y":PinchStrength.text!,
"z":GrabStrength.text!]
4         let message = ["up": content]
5         WCSSession.defaultSession().sendMessage(
6             message, replyHandler: { (replyMessage) -> Void in
7                 print("send success..")
8             }) { (error) -> Void in
9                 print(error)
10            }
11    }
12 }

```

Code 5.7: iOS 端与 watchOS 通信: 发送


```
1 extension InterfaceController: WCSSessionDelegate {  
2     func session(session: WCSSession, didReceiveMessage message: [String : AnyObject])  
3     {  
4         guard message["up"] as? [String:String] != nil else {return}  
5         let contents = message["up"] as! [String : String]  
6         self.interaction(contents)  
7     }  
}
```

Code 5.8: watchOS 端与 iOS 通信: 接收

6 Feild Study

6.1 User Study

Pre-Study

Questionnaire

Progress

Participants

由于时间所限，本文实施的用户调研一共从高校学生中招募了十位参与者，其中有七名男性，年龄从 20 至 23 岁不等。

Results

Analysis

6.2 Usability Test

Planning

Results

7 Future Work

7.1 Weakness

本文在此前的篇幅已经叙述过与 watchOS、LeapMotion 这些相关硬件开发上不可逾越的困难，在性能和功耗都严重受到限制 watchOS 上，交互消息的通信和处理都非常关键，一个好的设计能够让各部分硬件顺利工作。然而软件是依赖硬件设计而成，因此接下来我们考虑在硬件实现本身和择备交互方案本身存在的问题。

Hardware

作为一个原型项目，本文实现上使用 LeapMotion 完成对手部的建模和识别，但 LeapMotion 自身却又限制在必须与一个桌面端系统进行连接，这就极大的限制了交互的范围和场景，不能在其他场景下使用。

在文 [?] 中，Gilles 等人将一颗深度摄像头绑定在鞋子顶部，并在用户和腰部设置一个处理硬件用于识别交互，从而在鞋子顶部的上方制造了交互区域。类似的，我们可以设计图??所示的一个可移动的 LeapMotion 识别方案。

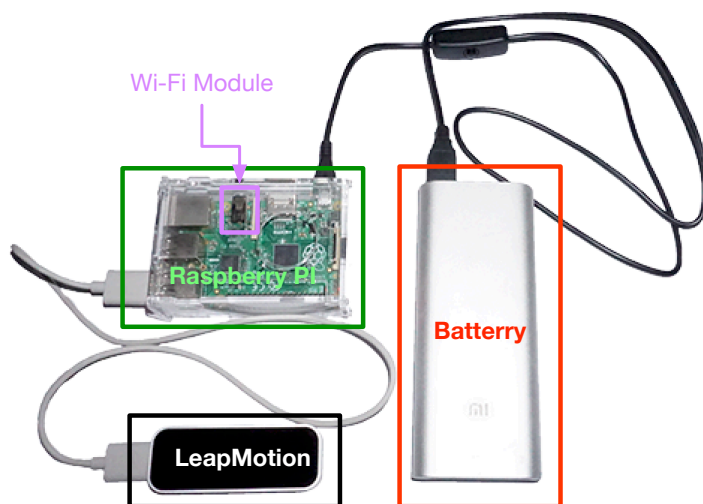


Figure 7.1: **硬件结构**: 图中由三部分硬件分别为一个 5V-2A 输出功率的移动电源、一个组装了 WiFi 模块的树莓派 B+、一个 LeapMotion 硬件

在这个方案中，移动电源提供 5V-2A 的输出功率，驱动树莓派；树莓派通过 WiFi 模组进而连接服务器进行交互信息通信，而 LeapMotion 则通过 USB 连接到树莓派上，只要树莓派上能够使用 LeapMotion 的软件部分，则此方案能够解决交互范围受限的问题。

遗憾的是，LeapMotion 软件要求宿主 OS 至少具备 2GB RAM，且 SDK 也仅能在 Ubuntu Linux 上得到支持，当前的树莓派第二代 B+ 仅有 512MB RAM，暂时得到应用，但随着 Windows 10 能够在树莓派第三代上运行，我们有望看到未来当树莓派扩展到 2GB RAM 时能够搭载 Windows 平台将图 ?? 的方案得到应用。

Interaction

本文给出的备择设计虽然从交互形式的表面上完整对接了原有的设计，但是一旦用户任务复杂时，此备择设计的效率则会劣于接触式设计。受到 Keystroke-Level Model (KL 模型) [?] 的启发，我们可以给出手表任务上的 KL 模型，并将一个用户任务分解为几个基础步骤，见表??：

变量	描述
K	普通点按操作
F	Force Touch 操作
S	滑动操作
T	交互目标选定
R	系统响应
M	用户思考
H	最高优先级操作

Table 7.1: 用户任务基础步骤

对普通点按而言，我们不妨假设双指捏合与屏幕点按的用时一致，注意到在备择设计中，目标的点按还需要通过双指滑动操作在屏幕上进行选取，故 $K_{\text{contact-free}} > K_{\text{contact}}$ ；对 Force Touch 而言，接触式交互能够直接测量用户的按压力度，故即便对于 Force Touch 依然能够在普通点按的时间内完成，我们也不妨假设这两个时间相等，在??一节中，我们给出的对 Force Touch 仿真的方法是设定触摸事件的两个阶段，在第一个阶段内将点按处理为普通点击事件，显然我们有 $F_{\text{contact-free}} > F_{\text{contact}}$ 。

对于 T 、 M 和 H 而言，如果用户已经熟悉这两种设计，我们不妨认为在两种设计上的用时相等。而对于 T 参数，背着设计需要一个一个将视图中的对象进行遍历，显然 $T_{\text{contact-free}} > T_{\text{contact}}$ 。另外，备择设计需要在通信上有所开销，因此 $R_{\text{contact-free}} > R_{\text{contact}}$ 。

最后，接触式交互和非接触式交互都只能是以上七个基础步骤的线性组合，而对于同一个任务来说，其系数相同，根据上面的分析，显然有式??成立：

$$\Sigma_{K,F,S,T,R,M,H} \text{Contact Task} < \Sigma_{K,F,S,T,R,M,H} \text{Contact-free Task} \quad (7.1)$$

并且当任务变得复杂时，式??左式的值会明显小于右式。

7.2 Improvement

最后，我们对本文所设计系统在未来应用时植入方式的改进，以及系统整体的扩展性进行初步探讨。

Recognize Methodology

本文在手部信息的检测上使用了 LeapMotion 硬件作为解决问题的关键，但实际上 LeapMotion 是一项作为桌面端交互或虚拟现实交互的输入设备而产生的，依赖视觉方法的 LeapMotion 并不完全适合作为用户的一个随身穿戴部件植入，因为视觉方法的依赖性很强。考虑手表本身的设计，我们可以考虑更换一种对手势进行识别的方式。

Myo [?] 是一个固定在手臂上的对手部肌肉电信号进行感知从而进行手势识别的装置，如图??所

示¹。Myo 通过内置的 Wi-Fi 装置将识别的电信号传递给其他的连接部件，Myo 开放了并规范化了对手部肌肉电信号时间序列的识别协议，这意味着我们可以对这种表现模式的手势数据进行分析，进而逐步完善手势系统。



Figure 7.2: **Myo 识别装置**: Myo 安装在手臂上对手部肌肉的电信号进行检测，从而根据这些电信号在时间序列上的变化特征进行手势识别。

Myo 的设计实际上完全符合手表的人体工程学设计，因为手表的表带就是 Myo 最好的存在形式。当 Myo 能够缩小并设计成手表的表带时，这时手表系统本身在识别上不再依赖其他额外的设备，大幅降低设备间通信的消耗，便更加高效的完成交互。

Interactio Perception Center

在 ?? 一节中，我们讨论了通信架构的设计，实际上这一设计也适用于一般情况。

图 ?? 为对前文中适用于手表备择设计的通信架构的推广设计，并在服务端和客户端引入了交互感知中心 (Interaction Perception Center, 在 IPC 中我们给出了一个基于桌面端和移动端设备的跨平台交互解决方案) 服务 [?]. 当我们将 IPC 这系统级的交互模型引入时，实现 IPC 核心服务的传感器处理协议 (Sensor Handle Protocol, SH 协议) 能够屏蔽掉硬件之间的差异，统一的将传感器形式化、标准化，而通过跨设备交互协议 (Cross-Device Interaction Protocol, CDI 协议) 能实现用户设备与 IPC 服务之间的通信。

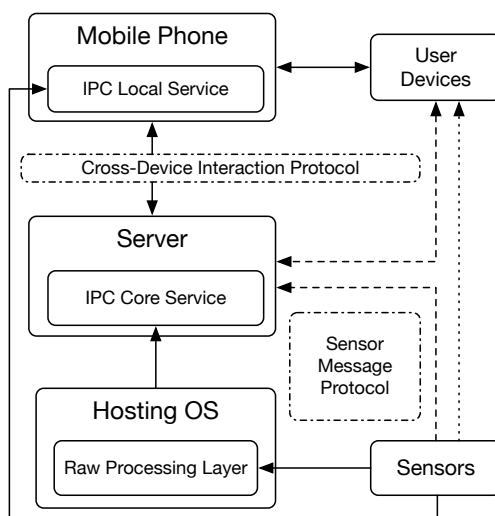


Figure 7.3: **通用通信架构**: 在服务端和客户端引入了 IPC 服务。

¹ 图片来源 : <https://www.myo.com>

不同的传感器先在与其连接的桌面端系统上按传感器各自的 SH 协议对原始数据进行初步加工，再由 OS 将这些消息集中发送给服务端交给 IPC 核心服务进行处理进行二次加工，完成后再由服务端请求层按 CDI 协议进行封装并分发给相应的设备；另一方面，当 IPC 核心服务不可用时，传感器的宿主 OS 依然能与客户端的本地 IPC 服务进行通信，使得在服务器离线状态下依然能够响应一些简单的交互。

IPC 设计之初是一个纯软件的解决方案，目的是解决桌面端设备和移动端设备的跨平台分布式数据通信与分析，而从前文中的基础通信架构设计中我们能够得到启示：首先，统一交互系统的设计不宜将传感器原始数据直接发送给 IPC 服务器进行分析，传感器的数据应当在传感器的宿主 OS 上完成对数据的基础加工，加工后的数据和 IPC 服务器之间增加一层消息协议来保证任何传感器数据都能接入 IPC 的接口；其次，IPC 传递给用户设备的交互消息需是用户可选的，这些交互在任何情况下，都至少有一种主交互方案。即被动交互应当是隐式、备择的。

7.3 Conclusions

本文对智能手表上现有的交互形式进行了备择设计，配合了 Haptic Engine 对用户的直观震动反馈，完成了对基础点按、滑动、Digital Crown、Force Touch 等系列原生手势的备择设计，将十种原生交互简化为了八种备择交互，且消除了接触式交互的依赖，并说明了此套备择设计的交互完备性。与主流交互不同，备择交互更像是编程语言上的一个语法糖，其侧重于临时、快捷、简便的实现一个用户期望的输入。在本次的设计中，我们克服了从硬件平台到软件框架接口不支持缺陷，在克服这些缺陷中发展出来的架构设计给与了我们在整合一套交互系统时的很多有益启示。

总的来说，人机交互的创新往往伴随着硬件和软件的结合，在软件层的方法有时往往会受到硬件平台的各种限制，一套打通硬件与软件的架构设计不仅涉及从硬件到软件的垂直整合能力，还需要进行横向设计与可用性的考量，这些内容浮与水面之上，也仅仅只是冰山的一角。

References

Acknowledgement

This thesis is under the guidance of my advisor, Prof. Yaxi. In my bachelor

本论文是在我的导师——陈雅茜老师的细心指导下完成。陈雅茜老师在我本科阶段四年的学习过程中给予了我极大的指导与帮助，从最初的美数学建模竞赛论文指导开始，再到大学生创新创业项目的申请，科研论文的撰写与发表，慕尼黑大学交换学期的学习，以及现在的本科毕业设计。陈老师为人亲和、严谨求实、认真负责，对我各方面严格要求，使得我的知识、能力和视野，甚至是日后的道路都有着不可小觑的影响，这些帮助是我目前人生阶段中其他任何一位给予我帮助的老师都无法企及的。首先，我在此对陈雅茜老师表示最崇高的敬意和衷心的感谢。

其次，我感谢我所有的亲人，特别是我的母亲与父亲。是他们的对我的关怀、支持与包容，才让我走到了今天，他们是我永不倒下的后盾；我感谢我的朋友，他们与我一同怀揣梦想，共同为了自己的理想奋斗不止，互相激励；我感谢其他关心、帮助、指导过我的老师，正是有了他们才使得我的本科阶段的学习过得充实无比。

随着本文的结束，我的本科学业也将完成，我会带着师长、亲人的鼓励与期望，迈向人生的下一个阶段，希望自己能保持永远年轻，永远热泪盈眶。

愿明天会更好。