

分 类 号 \_\_\_\_\_

学号 M201672904

学校代码 10487

密级 \_\_\_\_\_

# 华中科技大学

# 硕士学位论文

## 基于 Kinect 的手势识别系统的设计和 实现

学 位 申 请 人： 蔡碧海

学 科 专 业： 计算机技术

指 导 教 师： 张杰 讲师

答 辩 日 期： 2018 年 5 月 21 日

**A Thesis Submitted in Partial Fulfillment of the Requirements  
for the Degree for the Master of Engineering**

**A Gesture Recognition System on Kinect: Design  
and Implementation**

**Candidate : Bihai Cai**

**Major : Computer teconology**

**Supervisor : Lecturer Jie Zhang**

**Huazhong University of Science & Technology**

**Wuhan 430074, P.R.China**

**May, 2018**

## 独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

## 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于 ☐ 保密， 在\_\_\_\_\_年解密后适用本授权书。  
☐ 不保密。

（请在以上方框内打“√”）

学位论文作者签名：

日期： 年 月 日

指导教师签名：

日期： 年 月 日

## 摘 要

随着带有深度摄像头的 Kinect 体感设备的发布，基于 Kinect 的手势识别人机交互系统的研究层出不穷，这也使得这种技术变得更加成熟。但是由于 Kinect 本身也存在着一些比较明显的局限，研究工作者在使用 Kinect 获取到图像数据的同时，也需要采用一些算法对其做一些处理，然后才能稳定地获取实际中需要用到的数据。与此同时，自然手势有着更加丰富的内涵，因此目前对于自然手势的识别问题的研究一直是计算机视觉领域的热门方向。

本文旨在设计和实现一种利用 Kinect 获取手势信息的手势识别交互系统，该系统使用一种建立手势有限状态机（FSM）的方法，并基于动态时间规整（DTW）算法实现动态手势识别。本文中在建立手势 FSM 的基础上实现了四种基本的手势：手掌点击、手掌挥动、双掌缩放和双掌旋转，同时利用 FSM 中的状态切换实现手势连续轨迹点的捕获，并将捕获到的手势轨迹使用 DTW 算法与手势模板库中的 12 种手势轨迹进行匹配，从而实现动态手势识别。另外，该系统在微软提供 Kinect SDK 的基础上拓展一套实用的手势识别接口，并将这套接口应用到实际的人机交互中，代替现在 PC 机器上的鼠标或者键盘的一些基本操作。

本文的手势识别系统主要分为图像处理模块和手势识别模块，在图像处理模块中，由于 Kinect 中能够大略获取手掌心点，而这个结果由于波动性比较大，因此在这个基础上需要结合骨骼数据和深度图像数据经过一系列的处理获取到比较稳定的手掌心点，然后将之应用到手势识别模块，实现基于 FSM 的四种基本手势的识别和基于 DTW 算法的动态手势轨迹识别。实验结果表明，基于 FSM 的四种基本手势能比较准确的识别结果，同时，基于 DTW 算法的手势轨迹识别有着 90% 以上的平均正确识别率。

**关键词：**计算机视觉，图像处理，人机交互，手势识别，有限状态机，动态时间规整

---

## Abstract

With the release of the Kinect somatosensory device with a depth camera, research based on the Kinect-based gesture recognition human-computer interaction system has emerged in an endless stream, which also makes this technology more mature. However, due to the obvious limitations of Kinect itself, researchers use Kinect to obtain image data, which must be processing by some algorithms in need, and then they can stably obtain the data needed in practice. At the same time, natural gestures have richer connotations. Therefore, the research on the recognition of natural gestures has always been a hot topic in the field of computer vision.

This paper aims to design and implement a gesture recognition interaction system using Kinect to obtain gesture information. The system uses a method of establishing a gesture finite state machine (FSM) and achieves dynamic gesture recognition based on a dynamic time warping (DTW) algorithm. In this paper, four basic gestures are implemented based on the establishment of gesture FSM: Palm Click, Palm Swing, Palm Zoom and Palm Rotation, and in use of state switching in FSM, continuous gesture trajectories are captured, which will be obtained and used to match the 12 hand gesture trajectories in the gesture template library to achieve dynamic gesture recognition via the DTW algorithm. In addition, the system expands a set of practical gesture recognition interfaces based on Microsoft's Kinect SDK, and applies this interface to the actual human-computer interaction, replacing some basic operations of the mouse or keyboard on the PC.

The gesture recognition system in this paper is mainly divided into 2 modules: image processing module and gesture recognition module. In the image processing module, because the Kinect can get a palm point slightly, this result is more fluctuating, we need to combine bone data and the depth image data on this basis, and obtain a relatively stable palm point through a series of processes, and then apply them to the gesture recognition module to realize recognition of four basic gestures based on FSM and dynamic gesture

trace recognition based on the DTW algorithm. Experimental results show that the four kinds of basic gestures based on FSM can accurately identify the results. At the same time, gesture track recognition based on DTW algorithm has an average recognition rate of more than 90%.

**Key words:** Computer vision, Image processing, Human-computer interaction, Gesture recognition, Finite state machine, Dynamic time warping

## 目 录

摘 要.....	I
Abstract.....	II
1 绪论	
1.1 研究背景.....	(1)
1.2 国内外研究现状 .....	(2)
1.3 Kinect 开发技术.....	(6)
1.4 论文的内容和意义 .....	(9)
2 关键技术和系统框架设计	
2.1 手势识别技术 .....	(12)
2.2 系统总体框架设计 .....	(16)
2.3 本章小结.....	(18)
3 手势图像数据的获取和处理	
3.1 基于 Kinect 的需求分析.....	(19)
3.2 图像数据的获取 .....	(20)
3.3 图像数据的处理 .....	(23)
3.4 本章小结.....	(28)
4 手势识别模块的实现	
4.1 手势轨迹的获取 .....	(29)
4.2 建立手势 FSM .....	(31)
4.3 DTW 算法 .....	(35)
4.4 基于 DTW 的手势识别 .....	(41)
4.5 本章小结.....	(50)
5 系统测试和应用	
5.1 图像处理模块测试 .....	(51)

5.2	手势识别模块测试 .....	(52)
5.3	手势识别应用场景 .....	(58)
5.4	本章小结.....	(63)
6	总结与展望	
6.1	总结.....	(64)
6.2	改进和展望.....	(64)
	致谢.....	(66)
	参考文献.....	(67)



## 1 绪论

### 1.1 研究背景

随着计算机科学技术的革新,人和计算机的交互变得越来越多样化,然而新的人机交互方式也朝着更加自然和更加便捷的方向发展着。人工智能技术的发展使得基于图像和语音的人机交互方式更加稳定和准确地能表达人与机器之间的“交流”。目前在人机交互领域中,基于视觉的智能化、鲁棒性的手势识别方法是学者研究的热门方向<sup>[1]</sup>。

随着计算机互联网技术在我国迅速普及,人们发现传统的鼠标键盘等机械输入方式,在三维空间自由度方面越来越体现其所固有的局限性,这种交互方式可能并不是对所有人都能很快上手,有一个适应学习的过程,因此更加随心所欲的人机交互方式呼之欲出。随着一些科学技术的发展,越来越多的研究更多的在关注人脸识别、语音识别、人体识别等方面对于人机交互方面的应用。

一方面,新型的人机交互模式体现了人们对于人机交互中人扮演角色的思考,人往往是人机交互的主体,新的人机交互必然会朝着人性化便捷化自由化的目的发展,与相较于计算机的鼠标键盘交互模式、游戏机手柄的交互模式、智能家电遥控器的交互模式、平板电脑和智能手机基于触摸屏的交互模式想比较而言,基于空间的手势识别的交互模式将显示出其巨大的发展潜能,这种技术体现了对于人而言相对自由便捷的交互方式。

另一方面,在准确识别人的手势的基础上,也必定会影响机器人研究过程中对于人机交互的思考,准确的手势识别使得机器人能够真正看懂操纵者的“语言”。在花大精力研究遥控系统的同时,简单的手势会显得更加地便捷,并且能用简单的手势可以解决的问题,往往使得遥控器的交互过程会显得比较复杂。

其次,深度摄像头设备的出现,使得手势识别这一种技术的实现变得更加方便。相较于传统二维图像的基于图形学的轮廓检测和跟踪,三维摄像头由于提供了  $z$  轴

方向上的深度数据，因此类似于手掌按压这种二维图像这种几乎很难实现检测的动作，在三维摄像头中得到了很好地检测和跟踪。作为一个革命性的产品，Kinect 体感设备能够稳固地捕捉、跟踪身体的动作、声音和手势，并且十分方便地获取到彩色图像数据、深度图像数据以及骨骼图像数据，它提供了一种新的人机交互方式。因而基于 Kinect 手势识别的相关研究也层出不穷，这一点也使得新型的基于深度的人机交互方式有了很好地实现背景和应用前景。

在此基础之上，本课题旨在研究一种利用 Kinect 获取手势信息，建立手势状态机（FSM），并基于动态时间规整算法（DTW）的手势识别的交互系统，并将这种交互系统运用于实践，代替鼠标和键盘的一些基本的操作。

## 1.2 国内外研究现状

手势所能表现的含义是十分丰富的，并且手势表达有着非常简单自然的特点，因此手势作为人机交互的一种输入方法是比较理想的<sup>[1]</sup>。一般而言，手势识别有两种类别：静态手势识别和动态手势识别，这两种手势识别的方向都有着比较悠久的历史和相对成熟的研究理论，从不同特征而言，静态手势识别技术着重点在于研究手部的形状变化，对于二维图像中的手势识别技术而言，理论是趋于成熟的，与此同时，动态手势识别技术会较为复杂，它更着重于研究手掌在三维空间中位置变化，针对这些某段连续的位置变化信息产生手势语义，对其进行实时分类。

手势识别所具有其他一些交互方式不存在的优势：

其一、手势是比较容易别用户理解的，比如用手比一个“OK”的动作就可以表示确认；

其二、使用手势与计算机进行交互可以更加自由地表达用户的想法；

其三、准确的手势识别可以提高用户和机器之间交互的效率。

因此国内外的研究者在手势识别方面进行了大量的工作，也有着丰厚的研究成果。对于手势识别的方法，大致可以分为三种：基于数据手套的手势识别，基于双目摄像头的手势识别和基于 Kinect 的手势识别。

### 1.2.1 基于数据手套的手势识别

基于数据手套的手势识别时目前运用比较广泛的研究方法，同时也是比较经典一种研究方法。数据手套能直接收到传感器反馈给系统的用户手指尖在三维空间中的坐标位置和手指运动的相关信息，因此用于手势识别时，数据精度是比较高的，而且可识别的手势种类也比较多，但也存在一些缺陷：例如由于这种设备本身昂贵的造价，使得这方面的应用产品并不常见，同时这种设备本身对于用户来说体验度不是很好，长时间的佩戴可能会造成手掌心出汗等。

上世纪 80 年代，Grimes 发明了“数据手套”<sup>[3]</sup>，其前身是手套式传感器系统，早期的产品是 Sayre Clove<sup>[4]</sup>。此后，基于这种设备的研究层出不穷，Kim 等人利用一种叫做 KHU-1 的数据手套开发了一个三维手掌移动跟踪和手势识别系统，该设备包含一个三轴加速传感器、一个控制器和一套蓝牙机制，能够通过无线蓝牙给 PC 设备发送手掌移动信号，并利用获取到的三维数据手掌模型进行手势识别<sup>[5]</sup>；J.Weissman 和 R.Salomon 利用数据手套对手指关节的角度提取了 18 个测量值，实现了一套面向虚拟现实应用程序的手势识别系统<sup>[6]</sup>；D.L.Quam 做了一项使用可以准确获取到手指位置的数据手套进行手势识别实验，对 22 类手势进行了稳定的识别<sup>[7]</sup>；韩国成均馆大学的 In-Kwon Park 等人利用数据手套基于 FPGA（现场可编程逻辑门阵列）电路，实现了一套手势识别系统，其平均识别正确率达到了 94%以上<sup>[8]</sup>，解决了无线手势识别系统中用户和设备在空间和移动上的限制以及光照等环境因素造成的识别不稳定的问题。

### 1.2.2 基于双目视觉的手势识别

在早期的研究中，基于接触性设备的手势识别方法有着中重要的影响力，这种方法虽然对于识别的准确率和稳定性上有着足够的保证，但是也存在着一些不足的地方，比如手势识别不够便利，不够自由等。近 30 年来，人们越来越关注更加自然便捷地手势识别，这也是目前国内外手势识别的趋势。

双目视觉就是有两个摄像头，利用立体视觉成像的原理可以提取手掌在三维空间中的位置，然后对手部建立立体模型，根据获取到的手掌数据实现手势识别。

国外在双目视觉方面做了比较多的研究，其中，A.A.Argyros 等人<sup>[9]</sup>实现了一种利用两种彩色摄像头提供的视频流数据确定三维手势位置的手势识别方法，使用一种视角校准算法计算两种信息之间形成的三维数据；日本的 A.Utsumi 等人<sup>[10]</sup>使用多摄像头系统检测手部的骨架图片实现了基于 COG（手掌中心点）检测算法的一种可行的手势识别系统，解决了普通摄像头下获取到的手掌和手臂区域连接的情况会造成手势识别的结果非常不稳定的情况；在 90 年代初，Krueger 基于一个双摄像机的 Video Desk 系统，实现了对双手捏去和托转等手势的匹配<sup>[11]</sup>；J.Segen 实现了一个可以用双摄像头捕获手势的系统，该系统对点、点击和伸张这三种手势进行了准确的识别，并且利用这三种手势实现了对控制虚拟物体的功能<sup>[11]</sup>；多伦多大学的 Mailik 等人实现了一个叫做 Visual Touchpad 的系统，该系统在双摄像头和一块黑白的矩形纸板基础上，实现了对手指进行检测和跟踪，能比较精确地获取手指的位置和移动方向，对 PC 机实现了人机交互的功能<sup>[13]</sup>。

同时，国内在基于双目视觉方面研究也有一些研究成果：浙江大学的郭康德<sup>[14]</sup>通过计算机双目视觉技术实现了一个三维指尖检测系统，利用算法基础以及其他关键技术实现了一种框架，并利用该框架实现了三维鼠标和若干基于指尖点击、指尖移动和手掌翻转等多种手势的三维手势交互游戏；郑州大学的谭同德等人<sup>[15]</sup>根据双目视觉定位数学模型计算目标位置信息的算法，提出了一种一人手的质心为特征点提取的方法，实现了一个对虚拟物体进行抓取、移动和释放操作的系统。

### 1.2.3 基于 Kinect 的手势识别

针对双目摄像头进行的手势识别的研究，要获取比较精准地手势识别所需要的手势位置以及轨迹信息，可能需要时效性更高的双目摄像头设备，这是因为双目摄像头获取到图像必须是同步的，否则无法准确计算出场景中运动部分的深度信息<sup>[16]</sup>，同时，由于双目摄像头图像获取需要特定的光照环境，过明或者过暗的环境会影响手势识别的精度，这是目前利用普通双目摄像头实现手势识别时存在的不足之处，而深度摄像头技术在这个方面也显示出极大的优势。

目前相对成熟的带有深度摄像头的产品之一是 Kinect, 近年来, 国内外学者在 Kinect 手势识别方面做了很多相关的研究, 其中, 新加坡的 Zhou Ren 等人基于 Kinect 传感器实现了一种鲁棒的手势识别系统, 该系统提出了一种通过计算手指 EMD (FEMD, Finger-Earth Mover's Distance) 距离的计量方法, 来测量手势形状之间的不同点, 同时消除了从 Kinect 摄像头获取到手势形状带来的噪声, 使得手势识别的准确率达到 93.2%<sup>[17]</sup>; 印度的 K.K.Biswas 等人使用了支持向量机 (SVM, Support Vector Machine) 的方法, 构建了一个基于 Kinect 设备的手势识别系统, 并实现了对 8 种典型的手势的准确识别<sup>[18]</sup>; 美国的 Arun Kulshreshth 等人实现了一种基于 Kinect 设备的手指检测和追踪技术, 并将实验结果和传统的基于 K-曲率的手指尖识别和检测的技术进行了对比, 初步的结果证明了 Kinect 设备获取到的稳定的手指尖技术能达到传统的 K-曲率方法获取手指尖技术的效果<sup>[19]</sup>; F.Pedersoli 等人搭建了一个叫做 XKin 的开源动态手势识别的框架, 该框架从 Kinect 设备中获取动态手势轨迹, 并基于隐马可夫 (HMM) 分类模型的方法, 对 16 中美国手语进行了自然且直观的认识<sup>[19]</sup>; 基于 Kinect 的 3d 视觉和语音识别功能, S.Fakhteh 等人实现了一个游戏软件, 主要用于识别聋哑人的手势, 解析手势命令, 帮助他们得到实时的游戏体验<sup>[21]</sup>。

此外, 国内有很多将 Kinect 作为一种输入设备, 获取人体骨骼的坐标信息, 并进行手势识别, 实现了一些人机交互的应用。上海交通大学的钱鹤庆在利用 Kinect 传感器获取手部轮廓的基础上, 使用人工智能、计算机视觉和多媒体技术, 开发了一种结合手势识别和增强现实技术的教育辅助系统, 该系统选取 DTW 算法进行动态手势识别, 并提出一种动静态结合的手势识别方法<sup>[21]</sup>; 北京工业大学的李小龙使用结合 Unity3D 的三维场景渲染系统和脚本控制系统, 使用 Kinect 获取控制人体解剖学中的身体模型的输入数据, 定义了一些生活中常用的手势基本含义, 并实现接口化, 主要识别了手掌张开、握拳、两手收缩和扩张、左右手滑动等手势, 用于控制相应模型的放开、抓取、缩放和左右移动等操作<sup>[23]</sup>。通过以上方法实现了具有较高的实用性、交互性与趣味性的虚拟解剖教学系统。香港大学的 Chong Wang 等人提出了一种基于超像素的 EMD (SP-EMD, superpixel earth mover's distance) 的测量方法, 对



从 Kinect 传感器获取出来的深度信息进行处理，实现快速和高校的手势识别系统，并将这种系统应用于实践<sup>[24]</sup>。

由上可知，手势识别的一直是计算机视觉和人工智能领域中比较热门的研究方法，而由于近年来深度摄像技术的快速发展，一些基于深度图像的手势识别方法被提出或者被改进，稳定的鲁棒的手势识别方法一直是计算机领域研究的趋势，一些成熟的理论和技术已经应用于实践。

### 1.3 Kinect 开发技术

Kinect2 设备能够在获取深度图像数据的同时，捕获和跟踪到实时的全身骨骼数据<sup>[25]</sup>。Kinect2 具备红外感应器，因此即使在黑暗环境下也能稳定地获取深度图像数据，这也是其优势所在：它能够基本不受外界光照环境影响，稳定实时的获取深度数据。相较于比普通二维摄像头，它能够获取物体在三维空间中的具体位置。

#### 1.3.1 Kinect 简介

Kinect 设备是微软公司的产品，目前一共有两代。该设备包含了一个 RGB 彩色摄像头、一对深度摄像头一组麦克风阵列，具备即时的动态捕捉、影像识别、语音识别等功能<sup>[26]</sup>。该设备能对玩家全身上下的动作进行捕捉，使得游戏者可以脱离手柄等游戏控制器的束缚，更加直观地对游戏进行控制。

2014 年，微软发布了 Kinect 第二代的产品 Kinect for windows v2，相较于一代有着明显的提升。Kinect 的外观如图 1.1 所示。



(a) Kinect 1



(b) Kinect 2

图 1.1 Kinect 一代和二代外观图

Kinect 对于传统的人机交互方式有着关键性的改变:传统的图形用户界面(GUI)比较明显的特点就是用户必须适应软件中定义好的一系列操作,并在硬件设备上上进行实时交互,而 Kinect 提供了一种自然用户界面(NUI)的人机交互方式,使得用户能以最自然的方式和机器互动。

### 1.3.2 Kinect 原理

从系统架构的角度来看, Kinect 包括四层,从下到上依次是:硬件层、驱动层、接口层和应用层<sup>[27]</sup>。如图 1.2 所示,

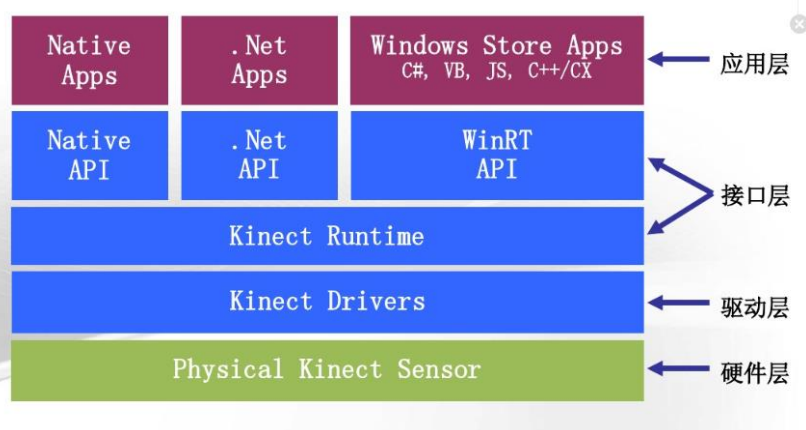


图 1.2 Kinect 系统架构示意图

1. 硬件层的深度图像数据原理:在硬件层,第二代的 Kinect 的红外摄像头和红外投影机可以用于获取可视范围内人体物体的深度图像,在一代的 Kinect 中,深度图像的获取是采用了以色列 PrimeSensor 公司的 Light Coding 技术<sup>[28]</sup>,即将红外光投影到一维或者二维的图像中,根据图像的变形确定被测物体的表面状况,当物体离 Kinect 设备距离不同时,图像的形变参数就会不同,根据存储在内部的模板进行匹配,就能获取每个像素的深度值,与一代的 Kinect 所不同的是,第二代的 Kinect 采用的深度测距原理是 TOF (Time of Flight, 光线飞行时间) 技术,它发射红外光主动探测目标物,并在反射时接收目标物的反射光,根据测量时间差算出目标物的距离<sup>[29]</sup>; Light Coding 技术和 TOF 技术获取深度数据如图 1.3 所示。

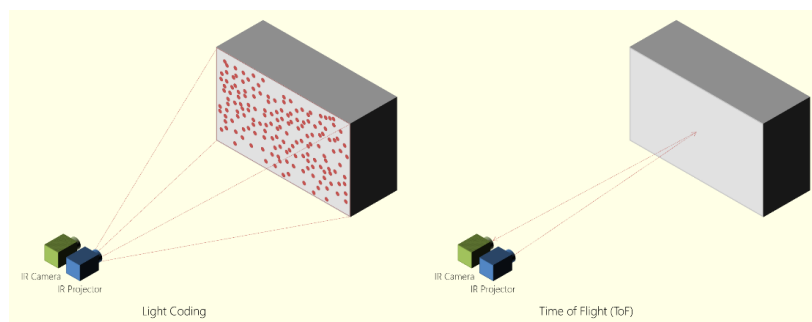


图 1.3 LigthCoding 和 TOF 获取深度数据的示意图

2. 接口层的六种数据：在 Kinect 2 的接口层，微软提供了六种数据源，以供开发者使用：

- (1) 彩色帧数据（ColorFrameSource）：提供 30fps 的 1920\*1080 分辨率的彩色图像，支持 RGBA、YUV 等格式；
- (2) 红外帧数据（InfraredFrameSource）：提供 30fps 的 512\*524 的红外帧数据，且每个像素用一个 2 字节的内存空间来存储；
- (3) 深度帧数据（DepthFrameSource）：提供测量范围为 0.5~4.5 米的物体深度图帧数据，每一帧中的深度表示红外摄像头到该物体的距离，单位是毫米；
- (4) 人体索引帧数据（BodyIndexFrameSource）：提供追踪到的最多 6 个人体的索引值，用一个字节来表示，其中 0~5 表示被追踪的人体索引编号，-1(0xFF) 表示未发现人体<sup>[29]</sup>；
- (5) 人体骨骼帧数据（BodyFrameSource）：提供最多 6 个人体的 25 个关节点的集合，每个帧中包含了关节点的三维坐标点和方向信息，每秒 30fps 的刷新率；
- (6) 音频帧数据（AudioFrameSource）：提供 Kinect 的四个麦克风阵列的声音接收覆盖角度为 100 度的声音信息。

在接口层提供者六种数据源的目的就是为了能够很方便地在应用层获取最后需要的数据。



### 1.3.3 Kinect 局限

虽然目前 Kinect 的技术比较成熟,但由于目前深度摄像头依旧处于发展的阶段,Kinect 也存在着一些局限,主要体现在以下四点:

1. Kinect 的获取深度数据范围比较有限:对于 Kinect 2,只有人体在 0.5~4.5 米的范围之内,才能比较准确的获取到深度图像数据<sup>[3]</sup>。
2. Kinect 的骨骼数据必须在全身出现在摄像头之内的情况下才能准确无误的获取到,否则会出现一些偏差。
3. 虽然 Kinect2 有自带的手势识别技术,但是对于动态的手势识别并没有比较完善的实现接口。
4. 经过测试,Kinect 中获取到的掌心点的坐标数值会发生晃动,其稳定性不高,因此在程序中需要通过自己获取手掌心的方法,获取到稳定的掌心点数据。

## 1.4 论文的内容和意义

### 1.4.1 课题的内容和意义

前文对手势识别领域的三个研究方面做了目前国内外研究现状描述,相较于基于数据手套这种接触性设备的手势识别和基于双目视觉这种基于视觉性设备的手势识别,Kinect 为研究者提供更加便利的图像数据,尤其是它具备可以同时获取深度图数据和人体骨骼数据,使得它是用作手势识别系统研究的一种比较好的选择,因此本文是基于 Kinect 获取的图像数据,实现的一种手势识别系统。本课题主要研究的内容概括为:

- (1) 在微软提供的 Kinect for Windos SDK v2 的基础上,基于手势的数值特征,构建一个基于 FSM 的动态手势识别接口系统,对单手挥动,手掌点击、双手张开和双手旋转四种手势进行识别。
- (2) 研究动态手势识别的相关算法,在第(1)步建立手势 FSM 之后,利用手势状态转换作为手势轨迹的起点和终点,捕获连续的手势轨迹,实现基于模板匹配的 DTW 动态手势识别算法,对具有特定含义的手势进行稳定的识别。

- (3) 基于第(1)步中实现的手势识别接口系统,实现一个简易的图片浏览器,并利用识别到的手势实现左右翻页、图片选择、图片缩放和旋转的功能,验证手势识别接口系统的实际意义。

本课题研究的主要意义在于:

- (1) 可以应用于虚拟教学领域:手势识别系统的特点之一就是它能够简单快捷自然地表达用户的想法,并将它传递给机器,这使得在虚拟教学领域能更加方便地对机器进行控制,比如在教学中可以直接用双手来演示教学 PPT;
- (2) 可以帮助语言残疾者:研究动态手势识别可以将手语动作翻译自然语言,实现这种手势识别的框架对于帮助语言功能残疾者是十分有意义的;
- (3) 可以虚拟现实辅助功能:目前虚拟现实技术还在发展中,而再一些娱乐领域正起着十分重要的作用,一些技术已经商业化,比如索尼的 PSVR 就是实现目前最新的 VR 技术。那么手势识别作为一种更加自然的人机交互方式必定会在未来的娱乐领域大放异彩。

#### 1.4.2 论文组织结构

本论文由六个章节组成,其组织结构如下所述:

第一章介绍的是本文的研究背景和意义,介绍有关手势识别的相关含义,之后从基于数据手套、基于双目视觉、基于 Kinect 这三个方面上逐一介绍了目前国内外手势识别在人机交互和计算机视觉中的相关研究和应用现状,然后简要介绍了 Kinect 设备、其中获取深度图像的原理以及 Kinect 设备自身的一些局限,最后说明论文的主要内容和意义、以及论文的组织结构。

第二章介绍的是手势识别的关键技术和系统的框架设计,首先介绍了手势特征提取和动态手势识别的关键技术,然后阐述了本文的手势识别系统的整体框架设计,为之后的章节做铺垫。

第三章首先提出的基于 Kinect 手势识别的需求分析,然后依次谈到图像数据获取的流程和步骤,详细说明每个模块的意义、具体要解决的问题以及具体的执行流程。

第四章首先介绍了手势轨迹获取的相关流程设计和实现步骤，之后说明基于数值特征的手势识别的具体实现。从总体程序设计的角度说明实现的具体内容，然后重点阐述了基于 DTW 算法的手势识别模块的具体实现思路，然后对于手势库建立给出具体实现过程。

第五章是对本文所设计和实现的手势识别系统进行测试，首先分别对图像获取和处理模块单独测试，之后再对手势识别模块进行测试，并对每一个测试的结果进行结果展示和分析，并对一个基于该手势识别系统的图片浏览器进行了单独测试，对最终的测试结果进行评估。

第六章对本文进行了总结和展望，首先总结了本文的工作成果，最后客观地分析了本文所实现的手势识别系统的不足，同时给出了未来可以改进的方向。

## 2 关键技术和系统框架设计

### 2.1 手势识别技术

手势识别的过程的第一步就是手势检测，在这一步中，主要完成的工作就是对计算机视觉中手掌位置的描述和定义，同时在时间序列中对手掌的位置进行实时的跟踪；那么手势识别过程的第二步就是对获取到的手掌信息进行分类处理，给获取到手掌位置特征点定义特定的手势含义，根据既定的手势含义对手掌位置信息进行实时分类。

#### 2.1.1 手部特征提取的方法

与数据手套实现手势识别的方法相比，在基于自然视觉实现手势识别的方法中，由于获取手部区域的视频图像往往不是单独存在的，无法十分方便获取到手部区域的一些新，所以在这种方法中对于手部区域的分割是比较重要的一点，也是影响到手势识别率的关键性因素。提取手部特征值进行手势识别的一般流程如图 2.1 所示。

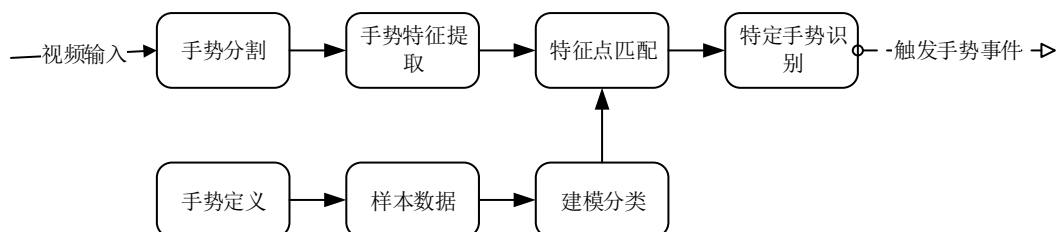


图 2.1 提取手部特征点手势识别流程

在这个过程中第一步就是手势分割，对于基于计算机视觉的手势分割可以有两种可行的方法：

1. 肤色模型<sup>[31]</sup>：在现实生活中，人种之间的差异导致不同的人种的肤色不尽相同，但是尽管存在着差别，在对人类肤色排除了环境造成的一些影响后，色调是基本一致的<sup>[32]</sup>。这种方法的基本原理是在 YCbCr（或者 HSV）颜色空间中构造一个肤色检测模型，然后对足够多的肤色样本进行训练，使用机器学习的方法对待识别的有肤色的图像数据进行分类识别<sup>[33]</sup>。肤色检测模型中有贝叶斯模型和椭圆模型等<sup>[34]</sup>，通

过前人的研究可知，皮肤像素点在 CbCr 构成的坐标系中可以近似看做一个椭圆分布，因此肤色模型可以归结为判定坐标点(Cb, Cr)是否在该椭圆和其边界上<sup>[35]</sup>。

基于肤色模型的手势提取主要局限在于：

- (1) 肤色模型对于外界环境要求比较高，光照等条件对于肤色检测影响比较大，强光或者黑暗条件会是肤色检测的准确率急剧下降；
- (2) 对于人体皮肤其他的部分往往不能绝对区分开来，对于需要准确识别到指尖点或者提取轮廓点这些步骤影响会比较大。

2. 深度阈值<sup>[35]</sup>：对于配备有深度摄像头的设备能够获取具有人和物远近距离的深度图像，同时在能够准确追踪到骨骼点坐标信息的情况下，可以通过深度阈值的方法对人体手部进行分割。在获取到的手掌骨骼点附近，规定一个深度阈值  $d$ ，手掌平面的宽  $W$  和高  $H$ ，然后在深度图空间中规划一个长为  $L$ ，宽为  $W$ ，高为  $H$  的立方体空间区域，其中长度  $L=2\times d$ ，这样可以将立体空间中的手部区域提取出来。这个过程如图 2.2 所示。

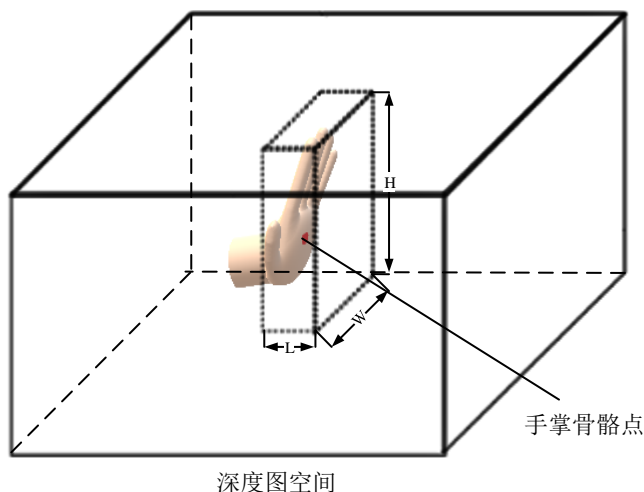


图 2.2 深度阈值提取手部示意图

这种方法主要利用到深度图数据中的深度信息能表示物体在三维空间中距离摄像头的远近，那么它的特点很明显：

- (1) 摄像头必须能够十分准确的获取并追踪人体骨骼数据

(2) 摄像头必须能够十分稳定地获取到深度图数据,以便在人体手掌骨骼点附近对深度数据进行划分。

Kinect 就是兼具了以上两种特性,所以在本文中是用 Kinect 对手掌区域进行分割提取,后续的具体步骤见第三章。

### 2.1.2 动态手势识别的方法

动态手势识别这个概念是相对静态手势识别而言的,它指的是对于一系列具有时间顺序的手势序列进行识别,准确表示用户要表达的意思。动态的手势包括手的位置运动轨迹,手的旋转等其他变化。其一般的具体的识别流程如图 2.3 所示。

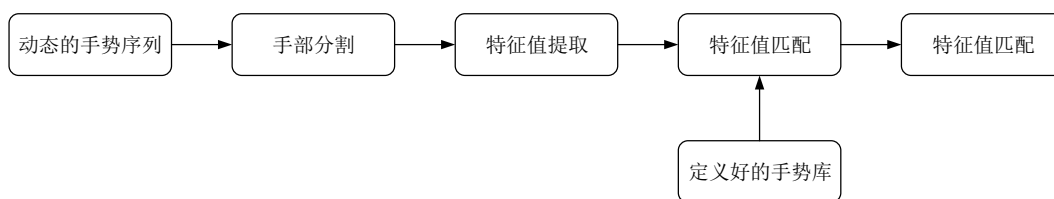


图 2.3 动态手势识别一般流程图

首先,获取动态手势的序列,然后对于手势序列进行手部分割,将手部区域提取出来,根据提取出来的手部区域计算特征值,一般是手势轨迹的坐标点,也可能是手对于某个方向上的夹角余弦值或者手旋转的某个角度值,对于这一序列的数据进行组合,形成一个特征值序列。最后在已经定义好的手势库中去匹配手势模板,并对匹配结果进行评估,得到最终的识别结果。动态手势识别的方法主要有以下几种方法<sup>[3]</sup>。

1.基于模板匹配的方法:这种主要是将每个动作的特征序列先用模板的方式保存起来,然后对于实际中获取到的手势序列进行与模板的匹配,计算与每个模板中手势序列之间的相似程度,选取相似程度最高的那个模板手势序列所表示的手势结果作为动态手势识别的结果<sup>[37]</sup>。这种方法可以分为两类:直接匹配法和基于动态 DTW(动态时间规整)算法的匹配算法。DTW 算法经过实践的检测,该方法能够达到比较高的识别率,但是由于计算量比较大,所以识别所耗的时间比较长。

2.基于统计的方法:这种算法是目前大多数手势识别技术使用的,其识别精度比较高。其识别原理是将每一种手势定义一种手势归类,然后根据归类对训练样本的特

征向量进行统计和分类,在实际中提取手势的特征向量,并根据训练产生的分类结果得出最后手势识别的结果。目前这种方法中使用较多的是 HMM (隐马可夫) 模型,训练时对每一种手势建立 HMM 模型,识别的时候从大量训练的样本中选取概率最大的那个作为最后的识别结果。例如美国虚拟计算实验室的 J.Schlenzing 等人<sup>[38]</sup>在实验室中使用了基于 HMM 模型的手势识别,对于每个手势,他们分别定义了一个 HMM,该研究表明 HMM 对于动态手势识别有着比较好的性能和比较高的识别率,但同时由于在这种方法中,对于训练的依赖性比较大,因此如果要加入新的手势,都要设计与之对应的 HMM 模型,并需要从大量的训练数据中得出参数,其可扩展性比较差。

3.基于数据分类的方法: 这种方法是基于某种学习准则,并循环地进行学习,具备自组织和自学习的能力,能有效的避免一些识别错误的情况。其中,较为典型的方法是基于 BP 神经网络 (Error Back Propagation Neural Network) 技术的方法,该技术是一种能向着满足给定的输入输出关系方向进行自组织的神经网络<sup>[39]</sup>,当输出层上的实际输出与给定的输入不一致时,通过下降方法校正层间的结合强度,直到最终满足给定的输入输出关系。这种方法最大的优势就是容错性能强,不易受噪声干扰,缺点是计算量比较大,识别过程比较耗时。

4.基于语法的方法: 语法是编译原理中提出的概念,它是一种高层含义,建立在运动特征的基础之上,从抽象的角度对运动的属性进行描述<sup>[39]</sup>。基于语法的手势识别方法之一就是基于有限状态机 (FSM) 的方法,例如林永强等人提出了一种基于姿势序列有限状态机的动作识别框架,对人体肢体点序列进行采样分析,采用正则表达式来表示肢体动作的轨迹,并够造一种有限状态机,实现对 17 种预定动作的识别,识别准确率在 94%以上<sup>[41]</sup>。基于 FSM 的手势识别方法也存在一些局限:

- (1) 状态定义的方法决定了这种手势识别方法没有用到样本来进行训练,因此这种手势识别方法在准确率上可信度不高;
- (2) 基于有限状态机的方法是一种相对于状态转移的方法,它对于状态要求比较高,即在特定的状态才会对手势进行识别,如果状态不正确就会导致错误的识别结果,这导致了这种方法的鲁棒性不高。

在本文中，主要用有限状态机建立手势识别模型，实现手状态之间的变化，实现基本手势的识别，同时在动态时间规整算法的基础上实现了手势轨迹的模板匹配。

## 2.2 系统总体框架设计

本文是使用 Kinect 设备获取手势图像数据，调用了 Kinect for windows SDK 2.0 提供的接口，对 Kinect 捕获到的深度数据和骨骼数据进行了处理，分割出手部，并获取了手掌中心点，根据中心点的坐标位置转换成用于动态识别的手势轨迹。接下来从开发环境和系统模块设计两个方面对系统框架实现进行说明。

### 2.2.1 开发环境

本手势识别系统的开发环境分为硬件环境和软件环境，硬件环境包括：

1. 64 位的 CPU、物理双核、配备有 USB3.0 接口的 PC 机：由于第二代 Kinect 摄像头获取图像要进行大量的计算，所以对于 CPU 的要求比较高；与第一代 Kinect 相比，对于获取到的数据需要通过 USB 接口快速传输到 PC，需要用到 USB3.0 接口；
2. Kinect v2：第二代的 Kinect 深度摄像头设备，获取手势图像数据。

软件环境包括：

1. Windows 10：第二代 Kinect 是必须要在 Window8 及以上的环境中才能运行，故本实验使用的操作系统是 Windows 10；
2. openCV 2.4.9：本实验中获取深度数据和骨骼点数据是用 OpenCV 来显示的，调用 OpenCV 中的图像显示模块；
3. Kinect for Windows SDK 2.0：微软提供给开发者的 Kinect 软件开发包，提供了获取 Kinect 的深度数据和骨骼数据的 C++语言和 C#语言的 API 接口；
4. Visual Studio 2013：windows 平台的应用程序开发环境，本系统使用 C++开发。

### 2.2.2 系统模块设计

手势识别系统主要可以划分成两个模块：图像处理模块和手势识别模块，如图 2.4 所示，其中图像处理模块主要是完成对从 Kinect 获取到的深度图像数据和骨骼数



据的处理，具体是利用深度阈值法对手掌进行分割，获取到手部的特征数据，根据特征数据获取到手掌心点的三维坐标信息；手势识别模块主要是将图像模块获取到的

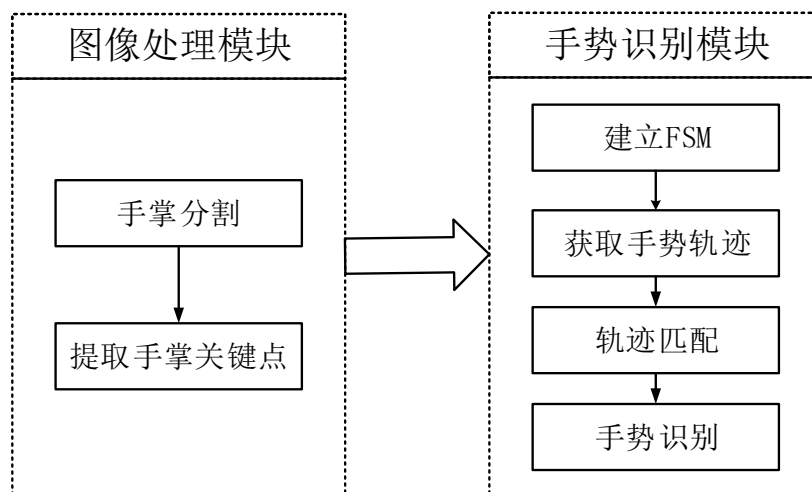


图 2.4 系统框架图

坐标信息提取出数值特征，根据数值特征划分手势的 FSM（有限状态机），在有限状态机中手掌张开并处于移动状态的时候，设定获取轨迹的起始点和终点，获取手势轨迹的时间序列，然后使用 DTW 算法对手势轨迹进行模板匹配，完成手势识别。

Kinect 获取到帧数据后，经过一系列的处理，转换成手势可以用的数据后，通过手势识别的相关算法过程，会将手势识别的结果转换成相应的事件，传递给应用层，以达到手势识别的通用性，那么本文设计的手势识别系统可以划分成三层：

第一层、Kinect 层：主要实现图像数据的处理和转换；

第二层、手势识别层：在 Kinect 层搭建的一层框架，把基于 Kinect 获取的数值特征转化成手势状态，根据手势状态提取动态手势轨迹，并实现特定的手势识别；

第三层、应用层：基于手势识别的结果对特定的完成应用程序中特定的手势功能，这三成架构的层次设计如图 2.5 所示。

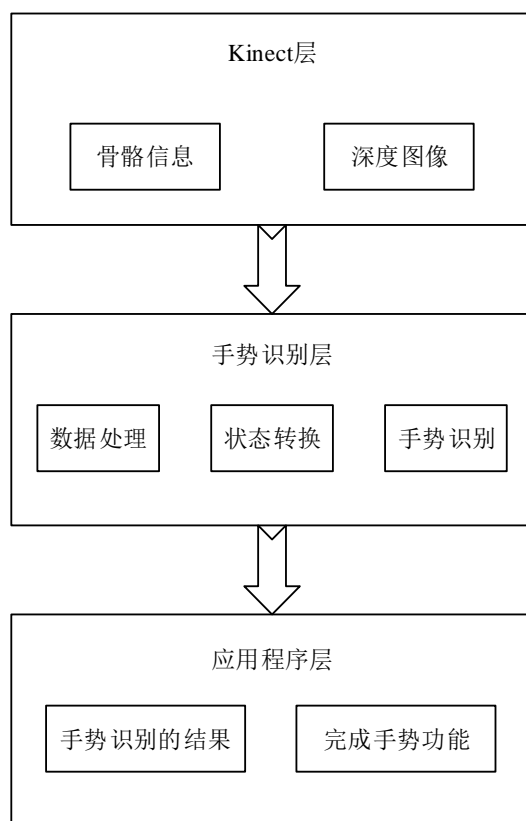


图 2.5 手势识别系统层次图

因此手势识别系统完成了接口间的耦合，可以在另一个程序中调用手势识别系统的识别接口，实现一些特定的功能。

## 2.3 本章小结

本章主要说明了本文中用到的一些关键技术，首先介绍了手势识别的相关技术，从手部特征提取和动态手势识别的方法来进行论述，通过对比，选择适合本文的一些方法，然后对系统的整个模块设计和层次设计进行了整个的介绍，详细说明了手势识别的框架设计，为后续章节做铺垫。

### 3 手势图像数据的获取和处理

#### 3.1 基于 Kinect 的需求分析

Kinect 设备很方便的提供了深度图像、彩色图像以及人体骨骼数据，这些数据使得开发人员对于人体骨架的 25 个关节点的坐标信息在摄像机坐标系中的获取和追踪变得十分方便。在微软的 Kinect for Windows SDK v2 中提供的接口函数中，人的手掌信息是由人体骨骼在手掌区域的 3 个关节点（手掌心点、手指尖点、手腕点）来确定的，然而这些信息并不能准确的提供手掌和手指的轮廓信息以及掌心点信息，因此本文中对于手势轨迹获取，是使用 Kinect 获取骨骼坐标点以及深度图像的相关数据，然后进行了一些必要的处理，用以更加方便和准确的获取手掌心点的坐标信息。

本文基于 Kinect 的骨骼图像和深度图像获取手势信息的流程如图 3.1 所示：首先通过骨骼信息中掌心点在 Kinect 摄像头三维空间坐标系中的位置，确定其在深度图像中的位置，基于这些信息利用深度阈值法获取手掌区域，然后对获取到的数据再一次结合深度图像进行手掌轮廓的提取，最后基于获取到的轮廓信息进一步分析手掌中心点和手指尖点。

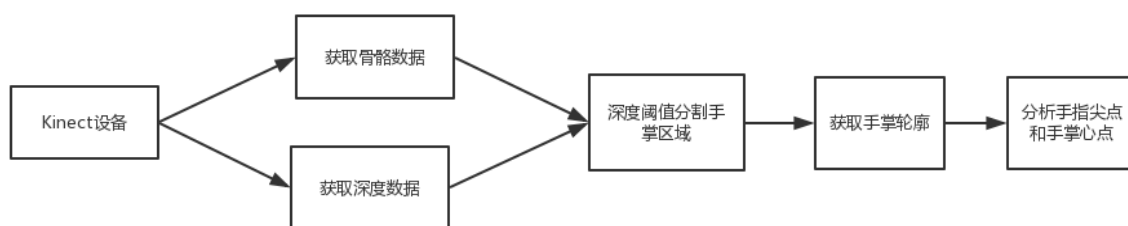


图 3.1 Kinet 图像模块的流程图

## 3.2 图像数据的获取

### 3.2.1 人体骨骼数据获取

与 Kinect 1 代相比, Kinect2 在数据稳定性上有着非常显著的提升, 其中一点就是 Kinect 对于骨骼数据的获取。Kinect 的骨骼空间坐标系是右手坐标系, 它以 Kinect 摄像头为原点, X 正方向朝右, Y 轴正方向朝上, Z 轴正放向下轴向用户。X 轴数据范围为  $-2.2 \sim 2.2$  m, 总共范围为 4.2 m, Y 轴范围为  $-1.6 \sim 1.6$  m, Z 轴范围为  $0 \sim 4$  m<sup>[42]</sup>, 如图 3.2 所示。

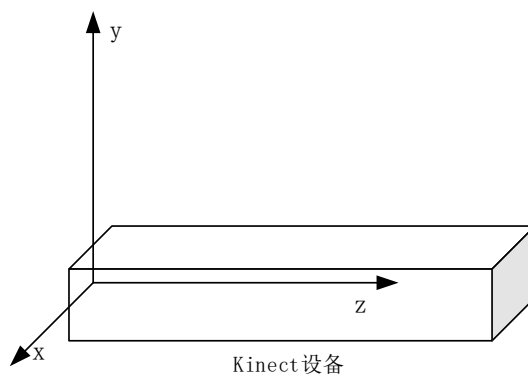


图 3.2 Kinect 骨骼空间坐标系示意图

Kinect 可以在三维空间中实时地追踪人体骨骼的 25 个关节点, 并获取这 25 个关节点的三维坐标值, 对这些人体的关节点进行追踪。如图 3.3 所示。

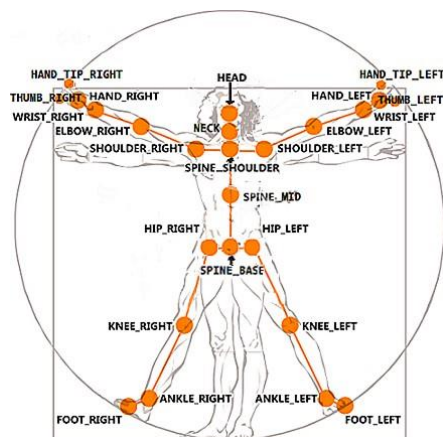


图 3.3 Kinect2 中获取并追踪的 25 个关节点信息

在微软提供的 Kinect for Windows SDK v2 中，关节点信息是存储在一个叫 Joint 的结构体中：

```
typedef struct _Joint
{
    JointType JointType;
    CameraSpacePoint Position;
    TrackingState TrackingState;
} Joint;
```

在这个结构体中存储了三个信息：关节点类型（JointType）、关节点的摄像头空间坐标点（Position）和追踪状态（TrackingState）；另外，Kinect 可以最多一次性跟踪 6 个人体的数据，存放在一个叫 IBody 的类对象的数组中，从每一个 IBody 对象出发可以获取到与每个人体相关的骨骼关节点。

在论文中，提取人体骨骼关节点的主要过程如图 3.4 所示，关键步骤是：

第一步：初始化 Kinect 传感器（IKinectSensor）结构体指针，打开 Kinect 设备，做好一系列的准备工作；

第二步：通过获取到的 IKinectSensor 结构体指针循环获取身体帧数据，提取出身体帧数据源（IBodyFrameSource），并打开身体帧读取器（IBodyReader）；

第三步：根据获取的身体帧读取器获取到各个关节点信息，存放到一个 Joint 结构体数组中，获取人体的 25 个关节点；

第四步：读取帧数据并重复第二步和第三步，直到最后程序关闭，关闭 Kinect 传感器，并释放相关指针所占据的内存空间。

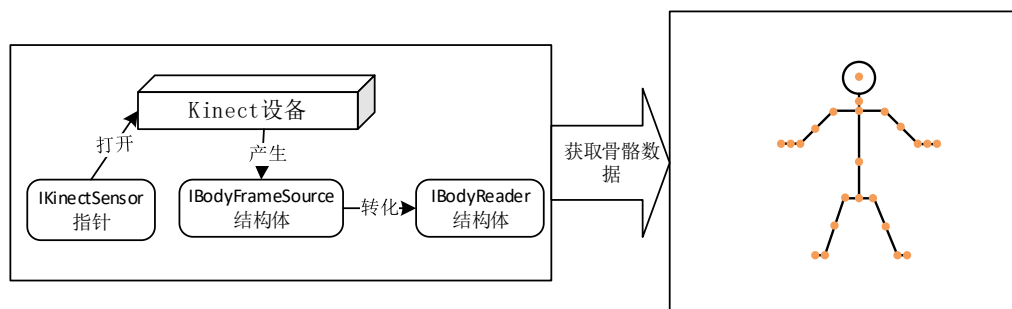


图 3.4 获取骨骼数据过程示意图

### 3.2.2 深度图像数据获取

Kinect 设备的另外一个特性就是可以很方便的物体在三维空间中的深度图像：相较于 Kinect1 代提供的 320\*240 的深度图像而言，Kinect2 能提供 512 \* 424 范围的深度图像。由于深度图像的坐标系是一个 x 轴正方向向右，y 轴正方向向下的坐标系，和 OpenCV 的坐标系统能够十分契合，可以很便捷地在 OpenCV 将获取到的深度图像数据用窗口显示出来，以便下一步的研究，因此本论文中的设计是将 Kinect 获取的骨骼数据和三维空间中的深度图数据结合起来进行处理的。

在 Kinect for Windows SDK v2 提供的 API 函数中，获取 Kinect 中获取深度图像数据和获取人体骨骼数据是相似的，所不同的是，深度图像获取的数据是存储在一个大小为 512\*424 的一维 2 字节（16 位）数组中，在这个数组中，每一个 16 位数据代表深度图中对应位置与摄像头之间的距离，那么这个数组中下标参数与对应下标的 16 位数据值关系是：

设获取的深度图数据用一个数组 `UINT16[512*424] depthArray` 来表示，x 表示深度图中的 x 坐标值，y 表示深度图中的 y 坐标值，那么数组的下标参数为 `index = y*424 + x`，且 `depthArray[index]` 中存储的就是深度图中坐标(x, y)处与 Kinect 摄像头之间的距离。

获取深度图数据的主要过程如图 3.5 所示，具体步骤如下：

第一步：初始化 Kinect 传感器（`IKinectSensor`）结构体指针，打开 Kinect 设备，做好一系列的准备工作；

第二步：通过获取到的 `IKinectSensor` 结构体指针循环获取深度帧数据，提取出深度帧数据源（`IDepthFrameSource`），并打开深度帧读取器（`IDepthReader`）；

第三步：根据获取的身体帧读取器获取到各个关节点信息，存放到一个 512\*424 大小的一维 2 字节（16 位）数组中，获取当前时间内的深度数据帧；

第四步：读取帧数据并重复第二步和第三步，直到最后程序关闭，关闭 Kinect 传感器，并释放相关指针所占据的内存空间。

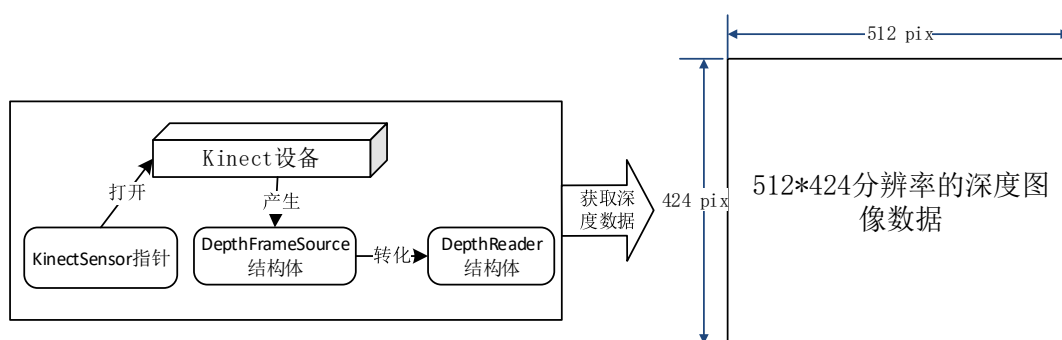


图 3.5 获取深度图像数据示意图

### 3.3 图像数据的处理

#### 3.3.1 数据封装

通过微软提供的 Kinect SDK 能十分方便的获取到人体骨骼数据和深度图数据，在此基础上，本文对获取到的数据进行了封装，以方便后面进一步的研究；首先，将获取到的深度图数据存储到一个二维数组中去，再获取到人体骨骼流数据，提取手掌心点的坐标值，然后将掌心点坐标转换到深度图的坐标系中去，将两者结合起来考虑，这样做的目的是使得 z 方向上的数据值更加准确。

从程序设计的角度来讲，本文中设计了两个主要的类来存储从 Kinect 获取到的三维坐标点：HandPoint 类和 Hand 类；Hand 类保存了三种与手掌相关的信息：

- (1) 关键点的坐标值：这些点有包含手区域矩形的左上角点和右下角点信息、手掌心点；
- (2) 手掌状态：Kinect 设备本身对手掌的五种状态进行了识别，它们是手掌张开（HandState\_Open）、手掌闭合（HandState\_Close，即握拳）、手指套索状态（HandState\_Lasso，即伸直食指和中指的状态）、未知状态（HandState\_Unknown）、未跟踪（HandState\_NotTracked）
- (3) 手掌轮廓点集合：用一个一维数组来记录手掌轮廓点，保存了从手掌区域提取出来的手掌轮廓上的点。

Hand 和 HandPoint 类之间的关系是：Hand 中包括了 HandPoint 中的点信息，包括一些关键点和轮廓点信息，故二者之间是关联与被关联的关系，具体是 Hand 直接关联 HandPoint，其 UML 类图如图 3.6 所示

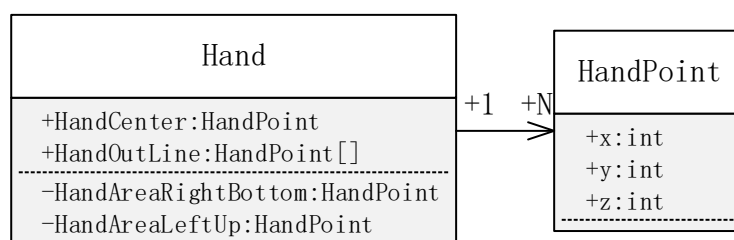


图 3.6 Hand 类和 HandPoint 类的 UML 图

### 3.3.2 手掌提取

对于 Kinect 获取到的深度数据和骨骼数据，要将手掌区域提取出来才能进一步进行手势识别，因此第一步就是提取手掌区域：由于 Kinect 对二维图像中手掌区域有优势的地方就是它可以提供二维图像无法提供的深度信息，且二维摄像头往往需要双目摄像头才能提供三维空间上的深度信息，而 Kinect 的摄像头本身就能获取红外信息，所以 Kinect 的摄像头能够非常便利的获取骨骼关节点的三维坐标信息，并且基于此获取到的数据是不受光照条件影响的，亦即在黑暗条件下也能准确的获取这些骨骼关节的深度信息，因而在第一步中获取手掌区域要利用 Kinect 提供的深度图数据，使用深度数据阈值方法提取出手掌区域，但是它也存在局限，其中比较重要的一点就是这样获取手掌区域只能利用 Kinect 提供的 SDK 来获取，而且目前 Kinect 在 Linux 和 Mac OS 平台上没有提供有效的驱动程序，所以平台会被限制在 Windows 操作系统上。

那么对于 Kinect 中获取手掌模块的步骤要分为三部分：获取手掌区域、获取手掌轮廓、获取手掌心点。

#### 3.3.2.1 分割手掌区域

分割手掌区域的方法主要是深度阈值法：根据骨骼点能准确确定掌心点所在位置的三维坐标点，假设该点的位置是  $P_{center}(x, y, z)$ ，那么根据  $x$  和  $y$  的值可以在



Kinect 获取的 424\*512 分辨率的深度图数据  $depth[424*512]$  数组中找到相应的点，这个转换公式(3.1)如下：

$$\begin{cases} index = y * 424 + x \\ HandDepth = depth[index] \end{cases} \quad (3.1)$$

假设设定的阈值为  $k$ ，那么选取深度图数据中深度值大小在范围  $[HandDepth - k, HandDepth + k]$  之间，则认为深度图数据中数值在这个范围内的深度值是属于手掌区域的，这是因为手掌区域的数据往往在深度值上和掌心点比较接近，而手掌本身往往是处于身体其他部位的前方，和手掌不在同一深度范围内，因此可以用这个深度平面内的数值来反馈手掌区域。其算法描述如下：

第一步：定义一个布尔类型的二维数组  $HandArea[424][512]$ ，初始化并全部赋值为 FALSE；

第二步：获取 Kinect 中骨骼帧数据中的手掌心点  $HandCenter$  点，转换到深度图坐标系中去，获取在深度图坐标系中  $HandCenter$  点的深度值  $CenterDepth$ ，即令  $CenterDepth = depth[index_{HandCenter}]$ ；

第三步：设置两个下标值  $i$  和  $j$ ，遍历整个深度图数据的数组，取出  $depth[j * 424 + i]$  的值，赋值给  $nDepth$ ，即令  $nDepth = depth[j * 424 + i]$ ，判断  $nDepth$  是否在

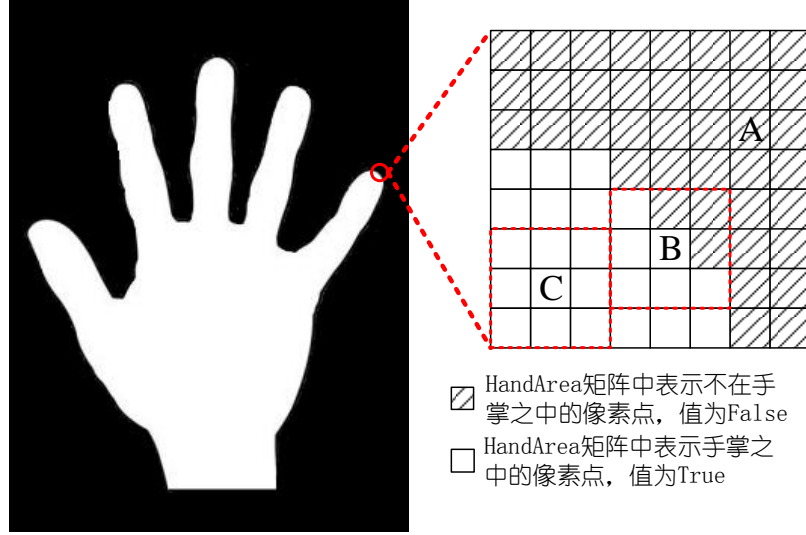
$$\{nDepth \mid CenterDepth - k \leq nDepth \leq CenterDepth + k\}$$

集合内，若  $nDepth$  的值在该集合中，则令  $HandArea[i][j] = TRUE$ ，否则令  $HandArea[i][j]$  为 FALSE。这样二维数组  $HandArea$  中值为 TRUE 与深度图对应的位置就是提取出来的手的区域。

### 3.3.2.2 手掌轮廓和手掌中心的提取

从前一步中提取出来手掌区域的二维数组后，下一步就是要根据提取到手掌区域数据进行手掌轮廓提取，提取手掌的思路是根据轮廓点附近的 8 个点的特征值变化，判别在手掌区域中哪一些点属于手掌中的轮廓点，哪些是属于手掌中的非轮廓点。

根据上面获取手掌区域的二维布尔矩阵 HandArea 可知，矩阵中的点，可以分为三大类：第一类就是手掌区域内的点，第二类是在手掌轮廓上的点，第三类是在手掌区域外的点。如图 3.7 所示。



- A: 自身为False，一定是在手掌外，不属于手掌轮廓点  
 B: 周边的8个点中有手掌区域内的部分，也有手掌区域外的部分，属于手掌轮廓点  
 C: 周边的8个点中全部都属于手掌区域内部的点，在手掌内部，不属于手掌轮廓点

图 3.7 HandArea 矩阵中的三类点

从数据结构角度来说，这三类点分别有以下特征：

- (1) 若  $\text{HandArea}[i][j] = \text{FALSE}$ ，则深度数据图中坐标点 $(i, j)$ 对应的点在手掌区域外部；
- (2) 若  $\text{HandArea}[i][j] = \text{TRUE}$ ，且在坐标点 $(i, j)$ 周围的 8 个点 $(i - 1, j - 1), (i, j - 1), (i + 1, j - 1), (i - 1, j), (i + 1, j), (i - 1, j + 1), (i, j + 1), (i + 1, j + 1)$ 中，存在一个点 $(x, y) \in \{(x, y) \mid \begin{cases} x = i - 1, i, i + 1 \\ y = j - 1, j, j + 1 \end{cases}, \text{且}(x, y) \neq (i, j)\}$ ，使得  $\text{HandArea}[x][y] = \text{FALSE}$ ，则深度数据图中坐标点 $(i, j)$ 对应的点在手掌轮廓上；
- (3) 若  $\text{HandArea}[i][j] = \text{TRUE}$ ，且在坐标点 $(i, j)$ 周围的 8 个点 $(i - 1, j - 1), (i, j - 1), (i + 1, j - 1), (i - 1, j), (i + 1, j), (i - 1, j + 1), (i, j + 1), (i + 1, j + 1)$ 中，对于所有的点 $(x, y)$

$\in \{(x,y) \mid \begin{cases} x = i-1, i, i+1 \\ y = j-1, j, j+1 \end{cases}, \text{ 且 } (x,y) \neq (i,j)\}$ , 使得  $\text{HandArea}[x][y] = \text{TRUE}$ , 则深度数据图中坐标点 $(i, j)$ 对应的点在手掌区域内;

根据三类点的划分, 可知获取手掌轮廓点的算法实现步骤如下:

第一步: 初始化一个布尔矩阵  $\text{HandOutline}$ , 其大小为  $424 \times 512$ , 将  $\text{HandArea}$  矩阵中的所有数据都复制到  $\text{HandOutline}$  中去;

第二步: 设定遍历下标  $i$  和  $j$ ,  $i$  的值从 1 到 424 变化,  $j$  的值从 1 到 512 变化, 获取  $\text{HandArea}[i][j]$  的值;

第三步: 声明一个  $\text{Byte}$  (一字节) 类型的变量  $\text{value}$ , 并初始化为  $0x00$  (二进制数值为  $00000000$ ), 若  $\text{HandArea}[i][j] = \text{FALSE}$ , 保持  $\text{value}$  的值为  $0x00$  不变;

第四步: 若  $\text{HandArea}[i][j] = \text{TRUE}$ , 依次获取  $\text{HandArea}[i-1][j-1]$ 、 $\text{HandArea}[i-1][j]$ 、 $\text{HandArea}[i-1][j+1]$ 、 $\text{HandArea}[i][j-1]$ 、 $\text{HandArea}[i][j+1]$ 、 $\text{HandArea}[i+1][j-1]$ 、 $\text{HandArea}[i+1][j]$ 、 $\text{HandArea}[i+1][j+1]$  得值, 若获取的值为  $\text{TRUE}$ , 则计算位运算  $\text{value} = \text{value} \ll 1 \mid 0x01$ , 否则令计算位运算  $\text{value} = \text{value} \ll 1 \mid 0x00$ ; 如此经过 8 次运算, 就可以将坐标点为 $(i,j)$ 位置的左上( $\nwarrow$ )、上( $\uparrow$ )、右上( $\nearrow$ )、左( $\leftarrow$ )、右( $\rightarrow$ )、左下( $\swarrow$ )、下( $\downarrow$ )、右下( $\searrow$ )这 8 个方向的临近点所存储的布尔值转换到  $\text{value}$  的第 7~0 位上, 并且该位置上位 1 表示对应方向的临近点在  $\text{HandArea}$  中为  $\text{TRUE}$ , 0 表示对应方向上的临近点在  $\text{HandArea}$  中为  $\text{FALSE}$ ;

第五步: 根据第四步计算的这八个方向上临近点在  $\text{HandArea}$  中的布尔值, 结合上文介绍的分类方法, 比较容易知道, 当  $\text{value}$  的值为  $0xFF$  (对应二进制数值为  $11111111$ ) 的时候, 表示坐标 $(i,j)$ 在深度图数据中对应的点是在手掌内部区域, 则将  $\text{HandOutline}[i][j]$  设定为  $\text{FALSE}$ , 而当  $\text{value}$  的第 7~0 位上存在任意一位位 1 的情况的时候, 表示坐标 $(i, j)$ 在深度图数据中对应的点是在手掌的轮廓上, 则将  $\text{HandOutline}[i][j]$  设定为  $\text{TRUE}$ 。

根据以上算法, 可以获取到手掌轮廓上的点信息, 并将这些信息存储到一个二维布尔矩阵, 同时用一个数组  $\text{points}$  来记录这些轮廓上的点信息。

前文提到, 利用 Kinect 设备获取到的掌心点用于提取手掌区域是比较可行的, 然而由于距离影响和数据晃动, 使得获取到的掌心点波动性比较大, 因此在获取了手掌轮廓之后, 接下来比较关键的一步就是寻找手掌心点。本文采取的手掌心点的检测方法是根据手掌的几何特征, 计算轮廓上的点到手中心点的中心点, 根据几何知识可以知道, 假设平面上有  $n$  个点:  $P_1(x_1, y_1), P_2(x_2, y_2), \dots, P_n(x_n, y_n)$ , 这些点的中心坐标设为  $C(x_c, y_c)$ , 则中心点计算算法如公式(3.2)所示。

$$\begin{cases} x_c = \frac{\sum_{i=1}^n x_i}{n} \\ y_c = \frac{\sum_{i=1}^n y_i}{n} \end{cases} \quad (3.2)$$

在上一步中, 将获取到的轮廓点的数据都放在一个数组中了, 用 `points` 来表示, 即 `points[i].x` 和 `points[i].y` 分别表示公式(3.2)中  $x_i$  和  $y_i$ , 根据公式(3.2)就可以计算出手掌的中心点, 从而进行下一步的研究工作。

### 3.4 本章小结

本章首先给出了手势识别系统的需求分析, 详细说明了 Kinect 提供的几种数据, 以及课题需要的内容。其次介绍了从图像数据的获取这个方面说明了获取整个手掌图像模块数据的实现过程, 并介绍了人体骨骼数据和深度数据结合的模块设计。接下来介绍了本文中图像数据的处理。最后对图像模块的显示进行详细说明分析。

首先详细叙述了图像模块的程序设计过程, 该模块作为本系统的核心功能之一, 能够获取用户输入的手势信息, 本章介绍图像模块方法的设计思想, 为后续实现章节提供指导。

## 4 手势识别模块的实现

在获取手掌数据后，至关重要的部分就是手势识别模块的实现了。本文在基于 Kinect 获取的图像基础之上，对动态手势识别进行了实现，搭建了手势库。手势识别模块主要分为手势轨迹处理、DTW 算法实现、手势库搭建三个步骤。本章主要从算法实现角度来介绍手势识别模块的实现。

### 4.1 手势轨迹的获取

手势轨迹是在一段时间内手掌坐标信息的集合，它有两个明显的特征：一、时效性，手势的轨迹是在一定时间内确定的，所以手势轨迹必定有个开始时段和结束时段；二、关联性，手势的轨迹是坐标点的集合，集合中点不是孤立的，往往点与点之间有着关联。

#### 4.1.1 坐标转换

从获取到的帧数据手掌心点信息中，可以比较精准地获取到掌心点在 Kinect 深度图像这个二维平面上的坐标信息。考虑到坐标系的本身的约束，获取到的坐标点往往只是在当前坐标系中才有实际意义，因此获取手掌心点坐标点后还要对这些数据进行坐标系转化，以提供给手势系统的坐标系使用。对于坐标系转换，有两种可行的方法：

第一种方法是等比例映射法：对于平面上的两个坐标系  $Oxy$  和  $Ox'y'$ ，用点  $(x, y)$  以及点  $(x', y')$  分别来表示这个两个坐标系中两个可以互相转化的点，设由  $x$  转换到  $x'$  的比例系数是  $k_1$ ，由  $y$  转换到  $y'$  的比例系数是  $k_2$ ，那么这两个点之间的转换算法如公式(4.1)所示：

$$\begin{cases} x' = k_1 * x \\ y' = k_2 * y \end{cases} \quad (4.1)$$

对于一般的可视化的坐标，都是以屏幕向右的方向为  $x$  轴的正方向，屏幕向下的放下为  $y$  轴的正方向，坐标值都为正值，可视的范围是整个屏幕或者整个窗口的大小，所以可视点的坐标值会规范到一个矩形之中。假设待转换的两个可视化的坐标系的

矩形分别为 frame1 和 frame2，那么如果由 frame1 表示的坐标系映射到 frame2 表示的坐标系， $k_1 = \text{frame2.width} / \text{frame1.width}$ ， $k_2 = \text{frame2.height} / \text{frame1.height}$ ；

第二种方法是变化量转化法：对于有时效性的坐标系  $Oxy$  和  $Ox'y'$ ，设某一时刻  $t$ ，两个坐标系中坐标点分别为  $(x(t), y(t))$ ， $(x'(t), y'(t))$ ；在  $t+1$  时刻， $(x(t), y(t))$  变化为  $(x(t+1), y(t+1))$ ， $t \sim t+1$  时间内， $Oxy$  坐标系中在  $x$  方向上的变化量为  $dx = x(t+1) - x(t)$ ， $dy = y(t+1) - y(t)$ ，对于  $Oxy$  坐标系中的变化量，指定一个坐标值变化的范围，设该范围是一个矩形区域 frame1，而假设  $Ox'y'$  坐标系中坐标值变化的矩形区域范围是 frame2，那么可以将  $Oxy$  坐标系中坐标值变化量等比例映射到  $Ox'y'$  坐标系中去，即：

$$\begin{cases} dx' = dx \times \frac{\text{frame2.width}}{\text{frame1.width}} \\ dy' = dy \times \frac{\text{frame2.height}}{\text{frame1.height}} \end{cases} \quad (4.2)$$

如此计算出在  $Ox'y'$  坐标系中  $x$  方向和  $y$  方向上的变化量，从而计算  $t+1$  时刻在  $Ox'y'$  坐标系中，由在坐标系  $Oxy$  上的点  $(x(t+1), y(t+1))$  转换而来的坐标值是  $(x'(t+1), y'(t+1))$ ，其中  $x'(t+1) = x'(t) + dx$ ， $y'(t+1) = y'(t) + dy$ ；

其中，第一种方法转换的特点是坐标点转换都是按等比例转换的，所以每一个时刻，两个坐标系中获取到的轨迹点是相似的，而且只要原坐标系中坐标点的数值没有发生突变，转换后的坐标点数值就不会发生突变，因而只要元坐标点足够精确，经过转换后的坐标点也是精确的；第二种方法中坐标点数值是基于坐标点变化量而言的，本身转换后的坐标点对于原坐标点的依赖性并不是太强，对于获取手势轨迹而言，往往会更加稳定，本文采用第二种方法对坐标点进行转换。

#### 4.1.2 数据封装

从 Kinect 中获取的帧数据，是按照时序一帧一帧获取的，所以获取到的数据和时间是有关系的。在微软提供的 Win32API 函数中，调用 GetTickCount 这个函数可以获取系统时间，并且精确到毫秒。本文中，申明了一个结构体 HandArgs，用来存当前时刻获取到手掌信息的各种参数：

```
typedef struct _HandArgs
{
    int x, y, z;
    int vx,vy,vz;
```

```
int dx, dy, dz;
...
}HandArgs
```

其中  $x, y, z$  的值是保存的当前时刻手掌心点的在各个方向上的坐标值,  $vx, vy, vz$  用来计算当前时刻和上一时刻掌心点在各个方向上的变化率,  $dx, dy, dz$  用来保存当前时刻和上一时刻掌心点在各个方向上的变化量。这些数据是通过以下步骤获取到的:

第一步: 定义两个时间参数 `currentTime` 和 `lastTime`, 分别记录当前时刻和上一时刻的毫秒数, 并均初始化为 0;

第二步: 从 Kinect 传感器中获取帧数据, 对帧数据进行处理后, 获取到手掌心点的坐标信息, 将坐标点信息保存到 `HandArgs` 的  $x, y, z$  变量中并将当前系统时间的毫秒计数保存在 `currentTime` 中。

第三步: 在判断 `lastTime` 非零的情况下, 计算获取到两帧数据之间的时间差  $\text{timeInterval} = \text{currentTime} - \text{lastTime}$ , 并获取 `HandArgs` 中保存的前一帧的  $x, y, z$  值, 计算两者之间的差值, 保存到  $dx, dy, dz$ , 同时计算  $dx / \text{timeInterval}, dy / \text{timeInterval}, dz / \text{timeInterval}$  分别保存到  $vx, vy, vz$  中。

## 4.2 建立手势 FSM

不同的手势有不同的数值特征, 包括手掌在某一特定时间内的坐标点的变化量和变化速率, 双手之间的距离和双手关于  $y$  轴方向上形成的角度值等, 根据这些基本特征可以利用数值算法对手势轨迹进行分类。

### 4.2.1 手掌的状态定义

在某一单独的特定时间、特定条件下, 手势会呈现出不同的状态, 本文中, 定义了 17 种手势识别的状态:

1. Hand Open: 从 Kinect2 中能够获取到的手掌张开的状态;
2. Hand Retain: 手掌张开后, 停留在屏幕上一段时间不动的状态;
3. Retain Move: 手掌在屏幕上停留后在一段时间内移动的状态;

4. Hand Click: 手掌在屏幕上停留后在 z 方向上快速向前移动的状态, 触发点击手势;
5. Hand Move: 手掌张开状态后移动的状态;
6. Hand Shift: 手掌张开后, 在平面坐标系上快速移动的状态;
7. Hand Hold: 手掌握拳状态, 可以从 Kinect2 中读取到的状态;
8. Hold Move: 手掌握拳后在三维空间中移动的状态;
9. Grab: 手掌从张开手掌到握拳变换的中间状态;
10. Release: 手掌从握拳到张开手掌变换的中间状态;
11. Drag: 手掌握拳后在平面坐标系上快速移动的状态;
12. Zoom: 手掌开始识别缩放旋转手势的状态;
13. Zoom In: 双手放大手势状态
14. Zoom Out: 双手缩小手势状态
15. Rotate CW: 双手顺时针旋转状态
16. Rotate CCW: 双手逆时针旋转状态
17. Unknown: 手掌未知状态;

#### 4.2.2 手势状态转换

有限状态机用于对系统的动态行为的建模, 一般用状态图来可视化表示。状态机是一个可以确定状态规则以及在特定条件下实现状态转移的设计模式<sup>[42]</sup>, 可以确定的是, 手掌数据一定是处于一种既定的状态之下, 在给定的外在条件下, 会发生状态转移。在本文中, 实现了手势状态转换的状态机, 这样手势状态构成了一个有向图, 如图 4.1 所示, 手势状态通过条件的变化就会发生改变。在状态发生改变的一些时机就会触发手势事件。



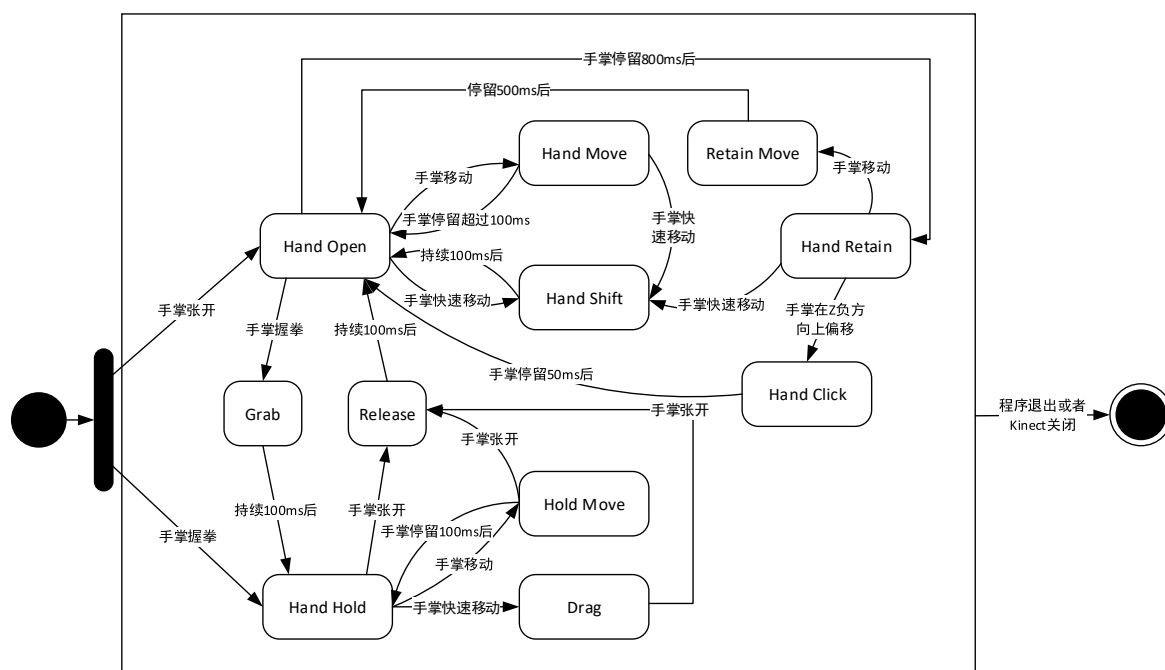


图 4.1 手势状态转换图

### 4.2.3 数值特征

对于单手的移动可以确定存在着三种类型：移动、快速移动、停留。根据获取到手掌参数，可以确定手掌在坐标系中的移动速度，移动变化量，再根据前后两帧之间的数值变化，可以对这三种单手移动类型进行判别。假设对于手势的前后两帧  $f_1$  和  $f_2$ ，获取到的参数是三维空间上的变化量  $dx$ 、 $dy$ 、 $dz$  和前后两帧的速度值  $vx$ 、 $vy$ 、 $vz$ ，根据大量的实验得到这几种手势移动的类型数值特征如下：

- (1) 手掌移动：手掌心点坐标在三维空间中发生变化，具体表现为  $|dx| > 1$  像素单位或者  $|dy| > 1$  像素单位或者  $|dz| > 1$  像素单位；
- (2) 手掌快速移动：手掌心点坐标在三维空间中速度出现快速变化，具体表现为  $|vx| > 60$  像素单位/ms 或者  $|vy| > 60$  像素单位/ms 或者  $|vz| > 60$  像素单位/ms；
- (3) 手掌停留：手掌心点坐标在三维空间中在一个数值范围内保持不变，具体表现为  $|dx| \leq 1$  像素单位或者  $|dy| \leq 1$  像素单位或者  $|dz| \leq 1$  像素单位；

对于双手，以前后两帧双手之间的距离  $d_{LR}$ 、双手之间距离的变化率  $v_{LR}$ 、双手之间连线与  $y$  轴负方向上的夹角  $\alpha_{LR}$  和夹角的变化率  $\omega_{LR}$  这四个参数作为特征数据，如图 4.2 所示，

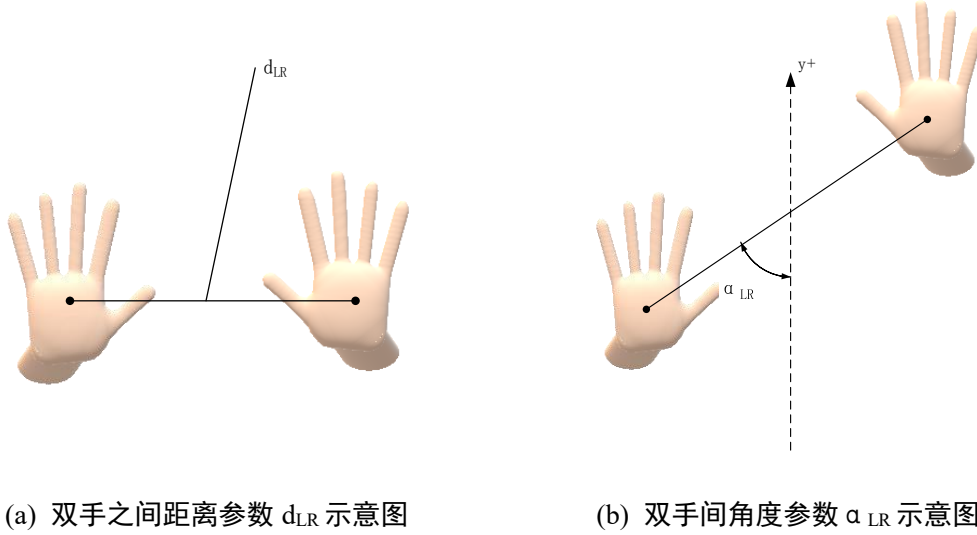


图 4.2 双手特征参数示意图

双手具体有缩放和旋转两种特征类，在大量的实验验证下，得到其特征值描述如下：

(1) 双手缩放： $d_{LR}$  的值发生快速变化，具体为  $|v_{LR}| \geq 8$  像素单位/ms，并且  $d_{LR}$  表示缩放率；

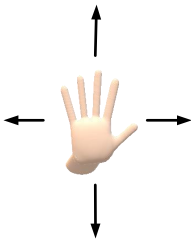
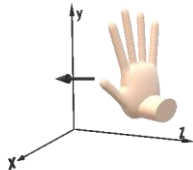
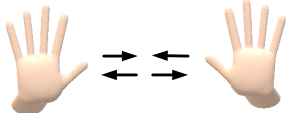
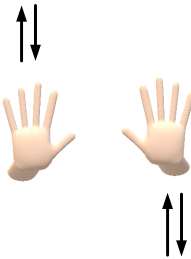
(2) 双手旋转： $\alpha_{LR}$  的值发生快速变化，具体为  $|\omega_{LR}| \geq 10$  弧度/ms，并且  $\alpha_{LR}$  表示旋转的角度值参数。

#### 4.2.4 手势设计

基于语法的手势识别方法是目前逐渐慢慢发展起来的一种手势识别方法，其中比较常见的基于语法的方法就是 FSM 了。初始时由于手掌的相关数据还没有被 Kinect 获取到，因此此时手势识别系统是处于未识别状态，一旦 Kinect 获取到深度数据和骨骼数据的时候，系统处于手势识别的开始状态，根据手掌的一些数值特征对手掌状态进行划分，在划分了手势状态之后，本文基于手势的状态改变设计四个可以识别的基本手势：手掌挥动，手掌点击，双手缩放和双手旋转的手势。

每一个手势都有对应的初始状态和结束状态，且手势分为离散和连续两种类型，具体设计如表 4.1 所示。

表 4.1 基于 FSM 的四种基本手势设计

手势示意图	手势名称	起始状态	识别状态	结束状态	是否连续
	手掌挥动	Hand Move	Hand Shift	Hand Move	否
	手掌点击	Hand Retain	Hand Click	Hand Open	否
	双手缩放	Zoom	Zoom In/Out	Zoom	是
	双手旋转	Zoom	Rotate CW/CCW	Zoom	是

### 4.3 DTW 算法

一段具有时序的数据往往是指将同一统计指标的数值按其发生的时间先后顺序排列而成的数列，对于大部分的学科而言，时间序列是数据处理中一种十分常见的表达方式，比如一段视频数据可以看成是图片数据的时间序列、一段语音数据可以看成

音节数据的时间序列等。在时间序列中,需要进行相似性比较的两个时间序列片段长度可能不相等,为了解决时间序列长度不同的模板匹配问题,DTW 算法在上世纪 70 年代左右被提出来,并最早用于处理语音方面的识别分类问题。

DTW (Dynamic Time Wrapping, 动态时间规整) 算法是语音识别中的一种经典的算法<sup>[44]</sup>,该算法是基于动态规划 (Dynamic programming, DP) 的算法思想实现的。它能够简单灵活地实现模板匹配,能解决很多离散时间序列匹配问题。本文中获取到手势轨迹就是基于时间序列,和语音识别的过程是十分相似的,因此可以利用该算法实现对既定手势的匹配,达到比较精准地识别结果。

#### 4.3.1 DTW 算法原理

假设有两个时间序列  $M$  和  $N$ , 他们对应的长度 (即帧数长度) 分别为  $m$  和  $n$ , 其中  $M$  是参考模板, 它对应的序列设为  $\{M_1, M_2, \dots, M_i, \dots, M_m\}$ ,  $N$  是待匹配的模板, 它对应的序列设为  $\{N_1, N_2, \dots, N_j, \dots, N_n\}$ , 规定一个距离函数  $Dis(M, N)$ , 计算时间序列  $M$  和时间序列  $N$  之间的距离, 由此可知, 如果两个时间序列之间的距离越小, 它们之间的相似度就越高。为了获取两个时间序列的距离, 假设  $i$  和  $j$  分别表示  $M$  和  $N$  中任意选择的两个对应帧的帧号, 则规定一个计算法则  $d$ , 用  $d(M_i, N_j)$  表示这两帧之间的距离, 在一般的 DTW 算法中, 距离计算通常采用的距离是欧式距离<sup>[45]</sup>。那么对于不同的  $m$  值和  $n$  值, 有以下两种情况:

(1) 若  $m=n$ , 说明序列  $M$  和序列  $N$  中帧数是一致的, 这样可以直接计算两个序列一一对应的距离, 即将  $M_1$  与  $N_1$  帧、 $M_2$  和  $N_2$  帧、...、 $M_m$  和  $N_m$  帧对应起来, 依次计算  $d(M_1, N_1)$ ,  $d(M_2, N_2)$ , ...,  $d(M_m, N_m)$  的值, 然后再求和, 得到两个序列的距离值, 如公式(4.3)所示

$$Dis(M, N) = \sum_{i=j=1}^m d(M_i, N_j) \quad (4.3)$$

(2) 若  $m \neq n$ , 说明序列  $M$  和序列  $N$  中的帧数是不一致的, 对于这种情况, 有两种可行的办法进行处理:

1. 采用线性缩放对齐的方法: 将短的序列线性放大到和长的序列长度相同, 然后再进行(1)中的距离计算, 或者将长的序列线性缩短到和短序列一样的长度再进行(1)中的距离计算。

下面对此方法进行说明: 假设  $m > n$ , 也即模板序列  $M$  中的帧数长度大于待匹配序列  $N$  的帧数长度, 这种情况下, 可以将待匹配序列  $N$  用线性放大的方式, 扩大成一个帧数长度为  $m$  的序列  $N' \{N'(1), \dots, N'(n)\}$ , 然后再和  $M$  模板序列进行匹配, 计算得到两者的距离。反之,  $m < n$  的情况也可以通过这种方法获取到最终结果。

这种方法, 虽然可以解决不同长度的时间序列之间的匹配问题, 但是对于不同的时间序列, 在各段的不同情况下会产生或长或短的变化, 即各个序列中元素的伸缩程度是不同的, 如果采用这种方法, 就忽略了这种不同伸缩程度带来的变化, 而统一地使用线性收缩的方式使得两个时间序列的长度变得相同, 势必会造成识别的结果精度不准, 为了更佳的识别效果, 一般会采用动态规划的方法;

2. 采用动态规划的方法: 为了将这两个序列对齐, 动态规划算法会构造一个  $m \times n$  的矩阵网格, 矩阵中元素  $(i, j)$  表示序列  $M$  中第  $i$  帧和序列  $N$  中第  $j$  帧的距离, 用  $d(M_i, N_j)$  来表示, 这个距离越小, 表示这两个序列中对应的这两帧之间的相似度越高, 也可以理解为失真度越低。

由于这两个序列是有时间顺序的, 也就是说整个匹配的过程, 必定是要寻找一条路径, 经过矩阵网格的  $(1, 1)$  到达矩阵网格的  $(m, n)$ 。直观来看, 如果把序列  $M$  的帧号  $1 \sim m$  放到一个二维坐标系的横轴, 做为横轴的标号, 把  $N$  的帧号  $1 \sim n$  放到这个坐标系的纵轴, 作为纵轴的标号, 那么动态规划的方法归结为从这个坐标系中找到一条路径, 使得这条路径能够从从左下角到达右上角, 并且这条路径经过的元素值之和最小。

为了方便描述这条路径, 假设这条路径经过的点依次是  $(X_1, Y_1), \dots, (X_i, Y_j), \dots, (X_m, Y_n)$ , 其中  $(X_1, Y_1) = (1, 1)$ ,  $(X_n, Y_n) = (m, n)$ , 定义一个二维函数  $\phi(i, j)$  表示到达点  $(X_i, Y_j)$  时, 与邻近点元素积累的距离之和的最小值, 那么满足公式(4.4)

$$\begin{cases} \phi(i, 1) = \sum_{k=1}^i d(M_k, N_1), \quad i = 1, 2, \dots, m \\ \phi(1, j) = \sum_{k=1}^j d(M_1, N_k), \quad j = 1, 2, \dots, n \\ \phi(i, j) = \min\{\phi(i-1, j-1), \phi(i-1, j), \phi(i, j-1)\} + d(M_i, N_j) \end{cases} \quad (4.4)$$

这个公式体现了局部最优路径的选择，那么从匹配起点就能沿着局部最优的路径反向找到最初的匹配点，获的全局最优路径。这种逐点求取帧匹配距离的算法就是 DTW 算法。

#### 4.3.2 DTW 算法的约束条件

根据上面介绍的 DTW 算法基本原理，对于满足可以用 DTW 算法匹配的两段时间序列  $M(M_1, M_2, \dots, M_i, \dots, M_m)$  和  $N(N_1, N_2, \dots, N_j, \dots, N_n)$ ，设最后找到的满足两者之间距离和最小的路径  $L$ ，这条路径一共经过  $s$  个点且  $L_i$  表示这条路径上面的第  $i$  个点 ( $1 \leq i \leq s$ )，那么它将满足三个约束条件：

- (1) 边界条件唯一性：这条路径的起点必定是  $(1, 1)$ ，终点必定是  $(m, n)$ ，即  $L_1=(1,1)$ ， $L_s=(m, n)$ ；
- (2) 连续性：对于路径  $L$ ，如果第  $k$  ( $1 \leq k < s$ ) 个点  $L_k$  为  $(i, j)$ ，那么在路径  $L$  上第  $k+1$  个点必定是  $(i+1, j)$ 、 $(i, j+1)$ 、 $(i+1, j+1)$  之中的一个；
- (3) 单调性：根据条件(2)，路径  $L$  上第  $k$  ( $1 \leq k < s$ ) 个点  $(i_k, j_k)$  和第  $k+1$  个点  $(i_{k+1}, j_{k+1})$ ，必定满足： $i_k \leq i_{k+1}$  且  $j_k \leq j_{k+1}$

这三个约束条件保证了在上面构造的  $m \times n$  的矩阵网格中每一个点的路径只有三个方向，如图 4.3 所示。

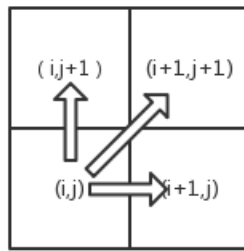


图 4.3 DTW 矩阵网格最优路径的三个方向

也就是说，若最优路径通过了网格中的点 $(i, j)$ ，那么在最优路径的下一个网格点必定只有三种情况： $(i, j+1), (i+1, j), (i+1, j+1)$

### 4.3.3 DTW 算法实例

下面用一个实例来说明 DTW 算法的计算过程，假设有模板时间序列  $M\{3,5,2,9,10\}$ ，有两个待匹配的时间序列  $N_1\{4, 13,12,12,6,4\}$ ， $N_2\{1,6,3,12\}$ ，它们的帧长度分别为  $m=5$ 、 $n_1=6$ 、 $n_2=4$ ，它们的时序图如图 4.4 所示，

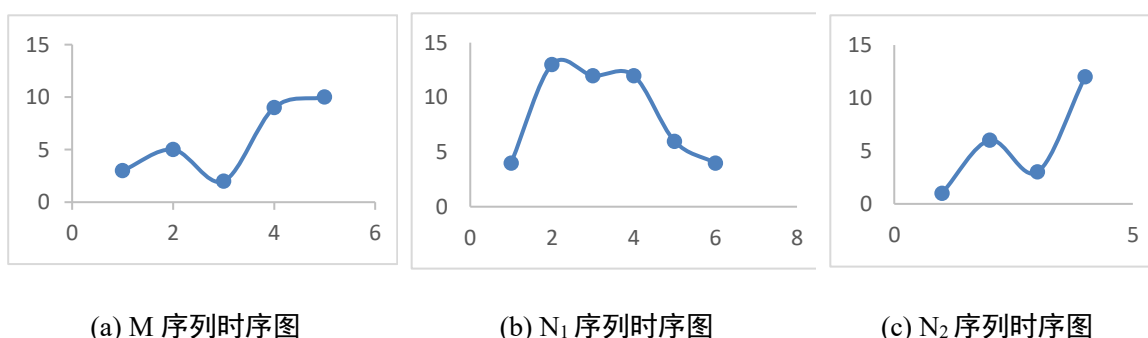


图 4.4 样例时序图

那么计算  $M$  序列和  $N_1$  序列的距离值（相似度）的 DTW 算法过程如下：

第一步：构造距离矩阵  $Dis[m][n_1]$ ，为方便理解，距离矩阵的右方向是  $i$  值的递增方向，上方向是  $j$  值的递增方向， $Dis[i][j]$  中计算  $M$  中第  $i$  帧和  $N_1$  中第  $j$  帧的欧式距离，即两个帧数据值之间的差值的绝对值，那么构造的距离矩阵如图 4.5 所示。

1	1	2	5	6
3	1	4	3	4
9	7	10	3	2
9	7	10	3	2
10	8	11	4	3
1	1	2	5	6

图 4.5 序列  $M$  和序列  $N_1$  的距离矩阵

第二步：构造累加求和距离矩阵  $D[m][n_1]$ ， $D[i][j]$ 表示经过 $(i, j)$ 是累加的最小距离值，根据公式(4.4)，计算所得的结果如图 4.6 所示

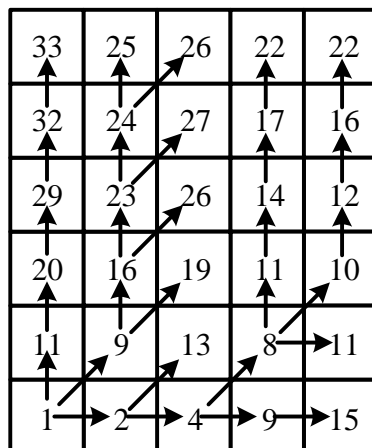


图 4.6 序列 M 和序列  $N_1$  的累加求和距离矩阵

右上角的数值  $D[m][n_1]=22$  就是最后所求的序列 M 和序列  $N_1$  的之间的距离，根据箭头指示的方向，由 DTW 算法求得的这条路径是 $(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (4,2) \rightarrow (5,3) \rightarrow (5,4) \rightarrow (5,5) \rightarrow (5,6)$ 。

计算 M 序列和  $N_2$  序列的距离值过程和上述算法相同，第一步构造的距离矩阵如图 4.7 所示。

9	7	10	3	2
0	2	1	6	7
3	1	4	3	4
2	4	1	8	9

图 4.7 序列 M 和序列  $N_2$  的距离矩阵

第二步中构造的累加求和距离矩阵如图 4.8 所示，求得两个序列之间的距离为  $D[m][n_2]=9$ ，再根据箭头的方向，算得距离最短的路径是 $(1,1) \rightarrow (2,2) \rightarrow (3,3) \rightarrow (4,4) \rightarrow (5,4)$ 。



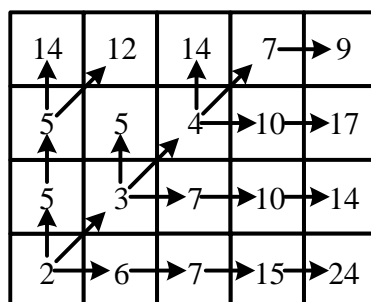


图 4.8 序列 M 和序列  $N_2$  的累加求和距离矩阵

由于  $\text{Dis}\{M, N_1\} > \text{Dis}\{M, N_2\}$ , 说明  $N_2$  与 M 的距离比  $N_1$  与 M 的距离更小, 即  $N_2$  与 M 更相似。

## 4.4 基于 DTW 的手势识别

前文提到, 手势轨迹是一个时间序列, 它必定是一个有起点和终点, 一段时间内有序的手掌坐标点数据的集合。因此, 借鉴语音识别的案例, DTW 算法也可以用来进行手势识别。

由于 DTW 算法是一种基于模板匹配的算法, 所以需要提提供样本模板数据, 建立模板手势库。在已规定好的轨迹特征模板手势库的情况, 再用 DTW 算法对手势和手势库中的模板数据进行匹配, 才能有效地识别出特定的手势。

### 4.4.1 建立模板手势库

对于手势库的建立, 首先第一点就是要设计几种已经可以实际使用的手势, 并把手势的特征轨迹点记录下来。考虑到易于理解和易于扩展手势库的使用场景, 手势库数据使用 xml 文件来记录, xml 的层次结构如图 4.9 所示, 即 xml 的根节点手势库 (gestureLib) 保存了多个手势 (gesture), 手势又是由多条轨迹 (stroke) 组成, 一个轨迹是有多个轨迹坐标点 (point) 组成, 如此就构建了一个保存样本模板的 xml 数据库文件。



图 4.9 手势库 xml 文件的层次结构图

构建 xml 文件的一个显著的优点就是文件易于理解，易于编辑，可读性高。但是对于大量数据的解析会比较缓慢，且文件格式复杂，如果读取到内存中会占据大量不必要的格式标签信息，因此对于读取到内存中的模板文件信息，定义了一个协议规范，将 xml 文件转换成二进制的 model 文件，用以缩减内存存储消耗。

model 文件的协议规范：首先定义了一个 ModelHeader，用来保存模板文件的基本信息，结构体定义如下：

```
struct ModelHeader
{
    char sign[4];
    short ver;
    short type;
    short width;
    short height;
    short pointDist;
    int count;
};
```

其中，各个变量的意义是：

**sign:** 模板文件的标志，是一个字符串，用来标识模板文件，默认为“SGR”，是简单手势识别（simple gesture recognizer）的意思；

**ver:** 模板文件的版本号，用来标识模板文件的版本；

**type:** 模板文件的类型；

**width:** 手势轨迹的默认宽度；

**height:** 手势轨迹的默认高度；

**pointDist:** 手势轨迹之间默认的距离差值，用以方便把轨迹稀疏化或者补全离散轨迹的临界阈值；

**count:** 模板文件中已经定义了的手势个数。

然后对于各个手势，定义了一个 UTF-8 字符来标识手势轨迹的信息，称为手势标识码，比如，用○（UTF-8 编码为&#x25CB;）来标识圆圈手势。在程序设计中定义 FeatureHeader 和 Feature 结构体与各个手势对应，FeatureHeader 结构体中存储了各个手势的基本信息，而 Feature 结构体中定义了手势的内容，包括手势坐标点的信息。他们的结构体定义如下：

```
struct FeatureHeader{
    char word[4];
    int trackCount;
    int count;
};
struct Feature{
    int trackCount;
    string word;
    vector<int> feature;
};
```

其中 word 表示手势轨迹的标识码；trackCount 标识一个手势中包含的轨迹数目，一般而言手势轨迹数目只有一条，但是例如对于与“×”对应的手势轨迹，可以看成需要两条轨迹来确定手势信息，所以需要定义一个 trackCount 变量，以保证程序可扩展性；FeatureHeader 中的 count 和 Feature 中的 feature.size()是一致的，Feature.feature 中存储的是各个手势轨迹的特征点坐标信息，具体内容是，将各个手势轨迹的特征点坐标的 x 值和 y 值拆分开来，依次按序保存到 Feature.feature 这个动态数组中。

定义的 model 文件格式协议如图 4.10 所示：

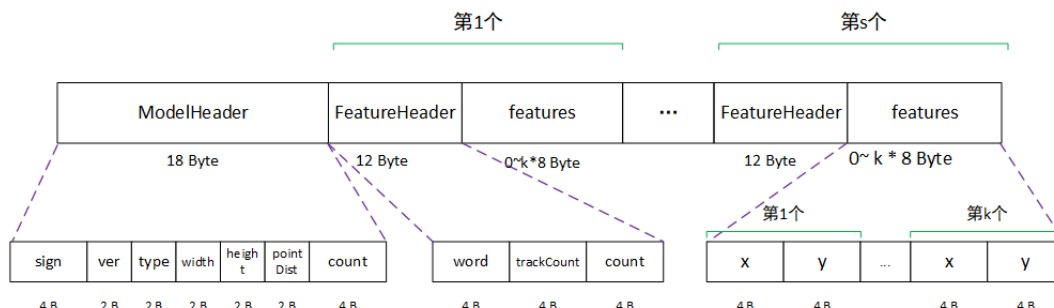


图 4.10 model 文件的数据格式

一个 model 文件在头部会插入一个 ModelHeader，占据 18 个字节，然后根据手势库 xml 文件中手势的个数  $s$ ，依次插入  $s$  个 FeatureHeader 和  $s$  个 features 字段，其中一个 FeatureHeader 占据 12 个字节，features 所占据的空间由手势确定的特征点坐标个数决定，如果特征点坐标个数是  $k$ ，那么 features 所占据的空间就是  $k * 8$  字节（ $x$  数值和  $y$  数值会分开来，因此是  $4B+4B=8B$ ）。

#### 4.4.2 模板手势库轨迹处理

xml 模板手势库文件中会对手势提取合适的特征坐标点然后记录下来，在组建手势库的过程中，由于 xml 文件中定义的手势轨迹点只有离散的几个特征点，手势数据往往需要通过轨迹补全算法是离散的点连续化，然后在获取连续轨迹点基础上做后续处理。例如假定存在一个三角形（ $\triangle$ ）手势，那么它的手势特征坐标点就是三角形的三个顶点。根据这三个顶点就可以补全完整的三角形轨迹，使得轨迹连续化。

##### 4.4.2.1 手势轨迹连续化

那么手势轨迹特征点的连续化这个问题归结为在相邻两个离散的坐标点之间插入连续线性变化的坐标点，其算法执行过程如下：

(1) 对于两个离散点 $(x_1, y_1)$ 和 $(x_2, y_2)$ ，定义两个变量  $\Delta x$  和  $\Delta y$ ，分别计算两个点  $x$  方向和  $y$  方向上的变化值绝对值，即  $\Delta x = |x_2 - x_1|$ ， $\Delta y = |y_2 - y_1|$ ，比较  $\Delta x$  和  $\Delta y$  的大小，取两者中的最大者并减一后的数值，设这个值为  $\Delta p$ ，即  $\Delta p = \max\{\Delta x, \Delta y\} - 1$ ，

那么在 $(x_1, y_1)$ 和 $(x_2, y_2)$ 之间插入的坐标点个数是 $\Delta p$  个,  $\Delta p$  确定了插入点是默认以平面上哪个方向进行推进。

(2) 设插入的 $\Delta p$  个坐标点是 $P\{p_1, p_2, \dots, p_{\Delta p}\}$ , 对于第 $i$  个坐标点 $p_i(x_i, y_i)$ , 有一个斜率值 $k_i$  与之对应, 且 $k_i$  的计算如公式(4.5)所示,

$$k_i = \begin{cases} \frac{|x_2 - x_i|}{|y_2 - y_i|}, & \text{若 } \Delta p = \Delta y \\ \frac{|y_2 - y_i|}{|x_2 - x_i|}, & \text{若 } \Delta p = \Delta x \end{cases} \quad i \in [1, \Delta p] \quad (4.5)$$

(3)  $k_i$  表示 $p_i(x_i, y_i)$ 与 $(x_2, y_2)$ 之间的斜率, 这个值的变化会影响第 $(i+1)$  个点 $p_{i+1}(x_{i+1}, y_{i+1})$ 的计算。根据计算出的 $p_i$ , 设第 $i+1$  个点是 $p$ , 设 $p_0=(x_1, y_1)$ , 若 $\Delta p = \Delta x$ , 则这个坐标点迭代的计算如公式(4.6)所示

$$\begin{cases} x_{i+1} = x_i + 1, k_i \leq \frac{1}{2} \\ y_{i+1} = y_i, k_i > \frac{1}{2} \end{cases} \quad (4.6)$$

而若 $\Delta p = \Delta y$ , 则这个坐标点迭代的计算如公式(4.7)所示

$$\begin{cases} x_{i+1} = x_i, k_i \leq \frac{1}{2} \\ y_{i+1} = y_i + 1, k_i > \frac{1}{2} \end{cases} \quad (4.7)$$

如此按照上述迭代计算过程, 就可以将轨迹上每相邻的两个离散点之间的数据补全了, 下面用一个示例来解释这种算法的正确性:

假设有两个离散点 $A(1,1)$ 和 $B(10,6)$ , 则上式中 $\Delta x=9$ ,  $\Delta y=5$ , 取 $\Delta p=8$ , 即需要在点 $A$  和点 $B$  之间插入 8 个连续的点, 根据上述计算过程, 算得计算所得的这个 8 个点是 $(2,2)$ 、 $(3,2)$ 、 $(4,3)$ 、 $(5,3)$ 、 $(6,4)$ 、 $(7,4)$ 、 $(8,5)$ 、 $(9,5)$ , 在坐标系中绘制得到的散点图如图 4.11 所示, 看上去似乎并不是一条连续的直线,

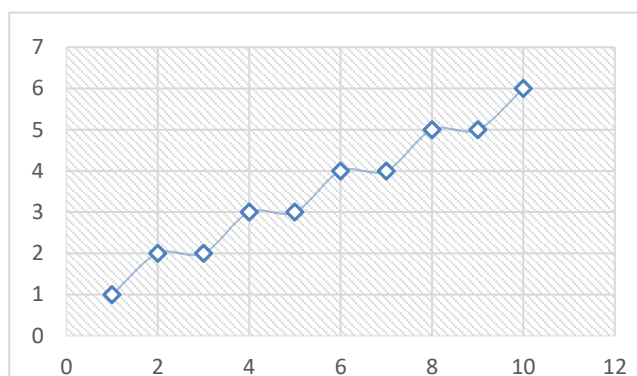


图 4.11 示例数据插入连续点后的散点图

但是在系统屏幕的坐标系中，坐标系变化往往是以整数值变化的，一般不存在小数的情况，也就是说屏幕坐标系中 0 到 1, 1 到 2 的变化是看做连续的，因此上述计算的结果在屏幕坐标系中是可以看成一条连续的直线的。

#### 4.4.2.2 手势轨迹稀疏化

对模板数据中离散的轨迹点进行连续化补全，这样取得数据量虽然整体而言比较精确，但是实际上在模式匹配的过程中，并不一定需要连续的坐标点数据量，由于基于 DTW 算法的模板识别是一种模糊识别算法，只需要计算实际时间序列与模板时间序列之间的相似度就可以了，因此一种比较可行的优化方法就是对获取到的连续轨迹点进行稀疏化，在连续的轨迹点中每隔一段距离后取值，取连续轨迹点中的部分轨迹点。对连续化的轨迹点进行稀疏化，然后存储在模板数据库的有二个好处：

第一、可以保证准确识别手势的同时又能节省模板数据库的存储空间：由于 xml 模板数据中定义的手势轨迹是过于“稀疏”（或者说过于离散）的，而对这种手势轨迹点进行连续化后的手势轨迹点又是过于“密集”（或者说过于连续）的，因此实际上对连续化后的手势轨迹点再次进行稀疏化，是一个对前两种情况一种折中的选择，在保证可以比较准确识别到手势的同时，又可以在存储这些模板数据时可以节省存储空间。

第二、可以节省手势识别所用的时间：由于 DTW 算法是一个计算量比较大的算法，因此对于连续化的轨迹坐标点信息，大量的模板数据的计算会占据比较长的算法

执行时间,为了提高手势识别的效率,将带有大量模板数据的手势轨迹坐标点进行稀疏化后,带来的手势识别效率的提高还是比较可观的。

对于手势轨迹稀疏化的处理过程如下:

(1) 对于某一手势的某一条轨迹  $P\{p_1, p_2, \dots, p_n\}$ , 初始化两个变量  $p$  和  $q$ , 以及一条新的轨迹  $Q\{\}$ , 令  $p=q=p_1$ , 将  $Q$  清空;

(2) 设定一个距离阈值  $d$ , 用  $p$  和  $q$  遍历这条轨迹,  $q$  向前遍历, 每遍历一次则比较一下点  $p$  和点  $q$  之间的距离, 用  $\text{dis}(p, q)$  保存计算的结果;

(3) 若  $\text{dis}(p, q) \geq d$ , 则将点  $p$  加入到轨迹点集合  $Q$  中, 然后令  $p=q$ , 重复步骤(2), 否则  $q$  继续遍历, 这样直到  $p=p_n$  为止。

上述处理过程中, 稀疏化手势轨迹比较关键的一点就是设置距离阈值  $d$ , 这个数值的设置会对算法识别效果有影响。有上述过程可知, 如果  $d$  越大, 手势轨迹的稀疏程度就越高, 那么会对手势识别的精确度有影响, 而如果  $d$  越小, 手势轨迹的密集度程度就越高, 那么手势识别算法执行过程所占据的时间就越长。经测试, 在本手势识别系统中  $d$  值设置为 50 时, 手势识别取得的效果会最好。

下面是对于手势库中三角形(用“ $\triangle$ ”标识该手势)手势的进行手势轨迹处理后的结果。整个过程如图 4.12 所示, 在 xml 模板文件中定义的三角形手势只给了四个点(起点和终点算两个点, 它们重合了)如图 4.12-(a)所示, 经过手势轨迹连续化处理如图 4.12-(b)所示, 最后根据连续化处理后的轨迹点再次进行距离阈值为 50 的稀疏化后, 得到的手势轨迹图如图 4.12-(c)所示。

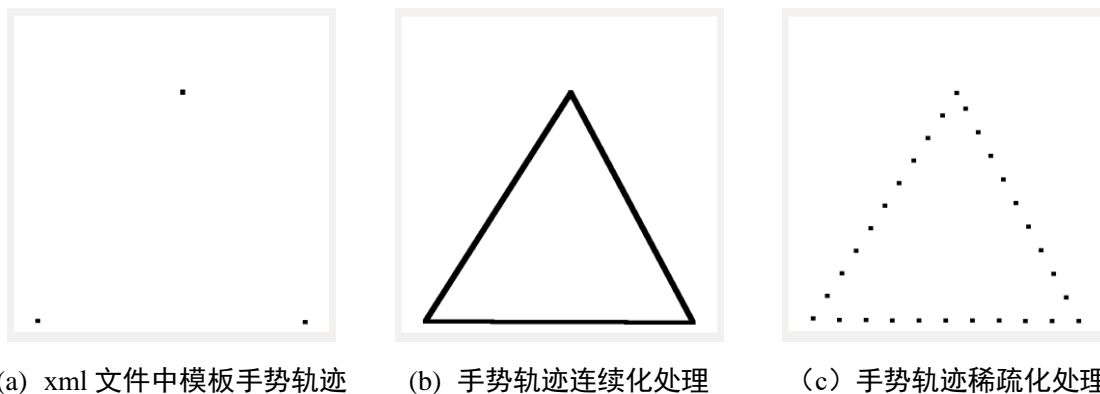


图 4.12 三角形手势轨迹处理过程示意图

#### 4.4.3 手势识别的过程

经过建立手势库模板文件，并对手势库中手势轨迹进行处理之后，接下来就是使用通过 Kinect 获取到手势轨迹，和手势库中的模板数据进行匹配了。由于获取的手势轨迹数据是相对于深度图的坐标系而言的，因此在进行手势匹配之前，首先要对获取到手势轨迹进行归一化处理。具体步骤如下：

(1) 坐标系转换：这一点在前文获取手势轨迹这一小节中已经说明，采取获取到的手势轨迹坐标点在  $x$  和  $y$  方向上的变化量参数进行坐标转换的方法；

(2) 手势轨迹区域映射：捕获手势轨迹点所在的矩形区域，将矩形区域尽量映射到和模板手势库中的手势矩形区域相近，以便于获得更加精确的识别结果；

(3) 手势轨迹的处理：由于获取到的手势轨迹可能并不是连续的，因此需要对获取的手势轨迹进行连续化处理，为了节省识别耗时，需要对连续化处理后的手势轨迹进行稀疏化处理，这个处理过程和对模板手势库中的手势轨迹点的处理相同。

接下来就是利用 DTW 算法进行模板匹配了，手势识别的流程图如图 4.13 所示，



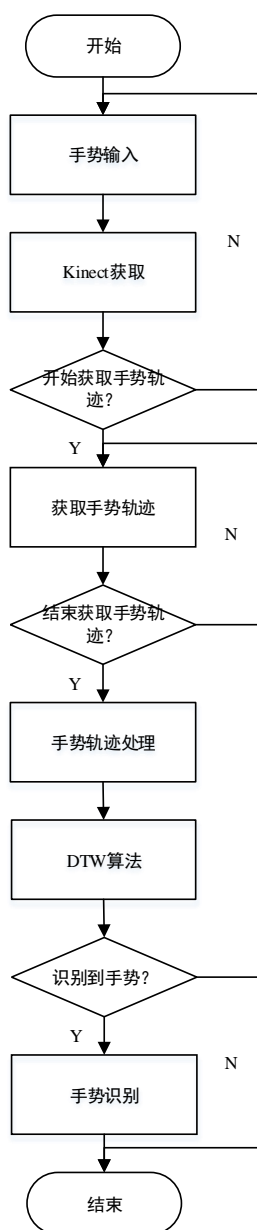


图 4.13 手势识别的流程图

这个流程中，有一个比较关键的点就是捕获手势轨迹的开始时机和结束时机的确定：只要手掌输入被 Kinect 检测和跟踪到，就会一直获取到手势数据，而根据建立起来的手势 FSM，系统会根据手掌的数值参数特征对 Kinect 获取的手掌数据进行状态划分，由于这些状态是比较容易确定的，所以在本文中，捕获手势轨迹的开始时

机是手掌处于移动（HandMove）状态，结束时机是手掌处于握拳（Grab）状态，就结束捕获手势轨迹点，如图 4.14 所示。

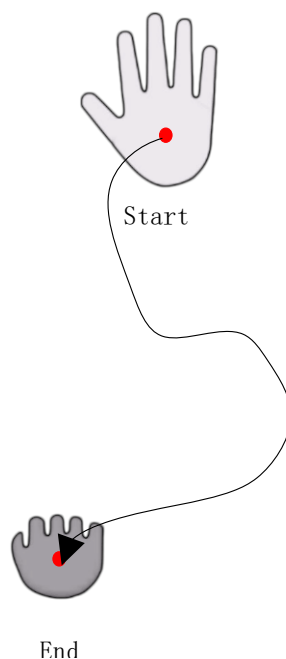


图 4.14 确立手势轨迹捕获的开始状态和结束状态示意图

## 4.5 本章小结

本章说明了整个手势识别系统的实现过程，首先阐述获取手势轨迹点的过程：说明对于获取到的手掌轨迹坐标点信息的转换和数据封装，接下来从数值特征方面对手势信息进行手势的状态划分，并设计了一套手势状态机，根据状态转换实现一些比较简单的手势识别过程，然后说明了 DTW 算法的核心技术，并介绍系统在 DTW 算法上的实现，从模板手势库的建立、模板轨迹处理、手势匹配这几个方面阐述了整个动态手势识别的核心部分。

## 5 系统测试和应用

第 3 章和第 4 章主要从手势图像数据获取和手势识别模块介绍了本文对于手势识别系统的实现,从算法理论和软件开发的角进行了详细说明,本章主要阐述系统框架搭建搭建和总体设计模式,以及对于手势识别系统的测试和部分实践应用到的场景。

### 5.1 图像处理模块测试

手势图像处理模块主要包括三个部分:

- (1) 手掌分割: 利用骨骼信息和深度信息进行手掌分割,将深度图像中的手掌部分分离处理;
- (2) 手掌轮廓获取: 根据分割出来的手掌部分计算手掌轮廓;
- (3) 寻找手掌心点: 在获取到轮廓的基础上计算手掌心点。

本文采用深度阈值的方法对手势进行了分割,并提取出手掌轮廓,通过实验,在距离摄像头约为 0.7m 的情况下,取不同的阈值  $k$  进行的测试,分割出来的手掌区域图像结果如图 5.1 所示

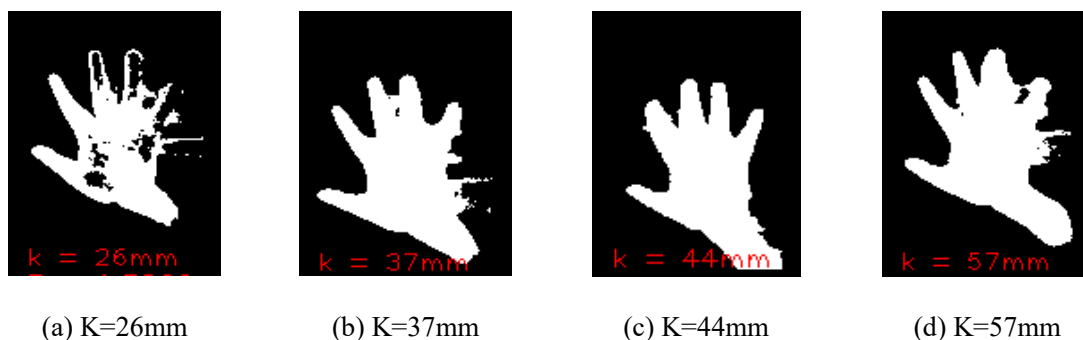


图 5.1 阈值  $k$  变化影响手部分割

根据大量测试结果可知,当阈值  $k$  取 40~45mm 之间的时候,可以分割出最清楚的手部区域,在稳定获取手部区域的情况下,对阈值  $k$  取 40mm,距离摄像头参数  $z$  取 0.82m 的情况是,可以比较稳定清晰的获取到手掌轮廓,测试结果如图 5.2 所示。



图 5.2 稳定获取手掌轮廓测试图

在稳定获取到手掌轮廓数据的情况下，然后进行手掌心点计算，较好的测试结果如图 5.3 所示。

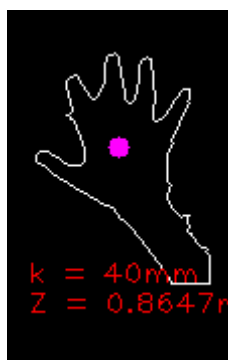


图 5.3 稳定获取手掌心点测试图

## 5.2 手势识别模块测试

获取手掌心点位置后，结合深度图数据可以比较有效的获取掌心点在三维空间中的坐标值，经过一系列的数值计算，坐标转换，可以对几种基本手势进行测试。

### 5.2.1 FSM 四种基本手势测试

基于手势状态转换机的手势一共有四种情况：手掌点击、手掌挥动、双手缩放和双手旋转；本文中计算测试手势识别率的方式是经过多次采样然后对正确识别手势的情况次数进行统计计算，假设手势识别测试数目总次数是  $S$ ，正确识别手势的情况次数是  $R$ ，那么手势正确识别率  $\eta$  的计算公式如公式(5.1)所示。

$$\eta = \frac{R}{S} \times 100\% \quad (5.1)$$

在稳定获取手掌三维坐标数据并多次采样统计的情况下,对于四种基本手势,分别进行了 30 次单独测试,对于正确识别手势的情况进行了统计,具体情况入表 5.1 所示

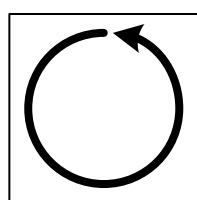
表 5.1 基本手势识别正确率测试

手势名称	总测试次数 (S)	正确识别数 (R)	正确识别率 ( $\eta$ )
手掌点击	30	25	83.3%
手掌挥动	30	27	90%
双手缩放	30	21	70%
双手旋转	30	22	73.3%

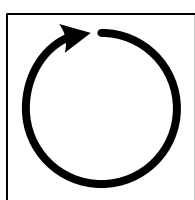
上述测试说明这种手势识别的情况,在双手缩放和双手旋转这些双手手势识别上不是特别灵敏,对于数值特征提取上需要比较合理的改进。

### 5.2.2 基于 DTW 的手势识别测试

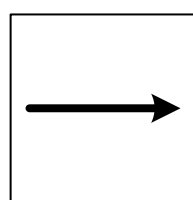
DTW 算法是一种基于模板匹配的算法,因此首先第一点就是要对模板数据进行定义,这个过程可以通过训练的方式,也可以自己定义手势轨迹,本文中采用的是第二种方法,考察人们生活中比较常用的几种手势动作,在模板手势库中一共定义了 12 种手势轨迹,具体如图 5.4 模板手势库中定义的 12 种动态手势轨迹所示。



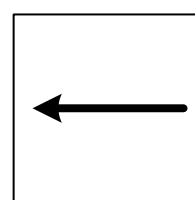
(a) 逆时针圆



(b) 顺时针圆



(c) 向右



(d) 向左

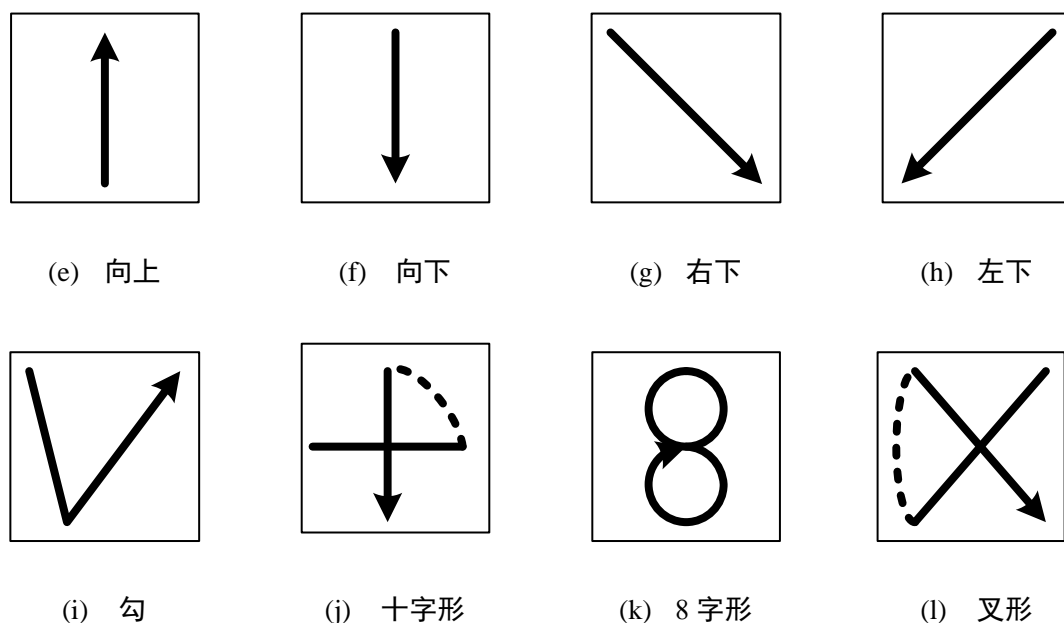


图 5.4 模板手势库中定义的 12 种动态手势轨迹

为方便观察到手势轨迹，本文用 Qt 图形化接口设计了一个可视化的获取轨迹并进行识别的程序，其界面如图 5.5 所示：



图 5.5 DTW 手势识别系统界面

以识别 8 字形的手势为例，利用 Kinect 获取到手势轨迹如图 5.6 所示

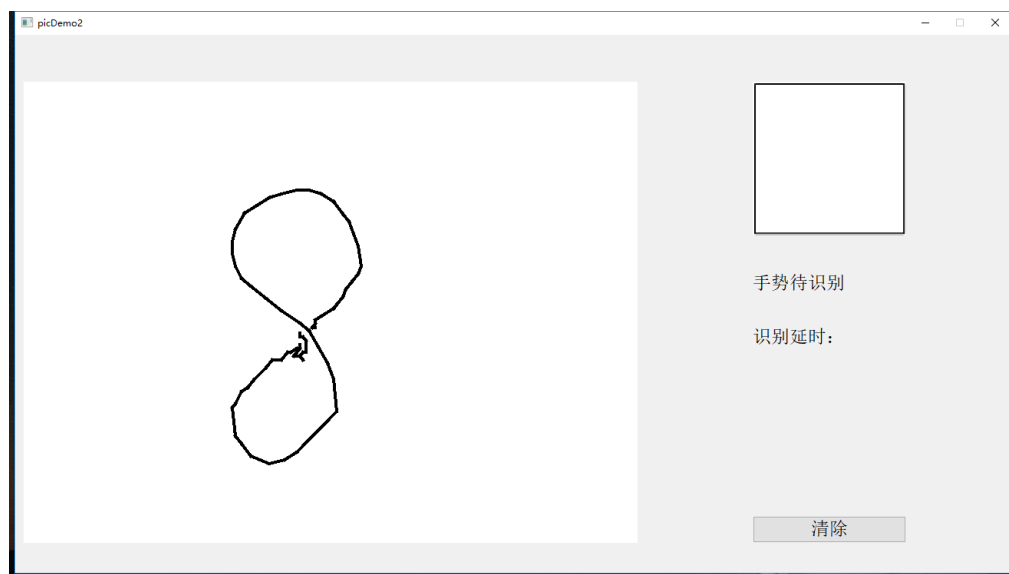


图 5.6 8 字形手势轨迹获取

经过 DTW 算法，其识别结果如图 5.7 所示

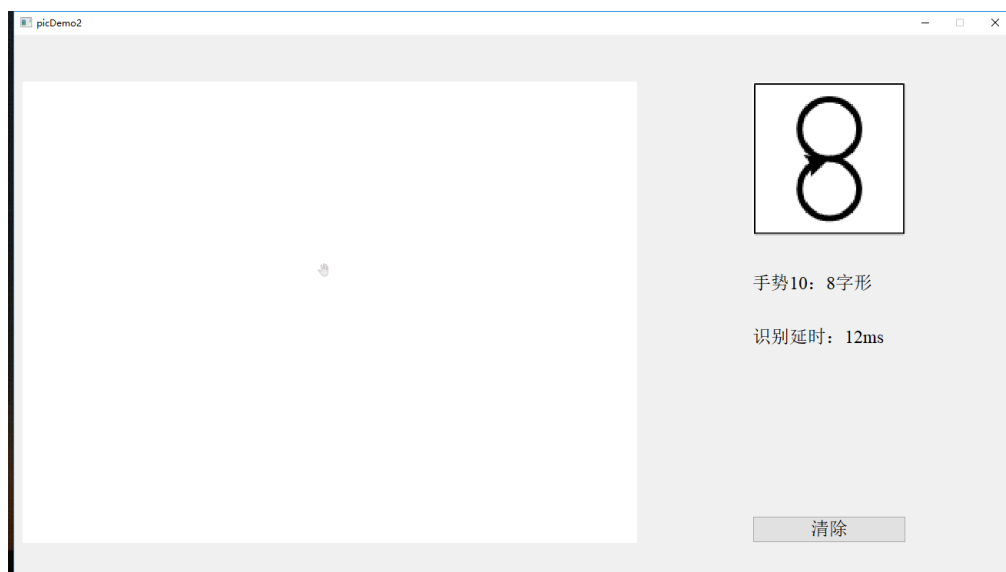


























图 5.7 8 字形手势的识别结果

在稳定获取手掌轨迹坐标数据的情况下，本文中对基于 DTW 算法的每一种手势轨迹进行了 30 次单独试验测试，对动态手势识别测试中，根据手势动作的预测值和观察值绘制了混淆矩阵，如表 5.2 所示，可知一些比较复杂的手势比如 8 字形或者

逆时针圆等，其识别率能达到 100%，而相对于一些比较简单且容易重复的手势，其识别率比较低。

表 5.2 基于 DTW 手势识别预测值和观察值构成的混淆矩阵

预 观												
	30	0	0	0	0	0	0	0	0	0	0	0
	0	30	0	0	0	0	0	0	0	0	0	0
	0	0	28	0	0	0	1	0	1	0	0	0
	0	0	0	28	0	0	0	2	0	0	0	0
	0	0	0	0	29	0	0	0	1	0	0	0
	0	0	0	0	0	28	0	1	1	0	0	0
	0	0	0	0	0	1	27	0	2	0	0	0
	0	0	0	1	0	0	0	29	0	0	0	0
	0	0	0	0	0	0	0	0	30	0	0	0
	0	0	0	0	0	0	0	0	0	30	0	0
	0	0	0	0	0	0	0	0	0	0	30	0
	0	0	0	0	0	0	0	1	0	0	0	29

分类算法的几个测试指标值，对于某一测试中符合预测的结果称为正例，而预测之外的结果称为负例，则有以下几个关键性的概念<sup>[46]</sup>：

- (1) True positives (TP)：分类算法中被正确划分，且为正例的个数；
- (2) False positives (FP)：分类算法中被错误划分，且为正例的个数；
- (3) False negatives (FN)：分类算法中被错误划分，且为负例的个数；
- (4) True negatives (TN)：分类算法中被正确划分，且为负例的个数。



(5) Positives(P): 分类算法中被划分为正例的总个数;

(6) Negatives(N): 分类算法中被划分为负例的总个数。

对于上述几个关键性的测试量, 有下列几种主要的评价指标<sup>[47]</sup>:

(1) 准确度 (Accuracy): 正确分类的个数的比例, 其计算公式是:

$$\text{Accuracy} = \frac{TP + TN}{P + N} \times 100\%$$

(2) 查全率 (Recall): 正确被划分为正例的比例, 其计算公式是:

$$\text{Recall} = \frac{TP}{TP + FN} \times 100\%$$

(3) 精度 (Precision): 预测被分为正例的示例中实际为正例的比例, 其计算公式是:







$$\text{Precision} = \frac{TP}{TP + FP} \times 100\%$$







(4) F1-测量 (F1-measure): 综合查全率和精度计算参数值为 1 时的加权调和平均, 其计算公式是:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

结合上述绘制的混淆矩阵, 可以对这 12 种手势得出如下表的分类测试分析结果如表 5.3 所示。

表 5.3 各种手势识别的分类测试结果分析情况

	TP	FP	FN	TN	P	N	Accuracy	Recall	Precision	F1
	30	0	0	330	30	330	100%	100%	100%	100%
	30	0	0	330	30	330	100%	100%	100%	100%
	28	0	2	330	28	332	99.4%	93.3%	100%	96.5%
	28	1	2	329	29	331	99.2%	93.3%	96.6%	94.9%
	29	0	1	330	29	331	99.7%	96.7%	100%	98.3%
	28	1	2	329	29	331	99.2%	93.3%	96.6%	94.9%

	27	1	3	329	28	332	98.9%	90%	96.4%	93.1%
	29	4	0	327	33	327	98.9%	100%	87.9%	93.6%
	30	4	0	326	34	326	98.9%	100%	88.2%	96.8%
	30	0	0	330	30	330	100%	100%	100%	100%
	30	0	0	330	30	330	100%	100%	100%	100%
	29	0	1	330	29	331	99.7%	96.7%	100%	98.3%

根据上述的结果,可以很直观的分析出,在保证系统的识别率的情况下,能比较稳定确保分类的召回率。且也能保证手势识别分类的精度在 87%以上,在综合考虑召回率和精度的情况下,计算所得的 F1 值也能保证在 93.3%之上,充分表现了基于 DTW 算法的手势识别的可靠性。

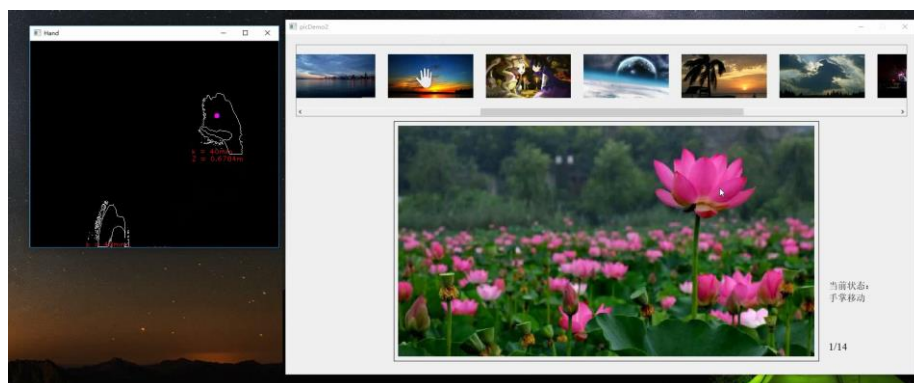
### 5.3 手势识别应用场景

在前面获取到手势识别结果的基础上,本文实现了一个基于动态手势识别的图像浏览器,将相应的手势识别结果应用到了实际中。手势识别结果和图片浏览器功能之间的对照关系如表 5.4 所示。

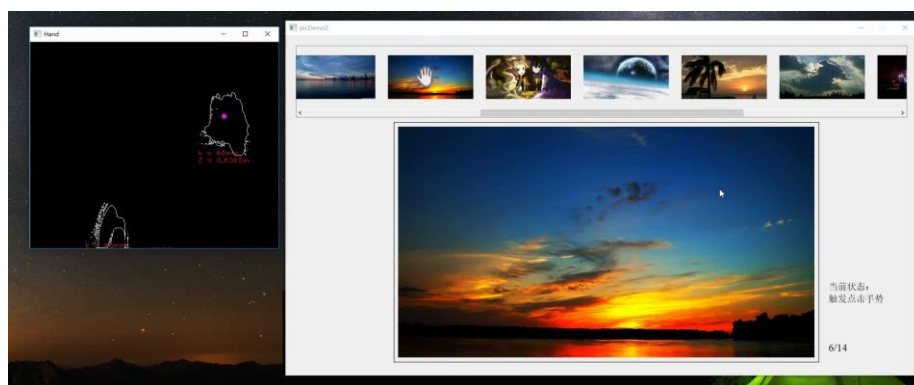
表 5.4 手势识别结果和图片浏览器功能对照表

基本手势	相应功能
手掌点击	选择图片
手掌挥动	图片翻页
双手缩放	图片缩放
双手旋转	图片旋转

1.手掌点击的手势识别应用:手掌处于停留状态后,在图片列表上点击需要选中的图片,然后在预览界面进行显示,如图 5.8 点击手势实现图片选择的效果图所示,通过点击手势实现选择第 6 张图片的功能。



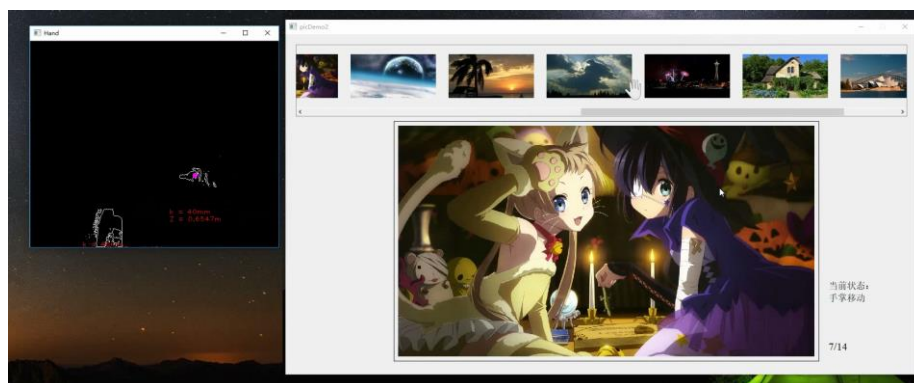
(a) 点击手势触发前



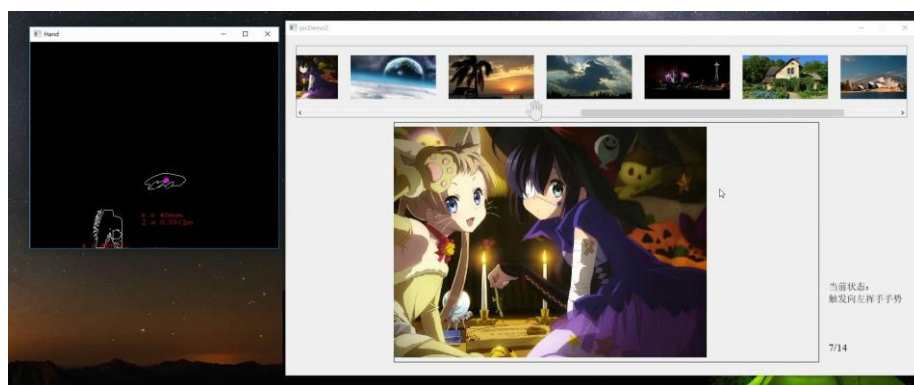
(b) 点击手势触发后

图 5.8 点击手势实现图片选择的效果图

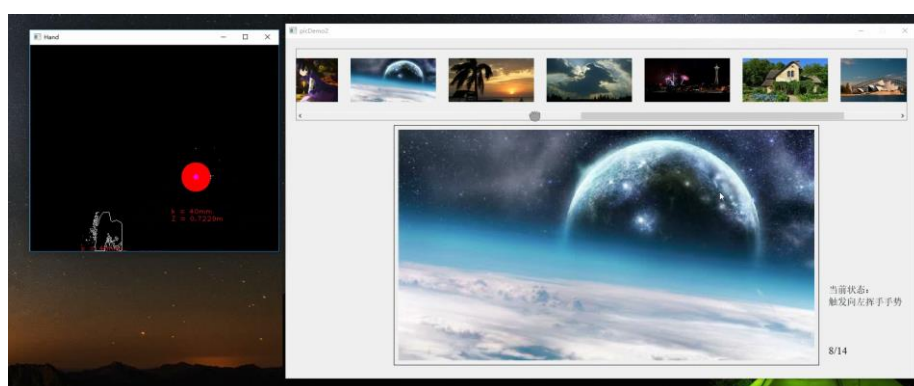
2.手掌挥动的手势识别应用: 手掌向左挥动实现图片的向后翻页, 右挥动实现图片的向前翻页显示效果, 如图 5.9 向左挥手实现向后翻页效果图所示, 实现了向左挥手手势完成第 7 张图片到底 8 张图片向后翻页的效果。



(a) 向左挥手动作触发前



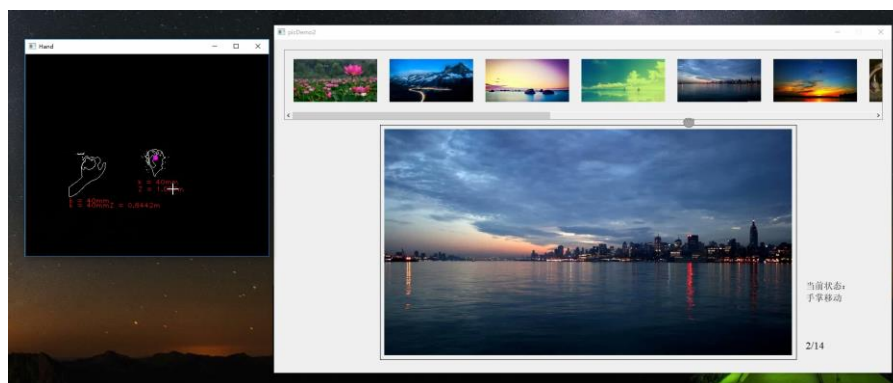
(b) 向左挥手翻页动画过程中



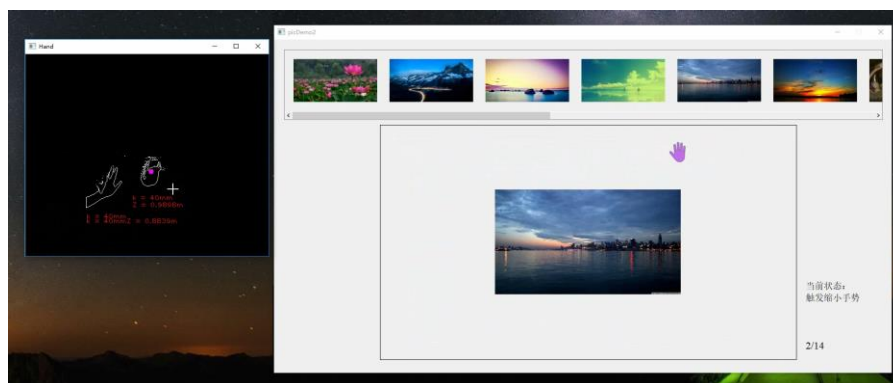
(c) 翻页动画完成

图 5.9 向左挥手实现向后翻页效果图

3. 双手缩放手势的应用：双手相向运动实现图片缩小显示的功能，双手背向运动实现图片放大显示的功能，图片缩小显示的效果图如图 5.10 缩小手势实现图片缩小功能效果图所示。



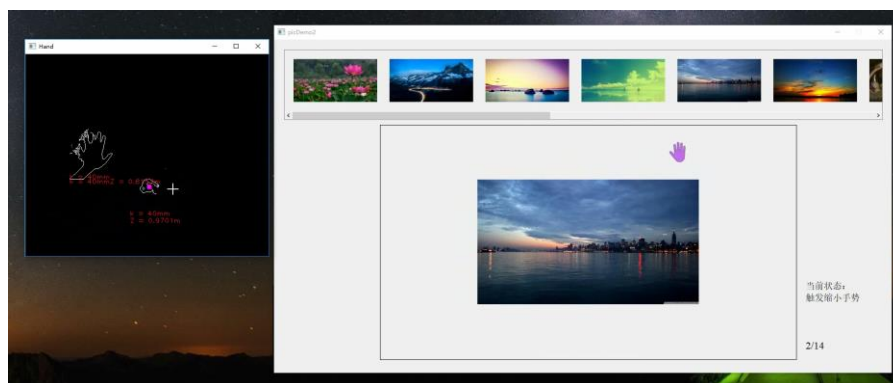
(a) 缩小手势触发前



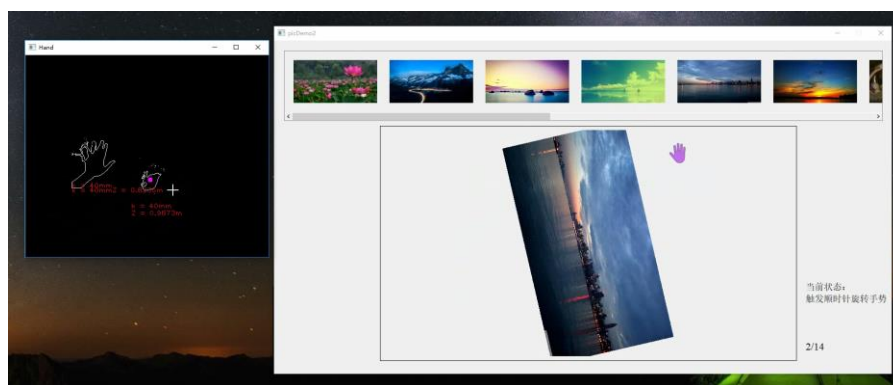
(b) 缩小手势触发后

图 5.10 缩小手势实现图片缩小功能效果图

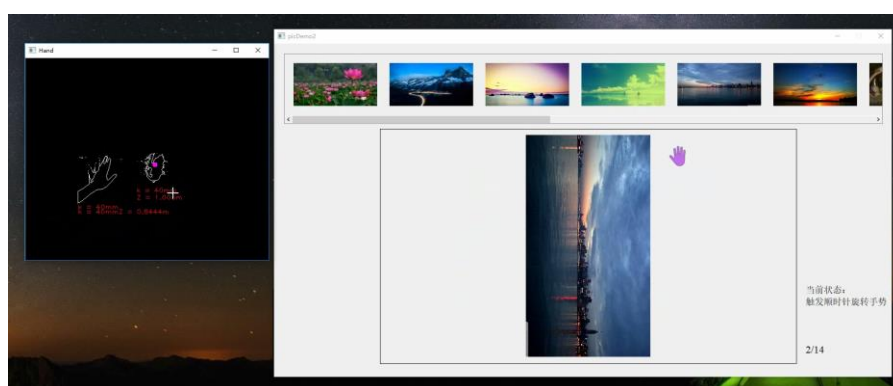
4. 双手旋转手势的运用：双手顺时针方向旋转实现图片的顺时针方向旋转，逆时针方向旋转实现图片的逆时针方向旋转，顺时针旋转图片的效果如图 5.11 旋转手势实现图片旋转的效果图所示。



(a) 旋转手势触发前



(b) 旋转手势触发后旋转过程中



(c) 旋转动画完成

图 5.11 旋转手势实现图片旋转的效果图

综合以上对于图片浏览器的各个功能的测试，这四种手势能完成图片浏览器的这几种基本的功能。这也体现了本文中手势识别系统的实际意义所在。

## 5.4 本章小结

本章主要从程序的框架设计角度介绍了系统实现，首先给出了开发环境和系统搭建过程，然后说明了手势识别系统的模块设计和层次设计，对于手势的框架搭建做了详细的阐述；然后接下来分别对图像处理模块和手势识别模块做了具体的测试，根据测试的结果对最终识别的正确率进行了分析，并给出了定性的结果；最后对于手势识别具体应用场景进行了测试，说明该手势识别系统应用到实际的价值和意义。

## 6 总结与展望

### 6.1 总结

手势是人机交互中比较直接和自然的方式之一，在近年来被研究者作为人机交互领域的重要的方向，做了大量与之相关的研究内容。手势识别作为计算机视觉领域里重要的一部分，也随着计算机技术的发展，不断地有着新的突破。与基于接触类设备的手势识别相比，基于视觉的手势识别体现了手势这种人机交互方式自由的特性，而 Kinect 使得基于视觉的手势识别对于手势的获取变得更加方便，因此，本文在利用 Kinect 获取手势数据的同时，完成了以下内容：

1. 分析当前手势识别研究行业的现状，从基于数据手套、双目视觉和 Kinect 三个方面对前人研究进行归纳总结，分析本文课题的内容和意义。
2. 实现了一个基于有限状态机的手势识别系统，根据手掌运动的数值特征对手掌的状态进行划分，在手势状态的转化过程中实现了对手掌点击，手掌挥动，双手缩放和双手旋转四种基本手势的识别。
3. 从手势的有限状态机出发，从手掌张开状态到握拳状态之间，捕获分割的手势坐标点的轨迹信息，在定义好手势并建立动态模板手势库的情况下，对轨迹点信息进行处理，然后使用 DTW 算法对手势轨迹进行特征点模板匹配，计算手势轨迹序列和模板手势库的手势轨迹点之间的相似度，对模板手势库定义的几种特定的动态手势进行了识别，然后给出了测试结果。
4. 在对上述动态手势识别的基础上构建手势识别的接口，并利用手势识别接口实现了一个图片浏览器，完成了利用手势对图片进行翻页、选中、旋转和缩放等基本功能，验证了这种手势识别接口对应用到实际场景中的意义和价值。

### 6.2 改进和展望

本文的研究主要是基于 Kinect 设备的手势识别系统，在实际测试阶段，可以验证几种手势能比较精确地被识别到，并从实际应用场景出发，测试了这个手势识别系



统的实践意义，但这项研究工作依然存在着许多的不足和亟需改进的地方，主要如下：

1.双手手势识别的性能：双手连续手势的稳定性不是很高，由于双手的手势是一段连续性的过程，因此对于手势起点和手势终点判定的要求比较高。而在实际中测试的时候发现缩放和旋转手势往往在手势未完成之前就结束，且手势过程采用的数值特征参数波动性比较大，不能稳定的表示缩放率和旋转率，无法完成实时的缩放和旋转，因此下一步的工作会使用更好的手势识别算法对双手手势进行判别。

2.DTW 手势的识别时间：在基于 DTW 算法手势识别的过程中，对于特定手势的识别耗时比较高，这是由于 DTW 算法的时间复杂度比较高的缘故。因此下一步的工作将采用改进后的 DTW 算法对手势进行识别，缩短手势识别耗时。

## 致谢

时光荏苒，岁月匆匆，两年的研究生时光转瞬即逝。回首这两年的时光，我不仅学会了很多知识，还成熟了许多。在我的毕业论文即将完成之际，我要衷心地感谢很多人。

首先，我要感谢我的导师张杰老师在学习和生活中给予我的帮助。张老师严谨踏实的学术研究态度以及丰富博学的计算机知识水平，都让我佩服不已。从开题、项目的最终完成到最后论文的撰写，张杰老师都给予了莫大的帮助和不懈的支持；生活中，张老师谦虚随和，严以律己宽以待人的为人处世方式也不断影响和感染着我，教会我做人的道理。

然后，我要感谢实验室所有的老师，所有的师兄师姐的耐心指教和热心帮助，感谢我们这一届的所有同学的真诚付出，是我在学习上感到温暖，在生活上感到快乐。是你们给了我日常生活中的诸多乐趣，给了我一个像家一样温馨的实验室，同时也收获了珍贵的友谊。

书到用时方恨少，在撰写论文的过程中，我发现自己知识量还远远不够，然而正所谓不积跬步无以至千里，在今后的日子里我会不断地充实自己，完善自己。

## 参考文献

- [1] 冯志全, 蒋彦. 手势识别研究综述. 济南大学学报 (自然科学版), 2013, 4: 336-341
- [2] 任海兵, 祝远新. 基于视觉手势识别的研究—综述. 电子学报, 2000, 2: 118-121
- [3] 王松林. 基于 Kinect 的手势识别与机器人控制技术研究:[硕士学位论文] 北京交通大学, 2014.
- [4] Defanti T, Sandin D. Final Report to the National Endowment of the Arts. US NEA R60-34-163, University of Illinois at Chicago Circle, Chicago, Illinois, 1977,
- [5] Kim J H, Thang N D, Kim T S. 3-D Hand Motion Tracking and Gesture Recognition Using a Data Glove. In, ed. IEEE International Symposium on Industrial Electronics. 2009. 1013-1018
- [6] Weissmann J, Salomon R. Gesture Recognition for Virtual Reality Applications Using Data Gloves and Neural Networks. In, ed. Neural Networks, 1999. IJCNN'99. International Joint Conference on. IEEE, 1999. 2043-2046
- [7] Quam D L. Gesture Recognition with a Dataglove. In, ed. Aerospace and Electronics Conference, 1990. NAECON 1990., Proceedings of the IEEE 1990 National. IEEE, 1990. 755-760
- [8] Park I, Kim J, Hong K. An Implementation of an FPGA-based Embedded Gesture Recognizer Using a Data Glove. In, ed. Proceedings of the 2nd international conference on Ubiquitous information management and communication. ACM, 2008. 496-500
- [9] Argyros A A, Lourakis M I. Binocular Hand Tracking and Reconstruction Based On 2D Shape Matching. In, ed. Pattern Recognition, 2006. ICPR 2006. 18th International Conference on. IEEE, 2006. 207-210

- [10]Utsumi A, Miyasato T, Kishino F. Multi-Camera Hand Pose Recognition System Using Skeleton Image. In, ed. Robot and Human Communication, 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE International Workshop on. IEEE, 1995. 219-224
- [11]郭铎. 基于双目视觉的手势识别算法研究与实现:[硕士学位论文]. 东北大学, 2011.
- [12]Segen J, Kumar S. Video-Based Gesture Interface to Interactive Movies. In, ed. Proceedings of the sixth ACM international conference on Multimedia: Technologies for interactive movies. ACM, 1998. 39-42
- [13]Malik S, Laszlo J. Visual Touchpad: A Two-Handed Gestural Input Device. In, ed. Proceedings of the 6th international conference on Multimodal interfaces. ACM, 2004. 289-296
- [14]郭康德. 基于视觉的三维指尖检测算法和应用:[硕士学位论文]. 浙江大学计算机学院 浙江大学, 2010.
- [15]谭同德, 郭志敏. 基于双目视觉的人手定位与手势识别系统研究. 计算机工程与设计, 2012, 1: 259-264
- [16]何云龙. 基于双目立体视觉的手势识别技术:[硕士学位论文]. 华中科技大学, 2013.
- [17]Ren Z, Yuan J, Meng J et al. Robust Part-Based Hand Gesture Recognition Using Kinect Sensor. Ieee T Multimedia, 2013, 5: 1110-1120
- [18]Biswas K K, Basu S K. Gesture Recognition Using Microsoft Kinect®. In, ed. Automation, Robotics and Applications (ICARA), 2011 5th International Conference on. IEEE, 2011. 100-103
- [19]Kulshreshth A, Zorn C, Laviola J J. Poster: Real-Time Markerless Kinect Based Finger Tracking and Hand Gesture Recognition for HCI. In, ed. 3D User Interfaces (3DUI), 2013 IEEE Symposium on. IEEE, 2013. 187-188

- [20] Pedersoli F, Benini S, Adami N et al. XKin: An Open Source Framework for Hand Pose and Gesture Recognition Using Kinect. *The Visual Computer*, 2014, 10: 1107-1122
- [21] Soltani F, Eskandari F, Golestan S. Developing a Gesture-Based Game for Deaf/Mute People Using Microsoft Kinect. In, ed. *Complex, Intelligent and Software Intensive Systems (CISIS)*, 2012 Sixth International Conference on. IEEE, 2012. 491-495
- [22] 钱鹤庆. 应用 Kinect 与手势识别的增强现实教育辅助系统:[硕士学位论文]. 上海交通大学, 2011.
- [23] 李小龙. 基于 Kinect 手势识别的虚拟人体解剖教学系统的设计与实现:[硕士学位论文]. 北京工业大学, 2014.
- [24] Wang C, Liu Z, Chan S. Supapixel-Based Hand Gesture Recognition with Kinect Depth Camera. *Ieee T Multimedia*, 2015, 1: 29-39
- [25] 张毅, 张烁, 罗元等. 基于 Kinect 深度图像信息的手势轨迹识别及应用. *计算机应用研究*, 2012, 9: 3547-3550
- [26] 石曼银. Kinect 技术与工作原理的研究. *哈尔滨师范大学自然科学学报*, 2013, 03: 83-86
- [27] Roy A K, Soni Y, Dubey S. Enhancing Effectiveness of Motor Rehabilitation Using Kinect Motion Sensing Technology. 第 2013. 298-304
- [28] Albitar C, Graebbling P, Doignon C. Robust Structured Light Coding for 3D Reconstruction. In, ed. *IEEE International Conference on Computer Vision*. 2007. 1-6
- [29] Wiley W C, McLaren I H. Time - of - Flight Mass Spectrometer with Improved Resolution. *Rev Sci Instrum*, 1955, 12: 1150-1157
- [30] 李畅. Kinect 设备功能探索. *数字通信世界*, 2017, 8: 37-38

- [31]Terrillon J C. Comparative Performance of Different Chrominance Spaces for Color Segmentation and Detection of Human Faces in Complex Scene Images. 2000, 180-187
- [32]陈锻生, 刘政凯. 肤色检测技术综述. 计算机学报, 2006, 2: 194-207
- [33]Hsu R, Abdel-Mottaleb M, Jain A K. Face Detection in Color Images. Ieee T Pattern Anal, 2002, 5: 696-706
- [34]张争珍, 石跃祥. YCgCr 颜色空间的肤色聚类人脸检测法. 计算机工程与应用, 2009, 22: 163-165
- [35]余旭. 基于 Kinect 传感器的动态手势识别:[硕士学位论文]. 西南大学, 2014.
- [36]陈保, 苏顺开. 基于 Kinect 的手势识别与阈值法. 数字技术与应用, 2016, 7: 56-57
- [37]孙丽娟, 张立材, 郭彩龙. 基于视觉的手势识别技术. 计算机技术与发展, 2008, 10: 214-216, 221
- [38]Schlenzig J, Hunter E, Jain R. Recursive Identification of Gesture Inputs Using Hidden Markov Models. In, ed. Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on. IEEE, 1994. 187-194
- [39]江立, 阮秋琦. 基于神经网络的手势识别技术研究. 北京交通大学学报, 2006, 5: 32-36
- [40]陈静. 基于 Kinect 的手势识别技术及其在教学中的应用:[硕士学位论文]. 上海交通大学, 2013.
- [41]林水强, 吴亚东, 余芳等. 姿势序列有限状态机动作识别方法. 计算机辅助设计与图形学学报, 2014, 9: 1403-1411
- [42]Zhang Z. Microsoft Kinect Sensor and its Effect. Ieee Multimedia, 2012, 2: 4-10
- [43]徐小良, 汪乐宇, 周泓. 有限状态机的一种实现框架. 工程设计学报, 2003, 05: 251-255
- [44]童红. 孤立词语音识别系统的技术研究:[硕士学位论文]. 江苏大学, 2009.

- [45]张闯, 王婷婷, 孙冬娇等. 基于欧氏距离图的图像边缘检测. 中国图象图形学报, 2013, 2: 176-183
- [46]宋枫溪, 高林. 文本分类器性能评估指标. 计算机工程, 2004, 13: 107-109
- [47]Sebastiani F. Machine Learning in Automated Text Categorization. ACM, 2002. 1-47