基于多分类器的三维姿态识别算法的研究

Study on 3d action recognition algorithm based on multiple classifiers

计算机科学与技术学院

秦亚伦 PB09210342

王明会 教授

二〇一三年五月

中国科学技术大学

University of Science and Technology of China

本科毕业论文

A Dissertation for the Bachelor's Degree

基于多分类器的三维姿态识别算法的研究

Study on 3d action recognition algorithm based on multiple classifiers

姓　　名	秦亚伦
B.S. Candidate	Yalun Qin
导　　师	王明会 教授
Supervisor	Prof.Minghui Wang

二〇一三年五月

May,2013

中国科学技术大学

学士学位论文

题　　目	基于多分类器的 三维姿态识别算法的研究
院　　系	计算机科学与技术学院
姓　　名	秦亚伦
学　　号	PB09210342
导　　师	王明会 教授

二〇一三年五月

University of Science and Technology of China

A Dissertation for the Bachelor's Degree

Study on 3d action recognition algorithm based on multiple classifiers

B.S. Candidate Yalun Qin

Supervisor Prof.Minghui Wang

Hefei, Anhui 230026, China

May,2013

致　　谢

时光飞逝，转眼间在中国科学技术大学的本科生涯即将结束。通过这四年的学习，我最大的收获莫过于找到了自己真正的兴趣点 —计算机科学，获得了巨大的学习动力，在科大诸位任课老师的教导下，深入了解了这一专业领域，并且在课程实践中，诸如编程能力、自学能力、信息检索能力，逻辑思维能力等相关能力有了极大的提升。能够有这么多的收获，一方面是因为自己的兴趣与勤奋，另一方面，便要归功于学校提供的宁静舒适的学习环境，和科大所有老师尽职尽责，一丝不苟的教学态度。因此，值此毕业论文完成之际，我要首先感谢学校和老师，你们的辛勤付出让我可以在知识的草原上自由驰骋，你们的谆谆教诲让我更加清晰地认识了自己，从而做出正确的人生选择。

衷心感谢王明会老师在本次毕业设计项目中的指导。从最初的项目构想，实验器材的提供，到最后毕业论文的完成，王老师都给予很大帮助。项目开始时，王老师清晰地讲解了大致思路，并且提供了相关论文，使我很快地熟悉了该项目。在项目进行过程中，王老师以其严谨的科研态度和敏锐的眼光发现了项目的几处漏洞，并给出建设性建议，使我得以完善项目，并且更深入地体会了科学研究的感觉。在进度上，王老师也给过几次关键的建议与提醒，使得我可以如期完成任务。

感谢刘元宁等师兄在毕设过程中无私的帮助。项目执行中遇到很多琐碎的细节问题，刘元宁师兄都给予了耐心的解答，使得我可以顺利的完成起初看似艰巨如山的毕设任务。记得曾经遇到过一个关于 SVM 多分类的问题，困扰了我很长时间，是刘元宁和邹亮师兄的热心讨论给我最终的解决思路。

感谢张泽群，郭品垚等同学。和你们讨论毕设的过程中往往能使我产生各种新奇的 idea。感谢陪伴我一起走过这四年的同学。四年的学习生涯充斥着各种艰难险阻，我们一起奋斗，互相学习，互相帮助，一路走来，一起克服了无数困难。和你们讨论甚至争执学习问题的过程是最让我难忘的场景。这四年，我学到很多，不仅仅从书本上，还有很大一部分是从你们身上。感谢 0911 全体同学，你们当中的很多人，都会成为我学习的榜样！

感谢我的父母。你们在生活和学习上给予了我无限的理解与支持，不仅仅在这四年。你们永远是我最强大的后盾和最要感激的人。

最后，我还要感谢学校给我做这次毕业设计的机会。尽管这是个人人都会有的机会，但是对于我这个项目狂，我还是非常珍视它的。当初之所

以选这个项目，就是因为之前自学过机器学习方面的一点知识，感觉很有意思，想“学以致用”。这个项目本身也许并不是多么出彩，也许还存在着许多漏洞，但对于我来说，这是我个人做过的最大的项目，同时也是最完整的，理论、实践的内容均有，克服重重困难，最后终于完成，带来的成就感也是最大的。我会用心完成这次毕设，为这四年的学习生涯划个完美的句号！

目　录

致　谢 **i**

摘　要 **v**

Abstract **vii**

第一章　引言 **1**

第一节 课题意义 1

第二节 相关研究 2

一、 基于彩色图像的人体姿态识别 2

二、 基于 Kinect 的人体姿态识别 3

第二章　基于骨骼图的姿态识别 **5**

第一节 数据采集 6

第二节 特征提取 7

第三节 识别模型构建 8

一、 SVM 8

二、 KNN 10

第四节 姿态预测 11

第三章　实验结果与分析 **13**

第一节 混淆矩阵（Confusion Matrix) 分析 13

第二节 SVM 和 KNN 分类准确率比较 15

第三节 缓存帧数与识别准确率的关系 16

第四节 实验小结 17

第四章　开源库 ARLK 的构建 **19**

第一节 库的意义和设计目标 19

第二节 ARLK 类结构详解 20

一、 类名字空间设计 20
二、 SkeData 类介绍 22
三、 BaseClassifier 类介绍 25
四、 SVMClassifier 类介绍 27
五、 KNNClassifier 类介绍 29
第三节 ARLK 示例程序介绍 31
一、 DataCollection 31
二、 SkeletonPlayer 32
三、 ActionInteract 33
四、 HAR_App 34

第五章 总结 **39**

参考文献 **40**

附录 A 数据集规范 **43**

附录 B ARLK 使用指南 **47**
第一节 配置要求 47
第二节 静态库的生成与使用 47
第三节 示例程序的构建与运行 49
第四节 ARLK API 说明 50

摘　要

人体姿态识别因其在监督、控制等方面的广泛应用，一直是一个吸引众多研究者的领域。微软发布的 Kinect 摄像头，除了可以产生彩色图像数据，还可以产生包含更详尽人体位置信息的深度数据，为该领域带来了新的研究切入点。不久之后，诞生了可以从深度图追踪人体骨架的技术，而我们的工作首先便是依靠这一技术，从人体骨骼数据中提取特征。除了我们之前工作中提取的 9 个表示静态位置信息的特征向量外，我们又增加了四个表示关节移动速度的特征向量，从而可以表示一些简单的动作信息。为了测试，我们一共定义了 27 个姿态动作，包括 16 个静态姿势和 11 个简单动作，这些姿态数据构成了一个可以对外发布的数据集。我们用的分类算法是 SVM 和 KNN，利用 10 重交叉验证，整体平均识别准确率可分别达到 99.1% 和 99.0%，在人体姿态识别领域达到领先水平。

除了提出一种识别人体姿态的新方法，我们还进一步开发了一个开源库 (ARLK–Action Recognition Library for Kinect)，用来帮助广大研发人员更方便地借助 Kinect 骨骼数据进行有关人体姿态识别的研究或者实际应用程序的开发。这应该是迄今为止世界上唯一一个完成该功能的开源库，并且具备以下特点（后文会详述）：

1. 完全开源并且跨平台
2. 与 C++,C# 等主流编程语言兼容，易于使用
3. 支持 Kinect SDK 和 openNI SDK
4. 具有良好的易扩展性，用户可以方便增加新的分类器，及其他骨骼点规范
5. 有详尽的文档和功能强大的示例程序。示例程序还可充当有用的工具，进行数据采集与查看，实时或非实时地执行训练、预测等功能

关键字:　Kinect; 人体姿态识别; SVM; KNN; ARLK

Abstract

Human posture and action recognition has long been an important issue that attracts many researchers' attention, due to its vast application in surveillance and control, as well as the great challenges it poses. After Microsoft Kinect sensor was launched, a new perspective for solving the problem is provided , since besides RGB data, the sensor can also generate depth data, which contains much more information about actions. Further study discovered ways to infer skeleton information from the depth data, and we extracted features from the skeleton data. Besides the 9 features calculated in our prior work, we added another 4 features representing velocity of certain joints, in order to better represent the user-defined actions. In addition to the 16 postures, we defined 11 different actions, and the data of all the 27 motions composed our public dataset. The classification algorithms we used were SVM and KNN, and after evaluated through 10-fold cross validation, both algorithms showed an overall accuracy of 99.1% and 99.0%, which is quite a successful performance in HAR.

Besides promoting a new way to recognize postures and actions based on multiple classifiers, we further developed an open-source library(ARLK ——Action Recognition Library for Kinect) to help researchers and developers conveniently do research or make applications concerning recognizing human actions and postures using Kinect skeleton data. As the only library achieving this function effectively, our project has the following features which will be expanded later in this paper:

1. Open-source and cross-platform
2. Compatible with major programming languages like C++, C# and very easy to use.
3. Supporting Kinect SDK and openNI SDK
4. Easy to expand. Users can add more classifiers and other specifications of inferred skeleton data
5. Detailed documentation and powerful samples. The samples can also serve as useful tools to collect data, view data, run training and predicting process on files or in real time.

Keywords: Kinect; Human Action Recognition(HAR); SVM; KNN; ARLK

第一章　引言

第一节　课题意义

所谓人体姿态识别，就是利用各种数理统计、机器学习的方法让计算机判断其接收到的“姿态”数据属于哪一类，这一过程本质上其实就是分类。这里所说的“姿态”，包括人静止时的姿势（Posture) 和全身动作（Action). 由于没有一个英文单词可以涵盖这二者的意思，所以规定本论文中的“action” 即”姿态“之意。

人体姿态识别之所以是计算机视觉领域越来越热门的研究课题，主要还是因为它具有广泛的应用前景。

1. 智能监督系统，识别被监视者的异常举动。可以应用在医疗系统中，识别病人的异常举动并作出反应；或者在安全监控系统中识别各种犯罪行为。甚至可以应用到运动比赛中，识别运动员的犯规动作。

2. 手势识别，翻译语言障碍者的手语动作。

3. 人机交互。让人可以通过动作控制计算机，例如移动鼠标，点击，拖动等。可以应用到各类体感游戏中，增加游戏交互性。这项技术更可以应用到机器人上，使机器人更好地服务人类。

4. 多媒体信息检索。允许用户输入文本检索对应相关动作的多媒体信息。

第二节　相关研究

一、　基于彩色图像的人体姿态识别

由于设备的限制，之前大部分人体姿态数据都是由普通摄像头采集的，因此只能产生彩色图像，大部分识别算法都是基于这种图像的。前人的一篇概览很好地总结了这方面的算法 [1].

所有的基于彩色图像的人体姿态识别都是按照以下三个步骤进行的：

1. 特征提取：根据图像数据提取出包含识别过程所需信息的向量
2. 动作建模：对上一步获得的数据进行分类（加标签），然后构建训练模型
3. 动作识别：将测试数据按第一步提取特征后输入到上一步的模型中，输出即为预测结果

基于彩色图像的人体姿态识别算法大致可分为两类：一类是基于已检测出对象的模型的 (Models of Pre-detected Objects)，另一类是基于全局统计数据的 (Global Statistics)。前者其实就是要求行为主体已被追踪到，得到其关节点 [2] 或者运动轨迹 [3]。后来又有学者提出仅基于主体轮廓信息进行识别的算法 [4]。后者相对来说不需要事先追踪到行为对象，是针对整个图像信息提取特征的。这个方法相比于前者的确少了很多限制，但是往往计算复杂度很高。为了减少这种复杂度，一种方法是对图像加以限制，比如对行为者的背景 [5], 另外一种方法是基于局部描述子的方法，可以只对直接与行为相关的信息进行计算 [6]。

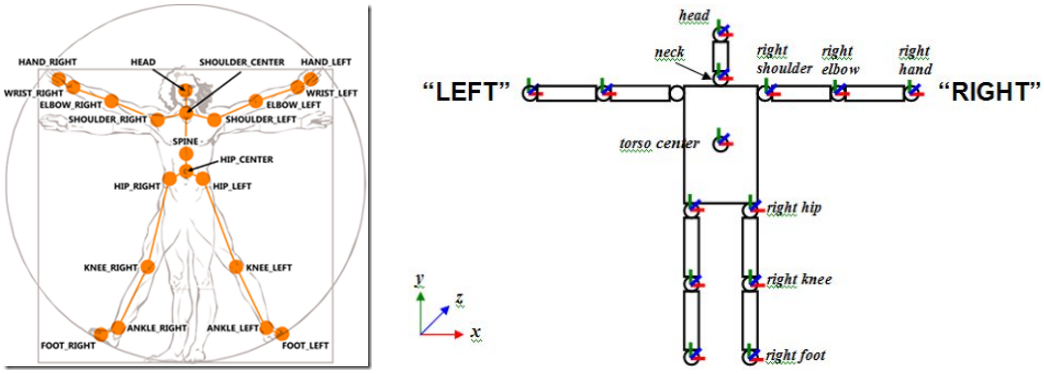
尽管基于彩色图像的识别方法可以有效解决不少场景中的问题，但还是存在一些不足。首先，行为者体征及背景对识别影响很大，而考虑到这些因素后，自然会需要庞大的数据集进行训练，提取特征、训练等过程的时间开销很大；其次，对于一些侧身的姿态识别不佳。传统的摄像头只能产生二维数据，而实际行为者的姿态是三维的，相当于丢失了一维的数据，因而难以将其与其他正身动作区分。虽然有研究人员提出了一些基于多视图的识别算法来处理此问题 [7]，但实际采集和预测数据时都会比较麻烦，可操作性不佳。

二、 基于 Kinect 的人体姿态识别

图 1.1 Kinect 结构简图

Kinect 是微软于 2009 年 6 月 2 日发布的新一代体感摄像头。不同于以往的摄像头，该外设不仅可以通过 RGB 摄像头接收彩色图像数据，还可以通过红外线发射器和红外线 CMOS 摄像头产生深度图。该外设的出现对于人体姿态识别研究起到了里程碑的作用，因为得到了深度图，就意味着可以得到人体的三维数据。相关研究结果表明基于深度图识别的方法 [8] 准确率还是比较高的，可以有效解决基于彩色图像算法侧身姿态识别不佳的问题，并且可以忽略背景色，光线，行为者穿着等因素的影响。然而背景杂物的存在还是会干扰整个识别过程，特别是特征提取过程。

Kinect 推出后不久，基于 Kinect 的各方面技术发展都很快。2010 年时负责 Kinect 一半制造任务的 PrimeSense 公司发布了一个中间件 NITE，其中集成了骨骼追踪技术，可以配合 openNI 机构推出的 Kinect 跨平台编程框架使用。2011 年微软推出了其官方软件包 Kinect SDK，其中也支持了骨骼追踪。骨骼追踪是一项非常具有开创性的技术，通过分析深度图数据把表示人体的数据分离出来，然后提取骨骼点三维坐标数据。具体算法实现无论是 PrimeSense 还是微软都没有公开，但是从骨骼官方规范上来看，由于提取的骨骼点数量不同，定义也不大相同，所以可以推测算法应该是不相同的 (图1.2)。



(a) Kinect SDK　　(b) openNI SDK(NITE)

图 1.2 骨骼规范示意图

基于骨骼图的动作识别目前还存在很大的空白。尽管有学者利用 Kinect 骨骼信息进行简单的动作识别 [9]，但其可识别的动作是预定义的，用户无法自行添加其他动作，识别时也是利用数据匹配的方法，虽然识别速度快，准确率也不低，但扩展性很差。

在本论文中，我采取的是基于 Kinect 骨骼数据，采用 SVM,KNN 算法进行动作识别。本方法的提出填补了基于骨骼图识别姿态这一研究课题的一块空白，并且相比于之前的研究有以下几个优点：

1. 识别算法不受行为者体征，穿着，背景色等因素影响
2. 用户可以自行定义各种姿态，灵活性强，扩展性好
3. 识别准确率高，可以区分静态姿势和动态动作

在提出本方法的同时，本人还做出以下两点贡献：

1. 创建了一个包含 16 种静态姿势和 11 种动态动作的数据集，方便以后的研究人员进行姿态识别的研究
2. 首次创建一个封装了基于骨骼数据进行姿态识别的开源库，面向研究和开发人员，并且具有很好的扩展性。

第二章　基于骨骼图的姿态识别

基于 Kinect 骨骼图的姿态识别流程大体与基于彩色图像的识别流程相同（图2.1)。整个过程分为训练和预测两个阶段。无论哪个阶段，一开始都要采

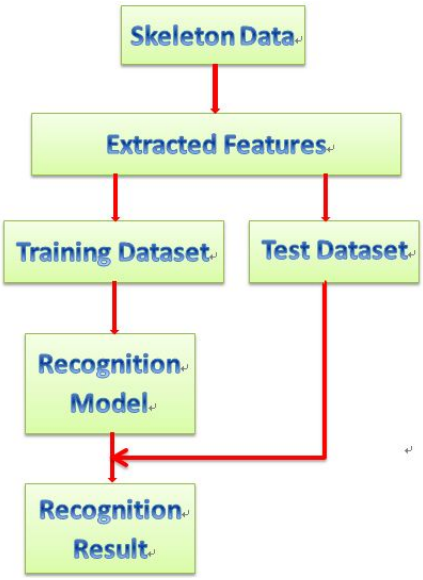


图 2.1 识别流程

集骨骼点数据，并提取特征。训练其实就是对采集到的数据建模的过程，模型的参数需要事先设定好，然后将预测时的数据输入到该模型中，得到预测（分类）结果。本章节就对这一过程的每一步进行详细介绍。除数据采集外，涉及到的编程均在 MATLAB 下完成。

第一节　数据采集

本论文算法验证基于 Kinect SDK 产生的骨骼图。所谓数据采集，即将捕捉到的每一帧骨骼数据存储在一个文件中。本课题中的数据集构建就是在这一步完成。数据集由两个不同身高的人的姿态组成，采样时行为者正对 Kinect 摄像头，并且保证一次采集摄像头视野中有且只有一个行为者，没有其余干扰物，并可以拍到整个行为者。注意本论文的识别方法并没有考虑侧身姿态，而侧身姿态其实也很容易转化为正对摄像头的姿态 [10]。每次采样时，摄像头尽可能在相同高度。由于我们整个算法是基于采集到的骨骼数据，而无论背景如何，微软已将骨架信息提取出来，所以我们的方法不用考虑背景色和人物穿着的影响。

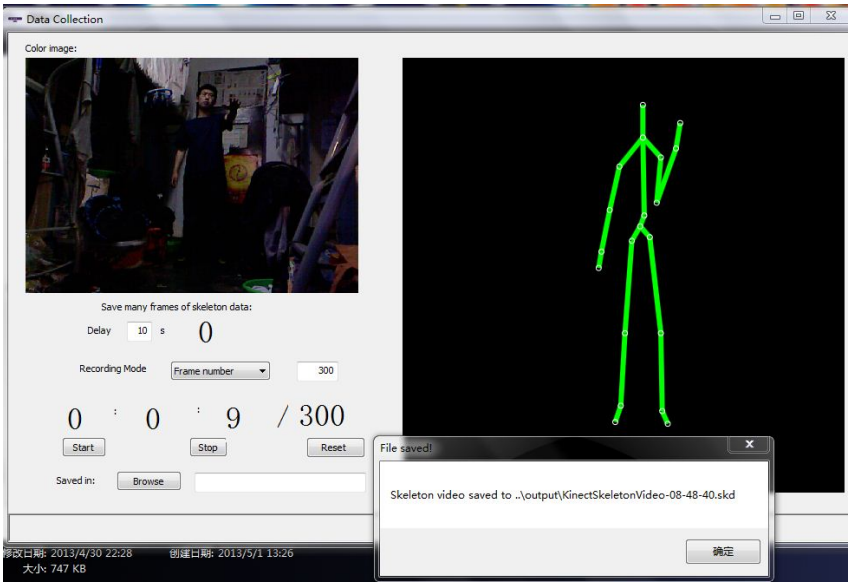


图 2.2 数据采集工具

数据集一共由 27 个姿态组成，包含 16 个静态姿势和 11 个动作。为了保证每个姿态识别的准确率，每个姿态对应的数据量都是相同的（从而保证训练集是平衡的）。具体数据集规范可以参看附录 A。

数据的采集是通过我自己写的一个程序完成的，这个工具同时也是我发布的开源库中的一个示例程序（图2.2)。

第二节　特征提取

特征提取对于姿态识别来说是非常关键的一步。我们从中提取的 13 个特征向量中，9 个用来表示人体静态特征（静特征, 表2.1），4 个表示动态特征（动特征, 表2.2）。

表 2.1 静特征 (引用 [11]TABLE I)

Feature(D) ID	$J_n[i]$	$J_n[j]$	Description
1	Left Wrist	Left Elbow	Left Forearm
2	Right Wrist	Right Elbow	Right Forearm
3	Left Elbow	Left Shoulder	Left Upper Arm
4	Right Elbow	Right Shoulder	Right Upper Arm
5	Left Hip	Left Knee	Left Thigh
6	Right Hip	Right Knee	Right Thigh
7	Left Knee	Left Ankle	Left Crus
8	Right Knee	Right Ankle	Right Crus
9	Hip Center	Shoulder Center	Spine

表 2.2 动特征

Feature(D) ID	$J_n[i]$	$J_{n+1}[i]$	Description
10	Left Wrist of Frame n	Left Wrist of Frame n+1	Velocity of Left Wrist
11	Right Wrist of Frame n	Right Wrist of Frame n+1	Velocity of Right Wrist
12	Left Knee of Frame n	Left Knee of Frame n+1	Velocity of Left Knee
13	Right Knee of Frame n	Right Knee of Frame n+1	Velocity of Right Knee

第 n 帧的数据可以表示成如下向量形式：

$$J_n = \{(x_{n,1}, y_{n,1}, z_{n,1}), (x_{n,2}, y_{n,2}, z_{n,2}), ..., (x_{n,20}, y_{n,20}, z_{n,20})\} \tag{2-1}$$

即 20 个骨骼点三维坐标。每一个坐标对应一个关节，其名称可以参考 SDK 官方文档。第 n 帧静态特征向量计算公式（参考 [11] 公式（2））：

$$D[m] = \frac{J_n[i] - J_n[j]}{||J_n[i] - J_n[j]||}(i \neq j, 1 \leq i, j \leq 20, 1 \leq m \leq 9) \tag{2-2}$$

其中，$J_n[i]$,$J_n[j]$ 分别为 J_n 向量中的某个三维坐标。m 对应着特征向量 ID

第 n 帧动态特征向量计算公式：

$$D[m] = \frac{J_{n+1}[i] - J_n[i]}{time_interval}(10 \leq m \leq 13) \tag{2-3}$$

其中 $J_{n+1}[i]$，$J_n[i]$ 为相邻帧对应关节点的坐标，m 含义同上。

由公式2-2可以看出，静态特征已被归一化，之后的模型输入都是建立在此归一后的特征上，所以静态特征不受行为者体型大小影响；而根据公式2-3, 动态特征本质上就是关节点的运动速度，因此也不会受行为者体型大小影响，故本文介绍的识别方法与行为者体型大小无关。

第三节　识别模型构建

通过上一步，我们得到了所有数据点的特征值，即一个 39（13×3) 维向量。然而为了更好地区分姿势和动作，我们假设所有静特征和动特征对姿态的贡献分别一致，即增加一个权重向量 W:

$$W = [1, 1, ..., 1, \alpha, \alpha, ..., \alpha] \tag{2-4}$$

其中，”1“为静特征的权重，α 为动特征的权重。即提取的特征向量中的动特征在输入到模型前要乘上权重 α

下面分别介绍 SVM 和 KNN 这两种分类器在姿态识别上的运用。

一、 SVM

SVM(支持向量机）是一个非常流行的机器学习算法，通过升维巧妙地将原来非线性可分问题化为在新的特征空间内线性可分问题，被广泛应用于各种分类问题。相关的软件包也很多，其中一个非常著名的便是台湾大学林智仁教授开发的 LIBSVM。其开源，跨平台，轻量级，支持多种开发环境，支持多种

核函数和参数选择，并附带多种实用小工具，对于本论文中的课题非常合适。因此本人选用该软件包，并用 MATLAB 编程实现后续理论验证。

根据 [12], 对于多分类问题，SVM 类型选择 C-SVC, 核函数选用 RBF, 此时该模型有两个参数：C 和 γ。参数的不同会影响该模型预测时的准确率，因此首先要解决的便是参数寻优问题。在只有训练集的情况下，检测准确率最好的办法便是利用 n 重交叉验证（n-fold cross validation), 即将训练集均分成 n 份，每次取其中一份作为测试集，剩下 n-1 份作为训练集，训练模型并预测结果，如此循环 n 次，累加结果，得出最终“准确率”。这种方法可以有效避免过拟合问题。一个普遍使用的参数寻优的方法是网格寻优法，就是遍历某区间内的所有值，比较选出使得交叉验证准确率最大的参数对。这种方法的本质就是穷举，一个明显的问题便是时间开销过大。LIBSVM 中提供的是改进的网格参数寻优的算法，先进行“粗网格寻优”，即选择一组 C 和 γ 的 2 的指数递增序列作为试验点，对两参数间每个取值组合进行一次 n 重交叉验证，找出使准确率比较大的一个大致范围，然后在该范围内进行“细网格寻优”，即将参数步距变小，从而找到使交叉验证准确率最大的精确参数组。

本实验中，数据集采用的是本章第一节中所构建的数据集，基于 10 重交叉验证的改进网格寻优，结果如图2.3。需要注意的是，此时权重 α 取值为 3，

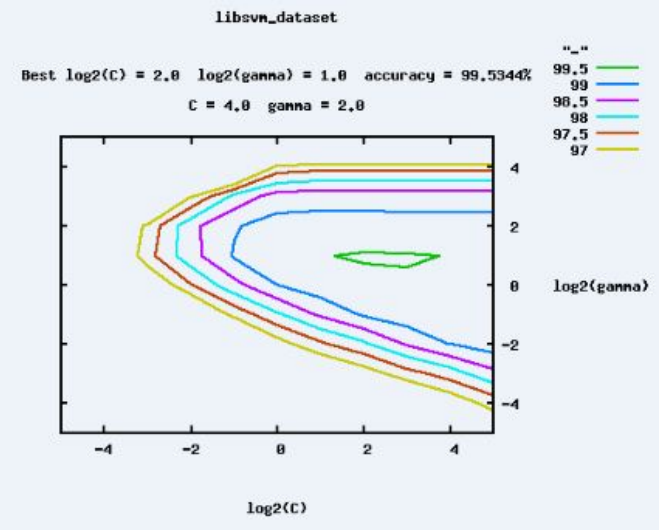


图 2.3 LIBSVM 改进网格寻优结果

至于为何不用考虑其他值，第三章将会说明。至此，模型参数全部得到，利用

LIBSVM 内置函数便可构建出识别模型。

二、 KNN

KNN（k-Nearest Neighbor algorithm，最近邻算法）是机器学习中最简单最常用的分类算法之一。与 SVM 最大的不同是这种算法无需事先训练建模。给出一个需要分类的数据，算出其与已有训练集中每个数据的距离（存在多种距离算法，一般选择欧式距离），然后找出距离最小的 K 个数据，其中占多数的类别便是最终测试数据所属类别。该算法影响分类准确率的参数只有 K。结合动特征的权重 α, 对于本文的姿态识别问题来说，需要进行寻优的参数是 K 和 α，寻优方法采用与 SVM 寻优类似的基于 10 重交叉验证的网格寻优, 寻优结果如图2.4 最佳参数 $K = 3$，$\alpha = 3$，识别模型构建完成。

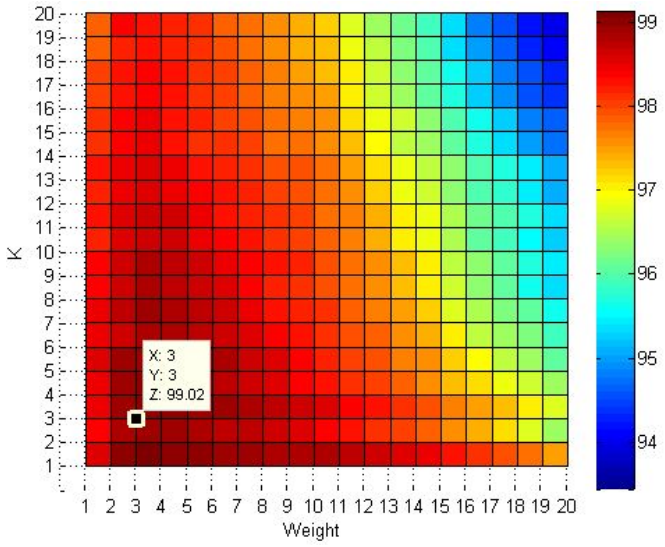


图 2.4 KNN 参数寻优结果

第四节　姿态预测

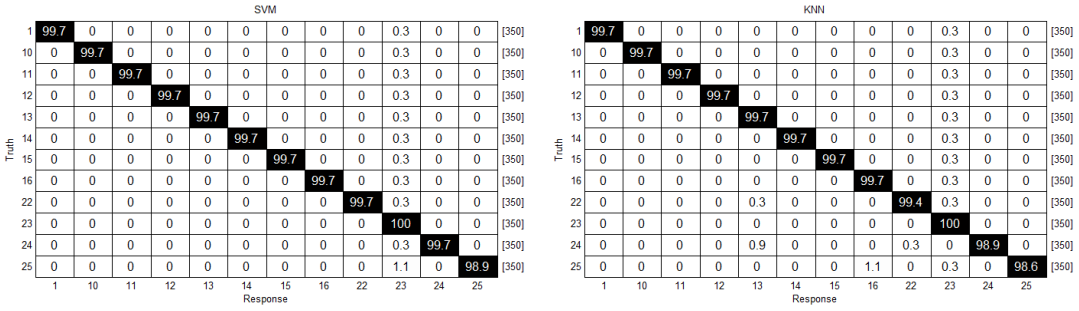
模型建好后，实际对姿态的预测就非常简单了，将测试集输入到模型中即可。由于训练集中每一帧骨骼点数据对应一个训练数据点，在预测时对于每一帧数据都将给出预测。为了减少预测的偶然性，实际预测时可以将每 n 帧的预测结果存起来，然后判断这 n 个结果中比重最大的分类，作为这 n 帧数据的最终预测结果。在下一章我们会利用 10 重交叉验证对模型分类准确率进行评测，其中会对 n 值的选择进行详细讨论。

第三章　实验结果与分析

本章将对上一章的实验预测结果进行评测。所用的数据集是上一章第一节建立的数据集，利用 10 重交叉验证进行识别准确率测定，在 MATLAB 下完成。识别模型已按照上一章找到的最佳参数建立好。

第一节　混淆矩阵（Confusion Matrix）分析

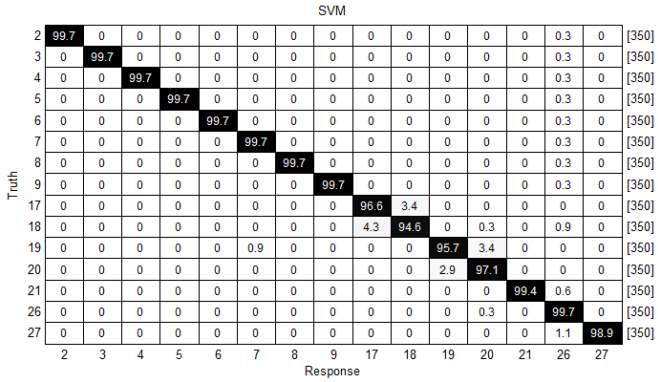
由于原数据集中姿态很多，分类结果不宜全部表示，这里将他们分为两类：腿部姿态和手部姿态。对于姿态识别来说，同一部位的姿态会比不同部位的姿态难分许多，所以同一部位姿态分类的混淆矩阵已经足够体现识别准确率。



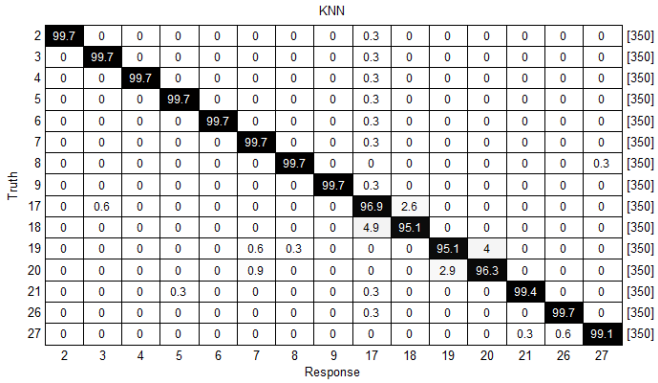
(a) SVM 腿部姿态　　(b) KNN 腿部姿态

图 3.1　两种分类器对于腿部姿态识别的混淆图

腿部姿态的混淆矩阵如图3.1所示。具体标号对应的姿态可参照附录 A。从图中可以看出，两种分类器对于腿部任何一种姿态分类准确率都达到 98% 以上，意味着不管是姿势（1, 10-16) 与姿势、动作（22-25) 与动作，还是姿势与动作都可以很好地被区分开。



(a) SVM 手部姿态



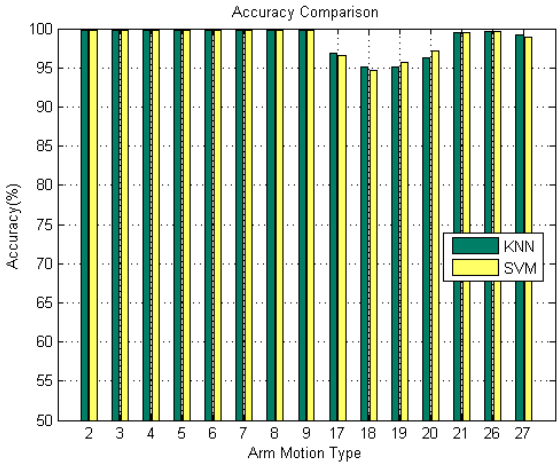
(b) KNN 手部姿态

图 3.2 两种分类器对于手部姿态识别的混淆图

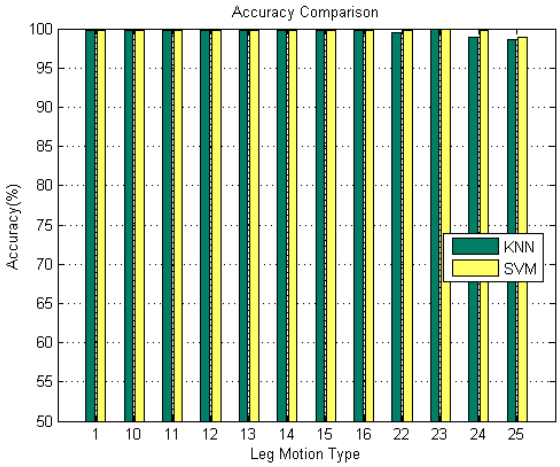
手部姿态的混淆矩阵如图3.2所示。图中 2-9 号对应的是手部姿势，其余均为手部动作。手部姿势的识别准确率和腿部姿势近似都为 99.7%, 四个挥手动作（17-20）的识别准确率相对低一点, 而且仔细观察会发现，左手左挥与左手右挥易混，右手左挥与右手右挥易混。这个结果还是比较合理的，因为能够区分同一只手不同挥动方向动作的特征只有一个：肘关节速度向量，而其余动作之间往往还存在大量静特征的区别，所以同一只手的挥手动作之间更易混淆。然而即使这样，最低分类准确率达到 94% 也是个不错的成绩。

第二节　SVM 和 KNN 分类准确率比较

本节主要比较对于每个姿态两种分类器的不同分类准确率。这里要说明一下，实验表明，当动特征权重 α 不小于 3 时，对于不同的 α，SVM 对于各姿态的识别准确率已经没有显著变化。所以我们可以固定 α。然而实际中的 α 还需要试验决定。



(a) 手部姿态准确率比较



(b) 腿部姿态准确率比较

图 3.3 两种分类器对于各姿态识别准确率比较

如图3.3可以看出，两种分类器在识别姿势上几乎没有区别，但在识别动作

上，SVM 要比 KNN 表现稍好一点，尤其是在识别腿部动作时。另外还可看出，本方法对于姿势的识别普遍要比动作好，一个原因是上一节最后提到的动作间的干扰问题，另一个是由于动特征计算值存在不稳定性（主要由于 Kinect SDK 无法直接从数据流中得到时间信息，而靠程序计算时间本身就存在一定的误差）。然而，需要注意的是本章目前为止讨论准确率都是基于每一帧的预测结果。上一章最后提到，实际应用中可以有 n 帧的缓存来减少预测误差，提高准确率，当然这样会付出一定的时间代价。下一节便详细讨论此问题。

第三节　缓存帧数与识别准确率的关系

所谓缓存帧数，即每隔多少帧进行一次预测，实际上就是统计一下这 n 帧数据的 n 个预测结果哪一类的比例最大。在实际应用中，为了获得更好的预测效果，往往需要找到一个合适的 n 值。尽管对于本课题，n=1 时已经能达到比较高的预测准确率，但是我们还是可以通过实验看看缓存帧数与分类准确率的关系，是否可以使得我们的预测效果进一步改善。

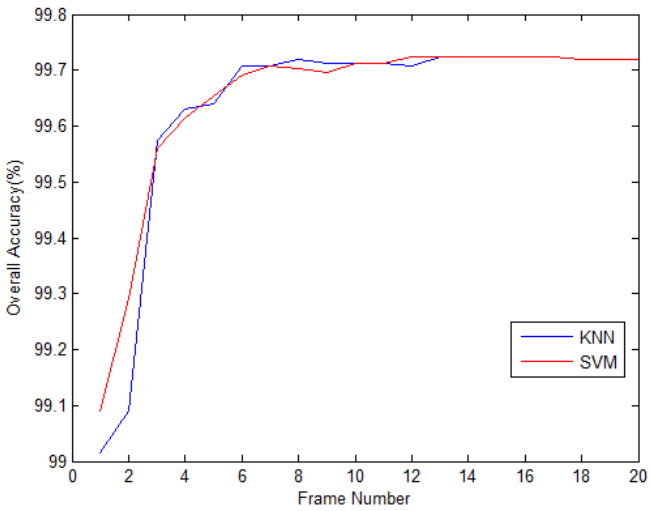


图 3.4 整体识别准确率 —缓存帧数图

这里之所以除了考虑整体识别准确率（所有姿态的平均识别准确率）外，还要考虑动作识别准确率（所有动作的平均识别准确率）主要是因为动作识别相对于静态姿势的识别来说更加不稳定，整体识别准确率与缓存帧数的关系不

一定与动作识别准确率与缓存帧数的关系一致。实际上，静态姿势识别因为当缓存帧数为 1 时就具有极高的识别准确率（99.7%)，很好地掩盖了动作识别准确率变化的不稳定性（图3.5)。

然而不管多么不稳定，两幅图都展现了相同的趋势：缓存帧数越多，识别准确率就越高。这个实验结论也非常符合理论常识，这里就不多解释。从两图还可看出，在缓存帧数不大于 3 时，SVM 平均识别准确率均高于 KNN, 只有在缓存帧数更多时，KNN 才与 SVM 不相上下。当缓存帧数不小于 7 时，两种分类器的预测准确率均在 99.5% 以上。由于 Kinect 平均帧率为 30FPS，这样等待时间大概为 233ms，在实际应用中一般都是可以接受的。

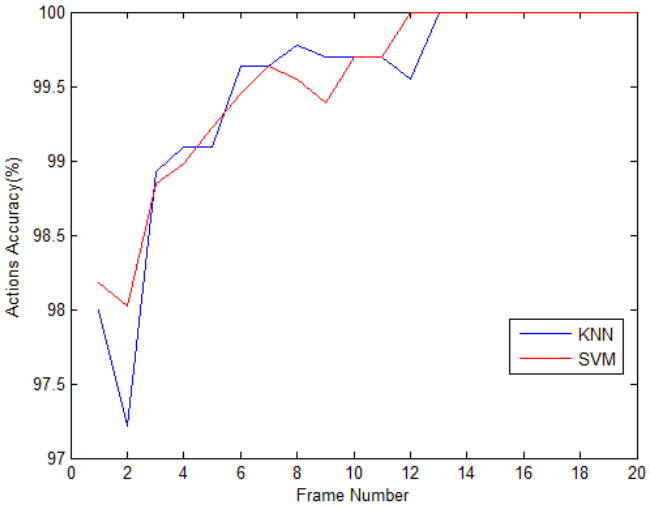


图 3.5 动作识别准确率 —缓存帧数图

第四节 实验小结

本节主要对上一节构建的数据集进行了姿态识别测试，基于 10 重交叉验证，SVM 和 KNN 分类器对各姿态识别最低准确率分别为 94.6% 和 95.1%，整体平均识别准确率分别约为 99.1% 和 99.0%, 动作平均识别准确率分别约为 98.2% 和 98%。两种分类器的表现都达到了一个比较高的水平。并且进一步实验结果表明，分类器分类准确率随缓存帧数增加而增加，对于本实验来说，缓存帧数为 7 时，两种分类器分类准确率可达到 99.5% 以上，而为预存所等待的时间仅为 233ms。

虽然实验结果表明 SVM 在准确率上比 KNN 略胜一筹，但是其参数相对较多，寻优耗时较长，并且训练时间也会随着训练集的增大而变长，而 KNN 本质上只存在预测时间，且要相对短许多，并且如果选用比较优化的数据结构实现，或者增加“模糊度”，其时间效率还会提高很多（详见下一章）。然而在实际应用中，SVM 的寻优并不需要严格用网格法，或是基于 n 重交叉验证，完全可以利用少量测试集直接测试，进行几组比较即可，在最佳参数附近的很大范围内，SVM 通常都可以获得很好的预测效果，所以 SVM 分类法还是具有很大实际应用价值的。

第四章　开源库 ARLK 的构建

第一节　库的意义和设计目标

为了使更多的人参与到 Kinect 相关应用的开发和研究中，许多面向 Kinect 的软件包发布了出来，其中最主流的便是 Kinect SDK 和 openNI SDK。这些 SDK 定义了各自的 API 标准，使得研发人员可以编写程序调用他们来控制 Kinect，操作相关数据。随着 Kinect 相关应用逐渐趋于复杂，加上相关技术发展迅速，单纯用这些 API 构建项目变得愈来愈复杂、繁琐，因此许多相关的库涌现了出来。然而这些库普遍表现出 API 局限性很强，因为他们本身调用了操控 Kinect 的 API。在功能上，基于 Kinect SDK 的库基本都是用来识别手势动作的。基于 openNI SDK 的库相对来讲种类更多些，库的功能也更加强大，比如 SigmaNIL, 3D Hand Tracking Library, VIIM SDK 等（详见 openNI 官网）。然而这些库鲜有完成姿态识别功能的，仅有的只是针对识别手势动作，并且只针对预定义好的动作，灵活性、扩展性较差，用户无法自定义新动作。之前的工作已经验证本论文提出的运用机器学习算法进行姿态识别方法的有效性，所以有必要将该方法封装在一个库中，方便研发人员的调用。

除了实现姿态识别功能，本库还将具有以下特性：

1. 开源，遵从 GPL 协议。这样允许程序员对该库进行修改和再发布，相当于拥有更多的维护者

2. 跨平台并且支持多种 Kinect 的 SDK。Kinect 的应用要想流行，就注定不会只停留在 Windows 系统上，跨平台特性还可满足更多程序员的使用习惯，鼓励更多人加入研发队伍。

3. 效率高。这是任何库所必须具备的特性。因为库要被多种应用调用，效率

低的库会给整个应用带来灾难性的影响。

4. 与尽可能多的编程语言兼容，即可以被多种语言调用。

5. 易扩展。使用者可以方便添加新的分类器，新的识别算法，扩展库的功能接口。使用者也可以方便添加新的骨骼规范，可以支持更多的深度图提取骨骼点的算法。

初步将本库的名字定为 ARLK（Action Recognition Library for Kinect)。下面将详细介绍 ARLK 的设计。

第二节　ARLK 类结构详解

为了满足第一节提出的特性 2，3 和 4，我选择用 C++ 来写这个库。选用面向对象的编程语言最大的优点就是结构清晰，便于扩展和增量开发。然而从执行效率来说 C++ 并不是最高的，高级语言里 C 的效率要更高，所以本库中一些需要提高效率的地方采用了 C。这种以 C++ 搭结构，以 C 实现部分接口的方法被广泛应用在各类后台程序及开源库的开发中。C++ 库的兼容性也比较好，其他很多编程语言如 C#,JAVA，都支持对 C++ 库的调用。

目前 ARLK 0.1 版本主要由四个类组成：SkeData, BaseClassifier, SVMClassifier 和 KNNClassifier，关系如图4.1

一、　类名字空间设计

名字空间（namespace), 即标识符的各种可见范围。C++ 中最常见的名字空间便是标准库的“std”。名字空间的引用，使得标识符在被引用时需要加名字空间为前缀，如:

```
std::cout << "Hello" << std::endl;
```

尽管这看上去会麻烦些，但却有效避免了名字冲突，这个问题在设计库时尤其需要注意，因为库是要被各种应用程序复用的，越大的程序，标识符种类就越多，如果不加名字空间限制，很容易发生名字冲突，有时如果库嵌套层数很多，想检查出哪里冲突会非常耗时，给程序员带来不必要的麻烦。这样一来花的代

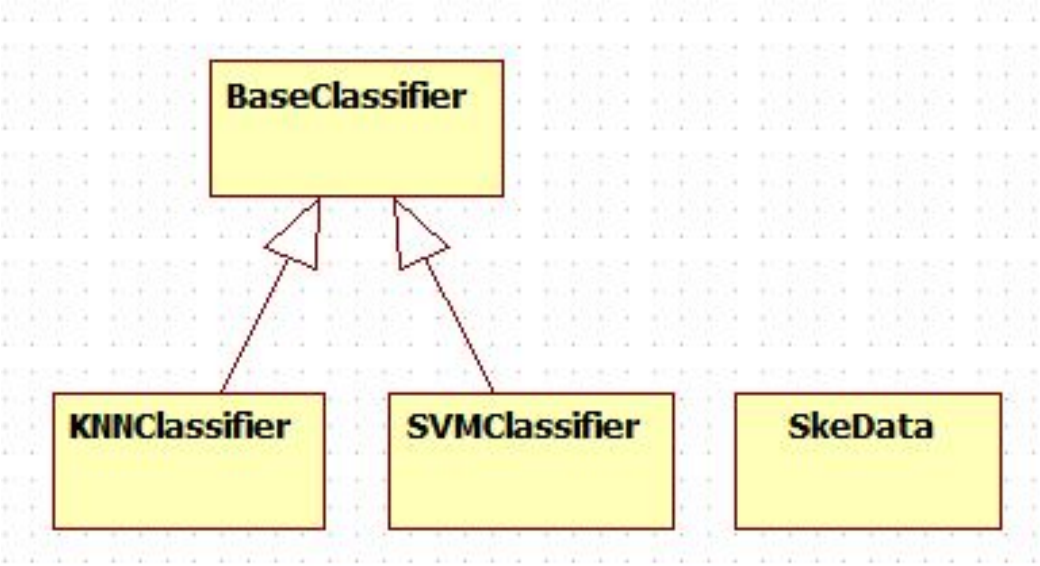


图 4.1 ARLK 主要类 UML 简图

价远远超过省下的那几个前缀字符，得不偿失。故名字空间的引入是非常必要的。

本库中，所有类都属于“ARLK”名字空间，如：

```
namespace ARLK {
  class BaseClassifier
  {
  public:
    BaseClassifier();
    ~BaseClassifier();
    ...
  protected:
    ...
  };
}

namespace ARLK {
```

```
    BaseClassifier::BaseClassifier():
      labels(NULL),
      dataNum(0),
      actWeight(3)
    {
      CleanData();
    }

    BaseClassifier::~BaseClassifier()
    {
      CleanData();
    }
      ...
}
```

ARLK 四个大类的定义和实现都在 ARLK 名字空间中，所以只要用到这四个类的方法时，都需要加上“ARLK”名字空间前缀。

二、 SkeData 类介绍

这个类的主要功能就是存储骨骼点数据及相关辅助信息，同时可以完成与骨骼存储文件的交互工作。

```
class SkeData
  {
  public:
    SkeData();
    ~SkeData();

    bool AddOneFrame(const Vector4* SkeVector);
    bool SaveStart(const char* FileName = NULL, int
        FrameLimit = 0, SKELONG TimeLimit = 0);
    bool SaveEnd();
```

```
    bool SavingIsStopped() ;

    int GetFrameSaved();

    SKEFILEHEADER* GetSFH();
    SKEINFORHEADER* GetSIH();

    static bool GetHeaders(FILE* pFile, SKEFILEHEADER *
        psfh, SKEINFORHEADER *psih);
    bool GetHeaders(FILE* pFile) ;
    bool GetFramesFromFile(FILE* pFile);
    bool AddFramesFromFile(FILE* pFile);
    bool WriteFrames2File(const char* FileName);

    void ClearContent();
    SKDPOINT* GetPoint(int frameIdx, int jointIdx);

private:
    std::vector<SKDPOINT*> m_SkePoints;
    int m_cFrame;
    bool IsStopped;

    SKEFILEHEADER* m_pSFH;
    SKEINFORHEADER* m_pSIH;

    clock_t         m_LastFrameTime;
    clock_t         m_StartTime;
    clock_t         m_EndTime;
    SKELONG         m_TimeLimit;

    int             m_iFrameLimit;
```

```
    FILE* m_pOutputFile;
};
```

SkeData 类在本库中是个非常基础的类，其他类的接口中会调用该类的对象。通常当一帧的数据流准备好时，会触发一个事件，这个事件又会触发相应的操作（函数），程序员如果要存储该帧数据，就必须要在这个函数里实现。SkeData 类的对象就像一个“智能存储箱”，如果在该函数中调用它的 AddOneFrame 方法，就可以将该帧的骨骼数据存入这个存储箱中。需要注意的是这个函数的输入是一个指向 Vector4 结构的指针，而该结构定义为：

```
typedef struct _Vector4
{
    float x;
    float y;
    float z;
    float w;
}   Vector4;
```

这实际上是从 Kinect SDK 定义的 API 中借鉴过来的，x,y,z 分别是骨骼点三维坐标，w 为摄像头高度，通常为 1，这个定义完全适用于任何 SDK, 因为任何骨骼点数据都是三维坐标的形式。需要注意的是，实际应用中所需存储的帧数通常是有限的，比如使用者通常希望存储一段时间或某个数目的帧数，并且存储的目标不一定仅在 SkeData 对象中，还可能希望存在文件里。为了满足这种需求，我给 SkeData 类加入了 SaveStart 方法，其三个参数分别代表输出文件名，帧数限制，时间限制。为了保持统一，规定无论对存储帧数有无限制，这个函数都要在读取数据流之前调用。该函数相当于对“智能存储箱”的开启和初始化，“存储箱”会根据施加的限制自动停止存储。在骨骼数据存储完成后，需要调用 SaveEnd 方法结束存储并进行一些收尾工作，比如文件头的存储等等。然而这个流程有个不足，就是需要使用者在存储数据前就要指定文件名。为了允许用户在存储数据后再指定文件名，我加入了 WriteFrames2File 方法，可以将 SkeData 对象中的数据导入到文件中，指定文件名即可。

上边介绍的是 SkeData 类对象对实时数据流中数据的存储调用，然而在实际研究中，不断的采数据是非常耗体力的事，更多的研究人员希望能够从“文件”中读取需要的数据。为了满足这种需求，我加入了 GetFramesFromFile 方法和 AddFramesFromFile 方法。从名字上可以看出，两种方法都可以将文件（skd 文件）中数据读入到 SkeData 类对象中。但不同的是，前者只能读取一个文件的数据，而后者可以读取多个文件并不断累加，只要使用者不断循环调用该方法即可。

以上便是 SkeData 类方法的设计思想和使用流程，使用者可以参照示例程序加以理解。为了更进一步剖析该类，这里介绍一下该类的非公有成员变量。m_SkePoints 是用来存储骨骼点的，注意到这是一个存有指向 SKEPOINT 指针的 C++ STL vector。SKEPOINT 是个很重要的数据结构：

```
typedef struct tagSKDPOINT{
    short x;
    short y;
    short z;
    short w;
    short t;
}SKDPOINT;
```

该结构存储一个骨骼点的三维坐标（x,y,z），摄像头高度 (w) 以及这一帧与上一帧的时间差（t)。这个结构中前 4 个变量与 Vector4 结构是对等的，不同之处在于存在单位的换算（这里便存在本库中的一个需要改进的地方，虽然换算后会节约存储空间，提高计算速度，但会存在精度丢失，对识别结果还是会有一点影响的）。除此之外，该类还存有帧数, 表明存储是否结束的指示器, 文件头指针（具体结构及含义见附录 A），各种时间存储变量和输出文件指针。为了能够方便得到这些变量的值，同时保证不破坏类的封装，SkeData 类还提供若干返回这些变量内容的方法，具体接口可以见 ARLK API 文档。

三、 BaseClassifier 类介绍

从该类名字不难看出，这是所有姿态分类器的基类：

```
class BaseClassifier
  {
  public:
    BaseClassifier();
    ~BaseClassifier();

    virtual bool FeatureExtraction(SkeData* inputData,
      const double* dataLabels = NULL, int labelNum = 0)
       {return true;}
    virtual bool train() {return true;}
    virtual double* predict(SkeData* inputData, int*
      pLength) { return NULL;}
    static int** genConfusionMatrix(double*
      predictedLabels, double* trueLabels, int num, int
      labelTypeNum);

    virtual void CleanData();

  protected:
    static bool CalFeatureVector(SkeData* inputData, int
      frameIdx, int vecIdx, FEATURE_VEC* pSkdPointOut,
      double wt);
    int dataNum;
    double* labels;
    double actWeight;
  };
```

该类的主要作用是定义一个所有分类器的公共接口，所有分类器都要继承该类，可以根据需要覆盖基类中方法。

对于任何一个分类器，通常都有特征提取、训练、预测这三个步骤，对应于该类的方法分别为 FeatureExtraction, train 和 predict。注意到包括这几个方法在内的若干方法都是虚方法，意味着如果是用基类指针调用这些方法的话，

会遵从动态绑定。这便是 C++ 的多态特性。这样的好处是如果该库最后被扩展出很多分类器并且都要使用同一方法时，可以用一个基类指针直接循环调用这一方法，而不用事先一个个判断分类器的类型。

当调用 FeatureExtraction 方法时，需要给出所有数据点数据及相应的标签，以及训练集的大小。这样规定是为了简化 train 方法的接口，因为在提取特征时已经可以得到训练时所需的全部信息，并存在相应的分类器对象中。这个方法虽然任何分类器都会执行，但通常情况下分类器内部数据结构会直接影响到该函数的实现，而本库是允许用户自行添加新的分类器和数据结构的，所以各个分类器通常会覆盖 FeatureExtraction 方法。然而不管是什么分类器，其特征提取都是具有相同的算法，即本论文第二章第二节所提出的，所以我将该算法实现提取了出来，加入到 BaseClassifier 类中，成为一个公用方法 CalFeatureVector。注意到该方法是静态成员函数，意即使用者可以不实例化直接调用，这样该方法可以用到更多的场合中。

用于训练的 train 方法，是任何有“训练”过程的分类器都要覆盖的方法。之所以这里没有用纯虚函数，是因为考虑到某些分类算法其实并不需要训练这一过程。predict 方法用来返回预测结果，返回值为指针意味着该方法可以给出多数据的预测结果。genConfusionMatrix 方法是用来返回混淆矩阵的，一定要在训练、预测方法后使用。

BaseClassifier 中的成员变量是所有分类器都会用到的变量，即训练集大小，标签数组和动特征参数。需要注意的是本库中规定任何分类器的标签都用 double 型（为了最大化精度，因为有些分类器默认便是 double，如果不是 double 的话会因转换而有可能丢失精度），所以如果使用者的数据集标签是其他类型（比如字符串），则需要自行映射。CleanData 用来清除数据内容并释放相应空间，如果重复运用该类对象时，一定要注意调用该方法。

四、 SVMClassifier 类介绍

SVMClassifier 类本质上是对 LIBSVM 的进一步封装。

```
class SVMClassifier : public BaseClassifier
  {
  public:
```

```
    SVMClassifier();
    ~SVMClassifier();

    void SetSVMParameters(double C, double gamma, double
        weight);
    virtual bool FeatureExtraction(SkeData* inputData,
        const double* dataLabels = NULL, int labelNum = 0)
        ;
    virtual bool train();
    virtual double* predict(SkeData* inputData, int*
        pLength);
    virtual void CleanData();

  private:
    static svm_node** SVMFeatureExtraction(SkeData*
        inputData, int* pFrameNum, int* pFeatureNum,
        double wt);
    void CleanNodes();

    svm_node** SVMNodes;
    svm_parameter params;
    svm_problem pro;
    svm_model* SVMmodel;
};
```

由于 LIBSVM 对数据存储有一套自己的结构，所以 SVMClassifier 需要对 FeatureExtraction 实现进行覆盖，其处理主要在 SVMFeatureExtraction 这个私有方法中。在调用 train 方法进行训练之前，通常要指定模型的参数，这个功能主要由 SetSVMParameters 方法实现，三个参数分别为 C, γ 和动特征参数 α。如果不调用该方法，模型参数将取默认值，详细可参考 API 文档和源码。训练的过程就是根据训练数据和模型参数建立一个 SVM Model, 这个 model 会被存成二进制文件形式。predict 的实现便是先载入之前训练的 model，可以选

择从文件读入，然后根据输入的测试集返回预测结果。由于 SVMClassfier 是对 LIBSVM 的封装，其私有成员变量也都是借用 LIBSVM 中的，所以它们的详细含义可以参考 LIBSVM 官方文档。

五、 KNNClassifier 类介绍

```
class KNNClassifier : public BaseClassifier
  {
  public:
    KNNClassifier();
    ~KNNClassifier();

    void SetKNNParameters(int K, double eps, double
       weight);
    virtual bool FeatureExtraction(SkeData* inputData,
       const double* dataLabels = NULL, int labelNum = 0)
       ;
    virtual bool train();
    virtual double* predict(SkeData* inputData, int*
       pLength);
    virtual void CleanData();

  private:
    static ANNpointArray KNNFeatureExtraction(SkeData*
       inputData, int* pFrameNum, int* pFeatureNum,
       double wt);
    double GetMostId(ANNidxArray idArray, int K, double*
       labelArray);
    ANNpointArray    dataPts;
    ANNkd_tree*      kdTree;
    struct _KNNPARAMS {
```

```
        int K;
        double eps;
    }knnParams;
};
```

KNNClassifier 类的设计是基于另一个开源库 ANN 的。ANN(Approximate Nearest Neighbor) 是一个运用优化数据结构实现 KNN 算法的开源库，之所以选择这个库主要是因为以下几个特点：

1. 用 C++ 写成，开源，跨平台。这个特点使得这个库可以与本人的 ARLK 搭配得非常好，并且满足 ARLK 跨平台的需求。

2. 轻量级，运用优化的数据结构，并且增加一些参数，使得该库是实现 KNN 算法的最高效的库之一。

3. 功能非常强大，可配置选项多。

使用 KNNClassifier 类的流程大致与 SVMClassifier 相同。首先调用 FeatureExtraction 方法进行特征提取，主要实现在 KNNFeatureExtraction 方法中。注意到该方法返回是 ANNpointArray 类型。这个是 ANN 库中定义的，用来存储所有数据信息，具体结构可以参考 ANN 文档。特征提取后就要设置模型参数了，如果不取默认值，就要调用 SetKNNParameters 方法。这个方法有三个参数，第一和第三个无需多加解释，第二个是 ANN 库引入的，该参数 (ϵ) 具体定义可参看 [13], 其值越大，预测准确率越低，但速度会大幅提升。参数设好后便可调用 train 方法。这个方法虽然叫做“train”，但其实际内容与 SVM 的 train 大不相同，因为第二章提到 KNN 并不需要训练过程。这里的 train 方法实际上是根据训练数据（无需标签）直接构建一个 ANN 中定义的查找树，本库中使用的是 kd 树，另一种选择是 bd 树。kd 树其实是一种特殊的二叉树，将数据点所在空间根据每个数据点各维数值依次切割，根据各数据点所处的子空间，安排成树的结构，寻找一个数据点最近邻时只要先找到它所处的子空间，再不断回溯，测试其与父节点和其他子空间的数据点距离即可。通常情况下，这种搜索方法可以避免计算测试点与数据集中每个点的距离，因此可以大幅缩短计算时间 [14]。bd 树与 kd 树类似，不同在于增加了一个收缩规则（shrinking rule), 对前者有一定程度上的优化，特别是当数据点极为集中时 [13]。

经过 train 操作建立起查找树后，便可以调用 predict 方法进行预测了。该方法主要通过调用 ANN 中定义的 annkSearch 函数，根据事先设好的参数来返回最近邻，然而这是不够的，要想得到最后的预测结果还要调用 KNNClassifier 的 GetMostId 方法，找出这些最近邻中比例最大的类别，即最终预测结果。

第三节　ARLK 示例程序介绍

示例程序是每个开源库所必不可少的一部分，因为它们可以直截了当地展示库所定义的 API 的调用方法和流程，使用者甚至还可以直接复用示例程序中的有关代码，更快速地上手。本节将一一介绍 ARLK 中的四个示例程序，其中前三个是 Windows 平台下的，最后一个是 Linux 平台下的，并且均带有 GUI。两种平台的示例程序也很好地展示了 ARLK 的跨平台特性。

一、 DataCollection

这个程序就是第二章第一节提到的数据采集工具（见图2.2)。该工具是在 Visual Studio(2010) 下完成，基于 Windows API 和 Kinect SDK 中 API 设计。该程序主要功能便是根据设定的限制（时间限制或帧数限制，或者无限制）将骨骼数据存入到 skd 文件中（规范见附录 A)。考虑到用户有可能希望录下自己的数据，而电脑距离测试点有距离，所以该程序允许设置延迟时间，具体数值用户可以输在”Delay” 后。用户也可进行中断数据流，或是重置等操作，还可指定 skd 文件存储位置和文件名（默认是存在程序目录下的 output 文件夹下，以存储时间命名)。

本程序主要展示了 SkeData 类有关骨骼数据读取和存储的 API 的用法。程序先注册窗口，再建立消息循环和窗口处理函数（Visual Studio 自动搭建)，然后在窗口处理函数中根据不同消息代号实现不同的处理。对于 Kinect 数据流处理，需要先创建相关事件，在消息循环中等待事件发生后进行相应的处理即可。SkeData 类接口的大量调用就在那个 CDataCollection 类的 ProcessSkeleton 方法里完成。本程序结构比较简单，在此不多介绍。

二、 SkeletonPlayer

该程序的本质就是个骨骼数据播放器，与 DataCollection 类似，也是基于 Windows API 和 Kinect SDK 中 API 设计（图4.2)。

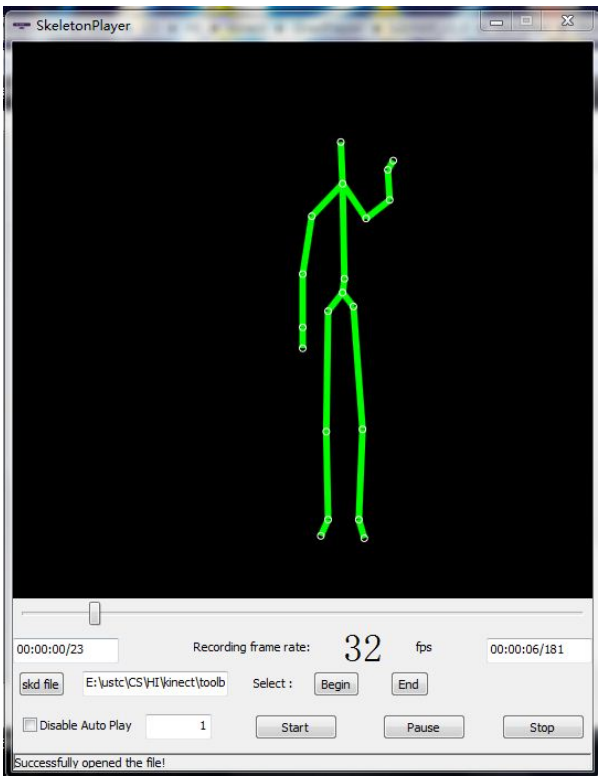


图 4.2 SkeletonPlayer 程序界面

该程序在播放 skd 文件的同时可以显示总帧数，总时长，当前帧号，当前时间，当前帧率等信息，基本上可以展示出所有 skd 文件信息（包括文件头）。使用者可以像使用电影播放器那样执行开始，暂停，停止，键盘前进、后退，拖动滚动条等操作，还可通过 begin,end 两个键截取一段“骨骼视频”存成另外一个 skd 文件。

本程序主要展示了 SkeData 类有关 skd 文件读取的 API 用法。程序结构大体与 DataCollection 相同。

三、 ActionInteract

与前两个程序不同，该程序是基于 MFC 框架的。

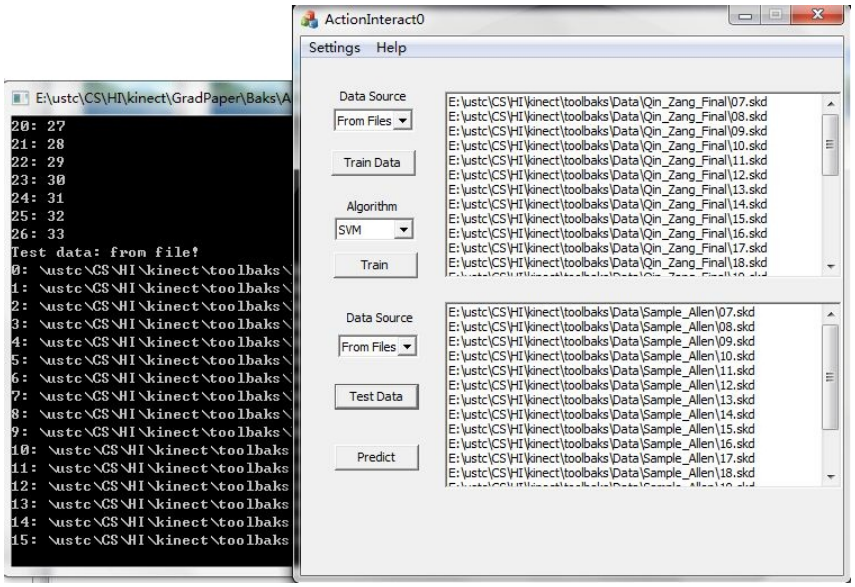


图 4.3 ActionInteract 程序主界面

通过该程序，用户可以选择训练文件和测试文件，用 SVM 或 KNN 进行分类，结果存在一个文本文档中。用户还可以指定参数（图4.4）

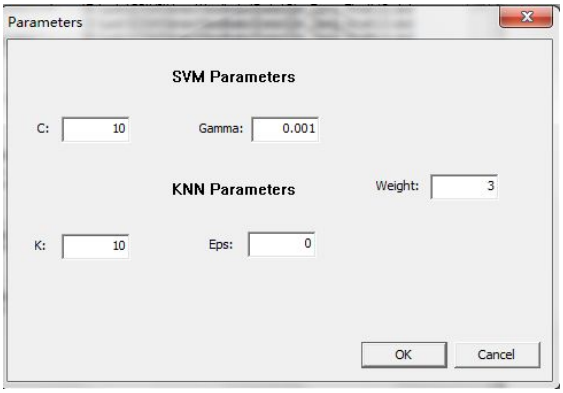


图 4.4 ActionInteract 程序参数设置

该程序运行的同时，背后还会出现一个控制台窗口，这个是用来输出各种

运行信息的，比如打开的文件，以及 SVM 训练时的输出等。用户如果想去掉它，只要将预定义的 _CONSOLEWIN 宏去掉即可。

本程序主要由 CActionInteract0App, CAboutDlg, CActionInteract0Dlg, CParametersDlg, MainModel 四大类组成，CActionInteract0App 用来对整个应用程序进行初始化和退出处理工作，CAboutDlg 主要负责“关于”对话框的显示，CParametersDlg 负责有关参数设置对话框的消息处理和内容显示，CActionInteract0Dlg 负责程序主界面相关的消息处理和内容显示，MainModel 主要负责各种后台计算，其中 ARLK 中各种分类器类的方法调用就在 MainModel 中完成，是本程序的功能核心，从 CActionInteract0Dlg 类和 CParametersDlg 类读入数据，返回处理结果。

四、 HAR_App

该程序是基于 Qt 框架在 Linux(Ubuntu 12.04) 下完成的。Qt 是一款非常流行的跨平台 C++ GUI 框架，实现了真正的模块化编程，用它开发 Linux 下应用非常合适。因此本人选择该框架，所写代码可以方便更多人的复用。

图 4.5 HAR_App 主界面

虽然在 Linux 下的示例程序只有这一个，但其功能却集成了前边三个程

序。用户可以像 DataCollection 那样设定取数据前的延迟时间，以及总帧数限制 (图4.5)。用户可以设定识别模型参数（图4.6），并且载入文件时可以对文件内容进行预览（播放, 图 4.7）。注意不同于 ActionInteract 程序，本程序的预测是实时的，即用户在做动作的同时，本程序便会同时显示预测结果（图4.8）。本程序附有一些示例动作 skd 文件，可供用户试验。

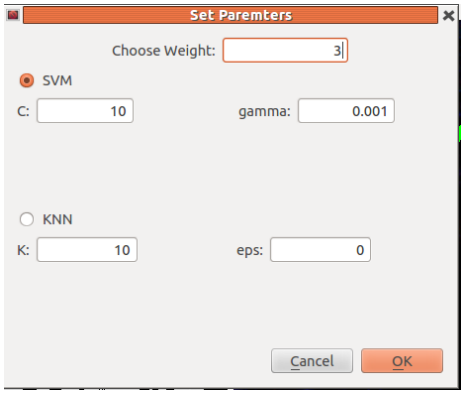


图 4.6 设参数界面

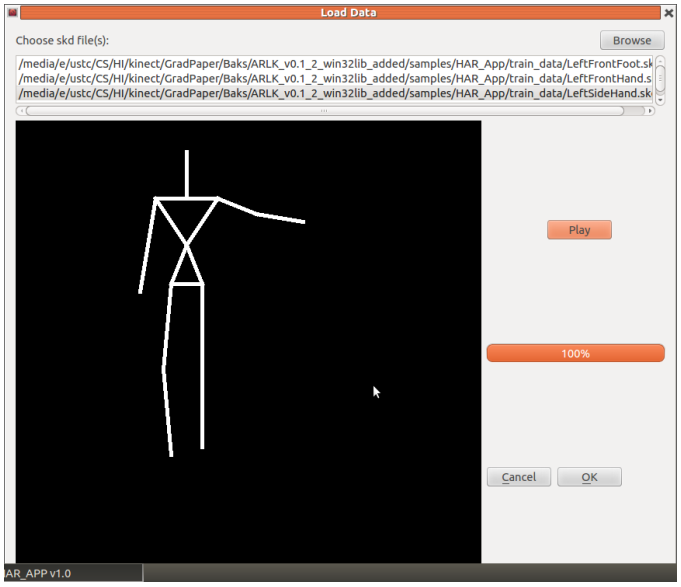
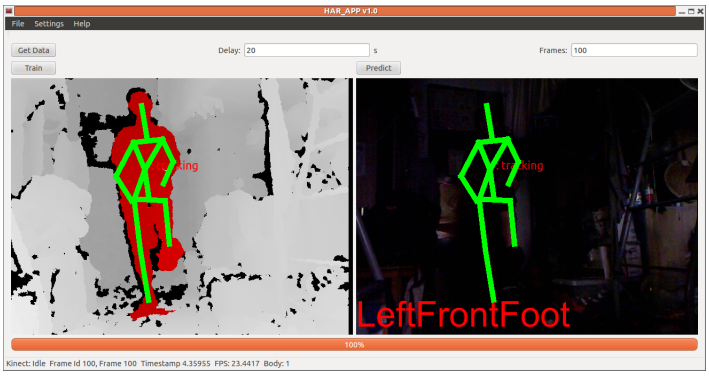


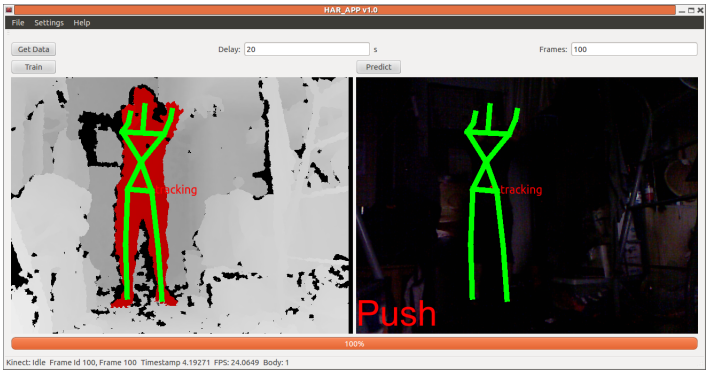
图 4.7 载入文件中数据界面

由于 Kinect SDK 只能在 Windows 下用，本程序是基于 openNI SDK 的，所以可以很好展示 ARLK 对 openNI SDK 的支持。注意本程序中，借用了网上



(a) 左腿前伸

(b) 右腿侧踢



(c) 双手推

图 4.8 实时动作检测示例

某人写的 QtKinectWrapper 类，其封装了 openNI SDK 的 API，更适于用在大型 Qt 应用程序中。本应用程序将界面与实现分离开，使得程序呈现清晰的模块

化结构，方便程序员修改界面，这也是 Qt 程序的特点之一。Ui_MainWindow, Ui_LoadDataDlg, Ui_SetParameterDlg 类分别用来描述主界面，载入数据界面和参数设置界面，MainWindow, LoadDataDlg 和 SetParameterDlg 分别用来处理界面传来的信息，及返回结果到界面。QKinectWrapper 类的对象负责读取数据，触发各种事件，供主程序处理。

第五章　总结

本次毕业设计提出了一种新的姿态识别方法：基于 Kinect 产生的骨骼数据，运用 SVM 和 KNN 分类器进行姿态识别。本方法的优势在于无需考虑背景色影响，受行为者体征影响也很小，适应力强；对于各种姿势和一些简单的手部、腿部动作，识别准确率非常高；识别速度也可以优化到很快。本文在提出本方法的同时，还给出了一个数据集，包含了多种姿势和动作，可供其他研发人员的进行实验。为了能够使得本方法得到应用，本人开发了一个开源库 ARLK，封装了本方法的实现，可用于多种应用中。该库强大的灵活性、扩展性、开源性，使得其有希望成为功能最强大的 Kinect 姿态识别库。

然而本方法也有不足之处，首先便是整个过程依赖于器材，训练、预测时的摄像头必须是 Kinect；其次是受特征提取方法的局限，目前只有针对简单手部和腿部的动作识别率才比较高，如果更加复杂的动作（如跑步，跳绳）则需要增加特征，然而这就意味着需要增加更多的模型参数。对于复杂动作的识别，选择更强大的模型是更好的选择。对于开源库 ARLK，目前发布的是第一版，在某些函数的实现上还并不是很高效，同时空间复杂度还有望进一步降低。目前该库姿态识别的功能还并不是很强大，只支持 SVM,KNN 两种算法，将来有望加入更多的识别算法和模型。

参考文献

[1] Ana Paula Brandão Lopes, Eduardo Alves do Valle Jr., Jussara Marques de Almeida, and Arnaldo de Albuquerque Araújo. Action recognition in videos: from motion capture labs to the web. *CoRR*, pages –1–1, 2010.

[2] M. Shah A. Yilmaz. Actions sketch: a novel action representation. *in:Proceedings of IEEE CVPR ' 05*, 1:984–989, 2005.

[3] M.-y. Chen C. Gao A. Bharucha A. Haupt-mann D. Chen, H. Wactlar. Recognition of aggressive human behavior using binary local motion descriptors. *in: Proceedings of IEEE EMBS ' 08*, pages 5238—5241, 2008.

[4] J. W. Davis A. F. Bobick. The recognition of human movement using temporal templates. *Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.

[5] S.-F. Chang A. Divakaran H. Sun L. Xie, P. Xu. Structure analysis of soccer video with domain knowledge and hidden markov models. *Pattern Recogn. Lett.*, page 767–775, 2004.

[6] B. Caputo C. Schuldt, I. Laptev. Recognizing human actions: a local svm approach. *in: Proceedings of IEEE ICPR ' 04*, pages 32–36, 2004.

[7] S. Lee M. Ahmad. Human action recognition using shape and clg-motion flow from multi-view image sequences. *Pattern Recognition*, (7):2237–2252, 2008.

[8] R. V. Babu V. Megavannan, B. Agarwal. Human action recognition using depth maps. *International Conference on Signal Processing and Communications (SPCOM)*, pages 1–5, 2012.

[9] B. Lange S. Rizzo E. Suma, D. Krum and M. Bolas. Faast: The flexible action and articulated skeleton toolkit. *In IEEE Virtual Reality*, pages 247–248, 2011.

[10] A. Li M. Wang Y. Liu, Z. Zhang. View independent human posture identification using kinect. 2012.

[11] A. Li M. Wang Z. Zhang, Y. Liu. A novel method for user- defined human posture recognition using kinect. *accepted by IEEECAI*, 2012.

[12] Chih-Chung Chang Chih-Wei Hsu and Chih-Jen Lin. A practical guide to support vector classification. 2010.

[13] David M.Mount. Ann programming manual. 2006.

[14] Andrew W. Moore. An into ductory tutorial on kd trees. 1991.

附录 A　数据集规范

第二章第一节提到，我们构建了一个由 27 个姿态组成的数据集，下面详细介绍每个姿态的定义以及存储形式。

表 A.1 姿势定义

No.	Name	Sample Size
1	Stand up	350
2	Left arm forward	350
3	Left arm up	350
4	Left arm out	350
5	Left arm across	350
6	Right arm forward	350
7	Right arm up	350
8	Right arm out	350
9	Right arm across	350
10	Both foot sideways	350
11	Left foot forward	350
12	Left foot sideways	350
13	Left foot up	350
14	Right foot forward	350
15	Right foot sideways	350
16	Right foot up	350

各姿态名称可参见表A.1和表A.2。姿势定义中 sample size 就是帧数，而动作定义中的 sample size 表示的是完整的动作数，虽然各个动作 sample size 各

表 A.2 动作定义

No.	Name	Sample Size
17	Left hand wave left	19
18	Left hand wave right	18
19	Right hand wave right	15
20	Right hand wave left	16
21	Push	10
22	Left side kick	11
23	Right side kick	15
24	Left front kick	14
25	Right front kick	14
26	Left hand up	18
27	Right hand up	19

不相同，但每个动作包含多帧，总帧数仍都是 350 帧，故整体仍然是平衡的数据集。

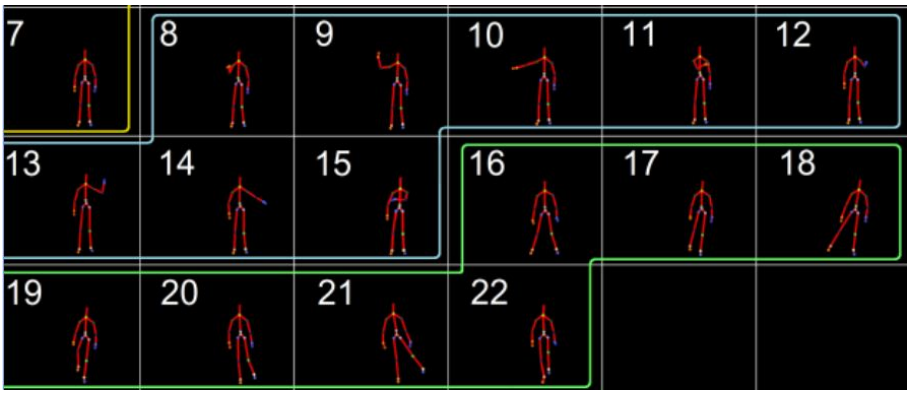


图 A.1 姿势图示 (参考 [11])

图A.1参考了此文 [11] 的定义，其中 7–22 号与本数据集中 1–16 号姿势一一对应。

图A.2即本数据集几个动作的图示。

对于数据集中每一个姿态，我们都存在一个后缀名为 skd 的文件里。这个

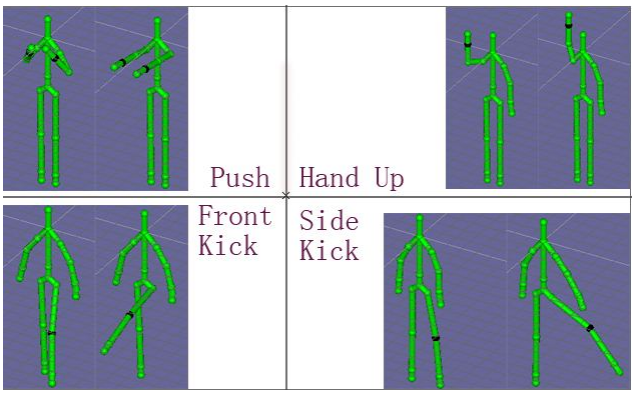


图 A.2 动作图示

文件依次由文件头、信息头和骨骼信息组成。文件头结构定义：

```
typedef struct tagSKEFILEHEADER{
    SKESHORT myfType;       //记录的视频类型（SD）
    SKESHORT myfSize;       //记录文件头大小
    SKESHORT myfReserved;   //文件保留字
    SKESHORT myfOffBits;    //偏移量
}SKEFILEHEADER;
```

信息头结构定义：

```
typedef struct tagSKEINFOHEADER{
    SKELONG     myiTime;        //视频时间长度
    SKEINT      myiFrame;       //帧数目，
    SKESHORT    jointNum;       //骨骼点数目
    SKESHORT    dimensionNum;   //骨骼信息数据维数
    SKESHORT    myiSize;        //信息头大小
    long    myiAllsize;         //整个文件大小
}SKEINFORHEADER;
```

骨骼点信息定义：

```
typedef struct tagSKDPOINT{
    short x;    //骨骼点坐标x
```

```
    short y;    //骨骼点坐标y
    short z;    //骨骼点坐标z
    short w;    //摄像头高度
    short t;    //此帧与上一帧时间差（ms）
}SKDPOINT;
```

附录 B　ARLK 使用指南

第一节　配置要求

本库目前支持两大操作系统 (Windows 和 Linux)。配置要求如下：

1. 已安装 Kinect SDK 或 openNI SDK(含 Kinect 驱动)

2. 已安装 CMake(用于编译生成 ARLK 静态库及 Windows 下示例程序解决方案)，Windows 下需已安装 Visual Studio

3. 如果想要运行示例程序，需要已安装 Visual Studio(Windows) 或 Qt(Linux)

以上要求满足后，便可下载 ARLK 到本地文件夹。根据所用的 SDK，还需要配置一下/include/configure.h 文件：如果是基于 Kinect SDK，则要保留 _ARLK_KSDK_ 宏定义；如果是基于 openNI SDK，则要保留 _ARLK_NITE_ 宏定义。该文件剩余部分定义了骨骼点坐标 id 对应的关节名，如果增加新的深度图骨骼提取算法后而导致这个对应关系的改变，可以在本文件中进行更改，但需要遵从已给出的定义格式。

第二节　静态库的生成与使用

该步骤并不是必须的，如果用户不用静态库，则需在工程中包含/include 路径, 加入/src 下所有源文件，并且包含静态库路径 (Windows 下是/win32/lib, Linux 下是/lib)。如果是 Windows 系统，则还要在环境变量中添加/win32/dll 路径，或者将该 ANN 动态链接库文件与可执行文件放在同一目录下。

如果用户选择使用 ARLK 静态库，需要用 CMake 进行构建。这里推荐使用外部构建，即构建中生成的中间文件都会放在自定义的文件夹下 (例如"build")。构建步骤如下 (设 ARLK 目录为 $(ARLK)：

1. 进入命令行控制台，cd $(ARLK)/build
2. cmake ..(Windows 下用 cmake .. -G "NMake Makefiles")
3. make(Windows 下用 nmake)

注意如果是 Windows 下，控制台最好用 Visual Studio 自带工具中的，因为该控制台下的很多环境变量已设置好，免去很多麻烦。

如果构建成功，Windows 系统的话在/win32/lib 下会生成以下几个静态库文件：SkeData.lib, BaseClassifier.lib, SVMClassifier.lib, KNNClassifier.lib, svm.lib，连同原有的 ANN.lib 共六个静态库文件。他们负责不同的功能（用户可以根据其名称找到对应的代码实现），但相互之间仍有依赖关系，如图B.1（用户可以从头文件包含关系中找出依赖关系）。A–>B 表示 A 依赖于 B。

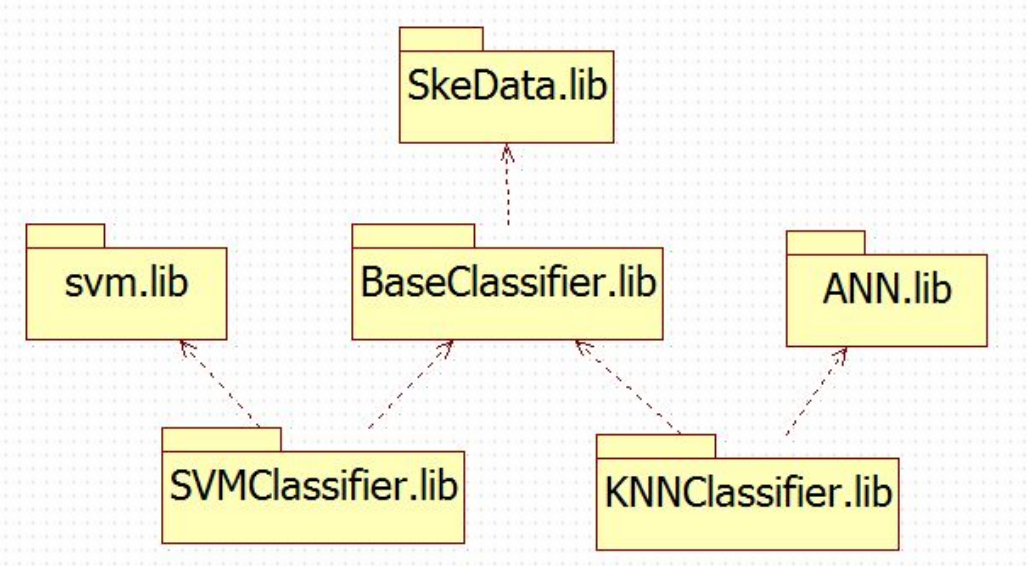


图 B.1 ARLK 静态库依赖图

Linux 下如果构建成功的话会在/lib 下生成静态库文件，基本与 Windows 下的一样，不再赘述。

实际项目中的使用步骤如下：

1. 在项目中包含 ARLK 头文件路径：$(ARLK)/include
2. 在项目中包含 ARLK 库文件路径：$(ARLK)/win32/lib(Windows), $(ARLK)/lib(Linux)
3. 添加 lib 文件至项目。
4. 如果是 Windows 下，并且需要用到 KNN 算法，则要添加 ANN.dll 所在路径至环境变量，或将其移到项目执行文件所在目录下。
5. 编译，运行

库文件可以按需加入，但添加库时一定要注意依赖关系，这点可以参考/sample 下的 HAR_App 的.pro 文件。

第三节　示例程序的构建与运行

所有示例程序都是调用 ARLK 静态库的，所以要生成静态库后再构建示例程序。

Windows 下三个示例程序在/win32/samples 下，需要用 CMake 进行项目文件还原，以 SkeletonPlayer 为例：

1. 进入命令行控制台，cd $(ARLK)/win32/samples/SkeletonPlayer/build
2. cmake ..
3. 此时 build 下生成了 Visual Studio 解决方案文件，用 Visual Studio 打开 build 下 Project.sln 文件，将 SkeletonPlayer 设为起始项目，编译后即可运行。

Linux 下示例程序构建和运行无需 CMake, 直接用 Qt 打开 /samples/HAR_App/HAR_App.pro 后即可编译，运行。

第四节　ARLK API 说明

ARLK API 说明由 doxygen 产生，请参考我的个人主页：
http://home.ustc.edu.cn/~allen90/ARLK_html/index.html