安徽工业大学

毕业设计(论文)任务书

课题名称	视觉手势识别系统
学　　院	计算机学院
专业班级	网络工程 网 093 班
姓　　名	王文胜
学　　号	099074375

毕业设计(论文)的主要内容及要求：

一、 设计(论文)目的

通过毕业设计，使学生在系统的调研、总体设计、详细设计、编程、测试等软件工程的各个阶段得到较为系统全面地的训练；培养学生编写基于 OpenCV 的 摄像头视觉手势识别系统，可通过手势赋予相应操作，并追踪手指移动，实现跟踪；培养学生的独立工作能力和快速学习新兴知识，综合运用所学知识与技能，及时 处理和解决实际问题的能力；培养学生严谨求实、实事求是的科学态度和工作作风。。

二、根据课题要求，软件完成如下功能：

1、能够正确的找到目标——人手

2、能够赋予不同手势不同动作

四、要求软件设计界面友好、使用方便，程序执行安全、可靠。

五、完成与设计内容有关的外文资料翻译，译文约 5000 字。

六、撰写毕业论文并提交设计软件及清单一份。

指导教师签字：_____

填写说明："任务书"封面请用鼠标点中各栏目横 线后将信息填入，字体设定为楷体－GB2312、四号字；在填写毕业设计（论文）内容时字体设定为楷体－GB2312、小四号字。

摘要

计算机视觉，是一个很久之前有的概念，随着科学技术的进步，大量的计算机视觉算法在硬件的支持下得到很好的验证，并投入的社会实践中的方方面面中去，其中，手势识别是计算机视觉领域中比较活跃的一个方面。虽然有无数的 IT 精英在此领域前赴后继，也取得了不少的成果，但是由于人手具有高度复杂性、灵活性，使得对于手势识别的研究仍然存在这大量的、高难度的问题。本文在旨在初步的探索手势识别，利用开源免费的计算机视觉库 opencv,在 linux 平台下实现简单的手势识别：利用肤色模型，提取目标——人手；利用 HOG(Histogram of oriented gradients 方向梯度直方图）提取图像中的信息；利用支持向量机 SVM(Support Vector Machine）算法进行图像识别的机器学习（ML）；利用 Linux 的底层驱动进行简单的动作控制。

关键词：opencv 手势识别 linux ML SVM HOG

装

订

线

Abstract

Computer vision, is a long time before some concept, with the progress of science and technology, computer vision algorithms is verified by the hardware support, and put into social practice in all aspects, including, gesture recognition is an active field in computer vision. Although there are numerous IT elite in the field of advance wave upon wave, also made many achievements, but owing to the people with a high degree of complexity, flexibility, that there is still a high difficulty, for the study of gesture recognition problem. This paper aims to explore the preliminary in gesture recognition, using open-source computer vision library opencv free, gesture recognition to achieve a simple Linux platform: the skin color model, the extraction of target -- hand; by using HOG (Histogram of oriented gradients histograms of oriented gradients) are extracted from the image information; using support vector machine SVM (Support Vector Machine) algorithm for image recognition and machine learning (ML); driving a simple motion control using Linux driven.

Key words：opencv gesture linux ML SVM HOG

装

订

线

目录

装 订 线

装

订

线

一 绪论

1.1 开发背景和意义

一直以来，视觉就只属于我们有机生物的专利，但是当我们赋予计算机眼（摄像头等）和大脑（计算机的计算能力），这一专属专利的垄断地位就被打破了。通俗的来说，计算机视觉是指计算机利用摄像头采集图像，并通过图像算法对采集的数据进行处理，由人指定有用的信息，计算机将这些有用的信息进行统计，分析，保存，然后在进行利用的过程。计算机视觉大体分为两个过程：数据采集和数据分析。

数据采集：

计算机虽然也是用“眼”来看我们的世界，但是计算机看的世界和我们人眼看的世界完全是两个世界（人的大脑直接把这一过程屏蔽了～），下图就比较直观的显示了“两个世界”

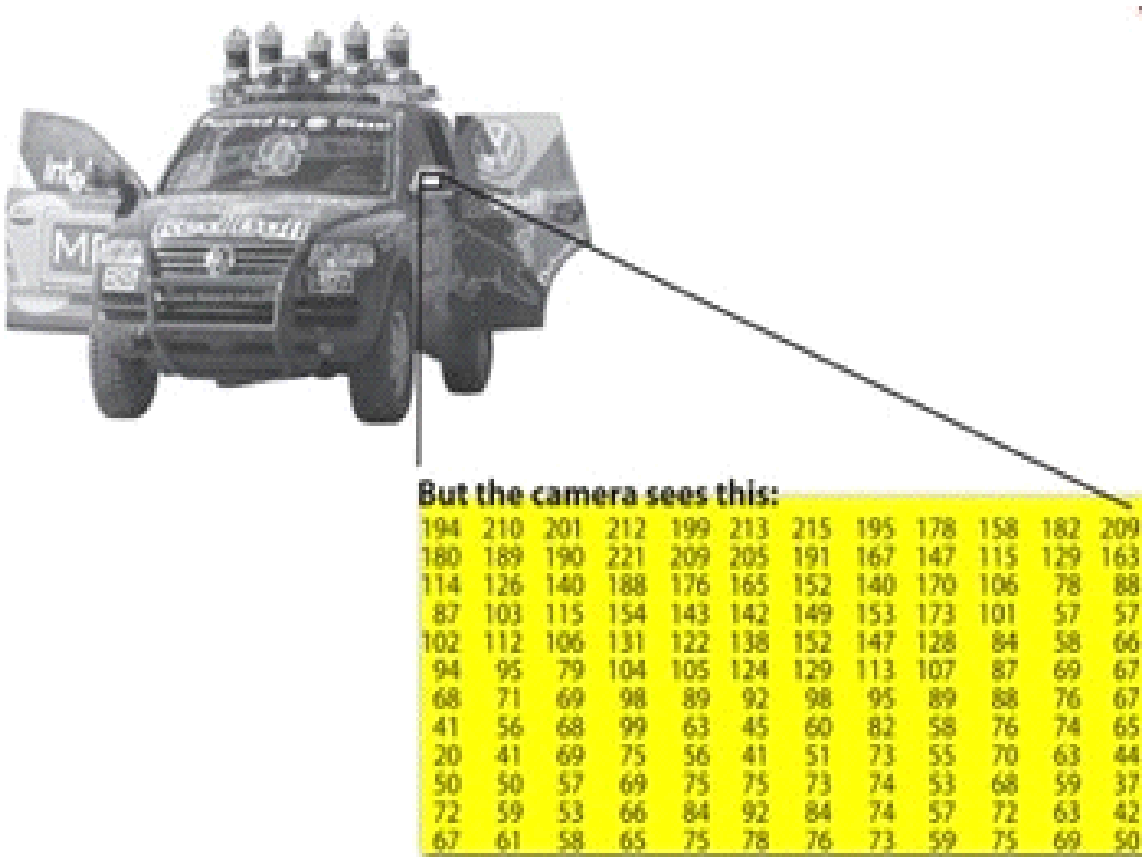


图 1.1 两个世界

如图 1.1 所示，计算机所看到的世界不过是一个数值矩阵，而不是我们所能接受的形形色色的花花世界。

数据分析：

计算机虽然有强大的计算能力，但毕竟来说，是人发明了它，它所能做的事情也只能是人指定的事情，没有人告诉它该怎么做，它也不会善做主张，事实上它也做不到。计算机采集到的数据只是一个尚未提炼的金矿，需要进一步的提取有用信息。目前对与图像处理的函数库有很多，如 opencv CxImage,CImg、FreeImage 等。

目前不少的学科研究目标就与计算机视觉相关或者相近，其中模式识别就是其中一个。目前，很多监控设备都使用了图像识别技术。日常生活中比较常见的有车牌识别、行人检测。比较活跃的研究科目有手势识别，人脸识别，这些技术的应用前景潜力巨大，而且目前的硬件水平已经达到了软件算法的需要，各大软件巨头都在此领域发力

图 1.2 微软 **Kinect** 宣传页面

装

订

线

1.2 开发环境

OpenCV 的全称是：Open Source Computer Vision Library，是有 inter 于 1999 年建立，是一个跨平台的计算机视觉库，可以运行在多个系统平台上，并提供多种

语言的接口，如 c/c++、java、python、ruby 等。其内容丰富，包含了许多图像算法，但是并未因此而变得臃肿，效率很高。
相对于其他的计算机视觉库，opencv 具有免费、开源、稳定、跨平台等优点，在做图形图像研究时候，建议使用该库。目前 opencv 最新版本为 2.4.5,其文档也比较详细，但是大多数资料为英文资料，中文资料比较少。

1.3 研究内容

本课题旨在利用 opencv 这个开源的计算机视觉库，去寻找目标——人手：通过一定的模式识别和提取算法，将目标提取出来。寻找到目标后，在进行持续的跟踪及判断手势，通过赋予手势的不同动作，来响应计算机操作，实现对于计算机简单的操控。

二 开发平台

2.1 系统平台

本课题研究采用 Linux 系统。Linux 是一种自由和开放源码的类 Unix 操作系统，存在着许多不同的 Linux 版本，但它们都使用了 Linux 内核。Linux 可安装在各种计算机硬件设备中，比如手机、平板电脑、路由器、视频游戏控制台、台式计算机、大型机和超级计算机。Linux 是一个领先的操作系统，世界上运算最快的 10 台超级计算机运行的都是 Linux 操作系统。严格来讲，Linux 这个词本身只表示 Linux 内核，但实际上人们已经习惯了用 Linux 来形容整个基于 Linux 内核，并且使用 GNU 工程各种工具和数据库的操作系统。Linux 得名于天才程序员林纳斯·托瓦兹。由于 Linux 免费开源，而且高效灵活，已经吸引了全球很大一批用户，并且还在不断的增加。本次课题采用的是基于 redhat 企业版的 Centos 6.4

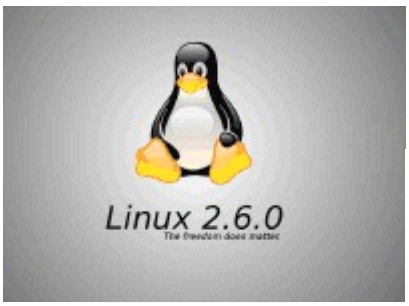


图 2.1

2.2 语言

前面介绍过了，Opencv 计算机视觉库支持很多种语言接口，如 java 、ruby、c&c++、python 等，经过本人实验，发现计算机视觉是一个比较耗计算机资源的课题，计算机须靠 cpu 在后台不停的计算，本人在实验的时候，cpu 占用率比较高，所以最终决定采用 c++作为编译语言，并利用 gcc 作为编译器。C++语言是大家所熟知的语言中的一种，其优点如下：

C++设计成静态类型、和 C 同样高效且可移植的多用途程序设计语言。

· C++设计成直接的和广泛的支持多种程序设计风格（程序化程序设计、资料抽象化、面向对象程序设计、泛型程序设计）。

· C++设计成给程序设计者更多的选择，即使可能导致程序设计者选择错误。

· C++设计成尽可能与 C 兼容，借此提供一个从 C 到 C++的平滑过渡。

· C++避免平台限定或没有普遍用途的特性。

· C++不使用会带来额外开销的特性。

· C++设计成无需复杂的程序设计环境。[8]

出于保证语言的简洁和运行高效等方面的考虑，C++的很多特性都是以库（如 STL）或其他的形式提供的，而没有直接添加到语言本身里。关于此类话题，Bjarne Stroustrup 的《C++语言的设计和演化》（1994）里做了详尽的陈述。

C++在一定程度上可以和 C 语言很好的结合，甚至目前大多数 C 语言程序是在 C++的集成开发环境中完成的。C++相对众多的面向对象的语言，具有相当高的性能。

C++引入了面向对象的概念，使得开发人机交互类型的应用程序更为简单、快捷。很多优秀的程序框架包括 MFC、QT、wxWidgets 就是使用的 C++。[

2.3 编辑工具

本次并没用采用任何 IDE 工具，采用纯文本编辑器 Emacs.Emacs 是一种强大的文本编辑器，在程序员和其他以技术工作为主的计算机用户中广受欢迎。EMACS，即 Editor MACroS（编辑器宏）的缩写，最初由 Richard Stallman(理查德·马修·斯托曼)于 1975 年在 MIT 协同 Guy Steele 共同完成。这一创意的灵感来源于 TECMAC 和 TMACS，它们是由 Guy Steele、Dave Moon、Richard Greenblatt、Charles Frankston 等人编写的宏文本编辑器。

图 2.2 Emacs

Emacs 和 Vi 一样，是在 Linux 平台下（其它平台也有）很流行的文本编辑器，以其强大的拓展性著称，至于如何配置，这里就不在介绍了。

2.4 编译器——gcc

GCC（GNU Compiler Collection，GNU 编译器集合），是一套由 GNU 开发的编程语言编译器。它是一套

GNU 编译器套装[1]

图 2.3

以 GPL 许可证所发行的自由软件，也是 GNU 计划的关键部分。GCC 原本作为 GNU 操作系统的官方编译器，现已被大多数类 Unix 操作系统（如 Linux、BSD、Mac OS X 等）采纳为标准的编译器，GCC 同样适用于微软的 Windows。[1]GCC 是自由软件过程发展中的著名例子，由自由软件基金会以 GPL 协议发布

[2]GCC 原名为 GNU C 语言编译器（GNU C Compiler），因为它原本只能处理 C 语言。GCC 很快地扩展，变得可处理 C++。之后也变得可处理 Fortran、Pascal、Objective-C、Java，以及 Ada 与其他语言。

2.5 硬件平台

CPU 英特尔 Core 2 Duo P8600 2.40 GHz
内存 4G DDR2 800 MHz
显卡 ATI Mobility Radeon HD 3470 [Sony]

装

订

线

三 基本理论知识

本课题研究的是基于 opencv 的手势识别，所以首先要做的事情就是提取目标——人手。常用 的特征描述及提取算法有很多，如基于肤色模型、基于 Harr_Like 特征、基于尺度不变 （SIFT）特征等

3.1 目标提取——肤色模型

尽管国外有过有色种人歧视的历史，不过经过研究表明，尽管人的肤色由于人种的不同而不同，但是除去亮度等环境因素后，人的皮肤的色调是基本一致的，这就为基于肤色特征提取目标的可能性提供了有力的理论依据，事实上，通过肤色模型提取目标已经成为了这一学科的研究热点。

3.1.1 YCrCb 模型

YCrCb 即 YUV，主要用于优化彩色视频信号的传输，使其向后相容老式黑白。与 RGB 视频信号传输相比，它最大的优点在于只需占用极少的频宽（RGB 要求三个独立的视频信号同时传输）。其中“Y”表示明亮度（Luminance 或 Luma），也就是灰阶值；而“U”和“V” 表示的则是色度（Chrominance 或 Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。“亮度”是透过 RGB 输入信号来建立的，方法是将 RGB 信号的特定部分叠加到一起。“色度”则定义了颜色的两个方面─色调与饱和度，分别用 Cr 和 CB 来表示。其中，Cr 反映了 RGB 输入信号红色部分与 RGB 信号亮度值之间的差异。而 CB 反映的是 RGB 输入信号蓝色部分与 RGB 信号亮度值之间的差异。

YcrCb 的优点是可以减小对光线的依赖，对于环境和肤色差异较大的环境的检测效率比较高，但是由于肤色不是皮肤特有，故不是一个特别理想的选择

3.1.2 HSV 和 HSL 颜色模型

HSL 和 **HSV**（也叫 **HSB**）是对 RGB 中点的两种有关系的表示，它们尝试描述比 RGB 更准确的感知颜色联系，并仍保持在计算上简单。

H 指 **h**ue（色相）、S 指 **s**aturation（饱和度）、L 指 **l**ightness（亮度）、V 指 **v**alue(色调)、B 指 **b**rightness（明度）。

色相（H）是色彩的基本属性，就是平常所说的颜色名称，如红色、黄色等。

饱和度（S）是指色彩的纯度，越高色彩越纯，低则逐渐变灰，取 0-100%的数值。

明度（V），亮度（L），取 0-100%

HSL 和 HSV 二者都把颜色描述在圆柱坐标体系内的点，这个圆柱的中心轴取值为自底部的黑色到顶部的白色而在它们中间是的灰色，绕这个轴的角度对应于“色相”，到这个轴的距离对应于“饱和度”，而沿着这个轴的高度对应于“亮度”，“色调”或“明度”。

这两种表示在用目的上类似，但在方法上有区别。二者在数学上都是圆柱，但 HSV（色相饱和度，色调）在概念上可以被认为是颜色的倒圆锥体（黑点在下顶点，白色在上底面圆心），HSL 在概念上表示了一个双圆锥体和圆球体（白色在上

顶点，黑色在下顶点，最大横切面的圆心是半程灰色）。注意尽管在 HSL 和 HSV 中“色相”指称相同的性质，它们的“饱和度”的定义是明显不同的。

因为 HSL 和 HSV 是设备依赖的 RGB 的简单变换，(*h*, *s*, *l*) 或 (*h*, *s*, *v*) 三元组定义的颜色依赖于所使用的特定红色、绿色和蓝色“加法原色”。每个独特的 RGB 设备都伴随着一个独特的 HSL 和 HSV 空间。但是 (*h*, *s*, *l*) 或 (*h*, *s*, *v*) 三元组在被约束于特定 RGB 空间比如 sRGB 的时候就变成明确的了。

HSV 模型在 1978 年由埃尔维·雷·史密斯创立，它是三原色光模型的一种非线性变换。

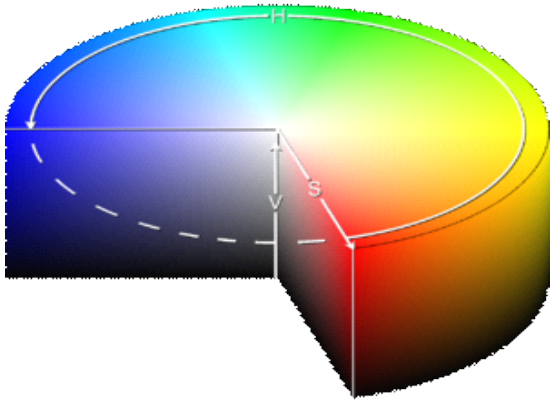


图 3.1 HSV

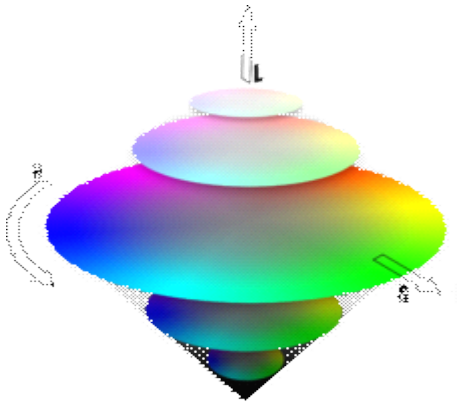


图 3.2 HSL

从 HSV 到 RGB 的转换

类似的，给定在 HSV 中 (*h*, *s*, *v*) 值定义的一个颜色，带有如上的 *h*，和分别表示饱和度和明度　的 *s* 和 *v* 变化于 0 到 1 之间，在 RGB 空间中对应的 (*r*, *g*, *b*) 三原色可以计算为(R,G,B 变化于　　0 到 1 之间)：

装 订 线

$$h_i \equiv \left\lfloor \frac{h}{60} \right\rfloor \pmod 6$$
$$f = \frac{h}{60} - h_i$$
$$p = v \times (1 - s)$$
$$q = v \times (1 - f \times s)$$
$$t = v \times (1 - (1 - f) \times s)$$

对于每个颜色向量 (r, g, b),

$$(r,g,b) = \begin{cases} (v,t,p), & \text{if } h_i = 0 \\ (q,v,p), & \text{if } h_i = 1 \\ (p,v,t), & \text{if } h_i = 2 \\ (p,q,v), & \text{if } h_i = 3 \\ (t,p,v), & \text{if } h_i = 4 \\ (v,p,q), & \text{if } h_i = 5 \end{cases}$$

装

订

线

本人在做此课题的时候，在网上搜集的资料都建议使用 YCrCb 去分离肤色，说 YCrCb 的分离效果最好，不过经过本人的验证，发现 HSV 肤色模型的分离效果更好点，可能是本人 的技术不够的问题吧。本人接下来肤色分离使用的就是 HSV 颜色空间肤色提取。HSV 颜色空间提取肤色原理如下：使用 HSV 颜色空间中的 H 和 S 空间。首先利用 opencv 库中的函数将从摄像头中采集到的三通道图分离到 H、S、V 空间中去，这样三通道图像就被分离到三个单通道图中去，因为只利用 H 和 S 空间，所以直接将 V 空间的数据抛弃。修改 H、S 空间值后，将 H、S 空间的图像再次合并到一起，这样就形成了新的图像。由于修改了 H 和 S 的值，所以在环境较好的情况下，能够顺利的将皮肤提取出来

3.1.3 其它特征模型简介

Haar_Like 特征

Haar-like 特征最早是由 Papageorgiou 等应用于人脸表示。Papageorgiou 在针对正面人脸和人体检测问题的研究中使用 Haar 小波基函数，他们发现标准正交 Haar 小波基在应用上受到一定的限制，为了取得更好的空间分辨率，他们使用了 3 种类型的 3 种形式的特征。Viola 等在此基础上作了扩展，使用 2 种类型 4 种形式的特征。3 种类型分别为：2-矩形特征、3-矩形特征、4-矩形特征。

SIFT 特征（Scale-invariant feature transform，尺度不变特征转换)

SIFT 特征是 一种计算机视觉的算法用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量，此算法由 David Lowe 在 1999 年所发表，2004 年完善总结。其应用范围包含物体辨识、机器人地图感知与导航、影像缝合、3D 模型建立、手势辨识、影像追踪和动作比对。此算 法有其专利，专利拥有者为 英属哥伦比亚大学。

局部影像特征的描述与侦测可以帮助辨识物体，SIFT 特征是基于物体上的一些局部外观的兴趣点而与影像的大小和旋转无关。 对于光线、噪声、些微视角改变的容忍度也相当高。基于这些特性，它们是高度显著而且相对容易撷取，在母数庞大的特征数据库中，很容易辨识物体而且鲜有误 认。使用 SIFT 特征描述对于部分物体遮蔽的侦测率也相当高，甚至只需要 3 个以上的 SIFT 物体特征就足以计算出位置与方位。在现今的电脑硬件速度下和小型的特征 数据库条件下，辨识速度可接近即时运算。SIFT 特征的信息量大，适合在海量数据库中快速准确匹配。

3.2 手势识别

3.2.1 基于模板的手势识别

基于模板的手势识别，原理比较简单，其流程如下

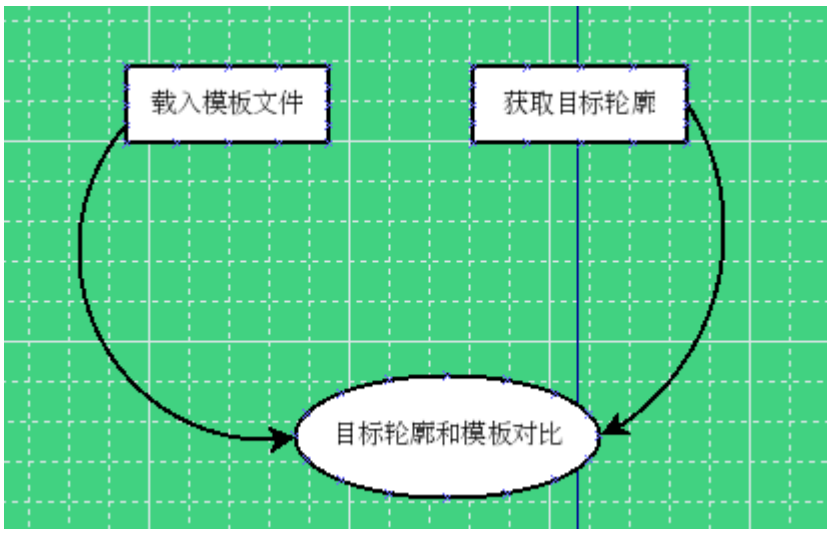


图 3.1

如图，目标轮廓和模板对比会产生一个匹配系数，这个系数和采用的算法有关。模板和目标的相似度就体现在匹配系数上。其主要利用如下函数：

```
double cvMatchShapes(
        const void* object1,
        const void* object2,
        int method,
        double parameter = 0
);
```

第一个参数是待匹配的物体 1，第二个是待匹配的物体 2

第三个参数 method 可以有

CV_CONTOURS_MATCH_I1CV_CONTOURS_MATCH_I2，CV_CONTOURS_MATCH_I3 三种输入，这其实就是三种不同的判定物体相似的方法，匹配的返回值也是不一样的其返回值分别通过各自的公式计算得出,根据这个返回值的大小，我们可以得出两个目标的相似度，限制相似度大小即可判断是否是目标。

这个方法可行，但是识别率比较低,实际中，我先采用了这种方法，效果不是很好，所以改换了另外一种方法

3.2.2 机器学习算法（ML)

机器学习是近 20 多年兴起的一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算　法复杂度理论等多门学科。机器学习理论主要是设计和分析一些让计算机可以自动“学习”的算法。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。因为学习算法中涉及了大量的统计学理论，机器学习与统计推断学联系尤为密切，也被称为统计学习理论。算法设计方面，机器学习理论关注可以实现的，行之有效的学习算法。很多推论问题属于无程序可循难度，所以部分的机器学习研究是开发容易处理的近似算法。机器学习已经有了十分广泛的应用，例如：数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA 序列测序、语音和手写识别、战略游戏机器人运用。

训练集和测试集

对于我们来说，区分一般的目标是比较容易的，因为从小到大我们接触了各种各样的目标。比如我在桌子上放置了一个杯子，你会辨认出来，而当我把杯子的颜色或者位置形状变换掉后，你依然可以清楚的识别杯子这个目标。对于机器来说，它们和我们一样，需要大量的样本供它去学习，去分析，在大量的数据中寻找规律。这个大量的数据就是训练集。计算机从训练集中的样本中提取特征，如边缘方向，边缘长度等，利用训练集中提取到的数据特征，进行数据建模。如果此时我们只是简单的分类，那么一个类聚算法就足够了，如果想要去预测其他的特征，那么我们就需要一个分类算法了，为了达到这个目的，机器学习算法分析我们收集的数据，分配权重，阙值和其他参数来提高性能，，这个过程就是“学习” 了。

训练完毕后，我们不能够马上投入使用，我们还应该进行测试。这时我们就需要另外一个数据集——测试集。通过记录分类器的预测结果，和真实的情况进行对比，可以初步得到分类器的好坏。如果没有达到要求，那么，我们就应该继续训练，或者换一个分类算法了。

监督数据和无监督数据

有时候我们只是想根据特征信息看看目标是属于哪一组的。有的时候，数据有标签，如年龄，这类的有标签的机器学习就是有监督的。数据向量没有标签则表明，这中机器学习是无监督的。

有监督的机器学习可以用来判断种类，如把名字和种类对应起来，或者赋予一个数字标签。当数据以名字作为标签，则表明我们在做分类；如果输出的是数值，则表明我们在进行回归，回归是通过类别的或者数值的输入数据来你和一个数值输出。

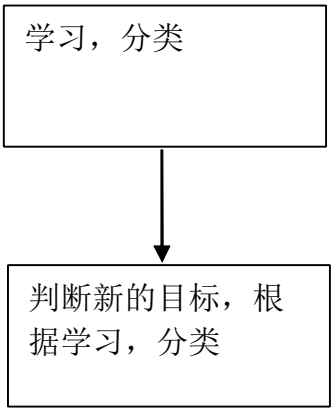
生成模型和判别模型

判别模型：opencv 中倾向于判别模型，通过给定的数据进行判别，利用条件判断（P（A|B））

生成模型：对数据的产生进行建模

目前已经有很多的计算机学习算法，本次采用的是支持向量机 SVM(Support Vector Machine）机器学习算法，故一下着重介绍 SVM 算法，其他算法只做简单介绍，有兴趣的读者可以自己去查阅相关资料

通常来说机器算法的流程为



3.2.2.1 机器学习算法——支持向量机 SVM(Support Vector Machine）

先简单说明下分类器

如下图，有很多的点

图 3.3

现在我们要求用线条给这两个颜色点分类，当然，我们有很多种方法去分类

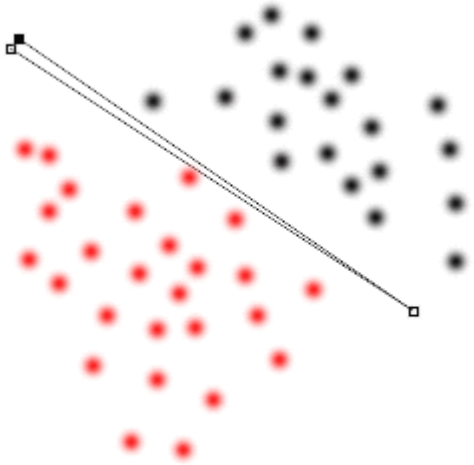


图 3.4

但是，如何去找到最优的划分才是最难的，我们利用另一种方法，对直线的正负偏移量 1，这样就产生了一个区域（下图的 Maximum margin 覆盖的区域），区域边界上的点到直线的距离是固定的，现在的问题是最近的点是否刚好在边界上或者在边界外。

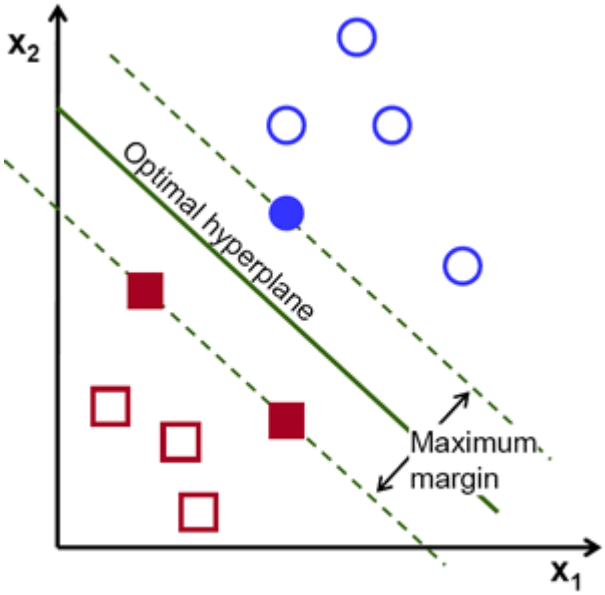


图 3.5

直线 Ax＋By＋C＝0，点（x_0, y_0）到直线的距离：

distance = $|Ax_0+By_0+C| / (A^2 + B^2)^{1/2}$

那么区域边缘到直线的距离：

distance = $(|Ax+By+C| + 1)/ (A^2 + B^2)^{1/2} = 1/ (A^2 + B^2)^{1/2}$。

并需要满足对于所有样本类别 y_i 满足：yi (Ax+By+C) > = 1，也就是所有样本都不在该区域以内。

于是我们可以找到适当的 A、B、C，从而得到：

Maximum margin = $2/ (A^2 + B^2)^{1/2}$。

以上只是对于现行可分情况进行简单描述，至于线性不可分情况，目前未能理解，这里就不在说明了。

支持向量机 SVM(Support Vector Machine）作为一种可训练的机器学习方法，依靠小样本学习后的模型参数进行导航星提取，可以得到分布均匀且恒星数量大为减少的导航星表。 Vapnik 等人在多年研究统计学习理论基础上对线性分类器提出了另一种设计最佳准则。其原理也从线性可分说起，然后扩展到线性不可分的情况。甚至扩展到使用非线性函数中去，这种分类器被称为支持向量机（Support Vector Machine，简称 SVM）。支持向量机的提出有很深的理论背景。在 opencv 中，SVM 主要涉及到一下内容：

SVM 参数 CvSVMParams

***struct* CvSVMParams**

SVM 训练参数结构。

该结构必须被初始化后，传给 CvSVM。

CvSVMParams::CvSVMParams

构造函数

C++: CvSVMParams::CvSVMParams()

C++: CvSVMParams::CvSVMParams(int **svm_type**, int **kernel_type**, double **degree**, double **gamma**, double **coef0**, double **Cvalue**, double **nu**, double **p**, CvMat* **class_weights**, CvTermCriteria **term_crit**)

参数

- **svm_type** –

指定 SVM 的类型，下面是可能的取值：

- **CvSVM::C_SVC** C 类支持向量分类机。 n 类分组 (n $\geq$ 2)，允许用异常值惩罚因子 C 进行不完全分类。
- **CvSVM::NU_SVC** 类支持向量分类机。n 类似然不完全分类的分类器。参数为 ν 取代 C（其值在区间【0，1】中，nu 越大，决策边界越平滑）。
- **CvSVM::ONE_CLASS** 单分类器，所有的训练数据提取自同一个类里，然后 SVM 建立了一个分界线以分割该类在特征空间中所占区域和其它类在特征空间中所占区域。
- **CvSVM::EPS_SVR** ϵ类支持向量回归机。训练集中的特征向量和拟合出来的超平面的距离需要小于 p。异常值惩罚因子 C 被采用。
- **CvSVM::NU_SVR** ν类支持向量回归机。 ν 代替了 p。

可从 [LibSVM] 获取更多细节。

- **kernel_type** –

SVM 的内核类型，下面是可能的取值：

- **CvSVM::LINEAR** 线性内核。没有任何向映射至高维空间，线性区分（或回归）在原始特征空间中被完成，这是最快的选择。$K(x_i, x_j) = x_i^T x_j$
- **CvSVM::POLY** 多项式内核：
$K(x_i, x_j) = (\gamma x_i^T x_j + coef0)^{degree}, \gamma > 0$
- **CvSVM::RBF** 基于径向的函数，对于大多数情况都是一个较好的选择：$K(x_i, x_j) = e^{-\gamma ||x_i - x_j||^2}, \gamma > 0$
- **CvSVM::SIGMOID** Sigmoid 函数内核：
$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + coef0)$

- **degree** – 内核函数（POLY）的参数 degree。
- **gamma** – 内核函数（POLY/ RBF/ SIGMOID）的参数γ。
- **coef0** – 内核函数（POLY/ SIGMOID）的参数 coef0。

- **Cvalue** – SVM 类型（C_SVC/ EPS_SVR/ NU_SVR）的参数 C。
- **nu** – SVM 类型（NU_SVC/ ONE_CLASS/ NU_SVR）的参数 ν。
- **p** – SVM 类型（EPS_SVR）的参数 ϵ。
- **class_weights** – C_SVC 中的可选权重，赋给指定的类，乘以 C 以后变成 $class\_weights_i * C$。所以这些权重影响不同类别的错误分类惩罚项。权重越大，某一类别的误分类数据的惩罚项就越大。
- **term_crit** – SVM 的迭代训练过程的中止条件，解决部分受约束二次最优问题。您可以指定的公差和/或最大迭代次数。

Opencv 中 SVM 算法所涉及的参数，基本上都在上面了

***class* CvSVM**

向量支持机

CvSVM::CvSVM

训练构造函数。

C++: CvSVM::CvSVM()

C++: CvSVM::CvSVM(const Mat& **trainData**, const Mat& **responses**, const Mat& **varIdx**=Mat(), const Mat& **sampleIdx**=Mat(), CvSVMParams**params**=CvSVMParams())

C++: CvSVM::CvSVM(const CvMat* **trainData**, const CvMat* **responses**, const CvMat* **varIdx**=0, const CvMat* **sampleIdx**=0, CvSVMParams**params**=CvSVMParams())

参数

- **trainData —**训练数据，必须是 CV_32FC1 （32位浮点类型，单通道）。数据必须是 CV_ROW_SAMPLE 的，即特征向量以行来存储。
- **responses —**响应数据，通常是1D 向量存储在 CV_32SC1 （仅仅用在分类问题上）或者 CV_32FC1格式。
- **varIdx —** 指定感兴趣的特征。可以是整数(32sC1)向量，例如以0为开始的索引，或者8位(8uC1)的使用的特征或者样本的掩码。用户也可以传入 NULL 指针，用来表示训练中使用所有变量 / 样本。
- **sampleIdx —** 指定感兴趣的样本。描述同上。
- **params —**SVM 参数。

CvSVM::train

训练一个 SVM。

C++: bool CvSVM::train(const Mat& **trainData**, const Mat& **responses**, const Mat& **varIdx**=Mat(), const Mat& **sampleIdx**=Mat(), CvSVMParams**params**=CvSVMParams())

C++: bool CvSVM::train(const CvMat* **trainData**, const CvMat* **responses**, const CvMat* **varIdx**=0, const CvMat* **sampleIdx**=0, CvSVMParams**params**=CvSVMParams())

参数参考构造函数。

CvSVM::train_auto

根据可选参数训练一个 SVM。

C++: bool CvSVM::train_auto(const Mat& **trainData**, const Mat& **responses**, const Mat& **varIdx**, const Mat& **sampleIdx**, CvSVMParams**params**, int **k_fold**=10, CvParamGrid **Cgrid**=CvSVM::get_default_grid(CvSVM::C), CvParamGrid**gammaGrid**=CvSVM::get_default_grid(CvSVM::GAMMA), CvParamGrid

pGrid=CvSVM::get_default_grid(CvSVM::P), CvParamGrid**nuGrid**=CvSVM::get_default_grid(CvSVM::NU), CvParamGrid **coeffGrid**=CvSVM::get_default_grid(CvSVM::COEF), CvParamGrid**degreeGrid**=CvSVM::get_default_grid(CvSVM::DEGREE), bool **balanced**=false)

C++: bool CvSVM::train_auto(const CvMat* **trainData**, const CvMat* **responses**, const CvMat* **varIdx**, const CvMat* **sampleIdx**, CvSVMParams **params**, int **kfold**=10, CvParamGrid **Cgrid**=get_default_grid(CvSVM::C), CvParamGrid **gammaGrid**=get_default_grid(CvSVM::GAMMA), CvParamGrid **pGrid**=get_default_grid(CvSVM::P), CvParamGrid **nuGrid**=get_default_grid(CvSVM::NU), CvParamGrid**coeffGrid**=get_default_grid(CvSVM::COEF), CvParamGrid **degreeGrid**=get_default_grid(CvSVM::DEGREE), bool **balanced**=false)

参数

- **k_fold** – 交叉验证参数。训练集被分成 k_fold 的自子集。其中一个子集是用来测试模型，其他子集则成为训练集。所以，SVM 算法复杂度是执行 k_fold 的次数。
- ***Grid** – 对应的 SVM 迭代网格参数。
- **balanced** – 如果是 true 则这是一个2类分类问题。这将会创建更多的平衡交叉验证子集。

这个方法根据 CvSVMParams 中的最佳参数 C, gamma, p, nu, coef0, degree 自动训练 SVM 模型。参数被认为是最佳的交叉验证，其测试集预估错误最小。

如果没有需要优化的参数，相应的网格步骤应该被设置为小于或等于1的值。例如，为了避免 gamma 的优化，设置 gamma_grid.step = 0，gamma_grid.min_val， gamma_grid.max_val 为任意数值。所以 params.gamma 由 gamma 得出。

最后，如果参数优化是必需的，但是相应的网格却不确定，你可能需要调用函数 CvSVM::get_default_grid()，创建一个网格。例如，对于 gamma，调用 CvSVM::get_default_grid(CvSVM::GAMMA)。

该函数为分类运行 (params.svm_type=CvSVM::C_SVC 或者 params.svm_type=CvSVM::NU_SVC) 和为回归运行 (params.svm_type=CvSVM::EPS_SVR 或者 params.svm_type=CvSVM::NU_SVR)效果一样好。如果 params.svm_type=CvSVM::ONE_CLASS，没有优化，并指定执行一般的 SVM。

CvSVM::predict

预测样本的相应数据。

C++: float CvSVM::predict(const Mat& **sample**, bool **returnDFVal**=false) const

C++: float CvSVM::predict(const CvMat* **sample**, bool **returnDFVal**=false) const

C++: float CvSVM::predict(const CvMat* **samples**, CvMat* **results**) const

参数

- **sample** – 需要预测的输入样本。
- **samples** – 需要预测的输入样本们。
- **returnDFVal** – 指定返回值类型。如果值是 true，则是一个2类分类问题，该方法返回的决策函数值是边缘的符号距离。
- **results** – 相应的样本输出预测的响应。

这个函数用来预测一个新样本的响应数据(response)。在分类问题中，这个函数返回类别编号；在回归问题中，返回函数值。输入的样本必须与传 给 trainData 的训练样本同样大小。如果训练中使用了 varIdx 参数，一定记住在 predict 函数中使用跟训练特征一致的特征。

后缀 const 是说预测不会影响模型的内部状态，所以这个函数可以很安全地从不同的线程调用。

CvSVM::get_default_grid

生成一个 SVM 网格参数。

C++: CvParamGrid CvSVM::get_default_grid(int **param_id**)

参数

- **param_id** –

SVM 参数的 IDs 必须是下列中的一个：

- **CvSVM::C**
- **CvSVM::GAMMA**
- **CvSVM::P**
- **CvSVM::NU**
- **CvSVM::COEF**
- **CvSVM::DEGREE**

网格参数将根据这个 ID 生成。

CvSVM::get_params

返回当前 SVM 的参数。

C++: CvSVMParams CvSVM::get_params() const

这个函数主要是在使用 CvSVM::train_auto()时去获得最佳参数。

CvSVM::get_support_vector

检索一定数量的支持向量和特定的向量。

C++: int CvSVM::get_support_vector_count() const

C++: const float* CvSVM::get_support_vector(int **i**) const

参数 **i** – 指定支持向量的索引。

该方法可以用于检索一组支持向量。

CvSVM::get_var_count

返回变量的个数。

C++: int CvSVM::get_var_count() const

如上介绍，opencv 中 SVM 算法是比较复杂的一种，想要完全弄懂这个算法，需要扎实的数学功底，虽然本次利用了这个算法，但是也只是限于使用，对于理论了解并不深，希望以后可以搞清楚。

3.2.3 方向梯度直方图（Histogram of Oriented Gradient, HOG）特征

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。Hog 特征结合 SVM 分类器已经被广泛应用于图像识别中，尤其在行人检测中获得了极大的成功。需要提醒的是，HOG+SVM 进行行人检测的方法是法国研究人员 Dalal 在2005的 CVPR 上提出的，而如今虽然有很多行人检测算法不断提出，但基本都是以 HOG+SVM 的思路为主。本次课题研究采用也采用了 HOG+SVM，用来检测手势。

原理：HOG 的核心思想是所检测的局部物体外形能够被光强梯度或边缘方向的分布所描述。 通过将整幅图像分割成小的连接区域（称为 cells），每个 cell 生成一个方向梯度直方图或

者 cell 中 pixel 的边缘方向，这些直方图的组合可表示 出（所检测目标的目标）描述子。为改善准确率，局部直方图可以通过计算图像中一个较大区域(称为 block)的光强作为 measure 被对比标准化，然后 用这个值(measure)归一化这个 block 中的所有 cells.这个归一化过程完成了更好的照射/阴影不变性。过程如下：

HOG 特征提取方法就是将一个 image：

1. 灰度化（将图像看做一个 x,y,z（灰度）的三维图像）
2. 划分成小 cells（2*2）
3. 计算每个 cell 中每个 pixel 的 gradient（即 orientation）
4. 统计每个 cell 的梯度直方图（不同梯度的个数），即可形成每个 cell 的 descriptor
5. 将若干干 cell 组成 block
6. 将若干 block 组成 image,就得到目标的 HOG 特征了

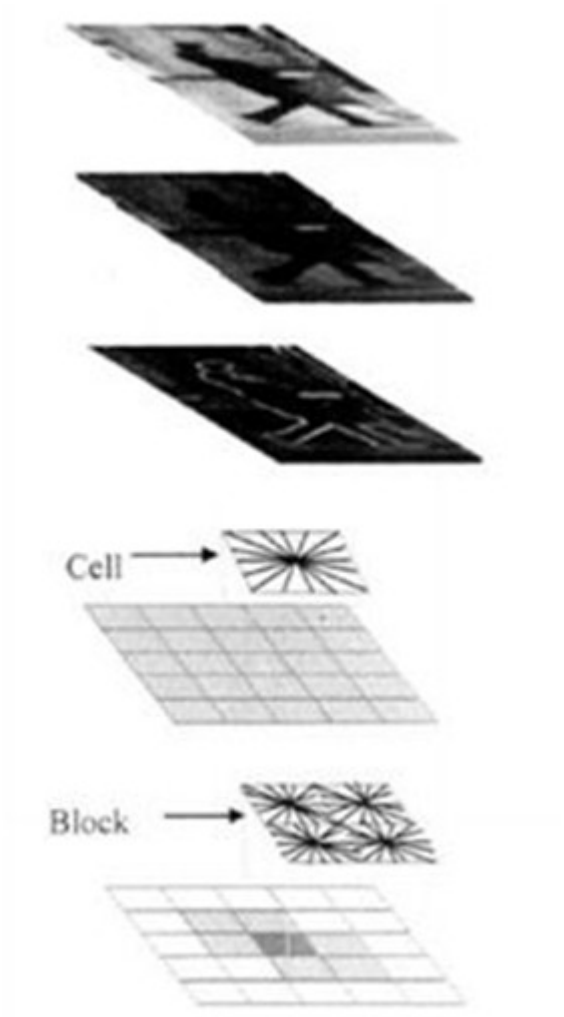


图 3.6

下面介绍 HOGDescriptor 类的构造函数

构造1. CV_WRAP HOGDescriptor() : winSize(64,128), blockSize(16,16),
blockStride(8,8),
cellSize(8,8), nbins(9), derivAperture(1), winSigma(-1),
histogramNormType(HOGDescriptor::L2Hys), L2HysThreshold(0.2),
gammaCorrection(true),
nlevels(HOGDescriptor::DEFAULT_NLEVELS)
{}
构造2.
CV_WRAP HOGDescriptor(Size _winSize, Size _blockSize, Size _blockStride,
Size _cellSize, int _nbins, int _derivAperture=1, double _winSigma=-1,
int _histogramNormType=HOGDescriptor::L2Hys,
double _L2HysThreshold=0.2, bool _gammaCorrection=false,
int _nlevels=HOGDescriptor::DEFAULT_NLEVELS)
: winSize(_winSize), blockSize(_blockSize), blockStride(_blockStride),
cellSize(_cellSize),
nbins(_nbins), derivAperture(_derivAperture), winSigma(_winSigma),
histogramNormType(_histogramNormType), L2HysThreshold(_L2HysThreshold),
gammaCorrection(_gammaCorrection), nlevels(_nlevels)
{}
构造3.
CV_WRAP HOGDescriptor(const String& filename)
{
load(filename);
}
构造4.
HOGDescriptor(const HOGDescriptor& d)
{
d.copyTo(*this);
}

参数如下：
winSize

装 订 线

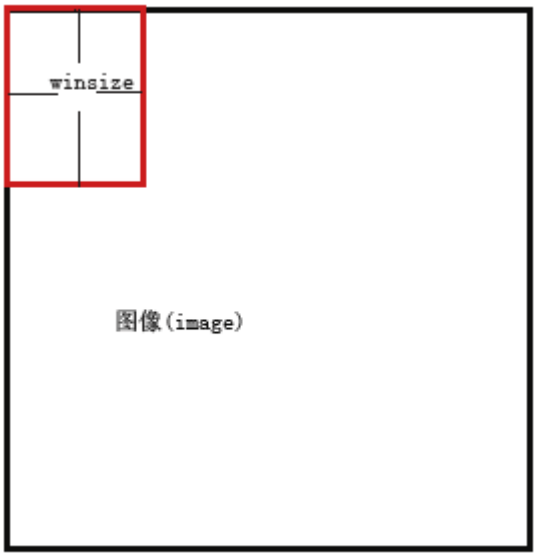


图 3.6 winSize 和图像关系

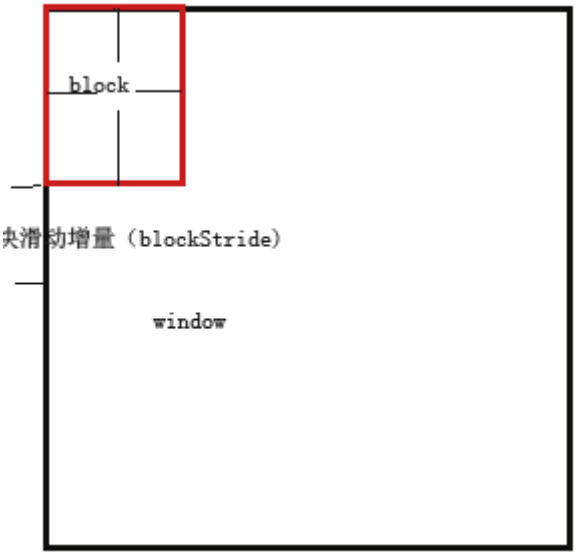


图 3.7 块和 winsize 关系

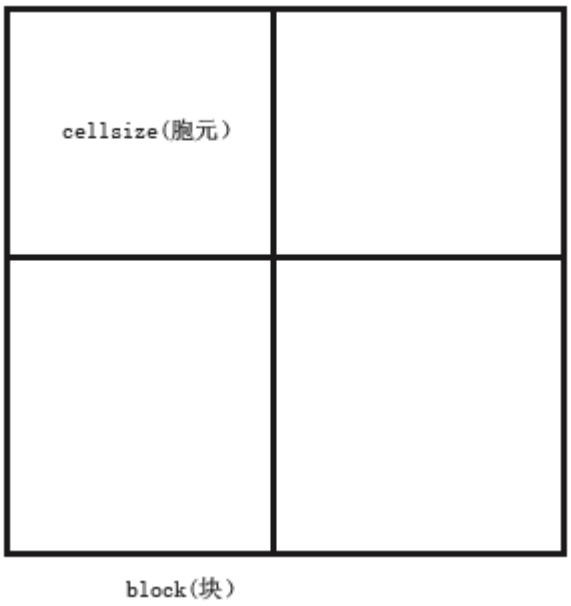


图 3.8 胞元和块关系

3.3 如何为手势添加动作——linux 底层驱动

本课题是为手势添加一定的简单动作，从而对电脑进行简单的操作,由于Linux 中的设备也是文件，所以想要模拟硬件操作的话，可以直接向相应的设备文件进行写数据操作。有关设备对应关系，可以利用 cat /proc/bus/input/device 命令在终端中查看设备。模拟按键的时候，可以查看“/linux/input.h”头文件中的对应。本次模拟的是鼠标动作和按键动作，相关操作就是向相应的设备文件写入如下的结构体：

```
struct input_event {
    struct timeval time; //按键时间
    __u16 type; //类型，在下面有定义
    __u16 code; //要模拟成什么按键
    __s32 value;//是按下还是释放
    };
```

code：

事件的代码。如果事件的类型代码是 EV_KEY,该代码 code 为设备键盘代码。代码植0~127 为键盘上的按键代码，0x110~0x116 为鼠标上按键代码，其中0x110(BTN_ LEFT)为鼠标左键，0x111(BTN_RIGHT)为鼠标右键，0x112(BTN_ MIDDLE)为鼠标中键。其它代码含义请参看 include/linux/input.h 文件。 如果事件的类型代码是 EV_REL,code 值表示轨迹的类型。如指示鼠标的 X 轴方向 REL_X(代码为0x00)，指示鼠标的 Y 轴方向 REL_Y(代码 为0x01)，指示鼠标中轮子方向 REL_WHEEL(代码为0x08)。

type:

EV_KEY,键盘

EV_REL,相对坐标

EV_ABS,绝对坐标

value：

事件的值。如果事件的类型代码是 EV_KEY,当按键按下时值为1,松开时值为0;如果事件的类型代码是 EV_ REL,value 的正数值和负数值分别代表两个不同方向的值。

```
/*
* Event types
*/
#define EV_SYN 0x00
#define EV_KEY 0x01 //按键
#define EV_REL 0x02 //相对坐标(轨迹球)
#define EV_ABS 0x03 //绝对坐标
#define EV_MSC 0x04 //其他
#define EV_SW 0x05
#define EV_LED 0x11 //LED
#define EV_SND 0x12//声音
#define EV_REP 0x14//repeat
#define EV_FF 0x15
#define EV_PWR 0x16
#define EV_FF_STATUS 0x17
#define EV_MAX 0x1f
#define EV_CNT (EV_MAX+1)
```

通过查看 input.h 中的相关对应，我们可以实现相关按键的输入，模拟鼠标的原理是一样的。

特别注意，Opencv 是一个跨平台的计算机视觉库，如果想要移植的话，则需要把 linux 底层驱动这一部分给去除掉。

装 订 线

四 需求分析

4.1 概述

本课题研究的是手势识别，故此需要提取手势和识别手势，并赋予手势相应的动作。

4.2 功能分析

这里需要程序完成以下功能：

A 手势的提取

B 手势的分类训练

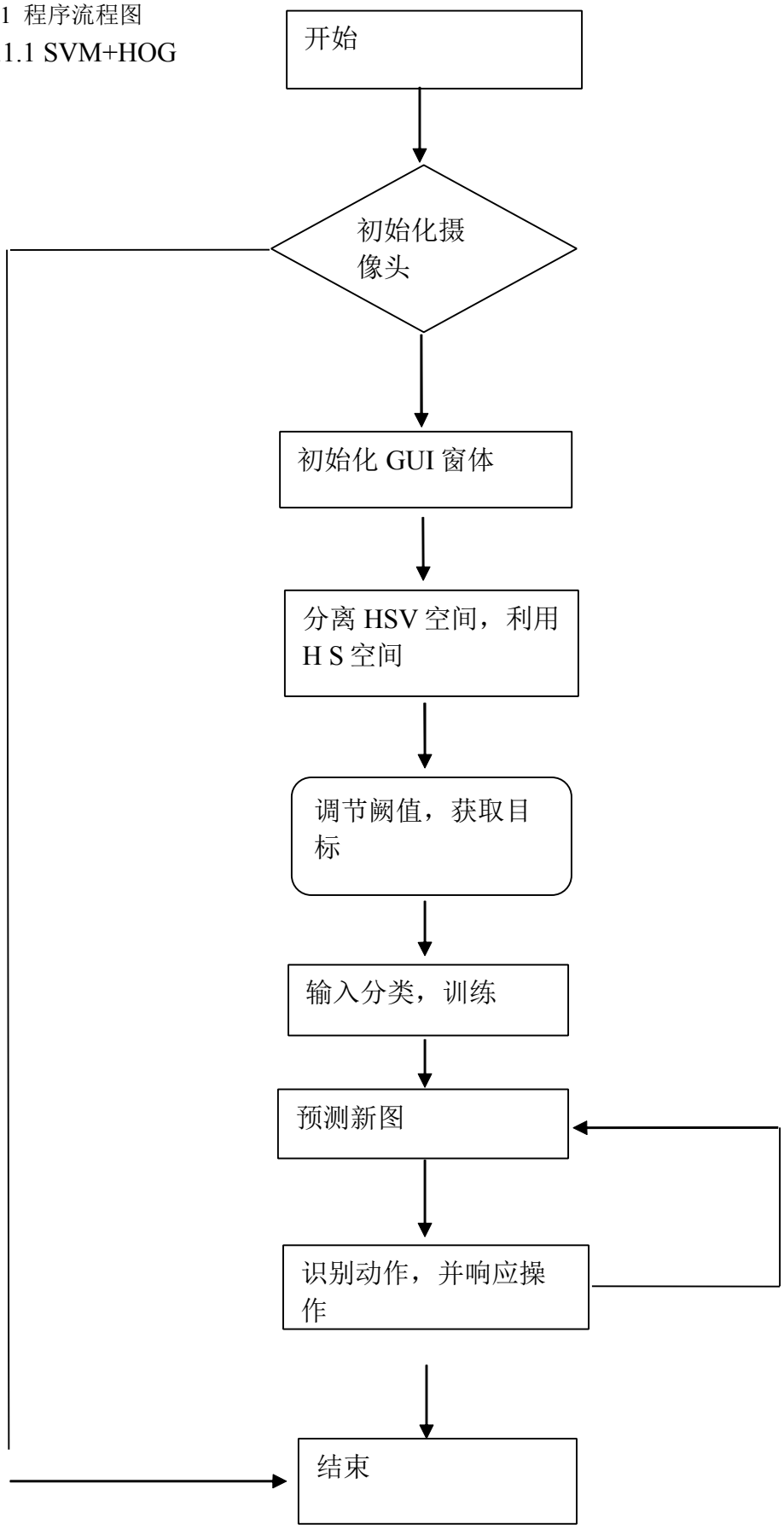
C 手势的预测识别

D 识别不动手势的动作，继而响应操作

五 程序部分

5.1 程序流程图

5.1.1 SVM+HOG

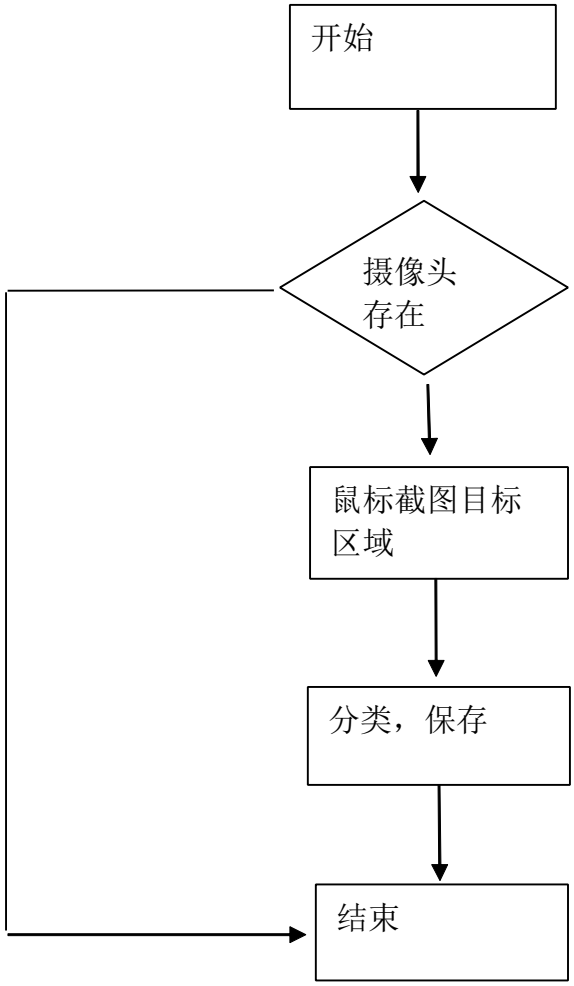


装

订

线

另，利用 ML 算法时，也可以通过鼠标截取目标区域的方法去获得训练样本，流程如下

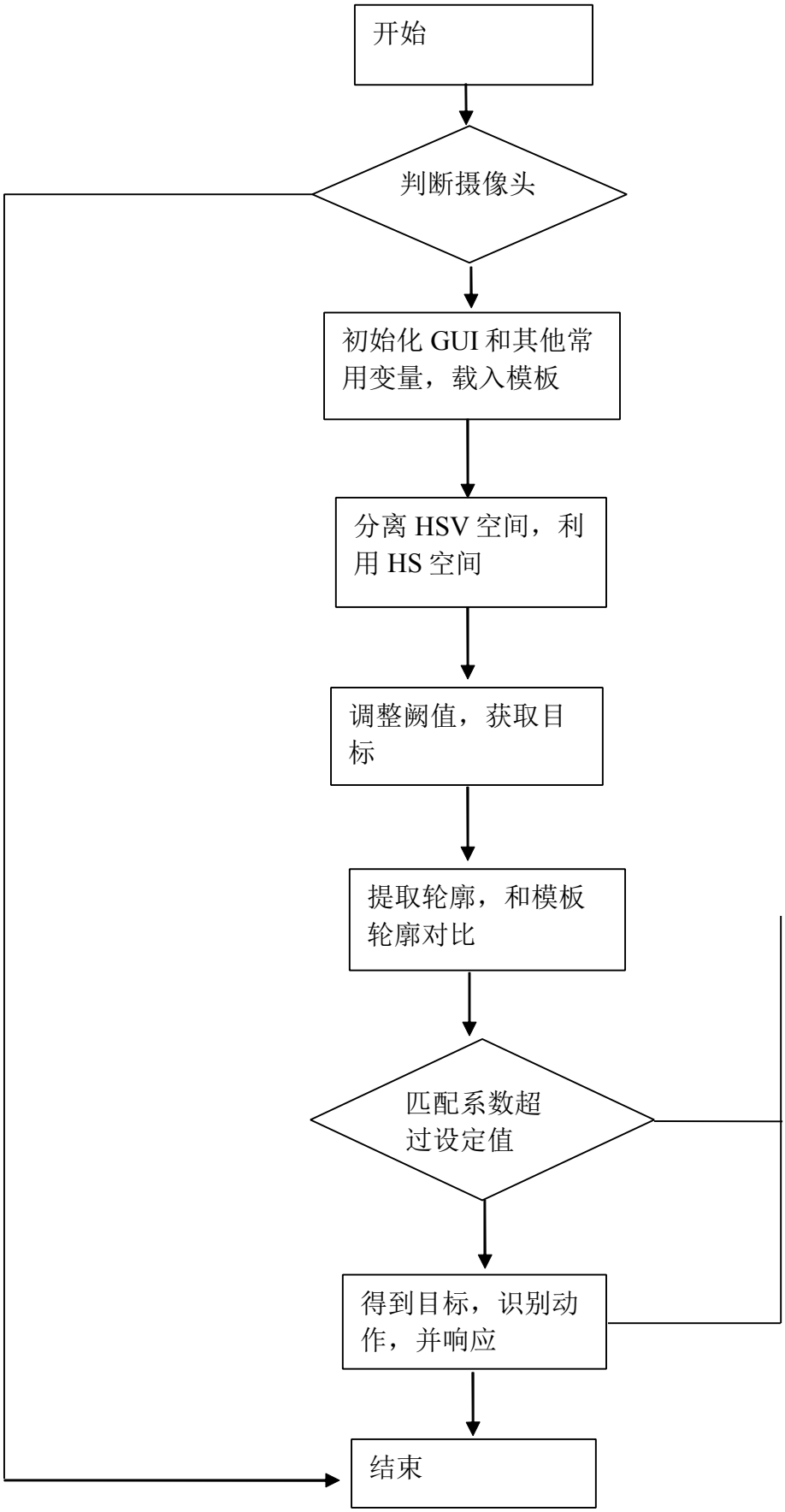


装

订

线

5.1.2 模板匹配


开始
判断摄像头
初始化 GUI 和其他常用变量，载入模板
分离 HSV 空间，利用 HS 空间
调整阙值，获取目标
提取轮廓，和模板轮廓对比
匹配系数超过设定值
得到目标，识别动作，并响应
结束

装

订

线

5.2 模块设计

本程序采用的 C++编程语言，对于这个课题来说，我没有做足够的深入研究，所以只用了一个类，所有的模块都放置在同一个类中，大致可以分为以下模块部分

A 数据采集（提取手势）

B 训练样本数据

C 模式识别

D 响应动作

对于模板匹配法，还有另外两个个模块——模板匹配模块和模板载入模块

5.3 详细设计

5.3.1 数据采集

数据的采集，根据前面的介绍，共分为两种方法：一种是获取图像后，直接用鼠标在图像上面化取图像，另外一种是直接利用肤色特征提取目标。这里推荐使用后者，这样可以减少干扰图像，提高识别效率。

鼠标提取法：

鼠标提取法的难点是如何在图像化取图像，这里我并没有使用 MFC 或者 QT 之类的，而是直接采取 opencv 自带的一个图形界面库——highGUi。对于鼠标事件，该图形库支持简单的几个操作，不过这已经足够了

voidSvmTrain::mouseCallback(int event,int x,int y,int flags,void *param)

这个便是我提取目标区间的函数。这是一个鼠标回调事件，其主要参数如下

Event 事件代号，这里我利用的是

CV_EVENT_MOUSEMOVE 鼠标移动事件

CV_EVENT_LBUTTONUP 鼠标左键放开事件

CV_EVENT_LBUTTONDOWN 鼠标左键按下事件

X 鼠标的 x 坐标

Y 鼠标的 y 坐标

Params 传递自定义对象

这里当鼠标按下时，会获取第一个坐标，当鼠标左键放开时，就是目标划取结束的时候，获取第二个坐标，利用这两个坐标，就可以获得一个矩形，矩形内部的图像就是我们想要的目标了，这时候，再利用 opencv 自带的函数进行提取即可

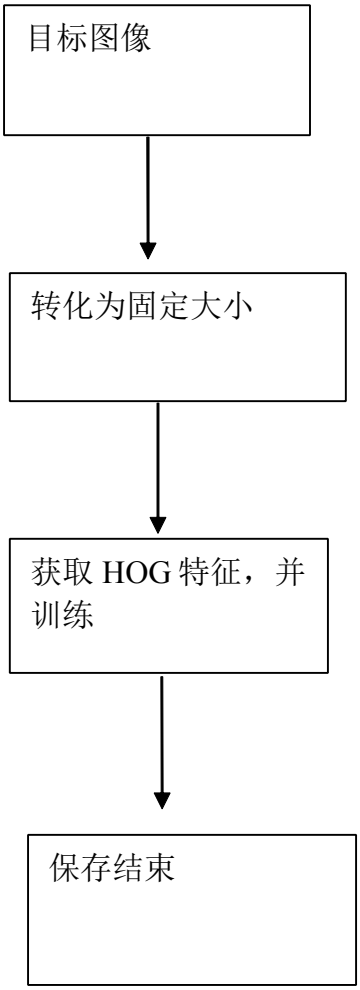
肤色模型提取法：

该方法利用肤色模型，提取目标，优点是能够减少多余的区域，同时可以更快的获得样本，推荐使用这种方法。

5.3.2 训练样本数据

这里并没有什么特别的地方，只是涉及到两个过程，一个是获取 HOG 特征，另外一个是训练。SVM 算法的训练部分函数，比较难懂，不过，我们只是利用其中的几个参数，对于其他的参数我们可以忽略掉。

这里需要注意的是，对于分类，必须有两种以上（包括两种），否则将会报错。所以在队列中需要判断分类数目。 分类训练的过程如下：



装

订

线

```
void MyHandle::Handle_Collection()
{
    ......
     cvResize( Roi, trainImg);    调整大小
     hog->compute( trainImg,res,Size(1,1),Size(0,0));        计算 HOG 特征
     h_hog.push_back(*item);   将 HOG 特征放入队列
     cat_res.push_back(pClass);
     Handle_Train();        训练
   }
}
void MyHandle::Handle_Train()
{
  ......
   //训练样本数据
  svm.train(data_mat,res_mat,NULL,NULL,param);
  //保存训练数据到文件
  svm.save("thand.xml");
}
```

这里，需要额外解释下 SVM 的参数设置，SVMParams，初始化过程为

```
criteria = cvTermCriteria( CV_TERMCRIT_EPS, 1000, FLT_EPSILON );
param = CvSVMParams( CvSVM::C_SVC, CvSVM::RBF, 10.0, 0.09, 1.0, 10.0, 0.5, 1.0,
NULL, criteria );
```

这里使用的是 **CvSVM::C_SVC** C 类支持向量分类机。 n 类分组 (n $\geq$ 2)，允许用异常值惩罚因子 C 进行不完全分类,所以我们只要填写好 cValue 值即可，其他的值可以不管。

对于 HOG 特征的初始化，请参考前面的介绍。

5.3.3 模式识别

这里主要用到 Opencv 中 SVM 算法的 predict 方法，当然，前提是也要将获取图的 HOG 特征。

```
hog->compute(trainImg, pres,Size(1,1), Size(0,0));
//预测结果
 Handle_Guesture(svm.predict(SVMtrainMat));
```

5.3.4 响应动作

从这里开始，就开始使用 linux 的底层了，前面介绍过的结构体

```
struct input_event {
    struct timeval time; //按键时间
    __u16 type; //类型，在下面有定义
    __u16 code; //要模拟成什么按键
    __s32 value;//是按下还是释放
    };
```

这里就开始派上用场了。

首先是模拟鼠标的动作

```
Handle_Simulate_key( fd,EV_KEY, BTN_LEFT, 1 );
```

其中 fd 为鼠标设备文件，其他参数见上面介绍。如下，自定义了模拟按键的方法

```
void MyHandle::Handle_Simulate_key(int file, int keytype,unsigned int keycode, int
keyvalue)
{
 event.type = keytype;
 event.code = keycode;
 event.value = keyvalue;
 gettimeofday(&event.time,0);
 write(file,&event,sizeof(event));
 memset(&event, 0, sizeof(event));
 gettimeofday(&event.time,0);
 event.type = keytype;
```

```
    event.code = keycode;
    event.value = 0;
    write(file,&event,sizeof(event));
    event.type = EV_SYN;
    event.code = SYN_REPORT;
    event.value = 0;
    write(file,&event,sizeof(event));
}
```

这个函数中向文件中写入了三次，分别代表：按键按下 按键放开 发送消息以使系统响应。特别要注意的是

```
event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(file,&event,sizeof(event));
```

必须要有，否则系统无法识别动作。

对于鼠标事件来说，有一个动作，就是双击，这里需要特别处理一下，就是两次双击之间的时间需要有限制，并且次数也要有限制。对于机器来说，cpu 足够的强大的话，其对动作的识别时很快的，一个动作在瞬间可能会被执行很多次，所以对于双击这种动作，必须加以限制，这里我使用了休眠方式

```
//延时用于处理双击
usleep(50000);
```

并且，在次数上，也要限制，本次限制为 2 次，至于拖动的动作，需要做的事情就是持续的点击，然后移动。

模拟按键事件

对于看图之类的应用来说，下一页和上一页，只要按下 空格键 和 回退键即可，所以模拟这两个按键即可模拟出换页的效果，这里直接调用模拟按键的函数即可，不在介绍。而对于放大缩小，就需要模拟两个按键的事件了一个是 CTRL，另外的就是+/-了。原理上与模拟单个按键是完全一样的，这里就简单的描述一下这个过程

1 模拟 ctrl 按键，按下去的事件，注意，这里不要模拟释放
2 模拟+/-按键按下去的事件
3 模拟释放+/-事件
4 模拟释放 ctrl 按键的事件

这样，就完成了对于放大缩小的动作。

六 程序运行结果

6.1 动作说明

我在这里给手势赋予了一下动作

0 抓取移动
1 移动鼠标
2 无
3 无
4 无
5 无
6 点击
7 放大
8 缩小

如下图

图 6.1

装 订 线

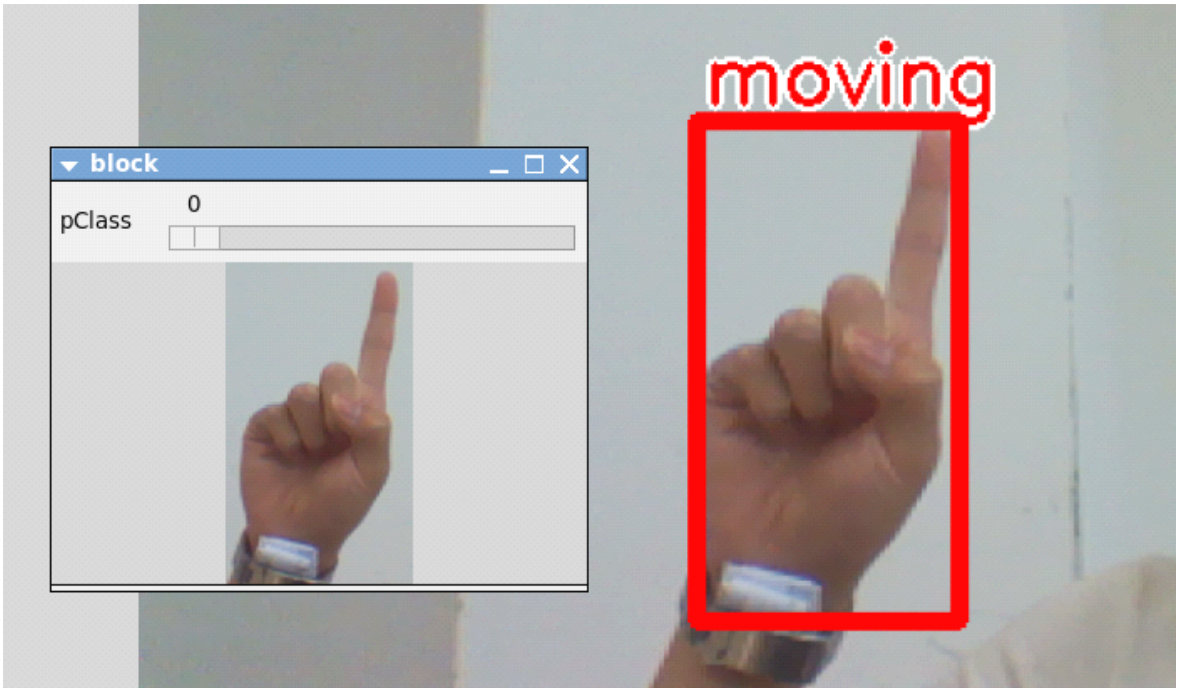


图 6.2

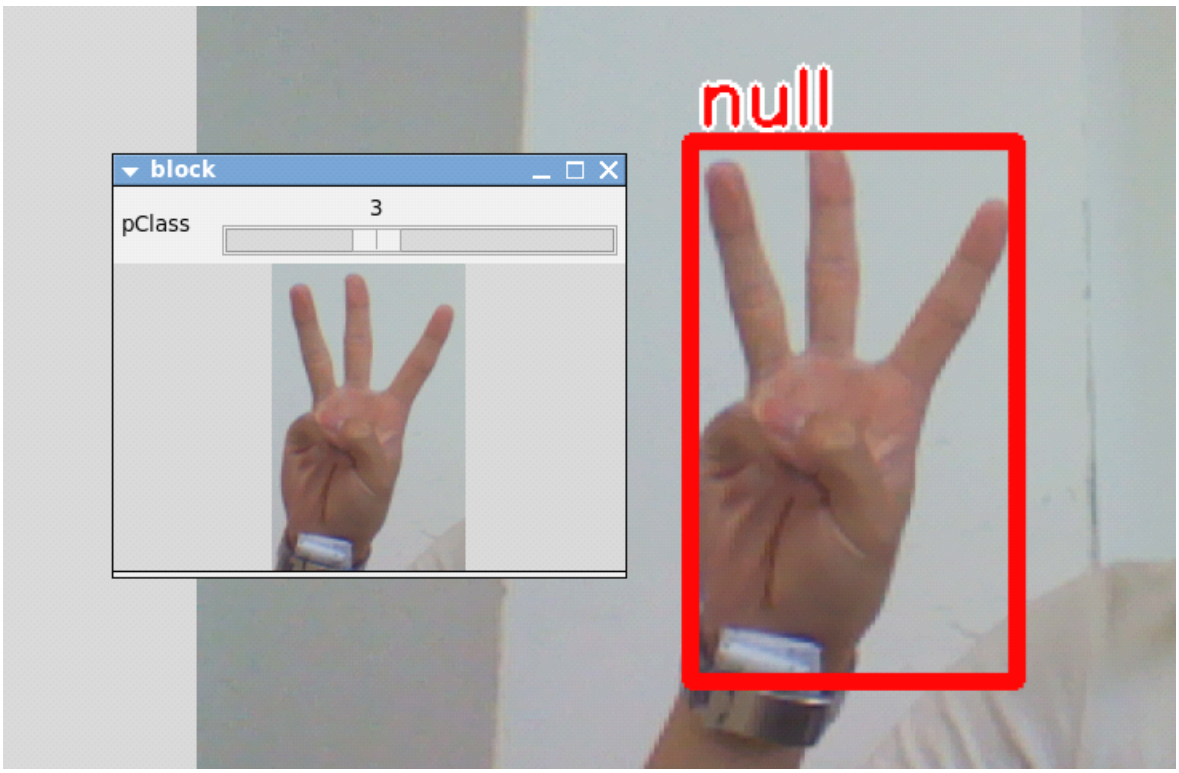


图 6.3

装

订

线

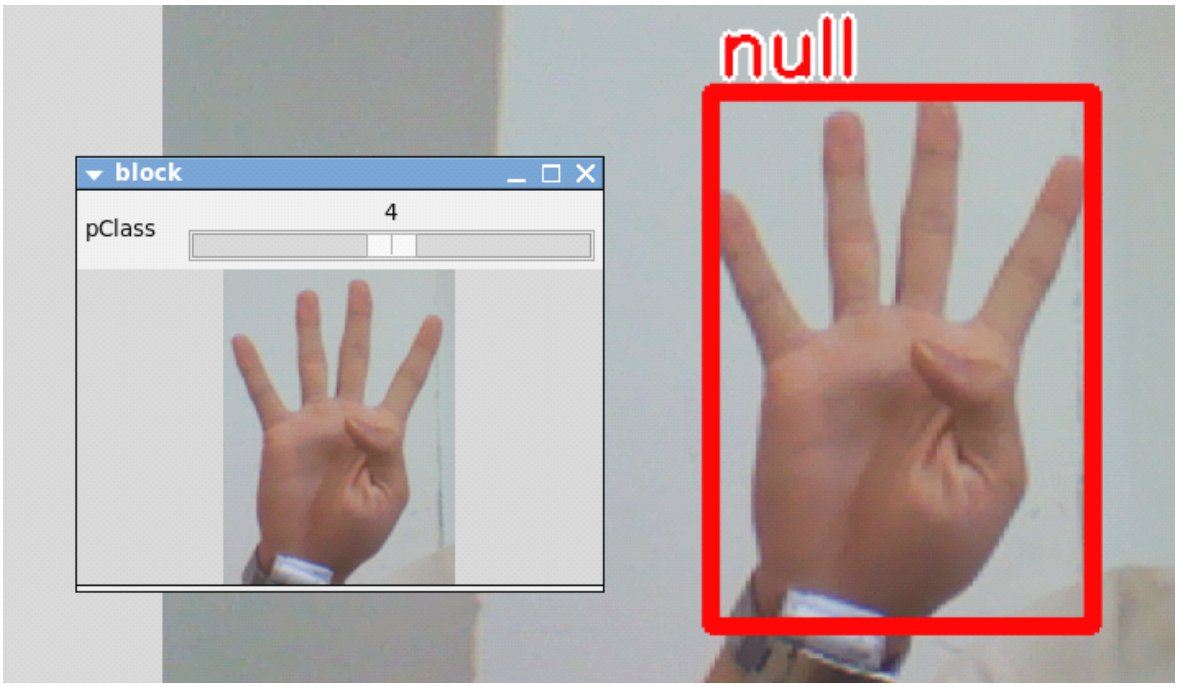


图 6.4

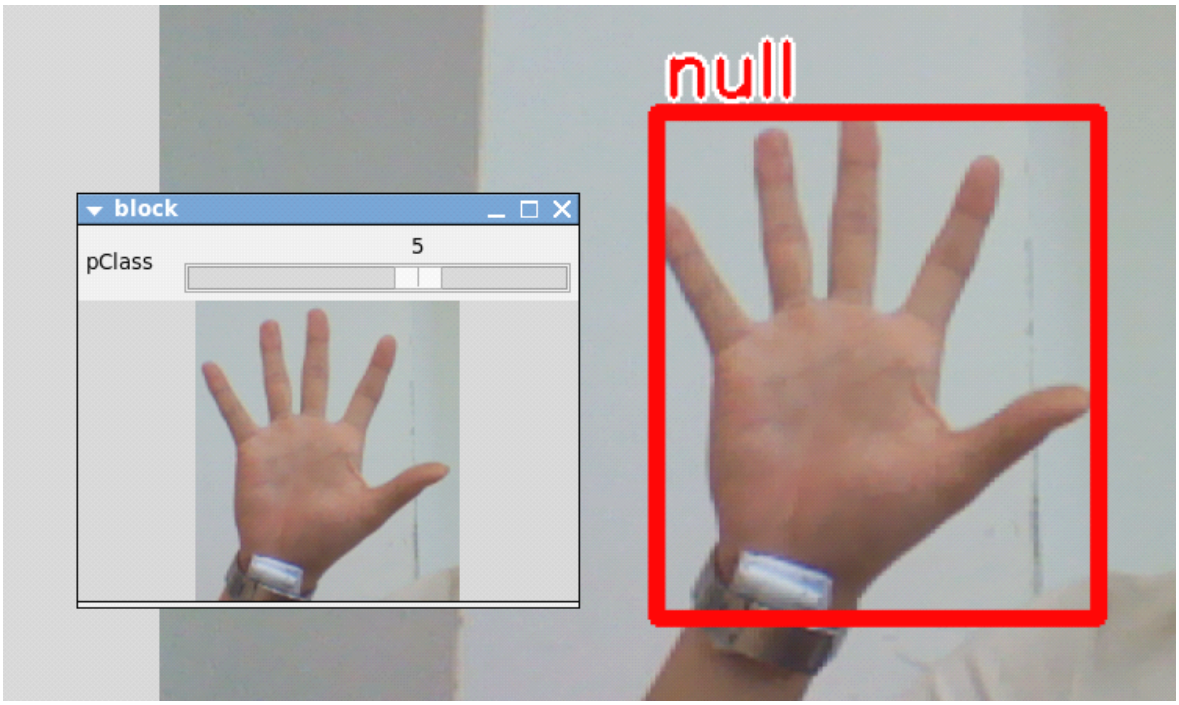


图 6.5

装 订 线

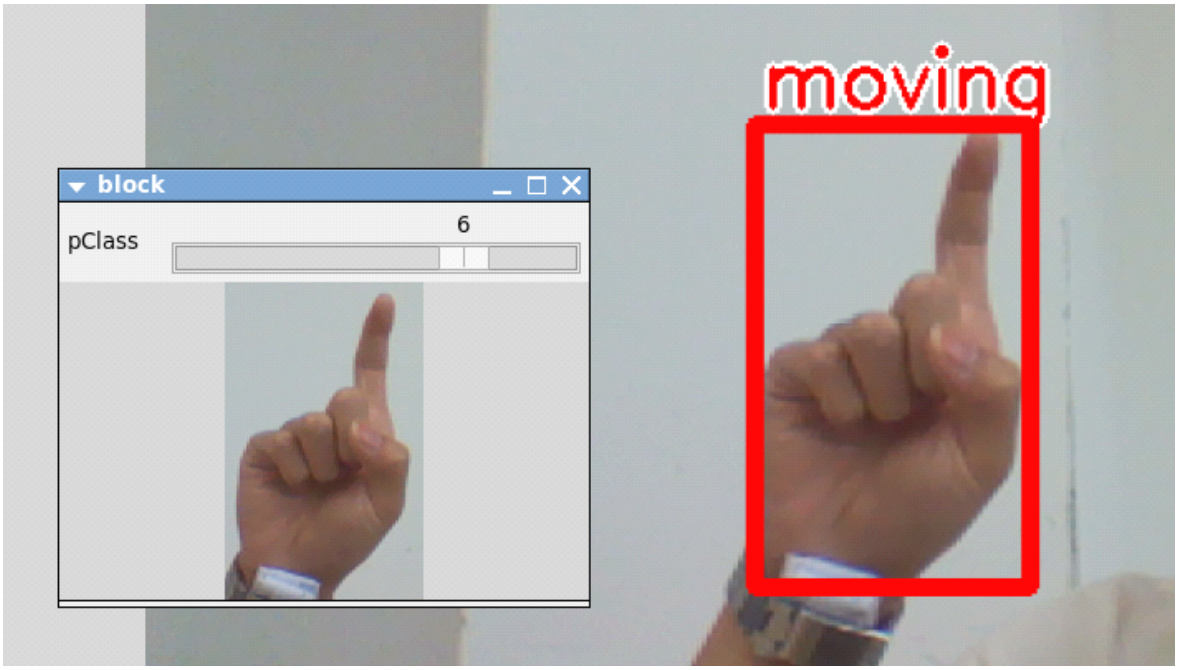


图 6.6

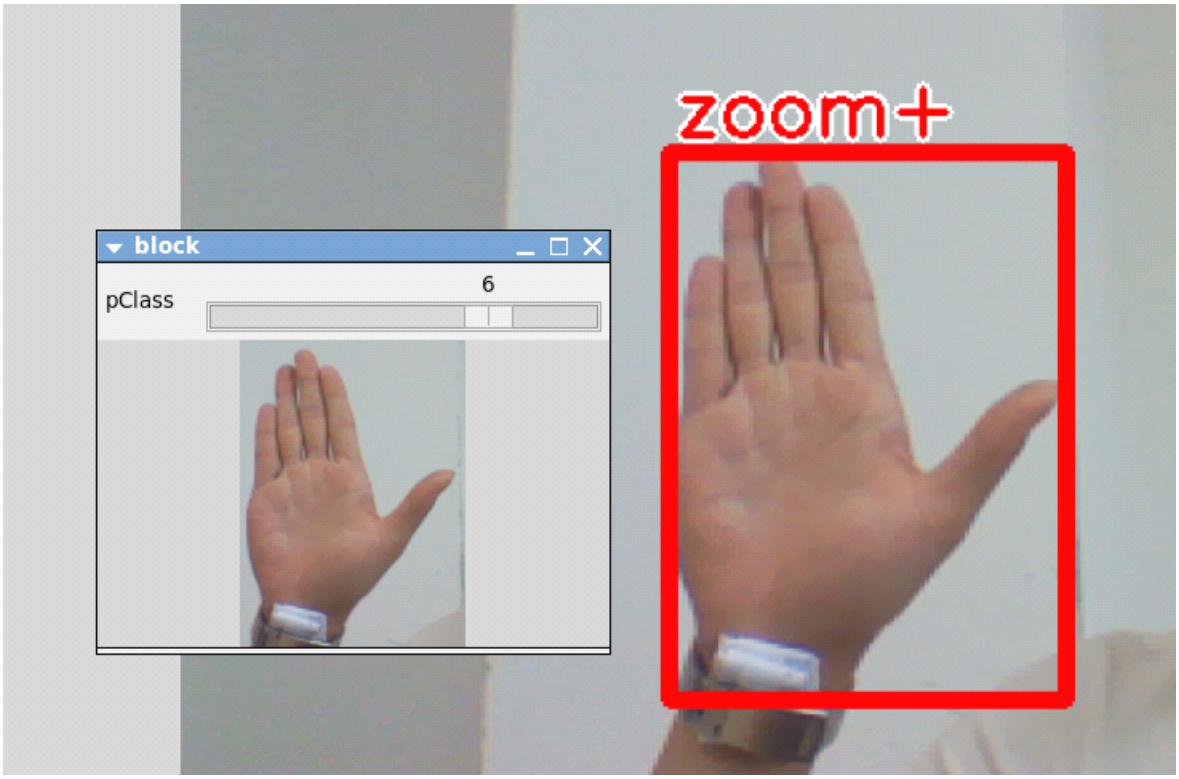


图 6.7

装 订 线

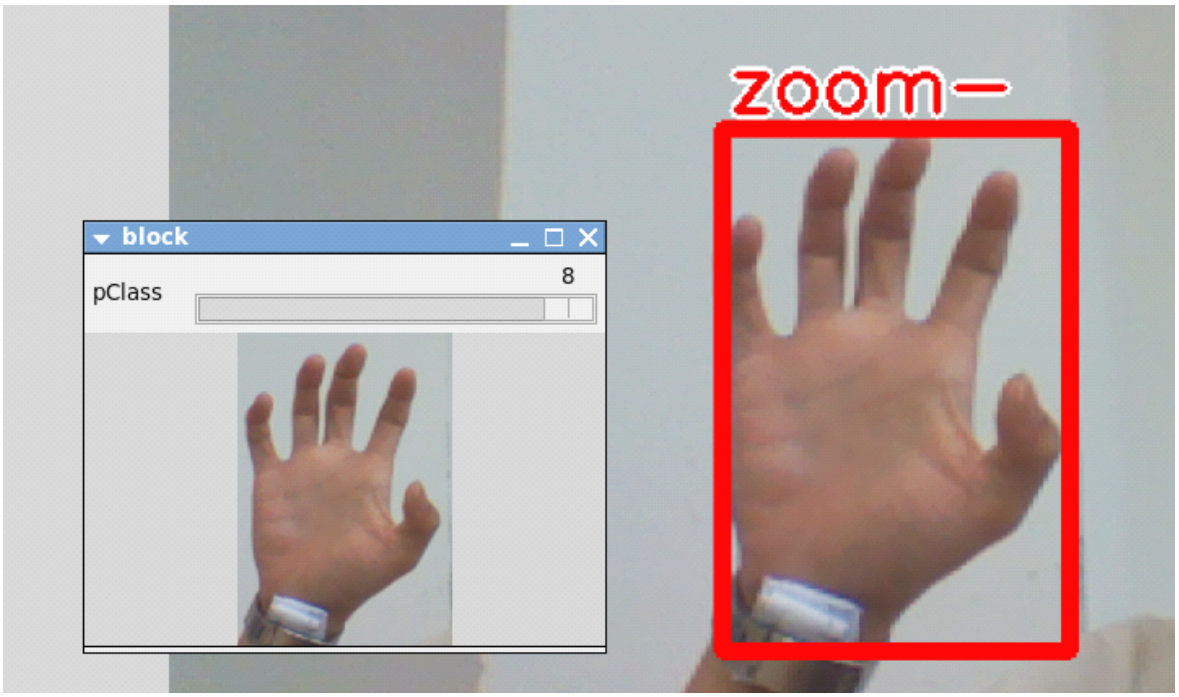


图 6.8

6.2 查看动作

各个动作，可以在控制台中查看

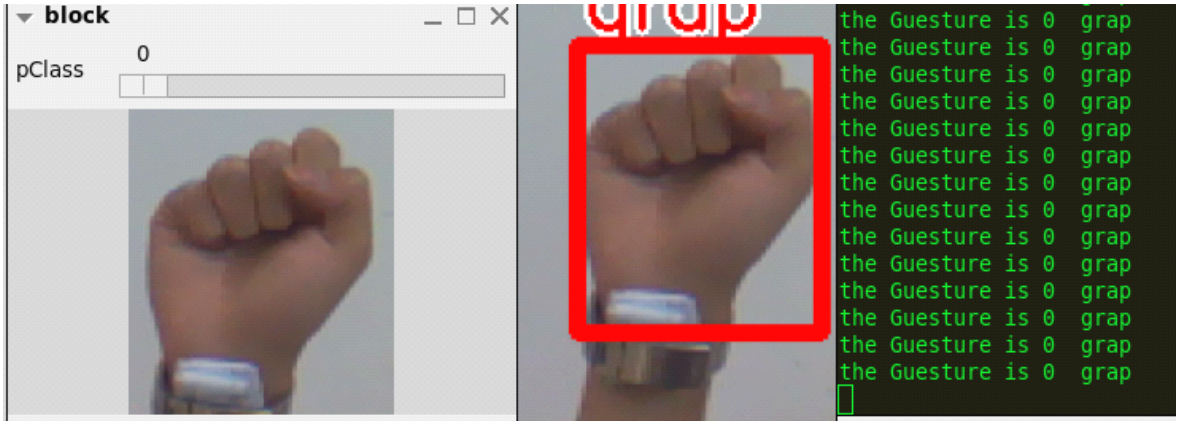


图 6.9

阈值可以用滑块调整

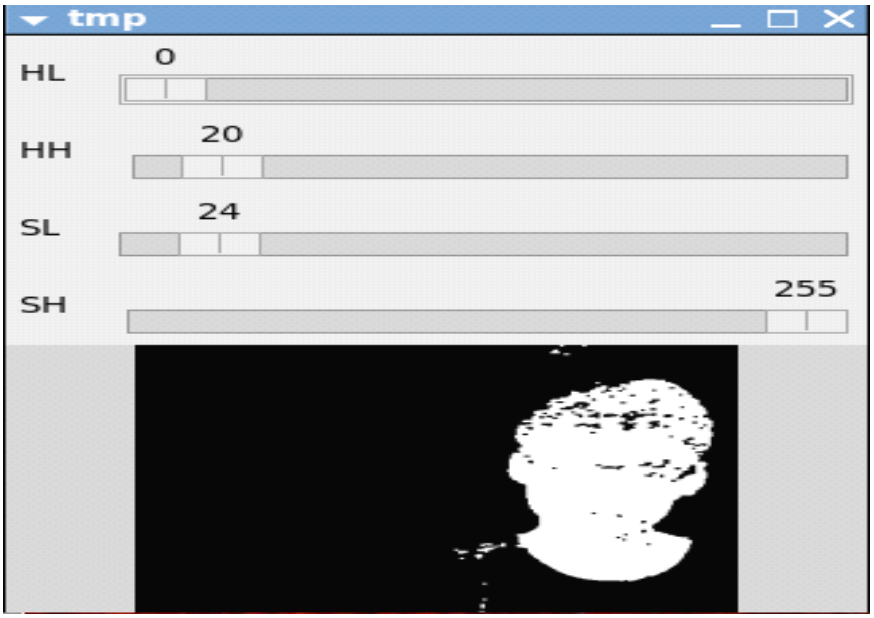


图 6.10

6.3 训练

训练时先按 S 键，分类后 按 O 键进行确认

图 6.11

6.4 测试结果

测试图如上所示，但动态的动作，就不好显示了，这里只显示控制台显示

```
the Guesture is 5  null
the Guesture is 5  null
==============================space
the Guesture is 5  null
the Guesture is 5  null
the Guesture is 5  null
the Guesture is 5  null
```

图 6.12

```
the Guesture is 5  null
==============================backspace
 the Guesture is 5  null
the Guesture is 5  null
the Guesture is 5  null
```

图 6.13

放大缩小功能

图 6.14 放大

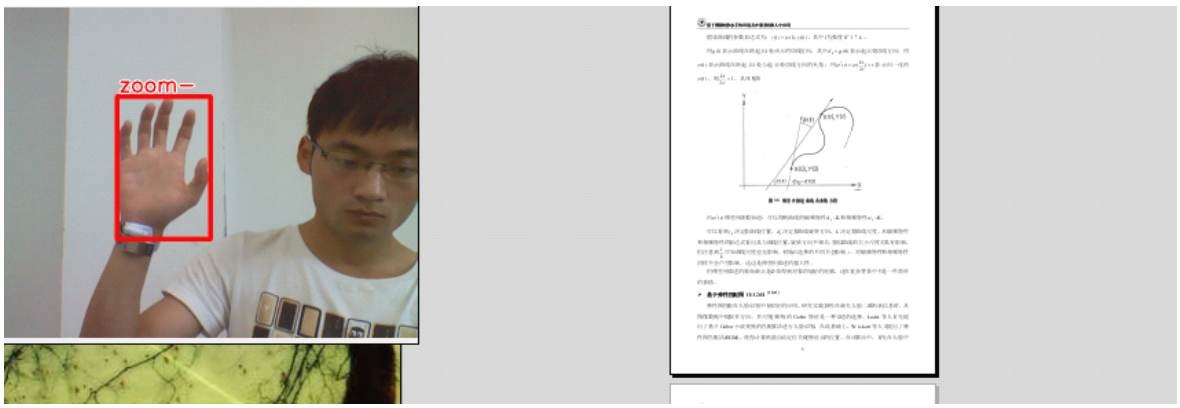


图 6.15 缩小

参考文献

《Learning opencv》 Adrian Kaehler / Dr. Gary Rost Bradski oreilly
如何在 CentOS6.0 上安装 OpenCV-2.3.2+FFmpeg0.8.2 - 春眠不觉晓，处处闻啼鸟 - 开源中国社区
皮肤检测算法三种，示例与代码 - onezeros 的专栏 - 博客频道 - CSDN.NET
OpenCV 中文论坛 • 查看主题 - 关于 HOG+SVM 的手势识别
目标检测的图像特征提取之（一）HOG 特征 - liulina603 的专栏 - 博客频道 - CSDN.NET
opencv.jp - OpenCV: サポートベクターマシン（Support Vector Machine）サンプルコード -
OpenCV 2.4+ C++ SVM 文字识别 - j_m 的个人空间 - 开源中国社区
OpenCV HOGDescriptor 参数图解 - Excalibur 的专栏 - 博客频道 - CSDN.NET
OpenCV 中的 HOG+SVM 物体分类 - Armily 的专栏 - 博客频道 - CSDN.NET
OpenCV 的 SVM 使用笔记（新手） - guyvercz 的日志 - 网易博客
OpenCV 2.4+ C++ SVM 介绍 - Justany_WhiteSnow - 博客园
百度文库
维基百科

装

订

线